Verifica e Validazione del Software

Andrea Polini

Ingegneria del Software
Corso di Laurea in Informatica

Verifica e Validazione - generalità

Dunque il problema a questo punto è capire se il sistema risolve effettivamente i problemi per cui è stato concepito e se lo fa "correttamente"

Are we building the right product?

vs.

Are we building the product right?

Verifica: il prodotto che stiamo sviluppando è corretto?

Validazione: stiamo sviluppando il corretto prodotto?

Verifica e Validazione - generalità

Dunque il problema a questo punto è capire se il sistema risolve effettivamente i problemi per cui è stato concepito e se lo fa "correttamente"

Are we building the right product?

vs.

Are we building the product right?

Verifica: il prodotto che stiamo sviluppando è corretto?

Validazione: stiamo sviluppando il corretto prodotto?

V&V

In qualche modo le attività di V&V intendono “rassicurare” l’utilizzatore che il sistema è “adatto allo scopo”

Livello di fiducia che si intende fornire non deve considerarsi un concetto assoluto ma dipende da diversi fattori:

- Funzione del software all’interno dell’organizzazione
- Attese dell’utente
- Politiche di mercato

Approaches to V&V

Approcci statici: analisi statica dei sorgenti e altri documenti di progetto (ispezione e revisione, verifica formale) - utili a verificare conformità e coerenza nelle fasi dello sviluppo (non permettono validazione)

Approcci dinamici: testing - prevede esecuzione del software o prototipi al fine di scoprire difetti (validazione e verifica). Tecnica di gran lunga più utilizzata per la verifica e validazione del software

E poi?

- Debugging: Attività che si occupa della localizzazione del guasto che ha generato un fallimento.
- Rivalidazione e Regression Testing

Ispezione del software

Il software è analizzato per trovare errori, omissioni ed anomalie

Può sembrare inefficace ma molte esperienze reali hanno dimostrato il contrario. In molti casi ispezione si è rivelata più efficace del testing

Vantaggi sul testing:

- Non possono verificarsi fenomeni di mascheramento degli errori
- applicabilità a parti del software non ancora eseguibili a causa della mancanza di componenti di interazione
- possibiltà di verificare caratteristiche non verificabili dal testing (aderenza agli standard, ed a stile di codifica

Processo di ispezione

attori

Come tutte le atticità si svolge strutturando un processo e degli attori che svolgono specifiche attività

Attori:

- Autore: responsabile della produzione del oggetto sotto ispezione
- Ispettore: responsabile della ricerca di errori, inconsistenze, omissioni
- Lettore: responsabile della lettura
- Scriba: responsabile della registrazione delle decisioni del meeting
- Moderatore: modera la riunione di ispezione
- Responsabile del processo: responsabile del processo di ispezione. Definisce il team ed i ruoli ed organizza il meeting

Processo di ispezione

attività

- Pianificazione: si decidono partecipanti e tempi del processo
- Overview: si presenta l'oggetto del attività ispettiva
- Preparazione Individuale: gli ispettori studiano l'oggetto e cercano di identificare possibili problemi
- Riunione di ispezione: nella riunione si discutono i possibili errori identificati - non si discutono soluzioni
- Rielaborazione: gli errori identificati come tali vengono corretti
- Follow-up: decidere se procedere con ulteriori fasi ispettive

Uso di checklist

- Errori sui dati: inizializzazione delle variabili, uso di nomi per le costanti, indici di vettori, buffer overflow ...
- Errori nel controllo: specifica delle condizioni, terminazione dei cicli, uso delle parentesi, `case` comprende tutte le possibilità, uso dei comandi di `break` ...
- Errori negli input/output: uso di tutte le variabili, assegnamento di valori alle variabili di uscita, comportamento in caso di ingressi imprevisti ...
- Errori nelle interfacce: verifica del corretto uso dei parametri, ordine dei parametri ...
- Errori nella gestione della memoria: uso e modifica di una struttura con collegamenti, allocazione e deallocazione dinamica dello spazio ...
- Errori nella gestione delle eccezioni: possibili sorgenti di errore sono considerate ...

Considerazioni quantitative

Produttività stimata:

- Overview - 500 LOC/h
- Preparazione Individuale - 250 LOC/h
- Riunione di ispezione - da 90 a 125 LOC/h

Questi parametri portano ad una stima di un “giorno uomo” per 100 LOC.

