UNIVERSITÀ DEGLI STUDI DI NAPOLI "FEDERICO II"

Scuola Politecnica e delle Scienze di Base

Area Didattica di Ingegneria

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Laurea Triennale in Ingegneria Informatica

Software Debugging

Relatori:
prof. Tramontana Porfirio

Candidato:
Simioli Arcangelo
Matr. N46001168

Anno Accademico 2018/2019

Indice

Introduzione **1**

1 Debug **4**

2 Questionario **7**

2.1 Prima sezione 7

2.1.1 Analisi delle risposte 8

2.2 Seconda sezione 10

2.2.1 Risposte domande 11

2.3 Terza sezione 15

2.3.1 Risultati 16

3 Possibili incongruenze **21**

4 Test sul campo **23**

5 Conclusioni **25**

Introduzione

Il debugging (o semplicemente debug), in informatica, nell'ambito dello sviluppo software, indica l'attività che consiste nell'individuazione e correzione da parte del programmatore di uno o più errori (bug) rilevati nel software, direttamente in fase di programmazione, oppure a seguito della fase di testing o dell'utilizzo finale del programma stesso.[3]

L'attività di debug è una delle operazioni più importanti e difficili per la messa a punto di un programma, spesso estremamente variegata ed imprevedibile a causa della complessità dei software oggi in uso. Tale operazione è definita delicata per il pericolo di introdurre nuovi errori o comportamenti difformi da quelli desiderati, nel tentativo di correggere i malfunzionamenti per cui si è svolta l'attività di debug.[3]

Prima del rilascio di un nuovo software, durante la fase di analisi del codice creato, il debug è l'operazione più costosa e che richiede più tempo. Nonostante ciò, non sempre si riescono ad individuare o correggere tutti gli errori, a dispetto di quante risorse si siano impiegate nel tentativo di tracciare ogni eventuale risposta indesiderata del programma.

Data la complessità dell'operazione, non sempre viene data la giusta importanza ad essa. In particolare, in programmi poco complessi, oppure in caso di carenza di budget, viene tolto spazio al debug per prediligere altri tipi di analisi meno invasive.

Scoprire come, al giorno d'oggi, gli sviluppatori si approccino al debug potrebbe costituire un valido aiuto per migliorare lo sviluppo di tale attività in

futuro. Pertanto, sarà analizzata la risposta a domande comuni che costituiscono il cruccio per la maggioranza degli sviluppatori che seguono questa fase. Sarà inoltre proposta una risposta che possa replicare in maniera esauriente e fugare eventuali dubbi ricorrenti.

1. Cosa gli sviluppatori conoscono circa il debugging e come ciò riflette su di loro?

2. Come gli sviluppatori usano gli strumenti di debug?

3. In che modo interpretare i risultati 1) e 2)[1]

Capitolo 1

Debug

Dopo aver scritto il programma, si svolge una verifica effettuando dei test su di esso. Si risponde così alla domanda: abbiamo costruito il software nel modo giusto?

Il testing è una fase importante durante lo sviluppo di un software e durante il quale vengono trovati i malfunzionamenti dovuti ai difetti. Essi vengono quindi corretti durante la fase di debug.

Esistono varie difficoltà durante il debug. Forniremo di seguito vari esempi:

- gli effetti sono lontani dalla causa dell'errore;
- il malfunzionamento può comparire o scomparire temporaneamente a seconda di alcune particolari condizioni;
- il sintomo può dipendere dallo specifico ambiente di esecuzione;
- da un solo difetto possono scaturire più malfunzionamenti, così come un solo malfunzionamento può dipendere da più errori.

Per localizzare i difetti esistono vari metodi. Uno di questi consiste nell'utilizzo di "watch point" e variabili di watch. In tal modo è possibile restringere l'intervallo tra il difetto ed il malfunzionamento, mantenendo un' immagine dello stato del processo in esecuzione in corrispondenza dell' esecuzione di specifiche istruzioni: watch point e variabili di watch. Un "watch", in generale, è una semplice istruzione che inoltra il valore di una variabile verso un canale di output. L'inserimento di una sonda è molto invasivo e anch'essa non esente da errori. Esse, fondamentalmente, sono asserzioni, espressioni booleane dipendenti da uno o più valori di variabili legate allo stato dell' esecuzione. Un watchpoint non riferisce ad una riga di codice eseguibile ma ad un attributo di una classe. In alcuni linguaggi interpretati, è possibile modificare il valore di determinate variabili in fase di esecuzione semplificando il problema dei test in grado di replicare il malfunzionamento.

