**Eksamiküsimuste vastused aines Tarkvaratehnika (MTAT.03.094)**
Aeg: 08. jaanuar 2008, 09:00 – 13:00

**Küsimused**

1. (4 p.)
   *Küsimus:*
   Mis on tüüpilise IDE põhifunktsionaalsuseks? Millal kasutada IDE asemel teisi vahendeid?

   *Lahendus:*
   IDE on keskkond mis võimaldab tarkvara arendajal efektiivsemalt töötada. Tüüpilises IDEs on integreeritud võimas tekstiredaktor, kompilaator (või interpretaator), silur, tugi ehitusskriptidele ning versioonihaldusele. Enamasti on IDE orienteeritud mingi kindla keele tarbeks ning kasutatav tekstiredaktor toetab just sellele keelele omaseid konstruktsioone (süntaksi esiletõstmine, code completion, error checking, ...).

   IDE asemel on teisi vahendeid (shell scripting, tekstiredaktor ...) otstarbekas kasutada peamiselt juhtudel kui soovitavat funktsionaalsust IDEs ei eksisteeri või kui soovitav funktsionaalsus on tihti kasutatava iseloomuga ning seetõttu otstarbekas automatiseerida.

   *Hindamine:*
   Kokku on küsimusele vastamise eest võimalik saada 4 punkti, mis jaguneb järgmiselt:
   - 1 pt IDE eesmärgi kirjeldus.
   - 0,4 pt iga nimetatud tüüpfunktsionaalsuse elemendi kohta. Kui on nimetatud muid funktsionaalseid lisandeid siis juhul kui tegemist on mõistlike lisanditega (drag'n drop GUI editor, modelling tools, ...) siis arvestada ka neid.
   - 1 pt IDE alternatiivide kirjeldus.

2. (4 p.)
   *Küsimus:*
   Nimetage andmetöötlussüsteemi neli olulisemat osa. Andke nende kirjeldus.

   *Lahendandus & hindamine:*
   - (1 p.) andmehoie (storage),
   - (1 p.) mudel,
   - (1 p.) keel,
   - (1 p.) liides (interface).

   *Materjal:*
   http://courses.cs.ut.ee/2007/tvt/uploads/Main/se_07_08.pdf

3. (4 p.)
   *Küsimus:*
   Name and describe three branching and code propagation models in configuration management. Which problems are they solving? Describe their advantages and disadvantages.

   *Lahendus & hindamine:*
   - (1 p.) No branching, code freeze
     - All new developments must wait until stabilization & production support

is finished. After new development is started, there's no easy way to make bugfixes to older releases.

- (1.5 p.) Branch-off on release
  - No code freeze is needed, one can develop new functionality and fix bugs in older releases in parallel. When one needs to start stabilization/release, new branch is created for those purposes. All development continues in main branch; bugfixes are synced back to main branch afterwards.
- (1.5 p.) Continuous forward integration
  - Bug-fixes are immediately propagated forward into newer development/stabilization/support branches. There is more merging happening per one change, but all the bugfixes will be included into new branches immediately.

*Materjal:*
http://courses.cs.ut.ee/2007/tvt/uploads/Main/se_07_16.pdf
Slides 22-25

4. (4 p.)
   *Küsimus:*
   What is the difference between a Web service and a software service? Give examples.

   *Lahendus:*
   - A web service is a particular type of service that is implemented and accessed using Web standards, such as HTTP, XML and derivatives thereof (e.g. SOAP, WSDL).
   - A software service is more general: it is any unit of software that is continuously maintained and abides by certain principles. A software service can be implemented using other technology (e.g. Java RMI).

   *Hindamine*:
   Give 2 points if the distinction "web service more specific than software service" is mentioned in the answer. Give one or two additional points if further details are given such as in the above answer (e.g. mentioning specific technology, or mentioning specific principles).

5. (10 p.)
   *Küsimus:*
   Andmetöötlus:
   Olgu vaja luua arvutimängu konfiguratsioonifail kasutades TLV (tag-length-value) formaati. Teadaolevalt on vaja hoida andmeid arvutimängu järgmiste seadete kohta: ekraani resolutsioon (kõrgus ja laius: nt 800x600 või 1024x768), helitugevuse kohta (väärtused 1..9), mängija nimi, mängija avatari faili asukoht operatsioonisüsteemis (oletagem, et arvestada tuleb vähemalt 300-se pikkusega), kõik mängu mängimise kuupäevad.

