Megoldások a mintavizsga kérdések a VIMIAC04 tárgy ellenőrzési technikák részéhez kapcsolódóan (2017. május)

Teszt kérdések

1. Melyik állítás igaz a folytonos integrációval (CI) kapcsolatban?
    a. Folytonos integráció esetén megfelelő a termék új változatának kézi ellenőrzése.
    b. Folytonos integráció bevezetése esetén a legfontosabb feladat a megfelelő CI eszköz kiválasztása.
    c. A folytonos integráció egy olyan jó gyakorlat, ami a csapattagok munkájának gyakori integrálását javasolja.
    d. Folytonos integrációs kiszolgáló használata esetén a fejlesztőknek nem szükséges a teszteket lokálisan futtatni.
2. Melyik állítás nem igaz a viselkedésalapú szoftverfejlesztéssel (BDD) kapcsolatban?
    a. BDD használata esetén a követelményeket leíró példák végrehajthatók és ellenőrizhetők.
    b. A BDD az aktív együttműködésre és a követelmények automatizált ellenőrzésére helyezi a hangsúlyt.
    c. A BDD a specifikációt példák segítségével teszi egyértelműbbé az érdekeltek számára.
    d. A BDD alkalmazásához a Cucumber eszközt kell használni.
3. Melyik állítás hamis? Statikus ellenőrzés képes felderíteni
    a. nem hatékony kódrészleteket,
    b. tömb túlindexelést,
    c. tesztek által le nem fedett utasításokat,
    d. fel nem használt változókat.
4. Melyik állítás hamis? Unit teszteket azért hasznos készíteni, mert…
    a. megvalósítják a fejlesztett rendszer validációját.
    b. segítségükkel a hibák korán megtalálhatóak.
    c. példákat és egyfajta szerződést definiálnak a unit használatával kapcsolatban.
    d. az általuk megtalált hibák javításának költsége általában alacsony.
5. Melyik állítás igaz az alábbi tesztelési fogalmakkal kapcsolatban?
    a. A teszt-orákulum egy olyan program, ami eldönti a tesztről, hogy sikeres.
    b. Egy teszt eredménye sikeres (pass) vagy sikertelen (fail) lehet.
    c. A teszteset bemeneti értékek, végrehajtási feltételek és elvárt eredmények halmaza, amelyeket egy konkrét célért fejlesztettek.
    d. A SUT a Specification Under Test rövidítése.
6. Melyik állítás igaz az alkalmazások telepítésével kapcsolatban?
    a. A Continous Delivery célja, hogy az alkalmazás folyamatosan olyan állapotban legyen, hogy telepíthető legyen az éles környezetbe.
    b. Continous Delivery esetén mindig futtatni kell a teljesítménytesztet a telepítés előtt.
    c. Continous Delivery esetén egy-egy új szoftververzió automatikusan kikerül az éles környezetbe.
    d. A Continous Delivery megoldások valamilyen felhőszolgáltást használnak a telepítéshez.

Feladatok

1. Specifikáció-alapú tesztelés

Adott a következő tesztelendő függvény:

```
int functionA(int a, int b)
```

ahol az *a* egy pozitív egész szám, *b*-nek pedig 1 és 10 közé kell esnie.

a) Mik az egyes paraméterek határértékei, és a határérték analízis technika alapján milyen teszt-bemeneteket választanánk ki ezekhez? [3 pont]
b) Táblázatban adja meg, hogy a kiválasztott értékek alapján milyen tesztkészletet állítanánk elő a functionA függvényhez? [3 pont]

MO:

a) Határértékek és javasolt bemenetek:
   - a: hatérték a 0
     - Megjegyzés: definíció kérdése, hogy mit tartunk még pozitív számnak, és így hol lesz az érvényes tartomány határa (0 vagy 1), de mivel ez egy tudottan problémás kérdés, érdemes akár több tesztbemenetet is kiválasztani.
     - Javasolt értékek (pont a határértéken kívül, határérték, határértéken belül)
       - -1, 0, 1
       - Ezekből a -1 és a 0 érvénytelen értékek, az 1 érvényes.
     - (Megjegyzés: lehet úgy gondolkodni, hogy a véges felbontás miatt van felső határa is az *a* paraméternek, így a MAXINT is lehet egy további hatérték.)
   - b: egy intervallum van megadva, így két hatérértéke van: 1, 10.
     - Javasolt értékek:
       - 0, 1, (tartományon belüli érték, pl. 5), 10, 11

b) tesztesetek előállításánál arra érdemes figyelni, hogy egyszerre ne legyen érvénytelen mindkét paraméter értéke, mert akkor elfedhetik az esetleges hibákat. Ezen kívül viszont érdemes minden értéket legalább egyszer lefedni.

ID	a	b	Elvárt kimenet
1	1	1	?
2	1	5	?
3	1	10	?
4	-1	5	HIBA
5	0	5	HIBA
6	1	0	HIBA
7	1	11	HIBA

Az elvárt működés definícióját nem tudjuk, így egyelőre csak azt tudjuk megmondani, hogy mikor várnánk valami hibajelzést.

