Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica (8009)

ORACLE VS GOOGLE: UNA ANALISI PRELIMINARE

Relatore:
Chiar.mo Prof.
DANILO MONTESI

Presentata da:
FILIPPO PERRINA

Correlatore:
Prof.
FLAVIO BERTINI

Sessione II, Ottobre
2020-2021

Indice

Capitolo 1

Introduzione

Il copyright esiste per promuovere la creatività. Infatti, le leggi sul copyright cercano di bilanciare diversi fattori per proteggere differenti autori. Troppa protezione verso un lato danneggia lo sforzo creativo dell'altro. Raggiungere il giusto bilanciamento è fondamentale, specialmente nell'ambiente dello sviluppo software, dove il riuso di codice è pratica diffusa[1] e la protezione del codice è economicamente appetibile. Un esempio della delicatezza dell'argomento è la causa legale "Oracle vs Google", intentata nel 2010 ai danni di Google[1] per violazione di copyright e brevetti nell'utilizzo della piattaforma Java, di proprietà di Oracle[2]. La stessa ha accusato Google di aver copiato ed incluso nelle prime versioni di Android il codice di 37 pacchetti dell'API di Java, risultanti in 11,500 linee di codice sorgente[2]. In seguito all'accusa, Google ha ammesso la presenza delle 11,500 linee nel codice sorgente di Android, ma riteneva che il tipo di utilizzo rientrasse nel *fair use*, pertanto svincolato dalla violazione del copyright.
Nel 2010 è così iniziato il primo processo presso la Corte Distrettuale del North Caro-

[1]Google LLC è una delle più importanti aziende informatiche statunitensi. L'azienda offre numerosi prodotti e servizi online: il browser web Google Chrome, il sistema operativo mobile Android, il sistema operativo Chrome OS, la piattaforma multimediale YouTube, il servizio di posta elettronica Gmail e molti altri.

[2]Oracle Corporation è una società multinazionale del settore informatico. L'azienda vende software e tecnologia per database, sistemi di ingegneria *cloud* e prodotti software aziendali, in particolare i propri marchi di sistemi di gestione di database. L'azienda sviluppa e costruisce strumenti per lo sviluppo di database, hardware, software di gestione delle risorse aziendali, gestione delle relazioni con i clienti e di gestione della catena di distribuzione e software per la gestione del capitale umano.

lina, nel quale Oracle ha chiesto un risarcimento di danni fino a 8.8 miliardi di dollari statunitensi[3]. Il caso ha poi seguito diverse vicissitudini, la Corte Distrettuale ha votato due volte a favore di Google, entrambe le volte la Corte Federale ha ribaltato il giudizio. Successivamente, Google ha presentato una petizione presso la Corte Suprema statunitense, la quale ha accolto il caso e, nell'aprile del 2021, ha decretato sei voti a favore di Google contro due, ritenendo che l'uso di Google dell'API di Java rientrasse nella dottrina del *fair use*.

L'intero processo è stato seguito con grande interesse nell'ambiente informatico, in quanto svariati programmi e librerie software, soprattutto nel mondo *open source*, sono sviluppate ricreando le funzionalità di API esterne, per garantire interoperabilità tra diversi sistemi e piattaforme. Il capitolo successivo racconta i retroscena precedenti al caso, il Capitolo III tratta la fase iniziale sull'infrazione del copyright. L'analisi del *fair use* è discussa nel quarto capitolo, seguito dal Capitolo V dedicato al verdetto della Corte Suprema, il sesto e ultimo capitolo contiene le conclusioni.

Capitolo 2

Background

Un'insieme di forze ha preparato il campo per la disputa tra Oracle e Google:

1. Lo sviluppo e l'elevata adozione di Java.
2. La decisione di Google di implementare Android utilizzando Java.
3. L'acquisizione di Sun da parte di Oracle.

2.1 Lo sviluppo e la crescita di Java

Sun Microsystems[1] è stata fondata nel 1982 da un gruppo di universitari composto da due studenti di Stanford, Vinod Khosla e Andy Bechtolsheim, e due di Berkeley, Scott McNealy e Bill Joy. Nel giro di pochi anni è diventata una delle migliori aziende tecnologiche a livello mondiale, grazie alla vendita di *workstations*, processori e server[4] . Nel 1991, Sun ha iniziato un progetto chiamato "Oak" capitanato da James Gosling, con lo scopo di implementare una macchina virtuale e un linguaggio con una notazione simile a C. Quattro anni dopo, da questo progetto è nato Java, un linguaggio di programmazione *object-oriented* votato alla crescente programmazione Web. La caratteristica principale del linguaggio è la capacità di creare programmi eseguibili su diversi sistemi operativi (*platform-indipendent*), da qui il motto di Java " *Write once, run anywhere*"[5] . Oltre al

[1]Sun Microsystems è stata un'azienda statunitense produttrice di software e semiconduttori nota per avere prodotto il linguaggio di programmazione Java.

linguaggio di programmazione, la piattaforma software Java include una macchina virtuale e un insieme di librerie (Java Standard Edition, Java Micro Edition), documentate per gli sviluppatori tramite Application Programming Interface (API). Java viene ufficialmente rilasciato nel gennaio 1995 da Sun, sotto la licenza *Sun Community Source*[2], la quale si concentra fortemente nello sviluppo di Java, con l'obiettivo di renderlo il linguaggio standard per lo sviluppo Web, trascurando come ottenere profitto dal prodotto. Questo sforzo produce il primo Java Development Kit[3] e centra l'obiettivo, stabilendo Java come linguaggio *de facto* per la programmazione Web[6]. Tuttavia, le vendite di Sun collassano nei primi anni duemila a causa dell'*Internet bubble*[4], numerose compagnie che ordinavano hardware da Sun dichiarano bancarotta. L'azienda attraversa un periodo di estrema difficoltà e decide di concentrare interamente il proprio business su Java. In questa ottica, nel 2006, Sun pone Java sotto la licenza GNU GPLv2[5][8], questa mossa

[2]La Sun Community Source License è un modello di licenza per software disegnato dalla Sun Microsystems. E' simile ad una licenza *open-source*, ma richiede che il codice prodotto sotto questa licenza sia compatibile con Java e che i prodotti con scopo commerciale siano soggetti a tasse di licenza.

[3]Il Java Development Kit è l'insieme degli strumenti utilizzati per sviluppare Java, tra cui il *debugger*, il compilatore e l'interprete.

[4]L'*Internet bubble* è stata una bolla speculativa sviluppatasi tra il 1997 e il 2000 quando l'indice NASDAQ, il 10 marzo 2000, raggiunse il suo punto massimo a 5132.52 punti nel trading intraday prima di chiudere a 5048.62 punti. Dalla fine degli anni novanta la capitalizzazione dei mercati dei paesi più industrializzati vide un rapido aumento del valore delle aziende attive nell'ambito di Internet e nei relativi settori. Il periodo fu segnato dalla fondazione (e successivi fallimenti) di un numero elevato di nuove aziende con lo scopo sociale di svolgere attività nel settore legato a Internet e più in generale nel settore informatico. Una combinazione di rapido incremento dei prezzi delle azioni, la convinzione del mercato che le società in oggetto avrebbero prodotto dei profitti in futuro, speculazione individuale sulle azioni e la presenza di numerosi venture capital crearono un ambiente in cui molti investitori trascurarono i tradizionali parametri di valutazione come il rapporto prezzo-utili in favore della convinzione nel progresso tecnologico. Il collasso della bolla si ebbe tra il 2000 e il 2001. Alcune società, come Pets.com, fallirono completamente, mentre altre persero una larga parte della loro capitalizzazione rimanendo comunque solide e redditizie come Cisco Systems, le cui azioni persero circa l'86%[7].

[5]La GNU General Public License è una licenza fortemente *copyleft* per software libero. Un'opera protetta da GNU GPL deve rimanere libera, ovvero col susseguirsi delle modifiche deve continuare a garantire ai suoi utenti le quattro libertà: libertà di eseguire il programma per qualsiasi scopo, libertà di studiare come funziona il programma e di modificarlo in base alle proprie necessità, libertà di redistribuire copie del programma in modo da aiutare il prossimo e libertà di migliorare il programma e di distribuirne

espande notevolmente l'adozione della Java Micro Edition nel mercato mobile, tuttavia non riesce a far ottenere una robusta fonte di guadagno dalla piattaforma, lasciando così l'azienda in pessime condizioni.

2.2 Android e il mondo smartphone

Google è nata nel 1998 dall'idea di Larry Page e Sergey Brin, studenti universitari di Stanford. I due hanno sviluppato un motore di ricerca (l'attuale Google Search) che ha riscosso un notevole successo e dal quale, a differenza di Sun, sono riusciti ad ottenere larghi profitti tramite AdWords[6]. Grazie alle entrate ottenute dal *browser*, Google ha finanziato e sviluppato altri progetti, quali Gmail, Google Maps, Google News e Google Immagini, inoltre aveva visto nel mondo mobile e degli *smartphone* un'ulteriore opportunità di crescita[9]. Per questo motivo nel 2005 ha acquistato, per 50 milioni di dollari[10], Android, azienda che stava sviluppando una piattaforma aperta per il mondo mobile. Il team di Android usava Java per la realizzazione della piattaforma, per questo motivo il presidente di Google, Eric Schmidt, approcciò il presidente della Sun Jonathan I. Schwartz per permettere l'utilizzo delle librerie Java nell'implementazione del sistema operativo. L'accordo tra le due parti non si trovò, Google voleva che la tecnologia Java diventasse una parte *open source* della piattaforma Android, adattata per i dispositivi mobili. Sun rifiutò l'accordo poiché riteneva che Google volesse solamente fare una *fork*[7] di Java, creando una propria versione indipendente e compromettendo l'interoperabilità dell'intera piattaforma. Non trovando un accordo, Google decise di sviluppare una sua versione delle librerie Java Standard Edition, il codice così realizzato è diventato il motore dietro alla macchina virtuale Dalvik di Android. Il codice della macchina virtuale di Google includeva 37 pacchetti API, consistenti in 11500 linee[2] di codice considerate

pubblicamente i miglioramenti.

[6]Google Ads (denominato Google AdWords fino al 24 luglio 2018) è un software che permette di inserire spazi pubblicitari all'interno delle pagine di ricerca di Google. Questi annunci sono visualizzati, fino a quattro, sopra i risultati di ricerca non a pagamento, o sotto i risultati di ricerca e vengono selezionati da un algoritmo che, tra le tante variabili, tiene conto delle parole chiave ricercate dall'utente.

[7]Un fork, nell'ambito dell'ingegneria del software e dell'informatica, indica lo sviluppo di un nuovo progetto software che parte dal codice sorgente di un altro già esistente, a opera di un programmatore.

centrali in Java, che erano state prese da Apache Harmony, un'implementazione *open-source* di Java sviluppata dalla Apache Software Foundation. Google afferma di aver usato questo codice per garantire interoperabilità ai programmatori con la Java Standard Edition. Grazie all'inclusione di queste parti di codice, Google rilasciò una *beta* di Android il 5 novembre 2007[11], il presidente della Sun si congratulò con Google e non mosse accusa ai suoi danni. Dopo un avvio graduale, Android prese il sopravvento nel mercato mobile, arrivando nel 2016 ad avere l'84% dei dispositivi mobili venduti con il proprio sistema operativo installato[12].

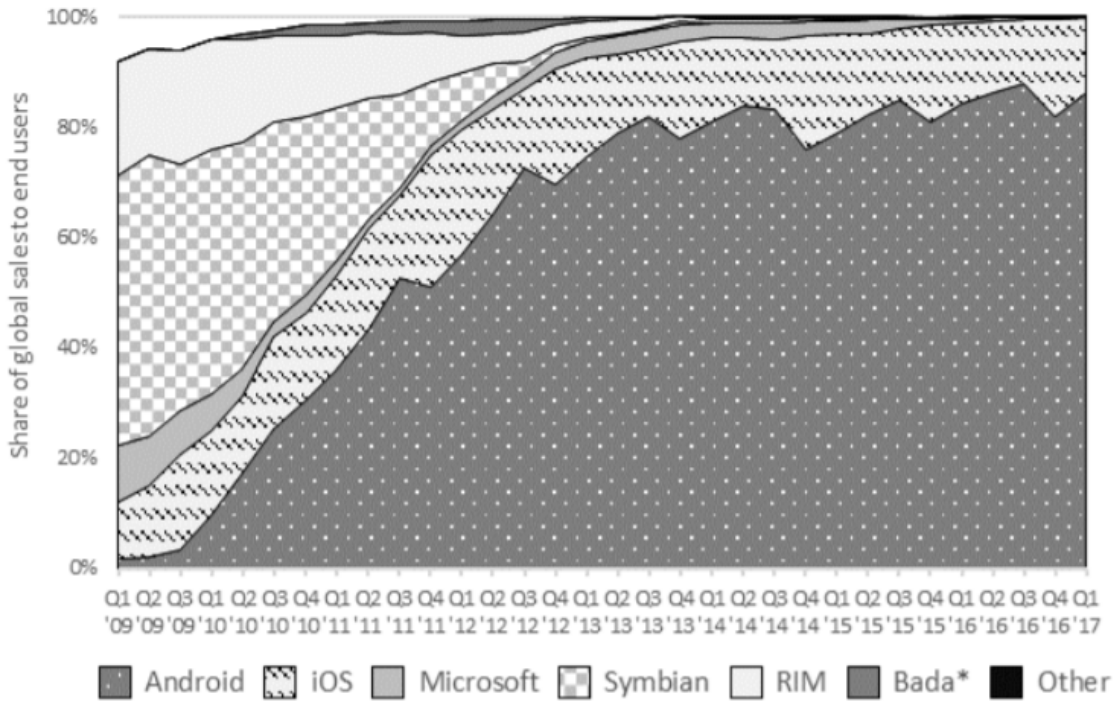


Figura 2.1: Distribuzione del mercato dei sistemi operativi mobili nel 2016[12].

2.3 Oracle acquisisce Sun

A causa del calo delle vendite e dell'incapacità nel monetizzare Java, Sun non riusciva più a proseguire come compagnia indipendente[13]. Nell'aprile del 2009, Sun cede all'of-

ferta di 7.4 miliardi di dollari di Oracle[14], l'acquisizione è stata completata un anno dopo. Oracle ha subito provveduto a denunciare Google per violazione di copyright e brevetto, così ha avuto inizio il caso tra i due colossi. Va precisato che dal rilascio di Android KitKat (4.4) nel 2013, Google ha sostituito la macchina virtuale Dalvik con Android Runtime, che è stata sviluppata senza nessuna linea di codice di Java.

2.4 Il sistema giuridico negli Stati Uniti d'America

L'intero caso segue le norme e la giurisdizione del diritto degli Stati Uniti d'America. Gli Stati Uniti sono una Repubblica federale di tipo presidenziale. Nell'esperienza statunitense, Federazione e Stati mantengono diverse sfere di competenze, i cui confini, tuttavia, non sono mai stati fissati in maniera rigida ed assoluta. La miglior sintesi della divisione dei poteri fra Stati e Federazione è probabilmente contenuta nel decimo emendamento alla Costituzione, in base al quale i poteri che non sono espressamente attribuiti al Governo federale e che non sono dalla stessa Costituzione sottratti alla competenza dei singoli Stati, sono riservati a questi ultimi.
I singoli Stati, tuttavia, con il consenso espresso del Governo e nei limiti posti da quest'ultimo, godono di una autonoma potestà legislativa. In linea di principio, può dirsi, per concludere, che nel sistema federale statunitense gran parte del diritto privato è di competenza statale. La Costituzione degli Stati Uniti adottata nel 1789 ed emendata solo raramente da allora, è la legge suprema del Paese. Essa prevede tre rami separati del governo:

- Il potere legislativo (Articolo 1)
- Il potere esecutivo (Articolo 2)
- Il potere giudiziario (Articolo 3)

I tre rami del governo federale operano nell'ambito di un sistema costituzionale noto con il nome di "checks and balances", sulla base del quale ciascuno di essi è formalmente separato dagli altri due e ha l'autorità, sancita dalla Costituzione, di controllare le azioni degli altri. Con una certa approssimazione, si può affermare che l'organizzazione dei singoli Stati riproduce quella federale appena descritta:

- il potere legislativo statale è esercitato dal Congresso, un'assemblea bicamerale composta da Senato e Camera dei Rappresentanti;
- il potere esecutivo è accentrato nelle mani del Presidente degli Stati Uniti d'America;
- il potere giudiziario è esercitato dalle corti supreme, o corti di ultima istanza.

2.4.1 Le Fonti del Diritto

Alla luce della particolare complessità del sistema giuridico statunitense, può essere utile offrire una sintesi semplificata della gerarchia delle fonti del diritto negli Stati Uniti:

1. la Costituzione degli Stati Uniti rappresenta il diritto supremo del Paese, al quale tutte le altri fonti sono subordinate;
2. i Trattati, sottoscritti dal Presidente degli Stati Uniti e approvati dai due terzi del Senato, hanno la stessa autorità delle leggi federali e sono subordinati alla sola Costituzione;
3. le leggi federali, emanate dal Congresso, sono subordinate solo alla Costituzione;
4. le costituzioni statali, rappresentano la massima autorità normativa all'interno di ciascuno Stato nel quale sono state emanate, ma sono tuttavia, subordinate alle leggi federali;
5. le leggi statali, emanate dai rispettivi organi legislativi locali, sono subordinate sia alla costituzione statale che alle leggi federali; rivestono, tuttavia, notevole importanza, all'interno degli Stati, in tutti quei settori del diritto che la Costituzione degli Stati Uniti non ha espressamente riservato al Congresso.

2.4.2 La Struttura dei Tribunali Federali

La competenza dei tribunali federali si estende ad un'ampia varietà di casi. Gli stessi giudici federali si occupano di procedimenti civili e penali, di controversie che rientrano nel settore del diritto pubblico e privato, di cause riguardanti persone fisiche, giuridiche

ed organismi governativi, di ricorsi in appello a seguito di provvedimenti di enti amministrativi e di questioni regolate dalla legge o dal diritto consuetudinario (*equity*).

Non esistono corti costituzionali separate, dal momento che ogni tribunale ha competenza in merito alla costituzionalità delle leggi federali e di altri atti dello Stato cui si faccia riferimento nel corso dei procedimenti istituiti davanti ad esso. In linea di principio, i tribunali federali sono competenti a giudicare i casi che riguardano il governo degli Stati Uniti o i suoi rappresentanti, la Costituzione degli Stati Uniti o le leggi federali, oppure controversie tra Stati o tra gli Stati Uniti e i governi stranieri.

Tribunali di Prima Istanza

I tribunali distrettuali sono i principali tribunali di prima istanza nel sistema giudiziario federale. Essi hanno competenza per quasi tutti i tipi di procedimenti federali. Ogni distretto giudiziario federale comprende un tribunale fallimentare che opera all'interno di una sua sezione. Esistono, inoltre, due tipi speciali di tribunali di prima istanza che hanno competenza su tutto il territorio nazionale per alcune materie specifiche. Il Tribunale del Commercio Internazionale si occupa appunto dei casi che riguardano il commercio internazionale e la normativa doganale. Il Tribunale degli Stati Uniti per i Ricorsi Federali è competente, invece, per le controversie riguardanti gli appalti del governo federale, l'appropriazione di beni privati da parte dello stesso governo e per una serie di altri ricorsi di natura economica nei confronti del governo. I procedimenti davanti ai tribunali di prima istanza sono condotti da un solo giudice, singolarmente o con l'appoggio di una giuria, composta da cittadini e incaricata di accertare i fatti del caso. La Costituzione prevede il diritto alla giuria in molte tipologie di casi, tra i quali:

- tutti i processi penali per reati gravi;
- i procedimenti per i quali il Congresso ha espressamente previsto il ricorso alla giuria.

Le Corti di Appello

I distretti giudiziari sono organizzati in circuiti regionali, in ciascuno dei quali è presente una Corte d'Appello. Ognuna di esse giudica i ricorsi provenienti dai tribunali distrettuali appartenenti al proprio circuito e da alcuni enti amministrativi federali. Inoltre, la Corte d'Appello federale ha competenza su tutto il territorio nazionale in merito ad alcuni casi specifici, tra i quali quelli relativi alle leggi sui brevetti e quelli sui quali si sono pronunciati i Tribunali per il Commercio Internazionale e il Tribunale per i Ricorsi Federali. Il diritto all'appello si applica a tutti i procedimenti sui quali si è pronunciato un tribunale distrettuale con una decisione definitiva. Le Corti d'Appello sono di regola composte da tre giudici.

La Corte Suprema

La Corte Suprema è il tribunale di ultima istanza del sistema giudiziario federale. È composta da un Presidente e da otto giudici associati. La Corte siede sempre "en banc", cioè con la presenza di tutti i giudici. I nove giudici conducono le udienze ed emettono le sentenze congiuntamente. La competenza della Corte Suprema è quasi interamente discrezionale e l'esame di ogni caso richiede l'accordo di almeno quattro giudici. In un ridotto numero di casi particolari, ad esempio controversie tra Stati per questioni di confini, la Corte Suprema funge da tribunale di prima istanza oppure svolge un riesame in appello obbligatorio. In generale, la Corte accetta di giudicare solo i casi in cui le corti d'appello hanno dato pareri discordanti o quelli che presentano importanti questioni costituzionali o relative alla interpretazione delle leggi federali e per le quali si rendano necessari chiarimenti.

2.4.3 La Giurisdizione Statale

Pur trovandosi in tutti gli Stati, i tribunali federali non sono gli unici fori a disposizione di chi intende promuovere un'azione legale. Al contrario, la stragrande maggioranza delle controversie portate davanti ai giudici statunitensi si svolgono nei tribunali statali, sistemi a parte istituiti in ciascuno dei 50 Stati. La maggior parte dei sistemi giudiziari statali sono strutturati in tribunali di prima istanza con competenza generale, Corti

d'Appello intermedie e una Corte Suprema dello Stato. Essi possono, altresì, prevedere tribunali di ordine inferiore per questioni specifiche, tribunali di contea, tribunali comunali, tribunali per questioni di piccola entità o giudici di pace chiamati a risolvere controversie minori. L'organizzazione delle corti statali varia, tuttavia, da Stato a Stato, così come la stessa denominazione delle corti.
Nello Stato di New York, ad esempio, la corte di ultima istanza non si chiama Corte Suprema, come avviene nella stragrande maggioranza degli altri Stati, ma è denominata Court of Appeals. La competenza attribuita ai tribunali statali è decisamente superiore a quella prevista in favore dei tribunali federali. Ad esempio, è davanti ai tribunali statali che si svolgono le cause di divorzio e quelle per l'affidamento dei minori, le questioni relative a successioni ed eredità, le cause immobiliari e quelle che coinvolgono minori, oltre alla maggior parte delle cause penali, le controversie contrattuali, le violazioni del codice stradale e le cause per lesioni personali.

È importante sottolineare che i tribunali federali e statali sono tenuti a riconoscere "piena fiducia e credito" alle rispettive sentenze. La "Supremacy Clause" della Costituzione stabilisce, tuttavia,che in caso di conflitto, le leggi federali prevalgano su quelle statali.