Esistono varie metodologie di debug:

1. Forza bruta
2. Backtracking
3. Eliminazione delle cause

Forza bruta: anche se è il modo meno efficace, ne esistono diversi tipi. Ad esempio, possiamo inserire nel codice varie sonde per cercare indizi e tentare di ricavare più informazioni possibili, oppure si possono utilizzare strumenti automatici di debug inserendo breakpoints o ancora osservando variabili. E' utile ricorrere a questo metodo solo se si è tentato in precedenza di ricorrere ad altri metodi, riscontrando esito negativo.

BackTracking: con il Debugging per BackTracking si cerca di ripercorrere il codice "all' indietro" partendo dal punto in cui si è verificato il malfunzionamento.

Eliminazione delle cause: Innanzitutto, si individua la tipologia dei dati che fanno fallire il programma. Successivamente, si cerca di formulare un' ipotesi sulla possibile causa del difetto, proponendo dati in ingresso in grado di rilevare il malfunzionamento. Si cerca quindi di verificare la validità di tale ipotesi.[2]

Capitolo 2

Questionario

Si procede dunque con un questionario preso da uno studio realizzato da Beller, Spruit, Spinnelis e Zaidman rivolto agli sviluppatori di software costituito da varie domande, suddiviso in tre sezioni.
Tutte le conseguenti conclusioni sono tratte in egual modo dallo stesso articolo.[1]

2.1 Prima sezione

La prima sezione risponde alla domanda:

Cosa sanno gli sviluppatori del debug?

Essa è costituita da quattro sottosezioni.

La prima sottosezione costituita da 3 domande, è atta a carpire informazioni di carattere generale riguardo gli intervistati. La seconda sottosezione esplicita le modalità e le tempistiche d'uso di strumenti di debug. La terza sottosezione asserisce l'importanza dei test con codice nelle operazioni di

debug. La quarta sottosezione riguarda l'opinione generale sull'invenzione della funzione "PrintF" per i debug.

2.1.1 Analisi delle risposte

Di seguito è elencata l'analisi delle risposte alle domande poste riguardanti i modi e tempi d'uso delle operazioni di debug.

Feedback numero 1 La maggior parte degli sviluppatori usa il debugger integrato. Solo una minima parte non lo usa oppure non usa affatto un debugger.

Feedback numero 2 Quasi tutti utilizzano strumenti esterni per il debug. Sono in pochi i detentori della conoscenza necessaria all'utilizzo di uno strumento di debug integrato.

Feedback numero 3 I breakpoints sono utilizzati dalla vasta maggioranza degli sviluppatori, in particolare vengono adottati spessissimo i line breakpoint, in maniera ridotta altri tipi di breakpoint e solo in quantità irrisorie i class prepare breakpoint.

Feedback numero 4 Molti sviluppatori hanno familiarità con i breakpoint. Sanno come usarli, non possedendo tuttavia dimestichezza con le variabili di watch, oppure adottandone in maniera marginale.

Feedback numero 5 Gli intervistati indicano che il testing è parte del processo di debug[1]. In particolare, i test vengono performati in maniera

massiccia all'inizio ed alla fine del processo. Nella cruciale fase di sviluppo, tuttavia, essi vengono mediamente trascurati.

Feedback numero 6 L'esperienza non è direttamente correlabile all' utilizzo del debug IDE. In particolare, hanno verificato che non esista correlazione tra l'uso di un IDE debugger e l'esperienza in sviluppo software, mentre hanno notato una debole proporzionalità tra l'esperienza e l'utilizzo di variabili di watches durante il debug[1].

