In che modo la co-presenza di Design Pattern e Code Smell influisce sulla comprensione degli sviluppatori durante la manutenzione del software?

HW3 - Metriche e Qualità del Software - A.A 2022/23

Delogu Nicolò, Mazza Dario

***Sommario*—Presentiamo i risultati di un esperimento controllato che è sono stato condotto per valutare l'effetto della co-presenza di code smells e di design patterns sulla comprensione e la manutenzione del codice sorgente. L'esperimento ha coinvolto studenti magistrali in Informatica presso l'Università di Salerno. I partecipanti all'esperimento hanno svolto un compito di comprensione e di manutenzione con e senza code smells e design patterns all'interno del codice sorgente. L'analisi dei dati ha rivelato che i partecipanti che trattavano classi solo con istanze di design pattern hanno ottenuto prestazioni migliori nel task di comprensione rispetto ai partecipanti che hanno trattato classi solo con code smells, con entrambi o con niente. D'altro canto, i dati ottenuti dal task di manutenzione hanno mostrato che i gruppi, che hanno trattato classi con solo DP o con code smells e DP, hanno ottenuto un punteggio peggiore rispetto ai gruppi che hanno manipolato classi con solo code smells o niente.**

I. Definizione

A. Introduzione

La manutenzione è una fase importante per i sistemi software sviluppati utilizzando qualsiasi modello del ciclo di vita del software e linguaggio di programmazione. La manutenzione del software è necessaria per garantire che un sistema continui a soddisfare i requisiti degli utenti. Indipendentemente dall'operazione di manutenzione richiesta (ad esempio, correttiva, perfezionativa e adattiva), gli addetti alla manutenzione devono dedicare un considerevole tempo alla lettura e comprensione del codice sorgente scritto da altri. La presenza di design patterns e l'assenza di code smells dovrebbero fornire un miglior supporto allo svolgimento delle operazioni di manutenzione, riducendo così lo sforzo necessario e influenzando positivamente l'efficienza con cui gli addetti svolgono tali operazioni.

Nel campo dello sviluppo Object Oriented, i design pattern sono riconosciuti come un mezzo che "può migliorare la documentazione e la manutenzione dei sistemi esistenti, fornendo una specifica esplicita delle interazioni tra classi e oggetti e della loro intenzione sottostante" mentre i code smells sono definiti come "serie di caratteristiche che il codice sorgente può avere e che sono generalmente riconosciute come probabili indicazioni di un difetto di programmazione". I code smells non sono e non rivelano "bug", cioè veri e propri errori, bensì debolezze di progettazione che riducono la qualità del software stesso, a prescindere dall'effettiva correttezza del suo funzionamento.

In questo articolo, presentiamo un esperimento controllato per valutare se la co-presenza di design pattern e code smells impatti sulla comprensione del codice sorgente da parte degli sviluppatori durante l'attività di manutenzione.

B. Obiettivo

Seguendo il paradigma GQM (Goal Question Metrics), forniamo una descrizione del goal dell'esperimento e dei suoi dettagli specifici:

> *L'obiettivo del nostro studio è analizzare la co-presenza di design pattern e code smells allo scopo di valutarli rispetto al loro potenziale impatto sulla comprensione del codice sorgente nelle attività di manutenzione dal punto di vista degli sviluppatori, nel contesto dell'evoluzione e della manutenzione del software.*

II. Pianificazione

A. Definizione del contesto

Abbiamo condotto l'esperimento all'interno di un laboratorio presso l'Università di Salerno (Italia) con 20 studenti del corso di laurea magistrale in *Software Engineering & IT Management*. L'esperimento rappresentava un'attività opzionale dei corsi di *Software Engineering for Artificial Intelligence* e di *Metriche e Qualità del Software*.