2.4.4 Il Sistema Processuale

La caratteristica principale del sistema giudiziario statunitense è, senza dubbio, costituita dal principio dell'iniziativa delle parti, noto come "adversary principle".
Altra caratteristica rilevante è la presenza della giuria, che ha comportato una separazione assai netta tra questione di fatto e questione di diritto: la prima di competenza della giuria, mentre la seconda di competenza esclusiva del giudice. Di qui lo svilupparsi del sistema delle impugnazioni, che si svolgono in sede di appellate jurisdiction e riguardano le sole questioni di diritto.
Le questioni di fatto sono, invece, lasciate alla decisione delle corti di primo grado. La Costituzione degli Stati Uniti garantisce la presenza della giuria sia nel processo civile che in quello penale.

Un altro aspetto interessante del sistema processuale americano è il ruling power attribuito alle corti; si tratta, in sostanza, del potere riconosciuto a ciascuna corte di autodisciplinare la propria procedura. Questo consente un più rapido ed efficiente emendamento delle regole procedurali, qualora se ne presentasse la necessità. Sempre in tema di norme procedurali, la Corte Suprema ha promulgato nel 1983 le Federal Rules of Civil Procedure che rappresentano il modello processuale seguito dalla pressoché totalità delle corti statali. Esse possono essere considerate, a tutti gli effetti, il vero codice di procedura civile americano.
Accanto alle codificazioni redatte direttamente dalle corti, un importante ruolo di fonte del diritto processuale spetta, infine, a diversi statutes che direttamente o indirettamente coinvolgono aspetti di procedura.

La Struttura del Processo:

Lo svolgimento del processo civile statunitense prevede, in linea generale, le seguenti fasi:

- Notifica dell'atto di citazione (*Service of process*);
- Scambio di note difensive (*Pleadings*);
- Fase predibattimentale (*Pre-trial*);
- Dibattimento (*Trial*);
- Sentenza (*Judgment*).

In linea di principio, il fulcro del sistema processuale risiede nella fase dibattimentale, dominata dall'iniziativa probatoria delle parti (*adversary system*) e concentrato in un'unica udienza davanti ad un giudice che ha la sola funzione di garantire la correttezza del contraddittorio e il rispetto delle norme di procedura, ma privo di poteri istruttori. Da un punto di vista prettamente pratico, tuttavia, solo una minima parte delle cause portate davanti ad un tribunale si protraggono sino al dibattimento: la maggior parte viene risolta nella fase predibattimentale, in via transattiva. È proprio in questa fase che

gli avvocati di parte si preparano e preparano i testimoni al dibattimento, entrando in possesso, attraverso una serie di attività istruttorie, delle informazioni e dei documenti probatori intorno ai quali si svolgerà il trial.
Oltre a questa funzione preparatoria, la fase *pre-trial* svolge l'ulteriore funzione di offrire importanti sbocchi per la soluzione extradibattimentale della controversia. La fase predibattimentale si svolge sotto la direzione del giudice, che è di regola diverso da quello che interverrà in fase di dibattimento. Il suo ruolo è principalmente quello di garantire che le parti rispettino le norme. A questo scopo, i giudici sono dotati di poteri sanzionatori particolarmente efficaci che possono prevedere anche la comminazione di multe e addirittura l'incarcerazione per chi tiene un comportamento oltraggioso alla corte. Aspetto decisamente interessante di questa fase del processo statunitense è la possibilità per gli avvocati di intrattenere intensi rapporti con i testimoni in assoluta assenza del giudice. Gli avvocati sono i veri protagonisti del Processo. È questa la ragione principale della estrema onerosità delle cause negli Stati Uniti: il coinvolgimento di più avvocati nella delicata fase pre-trial e la conseguente necessità per costoro di svolgere una serie di attività finalizzate alla raccolta del maggior numero di testimonianze possibili, fa sì che i costi aumentino vertiginosamente. Il dibattimento è informato al principio della oralità: ciascuna parte, tramite il proprio avvocato, introduce alla corte la propria tesi processuale formulando un discorso introduttivo. Segue, quindi,la fase dell'assunzione delle prove, le cui regole sono stabilite in maniera molto precisa e dettagliata dalla law of evidence, che disciplina sia il processo penale che quello civile. Il potere di ammettere o escludere una prova è di esclusiva competenza del giudice.
La fase più delicata ed importante dell'intero dibattimento è l'escussione dei testimoni. Ciascuna parte fornisce al giudice un elenco dei testimoni che intende chiamare a deporre. Al termine dell'interrogatorio, la controparte ha diritto a controinterrogare il testimone. Lo scopo è, evidentemente, quello di dimostrarne la inattendibilità. Esaurito il controinterrogatorio, la parte che aveva introdotto il testimone ha la facoltà di procedere ad un nuovo ed ultimo interrogatorio volto a ristabilire la credibilità del proprio teste. Terminata la fase dell'assunzione probatoria, le parti pronunciano le rispettive perorazioni finali in attesa di ricevere il verdetto della giuria.

Capitolo 3

Prima fase: le API sono soggette a copyright?

3.1 Primo processo Corte Distrettuale

Oracle ha querelato Google per violazione del copyright e di brevetto presso la District Court for the Northern District of California, in quanto reputava che Google sapesse di aver sviluppato Android e aver copiato i 37 pacchetti dell'API senza una licenza Java. Google avrebbe inoltre infranto sette brevetti[1][2] legati alla tecnologia Java, perciò Oracle chiedeva un risarcimento monetario e che Google interrompesse l'utilizzo del materiale sotto accusa.

3.1.1 Argomentazioni di Oracle

Oracle ha posto particolare enfasi sui seguenti temi:

1. la creatività come elemento chiave nello sviluppo di un API;
2. la violazione del copyright di Google causata dalla copia di 11500 linee di codice Java;

[1]I sette brevetti di cui si è discusso sono: 5,966,702, 6,061,520, 6,125,447, 6,192,476, RE38,104, 6,910,205 e 7,426,720.[15]

3. l'importanza della compatibilità dei programmi scritti in Java, la filosofia "Write Once, Run Everywhere";

4. i dipendenti di Google erano a conoscenza della necessità di una licenza Java per sviluppare Android, nonostante ciò hanno realizzato il sistema operativo senza alcuna autorizzazione.

Per dar forza al proprio attacco Oracle ha chiamato a testimoniare diversi esperti del settore giuridico e informatico: tra questi citiamo, John Mitchell[2], professore di informatica all'università di Stanford, e Alan Purdy, analista di software presso l'azienda Johnson-Laird Inc. Di seguito sono riportati i punti salienti degli *expert report*, realizzati dai due esperti.

Creatività API

Una API (*Application Programming Interface*) è un'interfaccia tra due programmi software, che permette lo scambio di dati e servizi tra due dispositivi (che chiameremo computer per semplicità). L'API collega due computer attraverso due tipi di programmi software, un'applicazione eseguita su un computer *provider* e una o più applicazione eseguite su computer *client*[16]. E' un'insieme di chiamate e funzioni che formano un'interfaccia documentata che indica ai programmatori quali informazioni fornire alle funzioni di libreria e quali risultati aspettarsi in risposta, eliminando ogni necessità per il programmatore di conoscere la libreria nel dettaglio.

Le API di Java hanno una struttura gerarchica e sono composte da tre elementi principali: pacchetti, classi e campi/metodi. L'elemento più alto della scala gerarchica sono i pacchetti, che fungono da contenitori per classi che implementano funzionalità simili. Ad esempio, il pacchetto java.io è il pacchetto contenente le classi per la gestione di input ed output. All'interno delle classi è presente un'ulteriore gerarchia, tale che una sottoclasse (figlio) può ereditare caratteristiche dalla sovraclasse (padre). La maggioranza delle classi contiene campi per memorizzare dati, ad esempio "out" è un campo, della classe

[2]La pagina web del professor Mitchell è disponibile all'indirizzo: https://theory.stanford.edu/~jcm/

java.io.FilterOutputStream, che contiene il dato di tipo OutputStream che verrà filtrato. In aggiunta, ogni classe contiene metodi, sottoprogrammi riusabili che svolgono operazioni specifiche. Ogni metodo ha diverse caratteristiche raggruppate nella dichiarazione del metodo. La prima è il nome del metodo, che descrive intuitivamente lo scopo dello stesso, così da facilitare l'utilizzo al programmatore. Ad esempio, nella classe java.lang.Math è presente un metodo chiamato "abs" che calcola il valore assoluto di un numero. Il secondo importante attributo di quasi tutti i metodi è l'insieme di argomenti che un metodo si aspetta in input quando invocato. Come quando un utente specifica il numero di copie che vuole stampate, il metodo quando invocato si aspetta di ricevere i parametri su cui eseguire. Nella funzione "abs" menzionata precedentemente abbiamo un solo argomento, il numero di cui vogliamo il valore assoluto. I parametri vanno definiti quando l'API viene programmata, questo per evitare incongruenze nel tipo di dato passato al metodo, ad esempio un carattere quando ci aspettiamo un numero. Per questo ragione, la dichiarazione del metodo specifica il tipo dei parametri accettato, ad esempio "integer" (numero intero), "string" (stringa di caratteri) e altri ancora. Quando un metodo viene utilizzato da un programma, tipicamente ritorna un risultato a quest'utlimo. Il risultato è l'ultima componente del metodo ed è chiamato il valore di ritorno. Lo scopo del valore di ritorno è di restituire informazioni che possono essere utilizzate dal programma per altri scopi. Anche il valore di ritorno ha un tipo predefinito, che viene specificato nella dichiarazione del metodo. La documentazione per un metodo combina tutte le caratteristiche e descrive brevemente lo scopo della funzione. La documentazione della funzione "abs" è:

int abs(int a)	Ritorna il valore assoluto di un valore int

- La prima parte("int") mostra il tipo del valore di ritorno, in questo caso un numero intero;
- La seconda parte "abs" è il nome del metodo;
- La parte in parentesi ("int a") specifica il tipo del parametro che va passato al metodo. Il nome "a" è solo di convenienza, qualsiasi nome per il parametro andrebbe bene.

- Le tre parti insieme formano la dichiarazione del metodo. L'ultima è una breve descrizione di cosa fa il metodo.

Le API vengono utilizzate dai programmatori per sviluppare software, se un API è chiara è più facile da usare e attrae più sviluppatori. Crescendo il numero di sviluppatori, aumenta anche la quantità di applicazioni scritte sulla piattaforma e di conseguenza la base di utenti. La correttezza di un'API è un elemento chiave per la popolarità di una piattaforma. Per questo motivo, le compagnie software impiegano ingenti quantità di tempo e risorse per realizzare API eleganti.
La specifica dell'API di Java contiene svariati elementi originali: la selezione, l'organizzazione e l'arrangiamento dei pacchetti, il nome e la definizione di una classe, la dichiarazione dei metodi e la descrizione di ogni elemento presente nell'API. Quando gli architetti delle API Java hanno deciso di includere particolari insiemi di classi nella piattaforma e di assegnare a quelle classi nomi specifici, hanno agito avendo un ampio spettro di possibilità. Per esempio, il pacchetto java.io, è legato alla gestione di input ed output. I programmatori hanno quindi incluso in tale pacchetto classi come java.io.PipedInputStream e java.io.PipedOutputStream che controllano rispettivamente input ed output. Tale scelta non è stata dettata da standard, è stata una scelta volontaria e ragionata degli sviluppatori. L'organizzazione dell'albero gerarchico riflette ulteriormente la creatività degli architetti. Nella Figura 3.1 vediamo un sottoinsieme dell'albero gerarchico del pacchetto java.io, la divisione tra classi per input ed output avviene ad alto livello.

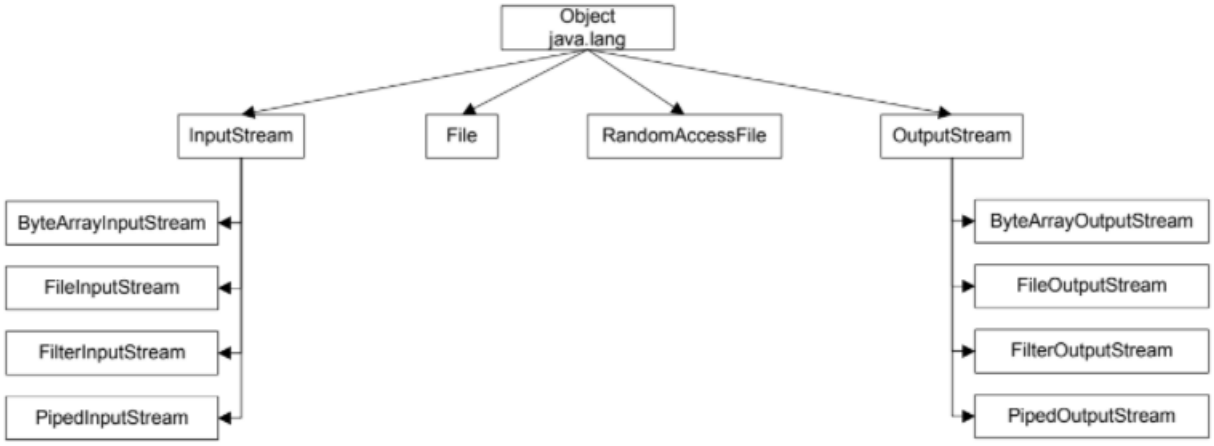


Figura 3.1: Gerarchia di un sottoinsieme di classi del pacchetto java.io[17].

Gli sviluppatori hanno voluto trasmettere l'idea che le classi di input, posizionate vicine nell'albero, condividono caratteristiche simili tra loro. I programmatori avrebbero potuto raggruppare diversamente le classi, posizionando tutte le "ByteArray-", "File-","Filter-" e "Piped-" insieme, prima di separare input ed output. La struttura così creata non avrebbe influenzato le funzionalità delle classi, ma sarebbe risultata più confusionaria. La decisione di inserire le classi in particolari pacchetti è un ulteriore prova di inventiva. Proseguendo con l'esempio precedente, invece che gestire i dati di input ed output nel pacchetto java.io, gli architetti avrebbero potuto dividerli in pacchetti separati, la visibilità delle classi sarebbe cambiata ma non la funzionalità delle stesse.
All'interno di ogni classe si aprono nuove opportunità di espressione originale, quali la selezione e l'ordinamento dei campi e metodi da includere. Inoltre, anche le componenti di un metodo richiedono creatività, comportano la scelta e l'arrangiamento dei parametri di input per una funzione. Ne è un esempio il metodo "exec" dell'API di Java, del quale ne esistono sei versioni differenziate dalla lista dei parametri, il computer tratta ogni versione diversamente. Le sei versioni sono:

- exec(String command)
- exec(String command, String[] envp)
- exec(String[] command, String[] envp, File dir)

- exec(String[] cmdarray)
- exec(String[] cmdarray, String[] envp)
- exec(String[] cmdarray, String[] envp, File dir)

Infine, anche i commenti che descrivono il codice sono una prova di originalità degli architetti Java. I commenti non sono funzionali, cioè non sono compilati per l'esecuzione da un computer, tuttavia sono fondamentali in quanto sono utilizzati per fornire informazioni utili sul codice.

Confronto codice e analisi copia

Il confronto tra i due codici in disputa si è svolto su tre livelli:

1. verifica delle somiglianze tra l'API di Android e l'API di Java;
2. verifica delle somiglianze tra il codice sorgente di Android e l'API di Java;
3. verifica delle somiglianze tra il codice sorgente di Android e il sorgente di Java.

API di Android e Java a confronto Per quest'analisi, sono state prese in considerazione l'API di Java 1.5 (JDK 5)[3], contenente 166 pacchetti, e l'API della versione 2.2 di Android[4] ("Froyo" o API Level 8), contenente 157 pacchetti. Il confronto visuale tra le due API dimostra che Java e Android strutturano classi e pacchetti in modo identico. La Figura 3.2 mostra l'organizzazione di un sottoinsieme delle classi del pacchetto java.util. E' facile osservare la simmetria delle due strutture e l'uguaglianza carattere per carattere delle descrizioni delle classi.

[3]disponibili al link https://developer.android.com/reference/packages.
[4]disponibili al link https://docs.oracle.com/javase/1.5.0/docs/api/.

Class Summary	
AbstractCollection<E>	This class provides a skeletal implementation of the Collection interface, to minimize the effort required to implement this interface.
AbstractList<E>	This class provides a skeletal implementation of the List interface to minimize the effort required to implement this interface backed by a "random access" data store (such as an array).
AbstractMap<K,V>	This class provides a skeletal implementation of the Map interface, to minimize the effort required to implement this interface.
AbstractQueue<E>	This class provides skeletal implementations of some Queue operations.
AbstractSequentialList<E>	This class provides a skeletal implementation of the List interface to minimize the effort required to implement this interface backed by a "sequential access" data store (such as a linked list).
AbstractSet<E>	This class provides a skeletal implementation of the Set interface to minimize the effort required to implement this interface.
ArrayList<E>	Resizable-array implementation of the List interface.
Arrays	This class contains various methods for manipulating arrays (such as sorting and searching).

Classes

AbstractCollection<E>	This class provides a skeletal implementation of the Collection interface, to minimize the effort required to implement this interface.
AbstractList<E>	This class provides a skeletal implementation of the List interface to minimize the effort required to implement this interface backed by a "random access" data store (such as an array).
AbstractMap<K, V>	This class provides a skeletal implementation of the Map interface, to minimize the effort required to implement this interface.
AbstractMap.SimpleEntry<K, V>	An Entry maintaining a key and a value.
AbstractMap.SimpleImmutableEntry<K, V>	An Entry maintaining an immutable key and value.
AbstractQueue<E>	This class provides skeletal implementations of some Queue operations.
AbstractSequentialList<E>	This class provides a skeletal implementation of the List interface to minimize the effort required to implement this interface backed by a "sequential access" data store (such as a linked list).
AbstractSet<E>	This class provides a skeletal implementation of the Set interface to minimize the effort required to implement this interface.
ArrayDeque<E>	Resizable-array implementation of the Deque interface.
ArrayList<E>	Resizable-array implementation of the List interface.
Arrays	This class contains various methods for manipulating arrays (such as sorting and searching).

Figura 3.2: Confronto delle classi presenti nel pacchetto java.util. Sopra Oracle, sotto Android le cui classi in grigio sono state inserite in versioni successive a *Froyo*.

La copia citata non è isolata, ma ripetuta per tutti e 37 i pacchetti in Figura 3.3, aventi nomi e struttura uguali per le due piattaforme. In aggiunta, tutti i pacchetti sono simili anche nel contenuto: nomi di classi, definizioni, campi e metodi.

java.awt.font	java.nio.channels.spi	java.util	javax.net.ssl
java.beans	java.nio.charset	java.util.jar	javax.security.auth
java.io	java.nio.charset.spi	java.util.logging	javax.security.auth.callback
java.lang	java.security	java.util.prefs	javax.security.auth.login
java.lang.annotation	java.security.acl	java.util.regex	javax.security.auth.x500
java.lang.ref	java.security.cert	java.util.zip	javax.security.cert
java.lang.reflect	java.security.interfaces	javax.crypto	javax.sql
java.net	java.security.spec	javax.crypto.interfaces	
java.nio	java.sql	javax.crypto.spec	
java.nio.channels	java.text	javax.net	

Figura 3.3: Lista dei 37 pacchetti copiati da Google[17].

Analizzando uno di questi pacchetti nel dettaglio (java.nio) emerge ancora di più l'evidenza della copia. Java.nio contiene dieci classi (Buffer, ByteBuffer, ByteOrder, CharBuffer, DoubleBuffer, FloatBuffer, IntBuffer, LongBuffer, MappedByteBuffer e ShortBuffer) e quattro eccezioni (BufferOverflowException, BufferUnderflowException, InvalidMarkException, ReadOnlyBufferException) tutte con nomi uguali tra le due API. Spostando lo sguardo sui metodi di una delle dieci classi (IntBuffer) notiamo che le dichiarazioni dei metodi sono pressoché identiche:

Metodi java.nio.IntBuffer (Java)	**Metodi java.nio.IntBuffer (Android)**
static IntBuffer allocate(int capacity)	static IntBuffer allocate(int capacity)
int[] array	final int[] array
int arrayOffset()	final int arrayOffset()
abstract IntBuffer asReadOnlyBuffer()	abstract IntBuffer asReadOnlyBuffer()
abstract IntBuffer compact()	abstract IntBuffer compact()
int compareTo(IntBuffer that)	int compareTo(IntBuffer otherBuffer)
abstract IntBuffer duplicate()	abstract IntBuffer duplicate()
boolean equals(Object ob)	boolean equals(Object ob)
abstract int get()	abstract int get()
abstract int get(int index)	abstract int get(int index)
IntBuffer get(int[] dst)	IntBuffer get(int[] dst)
IntBuffer get(int[] dst, int offset,	IntBuffer get(int[] dst, int dstOffset,

int length)	int intCount)
boolean hasArray()	final boolean hasArray()
int hashCode()	int hashCode()
abstract boolean isDirect()	abstract boolean isDirect()
abstract ByteOrder order()	abstract ByteOrder order()
abstract IntBuffer put(int i)	abstract IntBuffer put(int i)
IntBuffer put(int[] src)	final IntBuffer put(int[] src)
IntBuffer put(int[] src, int offset, int length)	IntBuffer put(int[] src, int srcOffset, int intCount)
IntBuffer put(IntBuffer src)	IntBuffer put(IntBuffer src)
abstract IntBuffer put(int index, int i)	abstract IntBuffer put(int index, int i)
abstract IntBuffer slice()	abstract IntBuffer slice()
String toString()	(ereditato dalla classe padre)
static IntBuffer wrap(int[] array)	static IntBuffer wrap(int[] array)
static IntBuffer wrap(int[] array, int offset, int length)	static IntBuffer wrap(int[] array, int start, int intCount)

Tabella 3.1: Dichiarazioni dei metodi della classe java.nio.IntBuffer[17].

Dei venticinque metodi nella versione Java, ventiquattro sono presenti nella versione Android, di questi quindici sono esattamente identici, come si evince dalla Tabella 3.1. Dei rimanenti nove, quattro variano solo per la parola chiave "final", che impedisce alle sottoclassi di modificare il comportamento del metodo, e gli altri cinque solo per il nome dei parametri. Entrambi i cambiamenti sono ininfluenti e non nascondono l'evidente copia letterale.

