Sicurezza e Affidabilità

Fabio Ferrario
@fefabo

2023/2024

Indice

Capitolo 1

Introduzione

Appunti di Sicurezza e Affidabilitá di Fabio Ferrario.

Il Corso

Gli appunti fanno riferimento alle lezioni di SEA erogate nel secondo semestre dell'anno accademico 23/24.

Programma del corso

Il programma si sviluppa come segue:

1. Garantire l'Affidabilitá del Software

    1.1 Introduzione al Test e l'Analisi del Software

    1.2 Test Combinatorio

    - Combinazione a Coppie
    - Metodi di partizione delle Categorie
    - Cataloghi per il Test

    1.3 Test Strutturale

    - Test basato sugli Statement
    - Test basato sui Branch
    - Test basato sulle Condizioni

    1.4 Esecuzione del Test

    - Speicifica e Implementazione del caso di Test
    - Scaffolding: Driver e Stub

- Oracoli

1.5 Analisi Statica

2. La sicurezza del Software

2.1 Rischi nell'uso dei sistemi informativi, ruoli e competenze

2.2 Tecniche e protocolli per la sicurezza

- Crittografia, errori di implementazione e attacchi
- Sicurezza nei sistemi operativi e nelle strutture di rete

2.3 Programmazione sicura

- Errori di sicurezza nelle applicazioni
- Analisi di noti programmi che presentano vulnerabilitá

2.4 Programmi pericolosi: Troiani, Back-door, Bombe logiche, Virus, Worm

2.5 Difese: intrusion Detection System, Attacchi di verifica, Firewall.

Capitolo 2

Test Funzionale

Il test funzionale è un tipo di testing che prende i casi di test dalla spcifica funzionale del programma. Viene anche chiamato Specification-Based Testing, ovvero dalle specifiche, e Black-Box Testing perchè non si ha visione del codice.

Una **Specifica Funzionale** è una descrizione di un comportamento atteso del programma.

2.1 Test Obligations

Una **Test Obbligation**, detta anche test objective, è una specifica parziale del caso di test che richiede alcune proprietà ritenute importanti per un test approfondito

2.1.1 Obbligazioni funzionali

le obbligazioni di tipo funzionale sono obbligazioni che testano il funzionamento del programma secondo la sua specifica, ad esempio:

- Testa almeno una volta con un array già ordinato
- Testa almeno una volta l'autenticazione con un utente errato

Le obbligazioni Funzionali vengono dalle **specifiche del software**.

2.1.2 Obbligazioni Strutturali

Le obbligazioni strutturali invece vanno a testare una parte specifica del codice del programma:

- Testa almeno una volta il ramo else di uno specifico if-statement
- Testa almeno una volta un loop su più di una iterazione

Le obbligazioni strutturale derivano direttamente dal codice, senza tenere conto della specifica.

2.2 Boundary Testing

Nel test funzionale vengono testati ogni categoria di input e i **boundary** tra di esse.

Osservazione: Non c'è garanzia che testare i boundaries trovi più errori, ma per esperienza si trova che spesso gli errori si trovano li.

Il Boundary Testing è un tipo di test che seleziona obiettivi per tenere conto di comportamenti rilevanti nel software, distinguendo tra comportamenti Normali, Eccezzionali/Erronei, e Casi di Boundary.
Il test perimetrale suggerisce che gli obiettivi del test devono tenere conto dei comportamenti distinguibili del software e dei confini tra di essi

Catalogi di Testing

I catalogi di test sono delle guide per identificare obbligazioni per una classe di elementi ben caratterizzati.

Il Boundary Testing può essere assistita e resa più sistematica con cataloghi di test che indichino scelte standard (di casi normali, eccezionali e limite) per tipi tipici di voci di specifica.

