Ingegneria del Software mod. A
Verifica e validazione: prove

Docente: Tullio Vardanega
tullio.vardanega@math.unipd.it

Definizione

- La prova *software* consiste nella verifica dinamica del comportamento di un programma
  - Su un insieme finito di casi
    - Selezionati nell'ambito del dominio delle esecuzioni possibili, normalmente *infinito*
    - Ciascun caso di prova specifica i valori di ingresso e lo stato iniziale del sistema
    - Ciascun caso di prova deve produrre un esito decidibile
      - *Il problema dell'oracolo*
  - Verificati rispetto ad un comportamento atteso

Caratterizzazione

- Parte vitale del processo di verifica
- Produce una misura della qualità del sistema
  - Aumenta il valore di qualità del sistema identificandone e rimuovendone difetti
- Il suo inizio non deve essere differito al termine della fase di codifica
- Le sue esigenze devono essere tenute in conto nella progettazione del sistema

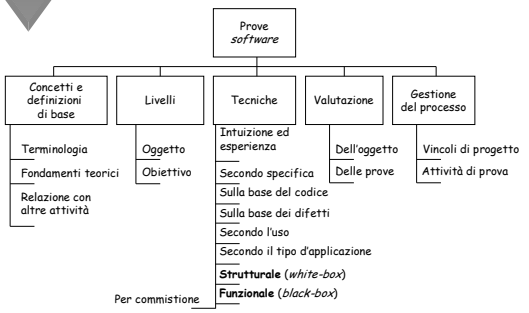
Fattori da bilanciare

- La definizione della strategia di prova richiede un bilanciamento tra
  - La quantità minima di casi di prova sufficienti a fornire certezza adeguate sulla qualità del prodotto
    - Fattore governato da criteri tecnici
  - La quantità massima di sforzo, tempo e risorse disponibile per il completamento della verifica
    - Fattore governato da criteri gestionali

Criteri guida - 1

- Oggetto della prova
  - Il sistema nel suo complesso
  - Parti di esso, in relazione funzionale, d'uso, di comportamento, di struttura
  - Unità singole
- Obiettivo della prova
  - Specificato per ogni caso di prova
  - In termini il più possibile precisi e quantitativi
  - Varia al variare dell'oggetto della prova
  - Risponde alle domande: quante e quali prove

Classificazione delle problematiche



Dentro la classificazione – 1

- Aree di studio e di conoscenza
  - Terminologia
    - Guasto → Errore (difetto) → Malfunzionamento
    - *Fault* → *Error* → *Failure*
  - Fondamenti teorici
    - Decidibilità, testabilità, criteri
  - Oggetto delle prove
    - Unità, aggregati, sistema completo

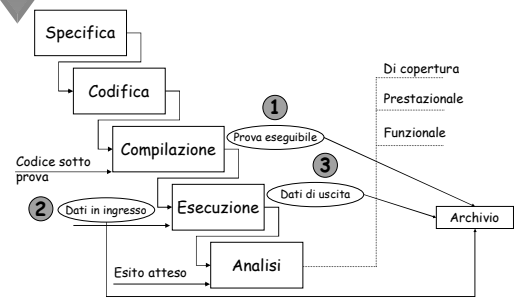
Dentro la classificazione – 2

- Aree di studio e di conoscenza (segue)
  - Obiettivo delle prove
    - Accettazione (collaudo), qualifica, conformità, installazione, regressione, prestazione, …
  - Vincoli di progetto
    - Definizione del processo, definizione dei prodotti, personale addetto alle prove (interno od indipendente), stima e controllo dei costi, criteri di terminazione
  - Attività di prova
    - Pianificazione, specifica dei casi di prova, sviluppo dell'ambiente di prova, esecuzione, valutazione, trattamento dei problemi (anomalie, discrepanze)

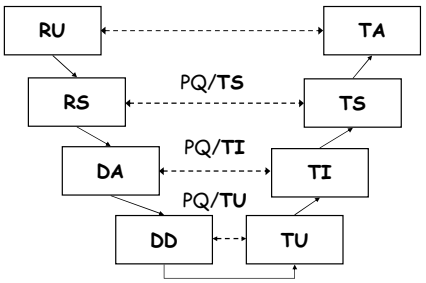
Criteri guida - 2

- Una visione riduttiva, ma espressiva, delle prove
  - "Il processo di eseguire un programma con l'intento di trovarvi difetti"
    The Art of Software Testing, G.J.Myers, Wiley-Interscience, 1979
  - Rappresenta lo spirito critico e l'atteggiamento scettico alla base di una strategia efficace di prova
- La testabilità del *software* va assicurata durante lo sviluppo, non a valle della codifica
  - Disegno architetturale e di dettaglio vanno raffinati finché assicurino testabilità
  - La complessità è nemica della testabilità

Attività di prova



Esecuzione delle attività di prova



Test di unità - 1

- Una unità *software* è composta da uno o più moduli
  - Modulo = componente elementare di progetto di dettaglio
- Ha riferimento nel progetto di dettaglio (DD)
- Completa quando ha provato tutte le unità definite entro le componenti del progetto architetturale (AD)
- ~2/3 dei difetti identificati tramite verifica dinamica vengono rilevati dal test di unità
  - 50% di essi viene identificato da prove strutturali (*white-box*)

