Naam: . . . . . .

Theorie

Beantwoord onderstaande vragen (*elke vraag staat op 2 punten*) door
- de antwoordzinnen KORT aan te vullen
- de fout(e) alternatief(ven) te schrappen (evt. met motivatie)

1. Wat is het voornaamste verschil tussen analyse en ontwerp (“design”)?

   Analyse concentreert op het *wat* . . . . . .

   terwijl ontwerp concentreert op het *hoe*.

2. Wat is de gelijkenis tussen een alfa en een beta-test? Wat is het voornaamste verschil?

   Alfa en beta-tests zijn gelijkaardig omdat ze beiden . . . . . .

   “acceptance tests” zijn / “end-user tests” zijn. . . . .

   Desondanks zal een alfa-test plaatsgrijpen in een gecontroleerde omgeving / onder toezicht

   terwijl een beta-test plaatsgrijpt in “real life” setting / zonder toezicht

   . . . . . .

3. Wat is de zwakst mogelijke pre- of post-conditie? Waarom?

   De zwakst mogelijke conditie is {true} . . . . . .

   omdat {X} => {true} voor alle X . . . . . .

4. Wat is de gelijkenis en het verschil tussen “responsibility driven design” (RDD) en “use-cases” (UC)?

   RDD en UC zijn gelijkaardig omdat ze beiden . . . . . .

   technieken zijn die toegepast worden tijdens behoeftenanalyse

   Desondanks zal RDD . . . . . .

   behoeftes valideren/verifieren. . . . . .

   terwijl UC . . . . . .

   behoeftes specifieert . . . . . .

Naam: . . . . . .

5. Gegeven het volgende UML klasse diagram. Welke van de 2 volgende uitspraken is correct?

   - ❑ ~~1 object van klasse A heeft minstens 2 associaties met een object van klasse B.~~
   - ❑ 1 object van klasse B heeft minstens 2 associaties met een object van klasse A.

6. Wat is het voornaamste verschil tussen “refactoring” en puur code schrijven?

   “Refactoring” is code transformatie met behoud van extern gedrag

   terwijl code schrijven ook het extern gedrag wijzigt. . . . . .

7. Geef een definitie voor “coupling” en “cohesion”? Hoe kunnen we ze gebruiken als criterium voor een goed ontwerp?

   “Coupling” = een maat voor hoe sterk een component gekoppeld is met een andere component via een connector

   . . . . . .

   “Cohesion” = de mate waarin de onderdelen van een component samenhangen

   . . . . . .

   Een goed ontwerp (“design”) zal “coupling“ minimaliseren . . .
   en “cohesion” maximaliseren . . . . . .

8. Wat is het verband tussen formele specificaties en “Cleanroom development”?

   “Cleanroom development” toont aan dat formele methodes ook in de praktijk bruikbaar zijn

   “Cleanroom” is voornamelijk gebaseerd op formele specificaties (aangevuld met testen).

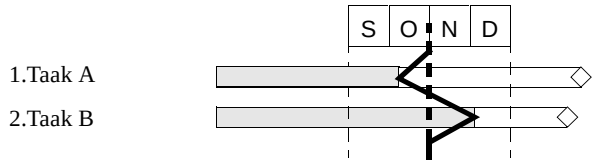
9. Is een correct functionerend stuk software noodzakelijkerwijs van goeie kwaliteit?

   Ja / ~~Neen~~ (schrap het foute alternatief)

   Waarom? Er zijn ook andere kwaliteitsfactoren: onderhoudbaarheid, efficientie, ...

   . . . . . .

10. Gegeven het volgende Gantt diagram met "slip line". Welk van de volgende uitspraken is correct? Motiveer je antwoord.



   - ❑ ~~Taak A is voor op schema; taak B is achter op het schema.~~
   - ❑ Taak B is voor op schema; taak A is achter op het schema.

   Waarom? Wit is de geplande hoeveelheid werk; grijs is hoeveel echt uitgevoerd werd.

   Taak B heeft al meer gedaan dan gepland; taak A minder.

11. Geef twee voor- en twee nadelen van functiepunten ("function points").

    Voordeel 1: implementatie onafhankelijk . . . . . .

