Ingegneria del Software

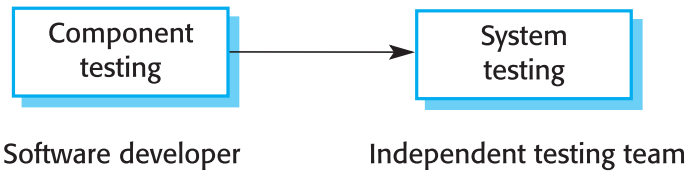
Test del Software

Antonino Staiano
e-mail: **antonino.staiano@uniparthenope.it**

Scopo della lezione

- Comprendere le differenze tra test di convalida e test dei difetti
- Comprendere i principi del test di sistema e dei componenti
- Conoscere le strategie utilizzabili per generare casi di test per il sistema
- Comprendere le caratteristiche essenziali degli strumenti software che supportano l'automazione dei test

Fasi di test



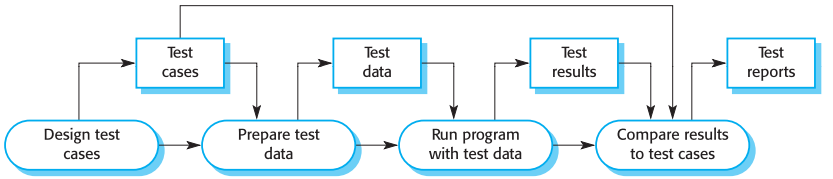
Il processo di test

- Test dei componenti
  - Lo scopo è individuare i difetti
  - Test dei singoli componenti del programma
    - Funzioni
    - Oggetti
    - Componenti riusabili
  - Generalmente è responsabilità dello sviluppatore del componente (ad eccezione talvolta del test per i sistemi critici)
  - I test sono messi a punto a partire dall'esperienza dello sviluppatore
- Test del sistema
  - L'obiettivo è confermare che il sistema soddisfi i suoi requisiti funzionali e non funzionali
  - Test di gruppi di componenti integrati che costituiscono un sistema o un sottosistema
  - E' responsabilità di un team di test indipendente
  - I test sono basati sulla specifica di un sistema

Obiettivi del processo di test

- Test di convalida
  - Dimostra allo sviluppatore ed al cliente del sistema che il software soddisfa i propri requisiti
  - Per i sistemi SW personalizzati
    - Dovrebbe esserci almeno un test per ogni requisito utente e di sistema
  - Per i sistemi SW generici
    - Dovrebbero esserci test per tutte le funzionalità del sistema che saranno incorporate nella release
  - Un test che ha avuto successo mostra che il sistema opera come inteso
- Test dei difetti
  - Dimostra fallimenti o difetti nel software laddove il suo comportamento è errato o non conforme alla sua specifica
  - Ricerca tutti i tipi di comportamenti indesiderati
    - Crash, interazioni non volute con altri sistemi, calcoli errati e corruzione dei dati
  - Un testo che ha avuto successo è un test che porta alla luce un'esecuzione errata evidenziando un difetto nel sistema

Il processo di test del software



Modello generale di test del SW

- I test case sono specifiche degli input del test e degli output attesi dal sistema
  - È contemplata anche una dichiarazione di cosa si sta testando
- I dati del test sono gli input creati per testare il sistema
  - Possono essere anche generati automaticamente

Politiche di test

- Solo un test esaustivo può mostrare che un programma è privo di difetti. Tuttavia, un test esaustivo è impossibile
- Le politiche di test definiscono l'approccio da usare per selezionare i test case del sistema:
  - Dovrebbero essere testate tutte le funzioni a cui si accede tramite i menu
  - Anche le combinazioni delle funzioni a cui si accede attraverso lo stesso menu dovrebbero essere testate
  - Dove è richiesto un input dall'utente, tutte le funzioni dovrebbero essere testate con input corretti ed errati

Politiche di test (cont.)