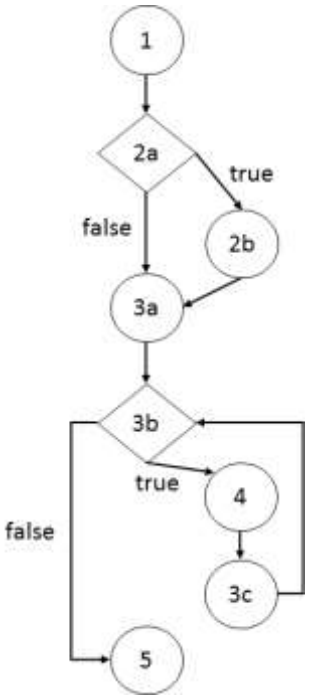
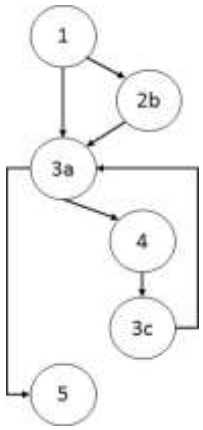
2. Struktúra-alapú tesztelés

Adott a következő forráskód részlet.

```
  int function13 ( int a , bool b){
1    int c = 10;
2    if (b) c = -c;
3    for ( int i = 0; i < a; i++){
4      c++;
     }
5    return c;
  }
```

a) Rajzoljuk fel a függvény vezérlési folyam gráfját (CFG)! [3 pont]
b) Adjunk meg egy tesztesetet, mely 100%-os utasítás lefedettséget garantál! [3 pont]

MO: a)

Egyszerűbb úgy rajzolni, ha a döntési utasításokat (`if`, `switch`, ciklusfeltétel) valami más szimbólummal jelöljük.  ahol a számok a kódban megszámozott utasításokat jelzik, az a/b/c pedig, ha valahol több van egy sorban (pl. a 3c a `for`-ban lévő i++)	A baloldalon szereplő gráfot lehet egyszerűsíteni, ha összevonunk csomópontokat (és pl. az `if` nem jelenik meg külön csomópontként):  (ugyanazokat az utakat kapjuk majd, csak kompaktabb a gráf)

b) Annyit kell csak garantálni, hogy minden utasítást legalább egyszer végrehajtjunk (tehát belépjünk az `if`-be, valamint a ciklusba is legalább egyszer). Ez a következő tesztesettel el is érhető:

a: 1

b: true

3. Forráskód átvizsgálása

Adott a következő forráskód részlet.

```
float func(float a, int t){
  if (a == 0) return 0;
  switch (t){
    case 0: {
      a *= 0.8;
      break;
    }
    case 1: a *= 0.7;
    case 2: a *= 0.5;
  }
  return a;
}
```

a) Soroljon fel legalább két problémát a kód stílusával kapcsolatban! [4 pont]
b) Soroljon fel legalább két olyan problémát, amely potenciális hibalehetőséget rejt magában! [4 pont]

MO:

a)

- több kódolási ajánlás szerint akkor is ki kell rakni a {} zárójeleket az `if` blokkban, ha csak egy utasításból áll (ez ajánlástól függ, hogy probléma-e)
- semmitmondó azonosítók (változók, függvény neve)

b)

- hiányzik a `break` az 1 és 2 esetnél
- nincs `default` ág a `switch`-ben
- az `if`-ben `float` típusú változónál == operátorral vizsgáljuk az egyenlőséget, ami pontatlan lehet (lásd pl. http://stackoverflow.com/questions/1088216/whats-wrong-with-using-to-compare-floats-in-java)

4. Specifikáció ellenőrzése

A repülőjegyhez választható extrák árának kiszámításához készülő modulhoz a következő specifikációt kaptuk.

- „Egy 15 kg-os csomag ára 10 EUR, a 20 kg-os ára pedig 20 EUR. Egy utas legfeljebb két csomagot hozhat. A reptéren vásárolt csomag felára 15 EUR."
- „Ülések foglalásának díja 8 EUR. Prioritásos ülések (1–3 sor) díja 13 EUR."
- „Lehetőség van Leisure csomagot választani (15 EUR), amely tartalmaz egy 15 kg-os csomagot és egy foglalt ülést."

Ellenőrizzük a kapott specifikációt. Milyen kérdéseink és észrevételeink lennének? [5 pont]

MO: Lehetséges kérdések és észrevételek (legalább ötöt érdemes megadni):

- A reptéren vásárolt csomag felára csomagonként értendő?
- Ha az utas hoz két csomagot magával, akkor is vásárolhat továbbiakat a reptéren?
- Lehet-e 20 kg-osnál nehezebb csomagot felvinni?
- Mennyit kell fizetni, ha valaki Leisure csomagot vett, de szeretne az első három sorba ülni? Ki kell fizetnie akkor a teljes 15+13 EUR díjat?
- Hasonlóan, mennyit kell fizetni, ha Leisure csomag esetén 20kg-os csomagot hozna?
- Lehet-e Leisure csomag a reptéren venni?
- …