UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Analisi sull'evoluzione dei linguaggi di programmazione lungo l'arco della Storia

RELATORI

Prof. Fabio Palomba

Dott. Giammaria Giordano

Università degli Studi di Salerno

CANDIDATO

Angelo Palmieri

Matricola: 0512111009

Anno Accademico 2022-2023

Questa tesi è stata realizzata nel

"Ha toccato! Ha toccato!" - Tito Stagno

Abstract

I linguaggi di programmazione sono uno degli strumenti più importanti presenti in ambito informatico, questo perché consentono agli esseri umani di impartire dei comandi alle macchine tramite istruzioni, andando di fatto a "programmarle".
Oggi giorno l'utilizzo dei vari linguaggi di programmazione ci circonda ed è alla portata di tutti poter imparare ad usare uno o più linguaggi, ma non sempre è stato così.
Ciò che questa Tesi si prefigge è: descrivere l'evoluzione dei linguaggi di programmazione fornendo un taglio e un contesto storico; individuare le motivazioni che fanno si che i paradigmi di programmazione siano in costante mutamento nel tempo; comprendere se è possibile progettare un modello concettuale che fornisca un supporto agli sviluppatori nella scelta di un paradigma di programmazione, a seconda delle esigenze che si hanno.
Dai risultati ottenuti sorgeranno tre principali motivazioni che causano l'evoluzione dei paradigmi di programmazione nel tempo: **sviluppo tecnologico**, **incremento qualità** e **didattica**.
Si riterrà poi possibile costruire il modello concettuale che aiuti gli sviluppatori nella scelta del paradigma di programmazione che più fa al caso loro, notando che con ogni probabilità potrebbe essere ancora più di supporto per tutti gli studenti che si avviano a conoscere il mondo della programmazione.
In futuro si potrebbero considerare altri paradigmi di programmazione, individuare nuove motivazioni e arricchire il modello concettuale.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Contesto applicativo

1.1.1 Linguaggi di programmazione: cosa sono e a cosa servono

Per definire cos'è un linguaggio di programmazione, la Treccani[1] fornisce la seguente definizione:

> "Un linguaggio di programmazione è un insieme di parole e regole, definite in modo formale, per consentire la programmazione di un elaboratore affinché esegua compiti predeterminati."

Stando all'appena citata puntualizzazione, si può pensare ai linguaggi di programmazione come una sorta di ponte che fa da collante tra:

- **Il Programmatore**, il quale invia le istruzioni che vuole vengano eseguite;
- **Il Computer**, che una volta comprese le istruzioni procede con l'eseguirle.

Difatti lo scopo di un linguaggio di programmazione è proprio questo, ovvero rendere possibile la comunicazione tra due figure così eterogenee e distanti tra loro, gli esseri umani e le macchine.

1.1.2 Categorie di linguaggi di programmazione

Attualmente esistono moltissimi tipi di linguaggi di programmazione, ma si possono raggruppare tutti in due macrocategorie come mostrato nel testo "Fondamenti di Programmazione"[2]:

- **Linguaggi a basso livello**: sono chiamati così perché considerati più vicini, in termini di grado di astrazione, alla macchina hardware.
  In questi linguaggi è direttamente la macchina hardware che va a leggere ed eseguire le varie istruzioni che compongono il programma. In tal modo si ha di certo un'elevata ed efficiente rapidità di esecuzione, a discapito però di un'enorme complessità di utilizzo per un programmatore.
  Essendo tutte le istruzioni scritte in notazione binaria, questo rende la vita difficile a chi deve scrivere anche il più banale dei programmi.

- **Linguaggi ad alto livello**: sono chiamati così perché considerati più vicini, in termini di grado di astrazione, all'essere umano.
  Questa categoria di linguaggi, a differenza di quelli a basso livello, semplificano di molto la scrittura di un programma grazie alla semantica delle istruzioni più vicina al linguaggio regolare.
  I programmatori sono di certo molto agevolati nello scrivere i vari programmi, ma d'altro canto questo richiede che ci siano degli strumenti aggiuntivi che permettano l'esecuzione sulla macchina hardware, andando così a perdere velocità ed efficienza.

1.1.3 I paradigmi di programmazione

Un concetto molto importante da introdurre è quello relativo ai **paradigmi di programmazione**.
Un paradigma di programmazione è un determinato approccio, basato su delle regole specifiche e degli strumenti concettuali, di cui il programmatore può scegliere di servirsi per scrivere del codice che risolva il problema desiderato.
In letteratura esistono molti paradigmi ma, come viene detto in "An Insight into Programming Paradigms and Their Programming Languages"[3], a dominare il panorama mondiale sono i seguenti:

- **Programmazione Imperativa**: in questo paradigma il programmatore impartisce degli ordini, da qui il nome *imperativa*, alla macchina per indicare cosa vuole che venga fatto ma soprattutto può specificare **come** deve farlo.
  Questo fornisce un controllo assoluto al programmatore su ogni aspetto, come ad esempio la gestione della memoria allocata nel programma da lui scritto, permettendogli dunque di sfruttare le piene potenzialità del sistema. D'altra parte però il codice risulta essere complesso da comprendere e manutenere, oltre ad essere soggetto a frequenti errori se il programmatore non è abbastanza ferrato.

- **Programmazione Funzionale**: in questo paradigma il programmatore può suddividere il programma che vuole scrivere in più **funzioni**, che hanno la capacità d'interagire a vicenda anche in maniera ricorsiva.
  Una funzione può prendere in input dei dati e restituire in output un risultato, calcolato dalle istruzioni che si trovano all'interno del cosiddetto *corpo della funzione*[1].
  Il poter suddividere un grande programma in tante piccole parti collegate tra loro, rende certamente più comprensibile il codice e la sua manutenzione. Il principale svantaggio sta nel fatto che, per programmare seguendo questo paradigma, si richiede d'imparare a ragionare in un certo modo, il che non è immediato.

[1] Non è obbligatorio che una funzione prenda in input dei dati e restituisca in output un certo risultato, ma può anche eseguire solo le operazioni presenti nel suo corpo.

- **Programmazione Orientata agli Oggetti**: in questo paradigma il programmatore può modellare tutte le entità del mondo reale all'interno del suo programma grazie ai concetti di **Classe** e **Oggetto**.
  Le classi servono ad astrarre l'entità reale, andandola a descrivere tramite un insieme di attributi (caratteristiche principali) e metodi (procedure che operano sugli attributi). Gli oggetti invece sono un'istanza specifica di una determinata classe.
  Questo paradigma si rivela molto utile nella realizzazione di progetti grandi e permette agilmente il riutilizzo del codice, tuttavia anche qui è richiesto l'apprendimento di nozioni chiave per padroneggiare questo paradigma, il tutto con la dovuta attenzione.
  Inoltre non è un paradigma consigliato quando si ha a che fare con problemi semplici, certamente risolubili seguendo altri approcci più efficaci.

- **Programmazione Event-Oriented**: in questo paradigma il programmatore può creare un programma che reagisca in un certo modo al verificarsi di determinati eventi.
  Ogni evento che si desidera considerare viene gestito da quello che è chiamato **handler**, il quale è in costante ascolto nell'attesa che lo specifico evento da lui gestito si verifichi, e quando ciò avviene esegue una predeterminata azione.
  Questo paradigma trova molta applicazione quando bisogna creare delle applicazioni Web, bisogna però prestare molta attenzione nella fase di debug e testing poiché non è semplice individuare gli errori.

Si noti che per poter utilizzare un certo paradigma di programmazione non è obbligatorio scegliere un linguaggio di programmazione specifico, bensì è più corretto affermare che ci sono dei linguaggi che favoriscono l'applicazione di uno o più paradigmi.

1.2 Obiettivi

Questa Tesi si prefigge diversi obiettivi, il primo tra questi è quello di fornire una panoramica su quelli che sono stati i punti salienti dell'**evoluzione della programmazione**, a partire dalla metà del XIX secolo fino a giungere ai giorni nostri.
In letteratura sono presenti molti articoli che analizzano l'evoluzione dei linguaggi di programmazione e dei paradigmi, molti di essi però si concentrano in particolare su un ristretto gruppo di linguaggi [2] oppure su uno specifico lasso di tempo, questa Tesi invece mira a considerare un **ampio insieme**[3] di paradigmi e linguaggi di programmazione, puntando inoltre a coprire l'**intero arco temporale** della Storia della programmazione.
Una parola fondamentale è proprio quella appena scritta sopra, **Storia**, infatti come si potrà vedere nel proseguo gli verrà data molta importanza lungo tutta l'analisi evolutiva, conferendo così alla Tesi un taglio storico.
In seguito verranno poi esposte delle **domande di ricerca** (sottosezione 2.1.1) che permettono di definire in maniera più precisa gli obiettivi della Tesi, come l'individuazione delle motivazioni che portano i paradigmi di programmazione e cambiare nel tempo e la costruzione di un modello concettuale che suggerisca la scelta di un certo paradigma, piuttosto che un altro, basandosi sulle esigenze degli sviluppatori.

[2] Molti articoli approfondiscono anche un singolo linguaggio.

[3] Per quanto ampio sia questo insieme chiaramente non rappresenta tutti i paradigmi o tutti i linguaggi di programmazione, sarebbe un numero spropositato, ma rappresenta quelli che sono ritenuti più importanti ai fini dell'analisi evolutiva.

1.3 Risultati

Una volta completata l'analisi sono affiorate le tre seguenti motivazioni principali che determinano il cambiamento dei paradigmi di programmazione col passare del tempo:

1. **Sviluppo tecnologico**
2. **Incremento della qualità**
3. **Didattica**

Si è inoltre notato che era possibile produrre un modello concettuale (mostrato in Figura 4.3) che supporti gli sviluppatori e gli studenti nella scelta di un paradigma di programmazioni, basandosi sulle esigenze di quest'ultimi.
Chiaramente quanto detto qui viene approfondito nel Capitolo 4 dedicando esclusivamente all'analisi dei risultati ottenuti.

1.4 Struttura della Tesi

Dopo avere compiuto una breve introduzione si riporta di seguito la struttura della Tesi a partire dal prossimo Capitolo.

- **Capitolo 2, Stato dell'arte**: qui si osserverà l'evoluzione avuta dai linguaggi di programmazione a partire da metà ottocento con **Ada Lovelace** fino al 2023, andando a citare gli articoli che vanno a comporre lo Stato dell'arte. Verranno inoltre definite le domande di ricerca.
- **Capitolo 3, Metodologia**: qui saranno descritti i vari passaggi che costituiscono la **Systematic Literature Review** condotta per questa Tesi, ovvero il metodo che è stato utilizzato per ottenere gli articoli utili all'analisi dei risultati.
- **Capitolo 4, Risultati**: qui saranno analizzati i risultati ottenuti in risposta alle domande di ricerca, il tutto completo di grafici per una migliore disamina.
- **Capitolo 5, Conclusioni**: qui saranno tratte le conclusioni dell'intera Tesi, con annessi possibili sviluppi futuri.

CAPITOLO 2

Stato dell'arte

2.1 Background

2.1.1 Presentazione delle domande di ricerca

Tra i diversi obiettivi di questa Tesi vi è anche quello di provare a rispondere a due **domande di ricerca** che si vanno di seguito a presentare.

> *RQ 1: Quali sono le motivazioni che hanno portato a questo cambio di paradigma nel tempo?*

Questa prima domanda di ricerca vuole evidenziare le motivazioni che sono dietro al cambio di un determinato paradigma di programmazione, com'è nato e la sua evoluzione nel tempo.
Per rispondere a tale domanda si analizzeranno le evidenze storiche che sorgeranno nelle prossime sottosezioni di questo capitolo.

RQ 2: È possibile creare un conceptual model che permetta agli sviluppatori di selezionare un certo paradigma in base alle sue esigenze?

La seconda domanda di ricerca vuole comprendere se sia realizzabile la creazione di un conceptual model in grado di indicare agli sviluppatori un preciso paradigma di programmazione, così da essere quanto più conforme a quelle specifiche esigenze. Per rispondere a tale domanda verrà utilizzato quanto prodotto a risposta della RQ1. Entrambe le domande di ricerca saranno discusse nel capitolo dedicato ai risultati.