- Quando le funzioni di un sw sono usate isolatamente, solitamente tutto procede bene
  - Sorgono i problemi quando si impiegano combinazioni di funzioni che non sono state testate insieme
- I programmatori hanno il compito di testare i componenti che hanno sviluppato
  - Completata questa fase, il lavoro viene consegnato ad un altro team che integra i moduli dei diversi sviluppatori, costruisce il sw e testa l'intero sistema
- Responsabile del test è un team separato
  - Per sviluppare i piani di testing lavora partendo dai documenti dei requisiti utente e di sistema

TEST DEL SISTEMA

Test del sistema

- Comporta l'integrazione di componenti in un sistema o sottosistema
- Può comportare il test di un incremento da consegnare al cliente
- Due fasi:
  - Test di integrazione – il team di test ha accesso al codice sorgente del sistema. Il sistema è testato non appena i componenti sono integrati
  - Test di rilascio – il team di test prova il sistema completo da consegnare come una scatola nera (black-box)

Test del sistema (cont.)

- Si può pensare al test di integrazione come al test di sistemi incompleti composti da raggruppamenti di componenti del sistema
- Il test di rilascio testa una release del sistema che si intende consegnare al cliente
- Questi test si sovrappongono soprattutto se si utilizza lo sviluppo incrementale e se il sistema da rilasciare è incompleto
- In generale
  - La priorità del test di integrazione è scoprire il difetti del sistema
  - La priorità del test di rilascio è convalidare che il sistema soddisfi i suoi requisiti
- Nella pratica c'è una parte di test di convalida e di test dei difetti in entrambi i processi

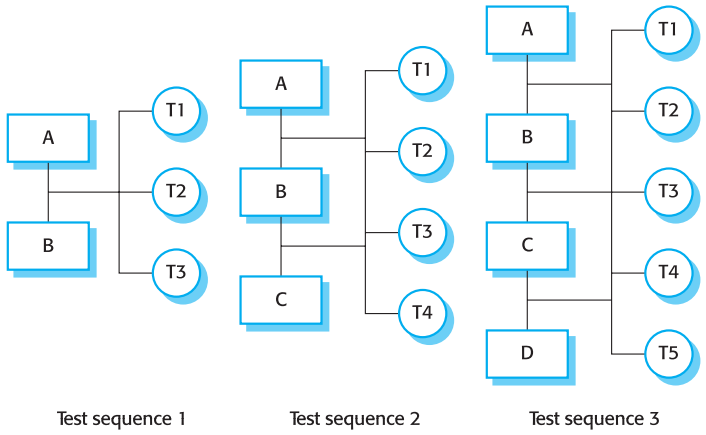
Test del Sistema: *test di integrazione*

- Comporta la costruzione del sistema a partire dai suoi componenti ed il suo test per verificare la presenza di problemi che derivano dall'interazione dei componenti
- Integrazione top-down
  - Sviluppa lo scheletro del sistema e lo popola dei suoi componenti
- Integrazione bottom-up
  - Integra prima i componenti dell'infrastruttura che forniscono i servizi comuni (accesso alla rete, al db) e dopo aggiunge i componenti funzionali
- In pratica, la strategia di integrazione è un misto di tali metodi
  - Componenti infrastrutturali e funzionali aggiunti negli incrementi
- Sia nella strategia top-down che bottom-up si sviluppa codice aggiuntivo per simulare gli altri componenti e permettere al sistema di essere eseguito

Test del Sistema: *test di integrazione* (cont.)

- Per semplificare la localizzazione degli errori, i sistemi dovrebbero essere integrati in modo incrementale
  - A causa delle interazioni complesse tra i vari componenti, può essere difficile identificare dove si trova l'errore dopo aver verificato un output anomalo
- Inizialmente si dovrebbe integrare una configurazione minima e testarla
  - Si aggiungono, poi, altri componenti
  - Si testa l'incremento

Test del sistema
Test di integrazione incrementale



Test del sistema
Test di integrazione incrementale

- Quando viene integrato un nuovo incremento è importante rieseguire i test sui precedenti incrementi oltre che sui nuovi
  - Test di regressione
    - Se evidenzia problemi
      - si deve controllare se questi sono dovuti a incrementi precedenti e individuati da quello nuovo
      - Oppure, se sono causati dall'incremento o dalle funzionalità aggiunte
  - Il test di regressione è costoso e poco pratico senza un supporto automatizzato