Feedback numero 7 Molto spesso gli sviluppatori che creano test per riprodurre bug utilizzano gli stessi anche per correggere gli errori. C'è una moderata correlazione tra l'adozione dei test all'inizio ed alla fine del processo di debug per riprodurre e verificare la correzione dei bug. Hanno verificato verificato inoltre che chi utilizza un test all'inizio della fase di sviluppo, tenderà ad utilizzarlo anche durante tutto il processo di scrittura del codice, fino alla fine dello stesso.[1]

Altri feedback Per quanto riguarda le altre conclusioni a cui si è giunti mediante le risposte alle domande poste agli intervistati, in molti sono d'accordo sul fatto che "la miglior invenzione nel debugging sia ancora la PrintF"[1], pertanto non sanno usare, nè intendono imparare a farlo, altri strumenti di debug. Altre interpretazioni vedono come metodologicamente superiori alla printf gli strumenti di debug messi a disposizione dall'ambiente di sviluppo integrato(IDE). Tra di essi rientrano i più disparati metodi, tra cui l'uso di vari tipi di breakpoint, condizioni o variabili watch. In generale, la PrintF risulta più semplice e flessibile, ma meno efficiente. Pertanto viene

usata per prima e/o allorché non venga trovata una soluzione meno impattante. In alternativa vengono valutate altre idee.
In estrema sintesi, si può affermare che "con la PrintF si viaggia a piedi, con un GUI debugger si viaggia in aereo. Si può arrivare nella maggior parte dei posti a piedi, ma non è possibile andare troppo lontano nel tempo desiderato." [1]

2.2 Seconda sezione

La seconda sezione risponde alla domanda

Come gli sviluppatori correggono i malfunzionamenti nel loro IDE?

Nella seconda sezione vengono posti in primo piano i risultati dell'uso dell'infrastruttura WATCHDOG. Essa è incentrata attorno al concetto di intervalli di tempo, i quali catturano l'inizio e la fine delle comuni attività degli sviluppatori, come ad esempio leggere e scrivere un codice. Si è esteso il concetto di intervallo di tempo anche ad intervalli senza una fine. Tali eventi tengono traccia di quando e quante modifiche apportino gli sviluppatori in determinati intervalli di tempo prestabiliti, ad esempio ai breakpoint. Una *sessione IDE* è una sequenza ininterrotta di intervalli di tempo, all'interno dei quali lo sviluppatore non chiude la IDE o sospende il computer. Una *sessione IDE debugging* è una sessione IDE in cui lo sviluppatore usa il debug almeno una volta.[1]

Domande seconda sezione

1. Quanto prevalente è l'uso dell' IDE debugging?

2. Quanto tempo viene impiegato nell'uso dell' IDE debugging?

3. Quali funzionalità del debugger utilizzano gli sviluppatori?

4. Qual è la relazione tra testing e debugging?

5. Che relazione c'è tra la lunghezza dei file e lo sforzo di debug?

6. Gli sviluppatori superano spesso il punto di interesse?

Viene di seguito fornita la risposta a queste sei domande.

2.2.1 Risposte domande

Conclusioni tratte dalle risposte alla prima domanda *La maggioranza degli sviluppatori non usa l'infrastruttura di debug dell' IDE* . Quasi un terzo ha avviato una sessione di debug durante il periodo di collezione dei dati. Tra di essi, quota parte ha utilizzato almeno una volta una sua funzionalità propria. Al massimo un decimo ha performato una sessione di debugging.

Nell'evenienza sia necessario trasferire pochi dati, non vengono utilizzati strumenti di debug. In totale meno di un decimo ha utilizzato l'IDE debugger. In seguito, saranno presi in considerazione solo questi ultimi. Circa il 20% degli sviluppatori è responsabile di oltre l'80% degli intervalli di debug nel nostro esempio. Tale risultato consente di valutare gli effetti delle operazioni di debug utilizzate dagli sviluppatori attraverso un modello Pareto-Anova. Pertanto, vale la pena interessarsi a conoscere la frequenza e la durata delle sessioni di debug. Circa la metà degli utenti, utilizzando l'infrastruttura di debug fornita dall'IDE, ha lanciato il debugger quattro volte o meno durante

il periodo di collezione dei dati. Il 21% ha lanciato un proprio debugger, diverso da quello standard, più di 20 volte.[1]

