Sviluppo software in gruppi di lavoro complessi[1]

Mattia Monga

Dip. di Informatica
Università degli Studi di Milano, Italia
mattia.monga@unimi.it

Anno accademico 2021/22, I semestre

Lezione XV: *Design by Contract* con Eiffel

Il *principio di sostituzione di Liskov* stabilisce che, perché un oggetto di una classe derivata soddisfi la relazione `is-a`, ogni suo metodo:

- deve essere accessibile a pre-condizioni uguali o piú deboli del metodo della superclasse;
- deve garantire post-condizioni uguali o piú forti del metodo della superclasse;

Altrimenti il "figlio" non può essere sostituito al "padre" senza alterare il sistema.

Principio di sostituibilità

Le due condizioni sono quindi:

$$PRE_{parent} \implies PRE_{derived} \quad (1)$$
$$POST_{derived} \implies POST_{parent} \quad (2)$$

- (1) in un programma corretto non può succedere che PRE_{parent} valga e $PRE_{derived}$ no; l'oggetto *evoluto* deve funzionare in ogni stato in cui funzionava l'originale: non può avere obbligazioni piú stringenti, semmai piú lasche.
- (2) in un programma corretto non può succedere che valga $POST_{derived}$ ma non $POST_{parent}$; un stato corretto dell'oggetto *evoluto* deve essere corretto anche quando ci si attende i benefici dell'originale.

Principio di sostituibilità (cont.)

Un modo per garantire che le condizioni (1) e (2) siano automaticamente vere consiste nell'assumere implicitamente che, se la classe evoluta specifica esplicitamente una precondizione P e una postcondizione Q, le reali pre- e post-condizioni siano:

$$PRE_{derived} = PRE_{parent} \vee P \qquad (3) \qquad PRE_{parent} \implies PRE_{derived}$$

$$POST_{derived} = POST_{parent} \wedge Q \qquad (4) \qquad POST_{derived} \implies POST_{parent}$$

In Eiffel: `require else` e `ensure then`

Contratti “astratti”

```
extend (x: G)
-- Add `x' at end of list.
  require
      space_available: not full
  deferred
  ensure
      one_more:
      count = old count + 1
  end

full: BOOLEAN
-- Is representation full?
-- (Default: no)
    do
        Result := False
    end
```

Stronger precondition. . . ma *weaker* (uguali in realtà) in astratto

```
full: BOOLEAN
  -- Is representation full?
  -- (Answer: if and only if
  --  number of items is equal
  --  to capacity)
  do
      Result := (count = capacity)
  end
```

Problema: i parametri...

- Animale mangia Cibo (is_a Cosa)
- Mucca (is_a Animale) mangia Erba (is_a Cibo)

Ma questa covarianza è contraria al principio di Liskov perché restringe le precondizioni. La controvarianza (Mucca mangia Cosa, Sather) e l'invarianza (Mucca mangia Cibo, Java) vanno bene.
Eiffel invece è covariante... (il che, impedendo un controllo di conformità statico, introduce parecchie complicazioni $\rightsquigarrow$ CATcall, run time type identification...).