UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

RELAZIONE TRA QUALITÀ DEL SOFTWARE IN TERMINE DI PRESENZA DI DIFETTI E RIUSO DEL SOFTWARE IN TERMINI DI EREDITARIETÀ DI SPECIFICA, DI IMPLEMENTAZIONE E DELEGAZIONE DURANTE L'EVOLUZIONE

RELATORE

Prof. **Gravino Carmine**

Prof. **Palomba Fabio**

Dott. **Giordano Giammaria**

Università degli studi di Salerno

CANDIDATO

Antonio Gambale

Matricola: 0512105346

Anno Accademico 2020-2021

Ai miei genitori, ancora della mia vita,

A mia sorella luce dei miei occhi,

A Raissa compagna, amica e confidente.

Indice

Abstract

L'ereditarietà è un aspetto fondamentale nei linguaggi di programmazione Object Oriented, la quale può portare molteplici benefici, come modularità e riusabilità del software ma anche diverse insidie come l'aumento della complessità. Uno dei vantaggi offerti dall'ereditarietà è la possibilità di riutilizzare il codice. Il riuso ricopre un ruolo fondamentale nei sistemi software capace di diminuire i costi di sviluppo, di testing e manutenzione. Il concetto di qualità del software in presenza di difetti è un aspetto fondamentale per la realizzazione di un prodotto che sia valido e funzionale. Ai fini della qualità, è più importante eliminare il difetto che non determinare l'errore che lo ha generato. Inoltre una scarsa qualità del software da parte delle aziende costituisce una perdita d'immagine e di leadership sul mercato mancando di affidabilità. L'obiettivo di questa tesi è quello di analizzare la relazione tra qualità del software in termine di presenza di difetti e riuso del software in termini di ereditarietà di specifica, di implementazione e delegazione durante l'evoluzione. In primo luogo le informazioni relative al riuso verranno recuperate dallo strumento InhMetrics capace di determinare le informazioni relative alle metrichecdi riuso utilizzate mentre per la valutazione della presenza di difetti verranno considerate le informazioni su un insieme di sistemi software Java gestiti da Defects4J ovvero un database e un framework estendibile in grado di consentire studi sul testing del software. In secondo luogo la relazione tra i difetti e il riuso sarà analizzata utilizzando plot e test statistici, i.e., Wilcoxon Mann Whitney. I risultati dello studio condotto mostrano

che il riuso espresso in termini di ereditarietà di implementazione risulta essere maggiormente utilizzato rispetto al riuso espresso in termini di ereditarietà di specifica e delegazione. Inoltre i valori delle metriche relative alle 3 variabili di riuso mostrano come l'ereditarietà di implementazione tende ad aumentare da versione a versione. Inoltre, è stato possibile evidenziare che esiste una correlazione tra il riuso espresso in termini di ereditarietà di implementazione e il numero dei bug.

CAPITOLO 1

Introduzione

1.1 Contesto applicativo

Con l'evoluzione informatica degli ultimi anni, le risorse di calcolo disponibili nei dispositivi moderni permettono la creazione e l'esecuzione di programmi sempre più complessi e ricchi di funzionalità. L'aumento della complessità porta con sè tante insidie in quanto l'aumento del codice sorgente scritto dai programmatori, aumenta il rischio di introdurre errori all'interno del sistema.
In informatica si dice che "non esiste codice privo di bug" [1], ciò è dovuto all'aumento della complessità, in cui non è possibile prevedere tutti i possibili scenari legati ai bug all'interno del codice sorgente sia per questioni di tempo, di risorse e soprattutto di costi di sviluppo.
La scoperta di un bug porta il programmatore ad eseguire azioni di riparazione per la sezione di codice che causa il malfunzionamento e a rilasciare una nuova versione del programma corretta chiamata patch.
Quando si cerca di trovare una soluzione per correggere il bug, è bene effettuare prima una fase di individuazione, cercando di ottenere quanti più dettagli possibili, mediante una breve descrizione oppure eseguire una sequenza di azioni in grado di ricercare tutte le classi che sono state coinvolte a seguito della scoperta del bug.

Tuttavia, estrarre, riprodurre e isolare questi bug reali richiede un notevole sforzo, e quindi bug reali sono raramente utilizzati nella ricerca di test di software.
Altro aspetto non meno importante durante il processo di risoluzione dei difetti è quello di analizzare la relazione tra qualità del software in termine di presenza dei difetti e riuso del software che incidentalmente potrebbe contribuire alla introduzioni di difetti.

1.2 Motivazioni ed obiettivi

L'ingegneria del software è la disciplina tecnologica e manageriale che si occupa dei processi produttivi e delle metodologie di sviluppo finalizzate alla realizzazione di sistemi software di alta qualità [2].
Un sistema software viene definito di qualità se rispecchia requisiti ben precisi, in particolare:

- **Funzionalità** : Un prodotto software risulta funzionale se è in grado di svolgere azioni utili in base ai bisogni espressi dagli utenti finali. Altro aspetto che rispecchia questa caratteristica è la capacità di svolgere compiti ben precisi per la risoluzione di obiettivi specifici.

- **Usabilità** : Un prodotto software risulta usabile se l'utente finale non riscontra difficoltà nell'utilizzo del prodotto stesso, quindi di facile comprensione per l'esigenze che deve soddisfare. Inoltre in alcuni casi il successo di un prodotto software dipende più dalla percezione che ha l'utente nell'utilizzo rispetto alle funzionalità che è in grado di offrire.

- **Affidabilità**: Un prodotto software risulta affidabile se riesce a mantenere per un determinato periodo di tempo e a determinate condizioni, alte prestazioni.

- **Efficienza**: Un prodotto software risulta efficente se è in grado di utilizzare in modo proporzionato il rapporto tra la quantità delle risorse e le prestazioni del prodotto.

- **Manutenibilità** : la manutenibilità è legata alla facilità di apportare modifiche, adattamenti e miglioramenti a seguito dell'evoluzione del software, in dettaglio in base ai cambiamenti dei requisiti e delle specifiche funzionali

- **Portabilità** : un prodotto software risulta portabile se risulta facilmente integrabile e adattabili in altri ambienti di sviluppo.

Altro concetto che influenza in particolar modo la qualità di un prodotto software legato ai requisiti di efficienza è la presenza dei difetti. La maggior parte delle attività di controllo sono legate proprio a questo concetto e alla scoperta e alla rimozione di ogni possibile difetto o malfunzionamento.
Per questo motivo durante lo sviluppo di un prodotto software è indispensabile effettuare una fase di refactoring, in cui viene effettuata una revisione riguardante i punti deboli e poco lineari del codice sorgente[1].

Altro principio chiave dell'Ingegneria del Software è il concetto di riuso in quanto consente agli sviluppatori di riutilizzare pezzi di codice che sono già stati precedentemente sviluppati e testati [3].

Il riutilizzo del codice durante lo sviluppo dei software (Object Oriented) ha un ruolo di cruciale importanza in quanto offre:

1. **riduzione dei tempi di sviluppo**: il riutilizzo del codice permette di ridurre il time-to-market. Vantaggio notevole per le aziende leader all'interno del mercato. Altro aspetto di fondamentale importanza legato alla riduzione dei tempi è dovuto ai programmatori che non creano codice da zero, bensì riutilizzano parti di codice già testate.

2. **costo ridotto** : i programmatori possono utilizzare codice già implementato da altri programmatori, in questo modo le aziende non hanno bisogno di utilizzare risorse aggiuntive con la possibilità di tenere sotto controllo i costi di sviluppo;

3. **minore quantità di codice**: il riuso di codice efficiente e sistematico evita una grande quantità di codice sorgente che spreca molte risorse da gestire.

[1] https://www.ionos.it/digitalguide/siti-web/programmazione-del-sito-web/refactoring/

Da qui l'obiettivo di studiare la relazione tra qualità del software in termine:

1. di presenza di difetti
2. e di riuso del software in termini di ereditarietà di specifica, di implementazione e di delegazione durante l'evoluzione.

Per quanto riguarda le informazioni relative alla presenza dei difetti nel codice, in questo lavoro è stato utilizzato Defects4J, un database e un framework estendibile che fornisce informazioni su bug reali per consentire studi riproducili nell'ambito della ricerca sul testing del software. Invece, per ottenere le informazioni relative al riuso in termini di ereditarietà è stato utilizzato il tool InhMetrics, che è uno strumento definito e sviluppato in una tesi precendente nell'ambito delle attività portate avanti nel Laboratorio di Ingegneria del Software dell'Università degli Studi di Salerno.
In particolare, attraverso tre metriche il plug-in InhMetrics è in grado di quantificare tre tipi di riuso: in termini di ereditarietà di specifica, di implementazione e di delegazione. Si ricorda che l'ereditarietà di specifica si può definire, come la possibilità di sostituire un oggetto al posto di un altro. L'ereditarietà di implementazione, invece, definisce l'implementazione di un oggetto in funzione a quella di un altro già esistente. Infine, la delegazione viene utilizzata per fornire una soluzione a problemi che possono essere introdotti dall'uso dell'ereditarietà di implementazione. I dettagli dello strumento ed un suo utilizzo possono essere trovati in [4].

In questo lavoro di tesi i valori delle tre metriche sono analizzate per caratterizzare l'evoluzione del riuso, e per studiarle in relazione alla presenza dei difetti e alla densità dei difetti, facendo riferimento alle informazioni contenute in Defects4J. A tal fine sono state formulate e quindi investigate tre research question.

La prima research question (**RQ1**) ha lo scopo di analizzare come varia il riuso in termini di ereditarietà di specifica, ereditarietà di implementazione, ed ereditarietà di delegazione nell'evoluzione del software.

La seconda research question (**RQ2**) è stata invece formulata per studiare come varia il riuso rispetto al numero di bug nell'evoluzione del software.

La terza research question (**RQ3**) ha lo scopo di determinare come come varia il riuso rispetto al variare alla densità dei bug nell'evoluzione del software.

1.3 Struttura della tesi

Il seguente lavoro di tesi è strutturato nei seguenti capitoli.

Nel Capitolo 2 vengono descritti i principali concetti legati al riuso del software nella programmazione Object Oriented. Inoltre viene fornita una panoramica del lavoro sulla predizione dei difetti e sull'importanza in termini di sviluppo del software. Infine vengono descritti i lavori relati allo studio. presentato in questo lavoro di tesi.

Nel Capitolo 3 viene descritta la metodologia di analisi considerata per lo studio condotto, specificando obiettivi, contesto, procedura di analisi, e le minacce alla validità dei risulati ottenuti.

Nel Capitolo 4 vengono presentati e discussi i risultati ottenuti per ogni reserach question formulata.

Infine nel Capitolo 5 vengono descritte le conclusioni del seguente lavoro di tesi, seguite da una indicazione di possibili sviluppi futuri.

CAPITOLO 2

Background e Stato dell'arte

Questo capitolo illustra i concetti di riuso nella programmazione Object Oriented. In particolare viene evidenziato il significato di ereditarietà di specifica, implementazione e delega. Inoltre vengono evidenziati concetti sulla predizione dei difetti e infine viene illustrato lo stato dell'arte.

2.1 Riuso del codice sorgente

Durante lo sviluppo di codice i programmatori spesso possono trovarsi a produrre codice simile a quello già esistente, ciò risulta controproducente poichè porta sprechi di tempo e maggiori costi di sviluppo. Per ovviare a queste due problematiche viene attuato il riuso, il quale permette di non creare componenti da zero bensì di utilizzare pezzi di codice già precedentemente creati e testati. Nella programmazione Object Oriented il riuso viene realizzato attraverso l'ereditarietà [5].

