TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Märten Mändla 155526IABB

ISC ÕPPEKESKKONNA SOBIVUSE ANALÜÜS PROGRAMMEERIMISAINETE ÕPETAMISEL

Bakalaureusetöö

Juhendajad: Gunnar Piho, PhD
Dotsent
Vello Kukk, PhD
Emeriitprofessor

Tallinn 2018

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Märten Mändla

21.05.2018

Annotatsioon

Käesoleva lõputöö eesmärgiks on analüüsida ISC õppekeskkonna sobivust programmeerimisainete õpetamisel. Äriinfotehnoloogia esimese kursuse tudengite seas viidi läbi eksperiment, kuidas ISC õppekeskkond mõjub programmeerimise õppimise efektiivsusele.

Eksperimendi jaoks on disainitud kaht tüüpi ülesandeid. Selleks on staatilised ülesanded ning tudengite poolset sisendit koguvad ülesanded, millest on võimalik genereerida uusi valikvastustega ülesandeid.

Eksperimendi käigus selgus, et ISC õppekeskkond pole piisavalt sobilik vahend programmeerimisainete õpetamisel, kuna sellele esineb liiga palju tehnilisi puudusi. Antud keskkonnast on mõistlik välja arendada raamistik, kuhu on võimalik sisestada mistahes ülesandeid ning milles säilib kompetentsipõhine õpe ja mälumudel.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 31 leheküljel, 5 peatükki, 20 joonist, 7 tabelit.

Abstract

Suitability analysis of the ISC environment in teaching software programming

The aim of the thesis is to analyze the suitability of the ISC environment in teaching software programming. An experiment was carried out among first year students studying Business Information Technology. The purpose of the experiment was to examine what effect does the ISC environment have on efficiency of studying programming.

There were two types of tasks designed for the experiment. These are static tasks and students' input gathering tasks from which new tasks with multiple choices can be generated.

The result of the thesis is that the ISC environment is not enough suitable tool in teaching software programming because of the many technical issues. The solution is to develop a competence based framework where any kind of tasks can be inserted. The framework should also include the forgetting model.

The thesis is in Estonian and contains 31 pages of text, 5 chapters, 20 figures, 7 tables.

Lühendite ja mõistete sõnastik

IKT	Info- ja kommunikatsioonitehnoloogia
ISC	TTÜ endine siduteooria ja -disaini õppetool
IDE	Integreeritud arenduskeskkond
Refaktoreerimine	Koodi struktuuri parandamine nii, et selle funktsionaalsus ei muutu
TDD	*Test-Driven Development* ehk testidel tuginev arendamine

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tänapäeval on IKT sektoris puudu tuhandeid töötajaid. Tallinna Tehnikaülikoolis alustab enamik infotehnoloogia teaduskonna tudengeid programmeerimise õppimisega juba esimesel semestril. TTÜ on läinud üle ka uuele struktuurile ning sellega seoses on uuendatud õppekavasid. Struktuurimuutusega suurenes ka ainete maht. Tänasel päeval on õppekava statuudis määratud, et õppeaine maht võib olla üldjuhul 6, 9 või 12 EAP. Kui õppida programmeerima, siis tudeng ei tea, kuidas seda teha võimalikult efektiivselt.

Aine õppimisel on tudengile oluline õppekeskkond. Iga õppejõud hoiab enda materjale talle sobivas keskkonnas: Maurus, Moodle, ained.ttu.ee jne. Igaüks neist on erinev ja omade plusside ning miinustega. Programmeerimist õppivale tudengile on eelkõige vaja interaktiivset keskkonda ning seda, et õppimine talle huvitav oleks. Õppekeskkondades puudub hetkel võimalus sisestada koodijuppe ning saada nendele kohest tagasisidet. Enamasti kontrollitakse teadmisi testidega, mis koosnevad valikvastustest. Kompetentsil põhinevad õppekeskkonnad on need, mis kinnistavad omandatud teadmisi kõige paremini. Saavutatud oskusi süvendabki nende regulaarne kontroll teatud ajaperioodi järel. TTÜ-s on infotehnoloogia teaduskonnas seni ainult üks ainekeskkond, mis on kompetentsipõhine ning selle eestvedajaks on olnud emeriitprofessor Vello Kukk.

Töö eesmärk on analüüsida ISC õppekeskkonna sobivust programmeerimisainete õpetamisel. Äriinfotehnoloogia eriala esimese kursuse tudengite seas viiakse läbi ka eksperiment, kuidas ISC õppekeskkond mõjub programmeerimise õppimise efektiivsusele. Oluline on välja selgitada, millistele oskustele pannakse programmeerimise õpetamisel rohkem rõhku ning mil moel need selgeks saadakse. Kuna tudengitel on kompetentsil põhinevate õppekeskkondadega üldjuhul vähene kokkupuude, siis läbi eksperimendi tulemuste on eesmärk välja uurida, kas tulevikus on mõistlik panustada selliste õppekeskkondade arendusse.

Töö on jaotatud peatükkideks, lähtudes töö eesmärgist. Esmalt on ära seletatud metoodika, kuidas töö läbi viiakse. Seejärel räägitakse tehtud tööst ehk kirjeldatakse

õppekeskkonna jaoks loodud materjale. Viimane peatükk keskendub tudengite peal läbi viidud eksperimendi analüüsile.

2 Teoreetilised ja metoodilised alused

2.1 Infosüsteemide arendamise õppeained

Õppeaine, mille raames ISC õppekeskkonda analüüsitakse, on Infosüsteemide arendamine I: baasoskused ainekoodiga ITB1701. Õppeaine maht on 12 EAP. Aine eesmärk on anda edasi programmeerimise ja infosüsteemide arendamise baasteadmised, et tulevikus jätkata iseseisvat õppimist. Kursus keskendub C# programmeerimiskeelele ning integreeritud arenduskeskkonnaks on Visual Studio. Aine on eelduseks jätkuainele Infosüsteemide arendamine II: veebirakendused ainekoodiga ITB1702.

Tudeng, kes on antud kursuse läbinud, tunneb õppeaine väljunditena tarkvaraarenduse kõiki etappe ja oskab luua nende vahel seoseid. Tudengile pole võõrad sellised mõisted nagu refaktoreerimine ja ühiktestimine. Üliõpilane on saavutanud oskuse programmeerida ja arendada lihtsamaid rakendusi. Tudeng oskab programmeerimisel kasutada refaktoreerimist, et kirjutada puhast koodi. Üliõpilane omab teadmisi testidel tuginevast tarkvaraarendusest ning suudab iseseisvalt kirjutada ühikteste koodile, mis on valideeritud [1].

Õppeaine käigus õpitakse, kuidas infosüsteeme praktiliselt kavandada ning arendada. Selleks kasutatakse integreeritud arenduskeskkonda ehk IDE. Lisaks õpitakse tundma erinevaid metoodikaid nagu *Extreme Programming, Test Driven Development* (TDD) ja *Clean Code* [1].