Un'ulteriore confronto tra le API è stato realizzato da Alan Purdy, il quale ha verificato le somiglianze tra il JDK 5 e Android API Level 3 (*Cupcake*) e API Level 8 (*Froyo*). Per realizzare entrambe le analisi ha sviluppato un programma Java capace di estrarre le informazioni volute dalle due API. In particolare, partendo dalla lista dei pacchetti

analizzati (solo i 37 interessati), il programma leggeva la documentazione online delle API (sui siti di Oracle e Google) e durante la scansione delle pagine web annotava le informazioni rilevanti, quali il numero di classi, interfacce, enumerazioni ed errori legate ad un pacchetto. Successivamente, riportava le informazioni ottenute su un database Microsoft Access per un'analisi accurata. I risultati ottenuti da Purdy sono i seguenti:

Package Elements Under Analysis	Java 5 Count	Android API Level 3 Count	In Java 5 but not Android 3 Count (%)	In Android 3 but not Java 5 Count	In Android 3 and Java 5 Count (%)
Packages	37	37	0	0	37 (100%)
Annotation Types	7	0	7 (100%)	0	0 (0%)
Classes	506	458	48 (9.4%)	0	458 (90.6%)
Enums	9	9	0 (0%)	0	9 (100%)
Errors	25	25	0 (0%)	0	25 (100%)
Exceptions	176	165	11 (6.25%)	0	165 (93.75%)
Interfaces	171	158	13 (7.6%)	0	158 (92.4%)
Totals	**931**	**852**	**79**	**0**	**852**

Figura 3.4: Risultati del confronto dei 37 pacchetti per JDK5 e Android API Level 3[18].

Package Elements Under Analysis	Java 5 Count	Android API Level 8 Count	In Java 5 but not Android 8 Count (%)	In Android 8 but not Java 5 Count	In Android 8 and Java 5 Count (%)
Packages	37	37	0	0	37 (100%)
Annotation Types	7	0	7 (100%)	0	0 (0%)
Classes	509	458	48 (9.4%)	0	458 (90.6%)
Enums	9	9	0 (0%)	0	9 (100%)
Errors	25	25	0 (0%)	0	25 (100%)
Exceptions	176	165	11 (6.25%)	0	165 (93.75%)
Interfaces	171	158	13 (7.6%)	0	158 (92.4%)
Totals	**934**	**852**	**86**	**0**	**852**

Figura 3.5: Risultati del confronto dei 37 pacchetti per JDK5 e Android API Level 8[18].

Dopo aver catturato le informazioni necessarie sui pacchetti, l'ingegner Purdy ha eseguito un'ulteriore analisi sulle classi contenute nei 37 pacchetti, verificando la somiglianza per metodi, costruttori e campi. Android API Level 3 non ha 48 delle classi presenti nel JDK5, l'analisi quindi riguarda solo le 458 classi presenti in entrambe le librerie. La tabella in Figura 3.6 mostra il numero totale di campi, costruttori e metodi presenti nelle 458 classi comuni. L'ultima colonna confronta ogni elemento (nomi, parametri, tipi) delle tre componenti analizzate e conta i casi identici, i risultati parlano chiaro il 70,91% degli elementi presenti nelle API di *Cupcake* è derivato dall'API Java.

Class Elements Compared	Java 5 Count	Android API Level 3 Count	Java 5 Element Matches Android 3 Count
Fields	1062	927	898
Constructors	745	639	438
Methods	5080	5052	3357
Totals	**6887**	**6618**	**4693**

Figura 3.6: Risultati del confronto delle classi presenti nei 37 pacchetti per JDK5 e Android API Level 3 [18].

La tabella in Figura 3.7 riporta solo le classi che dall'analisi precedente hanno segnalato un numero diverso di metodi tra le due piattaforme. Nella seconda e terza colonna sono presenti le quantità di metodi per classe, nell'ultima colonna è segnata la differenza tra le due quantità.

Classes With Different Number of Methods Per Class			
Name	Android 3 Method Count	Java 5 Method Count	Difference (Abs)
java.io.BufferedOutputStream	4	3	1
java.io.FileDescriptor	3	2	1
java.io.RandomAccessFile	40	39	1
java.lang.Class	53	57	4
java.lang.Enum	10	9	1
java.lang.Package	17	18	1
java.lang.reflect.Array	20	21	1
java.lang.reflect.Constructor	17	18	1
java.lang.reflect.Field	30	31	1
java.lang.reflect.Method	21	22	1
java.net.HttpURLConnection	16	17	1
java.net.Inet4Address	10	14	4
java.net.Inet6Address	16	19	3
java.nio.Buffer	14	13	1
java.nio.ByteBuffer	56	57	1
java.nio.DoubleBuffer	24	25	1
java.nio.FloatBuffer	24	25	1
java.nio.IntBuffer	24	25	1
java.nio.LongBuffer	24	25	1
java.nio.ShortBuffer	24	25	1
java.text.BreakIterator	22	25	3
java.text.CollationKey	3	5	2
java.text.DateFormat.Field	2	3	1
java.text.DateFormatSymbols	20	19	1
java.text.DecimalFormatSymbols	32	31	1
java.util.ArrayList	23	20	3
java.util.IdentityHashMap	14	15	1
java.util.jar.JarFile	6	5	1
java.util.regex.Pattern	11	10	1
java.util.TreeMap	16	18	2
java.util.zip.DeflaterOutputStream	6	5	1
java.util.zip.ZipInputStream	6	7	1
javax.security.auth.Subject	12	16	4
TOTAL	**620**	**644**	**50**

Figura 3.7: Conta e differenza del numero dei metodi per le classi con una quantità differente di metodi[18].

L'analisi sulle classi è stata effettuata anche per Android Level 8, ottenendo i dati in Figura 3.8. Anche in questo caso, sono state considerate le 458 classi comuni, poiché *Froyo* come *Cupcake* non contiene 48 classi del JDK5.

Class Elements Under Analysis	Java 5 Count	Android API Level 8 Count	Java 5 Element Matches Android 8 Count
Fields	1062	927	898
Constructors	745	639	438
Methods	5080	5052	3357
Total	**6887**	**6618**	**4693**

Figura 3.8: Risultati del confronto delle classi presenti nei 37 pacchetti per JDK5 e Android API Level 8[18].

In seguito, Alan Purdy ha svolto una seconda analisi, confrontando i campi privati definiti nell'Android API Level 8 e nel JDK5. Nello specifico ha confrontato i campi privati tra le 458 classi comuni, verificando quali avevano nome, nome e tipo o l'intera dichiarazione (inclusi i *modifiers*[5]) uguali. Ha nuovamente svolto lo studio tramite un programma Java da lui scritto. Il programma, partendo dalla lista delle 458 classi, le controllava singolarmente estraendo il nome, il tipo e i *modifiers* degli elementi privati. Le informazioni venivano poi riportate manualmente su un database Microsoft Access. Infine, un insieme di script verificava la somiglianza tra i nomi, i tipi e i modificatori di accesso. La Figura 3.9 evidenzia come il 94,7% delle classi contenenti un campo privato, ne abbia uno duale tra le API a confronto.

	JDK 5 Count	**Froyo Count**	**Classes With Matching Names**
Classes in 37 Packages	506	458	458
Classes with Private Fields	324	317	297

Figura 3.9: Confronto tra Android API Level 8 e JDK5 delle classi totali contenenti campi uguali e classi con campi privati contenenti campi uguali[19].

In Figura 3.10 sono presenti i risultati separati per nome, nome e tipo ed intera dichiarazione identica.

[5]Sono parole chiave inserite nella dichiarazione di un campo che impostano il livello di accesso del campo. Ad esempio, il *modifier public* permette l'accesso da classi esterne.

Private Field Metric	Total
Private Fields in JDK 5's Shared Classes	1831
Private Fields in Froyo's Shared Classes	1383
Private Fields with Matching Names	516
Private Fields with Matching Names and Types	490
Private Fields with Matching Signatures (Modifier, Name, and Type)	384

Figura 3.10: Conta dei campi privati con stesso nome, stesso nome e tipo e dichiarazione totale completa[19].

Un'anomalia si è verificata durante la realizzazione di questo studio. Una classe Java può essere annidata, cioè definita all'interno di un'altra classe. Per esempio, la classe GetField è definita all'interno della classe java.io.ObjectInputStream. Una classe annidata non è caricabile indipendentemente dalla sua classe padre. Come risultato, a tempo di esecuzione, il programma falliva quando provava a caricare la classe figlio, fortunatamente l'errore era rilevato. Sia JDK5 che Froyo hanno fallito a caricare le stesse 23 classi annidate. L'effetto dell'esclusione delle 23 classi è la potenziale sottostima delle componenti simili.

Sorgente di Android e API di Java a confronto Dal paragrafo precedente è emerso come l'API di Android contenga nomi, definizioni, campi e metodi sostanzialmente simili ai corrispondenti elementi nell'API di Oracle. Il codice Android che implementa le specifiche dell'API include le componenti copiate. Il sorgente di Android è presente al link `https://android.googlesource.com/`. Ad esempio, la cartella "security" contiene i sorgenti (*Froyo*), scritti in Java, con cui Google intendeva implementare le diverse componenti del pacchetto java.security. Un'altra prova è la classe java.security.ProtectionDomain che stando all'API di Java ha due costruttori:

- ProtectionDomain(CodeSource codesource, PermissionCollection permissions)
- ProtectionDomain(CodeSource codesource, PermissionCollection permissions, ClassLoader classloader, Principal[] principals)

Inoltre, la specifica indica sei metodi per questa classe con le seguenti dichiarazioni:

- public final CodeSource getCodeSource()
- public final ClassLoader getClassLoader()
- public final Principal[] getPrincipals()
- public final PermissionCollection getPermissions()
- public boolean implies(Permission permission)
- public String toString()

Il file ProtectionDomain.java sviluppato da Android copia le seguenti linee senza modificarle, cambiando unicamente il codice implementativo. ProtectionDomain.java è solo un esempio della copia trovata nel sorgente di Android, l'infrazione si rileva in centinaia di altri file. Il contenuto di ognuna di queste cartelle specchia il contenuto del corrispondente pacchetto Java: security (java.security), cert (java.security.cert), math (java.math), lang (java.lang) e nio (java.nio).
Inoltre, Google ha creato e distribuito, attraverso le sue *repository* online, classi[6] scritte in C che sono sostanzialmente simili alle corrispondenti classi di Google. Dall'esempio in Figura 3.11, vediamo che i due codici eseguono la stessa funzione, cioè bloccare un *thread*, cioè una suddivisione di un processo, per una quantità di tempo specificata in millisecondi o nanosecondi. Il commento sopra il codice di Google mostra proprio come si stessero ispirando alla funzione "wait" di Oracle.

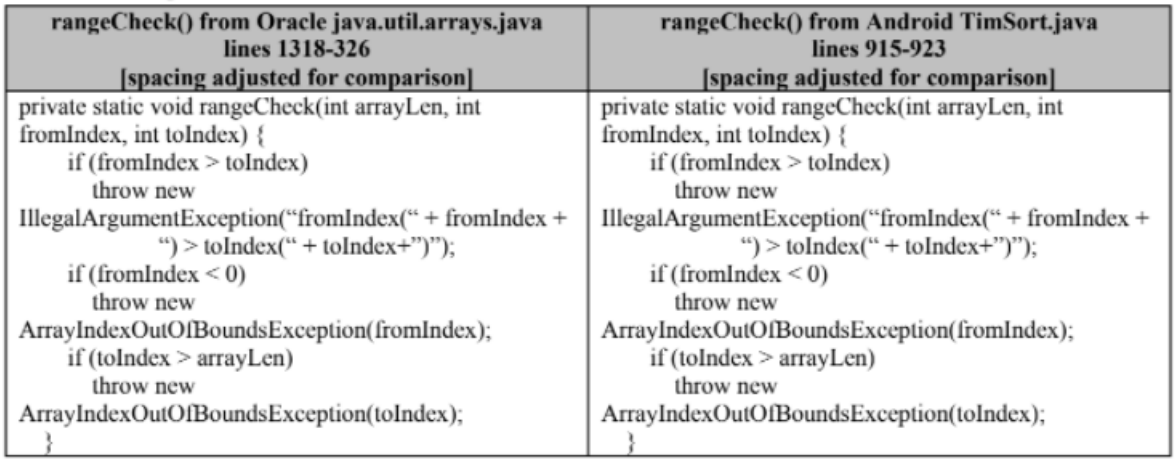
[6]Le classi incriminate sono le seguenti: java.lang.Object, java.lang.reflect.AccessibleObject, java.lang.reflect.Array, java.lang.reflect.Constructor, java.lang.reflect.Field, java.lang.reflect.Method, java.lang.reflect.Proxy, java.lang.Runtime, java.lang.String, java.lang.System, java.lang.Throwable, java.lang.VMClassLoader, java.lang.VMThread, java.security.AccessController.

```
  public final void wait(long ms, int ns)
    throws IllegalMonitorStateException, InterruptedException
  {
    if (ms < 0 || ns < 0 || ns > 999999)
      throw new IllegalArgumentException("argument out of range");
    VMObject.wait(this, ms, ns);
  }
} // class Object
/*
 * public void wait(long ms, int ns) throws InterruptedException
 */
static void Dalvik_java_lang_Object_wait(const u4* args, JValue* pResult,
    const Method* method, Thread* self)
{
    Object* thisPtr = (Object*) args[0];

    dvmObjectWait(self, thisPtr, GET_ARG_LONG(args,1), (s4)args[3], true);
    RETURN_VOID();
}
```

Figura 3.11: Implementazioni della funzione wait di Google (scritta in C, sotto) e di Oracle (scritta in Java, sopra).

Sorgente di Android e Java a confronto Google ha distribuito tramite Android codice sorgente e oggetto sostanzialmente simile al codice sorgente di Oracle o al codice oggetto di Oracle decompilato. Due file sorgenti Android, TimSort.java e ComparableTimSort.java, contengono linee di codice del metodo rangeCheck che sono esattamente identiche alle linee contenute nella classe java.util.Arrays. In questo caso anche il codice implementativo è esattamente uguale, vedasi Figura 3.12.



rangeCheck() from Oracle java.util.arrays.java lines 1318-326 [spacing adjusted for comparison]	rangeCheck() from Android TimSort.java lines 915-923 [spacing adjusted for comparison]
private static void rangeCheck(int arrayLen, int fromIndex, int toIndex) { if (fromIndex > toIndex) throw new IllegalArgumentException("fromIndex(" + fromIndex + ") > toIndex(" + toIndex+")"); if (fromIndex < 0) throw new ArrayIndexOutOfBoundsException(fromIndex); if (toIndex > arrayLen) throw new ArrayIndexOutOfBoundsException(toIndex); }	private static void rangeCheck(int arrayLen, int fromIndex, int toIndex) { if (fromIndex > toIndex) throw new IllegalArgumentException("fromIndex(" + fromIndex + ") > toIndex(" + toIndex+")"); if (fromIndex < 0) throw new ArrayIndexOutOfBoundsException(fromIndex); if (toIndex > arrayLen) throw new ArrayIndexOutOfBoundsException(toIndex); }

Figura 3.12: Confronto della funzione rangeCheck()[17].

Seppur vero che la funzione è lunga solo nove linee su un file (arrays.java) che ne conta 3180 in totale, è un metodo significativo che viene chiamato nove volte da altri metodi all'interno della classe. In aggiunta, il codice di otto programmi Android risulta essere identico al codice decompilato di classi Java. Un compilatore converte codice sorgente (file ".java", nel caso specifico) in codice oggetto (file ".class"), nel processo di compilazione vengono trascurati i commenti poiché non eseguibili da un computer. Il decompilatore opera inversamente, genera codice sorgente a partire da codice oggetto. I decompilatori Java sono abbastanza accurati, a partire dal codice oggetto riescono ad ottenere un sorgente di buon livello. Per questo motivo, Oracle ha ingaggiato Marc Visnick, un consulente software dell'azienda Johnson-Laird Inc., per investigare sulla presenza di file in Android, ottenuti tramite la decompilazione del codice oggetto di Java, disponibile sul sito di Oracle. I consulenti hanno usato il software JAD, un decompilatore Java, sulle librerie del JDK5. I file così ottenuti sono stati confrontati con i file sorgenti di *Froyo*. Otto file (in Tabella 3.2) sono risultati quasi identici, diversi solo per la spaziatura. In particolare, i nomi delle variabili "set1" o "flag1" presenti nel file PolicyNodeImpl.class sono sospetti poiché lontani dalle *best practices* di nomina di componenti. Uno sviluppatore professionista preferisce assegnare nomi più descrittivi alle variabili, al contrario un decompilatore può assegnare questi nomi seguendo una regola.

File Oracle decompilati	Corrispondenti file Android
PolicyNodeImpl.class	PolicyNodeImpl.java
AclEntry.class	AclEntryImpl.java
AclImpl.class	AclImpl.java
GroupImpl.class	GroupImpl.java
OwnerImpl.class	OwnerImpl.java
PermissionImpl.class	PermissionImpl.java
PrincipalImpl.class	PrincipalImpl.java
AclEnumerator.class	AclEnumerator.java

Tabella 3.2: Otto file risultati copiati dall'analisi tramite decompilazione[17].

Infine, due file Android (CodeSourceTest.java e CollectionCertStoreParametersTest.java) contengono commenti che sono pressoché identici ai corrispondenti file di Oracle (CodeSource.java e CollectionCertStoreParameters.java). Gli autori dei due file Android hanno apparentemente estratto i commenti dal codice Oracle o dalla documentazione delle API di Oracle e li hanno inseriti nel loro codice, per spiegare come il codice di test Android misuri le funzionalità del pacchetto java.security.

CodeSource.java (Java version)
[spacing adjusted for comparison]

Lines 242-59

```
*     <li>  If this object's port (getLocation().getPort()) is not
*           equal to -1 (that is, if a port is specified), it must equal
*           <i>codesource</i>'s port.
*
*     <li>  If this object's file (getLocation().getFile()) doesn't equal
*           <i>codesource</i>'s file, then the following checks are made:
*           If this object's file ends with "/-",
*           then <i>codesource</i>'s file must start with this object's
*           file (exclusive the trailing "-").
*           If this object's file ends with a "/*",
*           then <i>codesource</i>'s file must start with this object's
*           file and must not have any further "/" separators.
*           If this object's file doesn't end with a "/",
*           then <i>codesource</i>'s file must match this object's
*           file with a '/' appended.
*
*     <li>  If this object's reference (getLocation().getRef()) is
*           not null, it must equal <i>codesource</i>'s reference.
```

Figura 3.13: Commenti nel file CodeSource di Oracle[17].

CodeSourceTest.java (Android version)
[spacing adjusted for comparison]

Lines 598-601
```
/**
* If this object's port (getLocation().getPort()) is not equal to -1 (that
* is, if a port is specified), it must equal codesource's port.
*/
```

Lines 629-32
```
/**
* If this object's file (getLocation().getFile()) doesn't equal
* codesource's file, then the following checks are made: ...
*/
```

Lines 645-48
```
/**
* ... If this object's file ends with "/-", then codesource's file must
* start with this object's file (exclusive the trailing "-").
*/
```

Lines 667-81
```
/**
* ... If this object's file ends with a "/*", then codesource's file must
* start with this object's file and must not have any further "/"
* separators.
*/
```

Lines 693-96
```
/**
* ... If this object's file doesn't end with a "/", then codesource's file
* must match this object's file with a '/' appended.
*/
```

Lines 711-14
```
/**
* If this object's reference (getLocation().getRef()) is not null, it must
* equal codesource's reference.
*/
```

Figura 3.14: Commenti nel file CodeSourceTest di Google[17].

Analisi compatibilità

E' ormai chiaro, secondo gli esperti lato Oracle, che Google abbia copiato 37 dei 166 pacchetti dell'API di Java, non è però chiaro l'effetto che ha questa copia sulla piattaforma Java. Android avendo copiato solo un sottoinsieme dei pacchetti, compromettendo l'interoperabilità di Java. Applicazioni che dovrebbero eseguire su una piattaforma che supporta l'API di Java, su Android potrebbero non funzionare, poiché contenenti riferimenti ai 129 pacchetti non copiati. Per esempio, i programmatori Java vorrebbero

utilizzare pacchetti quali java.awt.font, java.lang.management o java.rmi, tuttavia i seguenti pacchetti non sono supportati nel codice Android, eliminando così la possibilità di sviluppare interfacce utente con quei pacchetti. Generalizzando, la frammentazione della piattaforma comporta l'interruzione dell'interoperabilità. Applicazioni sviluppate per Java potrebbero non eseguire su Android e viceversa. Inoltre, la frammentazione causa problemi anche in fase di sviluppo, in quanto i programmatori devono preoccuparsi di scrivere e testare codice differente a seconda della piattaforma. La frammentazione attuata da Google attacca il motto primario di Java " *Write once, run anywhere*" (WORA).

Oltre ad avere creato un sottoinsieme delle API di Java, Google ha anche implementato ed aggiunto diversi pacchetti non supportati dalla piattaforma Java. Ha così creato un sovrainsieme delle API di Java, portando allo stesso risultato precedente, l'alta probabilità di incompatibilità tra le due piattaforme. L'interesse di Google non era quello di rendere Android e Java interoperabili e compatibili, altrimenti avrebbe implementato la totalità dei pacchetti Java. Per di più, ha creato Android, un'implementazione indipendente, senza passare attraverso le licenze di Sun, che se concesse permettono la creazione di implementazioni indipendenti, seppur compatibili. La politica delle licenze di Oracle è basata sull'idea che la frammentazione e il *forking* sono minacce al raggiungimento della premessa WORA. Google stessa riconosce l'impatto della frammentazione di una piattaforma, infatti, nonostante aver parzialmente compromesso la compatibilità con Java, impone la compatibilità delle API per la sua piattaforma per evitare implementazioni indipendenti[20].

3.1.2 Argomentazioni di Google

Google ha risposto all'attacco di Oracle sollevando i punti seguenti:

1. Sun ha permesso l'uso libero di Java, incoraggiando l'uso delle sue API, facendo credere anche quest'ultime libere;
2. ha implementato indipendentemente la funzionalità dei 37 pacchetti in dibattito;
3. Il codice dichiarativo dell'API è una piccola porzione del codice totale di Android;

4. l'uso fatto dell'API rientra nel *fair use*.

Svariati esperti sono stati chiamati ad esporre le ragioni di Google, tra tutti, Owen Astrachan[7], professore di informatica all'università di Duke, è stato il più costante ed influente. Le argomentazioni esposte da Astrachan sono riportate di seguito.

Codice implementativo e dichiarativo

Ogni API, inclusa l'API Java in discussione, esiste in due forme: codice dichiarativo (che include gli elementi discussi precedentemente: nome, argomenti e valore di ritorno) e codice implementativo che include la logica del programma che realizza lo scopo del metodo. E' importante capire che le due parti sono diverse ma non scollegate, il codice dichiarativo fa parte anch'esso dell'implementazione dell'API. Un'API potrebbe avere più implementazioni, cioè algoritmi differenti che realizzano lo stesso risultato, tuttavia le diverse implementazioni devono necessariamente condividere il codice dichiarativo per operare tra di loro. In caso di codice dichiarativo differente, i software che si affidano all'API cambiata fallirebbero nell'esecuzione, il software non riuscirebbe a trovare e ad accedere alle funzionalità di cui ha bisogno. Implementazioni differenti della stessa API devono usare gli stessi elementi per evitare confusione, inefficienza e incompatibilità. Altre porzioni dell'implementazione di un'API (codice implementativo) non direttamente governate dal codice dichiarativo, variano tra le differenti implementazioni. Ad esempio, linguaggi di programmazione diversi possono essere usati per implementare un'API. Nel caso di Android, sia Java che C sono usati per scrivere codice implementativo. Le due implementazioni, tranne le parti richieste per compatibilità (codice dichiarativo), è difficile siano identiche se scritte da programmatori o compagnie differenti. Tuttavia, poiché limitate dall'API e da considerazioni pratiche, come l'efficienza del programma, alcune porzioni di codice implementativo possono sembrare simili. E' il caso di fare un esempio, prendiamo la funzione "abs" del paragrafo precedente. In Figura 3.15 è mostrata la documentazione dell'API Android per il metodo "abs". La prima linea "public static int abs(int i)" è la dichiarazione del metodo. Le linee restanti sono commenti che danno informazioni utili per usare il metodo.