Nel caso in cui ci sia una classe A già definita è possibile a partire da questa realizzare una seconda classe B che eredita lo stato e il comportamento della classe A.
In questo modo viene stabilita una gerarchia tra classi in cui la classe A viene chiamata Superclasse (classe padre) e la classe B Sottoclasse (classe figlia).
In Java questa gerarchia viene creata attraverso la parola chiave:

- **extends**: consente di creare una sottoclasse utilizzando le funzionalità di una superclasse.

In dettaglio l'utilizzo della parola chiave **extends**:

- non permette di sovrascrivere tutti i metodi di una superclasse;
- una classe può estendere solo una superclasse;
- un'interfaccia può estendere più interfacce;

Il riuso può essere realizzato anche con il concetto di Interface in Java, che si basa sull'utlizzo della parola chiave:

- **implements**: utilizzata per implementare tutti i metodi definiti all'interno di un'interfaccia.

Analogalmente l'utilizzo della parola chiave **implements**:

- prevede di implementare tutti i metodi di un'interfaccia che è stata implementata;

- permette ad una classe di implementare più interfacce;
- non permette ad un'interfaccia di implementare un'altra interfaccia;

In questo lavoro di tesi, è stato quantificato il riuso in termini di ereditarietà di implementazione, di specifica e di delegazione.

In particolare, **l'ereditarietà di implementazione** può essere definita come il meccanismo che permette l'implementazione di una sottoclasse in funzione di una superclasse quindi di un oggetto già esistente. Ciò permette alla sottoclasse di ereditare tutte le operazioni che sono già state definite nella superclasse. Con questo tipo di ereditarietà gli sviluppatori possono riusare codice in modo veloce estendendo una classe esistente e rifinendo il suo comportamento.

L'ereditarietà di specifica permette alla superclasse di specificare quali metodi dovrebbero essere disponibili ma non fornisce il codice, in java questo è supportato tramite le interfacce e i metodi astratti, i quali vengono definiti attraverso la sola signature all'interno della superclasse astratta e implementati all'interno delle sottoclassi[1].

Inoltre il concetto di ereditarietà di specifica risulta comune al **principio di Sostituzione di Liskov**[2] il quale afferma che :

" Se per ogni oggetto o1 di tipo S c'è un oggetto o2 di tipo T, tale che per tutti i programmi P definiti in termini di T, il comportamento di P non vari sostituendo o1 a o2, allora S è un sottotipo di T "

Infine la **delegazione** è un'alternativa all'ereditarietà di implementazione applicabile quando si vuole riutilizzare codice. Essa permette di utilizzare un oggetto di un'altra classe come variabile d'istanza e inoltrare messaggi all'istanza, inoltre può essere considerata come una relazione tra oggetti in cui un oggetto inoltra determinate chiamate di metodo ad un altro oggetto.

[1] http://www.btechsmartclass.com/java/java-forms-of-inheritance.html
[2] https://www.html.it/pag/32393/l-liskov-substitution-principle/

2.2 Predizione dei difetti

La previsione dei difetti nel software è una fase cruciale e risulta utile per gestire i costi che una azienda deve sostenere per la realizzaione del prodotto software. Identificare i problemi all'interno di un prodotto in fase di sviluppo è un processo che garantisce qualità, necessaria per soddisfare i clienti.

Molti sono i vantaggi di identificare quanto prima possibile i difetti. Il National Institute of Standard Technology (NIST) ha pubblicato uno studio nel 2002 osservando che il costo per correggere un bug riscontrato nella fase di produzione del software è di 15 ore rispetto a cinque ore di lavoro se lo stesso bug fosse trovato nella fase di codifica [6].

Il Systems Sciences Institute dell'IBM ha riferito che il costo per correggere un errore riscontrato dopo il rilascio del prodotto era da quattro a cinque volte superiore a quello scoperto durante la progettazione e fino a 100 volte superiore a quello identificato nella fase di manutenzione [7] (vedi ***Figura*** 2.1).

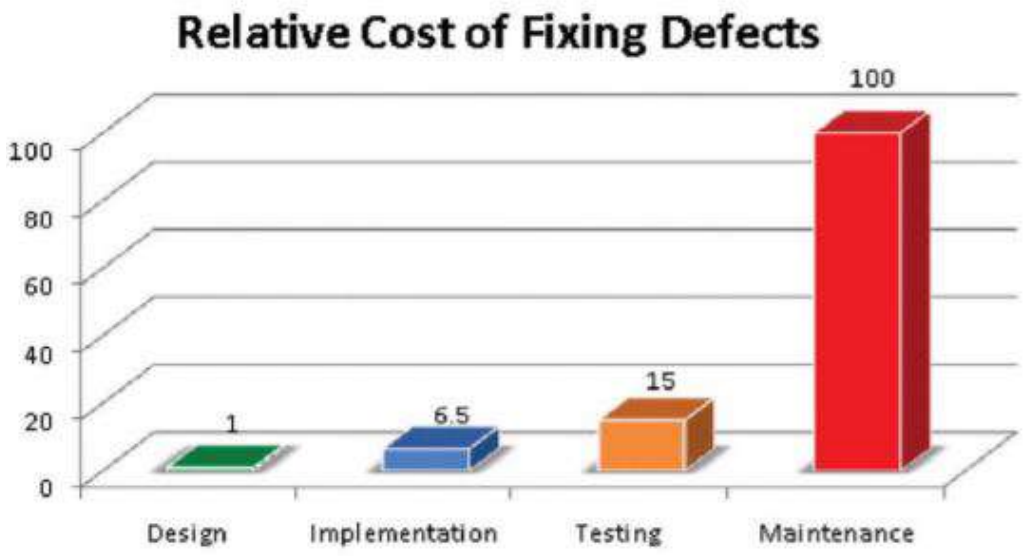

Figura 2.1: Costi relativi per correggere i difetti nel software (Source: IBM Systems Sciences Institute)

La predizione dei difetti richiede di adottare una metodolgia ben strutturata di risoluzione dei problemi per identificare, analizzare e prevenire il verificarsi dei difetti.
Effettuare una prevenzione dei difetti in un prodotto software comporta diverse attività come:

- la raccolta di una grande quantità di dati relativi ai difetti mantenuti all'interno di un database;
- un'analisi delle cause principali che hanno generato il difetto e un continuo aggiornamento di tale informazioni;
- l'individuzione e l'esecuzione di azioni correttive per evitare che si possono verificare difetti futuri;

Di seguito vengono riportate le attività generali da eseguire per la prevenzione dei difetti in un prodotto software.

1. **Software Requirements Analysis** (Analisi dei Requisiti)

   Gli errori nei requisiti software e nei documenti possono essere più frequenti degli errori nel codice, inoltre questi tipi di errori non possono essere trovati attraverso fasi di testing. Per evitare che questi errori evidenziati nella parte iniziale di un processo di sviluppo software impattino sul resto del sistema divenendo veri e propri difetti nelle fasi successive bisogna effettuare delle revisioni e ispezioni pre-test, eseguiti da un singolo membro del team per tutti i prodotti di lavoro.

2. **Review: Self-Review and Peer Review** (Autovalutazione e Revisione)

   Prima che i difetti vengano scoperti da un team di test o direttamente da un cliente, è bene eseguire un'autocorrezione sul codice che aiuta a ridurre i difetti legati alla logica errata.

3. **Defect Logging and Documentation** (Registrazione e documentazione dei difetti)

Una delle attività principali per la corretta risoluzione dei difetti è quella di avere a disposizione un tracciamento dei difetti individuati, per questo motivo occorre registrarli ed effettuare un reporting per la gestione. Quest'analisi è conveniente in termini di costi poichè si tiene traccia di quanti difetti rimangono alla fine di un buon prodotto.

Inoltre quando viene eseguita una fase di registrazione dei difetti bisogna documentare le informazioni relative a difetti in modo da avere :

- descrizione corretta e completa del difetto, in modo tale che tutti di membri del team di sviluppo possono capire di cosa si tratta e nel caso riprodurlo;
- descrizione del difetto tramite screenshot;
- descrizione della fase in cui viene rilevato il difetto, per poter adottare delle misure per evitare la propagazione del difetto in fasi successive;
- traccia dei nomi degli sviluppatori che scoprono i difetti, in questo modo il team sa a chi rivolgersi per una corretta risoluzione;

4. **Root Cause Analysis** (Analisi della causa principale)

   In generale il leader del progetto di sviluppo si occupa di effettuare una riunione per discutere e analizzare le cause profonde che hanno generato il difetto. Inoltre l'analisi può includere due approcci.

Analisi di Pareto

Tale analisi permette una classificazione in base alla priorità espressa in percentuale della risoluzione dei problemi. Mediante questa classificazione ci si concentra in questo caso sui difetti che hanno maggiore priorità e che quindi hanno un'elevato impatto sulla qualità del codice.

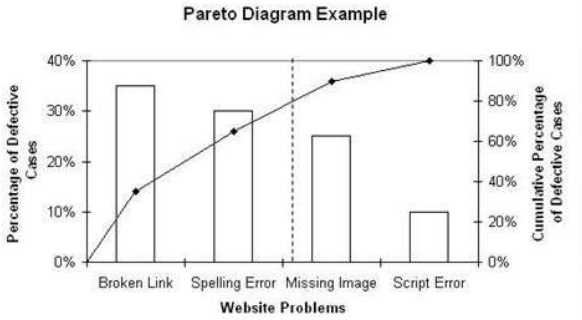

Figura 2.2: Diagramma Pareto

In ***Figura*** 2.2 un esempio generale del diagramma di Pareto in cui viene mostrato il principio 80/20 il quale afferma che l'80% dell'impatto del problema si manifestarà nel 20% delle cause.

Analisi di Ishikawa

Questa tipologia di analisi permette di identificare possibili cause di un problema. In questo tipo di analisi non vengono fornite statistiche bensì viene fornita una rappresentazione grafica per ordinare e mettere in relazione i fattori che contribuiscono a una determinata situazione. Inoltre questa tecnica si basa sul brainstorming al livello di team. In cui l'obiettivo è quello di determinare quali modifiche dovrebbero essere incorporate nei processi in modo da ridurre al minimo la ricorsività dei difetti.

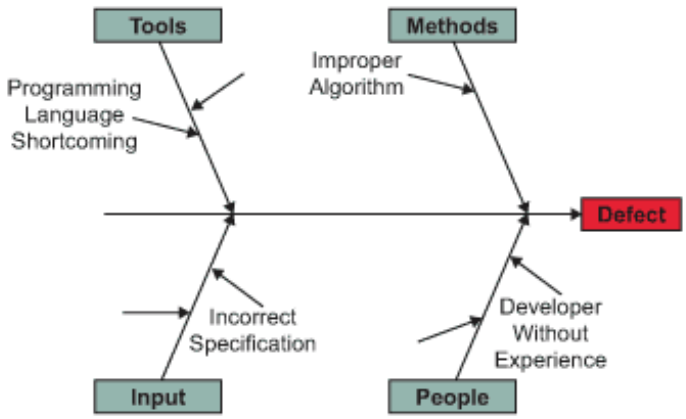

Figura 2.3: Diagramma Ishikawa

In ***Figura*** 2.3 nella pagina precedente un esempio generale del diagramma di Ishikawa in cui il problema viene posto alla destra del diagramma e sulla linea orizzontale vengono elencate le varie cause.