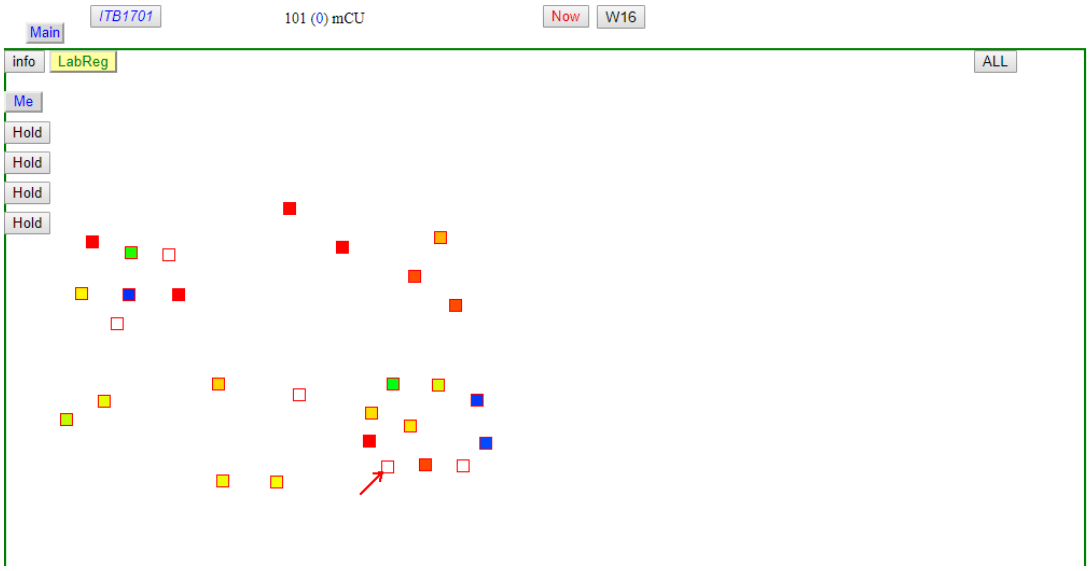
Kursuse üks peamisi teemasid on IDE ja selle võimaluste tundma õppimine, omandatakse arusaam versioonihaldusest ning infosüsteemide arhitektuurist. Tudeng õpib tundma programmeerimist. Antud kursusel kasutatav programmeerimiskeel on C#. Programmeerimise käigus omandatakse teadmised nii väljadest, muutujatest, meetoditest kui ka kasutatavatest andmetüüpidest, tsüklitest ja hargnemistest. Antud õppeaine käigus õpitakse tundma ka objekt-orienteeritud programmeerimist, mille peamine fookus on klassidel ja objektidel. Aine raames antakse ülevaade tarkvaratehnikast kui metoodikast ja protsessist [1].

2.2 ISC õppekeskkond

ISC õppekeskkonna nimi tuleneb lühendist, mis tähistab endist automaatikainstituudi siduteooria ja -disaini õppetooli. Õppekeskkond lähtub kompetentsipõhisest õppest ja selle eesmärgiks on, et tudeng omandab tugevad ning sügavalt kinnistatud uued teadmised ja oskused [2]. Teadmiste kinnistamiseks kasutatakse väikeseid elementaarkompetentse, mille käigus tuleb erinevaid elementaarülesandeid mitmeid kordi lahendada, et oskus ka ajas säiliks. Kompetentsipõhisel õppel kasutatakse väikeseid ülesandeid, mille lahendamine võtab aega mõne minuti. Oluline on ajas korrata sarnaseid ülesandeid ja kompetentse.

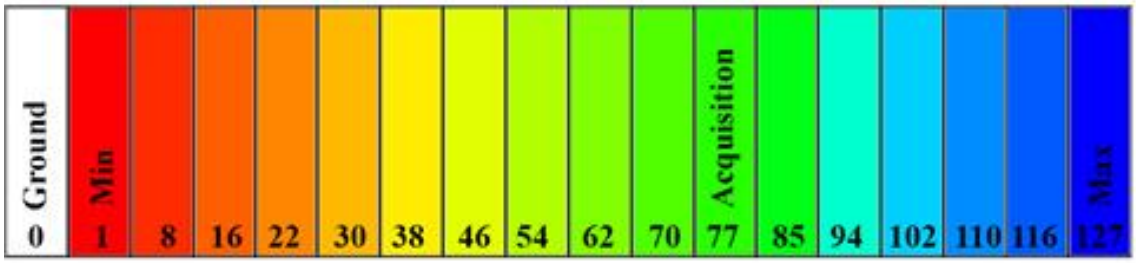
Kui ülesannet ei korrata, siis mälumudeli kohaselt vähendatakse kompetentside saavutatud väärtusi ning tulemusi ajas. Seda tehakse eesmärgil, et kinnistada tudengi teadmisi vastava kompetentsi kohta. Mälumudelit kasutatakse ka selleks, et vältida ebavajalikke pingutusi, mis õppetöös teadmisi juurde ei anna [3]. Õppetöö maht on individuaalne, mis tähendab, et puudub kindel arv ülesandeid, mis on vajalik aine läbimiseks. Õppekeskkond, kus paiknevad antud programmeerimise ülesanded, asub aadressil http://193.40.240.71. Süsteemi eestvedajaks on olnud emeriitprofessor Vello Kukk. Õppekeskkonda sisenemiseks on vajalik registreerida kasutaja, kus kasutajanimeks on tudengi matriklinumber või isikukood.

Süsteem on jagatud mitmeks leheküljeks. Aine õppimise alustamiseks tuleb minna leheküljele Ained ning sealt valida õppimiseks vajalik aine. Tegemist ei ole ametliku deklareerimisega, vaid eeldeklareerimisega. Õppeaine ametliku deklareerimise protseduur tuleb läbi viia õppeinfosüsteemis. Kui aine on eeldeklareeritud, siis ilmub see leheküljele *MyField*, kus on võimalik õppimisega alustada.



Joonis 1. MyField lehekülg ISC süsteemis

MyField on ISC õppekeskkonnas keskne lehekülg, kus toimub kogu õppetööga seonduv tegevus. Leheküljel on võimalik lahendada ülesandeid, registreerida end kontrolltööle ja kinnitada hinne. Aine leheküljel on nähtavad väikesed kastikesed. Need on kursuse sisuks olevad elementaarkompetentsid. Kastile klikkides leitakse tudengile sobiv ülesanne vastavalt valitud kompetentsile. Kui liikuda hiirega kasti kohale, siis ilmub nähtavale kompetentsi kirjeldus. Kui ülesanne on lahendatud, tuleb vajutada nupule Vasta. Ülesande hindamine toimub automaatselt ja koheselt kuvatakse ka tulemus. Kui ülesanne on edukalt lahendatud, siis kuvatakse sinine kast. Vale vastuse korral on tulemuseks punane kast. Võimalusel kuvatakse ka kommentaar vea kohta. Vajadusel saab saata õppejõule teate täiendavate küsimuste küsimiseks. Selleks tuleb vajutada nupule *Ask Vello*. Ülesannet lahendades on võimalik see ka ootele panna, vajutades nupule *Hold*. See tähendab, et ülesande juurde saab hiljem tagasi tulla ja edasi lahendada. Ülesannete lahendamisel hakkab aine leheküljel muutuma kastikeste ehk kompetentside värv. Mida sinisem kast, seda edukamalt on kompetentsi omandatud. Kui kast on punane või oranž, siis see tähendab, et ülesannete lahendamisel on esinenud vigu.



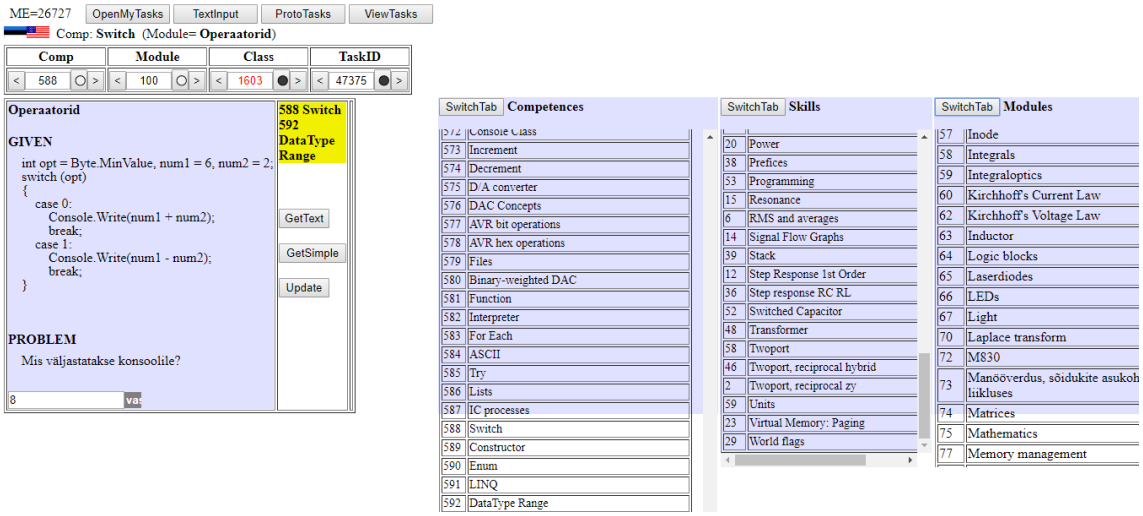
Joonis 2. Kompetentside tasemed värvidega [2]

