Helsingin yliopisto, Tietojenkäsittelytieteen laitos
Ohjelmistotuotanto, kurssikoe 15.11.2002, H. Laine
Arvostelu

Arvostelu kannattaa tehdä siten että maksimipistemäärä on 6 pistettä.
On sitä helpompaa arvostella kun on vähemmän pisteitä on käytettävissä
Pisteet kerrotaan sitten lopuksi kahdella.
Plussia, miinuksia ja puolikkaita voi käyttää jos se tuntuu välttämättömältä.
Pisteytys on seuraavissa ohjeissa 6 pisteen mukainen.

1. Vertaile demokraattisen tiimin ja johtajavetoisen projektiryhmän soveltuvuutta eri tyyppisiin ohjelmistoprojekteihin. Kiinnitä vertailussasi huomiota esimerkiksi tehtävän vaikeuteen, tuotteen kokoon, hankkeen kestoon, modulaarisuuteen, vaadittavaan laatuun ja aikataulurajoitteisiin (12p)

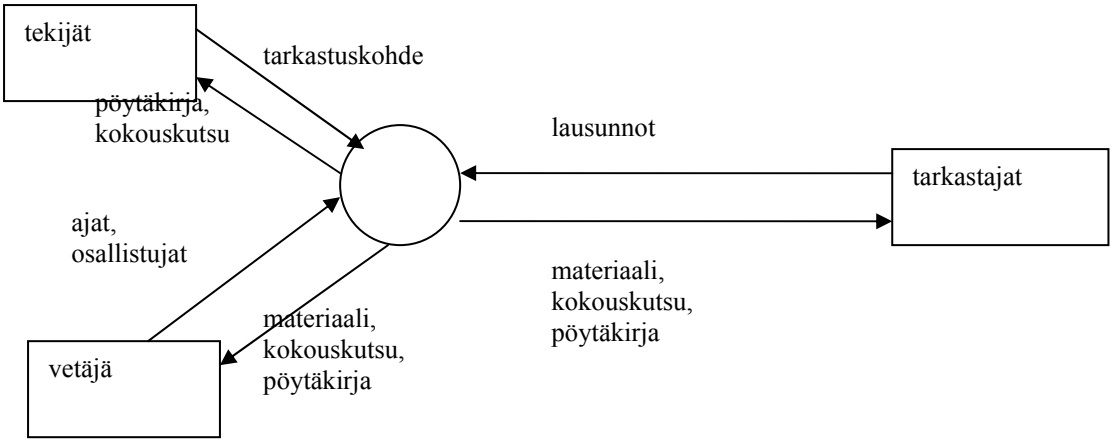
   Tässä on annettu kuusi kohtaa. Lyhyt yhden lauseen perustelu riittää kuhunkin kuhunkin kohtaan.

