Università di Roma TorVergata

Laurea Magistrale in Informatica

Corso: Modelli e Qualità del Software

Federica Magliocca, 0318517,
federica.magliocca@students.uniroma2.eu

9 CFU

Nome della prova: Prova Finale MQS

Data della Prova: 26/06/2023

DOMANDE

Quesito 1

Dire quale è il significato dell'attributo di qualità "dependability" e quale quello del

suo attributo "safety". Descrivere poi e spiegare la metrica di quest'ultimo.

Quesito 2

Si consideri un sistema software SYS costituito da 5 moduli e il cui modello di failure

sia espresso dal Fault Tree in figura.

Assumendo che la R(t) di ogni modulo abbia l'espressione ottenuta nella Prit2,

ottenere l'espressione della RSYS(t) dell'intero sistema e con essa calcolare la

probabilità che non si verifichi una failure del SYS prima di 9 ore.

Quesito 3

Descrivere la regola 10-90 nelle caratteristiche di affidabilità del software, e il profilo operativo da usare nel testing ST, nonché l'effetto saturazione e il suo rimedio attraverso transizione da black-box a white-box testing e successivo mutation testing.

Quesito 4

Dire quale è lo scopo di un modello di affidabilità statica e descrivere il modello Sherry-Malviya-Tripathi di stima difetti per software Object Oriented.

Quesito 5

Dare la formula e dire il significato del coefficiente RE (mean relative error) in un modello di affidabilità statica.

Quesito 6

Definire le metriche object oriented: OOFI, CDC, NOC darne le formule, dire perché i loro valori devono essere bassi. e darne i valori di soglia.

Quesito 7

Descrivere la finalità del Loop RE all'interno del processo di Reliability Engineering, e descrivere, con grafo e commenti, oppure pseudocodice e commenti: il Loop RE per la fase di progetto dettagliato.

Quesito 8

Dire quale relazione corre tra le tecniche di hazard analysis per software critico e le tecniche di software fail-safe, e in quale ordine si applicano. Descrivere poi la tecnica PFM di software fail-safe.

Quesito 9

Descrivere le relazioni che corrono tra Safety Engineering e Reliability Engineering

RISPOSTE

1) La “dependability” è un sotto-attirbuto dell’indice di qualità performance e rappresenta la capacità di un sfotware di essere affidabile, sicuro e disponibile per l’utilizzo. Questo concetto comprende i seguenti attributi:

- Availability
- Reliability
- Safety

L'attributo di qualità "safety" si riferisce alla capacità del sistema software di prevenire danni o rischi per gli utenti, gli operatori o l'ambiente circostante. La "safety" è fondamentale in settori critici come l'aviazione, l'industria automobilistica, la medicina e molti altri, in cui il malfunzionamento del software può avere conseguenze gravi o addirittura letali.

La metrica dell’attributo safety è:

prob(noaccid) : la probabilità che non si verficihino incidenti gravi ad un dato istante di tempo.

2) Dal Fault Tree in figura si vede che il sistema subisce una failure soltanto se entrambi i sottosistemi (P1, M1, M3) e (P2, M2, M3), subiscono una failure. E Il sottosistema (P1, M1, M3) a sua volta subisce una failure soltanto se fallisce P1 e falliscono entrambi i moduli M1 ed M3. Similarmente dicasi per il sottosistema (P2, M2, M3).

Inoltre sapendo che l’espressione della R(t) è quella ottenuta nella Prit2, ovvero:

$R(t) = e^{-0.0008t}$

Sotto queste assunzioni si può scrivere, per la distribuzione del tempo di vita del sistema:

$FSYS= [1-(1- e^{-0.0008t})(1- e^{-(0.0008t)^2})][1-(1- e^{-0.0008t})(1- e^{-2(0.0008t)})]$

Calcoliamo La R_{SYS} come $1-F_{SYS}$

$R_{SYS} = 1 - ([1 - (1 - (1 - e^{-0.0008t})) (1 - (1 - e^{-0.0008t})^2)] [1 - (1 - (1 - e^{-0.0008t})) (1 - (1 - e^{-0.0008t})^2)])=$
$= 1 - ([1 - x (e^{-(0.0008t)^2} - 2e^{-0.0008t})] [1 - e^{-0.0008t} (e^{-(0.0008t)^2} - 2e^{-0.0008t})]) =$
$= 1 - (1 - e^{-(0.0008t)^2}) (e^{-0.0008t} - 2))^2$

Quindi per calcolare la probabilità che non si verifichi una failure del SYS prima di 9 ore calcoliamo:

$R(9) = 1 - (1 - ((e^{(-0.0008 * 9)^2}) * (e^{(-0.0008 * 9)} - 2))^2 =$
$= 1 - (1 - ((0.99919859707)^2)* (0.99919859707 - 2))^2 = 0.9854973135$

3) Intuitivamente si potrebbe pensare che un prodotto software con molti difetti sia poco affidabile e che l'affidabilità del prodotto migliori via via che si riduce il numero di difetti.