Ad esempio, un catolog per testare un Range $L...U$ può essere

L-1, L, Valore tra L e U, U, U+1

2.3 Category Partition

Il metodo di Partizione delle categorie è un approccio sistematico al testing funzionale in cui si identificano le caratteristiche e i valori da cui poi possiamo generare le combinazioni, possibilmente con l'approccio pairwise

2.3.1 La partizione

L'approccio manuale si compone di questi passi:

1. Scomponi la specifica in features testabili indipendentemente.
   Per ogni feature vanno identificati:
   - Parametri
   - Elementi d'ambiente
2. Identifica valori rilevanti
   - Concentrati separatamente su ogni valore di input/parametro/risultato
   - Per ognuno di essi, identifica le caratteristiche elementari, ovvero le categorie di valori ch ti permettono di discriminare i vari comportamenti descritti nella specifica.
   - Per ogni caratteristica (cateogria) applica il boundary testing per identificare obiettivi di test rappresentativi (valori normali, erronei o di boundary).
   - Introduci dei vincoli alle possibili combinazioni, in modo da limitare gli obiettivi (combinazioni) irrangiungibili nella fase successiva.

Capitolo 3

Test Strutturale

il test strutturale, è un tipo di test che si basa su alcuni criteri che hanno lo scopo di trovare dati di test che consentano di **percorrere tutto il programma**.

A differenza del testing funzionale, in cui la completezza del test è giudicata sui requsiti senza tener conto del programma sotto esame, nel testing strutturale viene giudicata la completezza del test in base alla struttura del programma.

Il test strutturale è anche chiamato "white/glass box testing", mentre quello funzionale è chiamato "black box testing".

Si noti che il testing strutturale consiste ancora nel testare il prodotto (codice) rispetto alle specifiche: cambia solo la misura della completezza!

Osservazione: I test confrontano **sempre** un programma con una specifica

3.1 Adeguatezza dei test

Quando generiamo una suite di test, come possiamo grantirne la **scrupolosità**? Per farlo dobbiamo rispondere alle domande: *Quali* e *Quanti* test dobbiamo generare, e *quando dobbiamo fermarci.*
In linea di principio, l'obiettivo dovrebbe essere quello di generare una suite **adeguata**, vale a dire una suite di test che, se il software sottoposto a test viene superato con successo, garantisca una qualche proprietà del software stesso.

L'adeguatezza è quindi una sorta di "assicurazione" sull'abilità della suite di test nel trovare difetti.

Osservazione: Non possiamo garantire in nessun modo che una suite trovi tutti o alcuni dei difetti, e non possiamo garantire neanche che li trovi con alta probabilità:

≪*Testing can be used to prove the presence, not the absence, of errors*≫

In sostanza nessun metodo di progettazione dei test fornisce alcuna garanzia sulla capacità di scoprire difetti per le suite di test generate.

Cosa Facciamo? Quindi come costruiamo una suite di test *accettabile*? Generare Test randomicamente finché non finiamo tempo o budget non ci soddisfa euristicamente, utiliziamo quindi delle strategie basate sui **criteri di adeguatezza**.

3.1.1 Criteri di Adeguatezza

L'adeguatezza non ci da una garanzia sul potere di rilveamento dei difetti, ma é utile definire dei criteri euristici di adeguatezza simili a delle regole di progettazione.

Molte discipline progettuali utilizzano regole di progettazione per valutare *non* se un progetto è adeguato, ma se *esso è inadeguato*. L'idea è che un design che segue queste rule non è necessariamente adeguato, ma uno che non segue queste regole necessariamente sarà inadeguato!

Criteri pratici di (in)adeguatezza per i test Molti criteri di (in)adeguatezza per il testing derivano da osservazioni di buon senso su ciò che ci aspetteremmo come minimo da una suite di test.

Ricorda che questi criteri ci aiutano a capire perchè ci piace o non piace una suite di test, ma soddisgarli (o no) non implica niente sull'effettiva abilità della suite nel trovare difetti!