[7]La pagina web del professor Astrachan è disponibile all'indirizzo: https://users.cs.duke.edu/~ola/

```
public static int abs(int i)

Returns the absolute value of the argument.

If the argument is Integer.MIN_VALUE,
Integer.MIN_VALUE is returned.

Parameters

   i     the value whose absolute value has to be
computed.

Returns

   the argument if it is positive, otherwise the
negation of the argument.
```

Figura 3.15: Documentazione dell'API Android per il metodo abs[21].

Il codice sorgente che implementa il metodo "abs" è in Figura 3.16. La prima linea "package.java.lang" indica il nome del pacchetto API in cui il metodo risiede e suggerisce che questo file contiene una classe che fa parte di quel pacchetto. "Public final class Math" fa anch'essa parte dell'organizzazione, riflette la classe che contiene il metodo "abs". Entrambi queste linee (in nero nella figura) devono essere presenti, esattamente in questa forma, in tutte le possibili implementazioni del metodo Math.abs. Le linee in blu sono commenti. Infine, l'effettivo codice sorgente del metodo è mostrato in verde e rosso. Inizia ripetendo la dichiarazione del metodo "public static int abs(int i)" (in verde), presentando di seguito l'effettiva logica dello stesso "return i >= 0 ? i : -i;". Questa porzione è il codice implementativo, che comunica effettivamente al computer che operazioni fare. In questo caso, la logica in italiano sarebbe; "se il numero in input è maggiore o uguale a 0, restituisci quel numero, altrimenti restituisci l'opposto del numero".

```
package java.lang;

...
/**
 * Class Math provides basic math constants and operations such as
trigonometric
 * functions, hyperbolic functions, exponential, logarithms, etc.
 */
public final class Math {
        ...
    /**
            * Returns the absolute value of the argument.
            * <p>
            * If the argument is {@code Integer.MIN_VALUE}, {@code
Integer.MIN_VALUE}
            * is returned.
            *
            * @param i
            *            the value whose absolute value has to be
computed.
            * @return the argument if it is positive, otherwise the
negation of the
            *         argument.
            */

    public static int abs(int i) {
        return i >= 0 ? i : -i;
    }
```

Figura 3.16: Codice sorgente di Android per il metodo “abs”[21].

L’intera porzione di codice unisce l’implementazione e la documentazione del metodo “abs”. Di questa, oltre alle necessarie linee “organizzative” in nero, solo la dichiarazione del metodo è identica tra le implementazioni del metodo “abs” delle due API (Figura 3.17). La dichiarazione identifica il metodo, deve necessariamente essere identica a quella della documentazione. L’uso della stessa dichiarazione è necessario se due implementa-

zioni vogliono essere compatibili, come Android e Java. Un'implementazione che usa solo le componenti dell'API necessarie, ma non ripete l'implementazione sottostante, è un'implementazione indipendente.

Android Math.abs	**Oracle JDK 1.5 Math.abs**

`public static int abs(int i) {`
`  return i >= 0 ? i : -i;`
`}`
	`public static int abs(int a) {`
	`  return (a < 0) ? -a : a;`
	`}`

Figura 3.17: Implementazioni metodo abs nelle due piattaforme[22].

La realizzazione della funzione "abs" è molto semplice, richiede una sola linea di codice. Nonostante la banalità, le implementazioni di Oracle e Android sono differenti. La prima linea è identica, come richiesto dall'API, tranne per il nome del parametro che varia, poiché non è necessario sia identico per ottenere compatibilità. La seconda linea in blu è il codice implementativo, mostra come viene calcolato il valore assoluto. Le implementazioni sono differenti, ma entrambe corrette. Quella di Android in italiano sarebbe: "se il parametro i è maggiore o uguale a 0, restituisci i, altrimenti l'opposto di i", quella di Oracle invece: "se il parametro a è minore di 0, restituisci l'opposto di a, altrimenti a". Le implementazioni catturano la stessa funzionalità, ma lo fanno attraverso algoritmi diversi.
Un'ulteriore esempio, simile ma più di impatto, è il metodo String.compareTo della classe java.lang.String. Nella programmazione, una "string" è una sequenza di caratteri, come una parola o una frase. Il metodo "compareTo" confronta due stringhe, per determinare se una è minore, uguale o maggiore dell'altra. In un programma, una stringa minore di un'altra è precedente nell'ordine alfabetico, tra "arpione" e "gatto" il metodo indicherebbe che arpione è minore di gatto. In Figura 3.18 vediamo l'implementazione di String.compareTo per Android e Oracle:

#	Android String.compareTo	Oracle JDK 1.5 String.compareTo
1	`public int compareTo(String string) {`	`public int compareTo(String anotherString) {`
2	`  // Code adapted from K&R, pg 101`	`  int len1 = count;`
3	`  int o1 = offset, o2 = string.offset, result;`	`  int len2 = anotherString.count;`
4	`  int end = offset + (count < string.count ? count : string.count);`	`  int n = Math.min(len1, len2);`
5	`  char[] target = string.value;`	`  char v1[] = value;`
6	`  while (o1 < end) {`	`  char v2[] = anotherString.value;`
7	`    if ((result = value[o1++] - target[o2++]) != 0) {`	`  int i = offset;`
8	`      return result;`	`  int j = anotherString.offset;`
9	`    }`	
10	`  }`	`  if (i == j) {`
11	`  return count - string.count;`	`    int k = i;`
12	`}`	`    int lim = n + i;`
13		`    while (k < lim) {`
14		`      char c1 = v1[k];`
15		`      char c2 = v2[k];`
16		`      if (c1 != c2) {`
17		`        return c1 - c2;`
18		`      }`
19		`      k++;`
20		`    }`
21		`  } else {`
22		`    while (n-- != 0) {`
23		`      char c1 = v1[i++];`
24		`      char c2 = v2[j++];`
25		`      if (c1 != c2) {`
26		`        return c1 - c2;`
27		`      }`
28		`    }`
29		`  }`
30		`  return len1 - len2;`
31		`}`

Figura 3.18: Implementazioni metodo String.compareTo nelle due piattaforme[22].

L'implementazione di Android è basata sul libro "The C Programming Language", libro sulla programmazione in C scritto da Brian Kernighan e Dennis Ritchie. Ciò emerge dal commento in linea due "code adopted from K&R" (Kernighan & Ritchie). Tra le due implementazioni l'unica stringa uguale è la prima, che come già spiegato è necessaria per compatibilità. Lo scopo dei due codici è lo stesso, ma l'algoritmo è differente. Per esempio, nel confrontare le stringhe "catalogo" e "catastrofe" entrambi i codici scansionano i primi quattro caratteri, non rilevando incongruenze. L'algoritmo determina poi

l'ordine delle stringhe, confrontando il quinto carattere, che è diverso tra le due parole, restituendo che "catalogo" è minore di "catastrofe". Nell'implementazione Android i due caratteri confrontati sono catturati dalle espressioni value[o1++] e target[o2++], mentre nel codice Java, i due sono memorizzati nelle variabili c1 e c2 e catturati dalle espressioni v1[i++] e v2[j++] in una parte del codice e v1[k] e v2[k] in un'altra parte. In entrambi i codici la differenza tra le stringhe confrontate è rilevata al primo carattere differente, evitando la totale scansione che sarebbe inefficiente.

In entrambi gli esempi mostrati in questa sezione la logica di programmazione usata per implementare lo stesso metodo è differente, tranne per una porzione, il codice dichiarativo. Il professor Astrachan afferma che questo *pattern* si rileva in tutti i file da lui analizzati[21], in questi solo il codice dichiarativo era identico tra le implementazioni di Oracle e Google. Pertanto, ritiene l'implementazione di Android sostanzialmente differente da quella di Oracle.

API come metodo di operazione

Per introdurre l'argomento vediamo un esempio di utilizzo dei metodi di un'API. Quando un programmatore sta scrivendo un'applicazione e vuole utilizzare una particolare funzionalità, deve invocarla chiamando il metodo appropriato. Se un programmatore Java vuole usare la funzione della radice quadrata Java per trovare la radice quadrata di 25, deve farlo inserendo nel programma il seguente codice:

```
double result = Math.sqrt(25.0);
```

L'argomento 25.0 è passato al metodo Math.sqrt che ritorna il risultato 5.0. In questo esempio, il valore di ritorno è poi salvato in una variabile chiamata "result". Per scrivere questo codice, un programmatore Java inesperto avrebbe dovuto cercare la funzione nella documentazione dell'API Java. Le volte successive, però, avrà imparato questa parte dell'API e non perderà tempo a cercare nuovamente la funzione. In nessun punto lo sviluppatore ha bisogno di conoscere la logica dietro alla funzione "sqrt", questo è il grande vantaggio delle API.
In informatica, i metodi sono di frequente chiamati anche funzioni. Entrambi i termini

suggeriscono, correttamente, che funzioni e metodi siano un mezzo funzionale per usare un software. Quando un metodo dell'API è chiamato con i parametri corretti, l'API invoca la funzionalità, legata al metodo invocato, dal sistema software sottostante, eseguendo il codice implementativo legato al metodo chiamato.

Il Copyright Act definisce il concetto di metodo operativo come il mezzo attraverso cui una persona opera qualcosa, sia questo qualcosa una macchina, un computer o altro[21]. Un'API, rientra in questa definizione, è un metodo operativo funzionale. Basti pensare che lo scopo principale di un'API è di creare un'interfaccia con un'altra applicazione. Quest'interfaccia non è una discussione sociale, ma un comando formale di un programma all'altro: "Fai questa operazione per me e avvisami quando hai finito". Il programma che comanda sta usando l'API per operare il programma sottostante all'API, il quale sta venendo operato per mezzo dell'API. E' difficile, se non impossibile utilizzare il programma sottostante se non attraverso l'uso di un API. Infatti, è l'API che fornisce meccanismi per usare le funzionalità a lei sottostanti. Come visto nell'esempio precedente della funzione "sqrt", è attraverso i nomi che possiamo usare il codice implementativo dei metodi. Solo invocando il metodo, scrivendo esattamente "sqrt" otteniamo che sul nostro parametro vengano realizzate le operazioni necessarie per ottenere la radice quadrata. I nomi dei metodi sono il metodo operativo attraverso cui invochiamo il codice sottostante, come il pedale del gas è il metodo operativo con cui invochiamo il motore.

Creatività API

I nomi degli elementi dell'API Java ripetono di frequente alcuni termini chiave e seguono regole meccaniche presenti nella specifica del linguaggio Java. Le regole forniscono suggerimenti per la struttura e l'assegnamento dei nomi. Ad esempio, per i metodi la specifica Java dice: "i nomi dei metodi dovrebbero essere verbi o frasi verbali, con la prima lettera in minuscolo e la prima di ogni seguente parola in maiuscolo", e ancora "i nomi delle classi dovrebbero essere nomi descrittivi o frasi nominali"[23].
Nell'implementazione di Java 1.5, quasi un terzo dei nomi dei metodi in disputa (2578 su 7796) sono determinati attraverso queste regole, inclusi quasi 2000 nomi che iniziano con

"get" o "set" e 164 chiamati "equals". Altri 2347 nomi di metodi sono parole singole, come "run" o "add". I rimanenti 2871 non sono nomi lunghi o complicati, in media sono lunghi 2,34 "parole" (contando locateURL come due parole e findBestMatch come tre parole). In Android, dei totali 9297 metodi, 2676 sono lunghi una parola, 2909 sono nomi dettati dalle regole (come "set" o "get"), i restanti 3712 sono metodi lunghi 2.41 parole in media. Seguire meccanicamente queste regole, cercando di creare consistenza e coerenza all'interno dell'API riduce notevolmente il grado di creatività necessario per scrivere un'API. Ancor più importante, riduce il tempo per imparare e memorizzare l'API, rendendola così più appetibile per gli sviluppatori. Di conseguenza ogni buon API ha regole come quelle contenute nella specifica di Java. Questo mostra, come l'assegnamento dei nomi sia un processo semplice e ben definito nell'industria informatica. I nomi sono funzionali e primariamente dettati da parametri di efficienza, lasciano spazio alla creatività.

Le stesse valutazioni appena fatte valgono anche per l'organizzazione dei metodi. L'organizzazione dell'API in gruppi funzionalmente connessi, riflette le funzionalità del programma sottostante e rende facile ai programmatori muoversi all'interno interfaccia. Per esempio, metodi legati alla sicurezza, come AccessController.checkPermission e Signature.sign, devono essere posti in pacchetti legati alla sicurezza quali java.security o java.secuirtyx, poiché è li che uno sviluppatore se li aspetta intuitivamente. La richiesta pratica che anche l'organizzazione di un'API sia legata al sistema sottostante e alle aspettativa dei programmatori, restringe il numero di possibili strutture e di conseguenza il livello di originalità necessario per l'organizzazione dell'API.

Sempre sullo stesso punto, Owen Astrachan risponde al professor Mitchell in merito all'esempio in Figura 3.1. Come precedentemente trattato, l'esperto di Oracle affermava che le diverse classi "ByteArray-", "File-", "Filter-" e "Piped-" avrebbero potuto essere divise in due classi Input ed Output diverse, senza che il cambiamento avesse ripercussioni sulle funzionalità delle classi. Il professor Astrachan reputa quest'affermazione incorretta. Infatti, l'organizzazione delle classi InputStream, OutputStream, Reader e di altre classi, del pacchetto java.io, non menzionate dall'esperto di Oracle, è basata sul design pattern

Decorator[8]. Decorator è conosciuto e largamente adottato nell'industria informatica. L'uso dei design pattern nell'organizzazione di un'API è obbligatorio se le API vogliono essere adottate e ben capite. Ricapitolando, la strutturazione di un'API non è frutto di un processo creativo, ma bensì di un processo funzionale che segue parametri ben noti nell'industria e cerca di ottenere specifici requisiti di qualità per aumentare la base di utilizzatori dell'interfaccia,

Confronto codice e analisi copia

API di Android e Java a confronto Per confrontare i due codici Owen Astrachan ha realizzato uno script Python, "SlocCounter.py", basato sul *tool* sloccount[9], uno strumento software che misura la grandezza del codice sorgente di grandi progetti. Attraverso lo script, ha misurato l'implementazione dell'API Android Gingerbread, risultata in 57.076 file e 15.347.169 linee di codice[10]. Il codice implementativo dei 37 pacchetti è costituito da 259.474 linee di codice e 1.022 file, rappresenta l'1.6% della grandezza del sorgente di Android[21]. Questo dato evidenzia come Android non sia sostanzialmente simile alla piattaforma Java, poiché i 37 pacchetti in disputa rappresentano solo una piccolissima porzione del codice Android. Il restante 98.4% non ha nessuna derivazione da Java.

L'esperto di Google ha poi proseguito esaminando i metodi privati per entrambe le implementazioni. Due implementazioni possono essere compatibili avendo metodi privati differenti, l'importante è chi i metodi pubblici siano gli stessi. Differenti nomi per i metodi privati mostrano che la copia del codice sorgente non sussiste, poiché indicano che le due implementazioni hanno strutture sottostanti differenti[22]. Pertanto, attraverso

[8]Nella programmazione ad oggetti, il decorator è uno dei pattern fondamentali, definiti originariamente dalla Gang of Four. Il design pattern decorator consente di aggiungere nuove funzionalità ad oggetti già esistenti. Questo viene realizzato costruendo una nuova classe decoratore che "avvolge" l'oggetto originale. Al costruttore del decoratore si passa come parametro l'oggetto originale. Questo pattern si pone come valida alternativa all'uso dell'ereditarietà singola o multipla.

[9]Il tool è disponibile al link: https://dwheeler.com/sloccount/

[10]Numeri realizzati eseguendo SlocCounter.py su una copia della versione Android Gingerbread ed includendo solo linee di codice dei file con estensione .h, .c, .cpp e .java.

un programma da lui scritto, il professor Astrachan ha analizzato i pacchetti accusati, in particolare le 740 classi e interfacce in comune tra le implementazioni, cercando le differenze negli elementi privati. I risultati sono nella Tabella 3.3. La colonna "Metodi totali" mostra il numero totale di metodi (inclusi costruttori) trovati, la colonna "Metodi privati totali" specifica quanti di questi sono privati. L'ultima colonna, "Percentuale privati", stima quanto spesso i metodi privati sono usati dalle classi. Ognuna delle classi contribuisce alla stima con una percentuale tra 0 e 100, se tutti i metodi di una classe sono privati, la percentuale fornita sarà 100%. Al contrario, se tutti i metodi sono pubblici, la percentuale privata sarà 0%. La differenza tra Android e Oracle in quest'ultima metrica è significativa. Indica che le classi di Android usano in media meno metodi privati rispetto alle corrispondenti classi di Oracle, questo testimonia che le due implementazioni sono strutturalmente differenti.

	Pacchetti	**Metodi totali**	**Metodi privati totali**	**Percentuale privati**
Android Gingerbread	37	8994	970	5.92%
JDK5	37	8190	1369	7.17%

Tabella 3.3: Confronto tra Android e Java sul numero di metodi pubblici e privati[22].

Documentazione API Android e Java a confronto La documentazione di un API deve soddisfare requisiti pratici, deve descrivere esattamente le funzionalità di un elemento dell'API. Per questo motivo, è raro trovare due documentazioni differenti riferite allo stesso elemento. Come due scrittori di un dizionario che devono definire la parola zebra useranno entrambi parole come "strisce" "bianco" e "nero", due autori di un'API si trovano necessariamente ad essere limitati e ad usare termini simili. Prendendo come esempio il metodo "write" nella classe java.io.PrintStream, la cui dichiarazione è "void write(int OneByte)". La documentazione di Android per questo metodo è "writes one byte to the target stream" mentre quella di Oracle "write the specified byte to this stream". Le due documentazioni sono in gran parte simili, ma questo non implica che siano risultato di copia. In generale, le due documentazioni, Android e Java, non so-

no derivate. Ogni somiglianza è dettata dal fatto che documentano le stesse funzionalità.

Sorgente di Android e Java a confronto Per ultimo, Astrachan analizza i 12 file sorgenti, che secondo Oracle sono stati copiati da Google. Questi file rappresentano, per numero di file, il 0.02% di Android (12 su 57.076) e il 0.13% di Oracle (12 su 9.521) e percentuali inferiori quando misurati per linee di codice [21].

1. **Timsort:** I due file TimSort.java e ComparableTimSort.java sono una frazione minuscola di Android. Il file Arrays.java condivide undici metodi con TimSort.java e tredici con ComparableTimSort.java. Di questi solo uno è stato copiato, il metodo rangeCheck (Figura 3.12). Il metodo è lungo nove linee di codice e vale solo lo 0.01% del file TimSort.java (924 linee). La funzione rangeCheck è banale anche dal punto di vista qualitativo. Realizza un semplice controllo, assicurandosi che certi argomenti siano corretti prima di essere utilizzati da altre parti. La funzione rangeCheck è sia qualitativamente che quantitativamente ininfluente.

2. **File di test:** Gli otto file di test (3.2), non hanno alcun valore materiale e sono una porzione minuscola di Android. I file sono memorizzati in una *directory* chiamata "test" e la struttura del loro codice sorgente indica che fanno parte di un pacchetto per fare test. Non appartengono ad un pacchetto che fa parte dell'API pubblica di Android, quindi non hanno alcuna influenza sull'effettiva funzionalità della piattaforma Android. In aggiunta, sono stati rimossi da Android nel gennaio del 2011 e non sono stati rimpiazzati, questo testimonia la loro poca utilità. Infine, questi file rappresentano meno dello 0.1% del codice totale di Android e Java, anche questi file sono di poco peso sia per quantità che per qualità.

3. **CodeSourceTest:** I commenti scritti nei file CodeSourceTest.java e CollectionCertStoreParametersTest.java non sono stati scritti da Google e non aggiungono alcun valore ad Android. Un totale di 20 commenti sui 52 presenti nei due file risulta identico, nessuna porzione di sorgente è stata copiata. I commenti non hanno alcuna influenza sul prodotto software finale venduto al cliente, non aggiungono alcun valore materiale alla piattaforma Android. Inoltre, analizzando la storia di

questi file, emerge che i commenti non li abbiano scritti dipendenti di Google. I commenti sono stati scritti da programmatori della Intel che hanno poi donato i file al progetto *open source* Apache Harmony, dal quale poi Google ha preso i file in discussione. Infine, questi file rappresentano una porzione insignificante delle due piattaforme, meno dello 0.1% del totale.

3.1.3 Verdetto della Corte Distrettuale

Dopo aver ascoltato le diverse argomentazioni, il giudice William Alsup ha diviso il caso in tre fasi: violazione del copyright, violazione dei brevetti e danni. Il 16 aprile 2012 è iniziata la fase di copyright, il giudice decretava se le API fossero protette da copyright, la giuria se ci fosse violazione del copyright e se l'uso di Google rientrasse nel fair use.
Nel 2008 le API di Java erano composte da 166 pacchetti, contenenti più di 6000 metodi[2]. Google ha replicato il codice dichiarativo e di conseguenza le chiamate delle funzioni, per 37 di questi pacchetti[2], ma ha sviluppato da zero il codice implementativo, per ogni classe e metodo, senza alcun riferimento all'implementazione di Java. La scelta di copiare *verbatim* il codice delle dichiarazioni e la struttura dell'API è stata fatta per garantire agli sviluppatori continuità e familiarità con le librerie, tuttavia Oracle ritiene che sia stato fatto per motivi commerciali, proprio questo è il punto di contrasto tra le due parti.
Il 7 maggio 2012 la giuria ha decretato che Google avesse commesso violazione del copyright sul codice e sulla struttura dell'API, ma non hanno raggiunto un accordo se fosse *fair use*. Successivamente, è iniziata la fase sui brevetti, si è discusso in particolare di due brevetti RE38,104 e 6,061,520 che riguardavano rispettivamente metodi per risolvere riferimenti ai dati e per realizzare inizializzazione statica[24]. Per la fase brevettuale, la giuria si è dichiarata in favore di Google.
Il verdetto finale è stato emanato dal giudice il 31 maggio 2012. In primo luogo ha dichiarato che le API non fossero protette da copyright, annullando così l'infrazione del copyright di Google tranne per la funzione e gli otto file di test[11], per i quali ha posto

[11]La funzione RangeCheck è stata sviluppata dall'ingegnere Joshua Block che dal 1996 al 2004 ha lavorato per Sun, nel 2004 si è trasferito da Sun a Google. Nel 2009 Bloch ha lavorato ad Android, in quest'occasione ha incluso la funzione RangeCheck nel codice di Android, perciò l'errore è stato

un risarcimento di 150.000 dollari[2]. Le motivazioni del giudice sono state le seguenti:

- Confrontando i 37 pacchetti di Java e Android solo il 3% del codice risulta uguale, il codice dichiarativo dei metodi e delle classi[2].
- Per lo United States Copyright Office i nomi, i titoli e le frasi corte[12] non sono protette dal copyright, questo si riflette anche sui nomi di metodi, classi e parametri in un programma di un calcolatore.
- Per la dottrina del raggruppamento (dottrina *merger*), quando c'è uno solo o pochi modi per esprimere un'espressione nessuno può avere il copyright su quell'espressione, il codice dichiarativo e le chiamate di funzione rientrano in questo caso.
- Per la sezione 102(b) del Copyright Act il copyright non si estende a idee, procedure e processi.
- Il giudice ritiene che finché il codice implementativo è differente, chiunque è libero di implementare il suo metodo che realizza le stesse funzionalità di un metodo preesistente, non importa quanto questo metodo sia creativo e complicato.
- Oracle aveva ragione ad affermare che Google avrebbe dovuto organizzare la struttura dei 37 pacchetti in modo differente, poiché questa struttura è una tassonomia e in quanto tale è soggetta a copyright. Tuttavia oltre ad essere una tassonomia è anche un metodo operativo, questa seconda natura prevale e la rende priva di copyright.

considerato dal giudice involontario. Google ha anche copiato otto file per il *testing* che però sono stati usati raramente e non hanno mai fatto parte del codice Android, per questo non vengono reputati irrilevanti

[12]Categorie di frasi corte che rientrano in questa categoria sono: nomi di prodotti e servizi (anche funzioni, variabili e parametri), slogan, titoli e nomi di aziende. Non è presente una misura esatta oltre la quale una frase abbia diritto al copyright.