   Ülesanne:
   - Välja pakkuda oludele sobivad TLV jaoks vajalike parameetrite väärtused ning kirjutada nende abil 2 (kaks) konkreetsete andmetega täidetud näidist.
   - Kirjeldada selle konfiguratsioonifaili TLV-st tulenevaid puuduseid.

   *Lahendus:*
   - Parameetrid (tudeng ei tarvitse neid kõiki avalikult välja tuua, aga mingit üldkirjeldust ikka eeldaks)

- L pikkus: 3
- Märgendite pikkus: 4 (kuna erinevaid märgendeid on 6, siis piisaks ka pikkusest 1)
- Märgendid:
  - NAME – mängija nimi
  - FILE – avatari faili aadress
  - PLAY – mängimise päev
  - VOLU – helitugevuse tase
  - RESW – resolutsioon laiuti
  - RESH – resolutsioon kõrguseti
- Kodeeringud:
  - kuupäevad kujul YYMMDD

- Näide 1:

```
NAME006INDREKFILE035C:\GAMES\THISGAME\MY FUNNY ICON.PNG
VOLU0011RESW0041200RESH003480PLAY006070912PLAY006070913
PLAY006071025PLAY006080107
```

- Näide 2

```
NAME014ANNABELLACELLARESW003800RESH003600FILE009C:\AC.JPG
PLAY0 06071015VOLU0019PLAY006071017PLAY006071016
```

*Hindamine:*

- Esimene osa:
  - Kui näiteid pole, siis kohe -95% või -100% (kui ülesandes on teine pool ka, siis -90%, mis jätaks tudengile puuduste põhjenduse eest võimaluse 10% kirja saada)
  - Kui pole mingeid kommentaare ega selgitusi parameetrite kohta, siis -15%
  - Kui on ainult üks näide, siis kohe -30%
  - Kui L pikkus on näites ebaühtlane, siis -40% (põhimõtteline viga!). Ebaühtlus oleks näiteks: NAME6indrekFILE20.....
  - Kui eelmine on ok, aga L pikkus on erinevates näidetes erinev (sest ülesanne oli välja töötada üks lahendus ja illustreerida teda 2 näitega), siis -10%. Näiteks, 1. näide: NAME06indrekFILE09... ja 2. näide: NAME006indrekFILE101...
  - Kui L pikkus < 3, siis -10% (ülesannet ei loetud hoolikalt)
  - Kui T pikkus ebaühtlane, siis -40% (põhimõtteline viga). Näiteks: NAME....VOLUME...
  - Kui eelmine on ok, aga T pikkus eri näidetes on erinev, siis -10%
  - Kui veidi eksitud failinime pikkuse leidmisega, siis -0%
  - Aga kui on näha, et eksimus tuleneb tühikute ja kirjavahemärkide mitteloendamisest, siis -1%
  - Kui eksitud korra või kaks helitugevuse või reolutsioonide pikkuse leidmisega, siis -2%
  - Kui mängude kuupäevade osa ei kordu (st on mõlemas näites täpselt 1 kord), siis -3% (halva näite koostamise eest).
  - Kui lõpptulemus on <0, siis 0
- Teine osa:
  - Puudused võiksid olla: antud ülesande juures peab L osa olema vähemalt kolmekohaline, et mahutada arvu 300 ja sellest tulenevalt võtab ta igal pool mujal liigsed 2 kohta.
  - Korduvate märgendite korral ei tea me korduste arvu enne kui pole kogu faili läbinud (selle jaoks võiks ka muidugi eraldi märgendi teha).

- Hindamine: kui puuduseid ei teata, siis -10%
- Kui puuduseid ei teata, aga hämatakse midagi puuduvatest parameetritest (nt värvide arv), siis lisaks -10%.