Kompetentside tasemed ulatuvad 0-st 127-ni. Oluline tase on 77. Seda loetakse piiriks, millest kõrgemal olles eeldatakse, et tudeng on vastava kompetentsi omandanud [2]. *MyField* lehel on olemas nupp *ALL* ja *U77*. Nupp muudab enda nimetust selle peale klikkides. Kui nupp on nimetusega *ALL*, siis kuvatakse kasutajale kõik kompetentsid. *U77* korral kuvatakse vastavad kompetentsid, mille tase jääb alla 77. Mälumudel korrigeerib ajas automaatselt tasemeid. Aine lehekülje ülaosas on kaks nuppu, *NOW* ja *W16*. Kui vajutada nupule *NOW*, siis kasutajale kuvatakse kompetentside tasemed praegusel hetkel. *W16* nupule vajutades kuvatakse kasutajale vastavad kompetentside tasemed, millised need on 16 nädala möödudes.

MyField leheküljel on kuvatud ka nn elementaarkompetentside korrused. Korruseid on kokku kolm. Korrusel 0 on näha kõik võimalikud elementaarkompetentsid ja neid saab eraldi õppida. Korrus 0 on ka kõige detailsem. Korrus 1 kombineerib omavahel sarnased kompetentsid ning moodustab neist suuremad teemad. Kui vajutada teema pealkirjale peale, siis kuvatakse kasutajale kõikide selle teemaga seonduvate kompetentside tasemed. Korrus 2 on veelgi üldisem. Antud korrus kuvab kasutaja taseme kõigis süsteemis deklareeritud ainetes. Nii on võimalik tudengil jälgida enda õppimise arengut mitmes aines korraga.

ISC süsteemis hallatakse ülesandeid *QUIZScan* leheküljel. Leheküljele sisenemiseks on vaja sisestada parool. Leheküljel on kuvatud ülesanne, kompetentside tabelid, nupud ning väljad, mille järgi ülesandeid otsida. Väli *Comp* on seotud elementaarkompetentsiga. Igal kompetentsil on vastav identifikaator, mille leiab ülesande kõrval asuvast tabelist. Tabelit on võimalik sorteerida kas tähestiku või numbrilises järjekorras *SwitchTab* nupule vajutades. Väli *Class* esindab mingisugust ülesannete klassi. Tegu on suuremate teemadega, kuhu on kokku kogutud sarnased kompetentsid. *Class* on analoog *MyField* lehel olevale korrusele 1. Väli *Module* koondab väga üldisi teemasid ehk mooduleid. Ühes moodulis võivad esineda ülesanded mitmest erinevast klassist. Väli *TaskID* on

ülesande unikaalne identifikaator. Väljade all on kuvatud ülesanne koos küsimuse ning vastusevariantidega või tekstiväljaga. Lisaks on välja toodud ka õige vastus.



Joonis 3. ISC süsteemi QUIZScan lehekülg

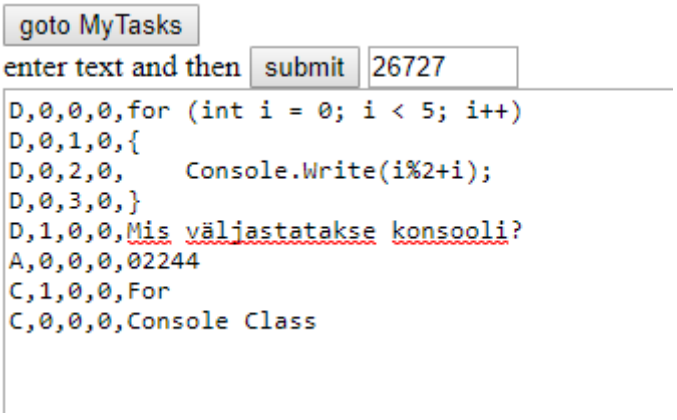
2.3 Ülesannete koostamise põhimõtted

Ülesannete koostamisel on lähtutud *Microsoft Virtual Academy* poolsest videosarjast *C# Fundamentals for Absolute Beginners*. [4] Videosarjas on läbi võetud kõik peamised vajalikud baasoskused, mis on vajalikud edasiseks iseseisvaks õppimiseks. Uuritud on ka erinevaid materjale internetis. Lisaks on loetud ka Bertrand Meyeri raamatut *Touch of Class: Learning to Program Well with Objects and Contracts*. Raamatu sisus leiduvad teemad on võetud aluseks, et neist luua ülesannete disainimiseks vajalikud kompetentsid. Raamatus käsitletud teemad on kirjeldatud programmeerimiskeeles Eiffel. Lisaks on välja toodud kokkuvõtlik sissejuhatus C# keelde. Antud teoses on kirjeldatud programmeerimiseks vajalikud baasteadmised. Erinevate definitsioonide abil on olnud võimalik leida kompetentse ning kogutud ideid ülesannete disainimiseks. [5]

Iga ülesandega on seotud üks või mitu konkreetset kompetentsi, mille omandamist kontrollitakse. Disainitud küsimuste eesmärk on võtta erinevad programmeerimise oskused, otsida välja nende detailid ning selle põhjal luua küsimus. Kui tudeng oskab vastata lihtsamatele ja lühematele küsimustele, siis saab ta vastata ka keerulisematele küsimustele, mis sisaldavad endas rohkem koodi. Küsimuste keerukus kasvab, kui

suureneb õigete vastuste arv. Kõige peamine kompetents, mida disainitud materjalidega kontrollitakse, on programmeerimiskeele C# fundamentaalsete teadmiste omamise oskus.

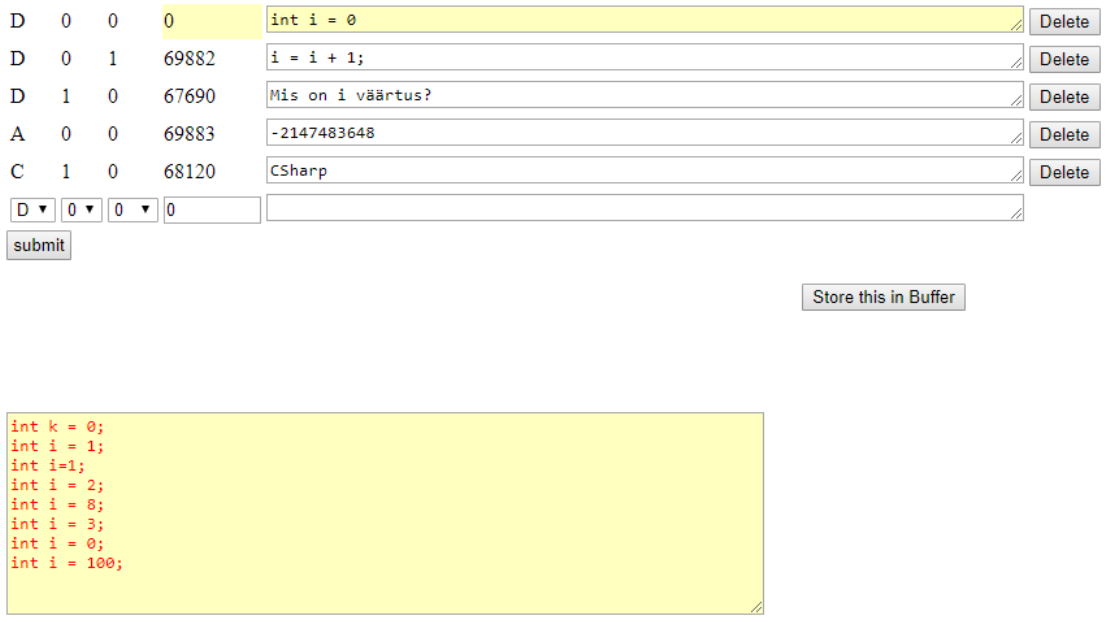
Ülesande lisamine toimub *QUIZScan* leheküljelt (Joonis 3). Selleks tuleb vajutada nuppu *TextInput*. Kuvatud leheküljel on suur tühi tekstiväli, kuhu tuleb kindlas formaadis ülesanne reahaaval sisestada. Esmalt sisestatakse andmete read ehk ülesanne ise ning küsimus. Andmete read on tähistatud tähega D. Seejärel sisestatakse vastuse rida, mis on tähistatud A tähega. Viimasena lisatakse juurde kompetentsidega seotud read, mis tähistatakse tähega C. Ülesande lisamise formaadi veergude väärtustel on ka kindel tähendus. Esimeses veerus on tähemärk, mis tähistab andmeid, vastust või kompetentsi. Teises, kolmandas ja neljandas veerus on numbrilised väärtused. Kui teise veeru väärtus on 0, siis see tähendab, et kirjeldatakse ülesande sisu. Seejuures kolmanda veeru väärtused kasvavad reahaaval. Kui teise veeru väärtus on 1, siis see tähendab küsimuse rida. Kui küsimus on mitmel real, siis vastavalt sellele kasvab ka kolmanda veeru väärtus reahaaval. Neljanda veeru väärtused on iga andmerea unikaalsed identifikaatorid. Kui sama koodirida kordub mingis muus ülesandes, siis kasutatakse ülesande juures sama identifikaatoriga koodirida. Ülesannet lisades on identifikaatori väärtuseks 0. Kui ülesanne saata puhverserverisse, siis süsteem korrigeerib identifikaatorid vastavalt sellele, kui leidub korduvaid ridu. Viiendas veerus on kuvatud andmed ise ehk koodiread. Ülesannet saab saata puhverserverisse, kui vajutada *Submit* nuppu. Et ülesannet puhverserveris näha ja kohendada, tuleb vajutada nupule *goto MyTasks*.