2.1.2 Nascita dei linguaggi di programmazione

1843, questo è l'anno riconosciuto da tutta la comunità Informatica per indicare la nascita del primo linguaggio di programmazione[1]. A descriverlo quasi 200 anni fa fu **Ada Lovelace**, una delle menti più eccelse che l'intera Umanità abbia mai avuto il piacere di ammirare.
Per capire come si arrivò a questo, grazie a quanto descritto puntualmente da Silvio Hénin in "Augusta Ada Lovelace (1815- 1852)"[4], facciamo un passo indietro.
Augusta Ada Byron nasce a Londra il 10 Dicembre del 1815, i suoi genitori erano il poeta Lord Byron e la matematica Annabella Milbanke e ad appena un mese di vita la madre lasciò casa del marito portandosi con se Ada. Lord Byron non rivendicò mai i suoi diritti di paternità e lasciò la tutela della figlia ad Annabella.
Sebbene all'epoca erano decisamente pochi gli istituti che consentivano la frequenza alle donne, ad Oxford e Cambridge per esempio era vietata la presenza di donne, la madre Annabella decise lo stesso di far studiare Ada Lovelace, studi che avvenivano sempre e soltanto in casa. Questo le permise di avere una formazione in diverse materie scientifiche come la matematica.
Passano gli anni e Ada Lovelace comincia a frequentare salotti in cui si tenevano conferenze scientifiche e nuove invenzioni venivano presentate. Uno dei salotti più famosi era quello del matematico Charles Babbage ed è proprio qui che i due si conobbero.

[1] Per essere precisi si tratta di un linguaggio meccanico, antenato dei linguaggi di programmazione.

Ada Lovelace era sempre più incuriosita degli studi di Charles Babbage e così cominciò a studiare la macchina differenziale e la macchina analitica. Proprio su quest'ultima ci furono i risvolti più importanti per la Storia dell'Informatica.
Nel 1840 Charles Babbage fu invitato all'Università di Torino per un seminario in cui parlava della sua macchina analitica, successivamente l'ingegnere italiano Luigi Federico Manabrea pubblicò nel 1842 un saggio sulla macchina analitica di Babbage, non essendo però scritto in lingua inglese Charles Babbage chiese ad Ada Lovelace di tradurlo e di aggiungerci delle note.
Giunti finalmente nel 1843, Ada Lovelace tradusse il saggio di Manabrea e aggiunse 7 corpose note, tanto corpose che la traduzione era tre volte più grande, in termini di lunghezza, rispetto all'articolo originale.
Ada Lovelace descrisse per la prima volta la macchina analitica come uno strumento programmabile con una sorta di intelligenza simile a quella dell'essere umano, anticipando il concetto di intelligenza artificiale. Ad ogni nota venne assegnata una lettera dalla A alla G, ed è in quest'ultima nota che Ada Lovelace descrive un algoritmo capace di calcolare i numeri di Bernoulli.
Questo algoritmo è riconosciuto come la prima apparizione nella Storia di un programma informatico.
Tutto questo fu possibile solo grazie ad Ada Lovelace, che viene descritta da Avery Elizabeth Hurt[5] come una donna che senza nessuna istruzione universitaria, sia stata capace di scrivere il primo programma informatico della storia più di un secolo prima in cui venisse realizzato un computer che potesse eseguirlo. Ada Lovelace non firmò neanche la sua celebre traduzione per esteso, ma scrisse solo le sue iniziali.
Si pensa che fece in tal modo per non attirare l'attenzione sul fatto che l'autrice fosse donna. Oggi è universalmente riconosciuta come pioniera dell'Informatica.

2.1.3 Herman Hollerith e le schede perforate

Qualche decennio dopo gli studi di Ada Lovelace, gli Stati Uniti d'America si stavano avviando verso il censimento del 1880.
L'organismo federale che aveva l'incarico di raccogliere ed elaborare tutti i dati presenti nel censimento era il Census Bureau di Washington, ed è qui che trovò impiego uno dei cosiddetti padri fondatori dell'Informatica, **Herman Hollerith**.
Come illustrato in "Buon compleanno Mr. Hollerith"[6], Hollerith fu uno dei più grandi ingegneri statunitensi che, dopo essersi diplomato, cominciò a lavorare per il Census Bureau nella posizione di Capo Agente Speciale.
Nel censimento del 1880 collaborò con il dottor John Shaw Billings nell'elaborazione delle statistiche sanitarie, fu proprio in questa occasione che Herman Hollerith si rese conto che era divenuto necessario automatizzare tutto il processo del censimento.
Tra il 1840 e il 1880 la popolazione statunitense era più che triplicata e anche la gamma di informazioni che venivano raccolte ad ogni censimento si ingrandiva sempre di più. Di conseguenza il tempo richiesto per l'elaborazione dei dati aumentava di decennio in decennio, e la situazione si fece drasticamente grave proprio nel censimento del 1880, nel quale furono necessari ben 7 anni per elaborare tutti i dati. Andare avanti così era impossibile e bisognava trovare una soluzione, così Herman Hollerith si mise al lavoro per risolvere il problema, con la speranza di riuscirci entro il censimento del 1890.
Come descritto da Fabio Venuda[7], nel 1889 Herman Hollerith inventò un nuovo sistema di codifica e analisi delle informazioni rilevate dai censimenti, andando difatti a rivoluzionare il modo in cui si calcolavano enormi quantità di dati.
Il sistema si basava sulle cosiddette **schede perforate**, le informazioni raccolte venivano di fatto codificate tramite dei fori eseguiti su delle schede di cartoncino, da qui il nome di schede perforate. I fori non venivano eseguiti casualmente ma in delle posizioni ben predefinite, per permettere agli aghi dell'apparato di lettura[2] di passare soltanto attraverso i fori. Una volta oltrepassati i buchi, gli aghi raggiungevano il circuito sottostante alla scheda e inviavano degli impulsi elettrici che innescavano dei contatori, i quali servivano per registrare le varie risposte degli utenti.

[2]Una pressa composta da degli aghi elettrificati che potevano ritrarsi.

Giunto il momento del giudizio con il censimento del 1890, il sistema ideato da Hollerith si dimostrò affidabile ed efficiente sia in termini di tempo che in termini economici, basti pensare che lo stesso Hollerith stimava di aver fatto risparmiare al Governo degli Stati Uniti circa 5 milioni di dollari dell'epoca.
La programmazione continuava così ad evolversi e a trovare sempre più applicazione, fino ad essere utilizzata anche dagli organi governativi.

2.1.4 L'avvento dei linguaggi di programmazione moderni

Facendo un balzo in avanti di circa mezzo secolo, si giunge agli anni '40 del XX secolo. Il Pianeta intero è teatro del più grande conflitto armato della Storia, la Seconda Guerra Mondiale.
Oltre alle trincee, al mare e ai cieli di tutto il mondo, la Guerra si combatteva anche tra i più grandi scienziati dell'epoca, che col passare del tempo progettarono macchine sempre più all'avanguardia.
Furono così creati i primi Computer moderni alimentati tramite corrente elettrica, ma per poter usufruire di tutte le loro potenzialità servivano dei linguaggi di programmazione.
Stando a quanto si dice in "Storia della programmazione"[8], quello che è considerato il primo linguaggio di programmazione ad alto livello progettato per un computer è **Plankalkül** . Ad idearlo fu **Konrad Zuse**, informatico ed ingegnere tedesco, e il nome che diede al linguaggio deriva da:

- Plan: che sta per *"pianificazione"*;
- Kalkül: termine tedesco che sta per *"sistema informale"*;

Era il 1945 quando Zuse era pronto a pubblicare un libro in cui andava ad illustrare le specifiche del Plankalkül, il fato però volle che nello stesso anno cadde il Terzo Reich e il disordine generale non permise di rende pubblico il libro, rendendo difatti il Plankalkül inutilizzato. Come descritto in "L'evoluzione dei linguaggi di programmazione: analisi e prospettive."[9], lungo quegli anni si programmava tramite codici binari e linguaggi assemblativi, ovvero linguaggi di programmazione di basso livello molto vicini alla macchina, il che rendeva complessa la scrittura di un programma.

Per veder dunque comparire i primi linguaggi di programmazione ad alto livello ampiamente utilizzati, bisogna arrivare alla metà degli anni '50 con **FORTRAN** (FORmula TRANslation), linguaggio che venne sviluppato tra il 1955 e il 1957 da una delle più importanti aziende informatiche al mondo, nonché la più antica, **IBM**[3]. FORTRAN nacque per soccorrere gli scienziati che necessitavano di un linguaggio di programmazione che rendesse possibile la scrittura di complesse formule matematiche e che consentisse poi di poterle elaborare. Trovò pertanto grande applicazione nel calcolo scientifico e divenne ben presto famoso in tutta la comunità, riscuotendo così un successo che permise a FORTRAN di essere usato per molti anni a seguire.
Poco dopo, nel 1958, **John McCarthy** ideò il **LISP** (LISt Processor), con l'obiettivo di trasportare il modello computazionale del λ-calcolo sotto forma di linguaggio di programmazione.
Il λ-calcolo era uno dei modelli computazionali più utilizzati, e riuscire ad avere un linguaggio di programmazione che si basasse su di esso fu la chiave per la risoluzione di una vasta gamma di problematiche che intaccavano lo sviluppo software. Inoltre l'apprendimento del LISP risultò molto facile per gli scienziati che avevano conoscenze pregresse del λ-calcolo, il che gli conferì un utilizzo duraturo.
Un altro linguaggio che originò in questi anni, più precisamente a inizio degli anni '60 fu il **COBOL** (COmmon Business-Oriented Language), un ruolo centrale nel suo sviluppo fu quello di **Grace Murray Hopper**, considerata da tutti come una delle personalità più importanti nella Storia della programmazione informatica.
COBOL venne ideato in seguito alla sempre più avvolgente espansione dell'informatica con il mondo dell'economia, con lo scopo di poter gestire le transazioni economiche in un linguaggio di programmazione, il che risultò molto utile per coloro che si occupavano di amministrazione, contabilità e finanza.
Riuscì così a trovare molta applicazione in settori come quello bancario nel quale ancora oggi viene utilizzato, dato che i sistemi utilizzati si sono evoluti a partire da quelli scritti negli anni '60 con COBOL.

[3]Tra i fondatori di IBM ritroviamo anche Herman Hollerith

Infatti, stando a quanto scritto da Federico Flacco et al[10], persistono tutt'ora dei corsi mirati alla formazione di programmatori con conoscenze di COBOL, questo perché ci sono delle procedure scritte in COBOL che pur essendo vecchie funzionano ancora in maniera efficiente a tal punto da non essere necessario sostituirle e, pertanto, vanno sapute gestire.

2.1.5 Pascal e C

Dopo diversi anni in cui linguaggi di programmazione come FORTRAN e COBOL si affermavano sempre di più nei rispettivi settori, un docente di programmazione svizzero di nome **Niklaus Wirth** creò uno dei linguaggi più diffusi di sempre, il **Pascal**[4].
Stando a quanto descritto in "Pascal Programming Language"[11], Wirth percepiva il bisogno di un linguaggio di programmazione che potesse essere usato anche nel mondo della didattica informatica, uno strumento che potesse insegnare le nozioni fondamentali della programmazione, sviluppò così il Pascal che divenne operativo nel 1970.
Il Pascal venne progettato in maniera tale da unire in un unico linguaggio quelle che erano considerate le caratteristiche migliori sparse tra gli altri linguaggi in voga all'epoca, come FORTRAN e COBOL, andando così a raffinare e correggere anche gli aspetti più deboli di questi linguaggi. Ciò portò ad ampliare di molto il bacino d'utenza di Pascal e lo rese uno dei linguaggi più diffusi al mondo.
Più o meno nello stesso periodo, tra il 1969 e il 1973, venne sviluppato un altro linguaggio di programmazione che, come viene anche detto in "Evoluzione dei linguaggi di programmazione educativi: Scratch e BYOB"[12], si differenzia dal Pascal dato che non era pensato per la didattica, infatti, sebbene non avesse molte regole da rispettare, la semantica complessa rendeva arduo il suo approccio e apprendimento.

[4]Chiamato così in onore del matematico ed inventore francese Blaise Pascal, che inventò quella che è considerata essere la prima calcolatrice della storia, la Pascalina.

Il linguaggio in questione è **C**[5], sviluppato da **Dennis Ritchie** per poterlo utilizzare nella progettazione del sistema operativo UNIX.
Stando a quanto viene illustrato in "History of C"[13], C aveva si una semantica complessa che rendeva più difficile imparare ad usarlo, ma questo era anche il suo punto di forza dato che rendeva i programmi scritti con questo linguaggio di programmazione più efficienti e veloci.
Un altro notevole e importante aspetto di C sono le funzioni, porzioni di codice adibite a compiere una determinata serie di istruzioni. Le funzioni possono essere già state scritte da altri programmatori e messe a disposizioni di tutti tramite le librerie, che basta semplicemente includere[6] all'interno del proprio programma per poter sfruttare le funzioni di cui si necessita, ma ogni programmatore può anche scrivere ed utilizzare le proprie funzioni. Questo all'epoca rivoluzionò il modo in cui si programmava rendendo il debugging, il testing e la manutenzione di un programma molto più facile da eseguire.
Per questi motivi, e anche grazie al fatto che UNIX continuava ad affermarsi sempre di più tra i sistemi operativi più usati al mondo, C ottenne enormi consensi nel mondo informatico, riuscendo addirittura a scalzare in poco tempo un linguaggio come Pascal, che negli anni '70 si era imposto a livello globale, il che la dice lunga su quanto il mondo dei linguaggi di programmazione sia in constante e continua evoluzione.

