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 Strutturale

2.1 Adeguatezza e Copertura

Thoroughness Come possiamo grantire la **scrupolosità** (completezza) dei test?
Per farlo dobbiamo rispondere alle seguenti domande:

- **Quali** test dobbiamo generare?
- **Quanti** test dobbiamo generare?
- **Quando dobbiamo fermarci** a generare test?

In linea di principio, l'obiettivo dovrebbe essere quello di generare un'**adeguata** suite di test, 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 in principio una sorta di "assicurazione" sull'abilità della suite di test nel trovare difetti.

Non possiamo avere quello che vogliamo! 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? Potremmo continuare a generare test random e continuare a farlo finche non finiamo il tempo o il budget, ma questa strategia non ci soffisfa euristicamente!

2.1.1 Criteri di Adeguatezza

Bisogna rinunciare all'idea disperata dell'adeguatezza come garanzia sul potere di rilevamento dei difetti, e definire criteri euristici di adeguatezza simili alle 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 di Criteri di Adeguatezza Diamo una definizione formale di criteri di adeguatezza:

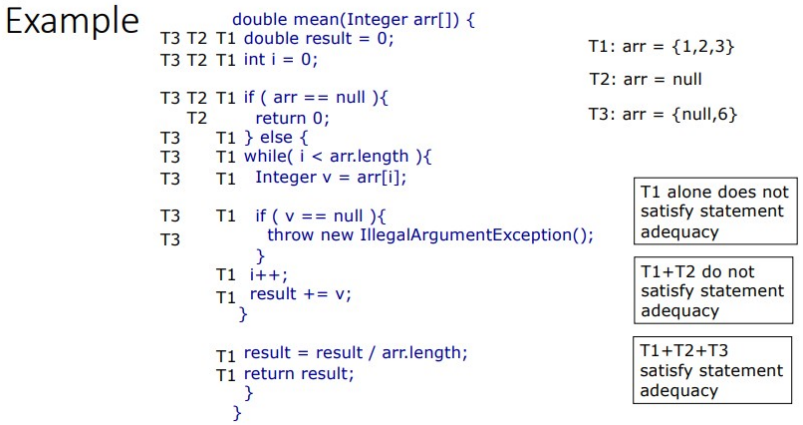
DEFINIZIONE

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.

2.2 Structural Testing

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

2.2.1 Statement Coverage

Per una suite di test T possiamo definire la **coverage** 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 se e solo se $C_{stmt} = 1$.