Joonis 4. Ülesande lisamise formaat

Disainitud ülesannete kohendamiseks ja redigeerimiseks tuleb *QUIZScan* lehel (Joonis 3) vajutada nupule *Update* või ülesannet lisades vajutada nupule *goto MyTasks* (Joonis 4). Selle järel saadetakse antud ülesanne puhverserverisse ja seal võetakse ülesanne ühe rea pikkusteks osadeks lahti. Esmalt kuvatakse ülesande andmete osa ehk koodiread, seejärel

vastus ning siis kompetentsid, mis on antud ülesandega seotud. Ülesande kohendamine toimubki reahaaval. Kui ühel real olevat teksti muuta, siis süsteem kuvab ülesande all olevas tekstiväljas ette erinevaid soovitusi. Need on süsteemi poolsed soovitused, et varieerida ülesandes olevaid andmeid ning luua tänu sellele uusi ülesandeid. Soovitused ilmuvad tekstivälja vastavalt sellele, kui palju on süsteemis juba sarnaseid ülesandeid olemas. Lisaks on võimalik ülesande kohendamisel ridu lisada ning ka kustutada. Ridade lisamine toimub *Submit* nupu abil ning kustutamine *Delete* nupu abil. Kui ülesanne on kohendatud, tuleb vajutada nupule *Store this in Buffer*. Seepeale saadetakse kohendatud ülesanne uuesti puhverserverisse. Tekib uus nupp nimega *Save this task*. Sellele vajutades salvestatakse kohendatud ülesanne tagasi süsteemi.



Joonis 5. Süsteemi poolsete soovituste kuvamine

Sarnaste ülesannete tegemiseks ehk varieerimiseks on kaks viisi. Üks neist on *QUIZScan* lehel (Joonis 3) nuppu *GetText* vajutades. Kui sellele nupule vajutada, siis viiakse see ülesanne puhverserverisse. Ülesanne tuleb reahaaval ära muuta, ridu võib lisada ja kustutada. Kui see on tehtud, siis tuleb vajutada nupule *Store this in Buffer*. *Save this task* nupule vajutades tekib uus ülesanne, millel on ka uus *TaskID*. Teine võimalus on *QUIZScan* lehel *GetSimple* nuppu vajutades. Kuvatakse tekstiväli, kus võib muuta ainult nii palju ridu kui on tekstiväljal kuvatud. Kui read on muudetud, tuleb vajutada *Submit*

nupule. Seejärel saadetakse muudetud ülesanne puhverserverisse, kus on veel võimalik seda kohendada. Vajutada tuleb nuppudele *Store this in Buffer* ja *Save this task* ning uus ülesanne on süsteemi salvestatud.

3 Disainitud materjalid

3.1 Kompetentsid ja staatilised ülesanded

Disainitud ülesandeid on kaht tüüpi. Esimese tüübi moodustavad staatilised ülesanded, millel on kindel küsimus ja üks konkreetne vastus. See tähendab, et ühel ülesandel ei saa olla kaks paralleelset õiget vastust. Staatilised ülesanded on võimalikult lühikesed ning lihtsad. Ülesanne on koodilõik mõnest suuremast programmist. Üldjuhul on ülesande pikkus kuni 10 koodirida. ISC õppekeskkonnas *QUIZScan* lehel (Joonis 3) asuvad need ülesanded klassis 1603.

Peamised kompetentsid, mida sai disainitud ülesannete abil kontrollitud, olid andmetüübid ja nende vormindamine, matemaatilised tehted, tingimuslaused, tsüklid (*for, foreach, while*), massiivid, meetodid ja loendid. Antud kompetentsid said valitud seetõttu, kuna just nendes oskustes on veel täpsemaid detaile, mida kontrollida. Oluline asjaolu on see, et esialgu võivad need detailid tunduda tähtsusetud, kuid tegelikkuses on võimalik neist moodustada uusi kompetentse, mis tähendab, et kontrollitavate oskuste hulk suureneb. Kuna iga ülesandega võib olla seotud rohkem kui üks kompetents, siis võib toimuda ka mitme oskuse paralleelne kontrollimine.

Järgnevalt on välja toodud tabel eksperimendis kasutatud kompetentsidega ning nendega seotud ülesannete arv. Kõige rohkem on disainitud ülesandeid matemaatiliste tehete ja *for* tsükli kompetentsiga. Antud tabelis olevad kompetentsid on seotud nii staatiliste ülesannete kui ka sisendit koguvate ülesannetega.

Tabel 1. Kompetentside tabel

Kompetents	**Kompetentsiga seotud ülesannete arv**
Andmetüübid, väärtuste vahemik	23
Stringid	26
Matemaatilised tehted	52
if tingimuslause	27

Kompetents	**Kompetentsiga seotud ülesannete arv**
switch tingimuslause	3
for tsükkel	52
foreach tsükkel	4
while tsükkel	22
Massiiv	40
Loendid	3

Tihti on oluline kompetentse koos rakendada ja mitte täiesti sõltumatult, see võib tagada oskuste efektiivsema kontrollimise. Kompetentsid jagunevad sisend- ja väljundkompetentsideks. ISC õppekeskkonnas *MyField* leheküljel olevad kastikesed tähistavadki sisendkompetentse. Nende kaudu pakutakse tudengile ülesanne. Ülesande lahenduse juures hinnatakse mõlemat tüüpi kompetentse, kusjuures väljundkompetentside hindamine on vähem prioriteetsem [6].

3.1.1 Andmetüübid ja nende vormindamine

Andmetüüpide puhul on oluline kontrollida, kas tudeng teab iga andmetüübi omadusi. Tõeväärtuse korral on võimalikeks väärtusteks *true* või *false*. Arvuliste andmetüüpide puhul tuleb teada ka väärtuste vahemikku. Andmetüüpide vormindamine võib toimuda näiteks siis, kui mingi arv on esitatud stringina ja seda konverteeritakse tagasi arvuks. Andmetüüpide juurde on lisanduvate kompetentsidena juurde toodud matemaatilised tehted. Matemaatiliste tehete abil on võimalik kontrollida teadmisi andmetüüpide omaduste osas.

```
int i = Int32.MaxValue;
i = i + 1;
```

Joonis 6. Andmetüüpide kompetentsi ülesande näide

Antud näites on deklareeritud täisarv i, mille väärtuseks on maksimaalne täisarv ehk 2147483647. Kui sellele liita 1 juurde, siis tulemuseks on täisarvu minimaalne väärtus ehk -2147483648.