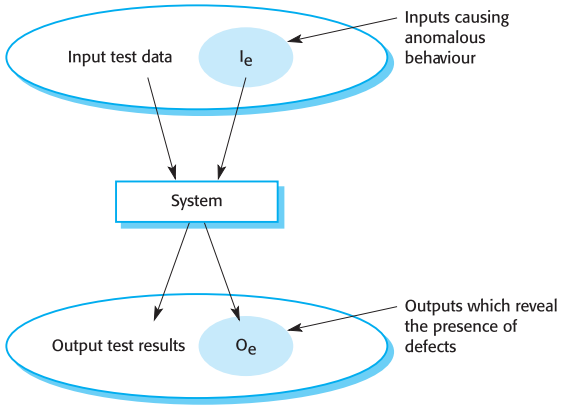
Test del sistema: *approcci per il test di integrazione*

- Convalida dell'architettura
  - Il test di integrazione top-down è più adeguato nello scoprire errori nell'architettura del sistema
- Dimostrazione del sistema
  - Il test di integrazione top-down consente una dimostrazione limitata nelle fasi iniziali dello sviluppo
- Implementazione del test
  - Solitamente più semplice con un test di integrazione bottom-up
- Osservazione del test
  - Ci sono problemi con ambo gli approcci. Può essere necessario lo sviluppo di codice aggiuntivo per osservare i test

Test del sistema: *test di rilascio*

- Il processo di test di un rilascio del sistema che sarà distribuito ai clienti
- Scopo primario è l'incremento del grado di confidenza circa il fatto che il sistema soddisfi i suoi requisiti
- Il test di rilascio è solitamente un test black-box o funzionale
  - Basato solo sulla specifica del sistema
  - I tester non conoscono l'implementazione del sistema

Test del sistema
Test della release: test black-box



Test del sistema
Test della release: linee guida per il test

- Le linee guida consistono in suggerimenti per il team di test per aiutarlo a scegliere test che rileveranno difetti nel sistema
  - Scegliere input che forzano il sistema a generare tutti i messaggi di errore
  - Concepire input che causano buffer overflow
  - Ripetere lo stesso input o serie di input numerose volte
  - Forzare la generazione di output non validi
  - Forzare i risultati del calcolo ad essere troppo grandi o troppo piccoli

Test del sistema

Test della release: scenario di test

A uno studente scozzese che studia storia americana è stato chiesto di scrivere un testo sulla "Mentalità di frontiera nell'America Occidentale dal 1840 al 1880". Per fare questo ha bisogno di trovare fonti da una serie di biblioteche. Si collega la sistema LIBSYS e usa la funzione di ricerca per scoprire se può accedere a documenti originari di quel tempo. Trova fonti in diverse biblioteche universitarie statunitensi e scarica copie di alcuni di questi documenti. Per un documento però, ha bisogno di avere conferma dalla propria università che si tratta di un vero studente e che l'utilizzo è per scopi non commerciali. Lo studente dunque utilizza la funzione di LIBSYS che può richiedere tale permesso e registra la propria richiesta. Se questa viene accettata il documento sarà scaricato sul server della biblioteca, registrato e stampato, e lui sarà avvisato da LIBSYS che riceverà un messaggio di posta elettronica quando il documento sarà disponibile per il ritiro.

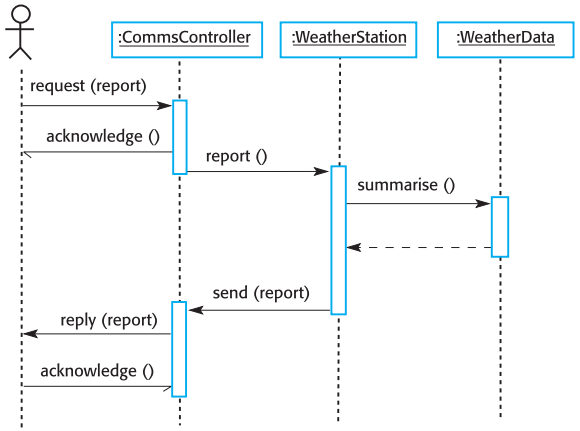
Test del sistema

- Testare il meccanismo di login utilizzando login validi e non per verificare che gli utenti validi siano accettati e quelli non validi siano rifiutati
- Testare le funzionalità di ricerca usando interrogazioni rispetto a fonti conosciute per verificare che il meccanismo di ricerca stia realmente trovando documenti
- Testare le funzionalità di presentazione del sistema per verificare che le informazioni sui documenti siano visualizzate nel modo giusto
- Testare il meccanismo per la richiesta di permesso del downloading
- Testare la risposta via posta elettronica che indica che il documento scaricato è disponibile