Il testing è tipicamente molto più costoso

Analisi statica e tecniche di automazione

Sono disponibili strumenti automatici per la ricerca ed evidenziazione di anomalie.

- Analisi del controllo di flusso: identificazione di codice non raggiungibile
- Analisi dell'uso dei dati: variabili usate prima dell'inizializzazione, variabili dichiarate ma non usate
- Analisi delle interfacce: uso scorretto di interfacce per linguaggi con controllo debole dei tipi
- Analisi del flusso di informazioni: permette di definire relazioni tra le varie variabili
- Analisi dei cammini: identificazione di tutti i possibili cammini nel sistema ed delle linee di codice correlate. Possibili cammini non possibili.

Diversi linguaggi di programmazione presentano problematiche differenti

Verifica formale

Applicabili nel caso di uso di formalismi di specifica formali.

Correttezza può essere derivata con argomenti matematici

Si tenga be presente che:

- specifiche formali non sono comprensibili a utente
- richiedono staff estremamente qualificato
- tempi di verifica possono essere estremamente lunghi
- specifica formali potrebbe non rispecchiare i requisiti reali
- le prove potrebbero essere errate
- prove potrebbero fare assunzioni non valide riguardo l'ambiente finale d'uso

Testing

Il testing del software prevede l'esecuzione di alcuni "esperimenti" in un ambiente controllato al fine di poter acquisire sufficiente fiducia sul suo funzionamento. Testing riguarda tipicamente proprietà funzionali ma può riguardare anche caratteristiche extra-funzionali.

Due obiettivi differenti:

- Dimostrare che il sistema risponde alle esigenze (almeno un test per ogni requisito)
- Scoprire guasti (cercare di far manifestare tutti i possibili guasti)

Il testing non può dimostrare l'assenza di guasti ma solo la loro presenza

E.W. Dijkstra

Testing

Il testing del software prevede l'esecuzione di alcuni "esperimenti" in un ambiente controllato al fine di poter acquisire sufficiente fiducia sul suo funzionamento. Testing riguarda tipicamente proprietà funzionali ma può riguardare anche caratteristiche extra-funzionali.

Due obiettivi differenti:

- Dimostrare che il sistema risponde alle esigenze (almeno un test per ogni requisito)
- Scoprire guasti (cercare di far manifestare tutti i possibili guasti)

Il testing non può dimostrare l'assenza di guasti ma solo la loro presenza

E.W. Dijkstra

Genesi dei Fallimenti

Errore: si riferisce al processo logico che porta ad una non corrispondenza del software rispetto a quanto necessario come risultato dell'attività compiuta da un progettista/programmatore (il quale può commettere un errore)

Guasto: parte del progetto che in qualche modo contiene la codifica dell'errore

Fallimento: manifestazione del guasto con osservazione di un funzionamento scorretto

Testing

- Strategie di testing - pianificazione e gestione del processo di testing
- Tecniche di testing - approcci alle fasi di derivazione ed esecuzione del testing

Caratteristiche comuni di qualsiasi strategia:

- Revisione tecnica formale
- Testing procede per fasi di integrazione
- Importante applicare più tecniche
- Coinvolge sviluppatore ed eventualmente team esterno (ITG)
- Testing e debugging si complementano

Un po' di tassonomia

- Test Case: esecuzione volta ad evidenziare la presenza di un guasto
- Test suite: insieme di casi di test
- Driver: strumento utilizzato per eseguire i test sul sistema
- Sistem Under Test (SUT): il sistema che in un certo momento è sottoposto al test
- Stubs/Mocks: oggetti necessari a riprodurre componenti mancanti al fine del test di integrazione
- Oracolo del Test: entità capace di valutare i risultati del test
- Workload: carico di lavoro a cui un SUT viene sottoposto tipicamente per evidenziare guasti dovuti a errori nel progetto della concorrenza o per evidenziare degradi di QoS
- ...