Nella realtà invece un prodotto con molti difetti potrebbe risultare più affidabile di uno con pochi, perchè eliminare difetti dalle parti del prodotto raramente usate ha piccoli effetti sull'affidabilità osservata.

Da qui la Regola 10-90:

– il 90% del tempo di esecuzione totale è speso eseguendo il solo 10% delle

istruzioni;

– detto 10% è chiamato core (nucleo) del programma.

• Il miglioramento dell'affidabilità per l'eliminazione di un difetto dipende se il difetto appartiene al nucleo del programma oppure no.

Il metodo ST si ispira direttamente al metodo DT, sia per quanto riguarda il processo che per quanto riguarda i tipi.

Il processo ST in particolare si svolge come illustrato a proposito del processo DT, e i tipi ST restano di tipo black-box, white-box e interface.

• La sostanziale differenza sta nella scelta dei test data, i quali rispettano ora il profilo operativo.

Il profilo operativo non è altro che una descrizione della probabilità con cui si manifestano vari tipi di dati di input.

Gli input di ciascun tipo saranno costruiti in maniera da essere statisticamente indipendenti e totalmente random.

Il processo ST procederà ripetutamente estraendo in modo totalmente random un dato di test e dandolo in input al prodotto software al fine di verificare se l'output corrisponde a quello atteso.

Nel caso in cui l'output non corrisponda a quello atteso si registra una failure e si procede alla correzione del relativo difetto.

Il processo viene iterato, di volta in volta registrando l'intervallo Ti tra due failure e per un numero di volte

– fino a quando consentito dal budget di testing

oppure

– fino a quando l'affidabilità stimata non abbia raggiunto il valore di requisito.

All'inizio del testing un programma contiene un certo numero N, di difetti.

Via via che il testing procede il numero di difetti rimanenti decresce.

Tuttavia, ogni modalità di testing ha un limite rispetto al numero di difetti che essa è in grado di rivelare.

Si inizia normalmente con il black-box testing e via via che questo procede l'affidabilità del programma cresce.

Tuttavia, quando il suo limite è stato raggiunto il fatto di non scoprire altri difetti o scoprirli molto raramente non deve far pensare che l'affidabilità cresca ancora.

Essa infatti rimane stabile semplicemente per l'incapacità del test di scoprire ulteriori difetti.

Prima di trarre conclusioni sull'affidabilità è opportuno passare alla modalità white-box testing la quale potrà a sua volta raggiungere un livello di saturazione, comunque superiore al precedente.

Un'analisi accurata potrebbe a questo punto scoprire ulteriori difetti usando il cosiddetto mutation testing. Il mutation testing genera un certo numero di cosiddetti mutanti di un programma P ovvero programmi sintatticamente corretti che si differenziano da P secondo regole predefinite.

Ad esempio, un mutante M di P può essere ottenuto rimuovendo un segmento di P. A questo punto si esercita M con gli stessi dati di test di P.

Se gli output rimangono gli stessi significa che il white box testing non andava ad esercitare il segmento di P rimosso da M. Sarà dunque necessario aggiungere opportuni dati di test al fine di incrementare la copertura del programma.

Ovviamente anche il mutation testing raggiungerà il suo livello di saturazione sempre comunque superiore al precedente.

A questo punto però la stima di affidabilità si può ritenere più attendibile.

4) I modelli statici di affidabilità sono modelli che, attraverso una analisi di regressione su dati da progetti esistenti, consentono di stabilire una relazione tra

– misure di complessità e

– numero iniziale N di difetti.

Detti modelli risultano abbastanza attendibili, come mostrano esperimenti che confrontano (in un ambiente di laboratorio)

– numero y di difetti effettivamente esistenti nel prodotto (artificiosamente introdotti) e

– numero $\hat{y}$ stimato (sopra chiamato N).

Una modalità per effettuare questo confronto è basata sul calcolo di:

• Errore relativo medio RE (Vedremo meglio nella risposta 5)

• Correlazione CR: capacità della stima $\hat{y}$ di inseguire il valore vero y (se y diminuisce anche $\hat{y}$ dovrebbe diminuire e analogamente dicasi se y aumenta o resta costante)

Alcuni modelli statici sono:

• **Akiyama**
• **Brooks**
• **Halstead**
• **Lipow**
• **Gaffney**
• **Compton-Withrow**
• **Rodriguez-Harrison-Satpathy-Dolado**
• **Sherry-Malviya-Tripathi**

In particolare il Modello Sherry-Malviya-Tripathi,che utilizza le metriche object oriented , calcola lo stimatore SMT del numero di difetti nel seguente modo:

ŷ=k [□□□□□ ∗ □□□∗ □□□∗ □□□∗ □□□□)/□□□]

dove
DEPTH:misura il livello di profondità della classe all'interno della gerarchia di ereditarietà.
WMC: misura la complessità dei metodi realizzati all'interno della classe e indica una classe di difficile comprensibilità.
RFC: misura il numero di metodi invocati in risposta ad un messaggio. Essa è un indice della complessità del codice e dell'uso del polimorfismo.
CBO:misura il numero di classi correlate a una data classe al di fuori della gerarchia di ereditarietà (livello di coupling, o numero di funzionalità fortemente connesse).
LCOM: misura la percentuale dei metodi che non accedono a specifici attributi definiti in altri metodi della stessa classe (livello di espletamento interno alla classe).
NOC: Descritta nella risposta 6.

5)Come abbiamo gia visto nella risposta 4, i modelli statici confrontano:

– numero y di difetti effettivamente esistenti nel prodotto (artificiosamente introdotti) e

– numero ŷ stimato (sopra chiamato N).

Una modalità per effettuare questo confronto è basata sul calcolo dell'Errore relativo medio RE che rappresenta lo scostamento della stima ŷ dal valore vero y.

Dato un prodotto software fatto di n moduli (m_1, m_2, …, m_n) l'errore relativo medio è definito da:

$$RE = (1/n) \sum_{i=1}^{n} RE_i$$

• dove RE_i:

$RE_i = (\hat{y}_i - y_i)/ \hat{y}_i$

RE può assumere valori tra -1 e +1: più è grande in valore assoluto maggiore è l'errore.

6) Nella metrica object-oriented la complessità del progetto software dettagliato è definita guardando la struttura delle classi e dei relativi oggetti e le caratteristiche dei paradigmi tipici, in particolare:

• Incapsulamento
• Ereditarietà
• Polimorfismo
• Coesione

Sulla cui base si definisce una serie di metriche di complessità (PD, CBO, LOCM, WMC, Depth, RFC, OOFI,CDC, NOC).
In particolare:

- La metrica **OOFI** viene utilizzata misurare la presenza di ereditarietà multipla, un elemento negativo di progetto. Il valore OOFI è calcolato con la formula:

  OOFI = $\sum_{i=1}^{N} C_i$

  dove:

  N = Numero di classi root (ossia classi dalle quali la classe in oggetto discende).

  C_i = 1 se si ereditano dati e attributi della classe i, = 0 diversamente.

  Il valore di soglia è OOFI ≤1.

- La metrica **CDC** misura la dipendenza di una classe da una classe figlia (dipendenza di dati ed attributi non ereditati) che indica presenza di ricorsività.

  Il valore CDC viene calcolato con la formula:

  CDC = $\sum_{i=1}^{N} C_i$

  dove:

  N = Numero di classi figlie

  C_i = 1 se la classe i dipende da una sua classe figlia, = 0 altrimenti.

  Il valore di soglia è CDC = 0.
- La metrica **NOC** indica il numero di immediate sottoclassi di una data classe. Un alto valore di NOC indica che si è fatto un ampio uso dell'ereditarietà.

  Non esiste un valore di soglia per NOC.

Le misure di complessità process oriented e quelle object oriented possono essere usate per la stima statica del numero iniziale N di difetti introdotti nel modulo in fase di codifica.

7)Il Loop RE (Loop di Reliability Engineering) è una parte fondamentale del processo di Reliability Engineering (Ingegneria dell'affidabilità). La sua finalità principale è quella di valutare e migliorare continuamente l'affidabilità di un sistema durante il suo ciclo di vita.

Nel ciclo RE si avanza identificando via via potenziali difetti di affidabilità nel ciclo di vita e intraprendendo per tempo eventuali azioni correttive.

Lo pseudo codice di seguito rappresenta una possibile implementazione del Loop RE per la fase di progettazione dettagliata di un sistema, con un commento linea per linea:

1. begin

2. ottieni la SC dalla fase di progetto preliminare: *Si ottiene la Specifica dei Requisiti dal lavoro svolto nella fase di progettazione preliminare.*

3. per ogni modulo j della SC: *Si itera su ciascun modulo della Specifica dei Requisiti.*

4. produci una versione-tentativo in pseudo-codice del modulo j: *Si sviluppa una versione preliminare del modulo in pseudo-codice per poter simulare la sua esecuzione.*

5. simula l'esecuzione di j: *Si esegue la simulazione dell'esecuzione del modulo.*

6. calcola numero difetti e tasso failure del modulo j: *Utilizzando Defect model e Statistical Testing, si calcolano il numero di difetti e il tasso di failure del modulo j.*

7. calcola la distribuzione Fj (t) del tempo di vita del modulo j: *Utilizzando Reliability Model, si calcola la distribuzione del tempo di vita del modulo j.*