[5]Prese questo nome perché molte delle sue funzionalità vennero prese da un linguaggio precedente di nome B, che a sua volta era stato il successore di BCPL.

[6]Un esempio d'istruzione per poter includere una libreria è la seguente: *#include<stdio.h>*, dove "stdio.h" sta ad indicare il nome di una libreria, in questo caso rende disponibili tutte le funzioni che lavorano sullo standard input/output.

2.1.6 La programmazione orientata agli oggetti

L'evoluzione dei linguaggi di programmazione continua imperterrita fino a giungere all'ennesima rivoluzione, la comparsa della **programmazione orientata agli oggetti**.
Il primo linguaggio che seguiva questo nuovo paradigma della programmazione, stando a quanto detto in "Object-Oriented Programming: Themes and Variations"[14], fu **Simula** che venne sviluppato da quelli che sono considerati i pionieri della programmazione orientata agli oggetti, gli informatici norvegesi **Ole-Johan Dahl** e **Kristen Nygaard**.
Come lo stesso Nygaard scrisse in "Basic concepts in object oriented programming"[15], il concetto di "programmazione orientata agli oggetti" ha genesi negli anni '40 quando ci fu l'introduzione di un importante strumento come la simulazione digitale.
Nel 1949 Nygaard era coinvolto nella creazione del primo reattore nucleare norvegiese, tra i vari problemi che si presentarono lungo tutto il processo vi fu quello di calcolare il diametro che avrebbero dovuto avere le barre di uranio all'interno del reattore. Il modello ad equazione integrale utilizzato per fare ciò venne sostituito dalla **simulazione Monte Carlo**, che permise di generare un gran numero di neutroni, effettuare un'analisi statistica delle loro proprietà e infine stimare quale fosse il diametro più adatto per le barre di uranio.
Si seguì questo approccio molte altre volte in seguito, laddove non era possibile generare nessun modello matematico si procedeva col rappresentare tramite un computer ciò che andava analizzato, facendo combaciare alle entità del mondo reale le entità presenti nell'esecuzione del programma.
Ed è proprio questo il fulcro di questo nuovo paradigma, avvicinarsi di più al modo in cui noi esseri umani interagiamo con gli oggetti presenti nel mondo reale.
Come illustrato da Matteo Baldoni et al[16], nella realtà possiamo avere delle interazioni con un oggetto anche senza conoscere i dettagli del suo funzionamento, ma ci basta sapere quale risultato produce. Un classico esempio è quello dell'accensione di un automobile, dove al conducente non è richiesto che conosca i dettagli tecnici che scaturiscono l'avviamento del motore, ma gli è sufficiente sapere che inserendo la chiave nell'apposita fessura, e ruotandola, l'automobile si accenderà.

Come detto prima, Simula fu il capostipite dei linguaggi di programmazione orientata agli oggetti, e come tale influenzò molti altri linguaggi tra cui **C++**.
C++ venne realizzato da **Bjarne Stroustrup** nel 1983 e, come riportato in "The History and Evolution of Java"[17], tra la fine degli anni '80 e l'inizio degli anni '90 sembrava che fosse il linguaggio di programmazione migliore al mondo, poiché possedeva la grande efficienza di C unita alle caratteristiche della programmazione orientata agli oggetti come l'incapsulamento e il polimorfismo.
Ma questo status non era destinato a perdurare in eterno perché l'evoluzione è costante e lo scenario sarebbe stato ribaltato dall'ennesima rivoluzione tecnologica, l'ascesa su scala mondiale del **World Wide Web** e di **Internet**.
Si creò così l'ambiente perfetto per **Java**, rilasciato nel 1995 dall'azienda **Sun Microsystems** (poi acquistata nel 2010 dall'Oracle America Inc.), grazie al lavoro di molti sviluppatori diretti da **James Gosling** per sopperire alla necessità di poter avere un linguaggio di programmazione che fosse platform-indipendent.
Subì molto l'influenza di Simula e C++, da quest'ultimo in particolare ereditò molte caratteristiche come la sintassi che, nello specifico, fu resa molto simile in modo tale da invogliare tutti i programmatori, già aventi familiarità con C++, ad usare Java.
Tuttavia, come anticipato poc'anzi, la sempre più totale espansione del World Wide Web e Internet verso la massa, che aumentò drasticamente la richiesta di programmi indipendenti dalla piattaforma, fu la causa scatenante dell'incredibile successo di Java, rendendolo di fatto uno dei linguaggi di programmazione più famoso e utilizzato nella Storia dell'Informatica, basti pensare che ancora oggi rappresenta una grande fetta del mercato dei linguaggi di programmazione.
Un altro linguaggio che giovò di questa situazione fu **JavaScript**, sviluppato dall'informatico statunitense **Brendan Eich** nello stesso periodo in cui venne rilasciato Java. Contrariamente a quanto si possa pensare a causa della somiglianza dei nomi, i due linguaggi non hanno nessuna correlazione, hanno solo una sintassi simile. Il motivo per il quale JavaScript deve l'origine della sua fama, come viene descritto anche in "Manual Javascript"[18], fu il suo uso nella programmazione delle pagine web, che permise così l'avanzare di siti sempre più dinamici e interattivi.

2.1.7 I linguaggi di programmazione ai giorni nostri

Dopo un viaggio nella Storia e nel tempo partito dalle geniali intuizioni di **Ada Lovelace**, e che ha attraversato le fasi più importanti dell'evoluzione dei linguaggi di programmazione, si giunge ai giorni nostri.
Diversi dei linguaggi creati tra gli anni '80 e '90, grazie ad un continuo lavoro degli sviluppatori volto a migliorare sempre di più rilasciando costantemente nuove versioni, sono ancora tuttora nel 2023, anno in cui viene scritta questa tesi, tra i più usati al mondo come JavaScript, C++ e Java, tuttavia non sono ovviamente gli unici.
Tra tutti svetta **Python**, linguaggio di programmazione creato dall'olandese **Guido Van Rossum** che ha come anno di rilascio il 1991, tuttavia solo ad inizio XXI secolo ha cominciato ad imporsi in tutto il mondo.
Le principali motivazioni che hanno spinto Van Rossum a creare Python vennero esposte tramite una proposta al **DARPA** dal nome "Computer Programming for Everybody"[19].
Come il titolo suggerisce l'obiettivo era quello di avere un linguaggio di programmazione facile ed intuitivo da imparare per chiunque ma che fosse allo stesso tempo efficiente almeno quanto gli altri linguaggi di livello mondiale. Altro aspetto fondamentale per l'ascesa di Python è il suo essere **open source**, che ha permesso a moltissimi sviluppatori di contribuire nella costante evoluzione del linguaggio, migliorandolo versione dopo versione. In aggiunta a questo, Python deve il suo grandissimo successo ai **Big Data**, mole di dati prodotta dagli utenti su Internet aumentata drasticamente negli ultimi anni. Questi dati sono molti importanti perché da essi è possibile ricavare i comportamenti e le abitudini delle persone e, analizzandoli propriamente, possono fornire informazioni preziose per le aziende.
In questo Python si è rivelato un ottimo strumento, come illustrato da Cilien et al[20], per l'amministrazione dei Big Data grazie al vasto assortimento di librerie che consentono ai programmatori di fare un gran numero di operazioni su qualsiasi insieme di dati desiderino.

Negli ultimi periodi Python si sta espandendo sempre più anche nell'ambito dell'**Intelligenza Artificiale**, più specificamente nel settore del **Machine Learning**, infatti come viene detto anche in "Importance of Python programming in Machine Learning"[21], Python è l'ambiente ideale per imparare ad usare modelli di Machine Learning, questo perché permette al programmatore di concentrarsi sulla logica invece di dover dedicare del tempo a comprendere una sintassi ostile all'apprendimento, tramite sempre delle apposite librerie come **TensorFlow** e **Scikit-learn**.
Tutti questi tasselli hanno contribuito a costruire quello che, stando ad un articolo pubblicato da **Forbes**[22], è il più famoso linguaggio di programmazione al mondo attualmente.

CAPITOLO 3

Metodologia

Come il titolo suggerisce, questa tesi si prefigge l'obiettivo di analizzare come i linguaggi di programmazione si siano evoluti lungo l'arco della Storia, andando così a fornire un quadro generale in cui vengono descritti i principali avvenimenti, che hanno appunto scatenato l'evoluzione della programmazione fino ad arrivare ai giorni nostri, dedicando inoltre la doverosa rilevanza ai vari contesti storici, che hanno fatto da sfondo in quegli anni, e ai personaggi che hanno ricoperto un ruolo fondamentale nell'evoluzione dei linguaggi di programmazione con le loro intuizioni rivoluzionarie.
Il fine ultimo è quello di far fare un viaggio nel tempo che sia quanto di più comprensibile e di supporto per qualsiasi lettore, da coloro che sono già nel mondo informatico e vogliono conoscere la Storia che c'è dietro agli strumenti che utilizzano all'ordine del giorno, fino a coloro che invece d'Informatica non sanno nulla e sono mossi da quello che considero essere il più bel tratto dell'umanità, la **curiosità**.

Per prima cosa si è scelto il punto di partenza di questo viaggio nella Storia, ovvero il **1843** con quello che è considerato il primo programma informatico di sempre realizzato da **Ada Lovelace**. Si è deciso senza alcun dubbio di partire proprio da qui, data la grande rilevanza storica che ha avuto il lavoro svolto da Ada Lovelace sull'articolo di **Luigi Federico Manabrea** riguardo la macchina analitica di **Charles Babbage**. Nel fare ciò, oltre a descrivere quanto compiuto da Ada Lovelace, si è dato risalto al contesto storico della Londra del XIX secolo, in cui purtroppo nella maggior parte degli istituti era del tutto vietata la frequenza alle donne. Ponendo attenzione su questo fattore i risultati ottenuti da Ada Lovelace acquisiscono ancora più importanza.
Tale procedimento, utilizzato per descrivere la parte relativa alla nascita dei linguaggi di programmazione, è stato poi applicato in ogni fase toccata dall'analisi evolutiva, ricercando sempre tramite Google Scholar degli articoli che hanno permesso di riportare diverse informazioni utili a questa Tesi come:

1. Le **motivazioni** che hanno portato alla nascita di un certo linguaggio di programmazione;

2. I **personaggi principali** che hanno rivestito un ruolo da protagonisti nella creazione di un certo linguaggio di programmazione;

3. Le **aziende** che hanno investito fondi per consentire lo sviluppo dei linguaggi;

4. Gli **aspetti caratterizzanti** dei linguaggi di programmazione che li differenziano dagli altri, ma anche quelli che invece mostrano una somiglianza tra linguaggi diversi, a testimonianza di una continua **influenza reciproca**;

5. I **fattori** che hanno permesso ad un determinato linguaggio di programmazione di **diffondersi**, chi più chi meno;

6. Il **periodo** in cui un determinato linguaggio è stato sviluppato o rilasciato, evidenziando anche i periodi in cui si è raggiunto **l'apice** del suo utilizzo;

A queste nozioni è stata ricamata sopra una spessa cornice storica, poiché si ritiene fondamentale evidenziare l'impatto che hanno avuto dei noti avvenimenti di importanza storica nel processo evolutivo dei linguaggi di programmazione.
Un esempio lampante lo si trova quando si parla della nascita dei linguaggi di programmazione moderni, infatti il periodo combacia con quello della **Seconda Guerra Mondiale** non per puro caso, ma anzi è strettamente correlato. La presenza di un conflitto mondiale in corso, infatti, ha portato le nazioni più potenti del pianeta ad investire ingenti risorse in diversi settori, tra cui uno di quelli fu proprio l'Informatica con la creazione dei primi **Computer moderni**. Come si può notare questa è definibile come una reazione a catena di eventi, strettamente collegati tra loro, che ha poi portato alla comparsa dei **primi linguaggi di programmazione moderni**.
Un ulteriore esempio che si riporta è l'aumento demografico avvenuto negli Stati Uniti alla fine dell'800 a seguito della massiccia emigrazione transoceanica. L'incremento più che notevole della popolazione complicò di molto le operazioni per eseguire il censimento dei cittadini, facendo poi venire l'idea ad **Herman Hollerith** di automatizzare il procedimento tramite la programmazione delle **schede perforate**.
Questo e molto altro è stato scritto per sottolineare un concetto indispensabile:

La relazione che sussiste tra il processo evolutivo dell'umanità e l'evoluzione dei linguaggi di programmazione è più importante che mai.