3.1.2 Stringid

Eraldi on disainitud stringidega seotud ülesandeid. Seda on tehtud seetõttu, et string on üks peamisi andmetüüpe programmeerimises ning selle meetodite (*Split, Concat, Substring, IndexOf* jne) tundmine on samuti oluline oskus. Disainitud ülesannetes on peamine eesmärk kontrollida tudengi teadmisi stringiga seotud meetodite kohta.

```
string t = "Main", s = "Void";
var d = String.Concat(t, s);
```

Joonis 7. Stringide kompetentsi ülesande näide

Antud näites on defineeritud kaks stringi, mis *Concat* meetodi abil kokku pannakse ja muutuja d väärtuseks saab MainVoid.

3.1.3 Tingimuslaused

Disainitud ülesandeid tingimuslausetega on kaht tüüpi – *if* ja *switch* laused. Kui tudeng lahendab *if* lausega ülesannet, siis oluline oskus antud kompetentsi puhul on arusaamine, kas tingimus *if* lause sees on täidetud või mitte. Sellest lähtuvalt käivitatakse vastav koodilõik ja vajadusel alternatiivsed koodilõigud, kui need on ülesandes välja toodud.

```
int tulemus = 67;
if (tulemus >= 91 && tulemus <= 100)
    Console.Write("Hinne on 5");
else if (tulemus >= 81 && tulemus <= 90)
    Console.Write("Hinne on 4");
else if (tulemus >= 71 && tulemus <= 80)
    Console.Write("Hinne on 3");
else if (tulemus >= 61 && tulemus <= 70)
    Console.Write("Hinne on 2");
else if (tulemus >= 51 && tulemus <= 60)
    Console.Write("Hinne on 1");
else if (tulemus < 51)
    Console.Write("Mittesooritatud");
```

Joonis 8. *if* tingimuslause kompetentsi ülesande näide

Antud näites on deklareeritud täisarvuline muutuja i, mille väärtus on 67. Kontrollitakse tingimust, kas i on vahemikus 91 kuni 100. Kuna i ei sobi sellesse vahemikku, siis minnakse alternatiivsetesse tingimuslausetesse, kus tingimuse tõesuse korral kuvatakse tekst.

Alternatiivne viis, kuidas kontrollida tingimuslausete kohta teadmisi, on *switch* lause, mille puhul kontrollitavateks teadmisteks on arusaamine, mida näitab *switch* lauses *case*

osa. Kui tingimuslauses sisaldub mitu tingimust, siis on nii *if* kui ka *switch* lausel sarnane ülesehitus – *if-else* ja *case* annavad sama väljundi.

```
int opt = Byte.MinValue, num1 = 6, num2 = 2;
switch (opt)
{
    case 0:
        Console.Write(num1 + num2);
        break;
    case 1:
        Console.Write(num1 - num2);
        break;
}
```

Joonis 9. *switch* tingimuslause kompetentsi ülesande näide

Antud näites on deklareeritud täisarvuline muutuja *opt*, mille väärtuseks on 0. Väljundina kuvatakse kahe arvu summa.

3.1.4 *for* tsükkel

Tsüklite puhul on oluline kontrollida, kas tudeng tunneb tsükli struktuuri ja sellega seonduvaid detaile. *for* tsükli juures on vajalik jälgida, millised tingimused on tsüklile rakendatud. Peamiselt on ülesandeid disainides peetud silmas tsükli päises määratud vahemikke ning algväärtuse suurendamist või vähendamist.

```
for (int i = 16; i < 25; i++)
{
    Console.Write("{0}", i%3+5);
}
```

Joonis 10. *for* tsükli kompetentsi ülesande näide

Antud näites on enne tsükli alustamist deklareeritud täisarv i, mille väärtus on 16. Tsükli tingimus on, et i peab olema väiksem kui 25. Tsükli lõpus suurendatakse muutujat i 1 võrra. Kuvatakse läbi käidud arvude jagatised mooduliga 3, millele on juurde liidetud 5. Saadud väärtused kuvatakse üksteise järel.

3.1.5 *foreach* tsükkel

Ülesandeid disainides on *foreach* tsükli kompetentsi kombineeritud massiivi kompetentsiga. Kuna *foreach* tsükkel on väga sarnane *for* tsüklile, siis on vaja kontrollida tudengi teadmisi selles osas, kas ta oskab ka seda tsüklit kasutada.

```
int[] testInts = new[] {4, 8, 2, 3, 76};
foreach (var testInt in testInts)
{
    Console.Write("{0},", testInt);
}
```

Joonis 11. *foreach* tsükli kompetentsi ülesande näide

Antud näites on deklareeritud viie elemendiga massiiv. Läbi *foreach* tsükli käies kuvatakse iga massiivi element ning lisatakse juurde koma.

3.1.6 *while* tsükkel

Antud kompetentsi puhul on oluline kontrollida, kuidas tudeng *while* tsüklit rakendab. Peamine oskus on see, et tudeng saab aru, et tsüklis olev koodilõik läheb käima ainult siis, kui see vastab tsüklis kirjeldatud tingimustele. *while* tsükli kompetentsi on koondatud ka *do while* tsükkel. Nende ülesannete puhul on oluline aru saada, mida teeb disainitud ülesandes *do* osa.

```
int i = 1;
do
{
    Console.Write("text");
    i++;
} while (i < 1);
```

Joonis 12. *do while* tsükli kompetentsi ülesande näide

Antud näites on deklareeritud täisarv i väärtusega 1. Kuna ülesandes on sees *do* osa, siis tänu sellele kuvatakse tekst ja muutujat i suurendatakse 1 võrra. Kuna i pole enam väiksem kui 1, siis rohkem midagi ei kuvata.

```
string s = "Hello World";
int i = 1, k = 0;
while (k <= s.Length - 1)
{
    if (s[k] == ' ')
    {
        i++;
    }
    k++;
}
Console.Write(i);
```

Joonis 13. *while* tsükli kompetentsi ülesande näide

Antud ülesandes kuvatakse täisarvuline muutuja i, mis tähistab stringis s olevate sõnade arvu. Algselt on muutujale i antud väärtuseks 1 ja muutujale k väärtuseks 0. Tsükli tingimuseks on, et k väärtus peab olema väiksem või võrdne stringi s tähemärkide arvuga, millest on lahutatud 1. Need hakkavad olema stringi elementide indeksid. Kui tsükli mõneks elemendiks on tühik, kasvab muutuja i väärtus 1 võrra. Tsükli käigus kasvab ka muutuja k väärtus.

3.1.7 Massiivid

Massiiv on mõeldud selleks, et selles hoiustada ühe ja sama andmetüübi muutujaid. Massiivid on käsitletud samuti eraldiseisva teemana nagu stringid. Sarnaselt stringidega on ka massiividel mitmeid erinevaid meetodeid (*Length, Rank, IsFixedSize, Count, Min, Max* jne). Ülesannete disaini puhul on peamine eesmärk kontrollida tudengi teadmisi ja arusaama massiivi korrektsest deklareerimisest ja selle meetoditest. Massiividega seonduvaid ülesandeid on hea kombineerida *for* või *foreach* tsükliga.

```
int[] numbers = new[] {3, 9, 7, 8};
int c = numbers.Count();
Console.Write(c);
```

Joonis 14. Massiivi kompetentsi ülesande näide

Antud näites on deklareeritud täisarvude massiiv, mis koosneb neljast elemendist. Kuvatakse muutuja c väärtus, mis on massiivi elementide arv.

```
int[] arr = new[] {1, 3, 5, 7, 9};
for (int i = 1; i < 5; i++)
{
    Console.Write(arr[i] * arr[i]);
}
```

Joonis 15. Massiivi ja *for* tsükli kombineeritud ülesande näide

Antud ülesanne on näide sellest, kuidas massiivi elementidega on võimalik tsüklite abil mängida. Ülesandes on deklareeritud täisarvude massiiv, mis koosneb viiest elemendist. *for* tsüklis on algtingimusena deklareeritud täisarv i väärtusega 1. Muutuja i peab olema väiksem kui 5 ja tsükli lõpus suurendatakse muutujat i 1 võrra. Tsükli käigus kuvatakse massiivi elementide ruudud alates teisest elemendist. Saadud väärtused kuvatakse üksteise järel.

