Fondamenti di informatica
Oggetti e Java

Luca Cabibbo

Correttezza
(prima parte)

Capitolo 16
ottobre 2015

Contenuti

- Introduzione alla correttezza
- Correttezza dei programmi
  - specifica di un programma
  - correttezza di un programma
  - verifica di correttezza di un programma
- *Correttezza dei metodi*
  - *specifica di un metodo*
  - *correttezza di un metodo*
  - *correttezza e responsabilità*
- *Verifica di correttezza*
  - *test a scatola nera*
  - *test a scatola trasparente*
- *Test di metodi, in pratica*
- *Individuazione e correzione degli errori*
- *Test di un insieme di metodi*

Correttezza

La correttezza è una delle qualità fondamentali dei prodotti software

- questo capitolo presenta la nozione di correttezza degli oggetti e dei programmi – studiando gli aspetti e gli strumenti che permettono di descrivere e verificare la correttezza del software

Introduzione alla correttezza

La correttezza è una delle qualità più importanti dei prodotti software

- detto in modo molto informale, un componente software (oggetto, programma, …) è corretto se “fornisce il comportamento per cui è stato pensato”
- l’uso di software non corretto può causare danni più o meno gravi
  - danni economici o materiali
  - danni alle persone
  - catastrofi
  - bocciatura di uno studente …

Attività legate alla correttezza

Attività principali legate alla correttezza

- caratterizzazione della correttezza
- verifica di correttezza
- identificazione degli errori
- correzione degli errori

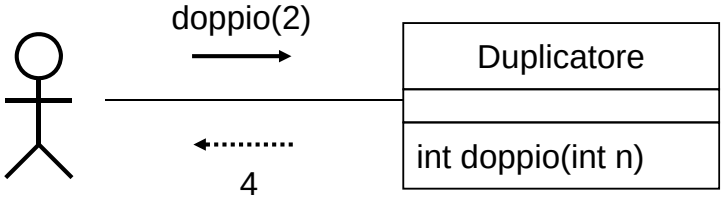
Caratterizzazione della correttezza

Per prima cosa, è evidentemente necessario capire che cosa vuol dire, per un'applicazione, un oggetto o per un metodo, essere "corretto"

- è necessario descrivere la "specifica" dell'applicazione, oggetto o metodo

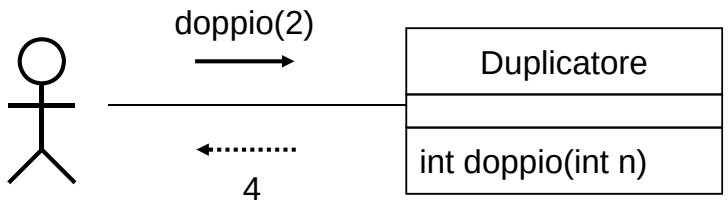
Ad esempio, un oggetto **Duplicatore**, con un metodo **doppio**, potrebbe avere la seguente specifica

- il metodo **doppio** deve calcolare il doppio del numero intero che gli viene passato come parametro



Verifica di correttezza

Si supponga di dover verificare la correttezza di un oggetto che calcola il doppio di un numero intero (passato come parametro)

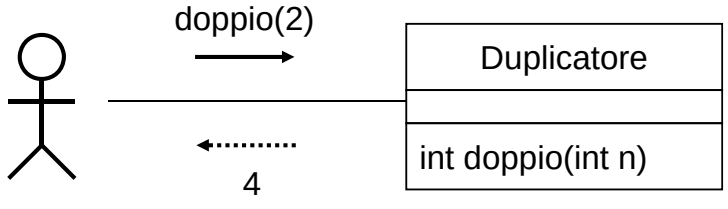


esempio di comportamento corretto

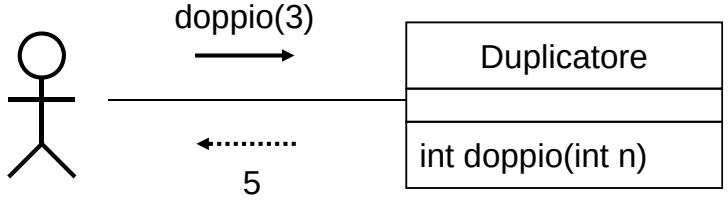
- che si può dire, in questo caso, circa la correttezza dell'operazione **doppio**?

Verifica di correttezza

Si supponga di dover verificare la correttezza di un oggetto che calcola il doppio di un numero intero (passato come parametro)



esempio di comportamento corretto



esempio di comportamento non corretto

- che si può dire invece ora circa la correttezza dell'operazione **doppio**?