6. (11 p.)
   *Küsimus & Lahendus & Hindamine:*
   1. (2 p.) Define a metric, explain the purpose of metrics and how they are applied.
      - (2/3 p) A metric is a quantitative representation of otherwise vague attributes (keywords: quantitative, objective; vague, subjective;)
      - (2/3 p) A general purpose of metrics is improvement: track how decisions (taken earlier) change the situation and obtain input for making new decisions. In other words helps one to control the situation.
      - (2/3 p) The most important aspect of metric application is iterativity (periodic, cyclic). One cycle consists of measuring and calculation of metric values, interpretation of metric values and reaction (decision-making).
   2. (2 p.) Describe the classification of software metrics. Explain each class with a couple of sentences and give two examples per class.
      - Subject of measurement:
        - *Process* - efficiency of process application. Examples: Number of completed tasks, length of iteration, number of tasks per iteration.
        - *Resource* - resource usage and properties. Examples: Staff competency (expertise), Staff attrition (fluctuation), Budget, CPU utilization, Staff utilization.
        - *Product* - product properties. Examples: Lines of Code, Function Points, Source Code Statements, McCabe's Cyclomatic Complexity, Density of Comments, Code Coverage
      - Direct Measuring vs Combination:
        - *Direct* - directly measurable, one measured attribute or entity. Examples: Lines of Code, Source Code Statements, McCabe's Cyclomatic Complexity, Number of Methods, Number of requirements.
        - Indirect - not possible to measure directly, metric value is a result of combining several metrics, hence involves several attributes or/and entities. Examples: bug density (number of bugs per Line of Code), Density of Comments (ratio of number of lines with commentaries to the total number of lines), requirement stability (ratio initial number of requirements to the total number of requirements).
      - Importance of measurement environment:
        - *External* - can be measured only with respect to environment. Examples: performance (on the particular CPU, under particular load), Usability (on the particular PC, by particular "class" of users).
        - *Internal* - can be measured in terms of only the entity itself, measurement environment is not relevant. Examples: Lines of Code, Density of Comments, Code Coverage, Requirement Stability.
      - **Evaluation**: Totally 7 (3+2+2) different classes. Each correctly described class -> 0.25 p (class description -> 0.15 p, examples -> 0.1 p). All 7 classes totally give 1.75 p. Structured and explained presentation of classes (classes grouped by principles) -> 0.25 p.
   3. (1 p.) What are common criticisms of software metrics.

- *People adjust* - when people know they are measured they seek to maximize managements perception of their performance.
- *Easy to mistreat* - more talented programmers get more difficult tasks, takes more time and so on. One should be careful and know the context of measurement. A person from outside (e.g. head of department) may mistreat the numbers and make false conclusions.
- *Either meaningful or accurate* - it is said that no existing metric is both meaningful and accurate at the same time. For instance, Lines of Code (as size estimation) is very accurate but doesn't have a global meaning (doesn't take complexity into account) on the opposite Function Points is more meaningful (estimates complexity too) but not accurate (complexity estimation is subjective).
- **Evaluation**: Any described reasonable criticism -> 0.3 points (reasonable = listed on the slide, listed in the wikipedia or any other with correct argumentation)

4. (3 p.) Characterize *Lines of Code*, *Function Points*, and *McCabe's Cyclomatic Complexity*.
   - (1 p.) *Lines of Code* - represents size of the software (product). Counts the number of source code lines of the software. While accurate and is easy to measure, should be used with care as may give different value for the same piece of code due to different formatting styles and counting/not-counting blank lines and commentaries. Heavily depends on the programming language. Doesn't respect complexity of the code.
   - (1 p.) *Function Points* - represents size of the software. Counted on the Use-Case scenarios. Each scenario is evaluated for 5 different types of "interactions" or "components": external input, external output, external inquiry, internal logical files and external interface files. Complexity of each such component is estimated: Low, Average and High. Complexity is used to award number of function points for the given component. Function points for all use-case components are summed up and is used to evaluate size of the use-case. When summed up for all use-cases -> size of the software. May be used earlier than lines of code as no source code is needed. Independent of programming languages. Complexity estimation is subjective - not accurate.
     **Evaluation**: List of "interactions"/"components" is not required!
   - (1 p.) *McCabe's Cyclomatic Complexity* - represents complexity of the software. Shows number of linearly independent paths through a program (or method). Usually calculated on the flow graph of the method. Two formulas to calculate: e-n+2 (where e- number of edges, n- number of nodes); d+1 (where d- number of control statements like IF, WHILE, FOR, THROW). Has strong relation with source code coverage by unit tests. A well known heuristic -> when a value for the method is larger than 10, then method is to complex and has high probability of bugs. McCabe's Cyclomatic Complexity doesn't respect data structure and data flow complexities.
     **Evaluation**: one formula is enough (-0.25 if no formulas). mention of relation with software testing -> required (-0.25 if not mentioned).
5. (3 p.) Propose one direct and two indirect metrics that represent attributes of an exam. The description of each metric should clearly state which attribute of an exam it represents and explanation of the metric idea.
   - See examples in exam question. Also slide 47 of lecture 23.
   - Evaluation: Each metric gives 1 p.
   - *Direct metric* - something directly measurable like number of questions, number of points, exam time.
   - *Indirect metric* - combines several metrics together like topic coverage

by exam, average number of points per exercise (number of exercises, total number of points)

7. (13 p.)
*Ülesanne:*
Disaini primitiivne pangasüsteem.