1) Soggetti: Tutti i partecipanti possedevano abilità medio-alte di comprensione del codice sorgente sviuppato in Java mentre le loro competenze di manutenzione di codice Java sono risultate di livello medio (solo il 55% ha dichiarato di essere confidente nelle proprie abilità). Il 65% dei partecipanti era familiare con il concetto di *code smell* mentre il rimanente 35% aveva unicamente sentito nominare l'argomento senza averne nozioni specifiche. Solo il 15% dei partecipanti aveva esperienze lavorative pregresse. I partecipanti erano studenti iscritti a un programma di laurea magistrale in Informatica presso l'Università di Salerno. Possono essere considerati non lontani dagli ingegneri del software neofiti o dai programmatori junior.

Per quanto concerne l'applicazione di design patterns a progetti universitari/lavorativi, il 65% dei partecipanti ha dichiarato di utilizzarli con una frequenza variabile mentre il rimanente 35% esclusivamente se richiesto.

2) Oggetti sperimentali: Abbiamo selezionato diverse classi con funzionalità distinte di progetti open source che includevano: (i) istanze di design patterns, (ii) istanze di code smells, (iii) una combinazione dei due e (iv) nessuno dei due elementi. Per ogni oggetto sperimentale abbiamo considerato: il tipo di code smell, design pattern o una combinazione di entrambi, la possibilità di isolare la classe, il numero di LOC, il progetto di appartenenza, il codice sorgente della classe, il compito della classe e le risposte corrette relative ai task di compresione e di manutenzione.

B. Formulazione delle ipotesi

Abbiamo definito e testato le seguenti ipotesi nulle:

Hn0

La co-presenza di design patterns e code smells non ha impatto sulla comprensione del codice sorgente nelle attività di comprensione.

Hn1

La co-presenza di design patterns e code smells non ha impatto sulla comprensione del codice sorgente nelle attività di manutenzione.

e la seguenti ipotesi alternative:

Ha0

La co-presenza di design patterns e code smells impatta negativamente o positivamente sulla comprensione del codice sorgente nelle attività di comprensione.

Ha1

La co-presenza di design patterns e code smells impatta negativamente o positivamente sulla comprensione del codice sorgente nelle attività di manutenzione.

C. Scelta delle variabili

Le variabili **dipendenti** che abbiamo individuato sono il livello di *comprensione* del codice sorgente da parte dei partecipanti (facilmente deducibile dall'ipotesi) e *l'effort*, ovvero il tempo, espresso in minuti, impiegato per fornire una risposta ai singoli task. Per valutare quantitativamente la variabile di comprensione, abbiamo chiesto loro di compilare un questionario per ciascun oggetto sperimentale. Questo questionario consisteva in 8 domande a scelta multipla, ciascuna composta da due parti e con lo stesso numero di possibili risposte, di cui solo una corretta. Inoltre ogni al termine di ogni risposta, ad ogni partecipante è stato richiesto di indicare il grado di confidenza nella risposta appena data.

Per valutare quantitativamente la comprensione raggiunta da ciascun partecipante, abbiamo adottato un approccio basato sul recupero delle informazioni. Nello specifico, abbiamo utilizzato il rapporto tra il numero di risposte corrette e il numero di risposte totali per valutare la performance di ogni soggetto sia nel task di comprensione, sia in quello di manutenzione:

$$rapporto_s = \frac{\sum_{i=1}^{n}(answer_{s,i} \cap correct_i)}{n}$$

dove $answer_{s,i}$ indica l'insieme degli elementi stringa forniti come risposta alla domanda i dal partecipante s e $correct_i$ indica l'insieme noto di elementi corretti attesi per la domanda i e n indica il numero totale di domande a cui il soggetto ha risposto.

Infine per valutare quantitativamente l'effort relativo ad ogni soggetto, abbiamo calcolato la media dei tempi di risposta alle singole domande per entrambi i task:

$$effort_s = \frac{\sum_{i=1}^{n}(end_{s,i} - start_{s,i})}{n}$$

dove $start_{s,i}$ indica l'orario in cui il soggetto s ha iniziato a rispondere alla domanda i e $end_{s,i}$ indica l'orario in cui il soggetto s ha finito di rispondere alla domanda i e n indica il numero totale di domande a cui il soggetto ha risposto.

D. Design dell'esperimento

Abbiamo utilizzato un design a un fattore con quattro trattamenti: con best practice (Design Pattern), con bad practice (Code Smell), con niente e con entrambi.