3.1 Metodo di ricerca degli articoli

Si andrà ora a mostrare nel dettaglio com'è avvenuto l'intero processo di ricerca e selezione degli articoli presenti in letteratura. Per fare ciò si è condotta una **Systematic Literature Review**[23] (SLR), ovvero un metodo accademico utilizzato per identificare e valutare tutti gli articoli che interessano un certo argomento, traendo delle conclusioni sulla questione che si sta considerando.
Pertanto nel proseguo del capitolo verrà dedicata una sottosezione per ogni passaggio che è stato effettuato per condurre la SLR, prendendo come punto di riferimento quanto fatto in "On the use of artificial intelligence to deal with privacy in IoT systems: A systematic literature review."[24].

3.1.1 Definizione delle domande di ricerca

In questo passaggio iniziale vengono definite ed analizzate le domande di ricerca a cui si vuole rispondere, cosa già fatta nella sottosezione 2.1.1.
Si riportano di nuovo le domande di ricerca per una miglior facilità di lettura.

> *RQ 1: Quali sono le motivazioni che hanno portato a questo cambio di paradigma nel tempo?*
>
> *RQ 2: È possibile creare un conceptual model che permetta agli sviluppatori di selezionare un certo paradigma in base alle sue esigenze?*

3.1.2 Definizione della query di ricerca

Per trovare risposta alle domande di ricerca definite, un passaggio molto importante da svolgere è quello di definire con cura la **query di ricerca**, poiché più è scritta bene più i risultati sono pertinenti ai fini della ricerca.
Una query di ricerca è un insieme di termini rilevanti, dette **parole chiave**, rispetto all'argomento che si vuole ricercare, separati dagli operatori booleani **AND** e **OR**.
Nel caso specifico di questa Tesi, le parole chiave identificate sono le seguenti:

- *"Imperative programming"*
- *"Functional programming"*
- *"Object Oriented programming"*
- *"Event Oriented programming"*
- *"evolution"*
- *"history"*
- *"approach"*
- *"benefits"*
- *"drawback"*

Tali parole chiave sono state individuate proprio per rispondere alle domande di ricerca precedentemente esposte.
La query di ricerca risultante è la seguente:

("Imperative programming" OR "Object Oriented programming" OR "Functional programming" OR "Event Oriented programming") AND ("evolution" OR "history" OR "approach" OR "benefits" OR "drawback")

Avendo scritto in questo modo la query, cioè con i primi tre OR che separano le parole chiave relative ai paradigmi di programmazione presentati nella sottosezione 1.1.3, gli altri quattro OR che separano le restanti parole chiave e l'AND messo tra le due parentesi tonde, si otterranno tutti gli articoli in cui è trattata la storia, i vantaggi, gli svantaggi per ognuno dei 4 paradigmi.

3.1.3 Scelta del database per effettuare le ricerche

Una volta definita la query di ricerca c'è bisogno di darla in input a dei database, che contengano naturalmente un assortimento di articoli di ricerca.
La scelta dei database su cui andare ad effettuare la query è molto importante per poter ottenere i migliori articoli presenti in letteratura, pertanto, sono stati interrogati i seguenti database essendo i più forniti ed utilizzati in ambito accademico:

- **ACM Digital Library**;
- **IEEEXplore**;
- **Scopus**

Dopodiché si è eseguita la query sui database sopra elencati e il numero di articoli ottenuti è stato il seguente: **638** per ACM Digital Library, **613** per IEEEXplore, **2829** per Scopus, raggiungendo un totale di **4080** articoli.
Il numero di articoli ricavati da Scopus sono in numero maggiore poiché si è scelto di effettuare la query specificatamente sulle parole chiave indicate negli articoli, mentre invece per gli altri due database la ricerca è stata eseguita solo sugli abstract.

3.1.4 Criteri di esclusione ed inclusione

Individuati i database sui quali effettuare la query di ricerca, si possono ottenere gli articoli dai quali ricavare le informazioni per rispondere alle nostre domande di ricerca.
In questo frangente però la mole di articoli è ancora troppo grande (siamo sull'ordine delle migliaia), dunque è necessaria una fase di scrematura per poter ridurre il numero degli articoli da selezionare.
Qui entrano in gioco i **criteri di esclusione ed inclusione**, una serie di vincoli che permettono di filtrare solo gli articoli che rispettano determinate condizioni ed escludere quelli che invece rispettano altre caratteristiche.
Nello specifico i criteri di esclusione scelti permettono di non considerare tutti quegli articoli che soddisfano uno o più dei seguenti vincoli:

- Articoli il cui file PDF non è presente online;
- Articoli non aventi abbastanza informazioni;
- Articoli scritti in una lingua diversa dall'italiano o dall'inglese;

Al contrario invece, i criteri di inclusione scelti permettono di considerare tutti quegli articoli che soddisfano tutti i seguenti vincoli:

- Articoli che contengono informazioni sulla Storia di un paradigma di programmazione;
- Articoli che contengono informazioni sull'evoluzione di un paradigma di programmazione;
- Articoli che contengono informazioni sui vantaggi e gli svantaggi di un paradigma di programmazione;

Una volta definiti i criteri di esclusione ed inclusione, vanno applicati agli articoli ottenuti dall'interrogazione iniziale ai database e solo a coloro che rispettano i criteri è consentito avanzare alla prossima fase.

3.1.5 Valutazione della qualità

Applicando i criteri di esclusione ed inclusione esposti nella sottosezione precedente, il numero di articoli selezionati crolla drasticamente dai 4080 iniziali fino a un totale di circa **200**.
Ora bisogna valutare la qualità degli articoli che sono stati prescelti, così da scartare quelli che non raggiungono il grado di qualità desiderato e basare l'analisi solo sugli articoli rimanenti, ovvero quelli che si ritengono in possesso di un grado di qualità accettabile.
Per definire il grado di qualità che si vuole raggiungere sono stati formulati i due seguenti quesiti:

> *Quesito 1: Le informazioni riguardo la Storia e il funzionamento dei paradigmi di programmazione, sono esposte chiaramente?*

> *Quesito 2: Le informazioni riguardo come si è evoluto l'utilizzo di un paradigma di programmazione, sono esposte chiaramente?*

Gli articoli che rispondono in modo affermativo ai quesiti sopra vengono definitivamente selezionati, facendo inevitabilmente scendere ancora il numero fino ad un totale di circa **80**, mentre tutti i restanti si ritengono carenti di qualità e dunque non sono presi in considerazione.

3.1.6 Estrazione delle informazioni

Dopo aver eseguito tutti questi passaggi si arriva finalmente alla fase finale, quella in cui è possibile estrarre le informazioni presenti negli articoli definitivi che sono "sopravvisuti" prima alla scrematura dei criteri di esclusione ed inclusione, e poi alla valutazione della qualità.
Le informazioni che si estraggono da questi articoli sono quelle che permettono di rispondere alle domande di ricerca poste all'inizio e sono presentate nel prossimo capitolo.

CAPITOLO 4

Risultati

Dopo aver condotto la **Systematic Literature Review**, e aver di conseguenza individuato gli articoli d'interesse, si può ora procedere con l'analisi dei risultati ottenuti e osservare com'è possibile rispondere alle domande di ricerca prefissate.
Dunque questo capitolo presenterà due sezioni, in ognuna delle quali si andrà a rispondere ad una domanda di ricerca, citando gli articoli che più si sono rivelati utili ai fini dell'analisi.
Si fa presente che i paradigmi di programmazione che sono stati considerati nell'analisi delle due domande di ricerca sono quelli presentati nella sottosezione 1.1.3, si aggiunge inoltre che per la comodità dei lettori, all'inizio di ogni sezione verrà riportata la dicitura per la rispettiva domanda di ricerca.

Prima di procedere però ci si vuole soffermare un momento sul seguente grafico in Figura 4.1, il quale è stato prodotto tramite un foglio di calcolo **Excel** in cui sono stati riportati uno ad uno tutti gli articoli considerati una volta conclusa la SLR (quindi dopo aver eseguito i criteri di esclusione ed inclusione e la valutazione della qualità) e il relativo paradigma di programmazione di cui trattano.

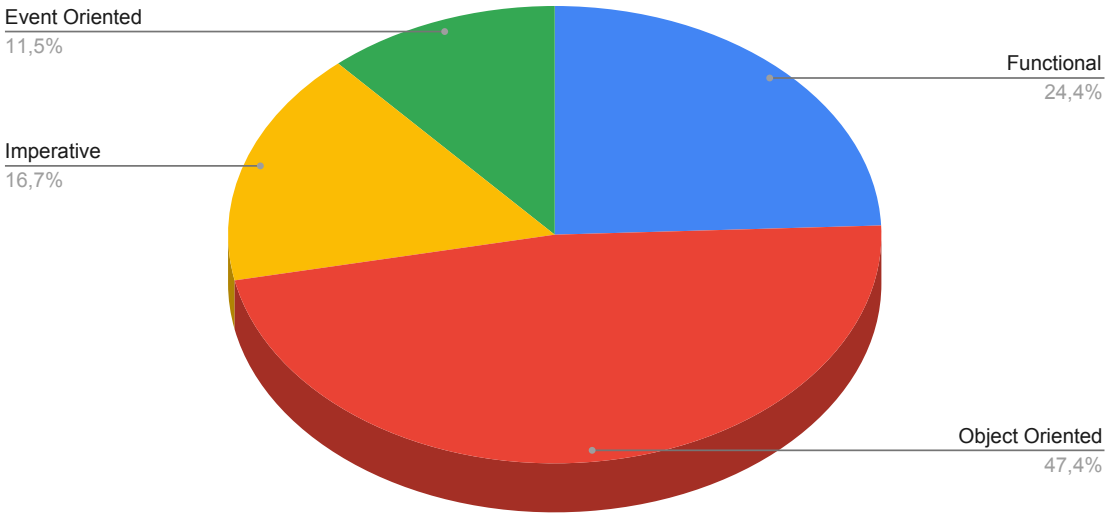

Figura 4.1: Numero di articoli per paradigma.

Il grafico in Figura 4.1 mostra, in percentuale, la schiera di articoli per ognuno dei paradigmi coinvolti ai fini dell'analisi.
Balza immediatamente all'occhio la fetta relativa al paradigma di programmazione Orientata agli Oggetti che, con una somma di articoli che va oltre il **47%**, non fa altro che evidenziare e rafforzare ancora di più quanto questo paradigma di programmazione predomini, sia nel mondo informatico che in quello accademico.
A seguire invece il paradigma di programmazione Funzionale che si assesta intorno al **24%**, scalzando non di moltissimo i paradigmi di programmazione Imperativa **(circa 17%)** ed Event Oriented **(circa 12%)**.

4.1 Risposte alla prima domanda di ricerca

RQ 1: Quali sono le motivazioni che hanno portato a questo cambio di paradigma nel tempo?

Come viene descritto diligentemente da Avacheva e Prutzkov[25], i vari paradigmi di programmazione che si sono susseguiti lungo l'intero arco della Storia Informatica, trovano applicazione quando forniscono al programmatore gli strumenti adatti per poter risolvere un dato problema in maniera efficace ed efficiente.
Tuttavia, nel momento in cui questa efficacia si va a perdere e diventa talmente complesso, se non impossibile, risolvere un problema utilizzando un certo paradigma, si ricerca la soluzione in un nuovo paradigma.
Un altro fattore fondamentale che ha portato i paradigmi ad evolversi costantemente è stato il problema della **duplicazione del codice**.
Il problema della duplicazione del codice consiste in quelle porzioni di codice del tutto identiche, o che differiscono di pochissimo, che sono ripetute più volte all'interno dello stesso programma. Questo è chiaramente qualcosa che si vuole evitare poiché causa notevoli problemi nella manutenzione del codice (ogni qualvolta c'è una modifica da apportare ad una porzione di codice, bisogna ricordarsi di effettuare la stessa identica modifica in ogni parte del programma in cui il codice modificato è duplicato).
Infatti con l'avvicendarsi dei paradigmi di programmazione sono comparse delle nuove tecniche che permettono di ridurre la duplicazione del codice, virando significativamente verso il **riuso** del codice.

In particolare, dagli articoli ottenuti dopo aver condotto la Systematic Literature Review, sono sorte tre principali motivazioni che hanno portato i paradigmi di programmazione a cambiare nel corso del tempo.

Motivazioni	**Descrizione**
Sviluppo tecnologico	A questa motivazione si riferiscono tutti quegli articoli in cui viene messo in evidenza come lo sviluppo tecnologico contribuisca al cambiamento dei paradigmi di programmazione nel tempo.
Incremento qualità	A questa motivazione si riferiscono tutti quegli articoli in cui viene messo in evidenzia come la costante ricerca d'incremento della qualità d'uso dei paradigmi di programmazione contribuisca alla loro stessa evoluzione.
Didattica	A questa motivazione si riferiscono tutti quegli articoli in cui viene messo in evidenzia come la didattica contribuisca a far mutare i paradigmi di programmazione con il passare del tempo.