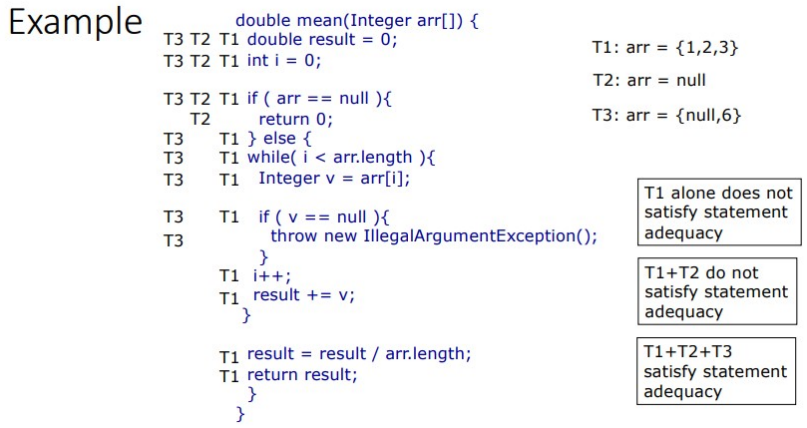
DEFINIZIONE

Un criterio di adeguatezza è un predicato che assume valore vero o falso per una coppia $< P, T >$, dove P è un programma e T è una suite di test. Se il criterio è $True$, diciamo che la suite è adeguata per il programma.

Un criterio di adeguatezza generalmente è fatto da sottopredicati chiamati **test obligations**.

Una suite T soddisfa i criteri di adeguatezza per un dato programma P *se e solo se*:

- Tutte le esecuzioni dei casi di test in T su P passano.
- Tutte le test **obligations** sono soddisfatte da almeno un test case nella suite.



In questo esempio abbiamo che i singoli test non riescono a soddisfare singolarmente i criteri di adeguatezza perchè non riescono a coprire l'intero codice.

3.2 Test Basato sugli Statement

Definiamo il primo criterio di adeguatezza che guardiamo: La copertura dei Basic Block e/o degli Statement.

3.2.1 Statement Coverage

DEFINIZIONE

Per una suite di test T possiamo definire la **coverage** degli statement come la frazione di enunciati di P eseguiti da almeno un caso di test in T:

$$C_{stmt} = \frac{\#\text{ executed stmts}}{\#\text{ stmts}}$$

Il test T **soddisfa il criterio di adeguatezza** di statement coverage *se e solo se* $C_{stmt} = 1$.

In sostanza, il critero di adeguatezza di statement coverage controlla quante 'righe' del codice vengono eseguite dalla suite di test, e il criterio é soddisfatto solo se tutte le righe di codice sono eseguite.

Obbligazioni insoddisfacibili In alcuni programmi potrebbero esistere degli statement irraggiungibili dai test, che renderebbero fisicamente insoddisfacibile il criterio di statement adequacy. Quindi ci troveremmo delle suite che risultano inadeguate, ma soltanto perché non é fisicamente possibile renderle adeguate.

Bisogna quindi tenere conto di questo problema, con due approcci possibili:

- Rimuovere dai criteri di adeguatezza tutte le obbligazioni di test insoddisfacibili.
- Usare la coverage come una misura di quanto ci siamo avvicinati all'adeguatezza.

3.2.2 Basic Block Coverage

Si puó notare che se due statements sono in sequenza, eseguirne uno automaticamente implica eseguire anche l'altro. Quindi possiamo considerare i **basic blocks**, dove le obbligazioni sono i blocchi del CFG[1] del programma.

DEFINIZIONE

Un Basic Block è una sequenza massima di istruzioni di programma contigue con un punto di ingresso e un punto di uscita.

Control Flow Graph

Come si costruisce un CFG? Un Control Flow Graph per un programma P é un grafo con:

- **Nodi** che rappresentano i Basic Blocks di P
- **Edges** che connettono i BB in una relazione sequenziale, possono essere etichettati con T o F se il BB finisce con un controllo condizionale

[1] Control Flow Graph

3.2.3 BB/Stmt Adeuqacy Rationale

3.2.4 Test suit Size VS Coverage

3.3 Test basato sui Branch

3.4 Test basato sulle Condizioni

Automazione del Testing

L'automazione del Testing non é presente nei parziali.