- Gruppo A: Classi in cui co-occorrono Design Pattern e Code Smell
- Gruppo B: Classi in cui ci sono solo Design Pattern
- Gruppo C: Classi in cui ci sono solo Code Smell
- Gruppo D: Classi in cui non ci sono né Design Pattern né Code Smell

III. Svolgimento

L'esperimento è stato eseguito presso l'Università degli studi di Salerno (sede di Fisciano), all'interno di uno dei laboratori universitari. La durata è stata di circa due ore, tuttavia non era è determinato alcun tempo limite entro cui svolgere i task in modo da mitigare eventuali pressioni provate dai partecipanti. Inoltre, per fare fronte alla minaccia dell'affaticamento, è stata effettuata una pausa di dieci minuti a metà dello svolgimento, durante la quale i soggetti non hanno potuto comunicare tra loro e non hanno avuto la possibilità di uscire dal laboratorio. Prima di svolgere i task, uno dei controllori ha spiegato ai partecipanti la struttura delle domande e degli oggetti sperimentali. Per svolgere i compiti di comprensione e di manutenzione, ogni partecipante ha utilizzato il proprio laptop. Abbiamo chiesto loro di utilizzare la seguente procedura sperimentale per ogni gruppo di domande: (i) specificare l'orario di inizio; (ii) rispondere alle domande su un form online, esaminando il codice sorgente (tramite un

editor di testo); e (iii) segnare l'orario di fine. Al termine della sperimentazione, ogni partecipante ha ricevuto, in segno di riconoscimento, un buono Amazon da uitilizzare a fini personali (i soggetti non sapevano a priori di questa scelta).

IV. Analisi e interpretazione

Per investigare le ipotesi nulle, abbiamo adottato test non parametrici a causa della dimensione del campione, relativamente ridotta, e soprattutto della non normalità dei dati, verificata mediante l'utilizzo del test di Shapiro-Wilk. In particolare, abbiamo utilizzato il test di Kruskal-Wallis a causa del design degli esperimenti poichè solo analisi non accoppiate sono possibili. Questo test statistico non parametrico confronta le posizioni centrali (mediane) di tre o più gruppi indipendenti. Esso viene utilizzato per determinare se almeno uno dei gruppi è statisticamente diverso dagli altri, non richiede l'assunzione di normalità dei dati e si basa sui ranghi dei valori osservati. Se il valore p associato al test è inferiore al livello di significatività scelto, si conclude che esistono differenze statisticamente significative tra i gruppi.In tutti i test statistici eseguiti, abbiamo deciso (come consuetudine) di accettare una probabilità del 5% di commettere un Errore di Tipo I, ovvero di respingere un'ipotesi nulla quando è effettivamente vera.

Le Tabelle I, II, III, IV mostrano rispettivamente alcune statistiche descrittive (ovvero mediana, media e deviazione standard) dei vari gruppi, suddivise per Effort e Comprensione per entrambi i task. Queste statistiche mostrano che il gruppo B, ovvero il trattamento Best Practice, e il gruppo C, ovvero il trattamento Worst Practice, hanno raggiunto in media un Rapporto_Comprensione piu alto rispetto ai gruppi A e D (il gruppo B ha performato leggermente meglio di C). Inoltre è possibile notare che l'Effort_Medio_Comprensione risulta minore per i partecipanti dei gruppi B e C che hanno dunque, in media, impiegato meno tempo a svolgere il relativo task rispetto a quelli dei gruppi A e D. Tenendo conto di questi risultati, abbiamo dedotto che il motivo di queste differenze possa essere spiegato dalle seguenti osservazioni:
(i) *la co-presenza di design patterns e di code smells nel codice sorgente (gruppo A) potrebbe introdurre un livello di complessità maggiore e di leggibilità minore rispetto a quelli che si potrebbero rilevare con la singola presenza di design patterns o code smells* ; (ii) *è noto che la presenza di design pattern nel codice sorgente, se correttamente individuati, rendano il codice più leggibile e più facile da comprendere per gli sviluppatori. Dunque la loro assenza (nei gruppi C e D) potrebbe, viceversa, rendere il codice meno leggibile e più difficile da comprendere*.