In seguito a questo decisione Oracle ha fatto appello chiedendo un *judgment as a matter of law*[13]. Il caso includeva anche brevetti, perciò l'appello è stato assegnato alla United States Court of Appeals for the Federal Circuit.

3.2 Analisi della copia da parte della Zeidman Consulting

In seguito alla decisione della Corte Distrettuale, l'azienda Zeidman Consulting si è chiesta se nel raggiungere il verdetto i dati considerati fossero veritieri. Per questo motivo hanno deciso di analizzare il codice di entrambe le piattaforme utilizzando gli strumenti software forniti da una compagnia vicina a loro, la Software Analysis and Forensic Engineering (SAFE Corporation). Analizziamo il loro report perché mostra un *modus operandi* per discernere le questioni di copia e plagio di codice sorgente a livello giuridico.

Hanno iniziato il loro esperimento confrontando gli stessi codici discussi nel precedente processo. Da un lato JDK5, contenente 12,262 file e dall'altro Android *Froyo*, risultante di 17,062 file[25]. L'enorme mole di file comporterebbe un totale di 209,214,244 confronti tra coppie di file, un lavoro troppo lungo per essere effettuato manualmente. Per questa ragione, per l'analisi è stato utilizzato il software CodeSuite, sviluppato dalla SAFE Corporation. Uno dei *tool* all'interno di CodeSuite è CodeMatch, programmato per confrontare migliaia di file sorgenti in diverse cartelle e determinare quali file sono i più simili. il funzionamento di CodeMatch è spiegato dettagliatamente nella sezione successiva.
I risultati ottenuti da CodeMatch sono stati filtrati eliminando algoritmi comuni, codice automaticamente generato e nomi di identificatori comuni. Dopo il filtraggio sono rimaste nove coppie di file che esibiscono prove significative di copia letterale. Le coppie elencate qui sotto (Tabella 3.4) non erano state considerate nell'analisi sulla copia. Nell'ultima Tabella sono mostrati i valori di similarità assegnati da CodeMatch. Il valore più alto è un 96 per la coppia di file AbstractExecutorService.java, il valore più basso considerato è

[13]Una mozione per *judgment as a matter of law* è una mozione mossa, durante un processo, da una parte sostenendo che la parte opposta non abbia prove sufficienti per supportare le sue argomentazioni.

il valore 64 della coppia SimpleNode.java. Nel report di CodeMatch sono presenti diversi esempi di porzioni di codice simili.

File Java	**File Android**	Valore CodeMatch
ParseException.java	ParseException.java	82
TokenMgrError.java	TokenMgrError.java	85
SimpleNode.java	SimpleNode.java	64
JJTParserState.java	JJTAddressListParserState.java	89
ArrayBlockingQueue.java	ArrayBlockingQueue.java	79
AbstractExecutorService.java	AbstractExecutorService.java	96
ConcurrentHashMap.java	ConcurrentHashMap.java	72
LinkedBlockingQueue.java	LinkedBlockingQueue.java	80
PriorityBlockingQueue.java	PriorityBlockingQueue.java	71

Tabella 3.4: Risultati di CodeMatch, coppie considerate copia evidente.

3.2.1 Funzionamento CodeMatch

CodeMatch usa una combinazione di cinque algoritmi per trovare il plagio: Source Line Matching, Comment Line Matching, Word Matching, Partial Word Matching and Semantic Sequence Matching[26].
Prima di applicare gli algoritmi il codice viene preprocessato per creare array, strutture dati ispirate alla nozione matematica di vettore, di stringhe. Ogni file è rappresentato da tre array:

- Un array composto da linee di codice funzionale, escludendo così i commenti;
- Un array formato da commenti, tralasciando il codice funzionale;
- Un array degli identificatori trovati nel codice, per identificatori consideriamo nomi di variabili, costanti e funzioni e tutte le altre parole che non sono *keyword* del linguaggio di programmazione. Per *keyword* intendiamo le parole strettamente legate al linguaggio di programmazione, alcuni esempi sono "integer", "public" e "string".

Per ogni coppia di file, l'algoritmo Word Matching conta il numero di parole uguali, escludendo le *keyword*. Per determinare se una parola è una *keyword* o meno, viene fatto un confronto con una lista contenente tutte le *keyword* del linguaggio di programmazione. Il confronto può essere *case-sensitive* o *case-insensitive* a seconda del linguaggio di programmazione.
Partial Word Matching esamina ogni *non-keyword* nei file sorgenti e trova tutte le sottostringhe che fanno match nell'altro file, anche questo passaggio è *case-insensitive*. Nell'esempio in Figura 3.19, notiamo che la sequenza "pdq" non viene rilevata in questa fase, perché è il match di due stringhe complete e quindi viene già considerato nell'algoritmo Word Matching.

Bob examines the tools and algorithms for uncovering plagiarism in source code.

(a)

```
Word1[0] = "abc"     Word2[0] = "Aabc"
Word1[1] = "abc1"    Word2[1] = "aBc"
Word1[2] = "abc123"  Word2[2] = "abc1111111"
Word1[3] = "abcdef"  Word2[3] = "abcXXXyz"
Word1[4] = "pdq"     Word2[4] = "i"
Word1[5] = "xxx"     Word2[5] = "j"
Word1[6] = "xyz"     Word2[6] = "pdq"
Word1[7] = "yyy"     Word2[7] = "X"
```

(b)

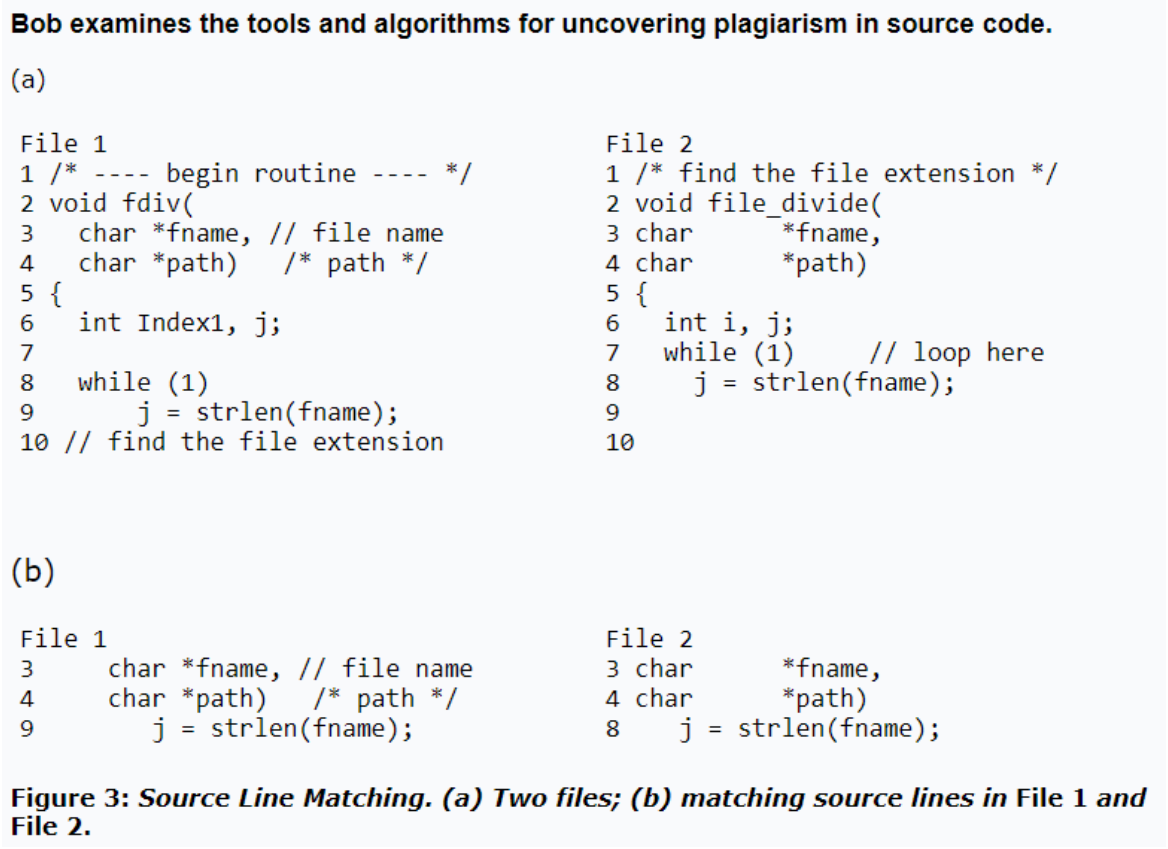
```
PartialWord[0] = "abc"
PartialWord[1] = "abc1"
PartialWord[2] = "xxx"
PartialWord[3] = "xyz"
```

Figure 2: ***Partial word matching. (a) Nonkeyword words in files 1 and 2; (b) matching partial words.***

Figura 3.19: Esempio computazione dell'algoritmo Partial Word Matching[26].

L'algoritmo Source Line Matching confronta ogni linea di codice di entrambi i file, ignorando la sensibilità alle maiuscole. In questa fase confrontato solo il codice funzionale, i commenti sono esclusi. Le sequenze di spazi bianchi sono convertite ad un singolo spazio bianco e le linee che contengono solo *keyword* non sono considera-

te. Una linea di codice per essere considerata match deve contenere almeno una *non-keyword*. Nell'esempio sottostante (Figura 3.20) vediamo un esempio del funzionamento dell'algoritmo.



Bob examines the tools and algorithms for uncovering plagiarism in source code.

(a)

File 1

```
1 /* ---- begin routine ---- */
2 void fdiv(
3   char *fname, // file name
4   char *path)   /* path */
5 {
6   int Index1, j;
7
8   while (1)
9       j = strlen(fname);
10 // find the file extension
```

File 2

```
1 /* find the file extension */
2 void file_divide(
3 char      *fname,
4 char      *path)
5 {
6   int i, j;
7   while (1)     // loop here
8     j = strlen(fname);
9
10
```

(b)

File 1

```
3     char *fname, // file name
4     char *path)   /* path */
9        j = strlen(fname);
```

File 2

```
3 char      *fname,
4 char      *path)
8    j = strlen(fname);
```

Figure 3: *Source Line Matching. (a) Two files; (b) matching source lines in* File 1 *and* File 2.

Figura 3.20: Esempio computazione dell'algoritmo Source Line Matching[26].

L'algoritmo Comment Line Matching confronta ogni linea di commento di entrambi i file (*case-insensitive*). Come prima, le sequenze di spazi bianchi sono convertite ad un singolo spazio.

Bob examines the tools and algorithms for uncovering plagiarism in source code.

(a)

```
File 1                                  File 2
1 /* begin routine */                   1 /* find the file extension */
2 void fdiv(                            2 void file_divide(
3    char *fname, // file name          3 char      *fname,
4    char *path)   /* path */           4 char      *path)    // path
5 {                                     5 {
6     int Index1, j;                    6   int i, j; /* begin routine */
7                                       7   while (1)     // loop here
8     while (1)                         8     j = strlen(fname);
9        j = strlen(fname);             9
10    // find the file extension        10   switch (x)
11    if (x == 5) {                     11   {
```

(b)

```
File 1                                  File 2
1 /* begin routine */                   6    int i, j; /* begin routine */
4     char *path)   /* path */          4  char      *path)    // path
10    // find the file extension        1  /* find the file extension */
```

Figure 4: ***Comment Line Matching. (a) Two files; (b) matching comment lines in*** **File 1** ***and*** **File 2.**

Figura 3.21: Esempio computazione dell'algoritmo Comment Line Matching[26].

L'algoritmo Semantic Sequence Matching confronta la prima parola di ogni linea di codice, dei due sorgenti, ignorando linee vuote e linee di commento. L'algoritmo trova sequenze di codice che sembrano eseguire le stesse funzioni, nonostante cambiamenti di identificatori o commenti. L'algoritmo trova la sequenza comune più lunga tra i due file. Nella Figura 3.22 vediamo che le due porzioni di codice fanno match perché le sequenze delle prime parole sono identiche (if, for, printf, else, while) chiaramente non considerando linee vuote, linee di commento o punteggiatura.

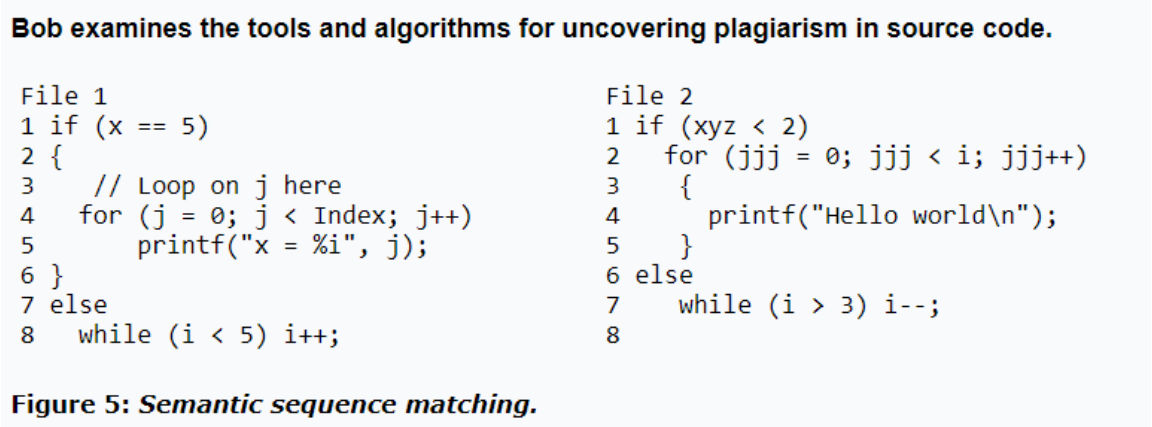


Figura 3.22: Esempio computazione dell'algoritmo Semantic Sequence Matching[26].

Infine, ogni algoritmo assegna un punteggio di similarità per la coppia di file, più il punteggio è alto più i due file sono simili. La scala è da 0 a 100. Per calcolare il punteggio finale di similarità della coppia, ogni punteggio è pesato con valori diversi e poi questi sono sommati. I pesi usati per l'analisi di similarità tra Google e Oracle sono stati chiesti al presidente della SAFE Corporation Bob Zeidman, ma purtroppo non ci è stata data alcuna risposta in merito. I risultati di CodeMatch sono mostrati in un report HTML, nel quale sono contenute tutte le analisi per ogni coppia di file.

3.3 Verdetto della United States Court of Appeals for the Federal Circuit

Il 4 dicembre 2013 è iniziato il processo presso la Corte Federale. Oracle ha fatto appello in quanto ritiene non corretto il verdetto della giuria precedente in quattro occasioni:

1. Ha concluso che il codice dichiarativo non può essere protetto da copyright poiché unisce idea ed espressione.

2. Ha concluso che il codice dichiarativo non può essere protetto da copyright poiché composto da frasi corte e nomi.

3. Ha concluso che la struttura e l'organizzazione dell'API non è protetta da copyright poiché è un metodo operativo.

4. Ha considerato le argomentazioni di Google sull'interoperabilità in un analisi sul copyright.

La giuria ha analizzato questi punti e su ognuno ha espresso il suo parere.

3.3.1 Il codice dichiarativo

La dottrina del raggruppamento è un'eccezione alla dicotomia idea/espressione, stabilisce che quando c'è un numero limitato di modi per esprimere un'idea, l'idea si unisce con l'espressione, rendendo l'ultima così non protetta. La giuria reputa che il codice dichiarativo potrebbe essere stato espresso da Google in una vasta gamma di modalità. Ad esempio, la funzione "java.lang.Math.max" potrebbe chiamarsi "Arith.larger" o "Math.maximum", quindi non c'era un numero limitato di vie e di conseguenza la dottrina del raggruppamento non si può applicare, Google avrebbe dovuto implementare diversamente le 11500 linee copiate, usando nomi differenti da Oracle per funzioni variabili e parametri[27].
Inoltre, la Corte Distrettuale aveva ragione ad affermare che i nomi, le frasi corte e i titoli non sono protette da copyright, tuttavia la stessa corte non ha considerato la creatività dietro alle frasi breve e ai nomi. La Corte Federale riconosce che una combinazione originale di elementi può essere protetta da copyright e il codice dichiarativo rientra in questa definizione. Quindi, poiché Oracle ha esercitato creatività nella scelta e nell'arrangiamento delle dichiarazioni, queste sono protette da copyright.

3.3.2 La struttura e l'organizzazione

La Corte Distrettuale afferma che la struttura e l'organizzazione dell'API siano creative e originali, ma in quanto metodo operativo non possono essere soggette a copyright, poiché prevale la loro parte funzionale. Per la Corte Federale tutti i programmi per elaboratori sono per definizione funzionali, poiché il loro scopo è compiere operazioni, se dovessimo reputare l'API di Oracle non protetta da copyright poiché funzionale, allora

non lo sarebbe nessun programma per elaboratore. Invece, un lavoro originale, anche se funzionale, ha diritto a copyright finché l'autore ha diverse vie per esprimere l'idea di base, la Corte Federale reputa la struttura e l'organizzazione delle API creativa. In aggiunta, anche in questo caso Google avrebbe potuto strutturare i 37 pacchetti in modo diverso, senza perdere di funzionalità. Date queste premesse la Corte ha deciso che in questo caso la sezione 102(b) del Copyright Act non sussiste, è falso che i 37 pacchetti non sono soggetti a copyright solo perché hanno una componente funzionale.

3.3.3 Interoperabilità

Oracle ritiene che la Corte Distrettuale abbia sbagliato nel considerare le argomentazioni di interoperabilità di Google, in quanto non condizionano in alcun modo l'analisi sul copyright, sarebbero rilevanti solo in un'analisi del *fair use*. In particolare, la Corte Distrettuale aveva reputato come necessaria la duplicazione della struttura delle API per fornire interoperabilità. Per la Corte Federale, Google avrebbe scelto di non modificare la struttura per capitalizzare sulla già presente conoscenza, dell'API di Java, tra gli sviluppatori, voleva facilitare il suo sviluppo attraendo i programmatori in Java[27]. Per queste ragioni l'argomentazione di interoperabilità non ha nessun valore nell'ambito del copyright.

Nell'ottobre 2014 la Corte Federale, dopo queste analisi, ha decretato che l'API di Java fosse protetta da copyright, rimandando il caso alla Corte Distrettuale per decidere se l'uso fatto da Google rientrasse nel *fair use*. Infine, la Giuria ha confermato la precedente infrazione del copyright sulle nove porzioni di codice, la funzione e gli otto file di sicurezza[27].
A seguito della sentenza Google ha fatto petizione alla Corte Suprema statunitense, ma la sua richiesta non è stata accettata.

Capitolo 4

Seconda fase: *fair use*

4.1 Secondo processo Corte Distrettuale

Il 9 maggio 2016 il caso è tornato alla Corte Distrettuale per decretare se l'uso di Google fosse *fair*, come ordinato dalla Corte Federale.
Il *fair use* è una dottrina della legge statunitense che permette l'uso limitato di materiale protetto da copyright, senza aver bisogno di acquisire il permesso dal detentore del copyright. Il *fair use* si valuta su quattro fattori:

1. L'oggetto e la natura dell'uso, in particolare se ha natura commerciale oppure didattica e senza scopo lucrativo;

2. La natura dell'opera protetta;

3. La quantità e l'importanza della parte utilizzata, in rapporto all'insieme dell'opera protetta;

4. Le conseguenze di questo uso sul mercato potenziale o sul valore dell'opera protetta.

Oracle riteneva l'uso di Google prevalentemente commerciale, che la copia fosse sostanziosa e avesse causato enormi danni al mercato mobile di Oracle, in virtù di questi elementi cercava un risarcimento di 8.8 miliardi di dollari[3].

4.1.1 Argomentazioni di Oracle

L'attacco di Oracle ha fatto particolare leva su tre punti:

1. Android è un prodotto commerciale, quindi l'uso di Google è commerciale;
2. L'uso delle API da parte di Google non è trasformativo;
3. il codice copiato è quantitativamente e qualitativamente importante, sia per Android che per Java.

Gli esperti per Oracle in questa fase del processo sono stati Adam Jaffe[1], professore di economia all'università di Brandeis, e Chris Kemerer[2], professore di informatica all'università di Pittsburgh. Jaffe ha analizzato il primo fattore del *fair use*, Kemerer il terzo.

Analisi del primo fattore

Google ha usato Java in Android per necessità di entrare sul mercato immediatamente. Durante il primo decennio di questo secolo, gli utenti si stavano spostando dai PC verso i telefoni, passando più tempo sulle applicazioni che sui web browsers. Questa tendenza minacciava il business principale di Google, la pubblicità web. In aggiunta, Apple, nel 2007, aveva lanciato il primo iPhone, sviluppando una piattaforma mobile solida, Google per competere doveva entrare nell'ecosistema mobile in fretta. Per questo motivo, ha individuato nell'API Java una via per lanciare Android brevemente e sfruttare l'ampia adozione, già ottenuta da Java, nell'industria mobile.
L'uso dei telefoni è largamente dovuto alle applicazioni, le quali sono *platform-dipendent* (un applicazione per iPhone non può essere eseguita su un telefono Android, a meno di sviluppo separato per le due versioni)[28]. Nello sviluppo di Android, un obiettivo chiave era quello di attrarre una larga base di sviluppatori per creare applicazioni sulla

[1]La pagina web del professor Kemerer è all'indirizzo: https://www.brandeis.edu/facultyguide/person.html?emplid=f6a0a167667c13f87d2454cfc399983a0fadbee0

[2]La pagina web del professor Kemerer è all'indirizzo: https://sites.pitt.edu/~ckemerer/kemerer.htm

piattaforma. La comunità Java era particolarmente robusta, contava 6 milioni di programmatori al lancio di Android. Copiando le API di Java, Google era in grado di accedere ad un numeroso gruppo di informatici, già a conoscenza della piattaforma Java, pronti facilmente a migrare su Android per lo sviluppo di applicazioni mobili. Grazie alla copia dei 37 pacchetti dell'API Java, Google è entrato nel mercato al momento giusto, ottenendo un enorme successo commerciale, testimoniato dai cospicui profitti ottenuti da Google, tramite Android. Google guadagna denaro dalla piattaforma Android in quattro modi: pubblicità sui dispositivi Android, vendita di applicazioni, vendita di contenuti digitali e vendita di hardware. L'ascesa di Android nel mercato mobile è stata continua dal momento del suo lancio, arrivando nel 2013 ad avere l'83% del mercato mobile. Android è un prodotto cruciale e altamente profittevole per Google, tuttavia, lo è diventato poiché in primo luogo ha copiato le API Java. Per queste ragioni l'uso di Google è da considerarsi fortemente commerciale.