Tabella 4.1: Motivazioni individuate

Nella Tabella 4.1 si mostra quali sono queste tre motivazioni che sono state individuate e la loro relativa descrizione.
Dopo aver osservato ciò, ogni articolo preso in considerazione è stato ricondotto ed etichettato ad una delle motivazioni sopra descritte, poi sempre tramite il foglio di calcolo Excel è stato prodotto il grafico seguente, in maniera tale da poter avere una visione quantitativa dell'analisi svolta.

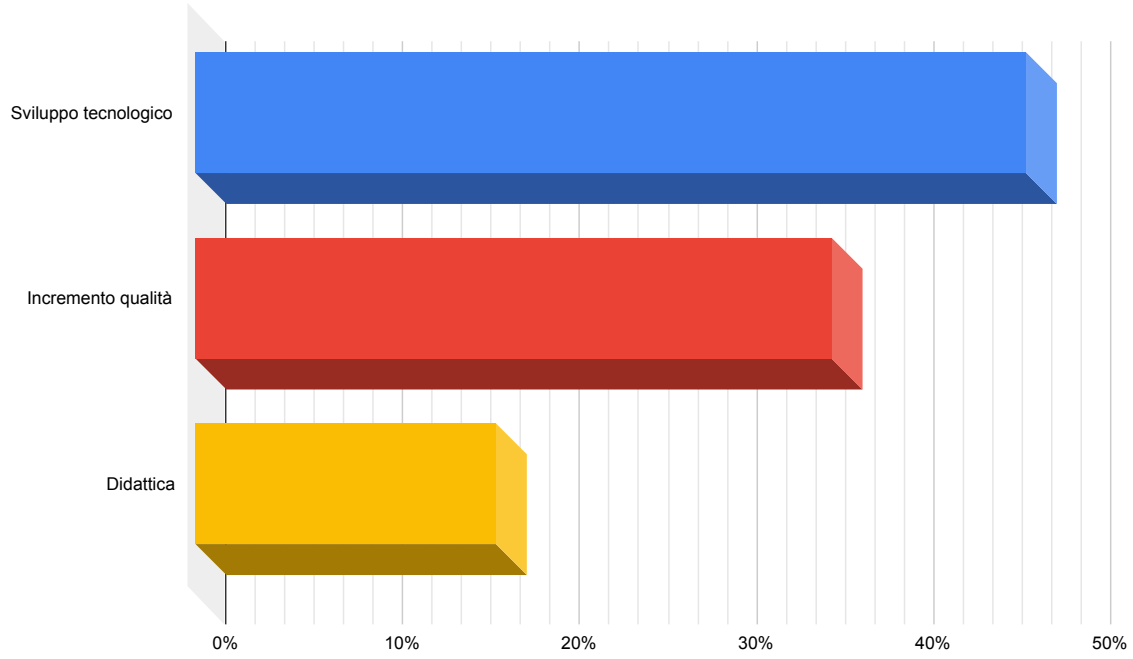

Figura 4.2: Numero di articoli per ciascuna motivazione

Nel grafico in Figura 4.2 è dunque possibile osservare come vengono raggruppati quantitativamente i vari articoli per ognuna delle motivazioni. Da qui si evince la predominanza di articoli (il **47%** del totale) in cui si nota come lo **sviluppo di nuove tecnologie** sia la motivazione che maggiormente porta i paradigmi di programmazione a cambiare nel tempo.

Subito a seguire c'è l'**incremento qualità**, in cui nei vari articoli (**36%**) ci si concentra principalmente sullo studio di nuove tecniche volte ad aumentare la qualità d'uso dei paradigmi di programmazione, il che porta alla successiva modifica e miglioramento dei paradigmi stessi.

Infine c'è la **didattica**, i quali articoli (**17%**) mostrano come i paradigmi di programmazione cambiano col passare del tempo per motivi legati appunto alla didattica, più i paradigmi divengono d'uso comune e più vi è necessità di fornire strumenti che ne supportino l'apprendimento, portando i paradigmi stessi a mutare arricchendosi di nuovi concetti e funzionalità.

Nel proseguo si continuerà l'analisi dei risultati dedicando una sottosezione per ognuno dei quattro paradigmi di programmazione considerati, citando tutti gli articoli ottenuti dalla SLR e usati per produrre i grafici in Figura 4.1 e in Figura 4.2.

4.1.1 Paradigma di programmazione Imperativa

Per quanto riguardo il **paradigma di programmazione Imperativa**, sebbene conferisse il massimo controllo nelle mani del programmatore, il suo utilizzo è andato un po a diminuire, complice il costante progresso tecnologico e il relativo aumento di domanda per lo sviluppo di sistemi software totalmente di un altro livello per poter essere programmati usando soltanto la programmazione Imperativa.
Ciononostante rimane uno dei paradigmi di programmazione più utilizzati e col passare del tempo ci sono stati anche diversi studi molto interessanti (Fonlupt et al.[26], Westbrool et al.[27], Leroy e Weis[28], Avacheva e Prutzkov[25], Pankratius et al.[29], Yeh e Kim[30], Demetrescu et al.[31], Ferrein et al.[32], Igwe e Pillay[33], Giegerich e Steffen[34], Zuhud[35], Chung et al.[36], Ortmeier et al.[37]).
In particolare quello di Igwe e Pillay[33], in cui si è provato ad usare degli **algoritmi genetici** ai quali si da in input un programma scritto seguendo il paradigma di programmazione Imperativa e si applicano poi le varie fasi degli algoritmi genetici (Generazione della popolazione, applicazione della funzione di fitness, selezione, crossover e mutazione) opportunamente progettati.
Iterando questi passaggi più volte si ottiene in output un programma ottimizzato rispetto al programma di partenza dato in input, e che continua ancora ad essere fedele al paradigma di programmazione Imperativa.
In tal modo si è potuto dunque notare che era possibile prendere una soluzione ad un problema scritta usando la programmazione Imperativa e ottenere, grazie agli algoritmi genetici, una soluzione più efficiente per lo stesso problema che continuasse ancora a seguire la programmazione Imperativa.

4.1.2 Paradigma di programmazione Funzionale

Il **paradigma di programmazione Funzionale** invece, stando a quanto scritto da Paul Hudak[38], consente di scrivere i programmi in maniera rapida e concisa, fornisce una notazione che si avvicina molto a quella della matematica tradizionale ed è inoltre facile da eseguire sulle architetture parallele.
D'altro canto questi aspetti sono piuttosto soggettivi, cosa che ha portato a continui dibattiti nel corso del tempo sull'effettivo utilizzo della programmazione Funzionale, consolidando sempre più il suo impiego nel campo scientifico e dell'analisi numerica.
Oggigiorno invece dopo anni di evoluzione e di studi (Galinac e Domazet[39], Ádám e Pataki[40], Phan e Hansen[41], Piñeyro et al.[42], Turner[43], Slodičák et al.[44], Shahzad et al.[45], Kusakabe et al.[46], Hudak[38], Hammond[47], Balik et al.[48], Chakravarty e Keller[49], Pitts[50], Hinze[51], Plasmeijer e Van Eekelen[52], Burn-Thornton[53], Stanchev e Radensky[54], Marino e Succi[55], Lin Y.-C. e Lin F.-C.[56]) la programmazione Funzionale viene usata molto anche nell'ambito dell'**Intelligenza Artificiale**, come descrive ad esempio Piñeyro et al[42], in cui il concetto di sicurezza delle soluzioni che vengono progettate ha acquisito grande attenzione specialmente negli ultimi anni, essendo essa applicata spesso in ambiti molto delicati come la sanità e le auto a guida autonoma. Quando si ha a che fare con applicazioni di questo tipo, un minimo errore può costare la vita di una persona e di conseguenza diviene di fondamentale importanza fornire delle ferree garanzie di sicurezza.
Spesso vengono scelti proprio dei linguaggi che fanno uso del paradigma di programmazione Funzionale, poiché si è notato che oltre a permettere di sviluppare delle soluzioni[1], consentono anche di assicurare un elevato livello di sicurezza.

[1]Nello specifico si parla spesso di soluzioni basate sull'uso del **deep learning** come le reti neurali profonde.

4.1.3 Paradigma di programmazione Orientata agli Oggetti

Relativamente al **paradigma di programmazione Orientata agli Oggetti**, riferendoci a quanto illustrato da Andrew P. Black[57], si cita testualmente quanto scritto da uno dei principali creatori di questo paradigma, **Kristen Nygaard**:

> "Molti dei compiti civili si sono rivelati presentare la stessa tipologia di problemi metodologici: **la necessità di usare la simulazione**."

Fu proprio questo bisogno impellente di astrarre e simulare tramite la programmazione quelli che erano i problemi e le entità del mondo reale che portò poi Nygaard e **Ole-Johan Dahl** alla progettazione del linguaggio di programmazione Simula, il quale ha un legame intrinseco con la programmazione Orientata agli Oggetti perché ha costruito le fondamenta sulle quali poi è stato edificato tale paradigma, infatti alcuni dei concetti presentati con Simula, come la creazione dinamica degli oggetti, sono rimasti invariati nel tempo. Nel frattempo però ci sono stati degli aspetti che sono stati via via consolidati col passare dei decenni, come ad esempio il concetto di **incapsulamento** che ha acquisito sempre più importanza.
Il paradigma di programmazione Orientata agli Oggetti si è evoluto così tanto col passare degli anni trovando crescente impiego in tutta la comunità (Black[57], Muncey[58], Siddique et al.[59], Brito e Medeiros[60], Perelli et al.[61], Zaw et al.[62], Silva e Dorça[63], Larrea et al.[64], Howroyd e Thring[65], Malekan[66], Abel e Faust[67], Jape e Bharadwaj[68], Igwe e Pillay[69], Wong[70], Wang e Haga[71], Rodríguez Corral et al.[72], Mohammedi et al.[73], Ami et al.[74], Chen Yen-Lin et al.[75], Ricci e Santi[76], Schenk et al.[77], Sasso e Biles[78], Pillay e Chalmers[79], Beck et al.[80], Chen C. et al.[81], Oliveira e Silva[82], Ding et al.[83], Mohammad et al.[84], Drummond e Strimmer[85], Zabaras e Srikanth[86], Vernalde et al.[87], Naidu et al.[88], Pandey e Browne[89], Smith et al.[90], Coppens et al.[91], Williams[92], Reed e Chen W.[93]).

Ad esempio esempio si è arrivati alla creazione di tools, come quello presentato da Andrew Muncey[58], che forniscono un supporto agli studenti per quanto riguarda l'apprendimento di concetti relativi alla programmazione Orientata agli Oggetti come la giusta implementazione del costruttore, il corretto uso dell'incapsulamento e via dicendo.
Il tool presentato in questo articolo è scritto in Java e funziona al modo seguente:

- Il docente scrive del codice in Java.
- Dopodiché il docente chiede agli studenti di provare a scrivere quel codice, senza mostrare la soluzione corretta ma fornendo solo una traccia.
- Gli studenti scrivono il codice, il tool provvederà poi in caso di errori a fornire dei feedback. Questi feedback possono essere quelli preimpostati di default dal tool oppure personalizzati dal docente.

Sebbene la sperimentazione di questo particolare tool è ancora in corso, serve a comprendere quanto il paradigma di programmazione Orientata agli Oggetti sia divenuto di uso talmente comune da giungere allo sviluppo di strumenti che hanno come obiettivo primario quello di aiutarne l'apprendimento, qualcosa che probabilmente neanche Nygaard e Dahl si sarebbero aspettati circa 60 anni fa.