Processo di testing

Come tutte le attività richiede definizione di processo:

- Design dei casi di test
- Selezione dei dati di test
- Esecuzione dei test
- Verifica e comparazione dei risultati osservati con quelli attesi

Quando abbiamo testato abbastanza? – Adeguatezza

- Approcci empirici
- Teoria della reliability

Processo di testing

Come tutte le attività richiede definizione di processo:

- Design dei casi di test
- Selezione dei dati di test
- Esecuzione dei test
- Verifica e comparazione dei risultati osservati con quelli attesi

Quando abbiamo testato abbastanza? – Adeguatezza

- Approcci empirici
- Teoria della reliability

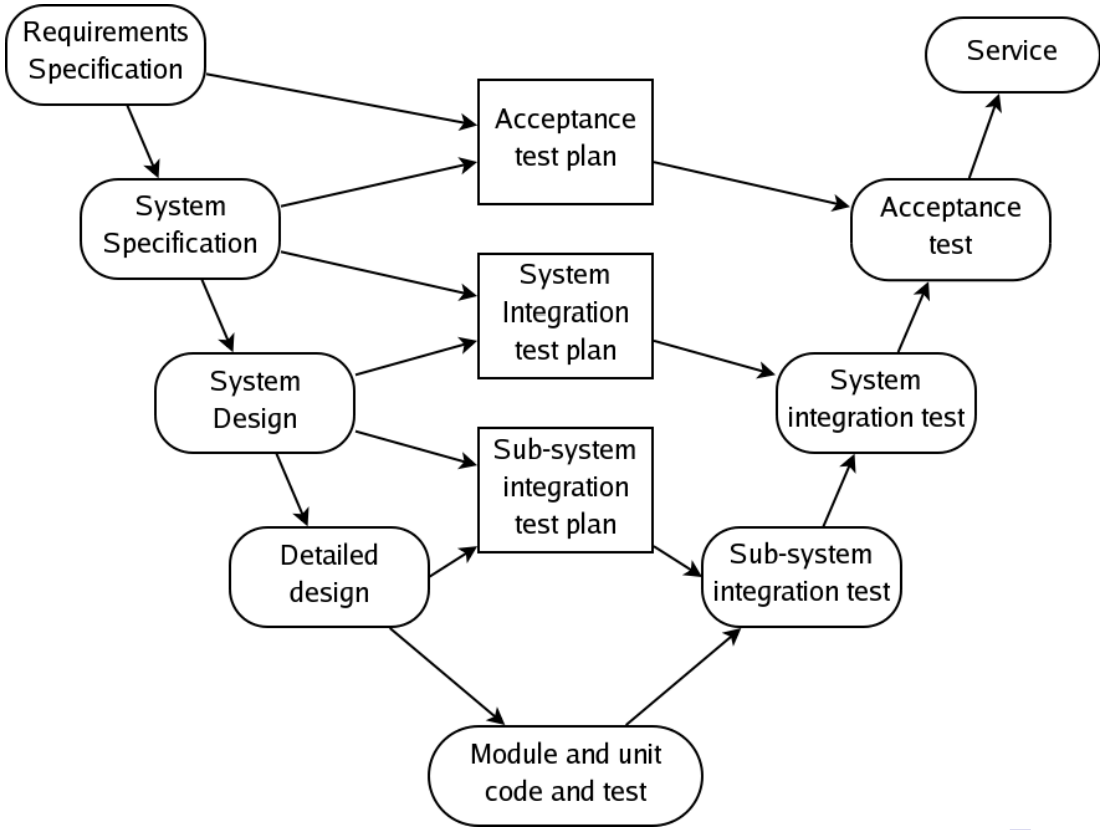
Fasi del testing

Le attività di testing tipicamente in diversi momenti si focalizzano su obiettivi differenti:

- Testing di unità (Unit test)
- Testing di integrazione (Integration test)
  - bottom-up
  - top-down
  - big-bang
- Testing di rilascio (System test)
- Testing di caratteristiche extra-funzionali

Fasi del testing e V model

- Unit testing
- Integration testing
- System Testing
- Acceptance Testing



Unit testing

Riguarda il testing di elementi software in isolamento, tramite uso di stub e/o mock

Strategie di definizione dei casi di test:

- Flusso dei dati
- Flusso del controllo
- Espressioni condizionali
- Dati di input
- Cicli

Unit testing in ambito OO

Unit testing

Riguarda il testing di elementi software in isolamento, tramite uso di stub e/o mock

Strategie di definizione dei casi di test:

- Flusso dei dati
- Flusso del controllo
- Espressioni condizionali
- Dati di input
- Cicli