8. se gli indici sono accettabili, va a 13: *Se gli indici di affidabilità del modulo sono accettabili, si passa alla prossima iterazione.*

9. se non è conveniente eseguire un nuovo tentativo va a 18: *Se non è vantaggioso eseguire un nuovo tentativo per migliorare il modulo, si torna alla fase di progettazione preliminare.*

10. identifica i difetti nel modulo j: *Si identificano i difetti presenti nel modulo j.*

11. elimina i difetti dal modulo j: *Si procede all'eliminazione dei difetti identificati nel modulo j.*

12. va a 4: *Si torna alla linea 4 per produrre una nuova una versione-tentativo*

13. se ci sono altri moduli da verificare va a 3: *Se ci sono altri moduli da verificare nella Specifica dei Requisiti, si torna alla linea 3 per iterare su di essi.*

14. produci progetto globale di tentativo integrando i moduli: *Si integra il progetto preliminare dei moduli in un unico progetto globale.*

15. calcola la distribuzione F (t) del tempo di vita del progetto globale: *Utilizzando Failure Models, si calcola la distribuzione del tempo di vita del progetto globale.*

16. se gli indici sono accettabili, va a 20: *Se gli indici di affidabilità del progetto globale sono accettabili, si accetta il progetto dettagliato ottenuto.*

17. se conveniente eseguire nuovo tentativo integrazione, va a 14: *Se è vantaggioso eseguire un nuovo tentativo di integrazione per migliorare il progetto globale, si integra di nuovo il progetto dei moduli in un progetto globale.*

18. torna alla fase di progettazione preliminare: *Dal momento che non è conveniente eseguire ulteriori tentativi di miglioramento, si ritorna alla fase di progettazione preliminare per ripensare l'approccio.*

19. va a 22: *Fine del codice senza accettare il progetto dettagliato ottenuto.*

20. accetta il progetto dettagliato ottenuto come artefatto di fase: *Si accetta il progetto dettagliato ottenuto come output finale della fase di progettazione dettagliata.*

21. trasferisci l'artefatto alla fase di codifica: *Si trasferisce il progetto dettagliato accettato alla fase successiva, che è la fase di codifica.*

22. end

Questo pseudo codice rappresenta un'implementazione semplificata del processo di progettazione dettagliata, che include la simulazione, il calcolo degli indici di affidabilità e il miglioramento iterativo dei moduli e del progetto globale.

8)L'hazard analysis, o analisi dei pericoli, è una tecnica utilizzata per identificare i potenziali pericoli e le situazioni di guasto che possono verificarsi in un sistema.
Una volta rilevati con l'hazard analysis i guasti potenziali, le tecniche di fail-safe sono utilizzate per evitare l'incidente avviando il sistema verso uno stato sicuro quando si verifica un guasto.
In particolare, il Program flow monitoring(PFM) è una tecnica fail-safe basata su black-box testing del software, dunque l'inserimento di dati di test (seed) e la verifica del risultato atteso(key).
La verifica può essere effettuata anche lungo il percorso del programma in punti detti PFM points, in corrispondeza dei quali si confronta il risultato intermedio raggiunto con quello atteso.
Ovviamente più alto è il numero dei PFM points introdotti e più ampia sarà la verifica, a discapito del tempo di esecuzione.
Si distringuono 3 diverse tecniche di PFM:

- Application independent: PFM points tra le chiamate di funzione
- Application dependent: PFM points codificati all'interno delle funzioni stesse
- Time dependent: verifica che date funzioni siano richiamate entro un determinato periodo di tempo.

9)La Reliability Engineering (Ingegneria dell'affidabilità) si concentra sulla valutazione, progettazione e gestione dell'affidabilità e della disponibilità dei sistemi.
La Safety Engineering (Ingegneria della Sicurezza) si concentra sulla prevenzione, identificazione e mitigazione dei pericoli e dei rischi per la sicurezza delle persone, delle strutture e dell'ambiente.
La RE utilizza una varietà di tecniche per ridurre al minimo i guasti come:

- ridondanza parallela
- riserva di standby
- margini di sicurezza
- rilassamento
- sostituzione anticipata

Queste tecniche aumentano l'affidabilità ma non è detto che aumentino la safety. In alcune circostanze possono anche ridurla.
Sussiste infatti una relazione tra affidabilità e safety che mostra come queste ultime abbiano una zona di sovrapposizione, ma non totale. Questo perchè:

- Molti incidenti non sono causati da guasti
- Potrebbe esserci un guasto senza che ne consegua un incidente

La safety è dunque al di fuori del campo dell' analisi della reliability, la quale considera solo la possibilità di incidenti legati a guasti dei componenti.

Concluso il 26/06/2023 alle ore 14:20