Test di sistema: casi d'uso

- I casi d'uso possono rappresentare una base da cui derivare i test di un sistema. Aiutano ad identificare le operazioni da testare ed aiutano la progettazione dei casi di test richiesti
- Possono essere identificati gli input e gli output da creare per i test da un diagramma di sequenza associato

Diagramma di sequenza per la raccolta di dati meteo



Diagrammi di sequenza e casi di test

- Formulare una richiesta di report causerà l'esecuzione della sequenza di metodi

```
ControlloreComunicazioni: richiedi(rapporto)->
StazioneMeteorologica:rapporto()->
DatiMeteorologici:riassumi()
```

- Il diagramma di sequenza può essere utilizzato anche per identificare gli input e gli output che devono essere creati per il test
  1. L'input di una richiesta di report dovrebbe avere una ricevuta associata e infine dovrebbe essere restituito un rapporto; durante il test si dovrebbero creare dati riassuntivi per verificare che il report sia organizzato correttamente
  2. Una richiesta di input per un report alla StazioneMeteorologica genera un rapporto riassuntivo che può essere testato separatamente usando dati grezzi creati in corrispondenza del sommario preparato per il test del ControlloreComunicazioni e verificando che l'oggetto StazioneMeteorologica produca correttamente tale riassunto
  3. I dati grezzi sono usati anche per testare l'oggetto Dati Meteorologici

Test di sistema

Test delle prestazioni

- Parte del test delle release può coinvolgere il test delle proprietà emergenti del sistema, come prestazioni ed affidabilità
- I test delle prestazioni generalmente comportano la pianificazione di una serie di test in cui il carico è incrementato continuamente fino a che le prestazioni del sistema risultano inaccettabili

Test di sistema

Test delle prestazioni: stress test

- Esercitare il sistema oltre il suo carico massimo progettato. Stressare il sistema spesso aiuta a portare alla luce i difetti
- Stressare il sistema è un modo per testare il suo comportamento in caso di fallimento. I sistemi non dovrebbero fallire in modo catastrofico. Lo stress test controlla le perdite inaccettabili di servizi o di dati
- E' particolarmente rilevante per i sistemi distribuiti che possono esibire degradi molto severi quando si verifica un sovraccarico della rete

TEST DEI COMPONENTI

Test dei componenti

- Il test dei componenti o delle unità è il processo che testa i singoli componenti in modo separato
- E' un processo di test dei difetti
- I componenti possono essere:
  - Singole funzioni o metodi all'interno di oggetti
  - Classi di oggetti con numerosi attributi e metodi
  - Componenti composti con interfacce definite per accedere alle rispettive funzionalità

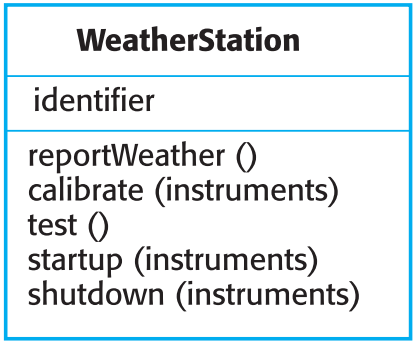
Test dei componenti

Test delle classi di oggetti

- Il test completo di una classe comporta
  - Testare tutte le operazioni associate con un oggetto
  - Impostare ed interrogare tutti gli attributi dell'oggetto
  - Esercitare l'oggetto in tutti i suoi possibili stati
- L'ereditarietà rende più difficile progettare i test per le classi di oggetti poiché le informazioni da testare non sono localizzate

Test dei componenti

Test delle classi di oggetti: interfaccia dell'oggetto Stazione meteo



Test dei componenti

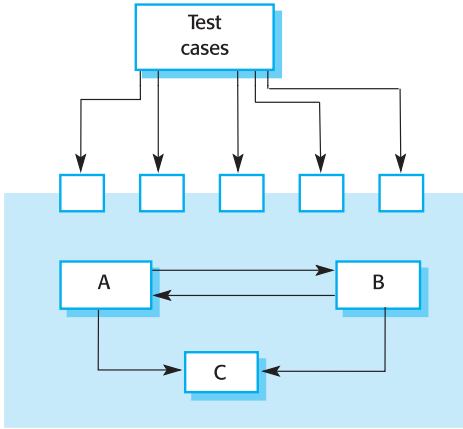
Test delle classi di oggetti: Test della stazione meteo