Unit testing in ambito OO

Unit testing

Riguarda il testing di elementi software in isolamento, tramite uso di stub e/o mock

Strategie di definizione dei casi di test:

- Flusso dei dati
- Flusso del controllo
- Espressioni condizionali
- Dati di input
- Cicli

Unit testing in ambito OO

Integration testing

Strategie di Integrazione:

- top-down
- bottom-up
- big-bang

Regression Testing

Integration testing in ambito OO

Integration testing

Strategie di Integrazione:

- top-down
- bottom-up
- big-bang

Regression Testing

Integration testing in ambito OO

System Testing

System testing di natura funzionale considera quei requisiti generali del sistema

Caratteristiche extrafunzionali generalmente sperimentabili affidabilmente soltanto a questo livello (proprietà non composizionali):

- usability
- security testing
- stress testing
- performance testing
- ...

System Testing

System testing di natura funzionale considera quei requisiti generali del sistema

Caratteristiche extrafunzionali generalmente sperimentabili affidabilmente soltanto a questo livello (proprietà non composizionali):

- usability
- security testing
- stress testing
- performance testing
- ...

Validation Testing

Test che cercano di valutare la soddisfazione del cliente e l'aderenza alle sue esigenze.

Tecniche principali:

- alfa testing
- beta testing

Debugging

Riguarda le attività di localizzazione e risoluzione dei bug.
Reso difficile da diversi fattori:

- sintomo e causa possono essere “distanti”
- sintomo si può manifestare sporadicamente
- il sintomo potrebbe non esser dovuto a guasti logici nel programma
- il sintomo può essere dovuto a temporizzazione ed interleaving particolari
- difficoltà di riproduzione delle condizioni di errore
- sintomo dovuto a diversi task in esecuzione su diversi processori

Una volta individuato il guasto dovrà essere risolto.
Attività delicata in quanto ogni nuova riga di codice può introdurre nuovi guasti

Caratteristiche

Una buon test è caratterizzato da:

- Alta probabilità di scoprire un errore
- non ridondanza
- è il migliore della sua categoria
- né troppo semplice né troppo complesso

Tipologie di testing

Si distinguono diverse tipologie di testing in base ai manufatti utilizzati per la generazione e la valutazione dei risultati di un test:

- Black-box: strategie basate su modelli e requisiti
- White-box: strategie basate su codice
- Grey-box: strategie che si basano su astrazioni del codice

Testing White box

Anche detto Testing Strutturale si basa sulla conoscenza dei sorgenti dell'oggetto da testare. Principi di definizione di una test suite:

- Copertura dei cammini
- Copertura delle condizioni
- Copertura dei valori di confine dei cicli
- Copertura delle strutture di dato

Flow Graph e coperture

Flow graph rappresentazione schematica della struttura di controllo di un programma. Tecniche di copertura voglio coprire la struttura con diversi livelli di dettaglio:

- Copertura delle righe di codice (statement coverage)
- Copertura delle condizioni (Branch coverage)
- Copertura dei cammini (Path coverage)

Copertura delle condizioni

Obiettivo è esercitare le condizioni cercando di fare assumere a queste sia il valore vero che il falso:

- condizioni semplici: copertura delle condizioni o delle decisioni
- condizioni composite: combinazione dei risultati associati a condizioni semplici
  - combinazioni delle condizioni?
  - MC/DC: approccio che riduce le possibili combinazioni da provare

Example

Si consideri un programma che prende in input due interi `x` e `y`, e ritorna un intero `z` in accordo alla seguente tabella:

x<0	y<0	output(z)
`true`	`true`	`foo1(x,y)`
`true`	`false`	`foo2(x,y)`
`false`	`true`	`foo2(x,y)`
`false`	`false`	`foo1(x,y)`

Si applichi la test suite $T = \{t_1 :< x = -3, y = -2 >, t_2 :< x = -4, y = 2 >\}$ al programmma seguente

```
begin
  int x,y,z;
  input(x,y);
  if (x<0 and y<0)
    z=foo1(x,y);
  else
    z=foo2(x,y);
  output(z);
end
```

Data flow testing

I casi di test cercano di coprire i cammini identificati da azioni di definizione ed uso di una variabile.