Alcuni termini rilevanti

Malfunzionamento (o *comportamento errato*) di un oggetto

- una discrepanza tra il funzionamento effettivo dell'oggetto e quello corretto
- i malfunzionamenti sono relativi al comportamento esterno degli oggetti

Oggetto **non corretto** (o *errato*)

- durante il suo uso possono verificarsi malfunzionamenti
- un oggetto è **corretto** se durante il suo uso non possono mai verificarsi malfunzionamenti

Errore (o *difetto*)

- è la causa di un malfunzionamento
- gli errori sono relativi alla realizzazione interna degli oggetti
- un malfunzionamento è una manifestazione di un errore (o di un gruppo di errori)

Verifica di correttezza

Le tecniche di **verifica di correttezza** prevedono solitamente l'uso dell'oggetto da verificare, cercando di provocare eventuali malfunzionamenti e comportamenti non corretti

- il verificarsi di un malfunzionamento ci garantisce la presenza di errori nell'oggetto
- l'assenza di malfunzionamenti certifica la correttezza dell'oggetto solo se l'oggetto è stato stimolato in tutti i possibili modi
  - in pratica, però, la verifica di correttezza di un oggetto viene in genere basata su un numero limitato di prove

La verifica di correttezza dei programmi può essere usata semplicemente per mostrare la presenza di errori, ma mai per dimostrare la loro assenza!

E. Dijkstra, 1970

Individuazione e correzioni degli errori

La verifica di correttezza non dice quanti e quali sono gli errori

Individuazione degli errori – attività di ricerca degli errori

- tipologie di errori
  - errori di uso del linguaggio di programmazione
  - errori di codifica
  - errori nella realizzazione dell'algoritmo
  - errori nella scelta dell'algoritmo
- l'attività più difficile è trovare gli errori non riconosciuti dal compilatore
- la correzione di un errore è tanto più costosa quanto più lontano l'errore viene rilevato e corretto rispetto al momento in cui è stato introdotto

Correzione degli errori

- sostituzione della componente errata dell'oggetto con una componente corretta – e propagazione di tale correzione

Correttezza dei programmi

Consideriamo (semplici) programmi che risolvono (semplici) problemi di ingresso-uscita

La correttezza di un programma è legata alla sua specifica

Specifica di un programma

La nozione di “specifica di un programma” formalizza l’idea di “comportamento per cui il programma è stato pensato”

La **specifica di un programma** consiste nella descrizione delle sue

- **pre-condizioni**
  - proprietà che si assumono vere riguardo ai dati di ingresso del programma
- **post-condizioni**
  - proprietà che devono risultare vere al termine dell’esecuzione del programma

La specifica di un programma corrisponde alla specifica del problema che il programma deve risolvere

Esempi di specifica

Un programma per calcolare il fattoriale di un numero naturale N

- pre-condizione
  - N è un numero naturale – N>=0
- post-condizione
  - il valore calcolato è uguale al prodotto dei primi N numeri interi positivi – cioè è il fattoriale di N

Un programma per calcolare il massimo comun divisore di due numeri interi positivi N e M

- pre-condizione
  - N>0 && M>0
- post-condizione
  - il valore calcolato è il massimo comun divisore di N e M – ovvero …

Correttezza di un programma

La correttezza di un programma viene definita con riferimento alla sua specifica

Un programma **p** è **corretto** rispetto alla specifica ⟨**pre, post**⟩ se

- per ogni insieme di dati di ingresso che soddisfa la pre-condizione **pre**
  - il programma **p** termina
  - e fornisce come risultato un valore che soddisfa la post-condizione **post**

Non correttezza di un programma

Un programma è **non corretto** se

- esiste almeno un insieme di dati di ingresso che soddisfa **pre** per cui l'esecuzione del programma non termina
- oppure, esiste almeno un insieme di dati di ingresso che soddisfa **pre** per cui l'esecuzione del programma termina in modo anormale (con un'eccezione)
- oppure, esiste almeno un insieme di dati di ingresso che soddisfa **pre** per cui l'esecuzione del programma termina ma il valore calcolato non soddisfa **post**

La verifica di correttezza di un programma avviene proprio cercando contro-esempi relativi alla sua correttezza

Un programma non corretto

```
... calcola il fattoriale di n ...
// pre: n>=0
int f;    // il fattoriale di n
int i;    // per iterare tra 1 e n

f = 1;
i = 1;
while (i<=n)
    f = f*i;
    i = i+1;
... il risultato è f ...
```

- se **n** vale 0, produce un risultato corretto
- tuttavia, se **n** vale 2, non termina

Un programma non corretto

```
... calcola mediante somme ripetute il prodotto n*m
    dei numeri interi n e m ...
// pre: true (nessuna condizione su n e m)
int p;    // il prodotto di n e m
int i;    // per iterare tra 1 e n

p = 0;
for (i=1; i<=n; i++) {
    p = p+m;
}
... il risultato è p ...
```