4.1.4 Paradigma di programmazione Event-Oriented

Infine per quanto concerne il **paradigma di programmazione Event-Oriented**, come descritto da Conal Elliott[94], il punto focale sta nell'importanza del ruolo che gli **eventi** svolgono nella stragrande maggioranza dei software che si basano sull'interazione con l'utente. Difatti non è un caso che il periodo in cui si diffusero su vasta scala le applicazioni con **interfaccia grafica utente**, gli anni '90, sia lo stesso in cui si è intensificato l'uso di questo paradigma. Premere un pulsante del mouse, della tastiera o cambiare la dimensione di una finestra sono tutti eventi che il paradigma di programmazione Event-Oriented permette di gestire come il programmatore desidera, il tutto senza troppi sforzi proprio perché si adatta bene in questo tipo di applicazioni.
In vari articoli è possibile osservare l'evoluzione di questo paradigma (Elliott[94], Fonseca et al.[95], Lukkarinen et al.[96], Spindlen et al.[97], Bruce e Danyluk[98], Bruce et al.[99], Christensen e Caspersen[100], Gasiunas et al.[101], Russo[102]), come ad esempio sempre nel documento scritto da Conal Elliot[94] viene proposto un nuovo approccio al paradigma di programmazione Event-Oriented, in particolare un approccio di tipo **dichiarativo**.
In seguito agli studi si è riscontrato che effettivamente la programmazione Event-Oriented dichiarativa rende più conveniente la dei moderni sistemi software interattivi, poiché abbassa di molto i costi sia per lo sviluppo che per la manutenzione.
Questo studio è utile per porre l'attenzione su quanto anche il paradigma di programmazione Event-Oriented, come del resto si è potuto osservare con tutti gli altri paradigmi analizzati, non sia rimasto invariato nel tempo ma bensì si è evoluto fino all'introduzione di un nuovo paradigma alternativo all'Event-Oriented chiamato **DEOP**[2].

[2]Declerative Event Oriented Programming.

4.2 Risposte alla seconda domanda di ricerca

RQ 2: È possibile creare un conceptual model che permetta agli sviluppatori di selezionare un certo paradigma in base alle sue esigenze?

Essendo i due quesiti strettamente correlati tra loro, si può rispondere alla seconda domanda di ricerca utilizzando ciò che si è ottenuto in risposta all'interrogativo precedente. In particolare le informazioni estrapolate dai seguenti articoli: *Avacheva e Prutzkov[25], Hudak[38], Black[57], Elliott[94].*

Dopo aver osservato come ognuno dei paradigmi considerati ha degli aspetti caratterizzanti che li differenziano gli uni dagli altri, si ritiene possibile creare il seguente modello concettuale:

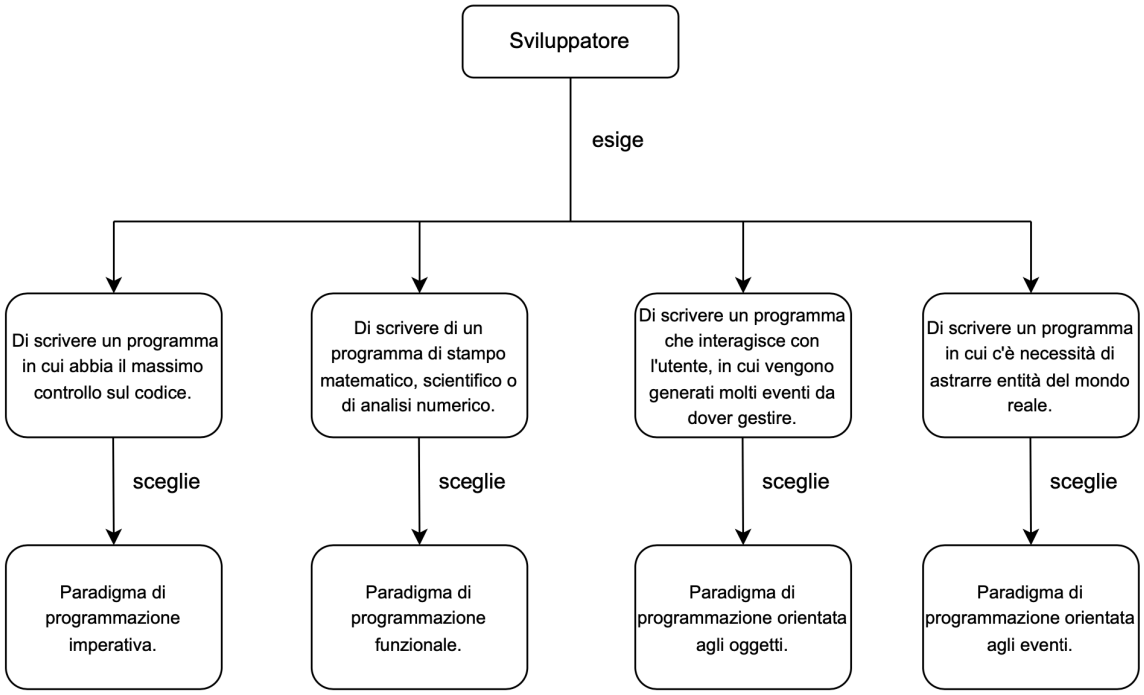

Figura 4.3: Modello concettuale prodotto

La Figura 4.3 mostra il modello concettuale che è stato prodotto in risposta alla seconda domanda di ricerca, difatti opera proprio come richiesto, ovvero indica allo sviluppatore quale paradigma di programmazione si ritiene più adatto da scegliere a seconda della particolare esigenza che si desidera soddisfare.
Le esigenze dello sviluppatore, che sono identificate nei quattro riquadri immediatamente sopra i riquadri raffiguranti i paradigmi di programmazione, sono state selezionate a partire da quanto ricavato nell'analisi dei risultati della prima domanda di ricerca, poiché si sono rilevate essere i concetti che maggiormente permettono di contraddistinguere un paradigma dagli altri.
Naturalmente, come si può notare, il modello concettuale suggerisce solo il paradigma non lo specifico linguaggio di programmazione, anche perché molti linguaggi supportano la possibilità di poter sfruttare più paradigmi e non sono chiusi su uno specifico.
Si pone attenzione anche sul fatto che per dei progetti di grandi dimensioni lo sviluppatore può ritrovarsi a dover usare più paradigmi di programmazione, non necessariamente uno soltanto.
Infine si ritiene che, con ogni probabilità, il modello in Figura 4.3 possa rivelarsi di maggior supporto per quegli sviluppatori che hanno dalla loro parte poca esperienza nella programmazione, pertanto potrebbe essere molto d'aiuto agli studenti, specialmente coloro che si trovano all'inizio del percorso nel mondo della programmazione e dello sviluppo software.

CAPITOLO 5

Conclusioni

5.1 Risultati ottenuti

A fine Tesi è dunque possibile trarre le conclusioni sui risultati ottenuti.
Come prima cosa è stato portato alla luce che la maggior parte dei cambiamenti, che avvengono col passare del tempo in ogni paradigma di programmazione, sono dovuti a tre principali motivazioni.
La prima fra queste è lo **sviluppo di nuove tecnologie**, quello del progresso è infatti un fenomeno costante dell'umanità in ogni ambito, compreso quello tecnologico e informatico, portando di conseguenza a far evolvere anche i paradigmi di programmazione per tenere il passo con le novità.
L'altra motivazione individuata è la continua ricerca di **incrementare la qualità** d'uso dei vari paradigmi, apportando così sempre più modifiche ai paradigmi in modo tale che gli sviluppatori traggano quanto più giovamento possibile.
L'ultima motivazione identificata è quella legata alla parte **didattica** di un paradigma di programmazione, infatti molte volte un paradigma si espande così tanto da rendere necessario il suo insegnamento nelle scuole e università di tutto il mondo, il che spesso fa si che gli stessi paradigmi evolvano per potersi adattare meglio alla didattica.

Si viene a creare così un circolo vizioso tra queste tre motivazioni; lo sviluppo tecnologico, per la sua natura e per quella dell'essere umano, non accenna ad arrestarsi portando così i paradigmi di programmazione ad evolvere nel tempo arricchendosi di nuovi concetti e funzionalità, di conseguenza si cerca di ottimizzare queste novità per far si che la qualità d'uso dei paradigmi stessi aumenti, portando così ad espandere il loro utilizzo fino a rendere necessario fornire un supporto didattico per gli studenti e così via.
Un altro risultato che viene riportato è quello del **modello concettuale** in Figura 4.3, nel corso dell'analisi infatti ci si è resi conto che era perfettamente possibile costruire un modello concettuale che aiutasse gli sviluppatori nella scelta di un determinato paradigma di programmazione quando devono far fronte ad una determinata esigenza.
Si è notato inoltre quanto questo modello, più che agli sviluppatori, possa **assistere gli studenti** che si introducono al mondo della programmazione. Questo perché si suppone che gli sviluppatori conoscano già quanto mostra il modello, mentre invece agli studenti può essere molto più d'aiuto in particolare all'inizio del loro percorso d'apprendimento, fornendo così una solida base nella distinzione e l'uso dei paradigmi di programmazione più utilizzati.

5.2 Sviluppi futuri

Quanto è stato tratto nella Tesi si ritiene più che soddisfacente rispetto agli obiettivi prefissati, lasciando anche spunto per degli sviluppi futuri.
In particolare: si potrebbe pensare di effettuare l'analisi tenendo conto di altri paradigmi di programmazione che non sono stati considerati in questa Tesi; si potrebbe ampliare lo specchio di motivazioni che portano i paradigmi ad evolvere nel tempo; e si potrebbe arricchire il modello concettuale in Figura 4.3 in modo tale da renderlo ancora più accurato e puntuale.

Bibliografia

[1] Treccani. (2023) Treccani linguaggi di programmazione. [Online]. Available: https://www.treccani.it/enciclopedia/linguaggio-di-programmazione/ (Citato a pagina 1)

[2] G. Rossi and T. Zolo, "Fondamenti di programmazione," 2009. (Citato a pagina 2)

[3] M. S. Samuel, "An insight into programming paradigms and their programming languages," *Journal of Applied Technology and Innovation*, vol. 1, no. 1, pp. 37–57, 2017. (Citato a pagina 3)

[4] S. Hénin, "Augusta ada lovelace (1815-1852)," *Mondo Digitale*, vol. 57, 2015. (Citato a pagina 8)

[5] A. E. Hurt, *Ada Lovelace: Computer Programmer and Mathematician*. Cavendish Square Publishing, LLC, 2017. (Citato a pagina 9)

[6] S. Hénin, "Buon compleanno mr. hollerith." (Citato a pagina 10)

[7] F. Venuda, "Le biblioteche ei primi sistemi elettromeccanici di gestione e controllo delle informazioni bibliografiche," *Le biblioteche ei primi sistemi elettromeccanici di gestione e controllo delle informazioni bibliografiche*, pp. 189–228, 2010. (Citato a pagina 10)

[8] K. Zuse and S. di Heinz Rutishauser, "Storia della programmazione." (Citato a pagina 11)

[9] G. Succi, "L'evoluzione dei linguaggi di programmazione: analisi e prospettive," *Mondo Digitale*, vol. 4, 2003. (Citato a pagina 11)

[10] F. Flacco, A. Melgrati, A. Nastri, and F. Varanini, "Scenari attuali e competenze emergenti: le imprese italiane e l'ict," 2005. (Citato a pagina 13)

[11] F. D'Cruze, "Pascal programming language," 2011. (Citato a pagina 13)

[12] A. Avdic, "Evoluzione dei linguaggi di programmazione educativi: Scratch e byob," 2011. (Citato a pagina 13)

[13] I. UNIT, "History of c," 2008. (Citato a pagina 14)

[14] M. Stefik and D. G. Bobrow, "Object-oriented programming: Themes and variations," *AI magazine*, vol. 6, no. 4, pp. 40–40, 1985. (Citato a pagina 15)

[15] K. Nygaard, "Basic concepts in object oriented programming," in *Proceedings of the 1986 SIGPLAN Workshop on Object-oriented Programming*, 1986, pp. 128–132. (Citato a pagina 15)

[16] M. Baldoni, G. Boella, and L. van der Torre, "I fondamenti ontologici dei linguaggi di programmazione orientati agli oggetti: i casi delle relazioni e dei ruoli," *Networks*, vol. 6, pp. 79–89, 2006. (Citato a pagina 15)

[17] B. Folio, "The history and evolution of java." (Citato a pagina 16)

[18] C. JavaScript, "Manual javascript," 2012. (Citato a pagina 16)

[19] G. Van Rossum *et al.*, "Computer programming for everybody," *Proposal to the Corporation for National Research initiatives*, 1999. (Citato a pagina 17)

[20] D. Cielen and A. Meysman, *Introducing data science: big data, machine learning, and more, using Python tools*. Simon and Schuster, 2016. (Citato a pagina 17)

[21] S. Sultonov, "Importance of python programming language in machine learning." *International Bulletin of Engineering and Technology*, vol. 3, no. 9, pp. 28–30, 2023. (Citato a pagina 18)

[22] Forbes. (2022) Forbes the top programming languages of 2023. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2022/12/28/what-your-software-partner-should-know-the-top-programming-languages-of-2023/ (Citato a pagina 18)

[23] T. U. Berlin. (2022) Technische Universitat Berlin what is a systematic literature review? [Online]. Available: https://www.tu.berlin/en/wm/bibliothek/research-teaching/systematic-literature-reviews/description-of-the-systematic-literature-review-method (Citato a pagina 21)

[24] G. Giordano, F. Palomba, and F. Ferrucci, "On the use of artificial intelligence to deal with privacy in iot systems: A systematic literature review," *Journal of Systems and Software*, vol. 193, p. 111475, 2022. (Citato a pagina 21)