- Necessità di definire casi di test per *reportWeather*, *calibrate*, *test*, *startup* e *shutdown*
- Usando un modello degli stati, identificare sequenze di transizioni di stato da testare e le sequenze di eventi che causano tali transizioni
- Esempio:
  - Waiting -> Calibrating -> Testing -> Transmitting -> Waiting

Test dei componenti
Test dell'interfaccia

- Gli obiettivi sono l'individuazione di fallimenti dovuti ad errori di interfaccia o ad assunzioni non valide inerenti le interfacce
- Particolarmente importante per lo sviluppo orientato agli oggetti poiché gli oggetti sono definiti mediante le rispettive interfacce

Test dei componenti
Test dell'interfaccia



Test dei componenti
Test dell'interfaccia: tipologie di interfacce

- Interfacce dei parametri
  - Dati passati da una procedura all'altra
- Interfaccia della memoria condivisa
  - Blocchi di memoria che sono condivisi tra procedure o funzioni
- Interfacce procedurali
  - I sottosistemi incapsulano un insieme di procedure che altri sosttosistemi devono invocare
- Interfacce di scambio di messaggi
  - I sottosistemi richiedono i servizi da altri sottosistemi

Test dei componenti
Test dell'interfaccia: errori di interfaccia

- Cattivo uso dell'interfaccia
  - Un componente chiamante invoca un altro componente e commette un errore nell'uso che ne fa dell'interfaccia, ad esempio, i parametri nell'ordine sbagliato
- Incomprensione dell'interfaccia
  - Un componente chiamante racchiude delle assunzioni errate circa il comportamento del componente invocato
    - Esempio: routine di ricerca binaria invocata su un array disordinato
- Errori di temporizzazione
  - I componenti chiamato e chiamante operano a differenti velocità con la conseguenza di accedere ad informazioni non aggiornate
    - Esempio: in sistemi real-time con interfaccia a memoria condivisa con produttori e consumatori

Test dei componenti

Test dell'interfaccia

- Il test dei difetti delle interfacce è complesso
  - Alcuni errori possono manifestarsi solamente sotto condizioni inusuali
  - Esempio: un oggetto implementa una coda come una struttura dati a lunghezza fissa
    - Un oggetto chiamante presume che la coda sia a lunghezza infinita e non si preoccupa di controlli di overflow in fase di inserimento
    - Una tale condizione può essere individuata solo durante il test con la progettazione di casi di test che obbligano la coda a causare un overflow

Test dei componenti

Test dell'interfaccia: linee guida per il test dell'interfaccia

- Progettare i test in modo che i parametri di una procedura chiamata sono agli estremi dei loro range
- Testare sempre i parametri puntatori con puntatori nulli
- Progettare test che causano il fallimento di un componente, se questo viene chiamato attraverso un'interfaccia procedurale
- Usare lo stress test nei sistemi a scambio di messaggi
- Nei sistemi a memoria condivisa, variare l'ordine in cui i componenti sono attivati

PROGETTAZIONE DEI TEST CASE

Progettazione dei casi di test

- Prevede la progettazione dei casi di test (input e output) usati per testare il sistema
- L'obiettivo è creare un insieme di test efficaci nel test di convalida e dei difetti
- Per progettare un caso di test:
  - Si seleziona una funzione del sistema o del componente che si sta testando
  - Si seleziona un insieme di input per eseguire la funzione
  - Si documentano gli output attesi ed i loro campi di variazione
  - Si progettano (ove possibile) controlli automatici per comparare gli output previsti con quelli reali
- Approcci alla progettazione:
  - Test basato sui requisiti
  - Test di partizione
  - Test strutturale

Progettazione dei casi di test

Test basato sui requisiti

- Un principio generale dell'ingegneria dei requisiti è che i requisiti dovrebbero essere testabili
- Il test basato sui requisiti è una tecnica di test di convalida dove si considera ciascun requisito e si deriva un insieme di test per quel requisito

Progettazione dei casi di test

Test basato sui requisiti: LIBSYS

L'utente sarà in grado di cercare all'interno di tutto l'insieme iniziale di database o di selezionarne un sottoinsieme.

Il sistema fornirà all'utente i visualizzatori adatti per leggere i documenti memorizzati.