Infine a seguito del principio utilizzato e all'analisi effettuata, l'ultimo passo consiste nell'implementare tutte le attività di prevenzione dei difetti.
L'implementazione risulta la parte più difficile in quanto richiede un notevole sforzo dal team di sviluppo, il quale a seguito di un piano di modifica dei processi esistenti si occupa di prevenire i difetti e quindi di eseguire azioni di riparazione.

2.3 Stato dell'arte

Il riuso del codice sorgente ha un ruolo centrale nell'ingegneria del software in quanto permette di riutilizzare blocchi di codice già precedentemente implementati e testati. Diversi studi hanno evidenziato l'importanza di effettuare il riuso durante il processo di sviluppo del software, il quale può portare numerosi vantaggi in termini di costi e affidabilità.
Nel seguito si riporta una discussione dei lavori presenti in letteratura, che evidenzia contributi e benefici dai vari meccanismi di riuso adottati.

Prechelt et al. [8] hanno condotto due esperimenti controllati per confrontare le prestazioni relative alle attività di manutezione, mostrando che lo sforzo di manutenzione del codice dipende da fattori correlati alla profondità dell'ereditarietà.

In precedenza Dali et al.[9] hanno condotto una serie di esperimenti controllati in cui viene messo in relazione l'effetto dell'ereditarietà sulla manutenibilità del codice. I loro studi hanno dimostrato che i soggetti che utilizzano l'ereditarietà all'interno di software Object-Oriented riescono ad eseguire attività di modifica del 20 % più velocemente dei soggetti che non utilizzavano l'ereditarietà nei proprio software.

Studi simili sono stati condotti da Goel e Bathia [10], i quali hanno condotto una valutazione empirica delle metriche orientate agli oggetti in C, prendendo in considerazione 3 programmi che utilizzavano ereditarietà multilivello, multipla e gerarchica. L'analisi eseguita ha dimostrato che l'ereditarietà multilivello ha un impatto maggiore sul riuso rispetto all'ereditarietà multipla e gerarchica.

Studi inerenti alle metriche di riuso sono stati eseguiti anche da Chawla e Nath[11] che hanno valutato come le metriche orientate agli oggetti aiutino a stimare la qualità del software.

Inoltre in questo lavoro di tesi è stato fatto riferimento all'articolo "On the Evolution of Inheritance and Delegation Mechanisms and Their Impact on Code Quality"[4] che riporta uno studio avente come obiettivo quello di fornire un'analisi dell'evoluzione dell'ereditarietà di specifica, di implementazione e delega e di come questi meccanismi impattano sulla qualità del codice sorgente. In particolare dallo studio condotto è stato riscontrato che l'ereditarietà e la delega si evolvono nel tempo ma non in modo statisticamente significativo inoltre la loro evoluzione riduce la gravità dei code smells, migliorando la manutenibilità del codice. Una versione preliminare dello studio è stata condotta in [12] ed l'analisi ha mostrato come le soglie inerenti al riuso tendono ad aumentare da versione a versione e in particolare come le metriche relative al riuso in termini di delegazione e implementazione risultano correlate al numero dei bug.

CAPITOLO 3

Metodologia di Analisi

In questo capitolo viene descritta la metodologia di analisi adottata per portare avanti lo studio. In particolare, venogno presentati l'obiettivo dello studio seguito dalla descrizione dei tool utilizzati, sia per quantificare le metriche relative al riuso in termini di ereditarietà di implementazione, delegazione e specifica e sia le informazioni dei difetti relative ad ogni sistema software. In sintesi, viene descritto tutto quello che è stato fatto per estrapolare tutte le informazioni indispensabili per la fase di analisi.

3.1 Obiettivi dello studio

L'obiettivo dello studio empirico presentato in questo lavoro di tesi è quello di analizzare e valutare in che modo il riuso espresso in termini di ereditarietà e la presenza di difetti all'interno di sistemi impattano sulla qualità e sull'evoluzione del software. Il focus quindi è sul riuso e di come varia all'interno dei progetti software da versione a versione.

Per analizzare in che modo il riuso espresso in termini di ereditarietà impatta sull'evoluzione del software verrà utilizzato lo strumento InhMetrics capace di determinare informazioni relative alle metriche introdotte per quantificare il riuso in termini di ereditarietà di specifica, ereditarietà di implementazione e delegazione.

Per la presenza dei difetti verranno recuperate informazioni relative ai bug contenuti all'interno di sistemi software Java attraverso Defects4J.

3.2 Contesto dello studio

Il contesto dello studio empirico consiste nell'analisi di 7 sistemi Java proposti da Defects4J. Nello specifico vengono analizzati Cli, Codec, Collections, Csv, JxPath, Lang, Math. In particolare vengono analizzate le variabili di Somma e Media riferite all'ereditarietà di implementazione, specifica e delegazione.

La selezione del contesto è stata guidata da un requisito principale, vale a dire la disponibilità di un insieme di sistemi Java, presentati dal framework Defects4J per valutarne i meccanismi di riuso, con l'obiettivo di fornire risultati che potranno essere integrati negli studi effettuati e approfondire le conoscenze sull'impatto dell'ereditarietà di implementazione, di specifica e delega in presenza dei difetti sulla qualità del software.

3.2.1 Defects4J

La riproduzione dei bug nell'ambito del collaudo del software risulta molto difficile a causa della mancanza di database di bug facili da utilizzare. D'altra parte estrarre questi bug, riprodurli e isolarli richiede un notevole sforzo. Per questo motivo è stato utilizzato Defects4J.

Defects4J è un database e framework estendibile che fornisce bug reali per consentire studi riproducibili nella ricerca di test.
La versione iniziale di Defects4J conteneva 357 bug reali da 5 programmi open source, in cui ogni bug reale è accompagnato da una completa suite di test che espone bug. Ad oggi sono presenti 853 bug relativi a 17 progetti open source.

All' interno di questo elaborato è risultato determinante per recuperare le informazioni relative ai bug di ogni sistema (come si può notare in ***Tabella*** 3.1). In particolare è stato possibile analizzare 7 sistemi dei 17 proposti da Defects4J in quanto per i restanti 10 non è stato possibile recuperare il source code dal repository Apache.

Tabella 3.1: Sistemi Defects4J Analizzati

Identifier	Project Name	Number of Bugs	Active Bug Ids	Deprecate Bug-Ids
Cli	commons-cli	39	1-5;7-40	6
Codec	commons-codec	18	1-18	None
Collections	commons-collections	4	25-28	1-24
CSV	commons-csv	16	1-16	None
JxPath	commons-jxpath	22	1-22	None
Lang	commons-lang	64	1,3-65	2
Math	commons-math	106	1-106	None

Le informazioni riportate in tabella sono:

- **Identifier**: specifica l' identificatore del progetto;
- **Project Name**: nome del progetto;
- **NumberOfBugs**: il numero di bug contenuti nel progetto;

- **ActiveBugsIds**: l'intervallo dei bug attivi per ogni progetto;
- **Deprecated Bug Ids**: i bug deprecati a seguito dei rilasci di versioni di ogni sistema;

Inoltre Defects4J offre un'interfaccia ad alto livello per la ricerca di test del software, il cui compito è facilitare l'esecuzione di studi empirici.
Per l'esportazione dei bug riferiti ad ogni sistema software analizzato sono stati utili i comandi forniti dalla guida di Defects4J presente su GitHub[1]:

- **info**: per ha permesso la stampa per ottenere le informazioni per ogni progetto o bug specifico;
- **checkout**: il quale ha permesso di controllare una particolare versione del progetto con errori nella directory di lavoro fornita;
- **compile**: ha permesso di compilare i sorgenti del progetto;
- **bids**: ha fornito un elenco di ID bug(identificatore bug) per un progetto scelto;
- **pids**: ha fornito l'elenco di tutti di ID dei progetti contenuti in Defects4J;
- **export**: il quale ha permesso nel nostro caso di esportare tutte le proprietà riferite ai sistemi analizzati;
- **query**: il quale ha permesso di interrogare i metadati per ottenere informazioni ancora più dettagliate in formato Csv per ogni sistema scelto;

[1] https://github.com/rjust/defects4j

3.3 Estrazione dei dati

All'interno di questa sezione vengono riportati i dati e le informazioni estratte per i sistemi analizzati dai tool InhMetrics e Defects4J necessarie per mettere in relazione il riuso in termini di ereditarietà con la presenza dei difetti durante l'evoluzione.

3.3.1 Download dei software

Una delle attività più importanti per poter realizzare questo lavoro, è stata quella di recuperare i dati. In particolare per ottenere il codice sorgente scritto in Java relativo ai sistemi proposti da Defects4J sono stati utilizzati diversi repository.

In primo luogo è stato utilizzato Apache Repository[2] il quale ha permesso lo scaricamento delle seguenti versioni Lang, Math, Codec, JxPath, Collections, Csv e Cli.

In particolare Apache Repository offre la possibilità di accedere agli archivi del sistema ricercato, il quale contiene sia i codici binari che il codice sorgente per ogni sistema analizzato.

In secondo luogo è stato utilizzato MavenRepository[3] il quale ha permesso di individuare i jar corrispondenti alle versioni importate su IdeEclipse e la possibilità di ottenere maggiori informazioni relative ai bug corretti per ogni versione del sistema considerato.

Infine a fini di una corretta compilazione dei sistemi è stato possibile recuperare ulteriori informazioni riferite ai bug e alle librerie di ogni versione, attraverso GitHub.

[2]https://commons.apache.org/
[3]https://mvnrepository.com/

3.3.2 Informazioni Defects4J

Le seguenti informazioni sono state ricavate per ogni sistema contenuto in Defects4J[4] ed esportate in file CSV per ogni versione del sistema:

- **BugId** –> Id bug assegnato;
- **ReportID** –> ID segnalazione bug dal tracker versione per ogni bug;
- **VersionRelease** –> Versione di rilascio del bug;
- **ProjectName** –> Nome del progetto;
- **RevisionDataBuggy** –> Data del commit del bug per ogni bug
- **RevisionDataFixed** –> Data del commit corretto per ogni bug
- **ClassesModified** –> Classi modificate dalla correzione del bug
- **TestTrigger** –> Elenco di metodi di test che attivano (espongono) il bug
- **TestsTriggerCause** –> Elenco dei metodi di test che attivano (espongono) il bug, insieme alla causa principale

In ***Tabella*** 3.2 nella pagina seguente viene riportato un esempio contenente tutte le informazioni esportate da Defects4J per il sistema Codec.

