Ingegneria del Software B

Allievi della Laurea Specialistica in Ingegneria Informatica
Tema d'esame - 14 Aprile 2005 – ore 9.00-10.30

NOME: …… COGNOME: …….
MATRICOLA: …….. FIRMA: ……..

Le risposte chiuse valgono 1/30 ciascuna. Il valore degli esercizi è riportato nel prospetto a lato.

Esercizio	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Valore	2	3	2	3	4	1,5	2,5
Valutazione							

Risposte chiuse

Affermazione	Vera o falsa?
Nell'esecuzione di un programma orientato agli oggetti che contiene gerarchie di classi si possono verificare down-call	
Nella programmazione a oggetti, l'ereditarietà è una forma di riuso white box	
L'ereditarietà senza polimorfismo è corretta secondo il principio di sostituzione di Liskov	
Il design pattern Composite è classificato come Object Creational (cioè creazionale e basato su oggetti)	
Il design pattern Observer è classificato come Object Behavioral (cioè comportamentale e basato su oggetti)	
Il testing basato sugli stati si adotta per la verifica di singoli metodi di programmi orientati agli oggetti	
La squadra di manutenzione di un sistema è composta interamente da persone che hanno partecipato allo sviluppo del sistema stesso	
L'attività di refactoring si applica esclusivamente a sistemi orientati agli oggetti	
Per configurazione di un sistema software si intende una famiglia di prodotti che condividono tutti il medesimo deposito di progettazione (*repository*)	
I grafi di tracciabilità vengono sfruttati per effettuare l'analisi di impatto dei cambiamenti	
Ai cinque livelli di maturità di processo stabiliti da CMM (Capability Maturity Model) corrispondono altrettanti livelli di qualità di prodotto	
COCOMO II è un modello di previsione dello sforzo di un progetto, dove tale sforzo è espresso in mesi-persona	

Esercizi

1) Descrivere come si conduce e a cosa serve l'analisi mutazionale.

2) Classificare il design pattern Decorator e descriverne sia l'intento, sia la struttura.

3) Puntualizzare le maggiori differenze fra ispezioni e testing.

4) Definire alcuni criteri di testing dei cammini, precisando il tipo di copertura degli stessi e se vengano adottati per il testing in piccolo e/o in grande.

5) Sfruttare le tecniche di “equivalence partitioning” e “boundary value analysis” per creare un insieme di test funzionali di unità per un modulo la cui specifica è la seguente:

acquisito in ingresso l’orario settimanale di due insegnamenti, determinare eventuali sovrapposizioni.

Si assuma che valgano le seguenti precondizioni: l’orario settimanale di ogni insegnamento (per un ammontare complessivo che varia, da insegnamento a insegnamento, fra un minimo di 6 ore e un massimo di 9 ore) è espresso da 3 blocchi, ciascuno dei quali

- è della lunghezza di 2 o 3 ore,
- è contenuto entro l’arco temporale giornaliero che va dalle ore 8.00 alle ore 13.00,
- è collocato in un giorno lavorativo distinto rispetto agli altri 2 blocchi relativi allo stesso insegnamento.

6) Tracciare il CFG (Control Flow Graph) del seguente programma.

```
main(){
1     scanf(“%d”, &b);
2     scanf(“%d”, &a);
3     if (a == 5) {
4          a = a + b;
5          if (b == 4)
6               a--;
      }
7     a--;
8     if (b)
9          while (a)
10              a++;
11    else a = 3.14;
12    printf(“%d\n”, b);
13    printf(“%d\n”, a);
      }
```

7) Stabilire, mediante esecuzione simbolica, se il cammino <1,2,3,7,8,9,12,13> del programma di cui al punto precedente è percorribile (*feasible*) o meno. Nel caso sia percorribile, determinare il relativo predicato del cammino.