Kõik toimingud käivad tellerite kaasabil. Telleri rakenduses peavad olema võimalikud järgmised tegevused:
- Kliendigruppide kirjeldamine.
- Igale (*) märgitud tegevusele kliendigrupipõhine teenustasu kirjeldamine.
- Uue kliendi registreerimine, kliendigrupi määramine.
- Sularaha sissemaks kontole (*) (enda kontole tasuta, teistele teenustasuga).
- Sularaha väljamaksmine kontolt (*).
- Pangasisesed maksed (*).
- Rahvusvahelised maksed (*).
- Tähtajaliste hoiuste tegemine, neile intressi maksmine tähtaja lõppedes.
- Etteantud perioodi konto väljavõtte trükkimine.
- Iga valuuta tähtajaliste hoiuste intresside registreerimine.
- Raport etteantud perioodil võetud teenustasudest kliendigruppide lõikes.

Vastus peab sisaldama:
- andmemudelit,
- klassidiagrammi,
- käitumist kirjeldavad skeemid,
- selgitav tekst,
- olulisemate meetodite pseudokood,
- olulised erijuhud, veasituatsioonid,
- kasutatud disaini mustrid.

Igasuguse disaini juures on oluline, et tulemus oleks selgelt arusaadav, kood hästi loetav ning hilisem muudatuste/täienduste tegemine lihtne.

Mõtle, kui lihtne oleks tulevikus:
- uute tegevuste lisamine (valuutavahetus, laenud jne.),
- lisaraportite tegemine (ülevaade päeval sooritatud maksetest, kasumlikumad kliendid jne.),
- internetipanga tegemine.

Hea disaini korral peaks nende funktsioonide lisamine tähendama minimaalseid muutusi olemasolevas koodis ja lihtsalt uue loogika juurde kirjutamist.

*Lahendus & Hindamine:*
- Olulised aspektid, millele peaks olema tähelepanu pööratud:
  - Mõisted klient vs. konto - eraldi ei ole ülesande püstituses välja toodud konto avamist - kuidas on tudengid selle lahendanud?
  - Valuuta mõiste - mõnest nõudest paistab välja vajadus erinevate valuutade käsitluse järgi - kas tudeng on mõelnud, et kontol olev raha on mingis kindlas valuutas ning kas kontol võib näiteks olla erinevaid valuutasid?
  - Mõistlik oleks kõik toimingud (sisse- ja väljamaksed,maksekorraldused, jne.) kirjata ühte tabelisse kui kontol toiminud liikumised - see lihtsustab

väljavõtete tegemist.
    - Ka teenustasude võtmine peaks olema kirjeldatud kui liikumine kontol, samal ajal ja ideaalis seotuna alustoiminguga
    - Kuidas on realiseeritud pangasisene makse - kas see registreeritakse korralikult mõlema osapoole kontol?
    - Tähtajaliste hoiuste tegemine - kuidas toimub raha liikumine kontolt tähatajalisele hoiusele? kuidas toimub intressiarvestus? Kas arvestatakse eri valuutade erinevaid intresse? kuidas toimub raha ja teenitud intressi tagasi maksmine kliendile deposiidi lõppedes?
- Vastus peaks ka sisaldama klassidiagrammi ning andmemudelit – erinevus peaks olema selles, et andmemudelis on välja toodud tabelid koos primaarvõtmete ning viidetega teiste seotud tabelite primaarvõtme(te)le; klassidiagramm peaks aga mõistlikult kirjeldama objektmudelit koos korralike seostega objektide vahel.
- Kuigi ülesande püstituses ei ole tähelepanu pööratud kasutajaliidesele, siis peaks heas lahenduses selgelt olema eristuv API panga funktsioonide jaoks, mida võiks kasutada erinevad kasutajaliidesed.
- Oluline on lahenduse selgus ning lihtsus ja erinevate alamosade eristatavus.
- Standardite (nt UML) täpne tundmine ei saa olla määrav.