[4]https://github.com/rjust/defects4j

Tabella 3.2: Metadati - Esportati Defects4J

Report ID	Bug ID	Version Release	ProjectName	RevisionDateBuggy		RevisionDateFixed	
CODEC-65	Codec	1.4	commons-codec	2008-04-27	00:33:06	2008-04-27	00:35:56
CODEC-77	Codec	1.4	commons-codec	2009-07-13	20:37:32	2009-07-13	22:33:28
CODEC-84	Codec	1.4	commons-codec	2009-08-02	02:56:37	2009-08-02	22:45:30
CODEC-89	Codec	1.5	commons-codec	2010-03-26	23:52:46	2010-03-27	00:02:22
CODEC-98	Codec	1.5	commons-codec	2010-06-01	21:04:06	2010-06-01	21:52:33
CODEC-101	Codec	1.5	commons-codec	2011-01-20	16:33:34	2011-01-21	19:19:51
CODEC-99	Codec	1.5	commons-codec	2011-01-22	05:57:53	2011-01-23	05:52:42
CODEC-105	Codec	1.5	commons-codec	2011-01-23	23:17:18	2011-01-24	00:46:09
CODEC-112	Codec	1.5	commons-codec	2011-01-26	19:09:00	2011-01-26	23:40:25
CODEC-117	Codec	1.5	commons-codec	2011-03-01	17:34:35	2011-03-01	17:56:14
CODEC-121	Codec	1.10	commons-codec	2012-03-02	20:58:43	2012-03-07	15:34:01
CODEC-130	Codec	1.7	commons-codec	2012-03-18	21:45:43	2012-03-19	18:33:42
CODEC-184	Codec	1.10	commons-codec	2014-04-10	13:50:21	2014-04-10	13:51:06
CODEC-187	Codec	1.10	commons-codec	2014-06-29	01:55:09	2014-07-05	19:58:38
CODEC-199	Codec	1.11	commons-codec	2015-03-22	17:32:16	2015-03-22	18:48:52
CODEC-200	Codec	1.11	commons-codec	2015-06-04	13:29:28	2015-06-04	14:01:15
CODEC-229	Codec	1.11	commons-codec	2016-11-19	04:18:00	2017-03-26	17:22:08
CODEC-231	Codec	1.11	commons-codec	2017-03-26	18:34:51	2017-03-26	21:43:36

Inoltre le informazioni non riportate riferite alle ClassesModified, TestTrigger e TestTriggerCause sono state recuperate mediante l'esecuzione degli script presenti al link GitHub contenuto nella nota[5].

In particolare di maggiore interesse è risultato il numero di classi modificate per ogni bug del sistema, che ha permesso il calcolo della densità per ogni versione del sistema illustrato in tabella. In Tabella 3.3 sono riportate le statistiche relative a numero di bug, numero di classe coinvolte e densità del bug per ogni versione di tutti i sistemi considerati.

Tabella 3.3: Calcolo Densità Versioni Analizzate

Version System	Num. Bug	Num_Classi_Coinvolte	Densità Bug
Cli 1.1	3	3	1
Cli 1.2	16	11	1,454545455
Cli 1.3	8	7	1,142857143
Cli 1.3.1	1	1	1

[5]http://defects4j.org/html$_doc/d4j/d4j - export.html$

Table 3.3 continued from previous page

Cli 1.4	3	3	1
Cli 1.5.0	2	1	2
Codec 1.4	3	5	0,6
Codec 1.5	7	3	2,333333333
Codec 1.7	1	1	1
Codec 1.10	3	10	0,3
Codec 1.11	4	3	1,333333333
Collections 4.1	3	3	1
Collections 4.2	1	1	1
CSV 1.0	8	4	2
CSV 1.1	1	1	1
CSV 1.3	1	2	0,5
CSV 1.5	1	1	1
CSV 1.6	2	2	1
JXPath 1.3	18	22	0,8181818182
Lang 3.0	14	11	1,272727273
Lang 3.0.1	1	1	1
Lang3.2	14	9	1,555555556
Math1.2	7	9	0,7777777778
Math2.0	13	11	1,181818182
Math2.1	14	13	1,076923077
Math2.2	10	10	1
Math3.0	17	17	1
Math3.1	19	15	1,266666667
Math3.1.1	2	3	0,6666666667
Math3.2	8	14	0,5714285714
Math3.3	4	6	0,6666666667

3.3.3 Metriche InhMetrics

Il riuso in termini di ereditarietà di specifica, di delegazione e di implementazione è stato valutato con l'ausilio del tool InhMetrics, il quale permette di estrapolare correttamente i valori delle variabili relative al riuso. Inoltre si sono cacolcate per ognuna delle tre variabili di riuso le statistiche su Media e Somma per ogni versione dei sistema software considerati al fine di valutarne l'evoluzione.

Le metriche esportate da InhMetrics sono le seguenti:

- **Ered_Spec**: metrica che permette di calcolare per un determinato tipo (classe) il livello di riuso inerente all'ereditarietà di specifica;
- **Ered_Impl**: metrica che permette di calcolare per un determinato tipo (classe) il livello di riuso inerente all'ereditarietà di implementazione;
- **Ered_Deleg**: metrica che viene incrementata appena viene rilevata la prima occorrenza d'uso di una variabile di istanza in uno dei metodi definiti nella classe;

In Tabella ***tabella*** 3.4 nella pagina successiva sono riportate le i valori delle statistiche Somma e Media per ognuna delle variabili considerate.

Tabella 3.4: Variabili Somma e Media relative al riuso per ogni versione analizzata (Inh Metrics)

Version System	Ered_Spec_Media	Ered_Spec_Somma	Ered_Impl_Media	Ered_Impl_Somma	Ered_Deleg_Media	Ered_Deleg_Somma
Cli 1.1	0,195652174	9	0,043478261	2	0,326086957	15
Cli 1.2	0,234042553	11	0,106382979	5	0,404255319	19
Cli 1.3	0,26	13	1,3	64	0.44	22
Cli 1.3.1	0,294117647	15	1,274509804	65	0,431372549	22
Cli 1.4	0,240740741	13	1,185185185	64	0,444444444	24
Cli 1.5.0	0,220338983	13	1,169491525	69	0,406779661	24
Codec 1.4	0,068965517	4	0,068965517	4	0,103448276	6
Codec 1.5	0,175	14	0,3375	27	0,075	6
Codec 1.7	0,115702	14	0,504132	61	0,123967	15
Codec 1.10	0,105263	14	0,278195	37	0,112782	15
Codec 1.11	0,096552	14	0,255172	37	0,137931	20
Collections 4.1	0,2645507	196	0,133603239	99	0,008097166	6
Collections 4.2	0,262948	198	0,132802125	100	0,007968127	6
CSV 1.0	0	0	0	0	0,357143	10
CSV 1.1	0	0	0	0	0,357143	10
CSV 1.3	0	0	0	0	0,294118	10
CSV 1.5	0	0	0	0	0,314286	11
CSV 1.6	0	0	0	0	0,297297	11
JXPath 1.3	0,539823	112	2,137168	483	0,353982	80
Lang 3.0	0,237197	88	0,237197	88	0,083558	31
Lang 3.0.1	0,234043	88	0,234043	88	0,082447	31
Lang3.2	0,224824	96	0,245902	105	0,084309	36
Math1.2	0,578834	268	0,809935	375	0,282937365	131
Math2.0	0,662042875	525	1,21185372	961	0,287515763	228
Math2.1	0,641330166	540	1,412114014	1189	0,283847981	239
Math2.2	0,668454936	623	1,39055794	1296	0,284334764	265
Math3.0	0,425327511	487	1,109170306	1270	0,281222707	322
Math3.1	0,441116406	648	0,918992512	1350	0,302927161	445
Math3.1.1	0,442556084	651	0,930659415	1369	0,302515296	445
Math3.2	0,433876221	666	0,900977199	1383	0,300977199	462
Math3.3	0,42575928	757	0,886951631	1577	0,29583802	526

3.3.4 Metriche di Chidamber-Kemerer (C-K)

Nello studio condotto in questo lavoro di tesi sono state prese in considerazione anche le metriche relative alla suite Chidamber-Kemerer per fornire una misurazione di altre caratteristiche strutturali e non dei sistemi software analizzati. Ancora una volta sono state considerate la Somma e la Media di queste metriche per ogni versione dei sistemi software considerati.

La suite è caratterizzata dalle seguenti metriche[13]

- **WMC: Weighted Methods per Class.**

  WMC rappresenta la somma delle complessità dei metodi per una classe. Inoltre se i metodi hanno pari complessità, WMC è uguale al numero di metodi della classe. Se il valore di WMC è alto, indica una grande quantità di lavoro di manutenzione, in quanto le classi risultano essere specifiche e quindi offrono poco riuso. Quindi, un aumento del valore di WMC porta ad aumentare la densità dei bug e a far diminuire la qualità;

- **DIT: Depth of Inheritance Tree**

  DIT indica il massimo numero di livelli dalla classe alla radice della gerarchia (radice =0). Maggiore è il valore del DIT per una classe, più difficile risulta capirne il comportamento, dato che la quantità dei metodi ereditati è elevato. Inoltre un valore del DIT alto indica maggiore complessità sull'intero sistema e quindi anche maggior riuso.

- **NOC: Number of Children**

  Numero di sottoclassi di una classe della gerarchia. Il DIT effettua una misurazione sulla profondità, mentre con il NOC riesce a definire l'ampiezza dell'albero gerarchico. Maggiore è il valore di NOC maggiore sarà il riuso della classe. Infine, un elevato NOC può indicare un minor numero di guasti, in quanto indica un elevato riutilizzo della classe base, ma incrementerà il livello di accoppiamento;

- **CBO: Coupling Between Object classes**

CBO rappresenta l'accoppiamento di una classe. Maggiore è il valore di CBO maggiore è la dipendenza di una classe da altre, ciò permette meno riuso e maggiore difficoltà nel modifcare e correggere la classe. Per questo motivo è desiderabile un CBO basso il quale evidenzia una classe indipendete e quindi maggiore possibilità di riutilizzare parti di codice.

- **RFC: Response for a Class** Rappresenta il set di risposta di una classe, cioè il numero di metodi eseguiti al ricevimento di un messaggio, ossia il numero di metodi di una classe + il numero dei metodi che essi invocano. In definitiva un valore RFC alto indica grande complessità quindi maggiore difficoltà di comprensione e di testing.

- **LCOM: Lack of Cohesion**

  LCOM indica la mancanza di coesione tra i metodi. Per calcolare LCOM per ogni campo della classe si calcola la percentuale dei metodi che usano tale campo, inoltre si effettua una media delle percentuali. Per aver un livello di coesione massimo, ogni metodo deve far riferimento a tutte le variabili di istanza dichiarate all'interno della stessa classe.

In ***Tabella*** 3.5 nella pagina seguente sono riportati i valori di Media e Somma di tutte le metriche di Chadember - Kemerer, per tutte le versioni dei sistemi analizzati.