3.1.8 Loendid

Loendid on sarnased massiivile. Ka loenditel on mitmeid erinevaid meetodeid. Loendit on võimalik samuti kombineerida nii *for* kui ka *foreach* tsüklitega. Kuna loenditel on rohkem meetodeid kui massiivil, siis annab see veelgi rohkem juurde võimalusi. Peamine eesmärk on kontrollida, kas tudeng tunneb ära loendi ning sellega seotud meetodi.

```
List<int> numList = new List<int> {43, 25, 78};
numList.Sort();
foreach (int i in numList)
{
    Console.Write(i);
}
```

Joonis 16. Loendi kompetentsi ülesande näide

Antud ülesandes on deklareeritud täisarvude loend. Loend on sorteeritud elementide kasvavas järjekorras. Seejärel käiakse loendi elemendid *foreach* tsüklis läbi ning elemendid kuvatakse üksteise järel.

3.2 Sisendit koguvad ülesanded

Teist tüüpi ülesanded asuvad klassis 1607. Need on ülesanded, milles palutakse tudengil kuni 10 sõnaga kirjeldada, mida antud koodilõik teeb. Koodilõiguks on mingi funktsioon või meetod. Koodilõigus on kombineeritud mitmeid kompetentse. Antud ülesannete eesmärk on koguda tudengite poolseid vastuseid ehk sisendit, et neist hiljem genereerida uued ülesanded, millel on valikvastused. Sisendit koguva ülesande pikkus on üldjuhul suurem kui staatilisel ülesandel. Peamised kompetentsid, mida on sisendit koguvate ülesannete disainimisel kasutatud, on tingimuslaused, matemaatilised tehted, tsüklid (*for, foreach, while*) ja massiivid. Järgnevalt on esitatud mõned näited, millega on kogutud tudengite poolset sisendit.

```
public static bool Prime(int n)
{
    for (int i = 2; i < n; i++)
    {
        if (n % i == 0)
        {
            return false;
        }
    }
    return true;
}
```

Joonis 17. Algarvu kontrolli ülesande näide

Antud ülesande peamisteks kontrollitavateks kompetentsideks on *for* tsükkel, *if* tingimuslause ja moodularvutus. Meetodis olevas *for* tsüklis kontrollitakse, kas parameetri n jagamisel täisarvuga i tekib jääk või mitte. Kui ei teki, siis tagastatakse tõeväärtus *false*, vastasel juhul on tulemuseks *true*.

```
public static int Count(string text)
{
    int count = 0;
    string letter;
    for (int i = 0; i < text.Length; i++)
    {
        letter = text.Substring(i, 1);
        if (letter == " ")
            count++;
    }
    return count;
}
```

Joonis 18. Tühikute lugemise ülesande näide

Antud ülesandes loetakse parameetris *text* leiduvad tühikud kokku. Esmalt deklareeritakse täisarv *count* ja string *letter*. Seejärel käiakse kontrollitakse *for* tsükli sees olevas *if* tingimuslauses, kas stringis leiduv element on tühik. Kui tingimus vastab tõele, suureneb täisarvu *count* väärtus 1 võrra. Peale tsükli lõppu tagastatakse täisarvu *count* väärtus.

```
public static bool IsPal(string text)
{
    text = text.ToUpper();
    int j = text.Length - 1;
    for (int i = 0; i < j; i++)
    {
        if (text[i] != text[j])
        {
            return false;
        }
        j--;
    }
    return true;
}
```

Joonis 19. Palindroomi kontrolli ülesande näide

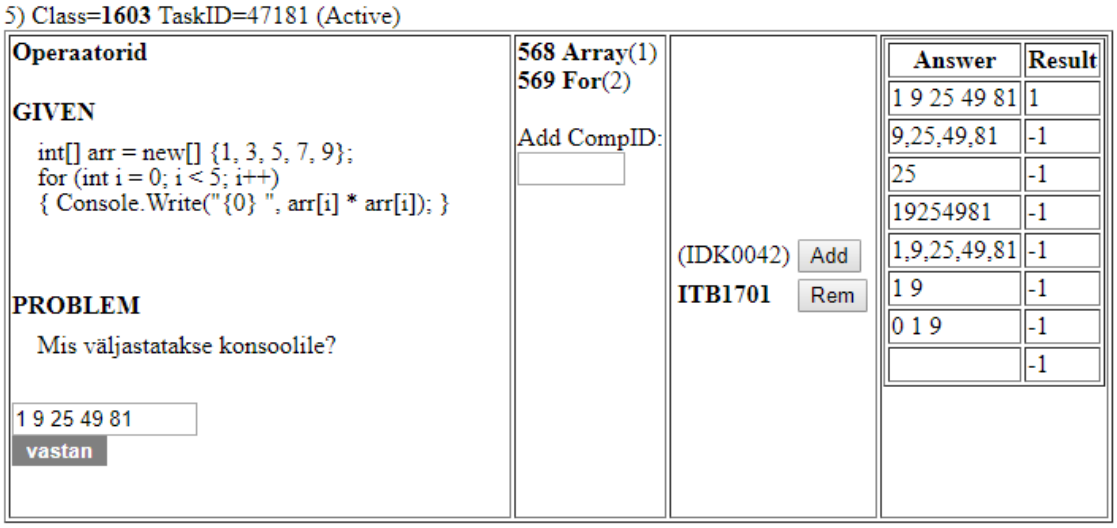
Antud ülesandes kontrollitakse, kas meetodi parameeter *text* on palindroom. Selleks muudetakse parameetris kõik tähemärgid suurtähtedeks ja deklareeritakse täisarv j, mille väärtuseks stringi pikkus, millest on lahutatud 1. *for* tsüklis kontrollitakse *if* tingimuslause sees, kas stringi elemendid kohal i ja j klapivad. Kui need ei ühti, tagastatakse tõeväärtus *false.* Tsükli jooksul vähendatakse muutuja j väärtust 1 võrra. Kui stringi elemendid ühtivad omavahel, siis tagastatakse tõeväärtus *true.*

4 Eksperimendi analüüs

4.1 Valideerimise tulemused

Eksperimendi läbiviimiseks äriinfotehnoloogia eriala esimese kursuse tudengite seas koostati kontrolltöö. Antud test avati tudengite jaoks 1. veebruaril 2018. Kontrolltööd oli võimalik sooritada tudengi poolt vabalt valitud ajal ja kohas. ISC õppekeskkonna parema kasutamise eesmärgi nimel koostati tudengitele juhend, kuidas süsteemis toimida, ainet deklareerida ning kuidas *MyField* leheküljel ülesandeid lahendada.

Ülesannete lahenduste kuvamiseks on ISC õppekeskkonnas olemas *QUIZScanQuery* lehekülg. Antud leheküljel on võimalik kuvada ning sorteerida ülesandeid ja selle juurde kuuluvaid lahendusi ülesannete klassi, kompetentsi või mõne muu numbrilise identifikaatori alusel. *QUIZScanQuery* leheküljel on igale kuvatud ülesandele võimalik ka kompetentse identifikaatori abil juurde lisada ja korrigeerida. Samuti on võimalik ülesannet aktiivseks muuta ning ainesse lisada. Kui ülesanne pole aktiivne, siis on ülesande identifikaatori kõrval kirjas olekuna *Task is not active* ja nupp *Activate*. Ainesse lisamist ja ainest eemaldamist saab teha vastavalt *Add* ja *Rem* nuppude abil. Kõige parempoolsemas sektsioonis on kuvatud vastused. *Result* tulbas on väärtusega 1 kuvatud õige lahendus ning kõik read, millel on väärtus -1, on valed lahendused.