Per quanto concerne il Rapporto_Manutenzione, abbiamo individuato uno scenario opposto a quello precedente. Il gruppo C è stato quello che ha svolto meglio, in media, il task di manutenzione, con un distacco significativo rispetto ai gruppi A e B. Questo risultato, inizialmente controinuitivo, potrebbe essere chiarito con la seguente considerazione: *se il soggetto che legge il codice è consapevole dei design pattern ed è in grado di applicarli nella pratica (non solo conoscenza teorica, utile nella loro identificazione), allora essi potrebbero effettivamente facilitare la manutenibilità del codice. Se però queste premesse non dovessero essere vere, allora il soggetto potrebbe incontrare delle difficoltà inattese in un task che richiede la modifica di una specifica occorrenza di DP.* Questa stessa spiegazione potrebbe essere valida per motivare anche la disparità dell'Effort_Medio_Manutenzione tra i gruppi. Analogalmente al caso precedente i gruppi A e B hanno, in media, impiegato più tempo a svolgere il task di manutenzione rispetto ai gruppi C e D. È importante notare che i risultati ottenuti nella fase di manutenzione potrebbero essere stati influenzati dal fatto che gli stessi oggetti sperimentali sono stati utilzzati sia nella fase di comprensione (svolta per prima) che nella fase di modifica. Questo potrebbe aver creato un effetto di familiarità che potrebbe aver contribuito alle differenze prestazionali osservate tra i gruppi.

Tabella I: Statistiche descrittive gruppo A

Variabile Dipendente	Mediana	Media	Dev. Std.
Rapporto_Comprensione	0.750	0.7083	0.06454972
Rapporto_Manutenzione	0.6250	0.5833	0.1290994
Effort_Medio_Comprensione	4.500	4.792	1.685724
Effort_Medio_Manutenzione	5.500	5.500	1.196349

Tabella II: Statistiche descrittive gruppo B

Variabile Dipendente	Mediana	Media	Dev. Std.
Rapporto_Comprensione	0.875	0.900	0.0559017
Rapporto_Manutenzione	0.750	0.675	0.2270738
Effort_Medio_Comprensione	2.625	3.250	1.427957
Effort_Medio_Manutenzione	3.875	4.550	1.788854

Tabella III: Statistiche descrittive gruppo C

Variabile Dipendente	Mediana	Media	Dev. Std.
Rapporto_Comprensione	0.8571	0.8571	0.1166424
Rapporto_Manutenzione	1.0000	0.9286	0.1428571
Effort_Medio_Comprensione	2.929	2.571	0.9828461
Effort_Medio_Manutenzione	4.214	3.929	1.301491

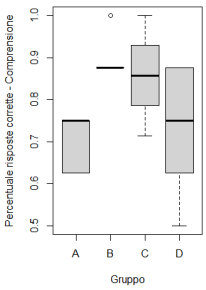
Tabella IV: Statistiche descrittive gruppo D

Variabile Dipendente	Mediana	Media	Dev. Std.
Rapporto_Comprensione	0.750	0.725	0.1629801
Rapporto_Manutenzione	0.875	0.850	0.0559017
Effort_Medio_Comprensione	5.625	6.050	2.750852
Effort_Medio_Manutenzione	3.375	3.750	1.011651

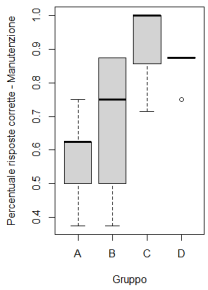
La Figura 1 mostra i boxplot per la percetuale di risposte corrette raggiunta da ogni ogni gruppo per entrambi i task dell'esperimento.

La Figura 2 mostra i boxplot per l'effort, espresso in minuti, impiegato da ogni ogni gruppo per entrambi i task dell'esperimento.