Tabella 3.5: Variabili Somma e Media per Metriche Chadember & Kemerer per tutti i sistemi analizzati

Version System	WMC_Media	WMC_Somma	NOC_Media	NOC_Somma	LCOM_Media	LCOM_Somma	DIT_Media	DIT_Somma	CBO_Media	CBO_Somma	RFC_Media	RFC_Somma
Cli 1.1	14,36956522	661	0,217391304	10	0,130391304	5,998	2,239130435	103	0,52173913	24	40,30434783	1854
Cli 1.2	13,12765957	617	0,276595745	13	0,129510638	6,087	2,361702128	111	0,446808511	21	44,95744681	2113
Cli 1.3	16,92	846	0,32	16	0,13278	6,639	1,4	70	0,48	24	53,88	2694
Cli 1.3.1	16,66666667	850	0,333333333	17	0,130176471	6,639	1,431372549	73	0,470588235	24	53,09803922	2708
Cli 1.4	16,03703704	866	0,296296296	16	0,141462963	7,639	1,351851852	73	0,5	27	52,22222222	2820
Cli 1.5.0	16,13559322	952	0,271186441	16	0,129237288	7,625	1,271186441	75	0,457627119	27	53,06779661	3131
Codec 1.4	23,48275862	1362	0,603448276	35	0,045775862	2,655	1,982758621	115	0,189655172	11	43,5862069	2528
Codec 1.5	20,4125	1633	0,625	50	0,0515	4,12	2,15	172	0,1125	9	38,9375	3115
Codec 1.7	17,1157	2071	0,446281	54	0,074884	9,061	1,140496	138	0,272727	33	39,95041	4834
Codec 1.10	15.09	2478	0,451128	60	0,088782	11,808	1,142857	152	0,278195	37	42,31579	5628
Codec 1.11	18,85517	2734	0,413793	60	0,101759	14,775	1,110345	161	0,310345	45	41,0069	5946
Collections 4.1	12,8583	9528	0,904184	670	0,8456	62,659	2,302294	1706	0,80972	60	28,36167	21016
Collections 4.2	12,82072	9654	0,89243	672	0,083894	63,172	2,26162	1703	0,081009	61	28,15139	21198
CSV 1.0	23,35714	654	0	0	0,179571	5,028	0,964286	27	2,071429	58	69,57143	1948
CSV 1.1	24,75	693	0	0	0,186357	5,218	0,964286	27	2,071429	58	74,21429	2078
CSV 1.3	22,38235	761	0	0	0,153559	5,221	0,882353	30	2,088235	71	0,882353	30
CSV 1.5	24,05714	842	0	0	0,150971	5,284	0,914286	32	1,285714	45	80,97143	2834
CSV 1.6	23,27027	861	0	0	0,142514	5,273	0,864865	32	1,324324	49	79,91892	2957
JXPath 1.3	29,40708	6646	0,752212	170	0,200496	45,312	2,017699	456	0,606195	137	19,99115	4518
Lang 3.0	22,78706	8454	0,269542	200	0,058075	21,546	1,191375	442	0,350404	130	65,4097	24267
Lang 3.0.1	23,11436	8691	0,265957	100	0,057261	21,53	1,146277	431	0,518617	195	69,98404	26314
Lang3.2	22,95316	9801	0,306792	131	0,054965	23,47	0,737705	315	0,5363	229	69,71663	29769
Math1.2	13,69546436	6341	0,641468683	297	0,134334773	62,197	1,982721382	918	0,492440605	228	38,37365011	17767
Math2.0	16,12105927	12784	0,664564943	527	0,141567465	112,263	1,667087011	1322	0,480453972	381	45,97225725	36456
Math2.1	15,56294537	13104	0,732779097	617	0,139627078	117,566	1,595011876	1343	0,511876485	431	46,01068884	38741
Math2.2	16,74678112	15608	0,745708155	695	0,142404506	132,721	1,629828326	1519	0,815450644	760	51,04077253	47570
Math3.0	15,37554585	17605	0,749344978	858	0,129338865	178,093	1,255895197	1438	0,812227074	930	53,1790393	60890
Math3.1	14,84479238	21807	0,761742682	1119	0,117415929	172,484	1,186521443	1743	0,750850919	1103	52,20830497	76694
Math3.1.1	14,88375255	21894	0,761386812	1120	0,117260367	172,49	1,187627464	1747	0,749830048	1103	52,26376615	76880
Math3.2	15,20390879	23338	0,755700326	1160	0,115912052	177,925	1,180456026	1812	0,740065147	1136	55,35504886	84970
Math3.3	14,99493813	26661	0,737907762	1312	0,16880765	207,814	1,141169854	2029	0,706411699	1256	55,87176603	99340

3.4 Analisi dei dati

In questo capitolo viene descritta l'analisi condotta discutendo i DataSet e gli strumenti utilizzati per l'intera attività di analisi, per rispondere alle research question definite nel seguito:

- **RQ1**: ***Come varia il riuso in termini di ereditarietà di specifica, ereditarietà di implementazione, ed ereditarietà di delegazione nell'evoluzione del software?***
- **RQ2**: ***Come varia il riuso in termini di ereditarietà di specifica, ereditarietà di implementazione, ed ereditarietà di delegazione rispetto al numero di bug nell'evoluzione del software?***
- **RQ3**: ***Come varia il riuso in termini di ereditarietà di specifica, ereditarietà di implementazione, ed ereditarietà di delegazione rispetto alla densità dei bug nell'evoluzione del software ?***

3.4.1 Procedura di Analisi

Per fornire un'attenta risposta alle domande di ricerca RQ1, RQ2, RQ3 vi è l'esigenza di costruire un nuovo dataset partendo da ogni singolo dataset che si riferisce ad una determinata versione.

In particolare per rispondere alla research question RQ1 sono stati presi in considerazione i dati calcolati nella Tabella 3.4 a pagina 28 in cui vengono analizzate le variabili di somma e media per valutare il riuso in termini di ereditarietà di specifica, implementazione e delega.

Per rispondere alla domanda di ricerca RQ2 sono stati presi in considerazione e messi in relazione i dati calcolati e relativi al riuso presenti nella Tabella 3.4 a pagina 28 con il numero di bug calcolato per ogni versione e riportati nella Tabella 3.3 a pagina 25.

Inoltre per rispondere alla domanda di ricerca RQ3 sono stati messi in relazione i valori riportati in Tabella 3.4 a pagina 28 con i valori relativi alla densità dei bug mostrati in Tabella 3.3 a pagina 25

Successivamente alla creazione del DataSet costruito per tutti i sistemi (Codec, Math, Lang, Csv, Collections e Cli) è stato possibile plottare i dati ottenuti e fornire una rappresentazione grafica per rispondere al meglio alle Research Question, in modo da poter osservare in che modo le variabili considerate nelle RQ tendono ad evolvere da versione a versione. I risultati di tale analisi sono mostrati nel prossimo capitolo.

Infine per effettuare l'analisi della Research Question RQ1 e per capire al meglio come varia l'uso dell'ereditarietà di specifica, di implementazione e delegazione è stato effettuato il test Wilcoxon di Mann-Whitney, il quale ha permesso di confrontare e valutare se ci siano differenze statisticamente significative tra le variabili inerenti al riuso tra le diverse versioni. Si tratta di un test parametrico, e si è considerato un livello di confidenza del 95% come si fa tipicamente per analisi ti questo tipo. Quindi, osservando il valore del p-value restituito dal test si può decidere se rigettare o meno l'ipotesi nulla (assenza differenza tra le distribuzioni considerate). In particolare, nel nostro caso, se il valore del p-value risulta > 0.05 l'ipotesi nulla non può essere

rigettata mentre se risulta < 0.05 si può rigettare l'ipotesi nulla e si può affermare che le versioni considerate sono significativamente differenti [14]

3.5 Minacce alla validità

Nel seguito vengono discusse le possibili minacce che potrebbero influenzare i risultati ottenuti.

Costruct Validity
Una delle prime minacce da considerare per gli aspetti legati alla construct validity è dovuta alla strumentazione utilizzata per la raccolta dei dati. All'interno di questo studio per la raccolta dei dati sono stati utilizzati i seguenti strumenti:

- Defects4J un framework che offre una raccolta di bug riproducibili. Il quale ha permesso di rilevare misurazioni legate ai bug di ogni sistema analizzato. I risultati ottenuti dovrebbero essere attendibili, anche se c'è da considerare che il seguente strumento risulta in continuo aggiornamento per l'integrazione e la rimozione di nuovi bug all'interno della raccolta.

- Metrics, un sistema open-source, supplemento di Eclipse il quale dovrebbe fornire dei risultati ben testati e validi.

- InhMetrics, un plug-in in grado di ottenere misurazioni quantitative sugli aspetti relativi al riuso, il quale si presume che sia ben testato e che riproduca risultati affidabili e precisi.

C'è da sottolineare che Defects4J può influire maggiormente sugli aspetti legati alla costruct validity in quanto il framework risulta essere in continuo aggiornamento sul numero di bug deprecati.

Internal Validity

Le minacce relative all' Internal Validity riguardano i fattori che possano aver in qualche modo influenzato i risultati ottenuti. In particolare, nella domanda di ricerca RQ2 viene espresso il calcolo del numero dei bug presenti in una determinata versione, questo è soggetto a continui aggiornamenti da parte della fonte di Defects4J.
Un'altra minaccia relativa alla validità interna può essere caratterizzata dalla domanda di ricerca RQ3 in cui vengono messe in relazione le metriche di riuso con la densità derivata dal numero dei bug e dal numero di classi modificate, quest'ultimo calcolato in base alle informazioni ottenute dagli script offerti da Defects4J a seguito di un accurato e dettagliato studio.

Conclusion Validity

Una grave minaccia legata agli aspetti della Conclusion Validity può essere dovuta ai metodi statistici utilizzati. In particolare per rispondere alla domanda di ricerca RQ1 oltre all'osservazione dei plot creati per osservare come varia il riuso in termini di ereditarietà di specifica, implementazione e delegazione, è stato utilizzato il test Wilcoxon-Mann-Whitney il quale ha permesso in primo luogo di confrontare e di verificare l'attendibilità dei plot costruiti a seguito delle metriche calcolate e in secondo luogo ha permesso di evidenziare e quantificare un grado di significatività preciso e in questo modo di rigettare l'ipotesi nulla.

External Validity

Infine altre minacce sono legate al concetto di External Validity, le quali riguardano la generalizzazione dei risultati. Dallo studio condotto sono stati analizzate le versioni di 7 sistemi software, tutte diverse tra loro, presentate da Defects4J. Ovviamente per migliorare questo aspetto legrato alla generalizzazione dei risulati, è necessario introdurre nuove versioni in modo da poter arricchire il contenuto presentato, oppure estendere lo studio condotto con i restanti sistemi proposti da Defects4J e le future versioni.

CAPITOLO 4

Analisi dei risultati

In questo capitolo vengono riportati e discussi i risultati ottenuti per ogni research question considerata. In dettaglio vengono riportati i plot, inerenti alle differenti versioni e viene analizzato il riuso in termini di ereditarietà di specifica, implementazione e delegazione considerando le variabili Somma e Media dei seguenti progetti Math, Lang, CSV, Collections, Cli e Codec. Inoltre vengono riportati i risultati dell'analisi statistica effettuata analizzare statisticamente le differenze emerse con l'analisi dei plot.

4.1 Risultati RQ1

RQ1 - Come varia il riuso in termini di ereditarietà di specifica, ereditarietà di implementazione, ed ereditarietà di delegazione nell'evoluzione del software?
Di seguito sono riportati i risultati ottenuti analizzando i valori delle variabili di ereditarietà di specifica, di implementazione e delegazione. In Figura 4.1 nella pagina seguente sono mostrati i plot di tali variabili per ogni sistema software considerato nell'analisi.

