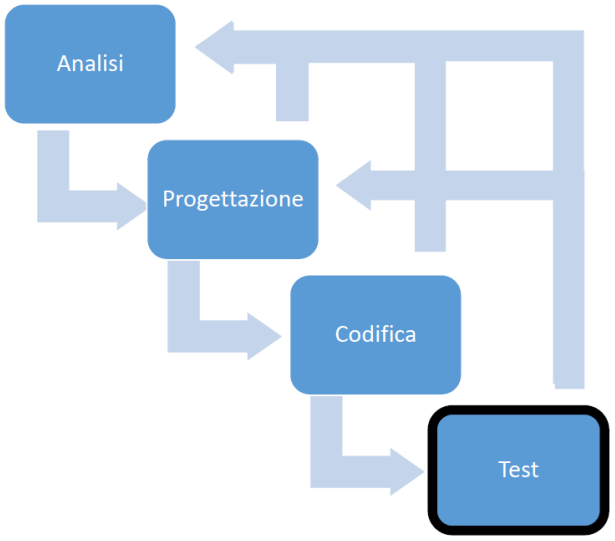
Testing di un Programma

Generalità

- Si tratta di parte integrante del processo di sviluppo del software, in cui si verifica con **approcci sistematici, oggettivi e ripetibili la correttezza** di un programma.
- Si distingue dal debugging, in cui si rimuovono errori logici (bugs) emersi durante il testing.



Generalità

- La correttezza non è l'unico aspetto che si valuta di un programma durante il testing.
- Altri aspetti includono:
  - **Prestazioni**: il programma viene eseguito in un tempo accettabile
  - **Usabilità**: l'utente interagisce col programma in modo chiaro e adeguato
  - **Portabilità**: il programma può essere eseguito anche su altre machine con sforzo marginale del programmatore
  - **Accettazione**: il programma risponde coerentemente agli input dell'utente

Casi di Test

- La verifica della correttezza prevede la definizione di **casi di test**, cioè situazioni in cui il programma può presentare degli errori logici.
- Il test di un programma può rilevare la presenza di malfunzionamenti, ma non escluderne l'assenza (**Tesi di Dijkstra**). In altri termini, i casi di test definiti potrebbero non scoprire tutti gli errori.
- D'altra parte, non si può pensare di realizzare un programma P che generi tutti i possibili casi per testare un programma Q (**Tesi di Howden**).

Metodologie di Testing

- Due principali categorie:
- Test funzionale-**black box**
  - Verifica la correttezza del programma **valutandone l'output**, senza entrare nel merito di come esso sia stato prodotto.
- Test strutturale-**white box**
  - Verifica la correttezza del programma **valutando la strutturazione** del codice sorgente.

Metodologie di Testing

- Due principali categorie:
- Test funzionale-**black box**
  - La definizione dei casi di test deve considerare possibili valori di input/output.
- Test strutturale-**white box**
  - La definizione dei casi di test deve considerare la struttura del codice sorgente

Test funzionale-**black box**

- Operativamente, si valutano i **risultati attesi**.
- Tre principali tecniche:
  - Verifica delle **condizioni limite**
  - Definizione delle **classi di equivalenza**
  - **Random guessing**

Test funzionale: condizioni limite

- La maggior parte dei bug si verificano in corrisponde dei limiti degli insiemi di valori di dati usati.
  - **Cicli:** Entra effettivamente nel ciclo? Esce correttamente dal ciclo?
  - **Array:** E' vuoto? La posizione/indice da usare esiste realmente?
  - **Input**: E' avvalorato? È nullo? È fuori range?
  - **Stream**: Il file esiste? Il file è stato creato/chiuso correttamente?

Test funzionale: condizioni limite

- Per definire le condizioni limite, è necessario conoscere gli insiemi di valori e determinarne i limiti. Questi vanno riportati a parte.
- E' buona prassi, definire condizioni limite innanzitutto sui blocchi di codice più semplici e su quelli più frequentemente utilizzatie poi sui blocchi restanti.

Test funzionale: condizioni limite

- Esempio:

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
      i++;
s[--i] = '\0';
```

- Cosa fa questo programma?
- Legge una sequenza di caratteri da un file e li memorizza in un array fino a quando non viene letto il carattere '\n' o si raggiunge la condizione MAX-1

Test funzionale: condizioni limite

- Esempio:

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
      i++;
s[--i] = '\0';
```

- Quali sono le condizioni limite?
  - Input vuoto (input nullo)
  - MAX==1 (input minimo)
  - Lunghezza del file== MAX (input massimo)
  - Lunghezza del file> MAX (input eccedente)
  - File non contiene il carattere ‘\n’ (uscita dal ciclo)

Test funzionale: condizioni limite

- Esempio:

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
      i++;
s[--i] = '\0';
```

- Definiamo i casi di test
  - Input vuoto (input nullo)

Test funzionale: condizioni limite

- Esempio:

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
      i++;
s[--i] = '\0';
```

- Definiamo i casi di test
  - Input vuoto (input nullo)
  - BUG: In caso di input vuoto (==‘\n’), non si entrerebbe nel ciclo, per cui avremmo i==0 quindi --0 !!

Test funzionale: condizioni limite

- Esempio:

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
      i++;
s[--i] = '\0';
```

- Definiamo i casi di test
  - MAX==1 (input minimo)

Test funzionale: condizioni limite

- Esempio:

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
      i++;
s[--i] = '\0';
```

- Definiamo i casi di test
  - MAX==1 (input minimo)
  - BUG: In caso di input minimo (MAX-1==0), non si entrerebbe nel ciclo, per cui non leggeremmo caratteri, quindi i==0. s[], --0 !!