Conclusioni tratte dalle risposte alla seconda domanda *In media il debug consuma meno del 14% del tempo attivo dello sviluppo in-IDE.* La maggior parte del tempo è usato per scrivere o leggere codice. Soltanto una minima parte di esso è impiegato per i test automatici. La maggior parte delle sessioni di debug richiede meno di 10 minuti. Circa la metà necessita al massimo di 40 secondi. Circa il 12% dura più di 10 minuti. [1]
Tale studio ha dimostrato quindi che, con un minimo sforzo di meno di un minuto (40 secondi), è possibile risolvere più della metà dei bug. Ciò accade in quanto la maggior parte dei bug consiste in piccole imperfezioni o anomalie inattese del codice, non in un suo completo malfunzionamento dovuto ad una cattiva gestione della fase di scrittura del software.

Conclusioni tratte dalle risposte alla terza domanda *I line breakpoints sono i più usati dalla maggior parte degli sviluppatori, anche tra coloro che adottano l'infrastruttura di debug.* Le opzioni di breakpoint non sono adottate dalla maggior parte degli utenti WatchDog 2.0. Riguardo l'evoluzione dei breakpoint durante il loro ciclo di vita, è possibile notare come la maggior parte dei loro cambiamenti sia relativa all'abilitazione o disabilitazione degli stessi. Altri tipi di breakpoints sono adoperati in maniera ridotta da un minor numero di sviluppatori. Essi consistono in poco più del 10%, così come resta piccola la percentuale di utenti che genera questi cambiamenti. Il metodo più diffuso per l'utilizzo dei breaktpoints è la loro creazione e

successiva distruzione, dopo aver effettuato un passaggio attraverso il codice. Altri tipi di funzionalità di debug, come per esempio le variabili watch, sono adottate in maniera marginale, così come risultano in numero minoritario gli sviluppatori che li impiegano.[1]

Conclusioni tratte dalle risposte alla quarta domanda *La maggior parte delle sessioni di debug inizia dopo aver letto o cambiato il codice, non dopo aver eseguito i test.*

Gli sviluppatori che spendono più tempo per i test probabilmente spenderanno più tempo anche per il debug.

Il target al quale si mira consiste nel ridurre il gap tra il tempo che gli sviluppatori investono per leggere o modificare i test sulle classi e quello di debug, ma gli intervistatori hanno notato che non vige correlazione tra questi due eventi.

Conclusioni tratte dalle risposte alla quinta domanda *Il debugging per le classi più piccole è maggiore rispetto alle classi più grandi*

.

Hanno esaminato la correlazione tra le dimensioni del file di una classe (nelle linee di codice sorgente) ed il numero di volte che lo sviluppatore visita l'editor durante una sessione di debug, trovando bassa correlazione. Inoltre si è esaminata la correlazione tra le dimensioni del file e la durata degli intervalli di debug nei quali essi sono aperti, non trovando correlazione.

Hanno quindi comparato il numero di classi in un singolo intervallo di debug sia con il numero totale di classi osservate con WATCHDOG per un

progetto (anche attraverso altri intervalli come leggere, scrivere ed eseguire test), sia con il numero di classi diverse che sono state oggetto di opera di debug durante un qualunque intervallo di debug. In entrambi i casi, i risultati sembrano indicare che il debug sia concentrato su un insieme relativamente piccolo di classi. Nel 75% delle sessioni di debug, al massimo il 5% delle classi del progetto sono state oggetto di debug.[1] Da ciò si deduce che nelle classi più piccole si concentrano la maggioranza dei bug, mentre le classi più grandi ne posseggono una minor concentrazione.

Conclusioni tratte dalle risposte alla sesta domanda *Nel 5% dei casi, gli sviluppatori superano il punto di interesse e devono ricominciare dall'inizio la sessione di debug.* Hanno calcolato la durata totale di tutti gli intervalli di debug per utente. Hanno quindi eseguito un test usando questi valori e l'esperienza di programmazione degli utenti che usano WATCHDOG 2.0. Per gli utenti per i quali era nota l'esperienza in tal campo e che avevano generato almeno un intervallo di debug, si è visto che è più alta la probabilità che tali sviluppatori utilizzino il debug dell'IDE.