[25] T. Avacheva and A. Prutzkow, "The evolution of imperative programming paradigms as a search for new ways to reduce code duplication," vol. 714, no. 1, p. 012001, jan 2020. [Online]. Available: https://dx.doi.org/10.1088/1757-899X/714/1/012001 (Citato alle pagine 28, 31 e 36)

[26] C. Fonlupt, D. Robilliard, and V. Marion-Poty, "Linear imperative programming with differential evolution," in *2011 IEEE Symposium on Differential Evolution (SDE)*, 2011, pp. 1–8. (Citato a pagina 31)

[27] E. Westbrook, A. Stump, and I. Wehrman, "A language-based approach to functionally correct imperative programming," *SIGPLAN Not.*, vol. 40, no. 9, p. 268–279, sep 2005. [Online]. Available: https://doi.org/10.1145/1090189.1086400 (Citato a pagina 31)

[28] X. Leroy and P. Weis, "Polymorphic type inference and assignment," in *Proceedings of the 18th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '91. New York, NY, USA: Association for Computing Machinery, 1991, p. 291–302. [Online]. Available: https://doi.org/10.1145/99583.99622 (Citato a pagina 31)

[29] V. Pankratius, F. Schmidt, and G. Garretón, "Combining functional and imperative programming for multicore software: An empirical study evaluating scala

and java," in *2012 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 123–133. (Citato a pagina 31)

[30] T. Yeh and J. Kim, "Craftml: 3d modeling is web programming," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, ser. CHI '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1–12. [Online]. Available: https://doi.org/10.1145/3173574.3174101 (Citato a pagina 31)

[31] C. Demetrescu, I. Finocchi, and A. Ribichini, "Reactive imperative programming with dataflow constraints," *SIGPLAN Not.*, vol. 46, no. 10, p. 407–426, oct 2011. [Online]. Available: https://doi.org/10.1145/2076021.2048100 (Citato a pagina 31)

[32] A. Ferrein, G. Steinbauer, and S. Vassos, "Action-based imperative programming with yagi," vol. WS-12-06, 2012, Conference paper, p. 24 – 31, cited by: 8. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84875605740&partnerID=40&md5=8602743f052f7f659fcaf815b63b9424 (Citato a pagina 31)

[33] K. Igwe and N. Pillay, "Automatic programming using genetic programming," in *2013 Third World Congress on Information and Communication Technologies (WICT 2013)*, 2013, pp. 337–342. (Citato a pagina 31)

[34] R. Giegerich and P. Steffen, "Implementing algebraic dynamic programming in the functional and the imperative programming paradigm," in *Mathematics of Program Construction*, E. A. Boiten and B. Möller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 1–20. (Citato a pagina 31)

[35] D. A. Z. Zuhud, "From programming sequential machines to parallel smart mobile devices: Bringing back the imperative paradigm to today's perspective," in *2013 8th International Conference on Information Technology in Asia (CITA)*, 2013, pp. 1–7. (Citato a pagina 31)

[36] M. J.-Y. Chung, M. Nakura, S. H. Neti, A. Lu, E. Hummel, and M. Cakmak, "Concodeit! a comparison of concurrency interfaces in block-based visual robot

programming," in *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, 2020, pp. 245–252. (Citato a pagina 31)

[37] F. Ortmeier, S. Struck, J. Meinicke, and J. Quante, "A pragmatic approach for debugging parameter-driven software," vol. P-213, 2013, Conference paper, p. 199 – 212, cited by: 0. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84922712095&partnerID=40&md5=3acbc4b8f1f22bfc26f4863b7f14b962 (Citato a pagina 31)

[38] P. Hudak, "Conception, evolution, and application of functional programming languages," *ACM Comput. Surv.*, vol. 21, no. 3, p. 359–411, sep 1989. [Online]. Available: https://doi.org/10.1145/72551.72554 (Citato alle pagine 32 e 36)

[39] T. Galinac Grbac and N. Domazet, "The role of functional programming in management and orchestration of virtualized network resources," in *Composability, Comprehensibility and Correctness of Working Software*, Z. Porkoláb and V. Zsók, Eds. Cham: Springer International Publishing, 2023, pp. 136–164. (Citato a pagina 32)

[40] A. Révész and N. Pataki, "Lambdakube - a functional programming approach in a distributed realm," in *Proceedings of the 2021 4th International Conference on Geoinformatics and Data Analysis*, ser. ICGDA '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 67–72. [Online]. Available: https://doi.org/10.1145/3465222.3465233

[41] A.-D. Phan and M. R. Hansen, "An approach to multicore parallelism using functional programming: A case study based on presburger arithmetic," *Journal of Logical and Algebraic Methods in Programming*, vol. 84, no. 1, pp. 2–18, 2015, special Issue: The 23rd Nordic Workshop on Programming Theory (NWPT 2011) Special Issue: Domains X, International workshop on Domain Theory and applications, Swansea, 5-7 September, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352220814000480 (Citato a pagina 32)

[42] L. Piñeyro, A. Pardo, and M. Viera, "Structure verification of deep neural networks at compilation time," *Journal of Computer Languages*, vol. 67, p. 101074, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2590118421000538 (Citato a pagina 32)

[43] D. A. Turner, "Some history of functional programming languages," in *Trends in Functional Programming*, H.-W. Loidl and R. Peña, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–20. (Citato a pagina 32)

[44] V. Slodičák, P. Macko, and V. Novitzká, *Some New Approaches in Functional Programming Based on Categories*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 517–532. [Online]. Available: https://doi.org/10.1007/978-3-642-32096-5_11 (Citato a pagina 32)

[45] S. K. Shahzad, M. Granitzer, and K. Tochterman, "Designing user interfaces through ontological user model: Functional programming approach," in *2009 Fourth International Conference on Computer Sciences and Convergence Information Technology*, 2009, pp. 99–104. (Citato a pagina 32)

[46] S. Kusakabe, Y. Ohmori, and K. Araki, "Leveraging light-weight formal methods with functional programming approach on cloud," vol. 1, 2009, Conference paper, p. 264 – 268, cited by: 0. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-74549128125&partnerID=40&md5=945c9cf5fd9e50b0d5bce645d1ea32da (Citato a pagina 32)

[47] K. Hammond, "Exploiting purely functional programming to obtain bounded resource behaviour: The hume approach," in *Central European Functional Programming School*, Z. Horváth, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 100–134. (Citato a pagina 32)

[48] H. H. Balik, B. Sevinc, and A. Akbal, "A new approach to spectral domain method: Functional programming," in *Computational Science – ICCS 2006*, V. N. Alexandrov, G. D. van Albada, P. M. A. Sloot, and J. Dongarra, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 638–644. (Citato a pagina 32)

[49] M. M. T. CHAKRAVARTY and G. KELLER, "The risks and benefits of teaching purely functional programming in first year," *Journal of Functional Programming*, vol. 14, no. 1, p. 113–123, 2004. (Citato a pagina 32)

[50] A. M. Pitts, "A fresh approach to representing syntax with static binders in functional programming," *SIGPLAN Not.*, vol. 36, no. 10, oct 2001. [Online]. Available: https://doi.org/10.1145/507669.507637 (Citato a pagina 32)

[51] R. Hinze, "New approach to generic functional programming," 2000, Conference paper, p. 119 – 132, cited by: 93. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0033722168&partnerID=40&md5=5326637b6ab57cefacdda4506493258c (Citato a pagina 32)

[52] R. Plasmeijer and M. v. Eekelen, "Keep it clean: a unique approach to functional programming." *ACM Sigplan Notices*, vol. 34, no. 6, pp. 23–31, 1999. (Citato a pagina 32)

[53] K. Burn-Thornton, "Mining the organic compound jungle - a functional programming approach," *IEE Colloquium (Digest)*, no. 434, p. 8/1–8/4, 1998, cited by: 0. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0031620536&partnerID=40&md5=fa984159811159a1c71712b5e37c89ac (Citato a pagina 32)

[54] S. Stanchev and A. Radensky, "Teaching some modern functional programming concepts: An approach based on an extended fp-like language," *SIGCSE Bull.*, vol. 23, no. 4, p. 31–36, nov 1991. [Online]. Available: https://doi.org/10.1145/122697.122702 (Citato a pagina 32)

[55] G. Marino and G. Succi, "A new approach to parallel functional programming," in *Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing*, 1991, pp. 95–102. (Citato a pagina 32)

[56] Y. Lin and F. Lin, "A functional programming approach to systolic design," *Journal of the Chinese Institute of Engineers*, vol. 11, no. 6, pp. 681–691, 1988. [Online]. Available: https://doi.org/10.1080/02533839.1988.9677120 (Citato a pagina 32)

[57] A. P. Black, "Object-oriented programming: Some history, and challenges for the next fifty years," *Information and Computation*, vol. 231, pp. 3–20, 2013, fundamentals of Computation Theory. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0890540113000795 (Citato alle pagine 33 e 36)

[58] A. Muncey, "Towards automated testing and feedback of object-oriented programming tasks in java," ser. UKICER '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3610969.3611129 (Citato alle pagine 33 e 34)

[59] S. Siddique, A.-A. I. Hridoy, S. A. Khushbu, and A. K. Das, "Cvd: An improved approach of software vulnerability detection for object oriented programming languages using deep learning," in *Proceedings of the Future Technologies Conference (FTC) 2022, Volume 1*, K. Arai, Ed. Cham: Springer International Publishing, 2023, pp. 145–164. (Citato a pagina 33)

[60] J. Agostinho de Medeiros Brito and A. A. D. de Medeiros, "A motivating approach to introduce object-oriented programming to engineering students," *International Journal of Electrical Engineering & Education*, vol. 59, no. 4, pp. 366–386, 2022. [Online]. Available: https://doi.org/10.1177/0020720919856247 (Citato a pagina 33)

[61] M. Perrelli, F. Cosco, G. Carbone, B. Lenzo, and D. Mundo, "On the benefits of using object-oriented programming for the objective evaluation of vehicle dynamic performance in concurrent simulations," *Machines*, vol. 9, no. 2, 2021. [Online]. Available: https://www.mdpi.com/2075-1702/9/2/41 (Citato a pagina 33)

[62] K. K. Zaw, W. Zaw, N. Funabiki, and W.-C. Kao, "An informative test code approach in code writing problem for three object-oriented programming concepts in java programming learning assistant system," *IAENG International Journal of Computer Science*, vol. 46, no. 3, p. 1 – 9, 2019, cited by: 4. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85077147724&partnerID=40&md5=5f4156dcb001ac855cbb33bd476b3c25 (Citato a pagina 33)

[63] V. J. Silva and F. A. Dorça, "An automatic and intelligent approach for supporting teaching and learning of software engineering considering design smells in object-oriented programming," in *2019 IEEE 19th International Conference on Advanced Learning Technologies (ICALT)*, vol. 2161-377X, 2019, pp. 321–323. (Citato a pagina 33)

[64] M. L. Larrea, J. I. R. Silva, M. N. Selzer, and D. K. Urribarri, "White-box testing framework for object-oriented programming. an approach based on message sequence specification and aspect oriented programming," in *Computer Science – CACIC 2018*, P. Pesado and C. Aciti, Eds. Cham: Springer International Publishing, 2019, pp. 143–156. (Citato a pagina 33)

[65] S. Howroyd and R. Thring, "An electric vehicle model and validation using a nissan leaf: A python-based object-oriented programming approach," *Advances in Mechanical Engineering*, vol. 10, no. 7, p. 1687814018782099, 2018. [Online]. Available: https://doi.org/10.1177/1687814018782099 (Citato a pagina 33)

[66] M. Malekan, L. L. Silva, F. B. Barros, R. L. Pitangueira, and S. S. Penna, "Two-dimensional fracture modeling with the generalized/extended finite element method: An object-oriented programming approach," *Advances in Engineering Software*, vol. 115, pp. 168–193, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0965997817304829 (Citato a pagina 33)

[67] K. C. Abel and K. M. Faust, "Modeling food desert disruptors: An object oriented programming approach," vol. 2, 2017, Conference paper, p. 1040 – 1049, cited by: 2. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85048629080&partnerID=40&md5=50066359d998ffb64a29763bc63f0a4e (Citato a pagina 33)

[68] V. S. Jape and D. G. Bharadwaj, "A new approach to object-oriented programming language for remote power quality monitoring and analysis considering harmonics," in *Artificial Intelligence and Evolutionary Computations in Engineering Systems*, S. S. Dash, M. A. Bhaskar, B. K. Panigrahi, and S. Das, Eds. New Delhi: Springer India, 2016, pp. 323–331. (Citato a pagina 33)