- Definizione: punto del codice nel quale il valore di una variabile viene definito
- Uso: punto del codice dove il valore di una variabile viene utlizzato
  - c-use: uso in un passo computazionale
  - p-use: uso della variabile in un predicato

Strategie tendono a coprire relazioni di tipo def-use che possono essere rappresentate in un grafo

Testing di cicli

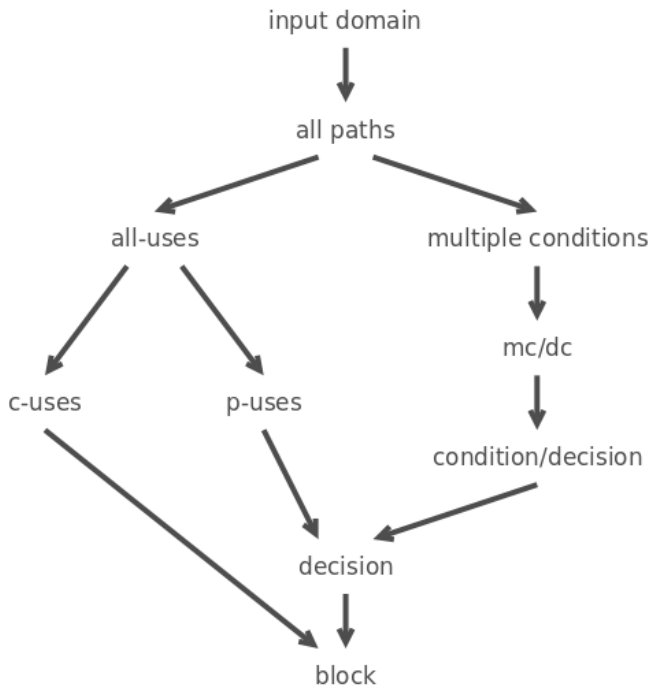
I casi di test possono poi imporre vincoli sulla copertura “finita” di cicli:

- Cicli semplici
- Cicli annidati
- Cicli concatenati
- Cicli non strutturati

Control flow vs. Data Flow

The subsumes relation

Un criterio di copertura C1 subsumes un criterio di copertura C2 sse ogni volta che il criterio C1 viene soddisfatto allora anche il criterio C2 sarà soddisfatto



Rappresentazione della relazione subsumes

Accenno al Mutation testing

Le tecniche di Mutation Testing danno una misura di adeguatezza empirica per la test suite. L'idea di base è quella di distribuire nel codice guasti e poi contare il numero di guasti che vengono scoperti dalla test suite. Un basso numero di mutanti uccisi suggerisce di migliorare la test suite.

Testing Black-box

- Testing basato su macchine a stati: i casi di test cercano di coprire stati o cammini della macchina
- Partition based testing: strategie definite sui domini delle variabili di ingresso
- Tecniche combinatorie: quando più domini di input si cerca di combinare i domini
- Analisi dei valori limite: strategie che considerano i valori di confine di domini di valori come particolarmente interessanti
- ...

Testing OO

Nel caso del testing di sistemi OO l'unità di testing è tipicamente costituita dalla classe. Il testing viene complicato da due fattori:

- incapsulamento
- ereditarietà

Testing di integrazione procederà integrando via via nuove classi.

- Tecnica del Built-in testing - classi instrumentate con casi di test da utilizzare per valutare le ipotesi fatte sulle classi serventi.

JUnit

- Cos'è?
- Quando e come usarlo?
- Documentazione? (`http://www.junit.org/`)
- Integrato in Eclipse
  - usa annotazioni - @Test,@BeforeEach,@AfterEach,@BeforeAll,@AfterAll

JUnit

- assertEquals(expected, actual)
- assertEquals(message, expected, actual)
- assertEquals(expected, actual, delta) - used on doubles or floats, where delta is the difference in precision
- assertEquals(message, expected, actual, delta) - used on doubles or floats, where delta is the difference in precision
- assertFalse(condition)
- assertFalse(message, condition)
- assertNotNull(object)
- assertNotNull(message, object)
- assertNotSame(expected, actual)
- assertNotSame(message, expected, actual)
- assertNull(object)
- assertNull(message, object)
- assertSame(expected, actual)
- assertSame(message, expected, actual)
- assertTrue(condition)
- assertTrue(message, condition)
- fail()
- fail(message)
- failNotEquals(message, expected, actual)
- failNotSame(message, expected, actual)
- failSame(message)