Ad ogni ordine deve essere assegnato un identificatore unico (ORDER_ID) che l'utente deve poter copiare nell'area di memorizzazione permanente dell'account.

Progettazione dei casi di test

Test basato sui requisiti: test LIBSYS

- Inizializzare le ricerche utente per articoli che si sa essere presenti o non presenti, con l'insieme dei database comprendente un solo database
- Inizializzare le ricerche utente per articoli che si sa essere presenti o non presenti, con l'insieme dei database comprendente due database
- Inizializzare le ricerche utente per articoli che si sa essere presenti o non presenti, con l'insieme dei database comprendente più di due database
- Selezionare un database dall'insieme dei database e inizializzare le ricerche per articoli che si sa essere presenti o non presenti
- Selezionare più di un database dell'insieme dei database e inizializzare le ricerche per articoli che si sa essere presenti o non presenti

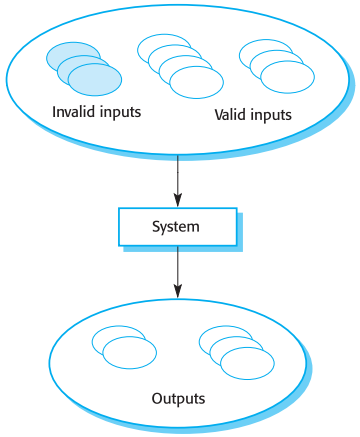
Progettazione dei casi di test

Test di partizione

- I dati di input ed i risultati di output spesso ricadono in classi differenti dove tutti i membri di una classe sono in relazione
- Ognuna di queste classi è una partizione di equivalenza o dominio dove il programma si comporta in modo equivalente per ogni membro della classe
- I casi di test dovrebbero essere scelti da ogni partizione

Progettazione dei casi di test

Test di partizione: partizionamento di equivalenza



Progettazione dei casi di test

Test di partizione

- Una regola pratica per selezionare i casi di test dalle partizioni è
  - Scegliere casi di test sui confini delle partizioni e test vicini al punto centrale
  - I valori di confine spesso sono atipici (ad esempio, 0 potrebbe comportarsi in modo diverso dagli altri numeri non negativi) ed ignorati dagli sviluppatori
    - I fallimenti, generalmente, capitano proprio quando si elaborano valori atipici

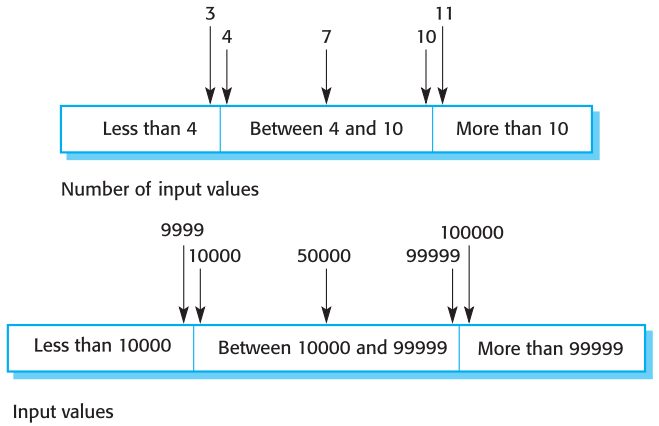
Progettazione dei casi di test

Test di partizione

- Le partizioni sono identificate usando le specifiche del programma o la documentazione utente e, per esperienza, dalle classi di valori di input che potrebbero individuare errori
  - Esempio: programma che accetta da 4 a 8 input che sono numeri interi a cinque cifre maggiori di 10000

Progettazione dei casi di test

Test di partizione: partizioni di equivalenza



Progettazione dei casi di test

Test di partizione: Specifica routine di ricerca

```
procedure Search (Key : ELEM ; T: SEQ of ELEM;
       Found : in out BOOLEAN; L: in out ELEM_INDEX) ;

Pre-condition
       -- the sequence has at least one element
       T'FIRST <= T'LAST
Post-condition
       -- the element is found and is referenced by L
       ( Found and T (L) = Key)
or
       -- the element is not in the array
       ( not Found and
       not (exists i, T'FIRST >= i <= T'LAST, T (i) = Key ))
```


Progettazione dei casi di test

Test di partizione: Routine di ricerca – partizioni di input

- Input conformi alle pre-condizioni
- Input per cui una pre-condizione non è mantenuta
- Input in cui l'elemento chiave è un membro dell'array
- Input dove l'elemento chiave non è membro dell'array