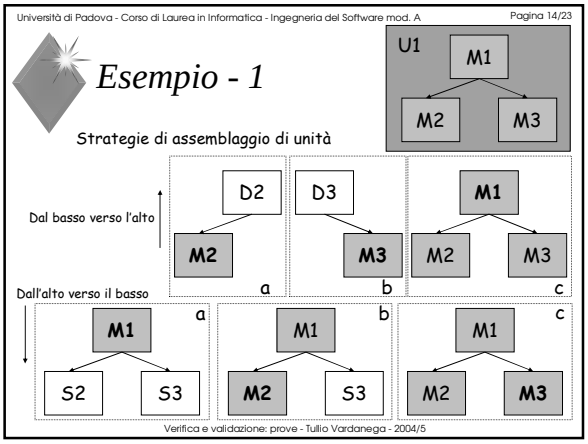
Test di unità - 2

- Il piano di test di unità viene stabilito al termine della fase di progetto di dettaglio
- Per ogni test si definiscono: oggetto, strategia, risorse necessarie e calendario di esecuzione
- La quantità minima di test necessari è sufficiente ad eseguire almeno una volta tutte le *linee di comando* di ciascun modulo dell'unita (*statement coverage*)
- Può essere necessario verificare anche ciascun ramo della logica del flusso di controllo (*branch coverage*)

Regole di integrazione

- Assemblare moduli in modo incrementale
  - I difetti rilevati in un test sono più probabilmente da attribuirsi al modulo ultimo aggiunto
- Assemblare moduli produttori prima dei moduli consumatori
  - La verifica dei primi fornisce ai secondi flusso di controllo e flusso dei dati corretti
- Assemblare in modo che ogni passo di integrazione sia reversibile
  - Consente di retrocedere verso uno stato noto e sicuro

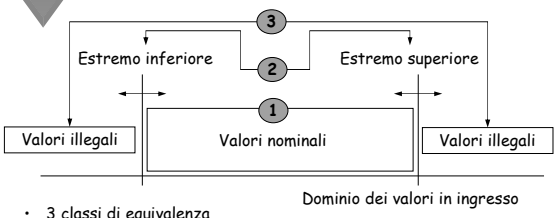
Esempio - 1



Test di unità - 3

- Test funzionale (*black-box*)
  - Complementa il test strutturale, è incapace di accertare correttezza e completezza della logica interna dell'unità
  - Fa riferimento alla specifica dell'unità ed utilizza dati di ingresso capaci di provocare l'esito atteso
  - Ciascun insieme di dati di ingresso che producono un dato comportamento funzionale costituisce un caso di prova
  - Classi di equivalenza invece che infiniti valori di ingresso
    - Valori nella medesima classe producono lo stesso comportamento

Classi di equivalenza



- 3 classi di equivalenza
  - Valori nominali interni al dominio (1)
  - Valori nominali di limite (2)
  - Valori illegali (3)

Test di unità - 4

- Test strutturale (*white-box*)
  - Verifica la logica interna del codice dell'unità
  - Ciascuna prova deve essere progettato per attivare ogni cammino di esecuzione all'interno del modulo
  - Ciascun insieme di dati di ingresso che attivano un percorso costituiscono un caso di prova
  - L'uso di *debugger* ne agevola l'esecuzione ma non esonera dalla progettazione dei casi di prova

Test di unità - 5

```
A11;
if (Cond)
{A12;
 A2;}
A3;
```

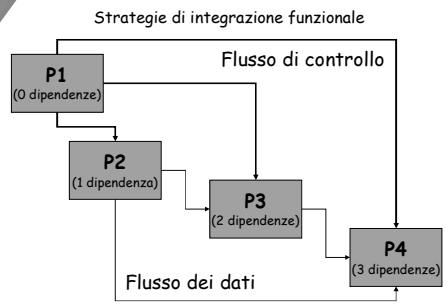
Immagina che A12 sia stato posto per errore nel ramo condizionale (privo di ramo alternativo), mentre dovrebbe essere sempre eseguito senza condizioni

La strategia di *statement coverage* non è capace di rilevare questo errore perché è solo interessata ad eseguire quante più linee di comando possibile
La strategia di *branch coverage* invece lo rileva, eseguendo la prova dove Cond vale Falso ed A12 non viene eseguito

Test di integrazione - 1

- Si applica alle componenti del progetto architetturale
  - L'integrazione di tali componenti costituisce il sistema nella sua interezza
- Logica di integrazione funzionale
  - Seleziona le funzioni da integrare
  - Identifica le componenti che eseguono le funzioni
  - Ordina le componenti per numero di dipendenze
    - Quelle con meno dipendenze hanno precedenza
    - Rappresenta quelle successive ma necessarie con *"driver di test"*
  - Esegue l'integrazione a partire da quelle con precedenza

Esempio - 2



Test di integrazione - 2

- I problemi rilevati durante i test di integrazione
  - Manifestano difetti di progettazione o di insufficiente qualità di verifica a livello unità
  - Un buon test di unità li rende improbabili
- Tanti test quanto serve per
  - Accertare che tutti i dati scambiati attraverso ciascun interfaccia aderiscano alla loro specifica
  - Accertare che tutti i flussi di controllo previsti in specifica siano stati effettivamente realizzati

Test di sistema

- Verifica il comportamento dinamico del sistema completo rispetto ai requisiti *software*
- Ha inizio con il completamento del test di integrazione
- È inerentemente funzionale (*black-box*)
  - Non dovrebbe richiedere conoscenza della logica interna del *software*

Altri tipi di test

- Test di regressione
  - Ripetizione selettiva di test di livello sistema o integrazione
  - Per accertare che modifiche intervenute non causino comportamenti erronei del sistema
  - È oneroso al punto da richiedere ogni sforzo possibile per evitarne la necessità
- Test di accettazione (collaudo)
  - Accerta il soddisfacimento dei requisiti utente