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 2024/25, II semestre

Lezione XV: Eiffel

Design by Contract™

Se usate estensivamente, le asserzioni possono costituire una vera e propria **specifica** delle componenti del sistema.
L'idea della **progettazione per contratto** (B. Meyer, 1986) è che il linguaggio per descrivere specifiche e implementazioni è lo stesso: la specifica è parte integrante del codice del sistema.
La specifica è parte del "contratto" secondo cui ciascun componente fornisce i propri servizi al resto del sistema.

Hoare triple

Che tipo di specifiche si usano?

$$\{P\}S\{Q\}$$

Ogni esecuzione di S che parta da uno stato che soddisfa la condizione P (pre-condizione) **termina** in uno stato che soddisfa la condizione Q (post-condizione).
Ogni programma che termina è corretto se e solo se vale la proprietà precedente.

Contratti

La tripla di Hoare $\{P\}S\{Q\}$ può diventare un **contratto** fra chi *implementa* (fornitore) S e chi *usa* (cliente) S

- L'implementatore di S si impegna a garantire Q in tutti gli stati che soddisfano P
- L'utilizzatore di S si impegna a chiedere il servizio in un stato che soddisfa P ed è certo che se S termina, si giungerà in uno stato in Q vale

Il lavoro dell'implementatore è particolarmente facile quando: Q è `True` (vera per ogni risultato!) o quando P è `False` (l'utilizzatore non riuscirà mai a portare il sistema in uno stato in cui tocchi fare qualcosa!). Weakest precondition (data Q) o strongest postcondition (data P) **determinano** il ruolo di una feature.

Eiffel

Eiffel

Un linguaggio *object-oriented* che introduce i **contratti** nell'interfaccia delle classi. Il contratto di default per un metodo ("feature") F è $\{True\}F\{True\}$.

Dispensa

```
feature
 decrement
 -- Decrease counter by one.
 require
  item > 0                -- pre-condition
 do
  item := item - 1        -- implementation
 ensure
  item = old item - 1     -- post-condition
 end
```


Lezione XVI: Eiffel

Organizzazione delle asserzioni

- pre/post-condizioni sulle feature (`require`, `ensure`)
- Invarianti di classe (`invariant`)
- asserzioni (`check`)
- loop invariant
  (`from .. invariant .. until .. variant .. loop .. end`)

Il supporto del linguaggio

- **invarianti di classe** sono condizioni che devono essere vere in ogni momento "critico", ossia osservabile dall'esterno. In pratica e come se facessero parte di ogni pre- e post-condizione.
- è possibile avere un supporto run-time alle **violazioni:** se una condizione non vale viene sollevata un'eccezione
- L'eccezione porta il sistema nel precedente stato stabile ed è possibile
  - terminare con un fallimento
  - riprovare


```
class
  GCD

feature

  gcd (x, y: NATURAL): NATURAL
    require
      x /= 0 and y /= 0
    local
      t: NATURAL
    do
      from
        Result := x;
        t := y
      invariant
        Result > 0
        t > 0
        mathgcd (Result, t) = mathgcd (x, y)
      until
        Result = t
      loop
        if Result > t then
          Result := Result - t
        else
          t := t - Result
        end
      variant
        t.max (Result).as_integer_64
      end
    ensure
      Result > 0 and Result <= x * y
      dividex: x.integer_remainder (Result) = 0
      dividey: y.integer_remainder (Result) = 0
    end
```



```
feature {NONE}

  mathgcd (x, y: NATURAL): NATURAL
    do
      from
        Result := x.min (y)
      until
        y.integer_remainder (Result) = 0 and then x.integer_remainder (Result) = 0
      loop
        Result := Result - 1
      end
    end

end
```


Procedurale vs. Dichiarativo

Spesso si scrivono le "stesse" cose due volte:

```
do                                  ensure
  balance := balance - x              balance = old balance - x
```

- Implementazione e specifica
- How & What

Il client è responsabile delle precondizioni, il fornitore di postcondizioni e invarianti.