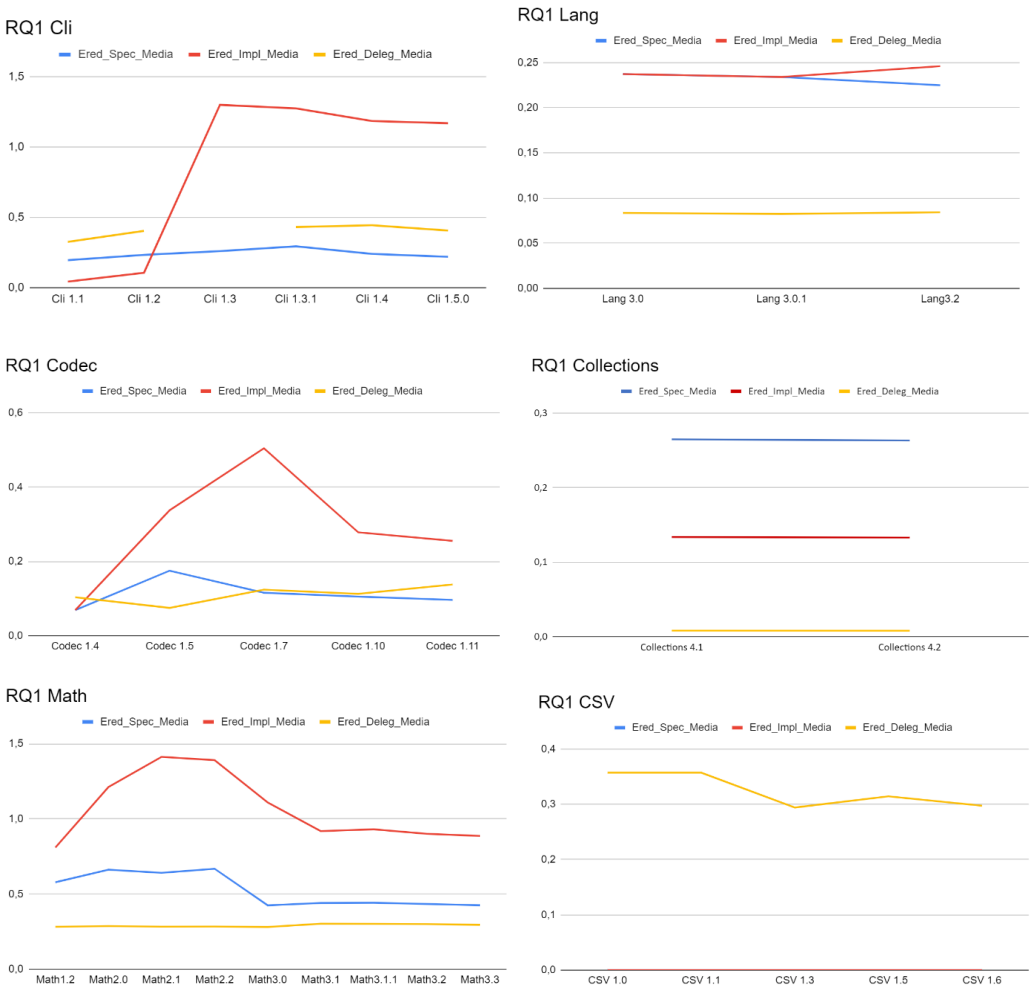

Figura 4.1: Plot valori Media di riuso per ogni sistema

Esaminando i valori della media calcolati e riportati in Figura 4.1 nella pagina precedente si evince che: per la gran parte dei sistemi il riuso espresso in termini di **ereditarietà di delegazione** tende a restare costante durante l'evoluzione dei sistemi. Ad eccezione dei sistemi Codec e CSV in cui in una parte iniziale la curva tende a diminuire per poi stabilizzarsi tra le successive versioni.

Il riuso espresso in termini di **ereditarietà di implementazione** ha un comportamento simile in 3 sistemi software, in particolare nei sistemi Math, Cli e Codec durante la loro evoluzione si può notare che la curva aumenta entro certi limiti per poi subire un lieve decremento fino ad arrivare a stabilizzarsi.
Andamento diverso per i sistemi CSV, Lang, Collections in cui la curva resta costante per tutto il periodo legato all'evoluzione dei sistemi.

Il riuso espresso in termini **di ereditarietà di specifica** per i sistemi Math, Codec e Cli ha un andamento simile, il quale evidenzia un leggero incremento nella parte iniziale, per poi subire una diminuizione fino ad arrivare ad una fase di stabilità. Comportamento diverso per i sistemi Collections, CSV e Lang in cui la curva risulta non destabilizzarsi per tutta la durata dell'evoluzione e quindi costante.

Dopo aver stabilito come le soglie inerenti al riuso variano graficamente, risulta opportuno analizzare le differenze tramite opportuni test statistici, confrontando le versioni considerate per ogni sistema. A questo proposito viene utilizzato il test Wilcoxon di Mann-Whitney. L'obiettivo è quello di verificare se le versioni considerate per ogni sistema risultano significativamente differenti tra loro.

Si ricorda che :

- se il valore **p-value risulta > 0.05** l'ipotesi nulla non può essere rigettata
- mentre se **p-value risulta < 0.05** si può rigettare l'ipotesi nulla e si può affermare che le versioni considerate sono significativamente differenti

Tabella 4.1: P-value ottenuuti eseguendo i test Mann-Whitney per confrontare il valore delle metriche di riuso per versioni successive del sistema Cli

Version System	Ered_Spec	Ered_Impl	Ered_Deleg
Cli 1.1 vs Cli 1.2	0,9643	0,2564	0,7525
Cli 1.2 vs Cli 1.3	0,8919	0,2562	0,8036
Cli 1.3 vs Cli 1.3.1	0,8252	0,8575	0,9576
Cli 1.3.1 vs Cli 1.4	0,6788	0,7136	0,8432
Cli 1.4 vs Cli 1.5.0	0,832	0,8709	0,7836

Dalla Tabella 4.1 è possibile osservare come per tutte le versioni del sistema Cli non esistono differenze significative in quanto il valore p-value ottenuto per l'ereditarietà di specifica, implementazione e delegazione non risulta in questo caso mai essere minore di 0.05.
Al contempo è possibile notare come **l'ereditarietà di specifica** tende a diminuire tra le versioni meno recenti a quelle più recenti, diminuzione non visibile a livello di plot in Figura 4.1 a pagina 37.

Mentre per l'**ereditarietà di implementazione** si conferma il risultato evidenziato in termini di plot, in quanto per le prime 2 versioni tende a rimanere costante per poi aumentare fino a stabilizzarsi.
Discorso simile che rispecchia il trend del grafico per la **delegazione** che tende leggermente ad aumentare tra le versioni meno recenti per poi stabilizzarsi.

In Tabella 4.2 vengono riportati i risultati del p-value calcolate mediante il test per il sistema Math.

Tabella 4.2: P-value ottenuuti eseguendo i test Mann-Whitney per confrontare il valore delle metriche di riuso per versioni successive del sistema Math

Version System	Ered_Spec	Ered_Impl	Ered_Deleg
Math 1.2 vs Math 2.0	0,8839	0,009229	0,9959
Math 2.0 vs Math 2.1	0,698	0,8632	0,9737

Table 4.2 continued from previous page

Math 2.1 vs Math 2.2	0,7322	0,5894	0,9976
Math 2.2 vs Math 3.0	0,0002484	0,03197	0,4641
Math 3.0 vs Math 3.1	0,7105	0,2015	0,4974
Math 3.1 vs Math 3.1.1	0,9742	0,8985	0,9869
Math 3.1.1 vs Math 3.2	0,7973	0,7273	0,8579
Math 3.2 vs Math 3.3	0,5786	0,7659	0,9695

In questo caso è possibile notare come tra le versioni confrontate esiste un netto cambiamento per il riuso espresso in termini di ereditarietà di specifica e implementazione tra le versioni considerate.

In dettaglio per l'**ereditarietà di specifica** è possibile notare come le versioni Math 2.2 e 3.0 si diversificano da tutte le altre. Infatti in questo caso avendo un p-value (evidenziato di giallo) < del 0.05 è possibile rigettare l'ipotesi nulla e affermare che tra le diverse versioni c'è una netta diminuzione dell'ereditarietà di specifica.

Per l'**ereditarietà di implementazione** discorso simile in quanto si può notare un valore piuttosto basso calcolato tra le versioni Math 1.2 e 2.0 del tutto diverso dai valori calcolati per le versioni successive.

Altro valore importante che permette di rigettare l'ipotesi nulla è stato ottenuto a seguito dell'analisi effettuata tra le versioni Math 2.0 e Math 3.0. Valore che suggerisce che sono state effettuate modifiche relative alle misure relative al riuso espresso in termini di ereditarietà di implementazione.

Mentre per la **delegazione** è possibile notare che le metriche variano da versione a versione non in maniera significativa.

In Tabella 4.3 nella pagina successiva vengono riportati i risultati del p-value calcolate mediante il test per il sistema Codec.

Tabella 4.3: P-value ottenuuti eseguendo i test Mann-Whitney per confrontare il valore delle metriche di riuso per versioni successive del sistema Codec

Version System	Ered_Spec	Ered_Impl	Ered_Deleg
Codec 1.4 vs Codec 1.5	0,07239	0,02819	0,5627
Codec 1.5 vs Codec 1.7	0,3291	0,69	0,4272
Codec 1.7 vs Codec 1.10	0,8275	0,02947	0,8009
Codec 1.10 vs Codec 1.11	0,8427	0,8428	0,5978

Dalla Tabella 4.3 è possibile notare che per il riuso espresso in termini di **ereditarietà di specifica** non sussiste una differenza significativa. Inoltre i valori calcolati tra le versioni rispecchiano il trend degli andamenti evidenziati tramite i plot in Figura 4.1 a pagina 37 riferito al sistema Codec4.1 a pagina 37.

Differenza notevole viene notata per il riuso espresso in termini di **ereditarietà di implementazione** tra le versioni Codec 1.4 e Codec 1.5 e le versioni Codec 1.7 e Codec 1.10 evidenziate in giallo in tabella 4.3. Valori che permettono di rigettare l'ipotesi nulla e di affermare un aumento del riuso espresso in termini di ereditarietà di implementazione. Andamento rispecchiato nel grafico riportato in Figura 4.1 a pagina 37 solo per le versioni del sistema Codec 1.7 e Codec 1.10 e non per le versioni 1.4 e 1.5.

Relativamente alla **delegazione**, è possibile notare che le metriche variano da versione a versione non in maniera significativa.

In Tabella 4.4 vengono riportati i risultati del p-value calcolate mediante il test eseguito sul il sistema Csv.

Tabella 4.4: P-value ottenuuti eseguendo i test Mann-Whitney per confrontare il valore delle metriche di riuso per versioni successive del sistema Csv

Version System	Ered_Spec	Ered_Impl	Ered_Deleg
Csv 1.0 vs Csv 1.1	NA	NA	1
Csv 1.1 vs Csv 1.3	NA	NA	0,6909

Table 4.4 continued from previous page

Csv 1.3 vs Csv 1.5	NA	NA	0,8412
Csv 1.5 vs Csv 1.6	NA	NA	0,9072

Dalla seguente tabella si può notare che per il riuso espresso in termini di **ereditarietà di specifica** ed **ereditarietà di implementazione** viene definito con la sigla NA in quanto per i risultati ottenuti non è stato possibile evidenziare differenze significative tra le versioni del sistema Csv. In particolare non è stato potuto rigettare l'ipotesi nulla a seguito dei risultati ottenuti.

Allo stesso modo per i valori della delegazione non è possibile affermare che ci sia una differenza significativa tra le versioni poichè il valore p-value risulta essere > 0.05 ma al contempo dalla versione Codec 1.1 alla versione Codec 1.6 si nota un leggero incremento non visibile sul plot di Figura 4.1 a pagina 37 riferito al sistema Csv.

In tabella 4.5 vengono riportati i risultati dei valori p-value calcolati mediante il test eseguito sul sistema Lang.

Tabella 4.5: P-value ottenuuti eseguendo i test Mann-Whitney per confrontare il valore delle metriche di riuso per versioni successive del sistema Lang

Version System	Ered_Spec	Ered_Impl	Ered_Deleg
Lang 3.0 vs Lang 3.0.1	0,9471	0,958	0,9685
Lang 3.0.1 vs Lang 3.2	0,8441	0,6843	0,8971

Per il sistema Lang il riuso espresso in termini di ereditarietà di implementazione, specifica e delegazione come è possibile notare in tabella 4.5 non permette di rigettare l'ipotesi nulla in quanto i valori calcolati sono tutti > 0.05.
Inoltre le andature rispecchiano il plot riportato in Figura 4.1 a pagina 37 riferito al sistema Lang.

Infine in tabella 4.6 nella pagina seguente vengono riportati i risultati del valore p-value calcolati mediante il test eseguito sul sistema Collections.