Progettazione dei casi di test

Test di partizione: Linee guida per il test (sequenze)

- Testare il software con sequenze che hanno un solo valore
  - Spesso si assume che le sequenze siano solo costituite da più valori
- Usare sequenze di dimensioni differenti in test differenti
  - Diminuisce la probabilità che un programma difettoso produca accidentalmente risultati corretti
- Concepire test che fanno in modo di accedere a agli elementi primo, centrale ed ultimo della sequenza
  - Rivela problemi ai confini delle partizioni
- Testare con sequenze di lunghezza zero

Routine di ricerca – partizioni di input

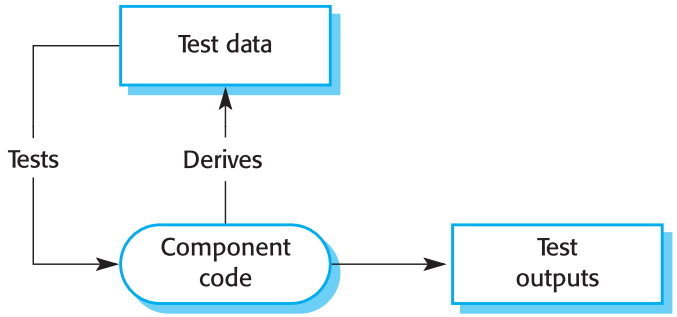
Sequence	**Element**
Single value	In sequence
Single value	Not in sequence
More than 1 value	First element in sequence
More than 1 value	Last element in sequence
More than 1 value	Middle element in sequence
More than 1 value	Not in sequence

Input sequence (T)	**Key** (Key)	**Output** (Found, L)
17	17	true, 1
17	0	false, ??
17, 29, 21, 23	17	true, 1
41, 18, 9, 31, 30, 16, 45	45	true, 7
17, 18, 21, 23, 29, 41, 38	23	true, 4
21, 23, 29, 33, 38	25	false, ??

Progettazione dei casi di test
Test strutturale

- Chiamato anche test white-box
- Deriva i casi di test in accordo alla struttura del programma. La conoscenza del programma è usata per identificare casi di test aggiuntivi
- L'obiettivo è di mettere in esercizio tutte le istruzioni del programma (non tutte le combinazioni dei percorsi)

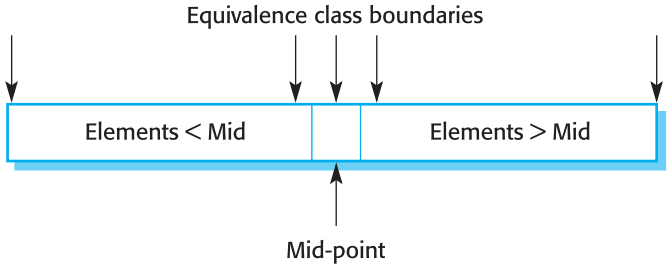
Progettazione dei casi di test
Test strutturale



Ricerca binaria – partizioni di equivalenza

- Pre-condizioni soddisfatte, l'elemento chiave è nell'array
- Pre-condizioni soddisfatte, l'elemento chiave non è nell'array
- Pre-condizioni non soddisfatte, l'elemento chiave è nell'array
- Pre-condizioni non soddisfatte, l'elemento chiave non è nell'array
- L'array di input ha un solo valore
- L'array di input ha un numero pari di valori
- L'array di input ha un numero dispari di valori

Ricerca binaria – partizioni di equivalenza



Ricerca binaria – casi di test

Input array (T)	**Key** (Key)	**Output** (Found, L)
17	17	true, 1
17	0	false, ??
17, 21, 23, 29	17	true, 1
9, 16, 18, 30, 31, 41, 45	45	true, 7
17, 18, 21, 23, 29, 38, 41	23	true, 4
17, 18, 21, 23, 29, 33, 38	21	true, 3
12, 18, 21, 23, 32	23	true, 4
21, 23, 29, 33, 38	25	false, ??