Il professore Jaffe continua la sua testimonianza mostrando come l'uso dell'API Java non sia stato trasformativo, da parte di Google. La sostituzione è un concetto base dell'economia. Due beni o servizi sono sostituti se possedere più unità di uno, riduce il valore marginale dell'altro bene. Ad esempio, se ho diverse matite nel mio astuccio, il valore che ha per me una penna è minore, poiché ho già uno strumento con cui scrivere. Anche se non uguali, penne e matite danno entrambe la possibilità di scrivere e disegnare, perciò sono sostituti che competono per il mio consumo. Questo concetto è conosciuto come sostituzione marginale, poiché è frequente decidere di possedere sia penne che matite. In altri casi, per prodotti particolari, come i telefoni, è raro possedere due unità dello stesso prodotto. Pertanto, se possiedo un modello (iPhone) è difficile che scelga di tenere contemporaneamnte un altro telefono (Android). Questo è un esempio di sostituzione non marginale. Google ha copiato e incorporato i 37 pacchetti in Android. Quest'uso ha sostituito Java nei prodotti dove Oracle aveva licenziato o avrebbe potuto licenziare Java. Un dispositivo solitamente contiene solo una piattaforma, in questo caso o Java o Android, per questo, l'uso di Google crea sostituzione non marginale. Un uso è da considerarsi non trasformativo in casi di sostituzione, come questo tra Oracle e Google.

Analisi del terzo fattore

Il professor Kemerer mostra l'importanza del codice copiato nei 37 pacchetti attraverso tre test differenti che misurano stabilità, valore e centralità.

Stabilità Un vantaggio che Google ha ottenuto dalla copia del codice e della struttura dei 37 pacchetti è la stabilità. Le API Java, al tempo di sviluppo di Android, oltre ad essere già scritte erano sul mercato da diversi anni, questo vuol dire che erano state controllate e aggiornate svariate volte. Garantivano a Google una base più solida rispetto ad API scritte da zero, più prone ad errori e a cambiamenti.
Per verificare la veridicità di quest'affermazione, l'esperto di Oracle ha condotto un'analisi sulla frequenza dei cambiamenti nelle due piattaforme. Il primo passo è stato scaricare ogni versione del JDK tra la versione 1.0.2 e la 1.8.0. Successivamente, è stato usato il *tool* Oracle Javadoc per convertire l'organizzazione dei file sorgenti in un file HTML contenente la documentazione dell'API. Il file HTML è stato scansionato tramite uno script PHP e i risultati ottenuti sono stati verificati manualmente confrontandoli con la documentazione web dell'API. Dopo questi passi di verifica, la documentazione dell'API Java è stata unita in un unico file tabellare (CSV) contenente tutte le versioni. Per Android, sono state scaricate tutte le documentazione dall'API Level 1 fino a Level 23. Dal livello 1 al 13 la documentazione era disponibile in formato XML, per questo motivo i file sono stati convertiti in file CSV usando il *tool* Moor XML to CSV. Per tutte le versioni della documentazione Android è stato effettuato un test di controllo duale a quello Java. Passato il test tutte le versioni sono state raggruppate in un file tabellare.
Per entrambe le piattaforme, il numero di metodi cambiati in ogni pacchetto tra due versioni seguenti è stato calcolato sommando le differenze negli step precedenti. Ad esempio, il numero di cambiamenti calcolati per il pacchetto "java.util" indica il numero dei cambiamenti occorsi tra l'Android API Level 1 e 2, 2 e 3, e cumulativamente fino ai livelli 22 e 23. Il numero dei cambiamenti è stato valutato sulla lista dei metodi presenti in API seguenti, quindi metodi aggiunti o rimossi non sono stati considerati. Per ognuno dei metodi esistenti in API continue, qualsiasi cambiamento (tipo, visibilità, ereditarietà...) nella struttura del metodo è stato contato nell'analisi. Uno script R è stato usato per calcolare i cambiamenti dei metodi. Per l'analisi effettiva sono stati create tre *repository*

con composizioni differenti.

- **Libcore:** Contiene 51 pacchetti API sviluppati da Oracle (tra cui i 37 accusati), 5 pacchetti API sviluppati da Google e 8 da terze parti.
- **Framework:** Contiene 100 pacchetti API sviluppati da Google e solamente 2 da Oracle.
- **External:** Contiene 32 pacchetti sviluppati da terze parti e 2 da Oracle.

I pacchetti Android sono stati divisi in base alle loro funzionalità, quelli di Oracle sulla base di riferimenti ai 37 pacchetti. In Figura 4.1 c'è un riassunto visivo di quanto detto.

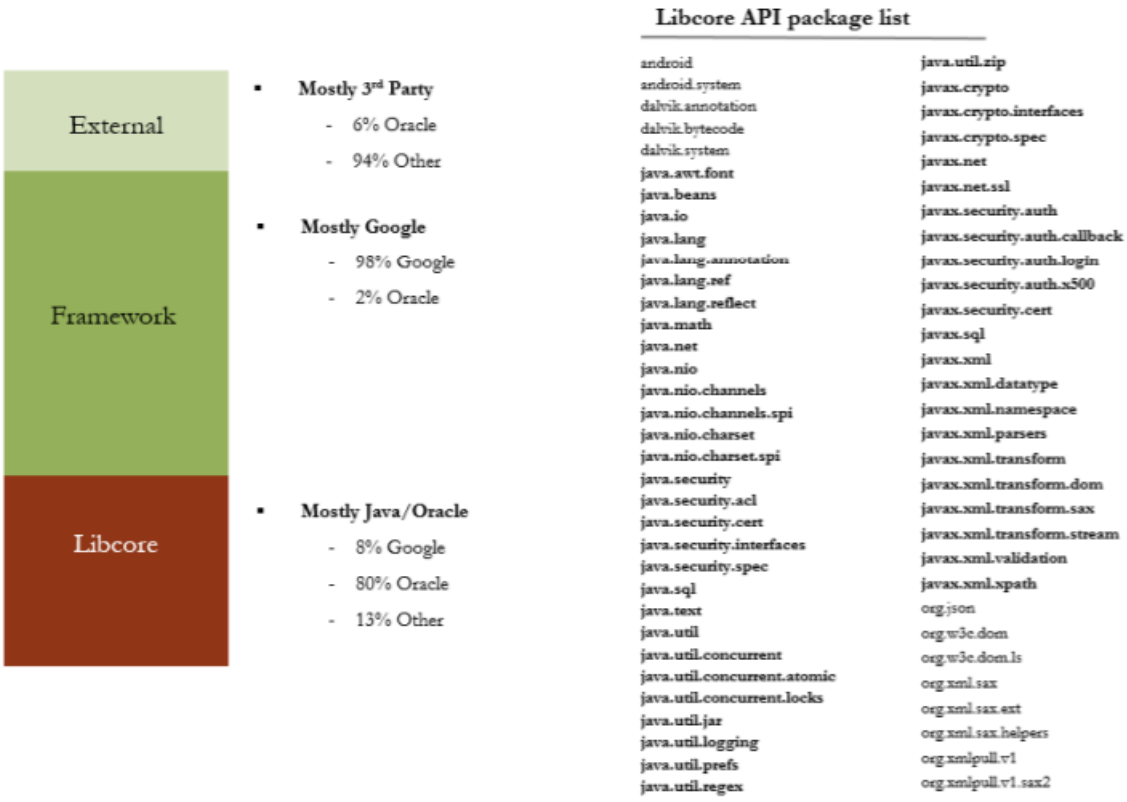


Figura 4.1: Composizione delle tre *repository* per l'esperimento di stabilità[29].

Per entrambe le piattaforme, il numero di cambiamenti di metodo tra due versioni seguenti è stato calcolato, il numero così ottenuto è stato poi sommato al totale della

categoria (i.e. Android API Level 1). Per ogni categoria è stata poi ricavata una media, ottenendo così il numero medio di cambiamenti per pacchetto di ogni categoria. In Figura 4.2 sono presenti i risultati. *Libcore*, composta in larga parte da API Java, si è stabilizzata molto prima e ha avuto molti meno cambiamenti rispetto a *Framework*, composta in larga parte da API Google. *Libcore* si è stabilizzata dopo il rilascio di Android API Level 9, avvenuto nel dicembre del 2010.

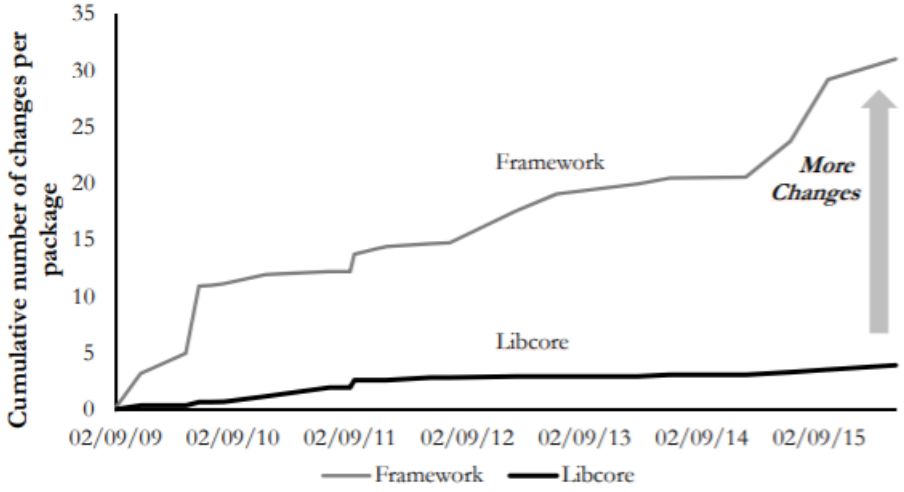


Figura 4.2: Risultati dell'analisi di stabilità per le *repository* Framework e *Libcore*[29].

La rapida stabilizzazione di *Libcore* è dovuta alla presenza delle API Java. Questo risultato è evidenziato da una successiva analisi che confronta i cambiamenti di tre "sottorepository" di *Libcore*. Il primo sottoinsieme (*Libcore* Oracle) è formato da tutti e 51 i pacchetti Java, il secondo sottoinsieme (*Libcore* Oracle *Infringed*) solo dai 37 pacchetti copiati e l'ultimo insieme (Libcore Google) dai 5 di Google. Come evidente dai risultati in Figura 4.3, le API Java si stabilizzano dopo il rilascio di Android API Level 9, nel dicembre 2010. Al contrario le API di Google hanno un numero elevato di cambiamenti e fanno fatica a stabilizzarsi. Questi fattori suggeriscono che la stabilità di Android è data dai 37 pacchetti Java copiati.

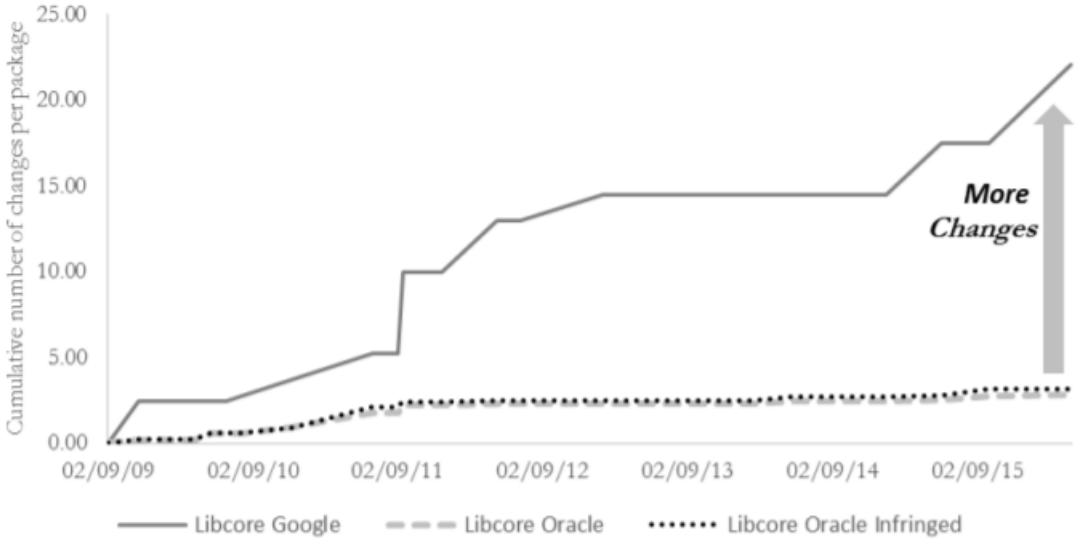


Figura 4.3: Risultati dell'analisi di stabilità per i tre sottoinsiemi di *Libcore*[29].

La *repository Framework*, composta per il 98% da API Google, ha impiegato molto più tempo per stabilizzarsi. Dalla Figura 4.4 vediamo che i 2 pacchetti Java in *Framework* non hanno mai subito cambiamenti, pertanto il 100% dei cambiamenti della *repository* è da attribuirsi alle API di Google.

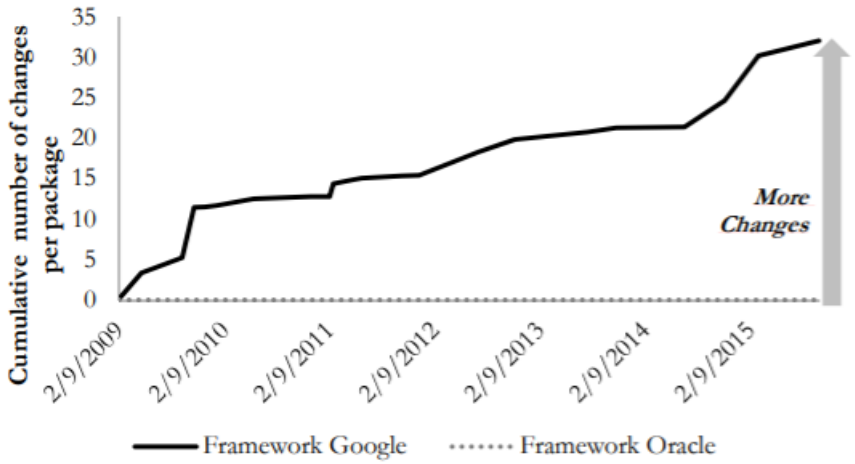


Figura 4.4: Risultati dell'analisi di stabilità per la *repository Framework*[29].

L'enorme vantaggio tratto dalla copia di Android emerge quando analizziamo il tem-

po impiegato dalle due piattaforme per stabilizzarsi. Infatti, i 37 pacchetti Java hanno impiegato dieci anni per stabilizzarsi più o meno definitivamente, nel dicembre del 2006. Sempre gli stessi 37 pacchetti, se analizzati per Android hanno impiegato solo due anni a stabilizzarsi, senza la copia di questi pacchetti è presumibile che le API Android avrebbero impiegato molto più tempo a stabilizzarsi, seguendo un trend simile a quelle di Java.

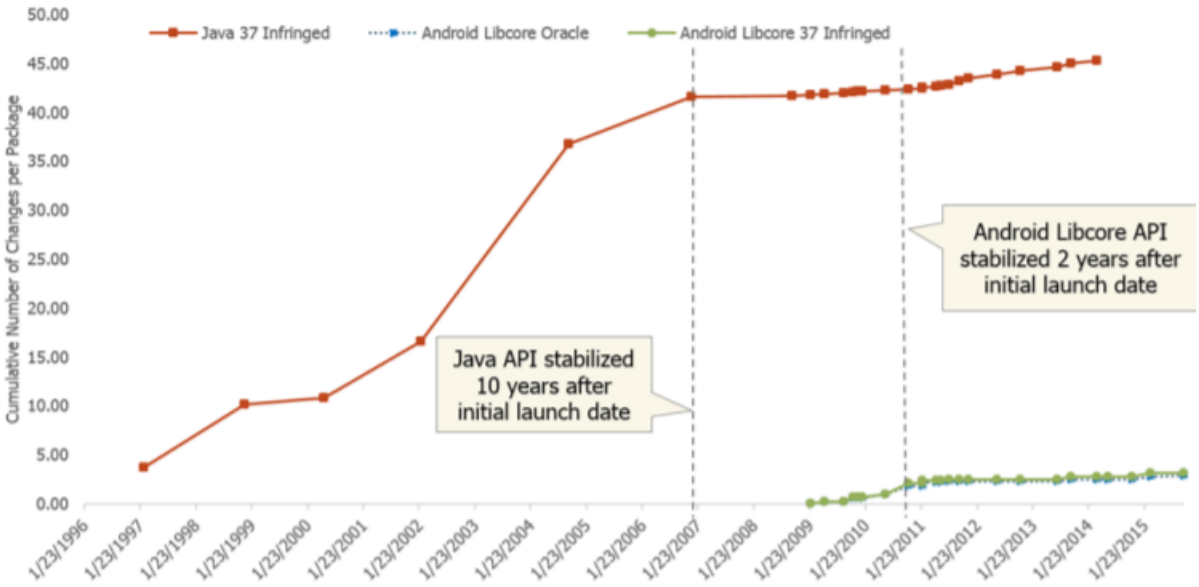


Figura 4.5: Confronto dell'analisi di stabilità, per i 37 pacchetti indagati, tra Java e Android[29].

La stabilità dei 37 pacchetti copiati ha contribuito alla significativa crescita di Android. Kemerer ha osservato la crescita in tre modi differenti: (1) crescita nel numero di attivazioni di dispositivi Android, (2) crescita nel numero di sviluppatori attivi e (3) crescita nel numero di applicazioni attive. Come si vede dalla Figura 4.6, è presente una considerabile crescita, nel numero di attivazioni di dispositivi, coincisa con la stabilizzazione delle API di Android nel gennaio del 2011.

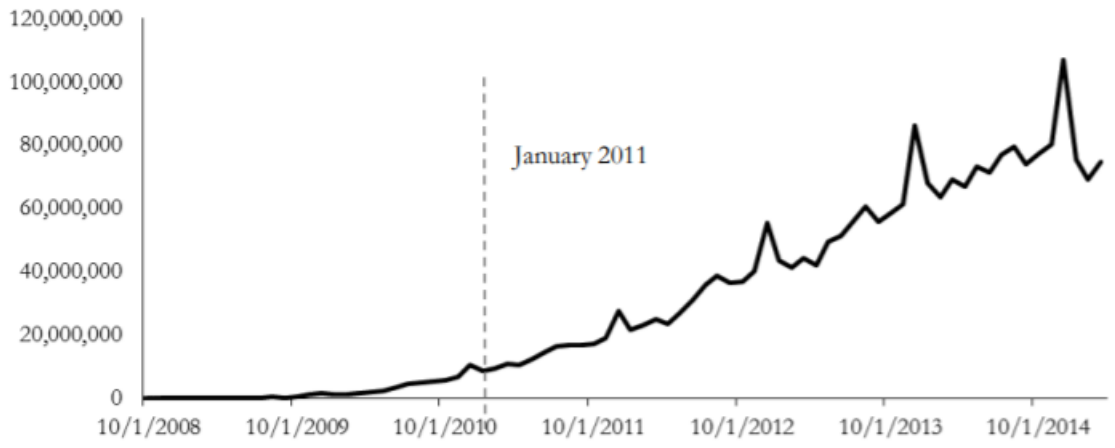


Figura 4.6: Crescita del numero di attivazione di dispositivi Android[30].

Anche per il numero di sviluppatori attivi c'è un notevole incremento da gennaio 2011 in poi. Vedasi Figura 4.7.

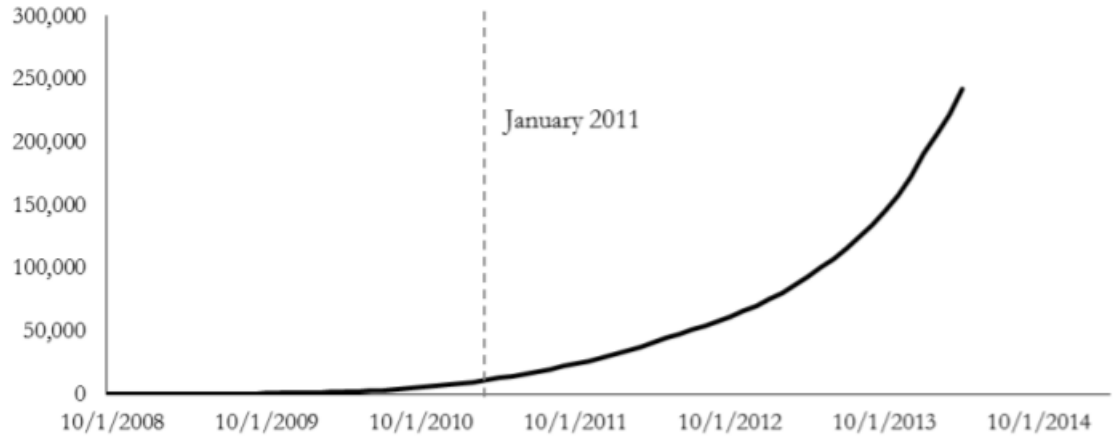


Figura 4.7: Crescita del numero di sviluppatori attivi[30].

L'impatto della stabilità delle API Java si misura anche nel numero di applicazioni Android disponibili. Come mostrato dal grafico in Figura 4.8 notiamo ancora una crescita a partire dal primo mese del 2011.