Tabella 4.6: P-value ottenuti eseguendo i test Mann-Whitney per confrontare il valore delle metriche di riuso per versioni successive del sistema Collections

Version System	Ered_Spec	Ered_Impl	Ered_Deleg
Collections 4.1 vs Collection 4.2	0,973	0,9524	0,98

Il riuso espresso in termini di ereditarietà di specifica, implementazione e delegazione non permette di rigettare l'ipotesi nulla, e rispecchia i trend del plot in Figura 4.1 a pagina 37 riferito al sistema Collections. Inoltre sarebbe utile analizzare più versioni del sistema considerato per poter effettuare aumentare il grado di precisione per il valore calcolato con il Test Wilcoxon di Mann-Whitney.

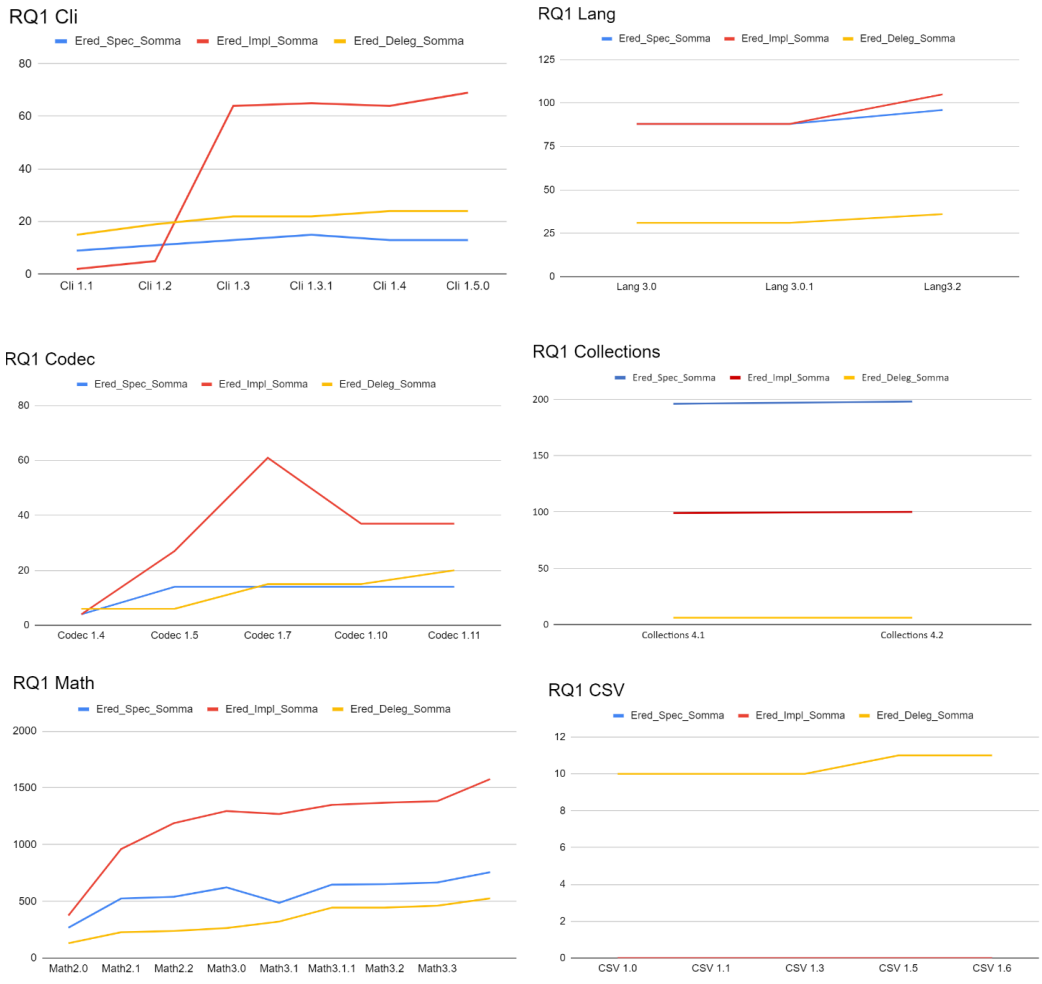

Figura 4.2: Plot valori Somma di riuso per ogni sistema

Esaminando i valori di somma, dai grafici presenti in ***Figura*** 4.2 si evince che:
Il riuso espresso in termini di **ereditarietà di implementazione** all'interno dei sistemi

Lang, Math e Cli segue un andamento crescente.
Per il sistema Codec la curva evidenzia un leggero incremento nella parte iniziale, per poi subire un diminuizione fino ad arrivare ad una fase di stabilità.
Mentre per i sistemi CSV e Collections risulta essere costante senza mai aumentare
Il riuso espresso in termini di **ereditarietà di delegazione** nella gran parte dei sistemi tende ad aumentare leggermente, ad eccezione del sistema Collections.
Il riuso espresso in termini di **ereditarietà di specifica** per i sitemi Lang e Math presenta un andamento crescente mentre per gli altri sistemi risulta stabile.
In generale per quanto riguarda i valori della somma possiamo dire che il riuso tende ad aumentare o a rimanere costante durante l'evoluzione del software.

4.2 Risultati RQ2

RQ2 - Come varia il riuso in termini di ereditarietà di specifica, ereditarietà di implementazione, ed ereditarietà di delegazione rispetto al numero dei bug nell'evoluzione del software?
Di seguito sono riportati i risultati ottenuti analizzando il valore del numero dei bug per ogni sistema. Successivamente per rispondere alla RQ2 vengono analizzati il numero dei bug in relazione ai valori della Media del riuso presentato nei plot in ***Figura*** 4.1 a pagina 37 e i valori della Somma del riuso presentati in ***Figura*** 4.2 nella pagina precedente.

Figura 4.3: Plot valori numero dei bug per ogni sistema

In primo luogo analizzando l'andamento dei bug in ***Figura*** 4.3 nella pagina precedente possiamo notare le differenze che presentato i vari sistemi.

In particolare per il sistema **CSV** si può notare che la versione **CSV 1.0** presenta una frequenza molto alta di bug che tende a diminuire nelle versioni successive fino ad arrivare quasi a stabilizzarsi, andatura diversa da tutti gli altri sistemi.

Andatura diversa viene presentata dal sistema **Cli** in cui tra le versioni **Cli 1.1** e **Cli 1.2** il numero di bug tende ad aumentare per poi avere una diminuzione notevole nella versione 1.5.1 ed infine stabilizzarsi tra le successive versioni.

Andatura ideale invece è rappresentata dal sistema **Collections** in cui si può notare che la versione **Collections 4.1** rappresenta un numero di bug elevato che tende a diminuire nella versione **4.2**, ciò suggerisce che tra le due versioni c'è stata una notevole attività di correzione dei bug.

Comportamento diverso viene rappresentato dai sistemi Codec e Math in cui viene rappresentato un andamento instabile tra le versioni difatti osservando il sistema Codec è possibile notare come l'andamento dei bug segue inizialmente un trend elevato per poi presentare una netta diminuzione a seguito di correzioni nella versione Codec 1.7 per poi aumentare di nuovo.

Infine comportamento del tutto diverso viene rappresentato dal sistema Lang in cui è possibile notare una diminuzione della curva relativa ai bug tra la versione 3.0 e 3.0.1 che infine tende a risalire nelle versioni successive a seguito dell'aggiunta di nuove funzionalità per il sistema.

In secondo luogo è bene analizzare e confrontare i plot relativi al numero di bug in ***Figura*** 4.3 a pagina 45 con i risulati dei valori di Media relativi al riuso in ***Figura*** 4.1 a pagina 37 e quelli di Somma in ***Figura*** 4.2 a pagina 43.

Analizzando l'andamento delle variabili di Media del riuso in figura ***Figura*** 4.1 a pagina 37 è possibile notare che esso non è caratterizzato dagli stessi cambiamenti e andamenti presentati dal numero dei bug proposti in ***Figura*** 4.3 a pagina 45.

Una osservazione può essere fatta per il sistema Cli in cui a partire dalla versione Cli 1.2 in ***Figura*** 4.1 a pagina 37 si può notare che l'ereditarietà di implementazione aumenta mentre in ***Figura*** 4.3 a pagina 45 il numero dei bug diminuisce, ciò suggerisce che a seguito di attività di risoluzione dei bug viene aumentato l'utilizzo dell'ereditarietà di implementazione.

Altra osservazione può essere fatta per il sistema Codec in cui e' possibile notare invece che dalla versione 1.5 alla versione 1.7 c'è una diminuzione del numero dei bug seguita da una diminuzione dell'ereditarietà di specifica e un aumento dell'ereditarietà di implementazione e delegazione, ciò suggerisce che ancora una volta a seguito della diminuzione dei bug aumenta l'utilizzo dell'ereditarietà di implementazione

Infine mettendo in relazione il numero di bug con le variabili di Somma del riuso rappresentate in ***Figura*** 4.2 a pagina 43 è possibile notare lo stesso comportamento dell'ereditarietà di implementazione appena descritto per la Media per i sistemi Cli e Codec, mentre i restanti sistemi seguono andamenti altalenanti sia per il riuso espresso in termini di ereditarietà di specifica che di delegazione.

4.3 Risultati RQ3

RQ3 - Come varia il riuso in termini di ereditarietà di specifica, ereditarietà di implementazione, ed ereditarietà di delegazione rispetto alla densità dei bug nell'evoluzione del software ?

Di seguito sono riportati i risultati ottenuti analizzando in primis il valore della densità dei bug, e successivamente tale valore viene messo in relazione con le variabili di Media e Somma del riuso espresso in termini di ereditarietà di specifica, di implementazione e delegazione rispetto alla densità dei bug. In ***Figura*** 4.4 sono mostrati i plot di tali variabili per ogni sistema software considerato.

Figura 4.4: Plot valori densità dei bug per ogni sistema

Considerando i valori della densità dei bug, calcolati a seguito del rapporto tra il numero di classi coinvolte e il numero dei bug, rappresentati in ***Figura*** 4.4 nella pagina precedente è possibile notare come le distribuzioni dei bug sono del tutto diverse tra di loro.

Per il sistema Collections si può notare che il rapporto tra il numero di classi coinvolte e il numero di bug sembra essere distribuito in maniera omogenea tra le versioni.

Analizzando invece la Media del riuso i cui valori sono presentati in ***Figura*** 4.1 a pagina 37 e confrontondola con la densità dei bug è possibile notare che anche in questo caso per il sistema Collections viene rispettato il trend descritto in ***Figura*** 4.4 nella pagina precedente.

Una osservazione può essere fatta per il sistema Cli in cui a partire dalla versione Cli 1.2 la densità dei bug tende a diminuire mentre l'ereditarietà di implementazione tende ad aumentare, ciò suggerisce che a seguito della diminuzione del rapporto tra il numero di classi e numero dei bug, l'ereditarietà di implementazione aumenta.

Altra osservazione simile può essere fatta per il sistema Codec per il riuso espresso in termini di ereditarietà di specifica e implementazione, in cui è possibile notare che dalla versione Codec 1.5 la curva che rappresenta la densità dei bug tende a diminuire di conseguenza il rapporto tra numero di classi e bug tende ad abbassarsi a seguito della risoluzione dei bug mentre l'ereditarietà di specifica e implementazione aumenta per poi stabilizzarsi.

Mentre per gli altri sistemi l'andamento del riuso non è caratterizzato dagli stessi cambiamenti e alterazioni rappresentanti dalla densità dei bug.