Hanno ascoltato molti sviluppatori frustrati che hanno raccontato aneddoti ed esperienze negative riguardo lo scavalcamento del punto di interesse durante il debug. Infine hanno cercato dati oggettivi per supportare la tesi secondo cui sia risonante la gravità del problema, identificando possibili casi di superamento del punto di interesse.

Quando si verifica questo problema vuol dire che lo sviluppatore, ogni volta che passa oltre, deve ricominciare sempre dall'inizio. Ciò può succedere quando, ad esempio, si preme il tasto per procedere avanti troppo veloce-

mente o verificando troppo tardi che la locazione di un punto di interesse sia stata fissata in uno step precedente. Per dimostrare tale considerazione adattandolo al concetto di intervallo sopra menzionato, si è cercato un insieme di intervalli di debug che soddisfino le seguenti condizioni:

- l'ultimo evento che si verifica nell'intervallo di debug è un evento di step;
- l'intervallo di debug è seguito da un altro nella stessa sessione IDE.

Si è creato un sottoinsieme di questi intervalli di debug tramite l'imposizione di un tempo massimo tra due intervalli consecutivi di debug.

Con un tempo massimo minore o uguale di quindici minuti, si è notato che la quantità di nuovi possibili casi di superamento del punto di interesse diminuisce significativamente. Dopo circa quattro minuti, infatti, sono stati già identificati il 95% degli intervalli di debug, corrispondenti a circa 150 possibili casi di superamento.[1]

2.3 Terza sezione

La terza sezione risponde alla domanda

In che modo i singoli utenti di debug ed esperti interpretano i risultati delle sezioni precedenti?

Per convalidare e mitigare possibili controversie riguardo ciò che abbiamo trovato nelle sezioni precedenti, si è proceduto a combinare le osservazioni dalle interviste, misurazioni obiettive dall'IDE e approfondimenti su aneddoti correlati alle interviste.

2.3.1 Risultati

Uso dell'IDE debugger Si è visto che l'80% degli intervistati afferma di usare WATCHDOG 2.0. Al contrario, durante il periodo di osservazione, 2/3 di essi non lo usava. Inoltre nessun utente ha impiegato più del 30% del tempo totale di sviluppo per la fase di debug. Ci potrebbero essere varie ragioni per questa discrepanza.

- La maggior parte dei dati viene da una piccola parte di utenti.
- Per alcuni utenti il periodo di osservazione è stato ridotto: ciò potrebbe coincidere con il non avere problemi di debug.
- Molti sviluppatori hanno usato la printf come metodo di debug, anche se durante l'intervista hanno affermato che non fosse ottimale e che non venisse da loro utilizzato. Tale conclusione è valida anche per quanto riguarda WATCHDOG.
- La discrepanza tra comportamenti osservati e la risposta alle domande non è un fenomeno nuovo in questo tipo di studi.

Debugging con printf Si è visto che gli sviluppatori sono ben informati riguardo il debug utilizzando printf. Essa è la scelta preferita all'inizio di un lungo processo di debug. Pur conoscendone i limiti, printf viene definita come uno strumento semplice e universale al quale si può sempre ricorrere specialmente quando si usa un nuovo linguaggio di programmazione.[1]

Uso delle funzionalità di debug *Raramente sono necessarie funzionalità di debug più avanzate.* La maggior parte degli intervistati usa i breakpoints semplici, mentre quello più avanzati sono stati di rado adoperati. Tra questi ultimi, il più usato è il breakpoint condizionale. Essi, in effetti, sono di gran lunga il secondo tipo di breakpoint utilizzato. Un risultato simile si può osservare in relazione ad altre funzionalità del debug: più è avanzata la funzionalità e meno viene utilizzata. Tuttavia i numeri osservati di queste ultime sono minori rispetto a quelle dichiarate.