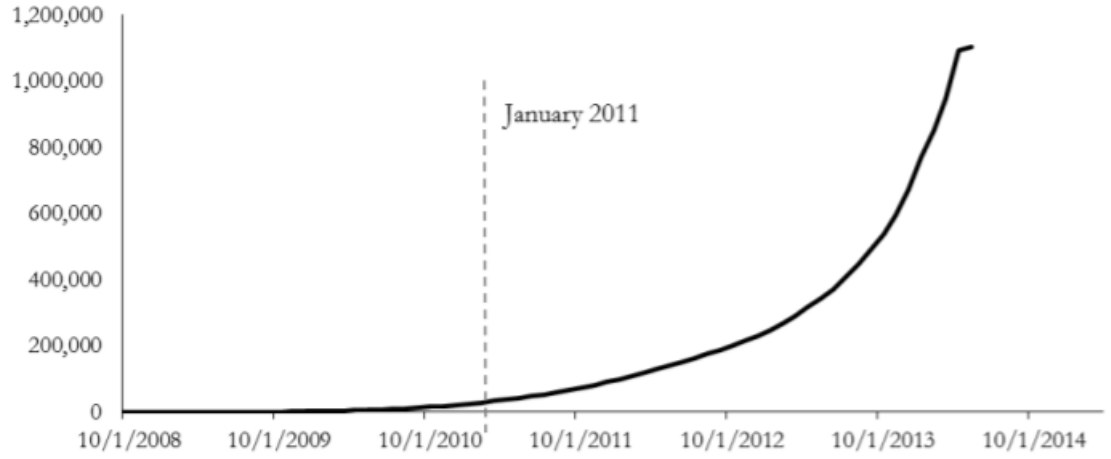


Figura 4.8: Crescita del numero di applicazioni Android attive[30].

La stabilità che i 37 pacchetti incriminati hanno impartito alla piattaforma Android ha permesso lo sviluppo e l'adozione di applicazioni sulla piattaforma. Questo ha alimentato l'aumento della comunità di sviluppatori Android, che hanno contributo allo sviluppo della piattaforma a loro volta, in un pattern ciclico. Uno studio indipendente suggerisce che il successo delle applicazioni Android è dato dalla stabilità e qualità delle API usate per creare le applicazioni[32]. Le applicazioni Android di bassa qualità tendono ad usare API Android che hanno un'alta frequenza di cambiamenti e sono più inclini ad errori. Per gli sviluppatori è complicato tenere il proprio lavoro al passo con API che variano continuamente, questo porta ad errori e difetti nel codice che rendono le applicazioni peggiori.

Valore Come descritto nel paragrafo precedente, le piattaforme software, Android incluso, devono la loro popolarità alle applicazioni disponibili sulla piattaforma. Per questa ragione, il professor Kemerer ha condotto un'analisi sul valore e l'importanza dei 37 pacchetti Java nell'ecosistema applicativo Android.
L'esame è stata realizzato in quattro step: (a) Scelta del campione per l'esperimento, (b) *Download* delle applicazioni selezionate, (c) Decompilazione e (d) Analisi dei risultati. Il primo step consisteva nello scegliere quali applicazioni Android usare per l'esperimento. Sono state scelte le 128 applicazioni più popolari sul Google Play Store, nel novembre del

2015. Più *download* ha un'applicazione, più è popolare. Alcuni esempi delle applicazioni selezionate sono: Facebook, Gmail, Google Maps e Google. Successivamente, il 21 dicembre 2015, ogni APK[3] (Android Application Package) è stato scaricato dal Google Play Store, usando il *plugin* Firefox APK downloader. Gli APK sono stati poi decompilati e dal codice sorgente ottenuto sono state estratte le dipendenze[4] di ogni applicazione. Il contenuto di ogni APK è stato scaricato e i file Android binari .dex contenuti nell'APK sono stati convertiti in file Java .class usando il *tool* dex2jar. I file .class sono poi stati decompilati in codice sorgente Java, rivelando le dipendenze dell'applicazione. L'ultimo passo ha previsto l'uso del *tool* Understand, un analizzatore di codice che a partire dal codice sorgente di una applicazione estrae la struttura delle sue dipendenze. Infine, l'output di Understand è stato processato per quantificare il numero di riferimenti ai 37 pacchetti API. A causa di problemi legati a restrizioni sulla regione di installazione e errori del software Understand, 28 applicazioni sono state scartate dall'analisi, lasciando così un totale di 100 applicazioni per l'analisi.
Guardando ai risultati dell'esperimento, può essere dimostrato che i 37 pacchetti Java sono un requisito fondamentale per quasi tutte e 100 le applicazioni. Infatti, ogni applicazione dell'esperimento (100%) dipende da un minimo di 3 dei pacchetti API discussi. Il numero medio di dipendenze è 11.5, quasi un terzo dei 37 pacchetti. Il risultato massimo tra le 100 applicazioni è stato di 23. In Figura 4.9 sono mostrate le dipendenze per le applicazioni direttamente prodotte da Google.

[3]L'estensione 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.

[4]Il sistema di dipendenze in Android permette di includere librerie esterne nell'applicazione che stai costruendo, queste librerie vengono dette dipendenze. Le dipendenze possono essere memorizzate localmente sul dispositivo di sviluppo o anche su *repository* remote.

Google App	Dependencies
Google Play Books	16
Google Drive	12
Google Play Newsstand	17
Maps	14
Gmail	14
Google Search	15
Google Talkback	8
Google Play Music	17
Google Play Games	12
Google Text-to-speech	15
Google Play Movies & TV	12
Google Streetview	9
Cloud Print	12
Google News & Weather	12
Google Translate	14
Google Calendar	15
Google Keyboard	9
Google Earth	12
Additional Apps	
YouTube	11
Hangouts	12
Chrome Browser - Google	17

Figura 4.9: Numero di dipendenze per le applicazioni direttamente prodotte da Google[29].

Da quest'esame concludiamo che i 37 pacchetti copiati sono significativi e centrali per l'ecosistema applicativo Android e gli sviluppatori Android.

Centralità La centralità è una metrica usata per descrivere l'importanza di una particolare entità, o nodo, all'interno di una rete di entità interconnesse[33]. Una connessione tra qualunque coppia di nodi può essere vista come una dipendenza. Trattando tutte le classi nel sorgente Java come nodi e tutte le connessioni come dipendenze, l'intero codice sorgente Java può essere analizzato come una rete. La centralità di un nodo è direttamente proporzionale al numero di connessioni che il nodo ha.
Applicando la nozione di centralità ai 37 pacchetti Java, nel contesto del sorgente di Java SE 5, possiamo stimare quanto essi siano importanti in Java. Un risultato alto di centralità per i 37 pacchetti copiati indicherebbe che le classi al loro interno siano connesse ad un alto numero di classi al di fuori dei pacchetti in disputa, quindi che il resto del sorgente dipende fortemente dai 37 pacchetti.

Per calcolare la centralità è stato usato PageRank, un algoritmo realizzato da Larry Page e Sergey Brin durante lo sviluppo di un nuovo motore di ricerca, nel 1998. In generale, un risultato dato da PageRank per un'entità è direttamente proporzionale al numero di connessioni che ha, con ogni connessione che viene pesata in base al valore PageRank dell'entità connessa. Infine, PageRank valuta tutte le entità di una rete attraverso una probabilità distribuita il cui totale fa 1. Per questa ragione, un'entità PageRank può essere pensata come la probabilità che un cammino casuale all'interno della rete porti a quella specifica entità.
Esaminare la piattaforma Java SE 5 come una rete interconnessa, implica trattare ogni classe del codice sorgente come un nodo della rete. Per costruire la rete viene utilizzato il software Understand, che permette di analizzare le dipendenze tra classi. La rete considera solo le classi pubbliche. Una volta creata, la rete è stata valutata tramite il *tool* NetworkX, che ha assegnato un valore PageRank a ogni classe all'interno della rete. Una media dei risultati ottenuti è presente in Figura 4.10, i risultati dettagliati per ogni classe sono disponibili nel report di Kemerer[30]. Dalla Figura 4.10 sottostante vediamo che le classi copiate hanno un valore di centralità 9 volte maggiore rispetto alle classi non copiate. Questo mostra la centralità e l'importanza delle classi copiate in Java.

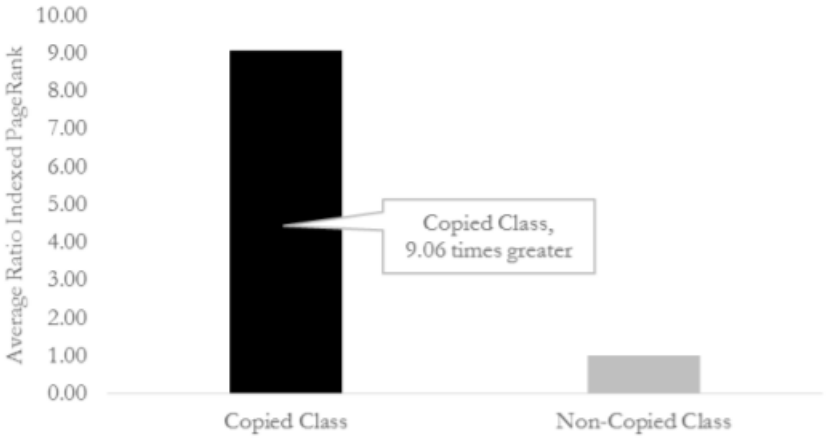


Figura 4.10: Media del valore PageRank per le classi copiate e non copiate in Java[30].

Andando ancora più nel dettaglio, Chris Kemerer ha calcolato la percentuale delle

classi copiate che compaiono nella top 10 delle classi (considerando tutto il sorgente Java) con valore PageRank maggiore. Lo stesso calcolo è stato realizzato per la top 20, la top 30 e così via fino alla top 100. I risultati sono in Figura 4.11. Quasi il 100% delle classi in Java SE 5 sono classi copiate da Google, anche guardando alla top 100, più dell'80% sono classi copiate.

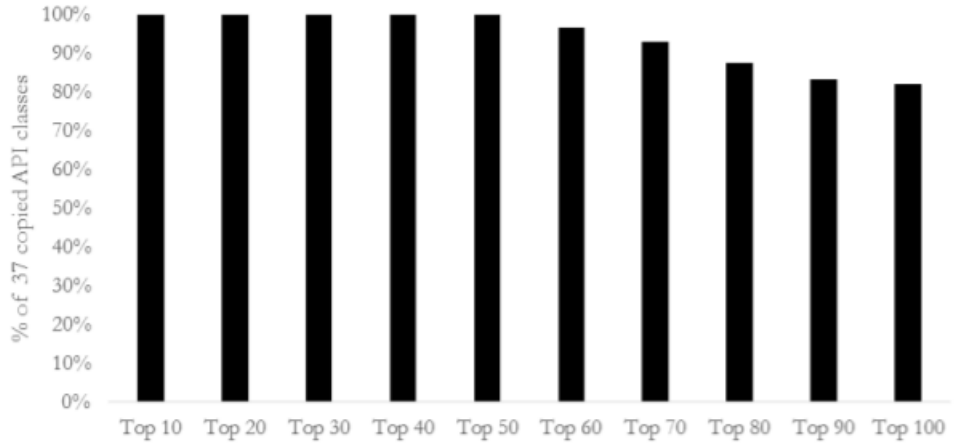


Figura 4.11: Percentuale delle classi copiate facente parti delle classi con i valori PageRank più alti[30].

La stessa esatta analisi è stata realizzata sul codice sorgente di Android, versione 5.1.0. (*Lollipop*, con lo scopo di misurare l'importanza del codice copiato all'interno di Android. Per quest'esame viene considerato solo il codice Android scritto in Java, poiché non è possibile per i 37 pacchetti Java dipendere direttamente su porzioni di codice non scritte in Java. Nuovamente i risultati sono stati ottenuti tramite la combinazione dei software Understand e NetworkX. I risultati in Figura 4.12 mostrano chiaramente come i pacchetti Java siano fortemente centrali per Android, registrando valori di centralità 32 volte maggiori rispetto al codice non copiato.

Figura 4.12: Media del valore PageRank per le classi copiate e non copiate in Android[30].

In Figura 4.13 vediamo invece i risultati per il calcolo della percentuale di classi copiate tra le classi più centrali in Android. L'esperimento è duale a quello realizzato precedentemente per Java. Abbiamo un'ulteriore prova della forte centralità dei 37 pacchetti Java nel codice Android. Infatti, quasi il 100% delle top 10,20,30,40 e 50 di Android sono classi che Google ha copiato. Quasi il 90% delle classi della top 100 sono classi copiate.

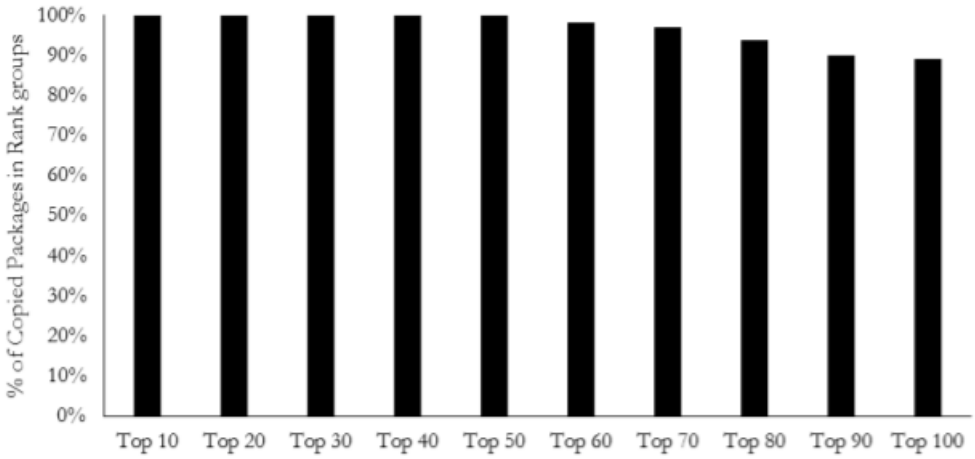


Figura 4.13: Percentuale delle classi copiate facente parti delle classi con i valori PageRank più alti[30].

Tutte e tre le metriche hanno dimostrato che il codice copiato è qualitativamente importante, sia per Oracle che per Google. Inoltre, stando ai dati di Bob Zeidman[34], Google ha copiato circa 11,000 linee di codice, il che rende la copia anche quantitativamente importante. Il numero di linee copiatale è tale da essere il nucleo di un intero programma software. Ad esempio, la navicella Apollo Lunar Module usata per missioni lunari conteneva un software di 10,000 linee di codice, in un contesto dove vite umane dipendevano dall'efficacia di quel codice.

4.1.2 Argomentazioni di Google

Google si è difesa dalle dichiarazioni di Oracle sollevando i punti seguenti:

1. Android non ha avuto alcun impatto negativo sullo scarso successo del mercato mobile di Sun (ora Oracle);
2. i 37 pacchetti non sono stati centrali per il successo di Android;
3. L'uso di Android è stato trasformativo, non è un sostituto per Java, ma una piattaforma *smartphone* innovativa;
4. la copia di Google è stata minima, il codice copiato costituisce solo l'1.6% del codice totale di Android.

L'esperto Google per questa fase è stato nuovamente il professore Owen Astrachan, purtroppo i suoi report per questa fase non sono pubblici. Per questo motivo, abbiamo richiesto gli *expert report* tramite mail al professore ma non ci sono stati forniti per motivi di privacy. Pertanto, le argomentazioni di Google non sono qui sotto supportate dall'analisi di esperti.

4.1.3 Verdetto della Corte Distrettuale

La Corte Distrettuale, dopo aver finito di ascoltare entrambe le parti, ha analizzato i quattro fattori e su ognuno ha decretato:

L'oggetto e la natura dell'uso

Oracle reputa che la natura dell'uso di Google sia negativa, ritenendo che Google abbia usato l'API solo per accelerare l'uscita sul mercato di Android, anche la Giuria ritiene che Google abbia sfruttato il codice di Oracle per uso commerciale. Tuttavia la natura *open source* e libera del codice Android, rende l'utilizzo non commerciale[31]. Per la Corte Distrettuale questa seconda componente prevale. Per valutare il primo fattore bisogna anche considerare se l'uso che si fa del materiale è trasformativo, cioè aggiunge qualcosa di nuovo, con uno scopo differente, modificando il materiale con una nuova espressione, significato o messaggio. L'uso di Google è stato giudicato fortemente trasformativo, in quanto ha implementato nuovamente i corpi delle funzioni, ha combinato il codice scritto con nuovi metodi e classi e lo ha adattato per il mondo *smartphone*. Considerato l'uso trasformativo e la componente non commerciale, il primo fattore è in favore del *fair use*.

La natura dell'opera protetta

Il linguaggio Java richiede di per sé la gerarchia pacchetto-classe-metodo, un'idea sulla quale Oracle non ha nessuna pretesa di copyright. Oracle reputa che poiché ci sono innumerevoli modi per nominare e strutturare le API, il processo sia da reputare creativo. La Corte ha reputato però che durante il processo sia prevalsa la parte funzionale rispetto a quella creativa, decretando così anche il secondo fattore favorevole al *fair use*.

La quantità e l'importanza della parte utilizzata

Google ha copiato solo le dichiarazioni, la struttura e l'organizzazione, ossia il minimo per preservare la coerenza tra i due sistemi, non ha copiato alcuna linea del codice implementativo. Il numero di linee ricopiate costituiva meno dell'un percento del totale del codice di Java (166 pacchetti API)[31]. Data la minimalità del codice copiato, il terzo fattore verte in favore di Google.

Le conseguenze dell'uso

L'uso di Google non ha causato danni ad Oracle, poiché le due parti operano su mercati diversi, Android è per il mercato degli *smartphone*, mentre le API di Java per il mercato *desktop* e *laptop*[31]. Oracle ritiene che Android abbia fatto da concorrente a Java ME, la versione per il mondo *mobile* delle API di Java, che Sun nel 2006 aveva iniziato a sviluppare. Tuttavia, la giuria ha reputato che il progetto di Java ME fosse già in declino prima che Android venisse rilasciato, quindi il sistema operativo di Google non ha causato nessun danno ad Oracle. Pertanto, il quarto fattore va a vantaggio di Google.

Tutti e quattro i fattori del *fair use* sono favorevoli a Google, la Corte Distrettuale ha decretato, il 26 maggio 2016, che l'uso di Google è *fair* e quindi non ha infranto il copyright. Oracle ha nuovamente fatto appello e il caso è cosi tornato alla Corte Federale per ridiscutere sul *fair use*.

4.2 Verdetto della United States Court of Appeals for the Federal Circuit

Nella fase iniziale, Oracle ha mostrato perché non si trovava in accordo con la sentenza della Corte Distrettuale su tutti e quattro i fattori del *fair use*:

1. L'oggetto e la natura di Google servono puramente scopi commerciali.
2. La natura del lavoro di Oracle è altamente creativa.
3. Google ha copiato 11330 linee in più di quelle necessarie per scrivere un programma basato su Java[35].
4. I clienti di Oracle hanno smesso di prendere le licenze di Java SE preferendo Android, poiché Google forniva libero accesso alle librerie.

La giuria si è espressa su tutti e quattro i punti.

4.2.1 L'oggetto e la natura dell'uso

In primo luogo, la natura *open source* e gratuita di Android non rende l'utilizzo dell'API educativo, infatti dare ad un cliente qualcosa gratis che solitamente è a pagamento può costituire uso commerciale, per questo motivo la giuria ha reputato l'uso dell'API prevalentemente commerciale. In aggiunta, l'uso non è reputato trasformativo, in quanto:

- Lo scopo del codice copiato è lo stesso in Android e in Java.
- Google non ha alterato l'espressione o il messaggio del materiale.
- Gli *smartphone* non erano un nuovo contesto, poiché le API di Java erano presenti negli *smartphone* (SavaJe, Danger, Nokia), prima dell'entrata sul mercato di Android[35].

Considerando l'uso fortemente commerciale e non trasformativo, il primo parametro va a favore di Oracle.

4.2.2 La natura dell'opera protetta

La giuria si era già espressa nell'appello precedente, reputando le dichiarazioni e la struttura e l'organizzazione sufficientemente creative per essere protette da copyright. Tuttavia, sempre la stessa giuria aveva definito minimo il livello di creatività, poiché la natura funzionale delle API è fortemente presente, basandosi su questa, le API di Oracle non vengono reputate abbastanza creative per ottenere il secondo fattore, che sorride a Google.

4.2.3 La quantità e l'importanza della parte utilizzata

Entrambe le parti affermano che solo 170 linee di codice erano necessarie per programmare usando Java, Google ha copiato 11330 linee in più del dovuto, e questo pesa contro il *fair use*. D'altra parte, Google ha usato 11500 linee su un totale di 2.86 milioni, la percentuale (0.4%)[35] è ridotta e perciò pesa a favore del *fair use*. Nessuno dei due punti sovrasta chiaramente l'altro, per questo la Corte valuta il terzo fattore neutrale.

4.2.4 Le conseguenze dell'uso

La Corte Federale ribadisce come Java SE fosse già utilizzato per anni negli *smartphone* prima del rilascio di Android, quindi Android è stato un competitore diretto di Java, nel mercato dei dispositivi mobili. Inoltre, Java SE era già utilizzato nei *tablet* Amazon Kindle, tuttavia dopo il rilascio di Android, Amazon ha abbandonato Java per adottare Android, questo prova che Android fosse usato come sostituto di Java e abbia avuto un impatto diretto sul mercato di Oracle[35]. Il quarto parametro va a discapito di Google.
Sommando i quattro fattori, due sono a favore di Oracle, uno in favore di Google e uno neutrale, perciò la Corte Federale, il 27 marzo 2018, ha decretato che l'uso di Google non rientrasse nel *fair use*, ha rimandato il caso alla Corte Distrettuale per decidere sull'ammontare dei danni.

Capitolo 5

Terza fase: decisione finale della Corte Suprema statunitense

5.1 Memorie legali

Nel gennaio 2019 Google ha presentato una petizione alla Corte Suprema degli USA, a questo punto il caso aveva preso interesse nazionale e diverse memorie legali sono arrivate alla Corte Suprema. Le memorie provenivano principalmente da esperti e professori di legge ed informatica, i quali ritenevano che il verdetto della Corte Federale fosse sbagliato e la Corte Suprema dovesse accettare la petizione di Google per rivedere il caso, con le seguenti argomentazioni della parte giuridica:

- La Corte Federale ha decretato che l'uso, dell'API di Java, da parte di Google non ha portato a una trasformazione del contenuto, affermando che Google non ha contribuito in modo creativo nell'uso del codice di Oracle. Questa decisione va contro numerosi precedenti (*Los Angeles News Serv. v. CBS Broad., Inc*[1],*Castle Rock Entm't, Inc. v. Carol Pub. Grp., Inc.*[2]) che giudicano un lavoro trasfor-

[1]Causa legale del 2002, in cui i due enti televisivi hanno discusso su l'uso di materiale video contenente riprese di una rissa. La Corte Suprema ha decretato che l'uso di CBS fosse *fair*.

[2]Causa legale del 1998, riguardante la celebre serie televisiva Seinfeld. L'azienda Carol Publishing aveva pubblicato un libro inerente alla serie, sulla quale Castle Rock detiene il copyright, senza avere una licenza, tuttavia anche in questo caso la Corte Suprema ha votato per il *fair use*.

mativo quando contiene un nuovo messaggio o contenuto, anche se questo porta pochi cambiamenti fisici all'originale. La Corte Suprema deve chiarire quale dei due approcci sia corretto[36].

- Il Copyright Act fornisce uno schema di quattro fattori che valutano il *fair use*, ma non gli dà un peso e non dice se altri parametri, oltre a questi quattro fattori, possono essere presi in considerazione. Riferendosi a questo caso, dando più importanza al primo fattore si favorisce Google, al contrario pesando più il quarto viene meno la pretesa di *fair use*. La Corte Suprema ha l'opportunità di definire i pesi dei quattro fattori[36].