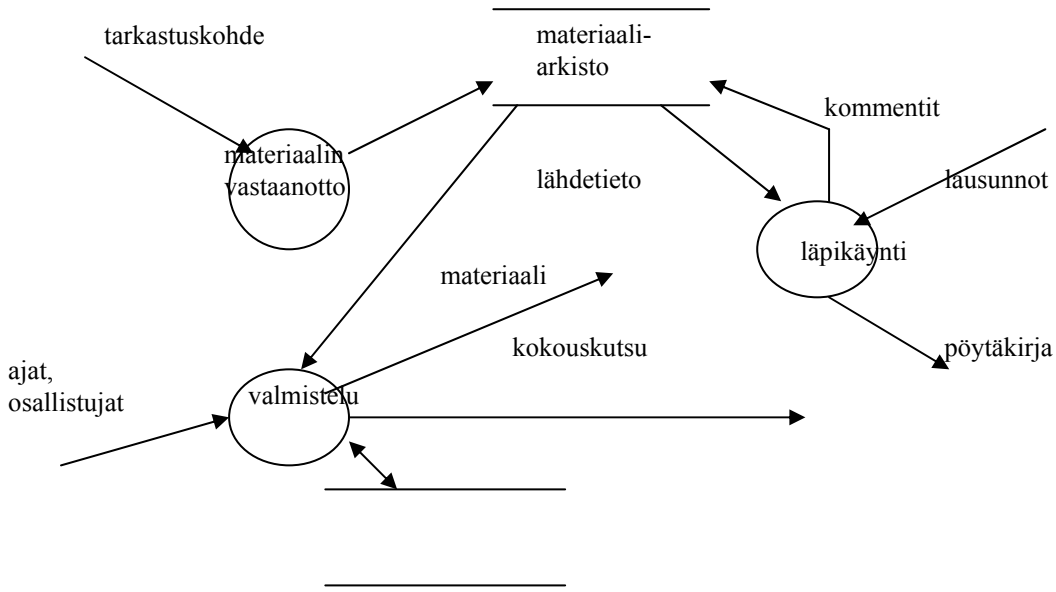
   a) Vaikea tehtävä edellyttää ratkaisuvaihtoehtojen kartoitusta ja erilaisten näkökohtien huomioimista ja niistä keskustelua, tiimi sopii tähän paremmin kuin johtajavetoinen
   b) Ison tuotteen kohdalla täytyy tehdä paljon päätöksiä ja silti pysyä aikataulussa. Tiimissä aikaa kuluu keskusteluihin, päätöksissä pitäisi saavuttaa yhteisymmärrys, joten johtajavetoinen soveltuu paremmin.
   c) Pitkäkestoisissa hankkeissa tiimi on yhteistyön kannalta parempi toisaalta se pidentää kestoa.
   d) Jos työ on jaettavissa helposti moduuleiksi se soveltuu hyvin johtajavetoiseen projektiin, ei tästä tiimityöskentelyssäkään ole haittaa.
   e) Tiimityöskentelyssä ratkaisuja käydään läpi perusteellisemmin mikä parantaa laatua. Johtajavetoisessa laatu on päällikön vastuulla.
   f) Tiimityöskentely vie enemmän aikaa. Tiukan aikataulun tapauksessa johtajavetoinen sopii paremmin.

   Jos perustelu on oikein, niin piste per kohta. Jos on todettu oikein, että a on parempi kuin b, mutta ei ole mitään perustelua, niin vähennetään ½.
   Oikea tulos, mutta väärä perustelu ½.
   Jos on hyvin perusteltu jokin muu kuin yllä esitetty, niin siitä voi antaa ½ tai jopa 1.

2. Tietovirtakaavio (data flow diagram) on perinteinen ohjelmiston toiminnallisuuden määrittelytekniikka. Esittele tekniikka lyhyesti. Laadi esimerkkinä karkean tason tietovirtakaavio kuvaamaan ohjelmiston katselmusprosessia (formal review) (yksi tarkennos riittää). (12p)

Tässä pitäisi esitellä ulkoinen olio, prosessi, tietovuo ja tietovarasto sekä hierarkkisen tarkennuksen periaate. Idean esittely 4 pistettä (eli puolikas per käsite) ja esimerkki 2 pistettä. Alla karkea DFD.





3. Luokan *QueueArray* tehtävänä on toteuttaa jono taulukkoa hyväksikäyttäen. Metodi *toQueue* lisää solmun jonon loppuun ja metodi *fromQueue* poistaa solmun jonon alusta. Jono muodostuu peräkkäisistä taulukon alkioista. Jonosta tiedetään alku- ja loppukohta. Lisäys vaikuttaa loppukohtaan ja poisto alkukohtaan. Taulukkoa käytetään renkaana, joten taulukon viimeistä alkiota seuraavana jonoalkiona voi olla taulukon ensimmäinen alkio. Kääntöpuolella on Tietorakenteet –kurssilta mukaeltu toteutus luokalle. Määrittele millaisia testiaineistoja tarvitaan metodien *toQueue* ja *fromQueue* toiminnalliseen testaukseen (black box - testaus, varsinaista dataa ei tarvitse antaa). Metodeja testattaessa luokan tietorakenne oletetaan tunnetuksi. Perustele aineistosi. Miten hoitaisit käytännössä luokan testauksen? Tarvitaanko jotain apumetodeja, millaisia? (12p)

Jos tämä on ratkaistu white box –ratkaisuna oikein annetaan max 3 pistettä.