Infine considerando i valori della Somma riferiti all'ereditarietà di specifica, delegazione e implementazione rispetto alla densità (calcolata a seguito del numero di bug / numero di classi coinvolte), vengono rispettati in gran parte gli stessi trend calcolati per il riuso espresso in Media.

CAPITOLO 5

Conclusioni e Sviluppi Futuri

Nel seguente capitolo vengono presentate le conclusioni relative allo studio condotto, e vengono discussi possibili studi futuri.

5.1 Conclusioni

Relativamente alla prima domanda di ricerca (RQ1), per **l'ereditarietà di implementazione** focalizzando l'attenzione sui 3 sistemi che presentano un maggior numero di versioni, come Cli, Math e Codec, dai plot riportati in Figura 4.1 a pagina 37 e osservando i valori ottenuti dal Test Wilcoxon Mann Whitney mostrati nelle Tabelle 4.1 a pagina 39, 4.3 a pagina 41 e 4.2 a pagina 39 è possibile notare come questo meccanismo di riuso segue un trend in aumento tra le versioni (in alcuni casi sussistono differenze significative) ciò suggerisce che gli sviluppatori tendono ad utilizzare questi meccanismi con maggior frequenza.

Per quanto riguarda **l'ereditarietà di specifica** si può osservare che segue un andamento stabile nel tempo per la maggior parte dei sistemi. Una leggera eccezione può essere osservata per le versioni Math 2.2 e Math 3.0 in Tabella 4.2 a pagina 39, in cui è stato notata una netta diminuzione dell'utilizzo di tale meccanismo, con differenze

significative tra le versioni come evidenziato dai risultati del Test Wilcoxon Mann Whitney.

Infine il riuso in termini di **delegazione** segue un andamento stabile tra le versioni ed in particolare l'analisi statistica condotta ha evidenziato che i risultati ottenuti per le versioni analizzate non si differenziano in modo statisticamente significativo.

Per la Research-Question RQ2 dopo aver analizzato i valori di Media del riuso presenti nei plot di Figura 4.1 a pagina 37 e i valori di Somma del riuso presentati nel plot in Figura 4.2 a pagina 43, messi entrambi in relazione con il plot riferito al numero dei bug per ogni sistema riportato in Figura 4.3 a pagina 45, è stato possibile evidenzare che per alcuni sistemi le variazioni nel numero di bug non erano conformi agli andamenti delle variabili di Somma e Media relativi al riuso espresso in termini di ereditarietà di specifica, implementazione e delega.

Inoltre, per i sistemi con più versioni, in particolare Cli e Codec, è stato possibile notare che alla diminuzione del numero dei bug tra le versioni il riuso espresso in termini di ereditarietà di implementazione e delegazione aumenta. Ciò suggerisce di indagare su quanto concluso in quanto la correzione dei bug porterebbe ad un maggior utilizzo dell'ereditarietà di implementazione e delegazione.

Per la RQ3, mettendo in relazione i valori delle variabili di Somma e Media del riuso con la densità del numero dei bug presentati in Figura 4.4 a pagina 48 è stato possibile notare un comportamento simile a quanto riscontrato nell'analisi effettuata per rispondere alla Research-Question RQ2 per i sistemi che presentano più versioni. Difatti a seguito della correzione dei bug è stata identificata una diminuzione della densità dei bug seguita da un aumento dell'ereditarietà di implementazione.

Si vuole anche far notare che per gli altri sistemi non è stato possibile specificare una relazione del riuso rispetto alla densità dei bug in quanto gli andamenti sono risultati altalenanti, molto probabilmente dovuto ad un numero ridotto di versioni analizzate.

Riassumento, possiamo osservare che il riuso espresso in termini di ereditarietà di implementazione risulta essere maggiormente utilizzato rispetto al riuso espresso

in termini di ereditarietà di specifica e delegazione. Inoltre i valori delle metriche relative alle 3 variabili di riuso mostrano come l'ereditarietà di implementazione tende ad aumentare da versione a versione.

Nello specifico per avere la prova di quanto osservato a livello grafico nei plot in Figura 4.1 a pagina 37 e 4.2 a pagina 43, sono stati eseguiti i test Wilcoxon di Mann-Whitney per effettuare un confronto tra le diverse versioni. Tale analisi ha permesso di evidenziare differenze significative che riguardano maggiormente il riuso espresso in termini di ereditarietà di implementazione. Inoltre per un solo sistema sono state evidenziate differenze significative in termini di ereditarietà di specifica, ossia per il sistema Math presentato in Tabella 4.2 a pagina 39. Infine il riuso in termini di delegazione non ha presentato differenze significative.

Inoltre mettendo a confronto il riuso con il numero dei bug (così come con la densità dei bug) di ogni sistema, è stato possibile evidenziare che esiste una correlazione tra il riuso espresso in termini di ereditarietà di implementazione e il numero dei bug. In quanto i sistemi analizzati con più versioni come Cli e Codec, hanno presentato un aumento della curva del riuso espresso in termini di ereditarietà di implementazione a seguito della risoluzione del numero dei bug. Ciò suggerisce che il riuso espresso in termini di ereditarietà di implementazione è correlato al numero dei bug presenti nei sistemi.

5.2 Sviluppi futuri

Il seguente lavoro potrebbe essere esteso considerando i restanti sistemi presentati da Defects4J, che prevede continui aggiornamenti sui bug presentati e sui sistemi, in modo da poter ampliare l'analisi su un numero maggiore di sistemi, e quindi permettere di ottenere maggiori informazioni sia sui bug presentati da Defects4J ma anche sulla predizione dei difetti.

Altro aspetto su cui focalizzare l'attenzione è quello di analizzare più versioni per i sistemi che hanno presentato dei limiti nel calcolo del riuso espresso in termini di ereditarietà di specifica, delegazione e implementazione, in modo da poter osservare se esistono differenze significative legate al riuso e se queste impattano in modo significativo sulla qualità del software.

Bibliografia

[1] Troy Dimes. *Programmer en Java*. Babelcube Inc., 2015. (Citato a pagina 3)

[2] Annalisa Binato, Alfonso Fuggetta, and Laura Sfardini. *Ingegneria del software. Creatività e metodo*. Pearson Education, 2006. (Citato a pagina 4)

[3] Ian Sommerville. *Engineering software products*. Pearson, 2020. (Citato a pagina 5)

[4] Giammaria Giordano, Antonio Fasulo, Gemma Catolino, Fabio Palomba, Filomena Ferrucci, and Carmine Gravino. On the evolution of inheritance and delegation mechanisms and their impact on code quality. In *Proceedings of SANER 2022*, pages 1–12, 2022. (Citato alle pagine 6 e 17)

[5] Timothy Christian Lethbridge and Robert Laganiere. *Object-oriented software engineering*, volume 11. McGraw-Hill New York, 2005. (Citato a pagina 9)

[6] Strategic Planning. The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology*, page 1, 2002. (Citato a pagina 11)

[7] Maurice Dawson, Darrell Norman Burrell, Emad Rahim, and Stephen Brewster. Integrating software assurance into the software development life cycle (sdlc). *Journal of Information Systems Technology and Planning*, 3(6):49–53, 2010. (Citato a pagina 11)

[8] Lutz Prechelt, Barbara Unger, Michael Philippsen, and Walter Tichy. A controlled experiment on inheritance depth as a cost factor for code maintenance. *Journal of Systems and Software*, 65(2):115–126, 2003. (Citato a pagina 16)

[9] John Daly, Andrew Brooks, James Miller, Marc Roper, and Murray Wood. The effect of inheritance on the maintainability of object-oriented software: an empirical study. In *Proceedings of International Conference on Software Maintenance*, pages 20–29. IEEE, 1995. (Citato a pagina 16)

[10] Brij Mohan Goel and Pradeep Kumar Bhatia. Analysis of reusability of object-oriented systems using object-oriented metrics. *ACM SIGSOFT Software Engineering Notes*, 38(4):1–5, 2013. (Citato a pagina 16)

[11] Sonia Chawla and Rajender Nath. Evaluating inheritance and coupling metrics. *International Journal of Engineering Trends and Technology (IJETT)*, 4(7):2903–2908, 2013. (Citato a pagina 16)

[12] Antonio Fasulo. *Tesi Magistrale, RIUSO DEL SOFTWARE RIUSO DEL SOFTWARE IN TERMINI DI EREDITARIETÀ DI IMPLEMENTAZIONE, DI SPECIFICA E DELEGAZIONE: UNA VALUTAZIONE DELL'IMPATTO SULLA QUALITÀ DEL SOFTWARE NELLA SUA EVOLUZIONE*. 2020/2021. (Citato a pagina 17)

[13] Brian Henderson-Sellers. *Object-oriented metrics: measures of complexity*. Prentice-Hall, Inc., 1995. (Citato a pagina 29)

[14] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012. (Citato a pagina 33)

Ringraziamenti

Per concludere è doveroso dedicare uno spazio del mio elaborato alle persone che hanno contribuito, con il loro instancabile supporto, alla realizzazione dello stesso.

Ringrazio i miei genitori punto di riferimento i quali mi hanno sostenuto emotivamente durante tutto questo percorso di studi, grazie per esserci stati soprattutto nei momenti di sconforto e per avermi donato ogni giorno i valori indispensabili delle vostre vite.

Ringrazio te mamma, per essere sempre presente senza chiedere nulla e per avermi cresciuto ed educato con amore e affetto incondizionato, insegnandomi a dimostrare il bene, l'altruismo e l'amore per il prossimo. Sei una donna forte e speciale, su cui ho sempre contato, mi hai sostenuto nei momenti difficili con semplici parole, consigli e sorrisi di cui faccio tesoro.

Ringrazio te papà, che prima di essere un grande padre, sei un grande essere umano. La tua forza, determinazione e costanza mi hanno reso una grande persona. Mi hai insegnato a non mollare mai e a trovare soddisfazione nel sacrificio. Ogni persona ha un pilastro, qualcuno che nonostante tutti gli imprevisti della vita riesce a trasmettere sicurezza e stabilità, quella persona per me sei tu.

Ringrazio te piccola Giulia, che nonostante la tua tenera età, hai saputo donarmi la leggerezza di cui avevo bisogno, magari assorbendo il più delle volte i miei stati d'animo, sei la sorellina che ho sempre desiderato.

Ringrazio Raissa, persona per me speciale, con cui ho condiviso l'intero percorso universitario. Grazie per avermi supportato in qualsiasi momento, grazie a te ho capito l'importanza di avere una compagna accanto, presente in ogni istante della mia vita, pronta a risollevarmi quando ce ne fosse bisogno. Grazie a te che hai reso ostacoli insormontabili, piccole barriere.

Ringrazio Anna, Raffaele, Vincenza e Salvatore, per avermi accolto nella vostra famiglia nel modo più semplice e generoso possibile, senza avermi mai fatto sentire di troppo e per aver riservato sempre una parola di conforto quando ce n'è stato bisogno.

Ringrazio Giuseppe, persona con la quale sin da subito sono entrato in sintonia, Grazie per tutti i piccoli consigli e per aver avuto sempre una parola di incoraggiamento.

Ringrazio gli amici del gruppo "Amicizia Pesante", per aver avuto sempre un pensiero per me, e per avermi regalato giornate e serate piene di serenità, semplicità e spensieratezza.

Infine ringrazio gli amici di sempre e tutte le persone che sono state presenti, avendo un piccolo pensiero per me, le quali inconsapevolmente hanno contribuito al raggiungimento di questo piccolo traguardo.