Joonis 20. *QUIZScanQuery* leheküljel ülesande ja lahenduse kuvamine

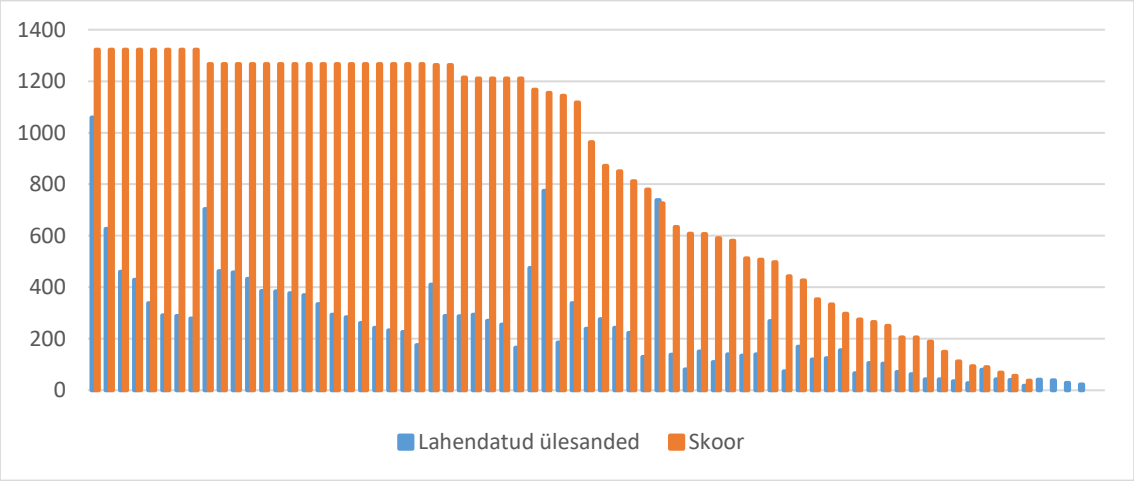
Järgnevalt on välja toodud tudengite eksperimendis saavutatud tulemused. Maksimaalne võimalik skoor oli 1323. Keskmine skoor kõikide tudengite peale oli 801. Tudengite saavutatud skoorid on jagatud kolme vahemikku. Kõige rohkem oli neid tudengeid, kelle skoor oli väga hea ehk üle 1000. Eksperiment näitab, et mida suuremat skoori tahab tudeng saavutada, seda rohkem ülesandeid ta ka lahendab. See tähendab omakorda, et vajalik on saavutada kõikides kompetentsides tulemusi. Mida suurem on tudengi saavutatud skoor, seda rohkem leidub ta positiivseid tulemusi. Selgub, et ka negatiivsete tulemuste arv suureneb sel juhul, kuid mitte nii suurel määral kui positiivsete tulemuste arv. Positiivne tulemus tähendab ülesandele antud õiget vastust ning negatiivne tulemus valet vastust. Selleks, et saavutada väga head tulemust, on vaja lahendada võimalikult palju ülesandeid, saavutada tulemusi nii paljudes kompetentsides kui võimalik ja hoida positiivsete tulemuste arvu suuremana kui negatiivsete tulemuste arvu.

Tabel 2. Tudengite tulemused eksperimendis

Vahemik	**Tudengite arv selles vahemikus**	**Keskmine lahendatud ülesannete arv**	**Keskmine tulemustega kompetentside arv**	**Keskmine positiivsete tulemuste arv**	**Keskmine negatiivsete tulemuste arv**
0 - 500	24	75	22	80	45
501 - 1000	13	211	30	248	94
1000 - 1323	35	375	32	482	158

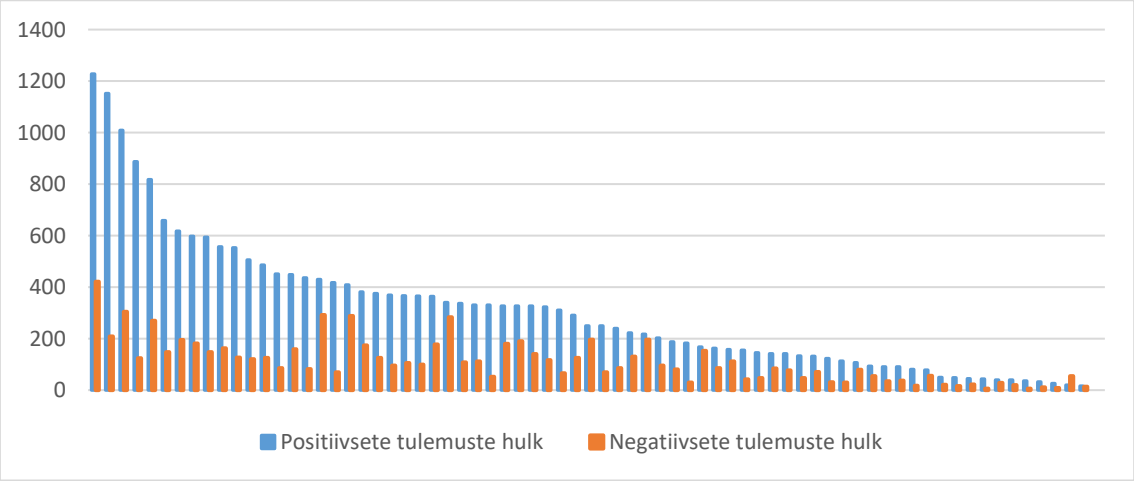
Eksperimendist selgub, et on kaht tüüpi tudengeid. Ühe tüübi moodustavad need, kes soovivad maksimaalse tulemuse saavutamiseks võimalikult vähe vaeva näha. Teist tüüpi tudengid on need, kes lahendavad väga palju ülesandeid, aga nende tulemus ei pruugi alati olla maksimaalne, kuid on näha, et nad on vaeva näinud ja panustanud õppetöösse ainele vastavas mahus. Umbes pooled kontrolltööst osa võtnud tudengitest saavutasid maksimaalse või sellele lähedase tulemuse. Samas nende poolt lahendatud ülesannete hulk on kõikuv. Eksperiment peegeldab ka kompetentsipõhise õppe olemust. Igal tudengil on oma tempo ja vajalike oskuste omandamiseks kulubki igaühel aega erinevalt [6].

Tabel 3. Tudengite skoorid ja lahendatud ülesannete hulk



Eksperiment näitab, et tudengite tase on erinev. Kontrolltöö sooritanud tudengitest leidub väheseid, kelle puhul õigete vastuste ning positiivse skooriga kompetentside hulk on märgatavalt suurem kui negatiivse skooriga kompetentside hulk. Selgub, et üldiselt jääb tudengil positiivse skooriga kompetentside hulk suuremaks. See võib tähendada, et tudeng ei karda väga palju eksida ja vastab ülesandele enamasti õigesti.

Tabel 4. Tudengite positiivsed ja negatiivsed tulemused



Eksperimendist selgub, et möödunud semestril õpitud programmeerimise alused on tudengitel enamasti selged, kuid siiski tehakse mitmeid vigu. Peamisteks vigadeks on hooletusvead. Kuna disainitud materjalides oli sarnase ülesehitusega ülesandeid palju, siis pikaajalise lahendamise tõttu võis tudengitel tekkida vigu tähelepanematusest.

Tudengid tunnevad üpris hästi *for* tsükli struktuuri ja selle olemust. Kuna just *for* tsükliga oli varieeritud mitmeid ülesandeid, siis läbi selle oli näha, et andmete muutusest

hoolimata vastatakse ülesandele enamasti ikkagi õigesti. Samas teevad tudengid vigu *for* tsükli osas siis, kui ülesandes trükitakse muutuja i väärtus välja. Tudengid arvavad, et trükitakse välja i kui tähtväärtus, kuid tegelikkuses on tegu muutujaga, mille väärtus on välja trükitud.