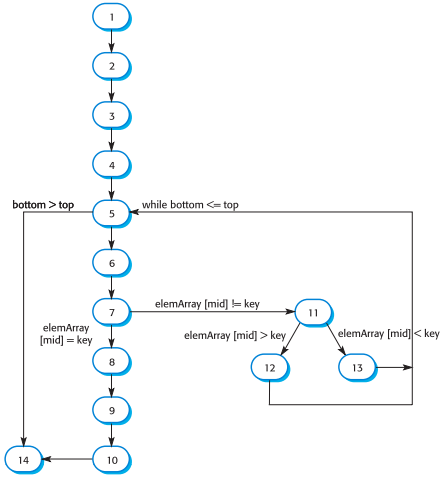
Progettazione dei casi di test

Test del percorso

- L'obiettivo del test del percorso è di assicurare che l'insieme dei casi di test è tale che ogni cammino attraverso il programma sia eseguito almeno una volta
- Il punto di partenza per il test del percorso è un grafo del flusso di programma i cui nodi rappresentano le decisioni di programma e gli archi rappresentano il flusso del controllo
- Le istruzioni con condizioni sono pertanto nodi all'interno del grafo di flusso

Progettazione dei casi di test

Test del percorso: grafo di flusso per la ricerca binaria



Progettazione dei casi di test

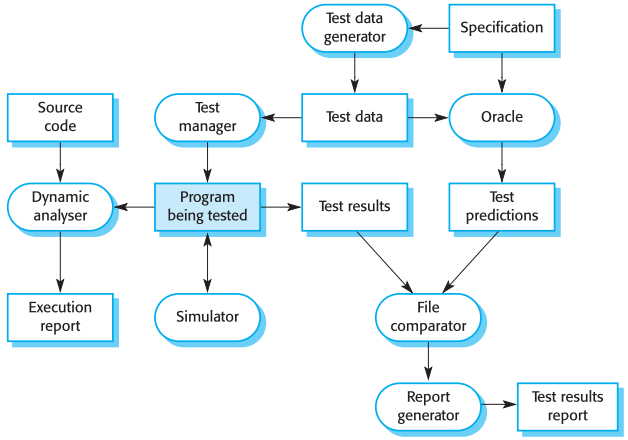
Test del percorso: percorsi indipendenti

- Percorso indipendente:
  - Attraversa almeno un nuovo arco nel grafo di glusso
- 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14
- 1, 2, 3, 4, 5, 14
- 1, 2, 3, 4, 5, 6, 7, 11, 12, 5, …
- 1, 2, 3, 4, 6, 7, 2, 11, 13, 5, …
- I casi di test dovrebbero essere determinati in modo tale da eseguire tutti i percorsi indipendenti
- Può essere usato un analizzatore dinamico di programma per controllare che tutti i percorsi siano stati eseguiti

Automatizzazione dei test

- Il testing è una fase di processo costosa. I workbench di test forniscono una gamma di strumenti per ridurre il tempo richiesto ed il costo totale del test
- Sistemi come Junit supportano l'esecuzione automatica dei test
  - Insiemi di classi Java che l'utente estende per creare un ambiente automatico di test
    - Ogni singolo test viene implementato come oggetto e un esecutore dei test li elabora

Un workbench di test



Adattamento del workbench di test

- Per i grandi sistemi gli strumenti devono essere configurati e adattati al sistema specifico
  - Possono essere sviluppati degli script per i simulatori dell'interfaccia utente e pattern per i generatori dei dati di test
  - Possono essere preparati manualmente a scopo comparativo gli output del test
  - Possono essere sviluppati comparatori di file a scopo speciale

Riassumendo (I)

- Il test può mostrare la presenza di fallimenti in un sistema; non può invece provare che non ci sono fallimenti
- Gli sviluppatori dei componenti sono responsabili del test dei componenti; il test del sistema è sotto la responsabilità di un team separato
- Il test di integrazione è il test di incrementi del sistema; il test di release coinvolge il test di un sistema da rilasciare al cliente
- Utilizzare l'esperienza e linee guida per progettare i casi di test nel test dei difetti

Riassumendo (II)

- Il test di interfaccia è progettato per scoprire i difetti nelle interfacce di componenti composti
- Il partizionamento di equivalenza è un modo per scoprire casi di test – tutti i casi in una partizione dovrebbero comportarsi allo stesso modo
- L'analisi strutturale si basa sull'analizzare un programma per determinarne i casi di test
- L'automatizzazione dei test riduce i costi della fase di test supportando il processo di test con una gamma di strumenti software