[69] K. Igwe and N. Pillay, "A study of genetic programming and grammatical evolution for automatic object-oriented programming: A focus on the list data structure," in *Advances in Nature and Biologically Inspired Computing*, N. Pillay, A. P. Engelbrecht, A. Abraham, M. C. du Plessis, V. Snášel, and A. K. Muda, Eds. Cham: Springer International Publishing, 2016, pp. 151–163. (Citato a pagina 33)

[70] Y. S. Wong, M. H. B. Mohammad Yatim, and W. H. Tan, "Learning object-oriented programming with computer games: A game-based learning approach," vol. 2015-January, 2015, Conference paper, p. 729 – 737, cited by: 2. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84955130279&partnerID=40&md5=0d7d4da26d9d49d9f45ab8c4f7150bf4 (Citato a pagina 33)

[71] Q. Wang and H. Haga, "A novel approach to the design and implementation of mutation operators for object-oriented programming language," in *2014 4th World Congress on Information and Communication Technologies (WICT 2014)*, 2014, pp. 102–106. (Citato a pagina 33)

[72] J. M. Rodríguez Corral, A. Civit Balcells, A. Morgado Estévez, G. Jiménez Moreno, and M. J. Ferreiro Ramos, "A game-based approach to the teaching of object-oriented programming languages," *Computers Education*, vol. 73, pp. 83–92, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360131513003370 (Citato a pagina 33)

[73] R. M. A. H. S. Arif, "An object-oriented programming approach to security-constrained unit commitment problem." (Citato a pagina 33)

[74] A. S. Ami and M. S. Islam, "An efficient approach for providing rationale of method change for object oriented programming," in *2014 International Conference on Informatics, Electronics Vision (ICIEV)*, 2014, pp. 1–6. (Citato a pagina 33)

[75] Y.-L. Chen, C.-M. Liu, C.-Y. Chiang, S.-M. Yuan, and J.-H. Wang, "Building communication software: A project-based approach for teaching c++

object-oriented programming," *International Journal of Innovative Computing, Information and Control*, vol. 9, no. 8, p. 3415 – 3436, 2013, cited by: 1. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84880085533&partnerID=40&md5=e6e88fc6a2c81205c7f3eac7269e79e0 (Citato a pagina 33)

[76] A. Ricci and A. Santi, "Concurrent object-oriented programming with agent-oriented abstractions: The aloo approach," in *Proceedings of the 2013 Workshop on Programming Based on Actors, Agents, and Decentralized Control*, ser. AGERE! 2013. New York, NY, USA: Association for Computing Machinery, 2013, p. 127–138. [Online]. Available: https://doi.org/10.1145/2541329.2541333 (Citato a pagina 33)

[77] J. Schenk, T. Löhning, and U. Starossek, "An object-oriented programming approach for the analysis of spatial reinforced concrete frames," 2009, Conference paper, cited by: 2. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84858419871&partnerID=40&md5=ba1b7346447e72ca9a3b805610af590c (Citato a pagina 33)

[78] D. Sasso and W. E. Biles, "An object-oriented programming approach for a gis data-driven simulation model of traffic on an inland waterway," in *2008 Winter Simulation Conference*, 2008, pp. 2590–2594. (Citato a pagina 33)

[79] N. Pillay and C. K. A. Chalmers, "A hybrid approach to automatic programming for the object-oriented programming paradigm," in *Proceedings of the 2007 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries*, ser. SAICSIT '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 116–124. [Online]. Available: https://doi.org/10.1145/1292491.1292505 (Citato a pagina 33)

[80] D. Beck, H. Brand, C. Karagiannis, and C. Rauth, "The first approach to object oriented programming for labview real-time targets," *IEEE Transactions on Nuclear Science*, vol. 53, no. 3, pp. 930–935, 2006. (Citato a pagina 33)

[81] C. Chen, R. Shi, and H. Xi, "A typeful approach to object-oriented programming with multiple inheritance," in *Practical Aspects of Declarative Languages*, B. Jayaraman, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 23–38. (Citato a pagina 33)

[82] Miguel Oliveira e Silva, "Concurrent object-oriented programming: The mp-eiffel approach," *Journal of Object Technology*, vol. 3, no. 4, pp. 97–124, Apr. 2004, proceedings of the TOOLS USA 2003 Conference, 30 September - 01 October 2003 — Santa Monica, CA. [Online]. Available: http://www.jot.fm/contents/issue_2004_04/article6.html (Citato a pagina 33)

[83] G. Ding, C. Zhang, J. Liu, W. Wei, M. Nakayama, M. Fukaya, G. Oh, and T. Inagaki, "Development of simulation software for evaporator with object oriented programming approach," 2003, Conference paper, p. 528 – 531, cited by: 1. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-4444235497&partnerID=40&md5=69a53c7ca130d4dc15ff544e1a9b8871 (Citato a pagina 33)

[84] S. Mohammad, A. Sureka, D. Ely, and J. M. Jenkins, "A signals and systems and object oriented programming approach to development of ecg analysis software," vol. 29, 2002, Conference paper, p. 153 – 156, cited by: 0. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0036950980&partnerID=40&md5=714cdf8ffdcc963f655294a5f96fe43f (Citato a pagina 33)

[85] A. Drummond and K. Strimmer, "PAL: an object-oriented programming library for molecular evolution and phylogenetics ," *Bioinformatics*, vol. 17, no. 7, pp. 662–663, 07 2001. [Online]. Available: https://doi.org/10.1093/bioinformatics/17.7.662 (Citato a pagina 33)

[86] N. Zabaras and A. Srikanth, "An object-oriented programming approach to the lagrangian fem analysis of large inelastic deformations and metal-forming processes," *International Journal for Numerical Methods in Engineering*, vol. 45, no. 4, pp. 399–445, 1999. (Citato a pagina 33)

[87] S. Vernalde, P. Schaumont, and I. Bolsens, "An object oriented programming approach for hardware design," in *Proceedings. IEEE Computer Society Workshop on VLSI '99. System Design: Towards System-on-a-Chip Paradigm*, 1999, pp. 68–73. (Citato a pagina 33)

[88] B. N. Naidu, R. Srinivasan, and S. M. Shankar, "Object-oriented programming approach to CCD data acquisition and image processing," in *EUV, X-Ray, and Gamma-Ray Instrumentation for Astronomy VIII*, O. H. W. Siegmund and M. A. Gummin, Eds., vol. 3114, International Society for Optics and Photonics. SPIE, 1997, pp. 260 – 269. [Online]. Available: https://doi.org/10.1117/12.278893 (Citato a pagina 33)

[89] R. Pandey and J. Browne, "Compositional approach to concurrent object-oriented programming," 1994, Conference paper, p. 124 – 135, cited by: 3. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0028251782&partnerID=40&md5=b77d20e62427ede1518ac32adc2b93eb (Citato a pagina 33)

[90] S. Smith, T. Levante, B. Meier, and R. Ernst, "Computer simulations in magnetic resonance. an object-oriented programming approach," *Journal of Magnetic Resonance, Series A*, vol. 106, no. 1, pp. 75–105, 1994. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1064185884710084 (Citato a pagina 33)

[91] A. Coppens, M. Sibomana, A. Bol, and C. Michel, "Mediman: an object oriented programming approach for medical image analysis," *IEEE Transactions on Nuclear Science*, vol. 40, no. 4, pp. 950–955, 1993. (Citato a pagina 33)

[92] T. Williams, "Object-oriented programming eases problems caused by program size and complexity, and its benefits are coming to real-time applications," *Electronic Systems Technology and Design/Computer Design's*, vol. 30, no. 8, 1991, cited by: 0. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0026155393&partnerID=40&md5=c51e5389596f4aae9b1423f1ba81bd72 (Citato a pagina 33)

[93] D. Reed and W. Chen, "An object-oriented programming approach to safety assessment," *Engineering Structures*, vol. 13, no. 4, pp. 351–356, 1991. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0141029691900214 (Citato a pagina 33)

[94] C. Elliott, "Declarative event-oriented programming," in *Proceedings of the 2nd ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, ser. PPDP '00. New York, NY, USA: Association for Computing Machinery, 2000, p. 56–67. [Online]. Available: https://doi.org/10.1145/351268.351276 (Citato alle pagine 35 e 36)

[95] A. Fonseca, J. Rafael, and B. Cabral, "Eve: A parallel event-driven programming language," in *Euro-Par 2014: Parallel Processing Workshops*, L. Lopes, J. Žilinskas, A. Costan, R. G. Cascella, G. Kecskemeti, E. Jeannot, M. Cannataro, L. Ricci, S. Benkner, S. Petit, V. Scarano, J. Gracia, S. Hunold, S. L. Scott, S. Lankes, C. Lengauer, J. Carretero, J. Breitbart, and M. Alexander, Eds. Cham: Springer International Publishing, 2014, pp. 170–181. (Citato a pagina 35)

[96] A. Lukkarinen, L. Malmi, and L. Haaranen, "Event-driven programming in programming education: A mapping review," *ACM Trans. Comput. Educ.*, vol. 21, no. 1, mar 2021. [Online]. Available: https://doi.org/10.1145/3423956 (Citato a pagina 35)

[97] J. Spidlen, P. Hanzlicek, and J. Zvarova, "Mudrlite-health record tailored to your particular needs," *Studies in Health Technology and Informatics*, vol. 105, p. 202 – 209, 2004, cited by: 6. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-14544290565&partnerID=40&md5=ec831f99d9c2ac56f82c9594743547ed (Citato a pagina 35)

[98] K. B. Bruce and A. Danyluk, "Event-driven programming facilitates learning standard programming concepts," in *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, 2004, pp. 96–100. (Citato a pagina 35)

[99] K. B. Bruce, A. P. Danyluk, and T. P. Murtagh, "Event-driven programming is simple enough for cs1," *ACM SIGCSE Bulletin*, vol. 33, no. 3, pp. 1–4, 2001. (Citato a pagina 35)

[100] H. B. Christensen and M. E. Caspersen, "Frameworks in cs1: a different way of introducing event-driven programming," in *Proceedings of the 7th annual conference on Innovation and technology in computer science education*, 2002, pp. 75–79. (Citato a pagina 35)

[101] V. Gasiunas, L. Satabin, M. Mezini, A. Núñez, and J. Noyé, "Escala: modular event-driven object interactions in scala," in *Proceedings of the tenth international conference on Aspect-oriented software development*, 2011, pp. 227–240. (Citato a pagina 35)

[102] M. F. Russo, "Doodlepad: next-gen event-driven programming for cs1," *Journal of Computing Sciences in Colleges*, vol. 32, no. 4, pp. 99–105, 2017. (Citato a pagina 35)

Ringraziamenti

Io mi reputo fortunato perché la mia vita mi ha permesso di nascere e crescere in una famiglia meravigliosa. Mi hanno sempre circondato di tanto amore e non riuscirò mai a ringraziarli abbastanza per tutti i sacrifici che hanno fatto per me e per tutti noi.

Io mi reputo fortunato perché la mia vita mi ha permesso di conoscere delle persone che mi limito a chiamare "amici" ma definirli in maniera così riduttiva è un oltraggio. Voglio un mondo di bene a tutti loro, da chi c'è da sempre a chi ho conosciuto col passare del tempo quasi per caso fino ad arrivare a coloro che ho incontrato in questi anni universitari, rendete la mia anima più leggera quando sto con voi.

Io mi reputo fortunato perché la mia vita mi ha permesso di studiare e incontrare sul mio cammino dei docenti che sono state figure molto importanti per la mia crescita personale, dalle elementari fino all'Università in cui, tra i tanti, mi soffermo sul Professore Fabio Palomba e il Dottor Giammaria Giordano che mi hanno supportato lungo tutto il percorso per la procedura di Laurea e di scrittura di questa Tesi.

Io mi reputo fortunato perché la mia vita mi da la possibilità di coltivare tutte le mie passioni come il cinema, lo sport e la musica che contribuiscono a colorare la mia intera esistenza.

Io mi reputo fortunato perché in vita mia non ho mai subito discriminazioni verso il mio genere, l'orientamento sessuale o il colore della mia pelle.

Io mi reputo fortunato perché non ho la minima idea di cosa significhi avere una guerra nel proprio paese.

Io mi reputo fortunato per tutto questo, perché purtroppo ci sono persone in tutto il mondo che non hanno la fortuna di nascere in una famiglia che ti sta accanto; che non hanno la fortuna di sentirsi costantemente circondati dal calore degli amici; che non hanno la fortuna di poter scegliere di studiare; che non hanno la fortuna di poter coltivare giorno per giorno le proprie passioni; che non hanno la fortuna di non aver mai subito degli atti d'odio nei propri confronti; che non hanno la fortuna di potersi addormentare senza ascoltare il frastuono delle bombe che esplodono nelle vicinanze.
Il mio pensiero va a tutti loro.