- se **n** vale 2 e **m** vale 3, calcola correttamente 6
- tuttavia, se **n** vale -2 e **m** vale 3, calcola 0 anziché -6

Un programma corretto

```
... calcola mediante somme ripetute il prodotto n*m
    dei numeri naturali n e m ...
// pre: n>=0 && m>=0
int p;    // il prodotto di n e m
int i;    // per iterare tra 1 e n

p = 0;
for (i=1; i<=n; i++) {
    p = p+m;
}
... il risultato è p ...
```

- restituisce proprio **n*m** – calcolato nell'aritmetica modulare

Comportamento dell'utente del programma

Un programma, pur corretto rispetto alla sua specifica, potrebbe non funzionare correttamente se il suo utente non rispetta le pre-condizioni del programma

- ad esempio, un programma per calcolare il prodotto di due numeri naturali – ma l'utente gli chiede di calcolare il prodotto di due numeri interi (negativi)
- **attenzione, questo NON vuol dire che il programma non è corretto**

La responsabilità del programmatore che scrive un programma è limitata dalle pre-condizioni del programma

- un programma può (e deve) normalmente assumere che i suoi utenti conoscano e rispettino le pre-condizioni del programma
- eventuali malfunzionamenti causati dal mancato rispetto delle pre-condizioni del programma sono da imputare all'utente del programma – e non al programma o al programmatore

Verifica di correttezza di un programma

Ci sono varie strategie/approcci/tecniche per verificare la correttezza (effettuare il **test**) di un programma (o di un metodo) rispetto alla sua specifica

- approccio formale
  - la correttezza del programma viene dimostrata
- approccio pragmatico
  - la correttezza del programma viene verificata sperimentalmente

Test

La verifica di correttezza di un programma procede, in modo iterativo, come segue

- determina una combinazione dei dati di ingresso (**dataset**)
- esegui il programma usando questa combinazione di dati di ingresso
- confronta il risultato ottenuto con quello previsto
  - se il risultato ottenuto è diverso da quello previsto, si dice che il programma **non passa** il test – vuol dire che il programma non è corretto
  - se invece il risultato ottenuto è quello previsto, allora torna ancora a eseguire questa sequenza di passi oppure considera concluso il test – si dice che il programma **passa** il test

Attenzione:

- il fatto che un programma passa un test NON vuol dire che il programma è corretto

Un problema legato al test

In pratica, il test di un programma viene di solito svolto su un opportuno sottoinsieme dei dataset che soddisfano la pre-condizione del programma

Perché non fare un test esaustivo?

- si supponga di voler verificare un programma che calcola il massimo comun divisore di due numeri interi positivi minori o uguali a 2147483647 (il massimo valore per un **int**)
- l'insieme dei possibili dati di ingresso ha cardinalità superiore a 10^{18}
- se ciascuna prova richiedesse un nanosecondo (10^{-9} s), allora il tempo richiesto per la verifica sarebbe superiore a 10^{9} s
  - un test completo richiederebbe più di 30 anni

Scelta dei dati di ingresso per un test

Con quali dati effettuare un test?

- ci sono varie tecniche di verifica, che si differenziano sulla scelta dell'insieme dei dataset su cui basare la verifica
  - una scelta casuale dell'insieme dei dataset non è da considerarsi adeguata

Due approcci principali

- test a scatola nera
  - l'insieme dei dati di ingresso viene scelto solo in riferimento alla specifica del programma, senza far riferimento ai suoi dettagli realizzativi
- test a scatola trasparente
  - l'insieme dei dati di ingresso viene scelto anche in riferimento ai dettagli realizzativi del programma

Test a scatola nera

Il **test a scatola nera** di un programma è basato sulla sua specifica

- l'insieme dei possibili dati di ingresso per il programma viene partizionato in sottoinsiemi – chiamati **insiemi di equivalenza**
- il programma viene eseguito utilizzando una sola combinazione dei valori per ciascun insieme di equivalenza

L'approccio di test a scatola nera è basato sulla seguente ipotesi

- se il programma si comporta correttamente per una combinazione dei dati di ingresso X scelta nell'insieme di equivalenza C_X, allora il programma si comporta correttamente anche su ogni altra combinazione dei dati di ingresso scelta nell'insieme C_X
  - è solo un'ipotesi – potrebbe essere sbagliata
  - si tratta in effetti di insiemi di "presunta" equivalenza

La scelta degli insiemi di equivalenza è cruciale

Scelta degli insiemi di equivalenza

Idea

- alcuni insiemi di equivalenza rappresentano dataset "normali" per il programma
- altri insiemi di equivalenza rappresentano dataset "particolari" per il programma
  - casi particolari e/o valori al limite (nel valore e/o nella posizione)

Scelta degli insiemi di equivalenza

Leggi dalla tastiera un numero naturale N e calcola la somma dei primi N numeri dispari