- Dando ad Oracle i diritti esclusivi, per l'API di Java, sulle dichiarazioni e le chiamate, la Corte Federale ha protetto attraverso copyright la funzionalità di un sistema informatico. Questa decisione è conflittuale con il giudizio dato dalla Corte Suprema sul caso *Baker v. Selden*[3], nel quale afferma che il copyright su un lavoro scientifico non si estenda al metodo di operazione o ai diagrammi che usa per spiegare il lavoro, in accordo con questo giudizio i processi e i metodi all'interno dell'API non sono soggetti a copyright[37].

- La Corte Federale non ha pesato il secondo fattore del *fair use*, riferendosi ad un caso precedente (*Dr. Seuss Enters v. Penguin Books*[4]) dove la natura dell'uso non ha avuto peso, questo secondo caso però trattava storie per bambini e bambole, un contesto totalmente diverso da quello attuale. Infatti, in un altro processo (*Sega Enterprises v. Accolade*[5]) riguardante software il secondo fattore è risultato centrale nella discussione sul *fair use*[37].

[3] *Baker v. Selden* è stato un caso del 1879 che è diventato punto di riferimento per la dicotomia idea-espressione. In questo caso, la Corte Suprema ha dichiarato che un libro non da all'autore il diritto di escludere altri dal riprodurre ciò che è descritto nel libro (idea). L'autore ha solo il diritto di escludere la riproduzione del materiale nel libro (espressione).

[4] Caso del 1997 riguardante la disputa sulla pubblicazione di un libro. La Corte Suprema nell'analisi di questo caso non ha approfondito il secondo parametro del *fair use*.

[5] Nel caso *Sega Enterprises v. Accolade* l'azienda Accolade ha emulato tramite *reverse engineering* il software per la pubblicazione di videogiochi *Genesis*, di proprietà di Sega. La Corte Distrettuale della California nel 1992 si è dichiarata a favore di Sega.

- La Corte Federale è stata creata per fare chiarezza sui brevetti, la sua giurisdizione è esclusiva e non si amplia in altre aree, come sul copyright. In questo caso la Corte non si è limitata ai brevetti, ma ha ascoltato anche appelli riguardanti copyright, questa mancanza di fedeltà permette alle compagnie che producono software di aggirare le leggi regionali facendo appello direttamente alla Corte Federale, avvantaggiandosi dell'interpretazione espansiva che la corte ha sul copyright, ne è un esempio l'azienda Cisco[6][37].

- La dottrina del raggruppamento promuove la competizione sana e il progresso della scienza e delle arti utili. Il giudizio della Corte Federale su questa dottrina è sbagliato, poiché conclude che possa essere considerata solo se l'autore originale non ha alternative per esprimere quell'idea al tempo della realizzazione. Questa analisi è contraria a quella della Corte Suprema, che ritiene la dottrina del raggruppamento valida per dividere copyright ed espressione quando il design di un primo autore limita le opzioni del secondo arrivato[37].

- Le dichiarazioni Java che servono come interfaccia per programmi prescritti, contengono nomi, la Corte Federale reputa che questi nomi se creativi possono essere protetti da copyright, contrariamente il terzo e sesto circuito della Corte d'Appello hanno negato il copyright in circostanze simili (*ATC Distribution Group v. Whatever It Takes Transmissions & Parts*[7])[38].

- La Corte Federale ha applicato una revisione *de novo* sul caso, questo è stato un errore legale. Il settimo emendamento proibisce una revisione *de novo* sul giudizio di una precedente giuria riguardante *fair use*[40].

Ulteriori argomentazioni sono arrivate dalla parte informatica:

[6]Cisco Systems Inc. è un'azienda multinazionale specializzata nella fornitura di apparati di *networking*. In seguito al primo verdetto della Corte Federale nel caso "Google v. Oracle" nel 2014, Cisco ha approfittato del metro della Corte Federale e ha preso azione contro l'azienda Arista Networks.

[7]*ATC Distribution Group v. Whatever It Takes Transmissions & Parts* è stato un caso gestito dal sesto sircuito conclusosi nel 2005 a favore dell'azienda Whatever It Takes. La disputa era centrata su un catalogo per la vendita di proprietà dell'impresa ATC successivamente copiato da Whatever It Takes. Tuttavia la Corte di Appello ha reputato il catalogo non fosse abbastanza creativo per ottenere il copyright.

- La decisione della Corte Federale estende il copyright alle interfacce software, come l'API di Java, comparandole erroneamente a programmi per elaboratori. Un'interfaccia mostra l'insieme di comandi per usare un programma o un sistema. Ogni comando definisce un obiettivo funzionale che un programma deve compiere (trovare il massimo tra due numeri, ordinare una lista di numeri o mostrare testo a schermo) e include nomi, input e output, che insieme compongono la dichiarazione del comando. Le dichiarazioni sono puramente funzionali, indicano cosa fa un programma o un sistema senza specificare come lo fa. In contrasto, l'implementazione è l'insieme di comandi ed istruzioni usati direttamente o indirettamente dal computer per realizzare le operazioni specificate nella sua dichiarazione. La dichiarazione è l'idea, l'implementazione è l'espressione[39].

- Le interfacce possono includere migliaia di dichiarazioni, per facilitare l'orientamento gli sviluppatori danno alle API un'organizzazione. L'API di Java è strutturata su tre livelli, pacchetti, classi e metodi che corrispondono rispettivamente a cartelle, file e righe di un file. I programmatori che reimplementano o danno la loro implementazione per un'interfaccia devono mantenere la struttura, cambiandola si renderebbe l'interfaccia incompatibile, causando confusione per gli sviluppatori. Quindi Google per promuovere interoperabilità e permettere a Java di funzionare su un'intera classe di nuovi dispositivi ha dovuto reimplementare l'API di Java, copiando necessariamente la struttura dell'API e le sue dichiarazioni[39].

- L'uso dell'API è stato trasformativo, poiché ha creato Android che è un prodotto trasformativo. Le differenze tra i due contesti in cui lavorano Oracle e Google emergono da un'analisi del processo che Google ha compiuto per realizzare le sue API partendo dal codice di Java. Infatti, nel processo ha dovuto eliminare funzionalità, come ad esempio la gestione del *mouse*, e aggiungerne altre per gli *smartphone* come *GPS cracking*, gestione del cambiamento di rete e gestione dei comandi *touch*. Inoltre, il gruppo è d'accordo con la Corte Distrettuale che ne Sun ne Oracle avessero mai sviluppato la loro piattaforma smartphone con successo[39].

- Java e Android sono entrambe *open-source*, la natura *open-source* del prodotto di Google ha permesso ai programmatori di contribuire al miglioramento di entrambe

le interfacce contemporaneamente[39].

- Le interfacce software sono essenziali per l'innovazione, sono la base per creare nuove tecnologie e promuovere lo sviluppo, sono come il volante per l'industria automobilistica. La Corte deve rivedere la decisione per preservare la vitalità dell'industria software. Le interfacce software hanno dato il via alla rivoluzione dei PC, alla nascita dei principali sistemi operativi e dei linguaggi di programmazione moderni. Implementare nuovamente le interfacce permette alle start up di nascere e competere con chi è già stabilito nel settore, riduce il *network effect*. Proteggere le API con copyright permetterebbe alle compagnie già stabilite nel settore di monopolizzarlo, chiedendo licenze per usare i propri servizi ci sarebbe un aumento di costo per gli sviluppatori e di conseguenza un aumento di costo per i clienti[39].

Successivamente, sono arrivate anche le memorie legali a favore di Oracle, che erano d'accordo con il verdetto della Corte Federale e ritenevano che la Corte Suprema non dovesse ascoltare l'appello di Google, per le seguenti ragioni:

- Un lavoro per essere protetto dal copyright deve avere una soglia minima di creatività, che nasce quando l'autore aveva diverse possibilità per esprimere l'idea, tra le quali ha scelto. Nell'implementare un API lo spettro di scelte è ampissimo, pertanto le API di Oracle sono creative e quindi protette da copyright[42][44][45].

- Google sostiene che le API di Oracle non abbiano diritto a copyright, tirando in ballo la dottrina *merger*. Quando Sun ha sviluppato la *Java Standard Library* aveva opzioni illimitate sulla selezione e l'arrangiamento dei metodi. Considerato che la protezione da copyright va valutata al momento della creazione, è evidente che la dottrina *merger* non sussiste in questo caso[42][43][44][45].

- La dottrina dell'interoperabilità permette la copia di un lavoro sinché questa copia permetta al lavoro secondario di funzionare con l'originale. In questo caso, Google non ha voluto rendere Android compatibile con Java, infatti ha rifiutato le licenze offerte da Oracle (che richiedono compatibilità con Java), la dottrina dell'interoperabilità non può essere applicata[42][43][44].

- Apple e Microsoft sono riuscite a sviluppare il loro codice per sistemi operativi mobili da zero, senza aver bisogno di copiare codice da Oracle, ciò dimostra che anche Google avrebbe potuto realizzare il proprio codice autonomamente[43][44][45][46].

- L'uso di Google non è trasformativo, ha copiato il codice *verbatim* e lo ha circondato con il suo codice implementativo, ma non ha aggiunto niente al codice preso da Oracle. In particolare, lo ha usato per lo stesso scopo e per le stesse funzionalità, non cambiandone l'espressione o il significato. L'applicazione del codice ad un nuovo contesto (dispositivi mobili) non cambia la posizione di Google, in quanto Java esisteva nei dispositivi mobili prima dello sviluppo di Android[43][44][46][47].

- Il codice di Oracle è classificabile come metodo di operazione, poiché permette ad utenti di indurre un computer a realizzare certe funzioni, questo lato funzionale basterebbe secondo Google per non concedere al codice copiato il copyright. Tuttavia, seguendo la definizione di prima qualsiasi programma sarebbe funzionale e non si qualificherebbe per ottenere il copyright, la pretesa di Google è sbagliata[43][44][45].

- L'uso di Google del codice non rientra nel fair use, come visto sopra l'uso non è classificabile come trasformativo quindi il primo fattore va a favore di Oracle. Google ha copiato una porzione significativa di codice e ha copiato più di quanto necessario (11330 linee di codice) per scrivere in Java, perciò il terzo fattore va contro Google. Il quarto fattore sorride a Oracle, Google ha danneggiato notevolmente il mercato di Oracle, infatti prima del rilascio di Android, Java era già utilizzato in alcuni *smartphone* tra cui Blackberry e Nokia. Il codice di Android compete direttamente con quello di Oracle nel mercato dei dispositivi mobili[44][45][48].

5.2 Verdetto della Corte Suprema

La Corte Suprema accetta la petizione e il processo inizia il 7 ottobre 2020, la discussione riguarda l'uso di Google, in particolare se questo sia *fair*. La corte procede quindi ad analizzare i quattro fattori del *fair use*.

5.2.1 L'oggetto e la natura dell'uso

La giuria trascura parzialmente il fattore commerciale dell'uso, poiché esistono innumerevoli casi di *fair use* che sono commerciali[49], si concentra sulla componente trasformativa dell'uso. L'utilizzo delle API di Google mira a creare nuovi prodotti, ad espandere l'uso e l'utilità degli *smartphone* Android. Ha copiato l'API il minimo indispensabile per fornire funzionalità utili nel mondo *smartphone* e per lasciare agli sviluppatori una certa familiarità col linguaggio. Inoltre, il riuso dell'API è comune nell'industria, favorisce l'innovazione e crea nuove opportunità di mercato. Sommando queste valutazioni la giuria reputa il primo parametro in favore del *fair use*.

5.2.2 La natura dell'opera protetta

Il codice di un'API è diviso in tre componenti: le dichiarazioni dei metodi/classi, l'implementazione dei metodi/classi e le chiamate ai metodi, di queste tre solo la prima è oggetto di discussione. Il codice dichiarativo richiede di organizzare e strutturare l'API in maniera intuitiva e facile da usare per gli sviluppatori, è funzionale per natura. Il suo uso è inevitabilmente legato a idee non protette da copyright (divisione dei compiti e organizzazione). Il codice dichiarativo è quindi meno creativo e più lontano dal nucleo del copyright rispetto a molti altri programmi per computer, ad esempio l'implementazione dei metodi/classi. Il secondo fattore va a vantaggio di Google.

5.2.3 La quantità e l'importanza della parte utilizzata

Focalizzandosi solo sul codice dichiarativo, la porzione copiata è considerevole, ma la giuria si concentra sull'intera API di Java (2.86 milioni di linee) e usando questo metro di paragone la parte copiata risulta minimale, solo il 0.4%[49]. Google non ha copiato le 11500 linee per la loro originalità o creatività, le ha copiate per fornire continuità ai programmatori, in più la Corte Suprema non ritiene che Google sarebbe riuscito a sviluppare le stesse funzionalità copiando solo le 170 linee discusse dalla Corte Federale. Per la giuria anche il terzo fattore è a vantaggio del *fair use*.

5.2.4 Le conseguenze dell'uso

La giuria ritiene che Android non abbia danneggiato il potenziale mercato di Java SE e che Oracle non sarebbe riuscita ad entrare con successo nel mercato degli *smartphone* indipendentemente dalla copia dell'API di Google[49]. Queste considerazioni nascono dalla scarsa posizione nel mercato mobile che Sun aveva al tempo dell'uscita di Android, il mercato principale di Java erano *desktop* e *laptop*. In aggiunta, i prodotti sviluppati da Google erano più avanzati sul piano tecnologico, tanto che Google faceva parte di un mercato distinto e più avanzato rispetto a Java. La giuria ha anche considerato che l'introduzione di Android ha portato ad un uso più ampio di Java ed ha espanso la rete di programmatori Java. Valutate queste analisi, anche l'ultimo parametro va con Google. Tutti e quattro i parametri sono in favore del *fair use* da parte di Google. Il 5 Aprile 2021 la corte, con una maggioranza di sei a due, ha decretato che l'uso di Google rientra nei limiti del *fair use*, pertanto Google non ha infranto il copyright. L'intera industria tecnologica ha seguito il caso da vicino, preoccupata dalla possibili vittoria di Oracle, che avrebbe comportato una frenata nello sviluppo software, considerato il vasto utilizzo moderno delle API.

Capitolo 6

Conclusioni

Il plagio e la copia, nel mondo informatico, sono tematiche altamente discusse e spiacevoli episodi che si verificano in svariati scenari, partendo dall'ambiente universitario fino ad arrivare a grosse compagnie software. Nonostante la notorietà di questo trend, gli strumenti per affrontare questo problema ad oggi scarseggiano. "Oracle vs Google" è l'esempio più lampante della categoria, poiché riguarda il riuso di codice tra due colossi mondiali del settore. Inoltre, permette la trattazione dei punti più delicati del copyright legato a codice software, quali la sottile linea che separa copia e *fair use* e la grossa confusione che regna attorno a quali strumenti e metriche usare per decretare se la violazione di copyright sussista o meno. Attraverso l'analisi di questa causa legale, la tesi prova a fornire chiarezza in questo ambito. Infatti, sono trattati diversi strumenti (CodeMatch, confronto visuale, script *self-made*) e metriche (creatività, numero di linee simili) utilizzati dagli esperti di entrambe le parti e le analisi che diverse Corti statunitensi hanno effettuato sulla base di esse. In generale, vediamo come attualmente non sia presente un procedimento ben definito per dimostrare che un'entità ha copiato il codice da un'altra, tuttavia grazie alle problematiche sollevate dal caso analizzato può darsi si instauri un *modus operandi* per i casi successivi, che segua le orme tracciate da Oracle e Google.

Bibliografia

[1] Gkortis et al., *Software reuse cuts both ways: An empirical analysis of its relationship with security vulnerabilities* (2019), p.1.

[2] *Google LLC, v. Oracle America, Inc., Order re copyrightability of certain replicated elements of the Java Application Programming Interface* (2012), Northern District of California, Case 3:10-cv-03561 WHA.

[3] *Google LLC, v. Oracle America, Inc., Responsive expert report of James E. Malackowski* (2016), James E. Malackowski, Case 3:10-cv-03561 WHA.

[4] *Sun Microsystems*, Wikipedia, https://en.wikipedia.org/wiki/Sun_Microsystems.

[5] *History of Java programming language*, Free Java Guide, https://www.freejavaguide.com/history.html.

[6] P. Floren, *Sun's Java: Can it burn down Microsoft?*, (1997).

[7] *Dot-com bubble*, Wikipedia, https://en.wikipedia.org/wiki/Dot-com_bubble.

[8] *Sun to Open-Source under GPL*, Pratical Tech, https://practical-tech.com/2006/11/11/sun-to-open-source-java-under-gpl/.

[9] *Form 10-k: Google Inc.*, Sec, https://www.sec.gov/Archives/edgar/data/1288776/000119312508032690/d10k.htm.

[10] J. Callaham, Google made its best acquisition nearly 16 years ago: Can you guess what it was?, Android Authority, https://www.androidauthority.com/google-android-acquisition-884194/.

[11] *Android*,Wikipedia, https://en.wikipedia.org/wiki/Android_(operating_system).

[12] P. Menell, *Rise of the API copyright dead?: An updated epitaph for copyright protection of network and functional features of computer software.*, (2018).

[13] J. Brodkin, *The downfall of Sun Microsystems*, Networkworld, https://www.networkworld.com/article/2268096/the-downfall-of-sun-microsystems.html.

[14] *Oracle Buys Sun Microsystems For $7.4B*, CBS News, https://www.cbsnews.com/news/oracle-buys-sun-microsystems-for-74b/.

[15] *Google LLC, v. Oracle America, Inc.*, *Claim construction order* (2011), Northern District of California, Case 3:10-cv-03561 WHA.

[16] Vaccari et al., *Application Programming Interfaces in Governments: Why, what and how* (2020), p.18.

[17] J. Mitchell, *Opening expert report of John C. Mitchell regarding copyright* (2011), Northern District of California, Case 3:10-cv-03561 WHA.

[18] A. Purdy, *(Revised) Opening expert report of Alan Purdy regarding copyright* (2011), Northern District of California, Case 3:10-cv-03561 WHA.

[19] A. Purdy, *Reply expert report of Alan Purdy regarding copyright* (2011), Northern District of California, Case 3:10-cv-03561 WHA.

[20] *Android 11 compability definition*, Source Android, https://source.android.com/compatibility/android-cdd.pdf.

[21] O. Astrachan, *Opening expert report of dr. Owen Astrachan* (2011), Northern District of California, Case 3:10-cv-03561 WHA.

[22] O. Astrachan, *Reply declaration of dr. Owen Astrachan* (2011), Northern District of California, Case 3:10-cv-03561 WHA.

[23] Gosling et al.,*The Java Language Specification, First Edition* (1996), p. 108.

[24] *Google LLC, v. Oracle America, Inc.*, *Order denying Oracle's motion for judgment as a matter of law re patent infringment* (2012), Northern District of California, Case 3:10-cv-03561 WHA.

[25] B. Zeidman & E. Kovanis, *Did Oracle Overlook the Smoking Gun in its Case against Google?*, (2012), IPwatchdog, https://www.ipwatchdog.com/2012/06/26/did-oracle-overlook-the-smoking-gun-in-its-case-against-google/id=25843/.

[26] B. Zeidman, *Detecting Source-Code Plagiarism*, (2004), Dr. Dobb's, https://www.drdobbs.com/architecture-and-design/detecting-source-code-plagiarism/184405734?pgno=1.

[27] *Google LLC, v. Oracle America, Inc.*, *Appeals from the United States District Court for the Northern District of California in No. 10-CV-3561, Judge William H. Alsup.* (2014), United States Court of Appeals for the Federal Circuit, 2013-1021, -1022.

[28] A. Jaffe, *Expert report of professor Adam Jaffe* (2016), Northern District of California, Case 3:10-cv-03561 WHA.

[29] C. Kemerer, *Expert report of Chris Kemerer* (2016), Northern District of California, Case 3:10-cv-03561 WHA.

[30] C. Kemerer, *Expert report of Chris Kemerer regarding fair use and rebuttal to Google's opening expert reports* (2016), Northern District of California, Case 3:10-cv-03561 WHA.

[31] *Google LLC, v. Oracle America, Inc.*, *Order denying rule 50 motions* (2016), Northern District of California, Case 3:10-cv-03561 WHA.

[32] Y. Tian et al.,*What are the characteristics of high-rated apps? A case study on free Android applications.* (2015).

[33] R. A. Hanneman,*Introduction to social network methods.* (2005).

[34] *Google LLC, v. Oracle America, Inc.*, *Joint stipulation and [proposed] order regarding facts and documents related to Google's use of declaring code from Java SE 5 in Android* (2016), Northern District of California, Case 3:10-cv-03561 WHA.

[35] *Google LLC, v. Oracle America, Inc., Appeals from the United States District Court for the Northern District of California in No. 3:10-cv-03561-WHA, Judge William H. Alsup.* (2018), United States Court of Appeals for the Federal Circuit, 2017-1118, 2017-1202.

[36] *Google LLC, v. Oracle America, Inc., Brief of eight intellectual property scholars as amici curiae in support of petitioner* (2019).

[37] *Google LLC, v. Oracle America, Inc., Brief of professors Peter S. Menell and David Nimmer as amici curiae in support of petitioner* (2019).

[38] *Google LLC, v. Oracle America, Inc., Brief of 65 intellectual property scholars as amici curiae in support of petitioner* (2019).

[39] *Google LLC, v. Oracle America, Inc., Motion for leave to file brief of 78 amici curiae and brief of 78 amici curiae computer scientists in support of petitioner* (2019).

[40] *Google LLC, v. Oracle America, Inc., Brief of amici curiae civ pro, ip & legal history professors in support of petitioner* (2020).

[41] *Google LLC, v. Oracle America, Inc., Brief amici curiae of 83 computer scientists in support of petitioner* (2020).

[42] *Google LLC, v. Oracle America, Inc., Brief of amicus curiae interdisciplinary research team on programmer creativity in support of respondent* (2020).

[43] *Google LLC, v. Oracle America, Inc., Brief of amici curiae copyright thought leaders in support of respondent* (2020).

[44] *Google LLC, v. Oracle America, Inc., Brief of Ralph Oman amicus curiae supporting respondent* (2020).

[45] *Google LLC, v. Oracle America, Inc., Brief for the United States as amicus curiae supporting respondent* (2020).

[46] *Google LLC, v. Oracle America, Inc., Brief for professor Eugene H. Spafford, Ph.D., professor Zhi Ding, Ph.D., professor Emeritus Lee A. Hollaar, Ph.D., professor*

Adam Porter, Ph.D., and mr. Peter Kent, bsc as amicus curiae in support of respondent (2020).

[47] *Google LLC, v. Oracle America, Inc., Brief of amicus curiae nine professors and scholars of intellectual property law in support of respondent* (2020).

[48] *Google LLC, v. Oracle America, Inc., Brief amicus curiae of 25 professors of Journalism and media law in support of respondent* (2020).

[49] *Google LLC, v. Oracle America, Inc., Certiorari to the United States court of appeals for the Federal Circuit* (2021), Supreme Court of the United States, No. 18-956.