Test funzionale: condizioni limite

- Esempio:

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
      i++;
s[--i] = '\0';
```

- Definiamo i casi di test
  - Lunghezza del file== MAX (input massimo)

Test funzionale: condizioni limite

- Esempio:

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
      i++;
s[--i] = '\0';
```

- Definiamo i casi di test
  - Lunghezza del file== MAX (input massimo)
  - BUG: In caso di input massimo (file ==MAX), la condizione MAX-1 chiude il ciclo prima della terminazione del file perdendo l'ultimo carattere.

Test funzionale: condizioni limite

- Esempio:

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
      i++;
s[--i] = '\0';
```

- Definiamo i casi di test
  - Lunghezza del file> MAX (input eccedente)

Test funzionale: condizioni limite

- Esempio:

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
      i++;
s[--i] = '\0';
```

- Definiamo i casi di test
  - Lunghezza del file> MAX (input eccedente)
  - BUG: In caso di input di lunghezza superiore (file >MAX), la condizione MAX-1 chiude il ciclo prima della terminazione del file perdendo tutti i successive caratteri.

Test funzionale: condizioni limite

- Esempio:

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
      i++;
s[--i] = '\0';
```

- Definiamo i casi di test
  - File non contiene il carattere '\n' (uscita dal ciclo)
  - BUG: La condizione !='\n' non sarebbe mai verificata, interrompendo il ciclo quando i>= MAX

Test funzionale: condizioni limite

- Valutando queste condizioni, in fase di debugging potremmo riscrivere il codice come sotto riportato:

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
      i++;
s[--i] = '\0';
```

```
int i = 0;
char s[MAX];
int stop = 0;
while (!stop && i < MAX) {
      s[i] = fgetc(file);
      stop = (s[i] == '\n');
      i++;
}
s[--i] = '\0';
```

Test funzionale: classi equivalenza

- Considerando che non si può testare un programma con tutti i possibili valori, si procede individuando **valori rappresentativi,** cioè valori che rappresentano sotto-insiemi di valori ammissibili, su cui il programma si comporterebbe in modo analogo, restituendo lo stesso risultato.
- Esempio:
  - Il programma che valuta il superamento di un esame universitario, si comporta in un certo modo per valori compresi in [18,30] ed in un alto modo per valori compresi in [1,17].
  - Si dice che il programma ha **due classi di equivalenza.**

Test funzionale: classi equivalenza

- Quando si individuano le classi di equivalenza, si selezionano **due valori puntuali** di riferimento, uno di validità, l'altro di non validità.
- Esempio dell'esame universitario:
  - siano [18,30] e [1,17] le due classi di equivalenza
  - per la classe [18,30] selezioniamo 22 (validità) e 40 o 10 (non validità)
  - Infine, si testa il programma con i due valori verificando che l'output sia quello atteso

Programmazione difensiva

- Spesso, per prevenire errori logici e ridurre il costo del testing e debugging, si può estendere il codice sorgente con istruzioni che coprono casi di test.
- Questa attività è nota come **Programmazione difensiva**, in cui si sviluppa un programma considerando i casi limite che possono caratterizzarne il comportamento, prevenendo e gestendo i malfunzionamenti.

```
if (age < 0 || age > MAX_AGE) {
    range = "???";
} else if (age <= 18) {
    range = "Teenager"
} ...
```

Esercizio

- Un programma che restituisce la media di un array

```
double avg(double a[], int len_a) {
    int i;
    double sum = 0.0;
    for (i=0; i < len_a; i++) {
        sum += a[i];
    }
     return sum / len_a;
}
```

- Individuare casi limite e classi di equivalenza

Esercizio

- Un programma che restituisce la media di un array

```
double avg(double a[], int len_a) {
    int i;
    double sum = 0.0;
    for (i=0; i < len_a; i++) {
        sum += a[i];
    }
    return sum / len_a;
}
```

- Caso limite: len_a=0 porta a divisione per 0

  len_a=-1 porta a media uguale a 0

Esercizio

- Un programma che restituisce la media di un array

```
double avg(double a[], int len_a) {
    int i;
    double sum = 0.0;
    for (i=0; i < len_a; i++) {
        sum += a[i];
    }
    return sum / len_a;
}
```

- Classi di equivalenza:

-len_a>0 (classe di comportamento corretto)

-len_a=0 (classe di comportamento non corretto in cui il programma potrebbe non terminare)

-len_a<0 (classe di comportamento non corretto in cui il programma potrebbe terminare con un risultato non congruo)

Test funzionale: random guessing

- Il tester intuisce possibili errori sulla base della sua esperienza
- Vi sono errori che si ripetono frequentemente in programmi diversi
- Non è una vera tecnica, molto soggettivo perchè affidato al programmatore

Test strutturale

- Casi di test definiti sul concetto di **copertura.**
- Più precisamente, non si considerano i valori di input/output (come nel test funzionale), ma si tende a valutare tutti i possibili percorsi che possono verificarsi, cioè tutti i possibili flussi di esecuzione.
- Esempi emblematici sono le istruzioni
  - *if-then-else*, in cui si devono valutare i due possibili percorsi
  - *switch*, in cui si devono valutare tutti i casi

Test di un programma

- A prescindere dalla metodologia da usare

- il test deve essere eseguito parallelamente alla implementazione
- la direttiva principale è eseguire il test di unità
  - procedura e/o funzione
  - blocco di codice