Ratkaisuun pitäisi sisältyä tekijäanalyysi:

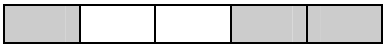
gSize: 1, jokin (=5), maxint-1
jonon koko: tyhjä, 1, usea(=3), täysi
jonon sijainti:

a) first=0, last>0

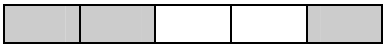
b) 0<first<qSize-1,first<last<qSize

c) 0<first<qSize-1,first<last=qSize

d) last<first<qSize-1



e) last< first = qSize-1



Taulukkokoolla 1: lisäys ja haku

jonon koko:	tyhjä	täysi=1
alkukohta:	0	0

Taulukkokoolla 5:

Tehdään lisäys ja haku

jonon koko:	tyhjä	1	3	täysi
alkukohta:	0	0	0	0
alkukohta:	1	1	1	2
alkukohta:	4	4	2	4
			3	
			4	

Taulukkokoolla maxint-1 vain ääripäät voivat poiketa, joten testataan lisäys ja poisto:

jonon koko:	tyhjä	1	3	täysi
alkukohta:	0	0	0	0

alkukohta:	max-1	max-1	max-3	2
alkukohta:			max-1	max-1

Arvot:

NULL lisäys tarvitsee testata vain kertaalleen.
Muuten testiaineistona on kätevintä käyttää järjestysnumeroista generoituja etunollallisia merkkijonoja.

Jonot voidaan ajaa testitilanteisiin lisäys ja haku sekvensseillä, joten siihen ei tarvita apurutiineja. Jonon alkiot ja alku ja loppukohdan tulostava metodi olisi hyödyllinen, ei kannata käyttää isoon taulukkoon. Testaus main-metodiin.

Oleellisten tapauksien puuttumista voisi suhteuttaa yllä oleviin taulukoihin

Miinusta:

- Rengasta ei ole testattu,
- Testataan vain yhdellä taulukkokoolla.
- Jonokoot suppeita
- Satunnaisesti generoitujen olioiden käyttö
- ei black box
- epämääräiset tai puuttuvat perustelut
- kuvattu yleisesti periaatetta mutta ei sovellettu

4.Mitä tarkoitetaan suunnittelumallilla (design pattern)? Kuvaa jokin (vain yksi) suunnittelumalleista Kooste (Composite), Komento (Command) tai Vierailija (Visitor) ja selvitä sen käyttötarkoitus ja edut sekä ongelmat. (12p)

Käsitteen kuvaus oikein 4 pistettä:
Kokemusperäinen hyväksihavaittu systemaattisesti dokumentoitu tapa ratkaista jokin suunnitteluongelma. Tässä kysyttiin design patternia, ei ole siis syytä selitellä muita.

Esimerkistä pitäisi selkeästi käydä esiin idea sekä edut ja mahdolliset ongelmat.
Miinusta:

- Kuvaus vaikeaselkoinen tai epämääräinen tai väärin,
- Kuvauksessa ei ole luokkakaaviota (kooste, vierailija) , komennolla ei ehkä tarvita.
- Perusidea jää epäselväksi
- Ongelmien puute korkeintaan
- Etujejen puute

```
public class QueueArray {

  private int qSize=0;              // maximum length of the queue
  private Object[] items = null;    // array
  private int first, last;          // index of head and tail
  private int cnt;                  // number of items

  public QueueArray(int maxsize) {
    // constructor makes an empty queue of at most maxsize items
    // if maxsize in negative or zero a queue of one items is established.
    if (maxsize>0) {
       items = new Object [maxsize];
       qSize= maxsize;
    } else {
       items = new Object [1];
       qSize= 1;
    }
    first=0;
    last= -1;
    cnt=0;
  }
  public boolean isEmpty() { return cnt==0; }
     // returns true if queue is empty otherwise false
  public boolean isFull() { return cnt==qSize; }
     // returns true if queue is full otherwise false
  public int itemCount() { return cnt; }
     // returns the number of items in the queue
  public boolean toQueue(Object newItem) {
     // appends a new item. Returns true if append succeeds and
     // false if it fails.
     // Fail situations: newItem is null or queue is full.
    if (newItem == null)
       return false;
    if (isFull())
        return false;
    else {
      ++last;
      if (last == qSize)
        last = 0;
      items[last] = newItem;
    }
    ++cnt;
    return true;
  }
  public Object fromQueue() {
     // returns the first item of the queue or null if the
     // queue is empty. Deletes the first item from the queue.
    if (isEmpty())
      return null;
    Object firstItem = items[first];
    items[first] = null;
    ++first;
    if (first == qSize)
      first = 0;
     --cnt;
    return firstItem;
  }

}
```