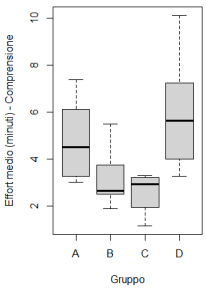
Entrambe le figure permettono di visualizzare graficamente quanto descritto precedentemente. Gli outlier presenti nei boxblot si riferiscono a singole occorrenze all'interno dei gruppi. In particolare identificano i soggetti che, rispetto alla distribuzione tra il primo e il terzo quartile, hanno raggiunto un punteggio piu alto o piu basso (Gruppi B - D, Figura



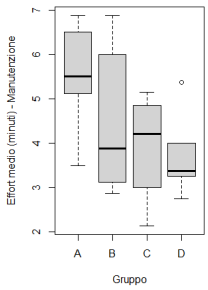
(a) Rapporto_Comprensione



(b) Rapporto_Manutenzione



(a) Effort_Comprensione



(b) Effort_Manutenzione

1) e che hanno impiegato piu tempo per svolgere il task di manutenzione (Gruppo D, Figura 2).

A. Studio della normalità

Le Tabelle V, VI, VII e VIII mostrano i risultati dello studio sulla normalità dei dati per ogni gruppo. In paricolare mostriamo il $p - value$ e il $test\_statistic$ per ogni variabile dipendente. Nonostante i gruppi A - B e C - D siano normalmente distribuiti rispettivamente per Rapporto_Comprensione e Rapporto_Manutenzione, i dati sono, complessivamente, non normali (abbiamo considerato una distribuzione normale se il suo $p - value < 0.05$). Questo fatto ci ha portato a testare le ipotesi nulle descritte nella Sezione II.B con test statistici non parametrici.

Tabella V: Test normalità dei dati gruppo A

Variabile Dipendente	P-value	Test statistic
Rapporto_Comprensione	0.001350753	0.6398937
Rapporto_Manutenzione	0.4732709	0.915459
Effort_Medio_Comprensione	0.5222842	0.922328
Effort_Medio_Manutenzione	0.69746	0.944726

Tabella VI: Test normalità dei dati gruppo B

Variabile Dipendente	P-value	Test statistic
Rapporto_Comprensione	0.0001309782	0.5521817
Rapporto_Manutenzione	0.2538465	0.8668359
Effort_Medio_Comprensione	0.4160075	0.9011022
Effort_Medio_Manutenzione	0.3005662	0.8780496

Tabella VII: Test normalità dei dati gruppo C

Variabile Dipendente	P-value	Test statistic
Rapporto_Comprensione	0.6829615	0.9446644
Rapporto_Manutenzione	0.001240726	0.6297763
Effort_Medio_Comprensione	0.149366	0.8226218
Effort_Medio_Manutenzione	0.6284697	0.93573

Tabella VIII: Test normalità dei dati gruppo D

Variabile Dipendente	P-value	Test statistic
Rapporto_Comprensione	0.4211497	0.9020198
Rapporto_Manutenzione	0.0001309782	0.5521817
Effort_Medio_Comprensione	0.7154719	0.9469613
Effort_Medio_Manutenzione	0.4430428	0.9058475

B. Test delle ipotesi

I risultati del test di Kruskal-Wallis sono riassunti nella Tabella IX, in cui riportiamo il $p - value$ e il $test\_statistic$ per ogni variabile dipendente. Essi mostrano che, complessivamente, esiste una differenza significativa per Rapporto_Comprensione, Rapporto_Manutenzione e Effort_Medio_Comprensione (poichè il loro $p - value < 0.05$) e dunque possiamo rigettare le ipotesi nulle *Hn0* e *Hn1* per queste variabili dipendenti. Assieme ai dati descritti nella Sezione IV possiamo dedurre che sommariamente: *la co-presenza di design pattern e code smells ha impattato negativamente sulla comprensione del codice sorgente nelle attività (i) di manutenzione, (ii) di comprensione e (iii) sull'effort richiesto per svolgere le attività di comprensione*.