I Debugger sono difficili da usare

La difficoltà nell'uso dei debugger è un'altra ragione per la quale gli sviluppatori, anche esperti ingegneri, non usano i debugger. Essi, nonostante siano strumenti molto potenti, non sono semplici da capire, dunque risultano difficilmente utilizzabili. *Non ci sono sufficienti conoscenze*

Molti sviluppatori hanno dovuto imparare gli strumenti di debug da autodidatti dopo aver raccolto frammenti da internet. E' una rarità incontrare qualcuno che conosca in modo approfondito gli strumenti di debug. Non si riesce a capire perché non ci sia abbastanza spazio per questo argomento nei piani di studio universitari.

Tempo per il Debug Il debug nella maggior parte dei casi è molto breve. Inoltre, tale tempo potrebbe essere sovrastimato, in quanto si tende a sovrastimare ciò che si considera noioso o faticoso. Un altro motivo è da ricercare nell'evidenza che vengano utilizzate funzioni come la printf, per le quali non sempre è possibile quantificare il tempo di utilizzo.

Una massima comune nell'ingegneria del software è che: 'piccolo è meglio'. Contrariamente, si è notato che a file di dimensioni maggiori non corrispondeva necessariamente un aumento proporzionale del tempo di debug. Le classi di problemi più difficili da affrontare sono quelle dove sono coinvolte le interfacce o le transazioni. Le interfacce, in particolare, sono brevi ma hanno molta logica al loro interno. Esse interagiscono con più applicazioni e parti del programma. Si reputano necessari ulteriori studi riguardo la non correlazione tra dimensioni dei file e tempo speso in debug.[1]

Uso di test per i debug Nel sondaggio, la maggior parte degli intervistati pensa che il test di unità sia parte del debug. In particolar modo, si ritiene opportuno per riprodurre malfunzionamenti all'inizio del processo. Tuttavia, vigono pareri contrastanti tra gli intervistati. Gli sviluppatori che vogliono ottenere qualità più alta eseguono più test di unità. Al contrario, alcuni affermano che lo sforzo di debug diminuisca all'aumentare dei test. Una risposta plausibile potrebbe essere che la creazione dei test stessi aggiunge codice e complessità che potrebbe essere oggetto di debug. Si ritengono comunque necessari ulteriori che avvalorino tale tesi.

Superamento del punto di interesse Circa il 5% degli intervistati oltrepassa il punto di interesse e deve cominciare una nuova sessione di debug. Ciò indica che esiste un limite nei debugger odierni che potrebbe essere superato dai debugger *back-in-time*. Questo particolare tipo di debugger potrebbe permettere agli sviluppatori di tornare indietro in ordine durante

l'esecuzione del programma, senza ricominciare il processo dall'inizio. Tutti gli intervistati si sono trovati in situazioni in cui tale tipo di debugger avrebbe semplificato il lavoro, consentendo un ampio risparmio di tempo. Ad oggi, esistono debugger con funzionalità di drop frame. Essi permettono di tornare all'inizio di un metodo, anche se non revocano gli effetti collaterali già accaduti. Ad oggi i principali debugger non supportano la funzione *back-in-time*.

Miglioramenti negli IDE debugger E' stato chiesto agli intervistati come i creatori di debugger potrebbero migliorare il loro prodotto. Le risposte si possono dividere in due categorie

- Rendere le funzioni principali più semplici da utilizzare preservando contemporaneamente tutte le funzionalità esistenti;
- Creare tools che catturino la totalità del processo di debug non soffermandosi sulle singole componenti.

Partendo da questo studio in via sperimentale, questi dati sono stati utilizzati da una compagnia commerciale per migliorare i loro prodotti. Essa, mentre era già in possesso di dati e conclusioni varie su uno stretto numero di sviluppatori, non era in possesso di dati su una larga scala di utilizzatori di debugger e dei relativi dettagli che questo studio fornisce. Inoltre, sempre partendo dai dati di questo studio, ha provveduto ad aggiornare varie funzionalità e ad aggiungerne di nuove.
Ad esempio, si è scoperto che molti sviluppatori non vedevano di buon oc-

chio il fatto che i breakpoint dovevano essere ogni volta rimossi manualmente. Alcuni, addirittura, osservano che i vecchi punti di interruzione bloccano o alterano inaspettatamente il loro flusso di dati. Pertanto, alcuni debugger hanno la possibilità di rimuovere i breakpoint con un semplice click, in base a quando essi sono stati creati. Oltretutto, si è scoperto che il debug con printf (anche se è un artificio) sia migliore dei debugger con variabili watch, in quanto mantiene la storia passata anche solo tenendo memoria delle entità passate nella griglia a schermo.

