Introduzione al collaudo del software

Indice

Introduzione al collaudo del software

The proof of the pudding is in the eating.
Proverbio inglese del '600. [1]

Scopo della lezione

- concisa panoramica introduttiva sul collaudo del software, basata su (Springl 1998), con aggiornamenti essenziali
- proposte di temi per approfondimenti e discussioni

In questa lezione:

- **il progetto del collaudo del software (Sez. 10.1), principi e metodi**
- **strategie di collaudo del software (Sez. 10.2)**
- **metriche nel collaudo del software (Sez. 10.3)**
- **collaudo di software ad oggetti (Sez. 10.4)**
- **altri aspetti del collaudo del software, per approfondimenti e discussioni (Sez. 10.5)**

Il progetto del collaudo del software

Il collaudo del software è un'attività complessa, essa stessa oggetto di

- **pianificazione (richiede risorse)**
- **progettazione (della sua architettura)**

Principi di collaudo del software

Principi utili alla pianificazione e al progetto del collaudo del software:

- **basato sulla specifica dei requisiti**
- **tempo e risorse per il collaudo hanno dei limiti**
  il collaudo non mostra la correttezza, bensì l'*errore*
- **il collaudo non può essere esaustivo**
  (impossibilità di esplorare *tutte* le possibilità di esecuzione)
- **efficacia delle risorse impiegate per il collaudo**
  - **risorse umane**
    competenze specifiche ed esperienza di collaudo
  - **risorse materiali**
    automazione di procedure di collaudo

- **pianificazione precoce del collaudo**
  il collaudo comincia dalla specifica dei requisiti! (?)
- **il collaudo procede dalle unità al sistema: bottom-up**

Metodi di collaudo del software

progetto di casi di collaudo (test case)

- **white-box testing**
  collaudo di *struttura interna*
  - **collaudo di sequenze di base**
    sequenze di esecuzione definite da un insieme di base
  - **collaudo di strutture di controllo**
    complementa il metodo precedente
- **black-box testing**
  collaudo di comportamento all'*interfaccia*
  - **graph-based**
    nodi del grafo = oggetti del collaudo
  - **equivalence partitioning**
    partizionamento dell'input
  - **boundary value analysis**
    frontiere delle classi di equivalenza
  - **condition (back-to-back) testing**
    sviluppo di due versioni

Strategie di collaudo del software

dipendono dall'ambito del collaudo:

- **di unità**
- **di integrazione**
- **funzionale**
- **di sistema**

Strategie nel collaudo di unità

collaudo locale di una unità software
granularità : package, modulo, sottoprogramma, classe, metodo, ...

- **in prevalenza, metodi white-box,** ad es. (JUnit)
- **metodi black-box applicabili se:**
  - **sono specificati requisiti all'interfaccia, ad es.:**
    - *vincoli OCL* (pre-, post-condizioni) su *modelli UML di unità,* ad es. diagrammi di stato
    - in Eiffel, specifiche BON

    secondo la metodologia del Design by Contract (Meyer 1992)
  - **il codice di collaudo sostituisce il codice di contesto**

Strategie nel collaudo di integrazione

collaudo di più unità software interagenti
ambito : sistema, sottosistema, package, modulo, ...

- **si assume una gerarchia del software in livelli**
- **strategie: top-down, bottom-up, big-bang**
  - **top-down:**
    - :) ben si presta alla *prototipazione partecipativa*
    - :(richiede l'uso di unità fittizie (*stubs*)
  - **bottom-up:**
    - :) ben si presta al collaudo precoce di *sottosistemi critici*
    - :(richiede lo sviluppo di *test-drivers*
      ... e il collaudo dei test-drivers (pericolo di regresso infinito?)
  - **big-bang:**
    collaudo di integrazione solo al livello di sistema
    - :) richiede *meno sforzo* di sviluppo di codice di collaudo
    - :(può costare *più sforzo* di correzione (e collaudo)

Strategie nel collaudo funzionale

collaudo black-box della funzionalità del sistema

- **coinvolge gli utenti del sistema**
- **si applica a tutte le interfacce esterne del sistema**
- **mira a scoprire violazioni dei requisiti funzionali**
- **effetti collaterali:**
  - **può rivelare problemi di formulazione dei requisiti funzionali**
  - **può produrre statistiche utili sull'uso tipico o prevalente del sistema**

Strategie nel collaudo di sistema

stadio finale di collaudo:
software, hardware, operatività, ...