Tabella IX: Test di significatività statistica (Kruskal-Wallis)

Variabile Dipendente	P-value	Test statistic
Rapporto_Comprensione	0.03033321	8.922938
Rapporto_Manutenzione	0.01673363	10.22678
Effort_Medio_Comprensione	0.04383816	8.107668
Effort_Medio_Manutenzione	0.2201756	4.41308

C. Analisi Ulteriore

Il Dunn test è un test statistico non parametrico che viene utilizzato per confrontare coppie di gruppi in un contesto di analisi di post-hoc dopo un test di significatività globale, come ad esempio il test di Kruskal-Wallis che abbiamo applicato nella Sezione IV.B. Una volta stabilito se esistesse una differenza significativa generale tra i gruppi, il nostro interesse si è focalizzato su un'analisi a grana più fine rispetto a quella offerta dal test precedente. In particolare, eravamo interessati a determinare in quali particolari combinazioni di gruppi fosse possibile individuare una significatività statistica specifica. La Tabella X, in cui riportiamo i risultati del test di Dunn, mostra il $p - value$, la variabile dipendente a esso associata e la combinazione di gruppi tra cui è stato eseguito il confronto. Possiamo notare che esiste una differenza specifica tra i gruppi A e B per la variabile Rapporto_Comprensione e tra i gruppi A e C per la variabile Rapporto_Manutenzione. Consultando i dati descritti nella Sezione IV e valutando i $p - value$ ottenuti possiamo dedurre che:

(i) *la co-presenza di design pattern e code smells (gruppo A) ha impattato negativamente sulla comprensione del codice sorgente nelle attività di comprensione (il gruppo B - Best Practice ha ottenuto un punteggio maggiore)*;
(ii) *la co-presenza di design pattern e code smells (gruppo A) ha impattato negativamente sulla comprensione del codice sorgente nelle attività di manutenzione (il gruppo C - Worst Practice ha ottenuto un punteggio maggiore)*.

Infine non abbiamo riportato la variabile Effort_Medio_Comprensione poichè nessun confronto specifico tra coppie di gruppi ha raggiunto un livello di significatività statistica adeguato. In merito a questo risultato è opportuno precisare che, quando si utilizza il Dunn test per confrontare tutte le coppie di gruppi, è consigliabile correggere il p-value per il problema della molteplicità dei confronti. Ciò significa che è necessario applicare un metodo di correzione del p-value per controllare il tasso di errore di tipo I complessivo (ovvero ridurre il rischio di ottenere risultati significativi casuali nelle analisi post hoc). Il metodo di correzione del $p - value$ per cui abbiamo optato è il metodo di Bonferroni (uno dei piu utilizzati). Questo metodo divide il livello di significatività prefissato per il numero totale di confronti effettuati. Queste correzioni possono rendere più difficile raggiungere la significatività statistica per le singole coppie di gruppi, anche se il test globale di Kruskal-Wallis è significativo. Ciò può essere dovuto a una serie di fattori, come il numero di confronti effettuati, la dimensione campionaria e la variabilità dei dati. Quindi, sebbene Effort_Medio_Comprensione è significativa nel test di Kruskal-Wallis, è possibile che nel Dunn test essa non raggiunga la significatività statistica a causa delle correzioni.

Tabella X: Test di significatività statistica (Dunn)

Variabile Dipendente	P-value	Gruppi
Rapporto_Comprensione	0.0263778	A - B
Rapporto_Manutenzione	0.02475755	A - C

V. Minacce alla validità

Conclusion validity riguarda le questioni che influenzano la capacità di trarre una conclusione corretta. Nel nostro studio, abbiamo utilizzato test statistici appropriati. In particolare, sono stati utilizzati dei test non parametrici (ovvero i test di Kruskal-Wallis e di Dunn) per respingere statisticamente le ipotesi nulle. Quando abbiamo rigettato le ipotesi nulle, i $p - value$ erano inferiori a 0,05. Questo conferisce solidità ai risultati ottenuti. Per quanto concerne la misurazione, abbiamo optato per misure oggettive per il calcolo di entrambe le variabili dipendenti: (i) i minuri per quanto riguarda l'effort e il rapporto di risposte corrette su totali per la comprensione (analogo alla precision). L'ambiente in cui si è svolta la sperimentazione era privo di rumore. Tutti i partecipanti possedevano conoscenze e competenze simili, dunque la loro distribuzione tra i gruppi era opportunamente omogenea.