Al contrario, gli sviluppatori non possono arricchire il debugger con librerie di terze parti.

Per mantenere una sorta di cronologia, viene usato un costrutto artificiale. Esso crea un breakpoint condizionale che stampa informazioni e restituisce sempre un valore falso. Tale funzione, però, non è un vero breakpoint condizionale. Esso richiede un'approfondita conoscenza del debugger e possiede scarse performance, aggiungendo inutilmente ulteriore sovraccarico per il sistema. Oggi, invece, esiste un nuovo tipo di breakpoint detto *tracepoint*, il quale ne supera i limiti lavorando velocemente e senza aggiungere sovraccarico inutile al sistema per una semplice stampa.[1]

Capitolo 3

Possibili incongruenze

In questo capitolo andremo a valutare alcune possibili incongruenze.

Validità interna Un esempio potrebbe derivare dall'osservazione che alcuni sviluppatori potrebbero avere un interesse personale a non dare risposte veritiere. Per evitare il problema, si è ricorso ad un grande numero di intervistati provenienti da gruppi di studio diversi. Inoltre, a conferma della validità di tale tesi, sono stati riscontrati risultati simili su gruppi di studio diversi abbastanza grandi, tali da poter sopperire anche a piccole inesattezze. Alcuni dati sono stati presi in maniera automatica andando a mitigare l'influenza dei singoli intervistati. Discrepanze tra dati raccolti e risposte ai sondaggi sono ampiamente riconosciute e verificate in tutte le attività di questo tipo.

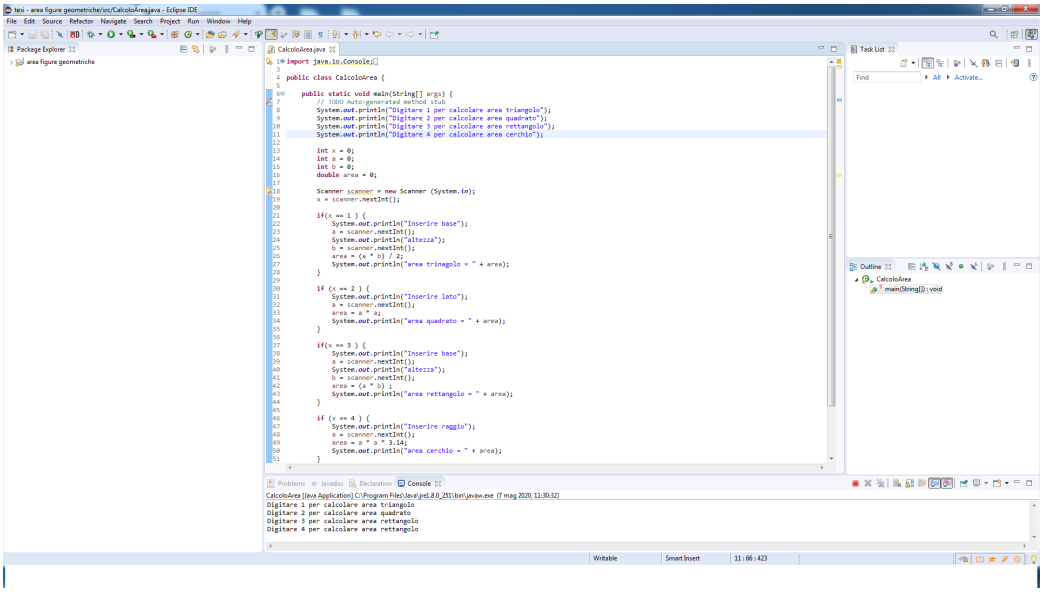
Validità esterna Per questo studio sono state intervistate moltissime persone e si è fatto riferimento ad una quantità enorme di dati, in particolare

relativa al linguaggio di programmazione Java ed allo strumento di sviluppo Eclipse. Tuttavia, tali risultati possono essere realisticamente utilizzati anche per altri linguaggi di programmazione.[1]

Capitolo 4

Test sul campo

Si è provato, dunque, a passare in prima linea scrivendo un piccolo codice ed a rispondere in prima persona alle stesse domande poste agli sviluppatori.