- **mira a scoprire violazioni di requisiti non-funzionali**
- **quattro aspetti tipici del collaudo di sistema:**
  - **recovery testing**
    comportamento durante, e recupero da, *calamità*
  - **security testing**
    mira a rivelare violazioni dei requisiti di *sicurezza* (funzionali!)
  - **stress testing**
    comportamento in condizioni di *sovraccarico*
    connessione con *recovery testing* e *performance testing*
  - **performance testing**
    mira a determinare le *prestazioni* in varie condizioni di carico

Metriche nel collaudo del software

- **metriche per il collaudo del software**
  nella sua *pianificazione* e *progettazione*
- **metriche dal collaudo del software**
  nella sua *esecuzione*

Metriche per il collaudo del software

prodotte dalla pianificazione e progettazione, servono a:

- **stimare lo sforzo di collaudo**
  - **metriche globali:**
    sforzo *aggregato*
  - **metriche locali:**
    sforzo *disaggregato* nel collaudo di *unità*
    ad es.: *complessità ciclomatica*
- **predire la complessità del collaudo di integrazione**
- **scegliere le strategie di collaudo più efficaci**

Metriche dal collaudo del software

misure sul numero e tipo di errori rilevati dal collaudo, utili per:

- **ulteriore stima dello sforzo di collaudo, o sua calibrazione**
- **valutazione di qualità del processo e/o del prodotto**
- **costituzione di archivi storici**

Collaudo di software ad oggetti

area di intensa ricerca

Metodi di collaudo di software ad oggetti

progetto di casi di collaudo (test case) basati su

- **le classi (diagrammi di stato)**
- **plausibili aree di errore (fault based)**

metodi di collaudo delle classi:

- **random testing**
- **partition testing, basato su:**
  - **stati**
  - **attributi**
  - **categorie (di operazioni)**

collaudo basato su scenari (interazioni di utente)

Strategie di collaudo di software ad oggetti

negli ambiti delle strategie tradizionali, sfumature di significato diverse:

- **collaudo di unità:**
  raramente al livello di singola operazione, l'unità tipica nel collaudo OO è la *classe*
- **collaudo di integrazione: di classi**
  Il raggruppamento di classi in un caso di collaudo può essere:
  - **thread-based:**
    il thread è costituito dalle classi che intervengono in reazione a un dato input
  - **uses-based:**
    gerarchico: il gruppo *n+1* è costituto dalle classi che usano quelle del gruppo *n*
  - **collaboration-based:**
    detto anche *cluster testing*

- **collaudo funzionale, di sistema:**
  analogo al collaudo tradizionale ma basato sulle specifiche dei *casi d'uso*

Metriche nel collaudo di software ad oggetti

in aggiunta a quelle tradizionali, metriche tipiche del collaudo OO, ad es.:

- **metriche di incapsulamento**
  - **LCOM:** *lack of cohesion in methods*
  - **PAP:** *percent public and protected*
  - **PAD:** *access to data members*
- **metriche di ereditarietà**
  - **NOR:** *number of root classes*
  - **FIN:** *fan-in* (se > 1: ereditarietà multipla)
  - **NOC:** *number of children*
  - **DIT:** *depth of inheritance tree*

Approfondimenti

- **alcuni siti per la ricerca di documentazione:**
  (ApTest), (R.S. Pressman & Associates)
- **ulteriori letture consigliate:**
  (Janzen et al. 2005), (Louridas 2005)

Alcune questioni per la discussione

- **maturità del processo di produzione**
  procedure formali di collaudo, quali ad es.:
  - **Software Inspection**
  - **Software Quality Assurance**

  richiedono gestione, con attenzione ai problemi interpersonali
- **il collaudo fatto dagli stessi sviluppatori tende ad essere soggettivo**
- **testing by poking around:**
  variante OO del *random testing,* ne eredita tutti i limiti
- **model-based testing:** (MBT 2005), (MBT 2006)
  derivazione *automatica* di casi di collaudo da modelli

Note

1. Si saggia il budino mangiandolo.

Bibliografia

- **Springl, M.**, 1998. *Software Testing*. Alberta, Canada, T2N 1N4: University of Calgary, SENG 621.
- **Janzen, D. & Saiedian, H.**, 2005. Test-Driven Development: Concepts, Taxonomy and Future Direction. *IEEE Computer*, 38:9, 43-51.
- **Louridas, P.**, 2005. JUnit: Unit Testing and Coding in Tandem. *IEEE Software*, 22:4, 12-15. Web: http://www.computer.org/portal/cms_docs_software/software/content/junit.pdf
- **MBT**, 2005. Proc. Workshop on Model Based Testing (MBT 2004). *Electronic Notes in Theoretical Computer Science*, 111, 1-182. Web: http://www.sciencedirect.com/science/journal/15710661
- **MBT**, 2006. Proc. Second Workshop on Model Based Testing (MBT 2006). *Electronic Notes in Theoretical Computer Science*, 164:4, 1-128. Web: http://www.sciencedirect.com/science/journal/15710661
- **Meyer, B.**, 1992. Applying Design by Contract. *IEEE Computer*, 25:10, 39-51. Web: http://se.ethz.ch/~meyer/publications/computer/contract.pdf
- **ApTest, Inc.**. *Applied Testing and Technology, Inc., Software QA Testing and Test Tool Resources*. Web: http://www.aptest.com/resources.html.
- **R.S. Pressman & Associates, Inc.**. *Software Engineering Resources*. Web: http://www.rspa.com/spi.
- **JUnit**. *JUnit.org*. Web: http://www.junit.org.