Internal validity è mitigata dal design della sperimentazione. Ogni gruppo di partecipanti ha lavorato su entrambi i task in modo indipendente e gli oggetti sperimentali erano diversi per ogni gruppo. Un'altra possibile minaccia riguarda lo scambio di informazioni tra soggetti dello stesso gruppo o di gruppi diversi. Abbiamo impedito questo scenario con un monitoraggio costante dei soggetti e distribuendoli in maniera eterogenea tra le stazioni del laboratorio. L'affaticaticamento di quest'ultimi è stata mitigata permettendo loro di fare una pausa a metà della durata dell'esperimento, anch'essa supervisionata. L'esperimento è stato effettuato in un periodo relativamente tranquillo (non durante sessioni d'esame e alla fine dei corsi), dunque abbiamo mitigato gli effetti history. I questionari pre e post test ci hanno permesso di determinare che l'esperimento non è stato influenzato da effetti di maturation, in quanto i soggeti hanno affermato di non aver manifestato stati di noia o stanchezza. Inoltre abbiamo potuto constatare che il training pre-test, riguardante i task da svolgere, è stato sufficiente affinchè i soggetti capissero in che modo trattare gli oggetti sperimentali. Tutti i partecipanti erano studenti magistrali volontari e hanno utilizzato la propria macchina con l'IDE con cui hanno avuto piu esperienza. Nessun soggetto ha abbandonato l'esperimento durante la sua esecuzione. Abbiamo evitato compensation rivarly e resentful demoralization distribuendo i soggetti tra i vari gruppi senza specificare a quale ognuno di loro appartenesse (dunque i soggetti non conoscevano a priori a quale treatment appartenessero).

Tuttavia, nonostante le misure adottate per affrontare le minacce interne, è importante considerare che potrebbero ancora sussitere alcune minacce non completamente mitigate. Prima tra tutte la disparità nel numero di partecipanti tra i gruppi: il gruppo C (con meno partecipanti) potrebbe essere più suscettibile a variabilità casuali o a influenze individuali, alterando così i risultati complessivi. Inoltre come già discusso, potrebbe entrare in gioco l'effetto di familiarità nell'utilizzo degli stessi oggetti sperimentali per entrambi i task.

Construct validity può essere influenzata dalle metriche utilizzate e dalle minacce sociali. Le metriche scelte sono state utilizzate in passato con scopi simili ai nostri (il calcolo dei rapporti corrisponde al calcolo di precision e recall effettuato per ogni partecipante). Per quanto riguarda la variabile di comprensione, il questionario è stato costruito in modo sufficientemente complesso da non essere ovvio. Per evitare apprensioni, non abbiamo valutato gli studenti in base ai valori di comprensione e di effort. Abbiamo evitato problematiche di mono-operation bias non impiegando variabili indipendenti singole e assegnando a piu soggetti diversi oggetti sperimentali da trattare. Inoltre abbiamo prevenuto effetti d'interazione tra treatments facendo in modo che ogni soggetto di un determinato gruppo eseguisse un solo treatment, ovvero quello scelto per il gruppo.

External validity riguarda la generalizzazione dei risultati. Queste minacce sono legate ai di comprensione e di manutenzione e all'uso degli studenti come partecipanti. Riguardo alla prima problematica, abbiamo utilizzato istanze reali di classi contenenti code smells e design patterns (o solo uno dei due o nessuno) da progetti open source disponibili su github.

Sull'uso degli studenti come partecipanti, possiamo dire che

sono stati specificamente formati su compiti di ingegneria del software e di programmazione ad oggetti. Pertanto, riteniamo che non dovrebbero essere inferiori a sviluppatori junior professionisti. Potrebbe anche essere possibile che i partecipanti siano più preparati sui design pattern e code smells rispetto a molti professionisti delle piccole/medie aziende software.