Cercando di dare una risposta riassuntiva a tutte le domande, possiamo affermare quanto segue.

Partiamo dicendo che, data la semplicità e le piccole dimensioni di questo

programma, si è scelto di non utilizzare strumenti esterni di debug, ma soltanto le funzionalità messe a disposizione da Eclipse.
Non è stato necessario l'utilizzo di breakpoint. E' bastata, infatti, semplicemente la consolle grafica di Eclipse. I pochi test eseguiti sono stati parte della fase di debug e sono stati performati alla fine del processo di programmazione.
Principalmente, le operazioni di debug sono state effettuate mentre veniva scritto il codice stesso. La printf è risultata la funzione più semplice, dunque largamente utilizzabile e fruibile per il debug.
Non è stato utilizzato WATCHDOG. Il tempo utilizzato per il debug è stato circa un decimo del tempo totale.

Capitolo 5

Conclusioni

Esiste una forte dicotomia tra le opinioni degli sviluppatori, la conoscenza ed i comportamenti. Per esempio, ad oggi molti affermano che gli strumenti di debug sono superiori alla printf, ma essa è ancora largamente usata. Solo un terzo degli sviluppatori usa gli strumenti integrati di debug della propria piattaforma di sviluppo. Il debug è un'attività definita dalle necessità, veloce e di breve durata, quindi gli strumenti per implementarlo dovrebbero essere quanto più semplici possibili.

Di conseguenza, gli sviluppatori adoperano solitamente funzioni di base e raramente ricorrono a funzionalità avanzate.

Mediamente, essi impiegano poco tempo per il debug, al contrario di ciò che è emerso dalle interviste, nelle quali si è dichiarato un tempo maggiore per tale attività.

Si è notato che, più lo sviluppatore diventa esperto, maggiore è il tempo usato per il debug, utilizzando in parallelo anche funzionalità più complesse.

Avere più test non riduce la fase di debug, probabilmente perché i test ag-

giungono ulteriore codice.

Gli sviluppatori hanno limitate e superficiali conoscenze di debug. La maggior parte di esse sono state ottenute in maniera autonoma, oppure dettate dall'esperienza. Premesso ciò, avendo constatato che con un piccolo sforzo ed un intervallo temporale trascurabile si possa ottenere un grande risultato per la risoluzione della maggioranza dei bug, si ritiene che dovrebbe essere inserita nei vari corsi di studio universitari una conoscenza più approfondita di questo argomento. Ciò renderebbe più semplice l'utilizzo di funzionalità anche complesse.

Oltre all'utilizzo di debugger back-in-time, gli sviluppatori non hanno espresso particolari richieste, a prova del fatto che si potrebbe semplicemente utilizzare ciò che è già a disposizione, invece di inventare nuove e disparate funzioni.[1]

L'area del debugging è molto vasta e per certi aspetti inesplorata. La maggior parte degli sviluppatori non usa debugger, oppure ricorre a metodi *casalinghi* in quanto ritenuti più semplici e veloci da utilizzare. In conclusione, possiamo affermare che, nonostante il debug sia una parte molto importante ed obbligatoria durante lo sviluppo software, sia tuttora molto sottovalutata dalla comunità scientifica. Pertanto, il debug dovrebbe essere oggetto di ulteriori ed approfonditi studi.

Bibliografia

[1] Moritz Beller, Niels Spruit, Diomidis Spinellis, and Andy Zaidman. On the dichotomy of debugging behavior among programmers. In *Proceedings of the 40th International Conference on Software Engineering*, pages 572–583, 2018.

[2] Tramontana Porfirio. Debugging.

[3] Wikipedia. Debugging — wikipedia, l'enciclopedia libera, 2020. [Online; in data 4-maggio-2020].