- scelta degli insiemi di equivalenza
  - N vale 0 – caso particolare: estremo inferiore del dominio della funzione
  - N vale 1 – caso particolare: primo valore per cui il risultato è diverso da 0 (che è l'estremo inferiore del codominio della funzione)
  - N è maggiore di 1 – caso normale

Il test viene poi svolto effettuando una prova per ciascun insieme di equivalenza

- scegliendo un dataset rappresentativo per ciascun insieme di equivalenza – ad esempio, nel terzo caso, N=5

Scelta degli insiemi di equivalenza

Leggi dalla tastiera una sequenza di numeri interi e calcolane la somma

- scelta degli insiemi di equivalenza
  - la sequenza è vuota – caso particolare
  - la sequenza contiene un solo elemento – caso particolare
  - la sequenza contiene più elementi – caso normale

Il test viene poi svolto effettuando una prova per ciascun insieme di equivalenza

- scegliendo un dataset rappresentativo per ciascun insieme di equivalenza

Scelta dei dataset

Leggi dalla tastiera una sequenza di numeri e calcolane la somma

- scelta di un dataset per ciascun insieme di equivalenza
  - la sequenza è vuota –
  - la sequenza contiene un solo elemento – 8
  - la sequenza contiene più elementi – 1 2 3 4 5

Che cosa può accadere se la sequenza vale 1 2 3 4 5?

- il risultato è 15 – ok
- il risultato è 14 o 10 – errore di uno
- il risultato è 0 – gli elementi non sono stati sommati correttamente
- il programma rimane in attesa di ulteriori dati in ingresso

Attenzione – alcune sequenze sono meno significative – perché?

- la sequenza 0 0 0 0 0
- la sequenza 1 1 1 1 1
- la sequenza 3 4 0 -7

Esempio – massimo comun divisore

Insiemi di equivalenza per verificare la correttezza di un programma che calcola del massimo comun divisore di due numeri interi positivi N e M

(1) N vale 1 e M vale 1
(2) N vale 1 e M è maggiore di 1
(3) N è maggiore di 1 e M è un multiplo di N
(4) N è maggiore di 1, M è maggiore di N, N e M sono primi tra loro
(5) N è maggiore di 1, M è maggiore di N, N e M non sono primi tra loro (ma M non è multiplo di N)
(6) le classi precedenti, con N e M scambiati

- i casi particolari sono (1), (2) e (6) (in parte)
- i casi normali sono (3), (4), (5) e (6) (in parte)

Test a scatola trasparente

I **test a scatola trasparente** di un programma sono basati anche sulla sua struttura – ovvero, sulle istruzioni che contiene

- l'idea alla base di questo approccio è che un test può considerarsi concluso solo quando ciascuna istruzione del programma sia stata eseguita almeno una volta – e possibilmente se le istruzioni presenti sono state eseguite in tutti gli ordini possibili
- questo richiede, ad esempio, di scegliere i dati di ingresso in modo tale che ciascuna istruzione sia eseguita almeno una volta, ed inoltre che ciascuna condizione sia verificata almeno una volta e non verificata almeno una volta

Individuazione e correzione degli errori

Una volta che un programma si è rivelato non corretto, è necessario individuare gli errori (ovvero, le cause dei malfunzionamenti) per poterli correggere

- la verifica di correttezza non dice quanti e quali errori ci sono
- in genere però l'esito di un test fornisce delle utili indicazioni circa il tipo e la posizione degli errori presenti

Verifica di correttezza

```
... calcola la radice quadrata del numero naturale n,
    eventualmente approssimata per difetto ...
// pre: n>=0
int r;    // la radice di n

r = 1;
while (r*r<n) {
    r++;
}
... il risultato è r ...
```

- se **n** è un quadrato perfetto, non è corretto se **n** vale 0 (il risultato è 1)
- se **n** non è un quadrato perfetto, approssima per eccesso anziché per difetto

Individuazione e correzione di errori

Considero un errore

- **r** non vale mai zero

```
... calcola la radice quadrata del numero naturale n,
    eventualmente approssimata per difetto ...
// pre: n>=0
int r;    // la radice di n

r = 0;
while (r*r<n) {
    r++;
}
... il risultato è r ...
```

Il test viene ripetuto

- se **n** è un quadrato perfetto, il risultato è giusto
- se **n** non è un quadrato perfetto, approssima per eccesso anziché per difetto

Individuazione e correzione di errori

Un errore di uno

- se **n** non è un quadrato perfetto, approssima per eccesso anziché per difetto

```
... calcola la radice quadrata del numero naturale n,
    eventualmente approssimata per difetto ...
// pre: n>=0
int r;    // la radice di n

r = 0;
while ((r+1)*(r+1)<=n) {
    r++;
}
... il risultato è r ...
```