    Voordeel 2: kan je meten na design . . . . . .

    Nadeel 1: vereist subjectief oordeel . . . . . .

    Nadeel 2: afhankelijk van specificatiemethode . . . . . .

Naam: . . . . . .

Oefeningen

2 oefeningenvragen, elke vraag staat op 4 punten.

12. Gegeven de volgende Z-specificatie.

Point

x: Integer
y: Integer

Rectangle

topLeft: Point
bottomRight: Point

(topLeft.x < bottomRight.x) and
(topLeft.y < bottomRight.y)

Point_Translate

Δ p?: Point
distance?: Point

p.x' = p.x + distance.x

p.y' = p.y + distance.y

(a) Geef een Z-specificatie voor een operatie "Rectangle_Translate" die, gegeven een rechthoek en een punt, de "topLeft" en "bottomRight" van de rechthoek zal verschuiven a.h.v. Point_Translate. Dit laatste mag alleen als de coordinaten van het verschuifpunt positief zijn!

Rectangle_Translate

Δ r?: Rectangle
distance?: Point

(distance.x > 0) and (distance.y > 0)

Point_Translate(r.topLeft, distance)

Point_Translate(r.bottomRight, distance)

(b) Schrijf C++ code voor een operatie "translate" die overeenkomt met je specificatie van (a). Gebruik daarbij "design by contract" stijl, dus inclusief pre- & post-condities en klasse-invarianten.

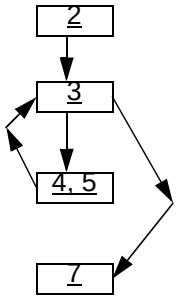
```
class Rectangle {
 private:
   Point bottomRight, topLeft;
 public:

      ...

   void translate (Point distance);
   {
   Point oldBottomRight, oldTopLeft;

   assert((topLeft.x() < bottomRight.x())
      && (topLeft.y() < bottomRight.y())); // invariant
   assert((distance.x() > 0) && (distance.y() > 0)); //pre
   assert(oldBottomRight = bottomRight || 1); //prepare post
   assert(oldTopLeft = topLeft || 1); //prepare post

   topLeft.translate(distance);
   bottomRight.translate(distance);

   assert((topLeft.x() < bottomRight.x())
      && (topLeft.y() < bottomRight.y())); // invariant
   assert((topLeft.x() == oldTopLeft.x() + distance.x())
      && (topLeft.y() == oldTopLeft.y() + distance.y())
      && (bottomRight.x() == oldBottomRight.x()
            + distance.x())
      && (bottomRight.y() == oldBottomRight.y()
            + distance.y())
      ); // post
   };

   ...

}; //End Class Rectangle
```

Naam: . . . . . .

13. Gegeven het volgende stukje code.

```
1. public void helloWorld (int x, y) throws assertionViolation
{
2.    assert(x <> y);
3.    while (x > y) {
4.       System.out.print(“Hello World”);
5.       x = x - 1;
6.    };
7.    assert (x == y);
8. }
```

(a) Teken de “flow chart” (plaats de lijnnummers in de knopen)



(b) Wat is het maximaal aantal onafhankelijke paden (formule + aantal).

pijlen - # knopen + 2 = 4 - 4 + 2 = 2. . . . . .

(c) Geef alle onafhankelijke paden doorheen de “flow-chart”.

{(2), (3), (7)} . . . . . .

{(2), (3), (4,5), (3), (7)} . . . . . .

. . . . . .

. . . . . .

Naam: . . . . . .

(d) Geef in- en uitvoerwaardes voor x en y opdat die paden uitgevoerd zouden worden.

{(2), (3), (7)} >> onmogelijk. . . . . .

{(2), (3), (4,5), (3), (7)} >> in { x = 1; y = 0 } uit: {1 x “Hello World”}

. . . . . .

. . . . . .

(e) Geef een extra in- en uitvoerwaarde voor x en y om de pre-conditie te testen. Doe dit nog eens voor de post-conditie.

pre-conditie >> in: {x = 0, y = 0} uit: {assert exception} . . .

post-conditie >> in: {x = 1, y = 2} uit: {assert exception} . .

. . . . . .

. . . . . .