Samuti tunnevad tudengid hästi *while* tsüklit. Väga hästi saadakse aru, milline on tsüklis sisalduv tingimus ning mida konsoolile välja trükitakse. Kui ülesandes on deklareeritud mitu muutujat ja ühe muutuja väärtust küsitakse, siis tudeng satub segadusse ja annab rohkem valesid vastuseid. Lisaks pole tudengil selge, mida teeb *do-while* tsüklis *do* osa. Vastustest selgub, et tudengid vaatavad sellest mööda ja keskenduvad ainult tsükli *while* tingimusele.

Massiividega seotud ülesannetel leidus kõige rohkem erinevaid valesid vastuseid. Kõige enam tuli ette hooletusest ja tähelepanematusest tulenevaid vigu. Kui massiivi elemente trükiti välja kas tühiku, alakriipsuga või kokku kirjutatuna, siis tudeng ilmselt ei pannud piisavalt tähele, kuidas ülesandes elementide trükkimist nõuti. Sarnane probleem ilmnes *for* tsükli juures muutuja i trükkimise osas. Teine suurem viga, mis just massiividega ilmnes, on elementide nullipõhine indekseerimine. Näiteks tudeng arvab, et massiivi element indeksiga 1 on esimene element, kuigi tegu on hoopis järjekorras teise elemendiga. Samas massiividega seotud meetodid on tudengitel selged ja arusaamine väga hea.

Matemaatiliste tehete sooritamine on tudengitel üsna selge. Tuntakse elementaarseid matemaatika reegleid, sealhulgas ka tehete järjekorda. Paraku on matemaatiliste tehete puhul problemaatiliseks kohaks moodularvutus. Kui moodularvutust eraldi kompetentsina kontrollida, siis valdavalt vastatakse õigesti. Kompetentsi integreerimisel muudesse ülesannetesse (näiteks *for* tsükliga seotud ülesannetesse) kasvab erinevate vigade hulk märgatavalt.

Tekstitöötlus on kompetents, millega seotud ülesannetel oli üpris vähe vigu. Stringide töötlemist oskavad tudengid väga hästi. Stringidega seotud meetodeid osatakse rakendada. Samas oli tekstide töötlemisel üks konkreetne viga, mida tudengid tegid. Näiteks on olukord, kus konsoolile trükitakse täisarvuline muutuja i ja liidetakse juurde kahe ülakoma või jutumärkide vahel olev tühik. Kui tühik on ülakomade vahel, siis seda loetakse tähemärgiks. Igal tähemärgil on oma ASCII kümnendväärtus. Tühikul on selleks

32 [7]. Tudengid kippusid arvama, et pole vahet, kas tühik on ülakomade või jutumärkide vahel.

Tingimuslauseid tunnevad tudengid väga hästi. Kõikide tingimuslausetega seotud ülesannete puhul leidusid vaid üksikud vead. Ülesannete valed lahendused olid seotud *switch* tüüpi tingimuslausetega. Põhjus on ilmselt selles, et seda kasutatakse vähem kui *if* tingimuslauset, mistõttu tunnevad tudengid antud kompetentsi veidi vähem.

Andmetüüpe oskavad tudengid kasutada, aga nende väärtuste vahemikke ei tunta. Kõige problemaatilisem asjaolu andmetüüpide väärtuste vahemike kompetentsi kontrollimisel oli olukord, kus andmetüübi maksimaalsele väärtusele liideti 1 juurde. Kuigi täisarv ehk *integer* on disainitud ülesannetes kõige tihedamalt kasutatud andmetüüp, siis nii selle kui ka teiste andmetüüpide väärtuste vahemikud on tudengitel teadmata või aetakse sassi.

Kõik disainitud ülesanded koostati meelega nii, et nendes ei leiduks mitte ühtegi veaolukorda või tekiks kompileerimise viga. Sellest hoolimata arvasid tudengid mitmele ülesandele vastates, et ülesande sisus on viga ning tänu sellele esineb veaolukord.

Sisendit koguvatele ülesannetele vastati valdavalt õigesti. Tegu oli ülesannetega, kus ei antud peale vastuse sisestamist koheselt tagasisidet. Vastuse analüüsi käigus selgus, et tudengid saavad meetodi või funktsiooni toimimise põhimõttest väga hästi aru, aga sõnastavad vastuse erinevalt. See ei muuda vastuse õigsust.

4.2 Tudengite tagasiside

Kontrolltöö sooritamise järel oli tudengitel võimalus anda omapoolset tagasisidet kontrolltöö olemuse ning läbiviimise kohta. Selle tarbeks oli koostatud ained.ttu.ee õppekeskkonnas tagasisideküsimustik. Äriinfotehnoloogia esimese kursuse tudengitest oli vastajaid 15. Küsimustiku eesmärk oli välja uurida, mis on tudengite arvates ISC õppekeskkonna positiivsed ja negatiivsed aspektid ning kas nad omandasid kontrolltöö sooritamise käigus uusi teadmisi ja oskusi.

Tudengid tõid positiivsena välja selle, et kontrolltöö tegemine oli huvitav, sest küsimustele oli võimalik vastata lõpmatu hulk kordi. Tänu sellele, et küsimustele sai vastata nii palju kui sooviti. Õpiti tehtud vigadest nii kaua, kuni teema selgeks sai. Lisaks toodi välja, et ISC õppekeskkonnal on üsna lihtne ja minimalistik välimus, mida on hea

kasutada. Kasutajaliidese puhul kiideti kompetentside värvide süsteemi. See aitas tudengitel ära eristada, milliste oskuste omandamisele on vaja rohkem aega pühendada.

Kuna tänu mälumudelile ei olnud skoor püsiv, siis motiveeris see tudengeid rohkem ülesandeid lahendama ning nendele oma aega panustama. Tudengite poolt kiideti ülesannete varieeruvust. Ülesanne võis tunduda neile sarnasena, kuid oluline oli jälgida, millised on ülesandes sisalduvad andmed. Positiivse aspektina toodi välja ka veel, et tagasiside vastuse osas on kohene. Lisaks meeldis tudengitele võimalus õppejõult küsida lahendatud ülesande lahenduse kohta. Kui tudeng vastas valesti ja tal tekkis küsimus, mille tõttu tema vastus valeks loeti, siis oli võimalus saata küsimus. Tänu sellele funktsionaalsusele paranes tudengi skoor ja vähenes vigade arv.

Tudengite tagasiside kohaselt on ISC õppekeskkonnal juures palju tehnilisi vigu. Süsteem on üpris vana ja võiks saada uue ja kaasaegsema kasutajaliidese. Tihti esines ka süsteemi sisselogimisel probleeme, mistõttu oli vahepeal kontrolltöö sooritamine häiritud. Lisaks toodi välja, et kui ülesandele vastati valesti, siis võiks olla ka tagasiside selles osas, et mida ülesande lahendamisel valesti tehti. Igale kompetentsile oleks võinud leiduda ka viidatud materjal, mida oleks saanud võtta abiks, kui ülesande lahendamisel probleeme tekib.

Tudengid tõid välja, et oluline on oma koodis detailselt üle kontrollida iga rida, sest isegi pisike viga andis veateate. Möödunud semestril õpitut on hakatud unustama, mistõttu teatud kompetentside ülesannete lahendamisel tekkis rohkem vigu. Osati oli kontrolltöö rohkem kordamine, kuid leidus ka tudengeid, kes väitsid, et nende oskused paranesid võrreldes möödunud semestriga. Lisaks arvasid tudengid, et ISC õppekeskkond on aidanud neil loogilist mõtlemist arendada.

Järgnevalt on välja toodud kokkuvõtlik tabel, milles sisaldub tudengite poolt antud positiivne ja negatiivne tagasiside ISC õppekeskkonna kohta.

Tabel 5. Tudengite poolne tagasiside

Positiivne tagasiside	**Negatiivne tagasiside**
▪ Huvitav ▪ Ülesandeid võimalik lõpmatu hulk kordi lahendada ▪ Vigadest õppimine ▪ Lihtne välimus ▪ Kompetentside värvide süsteem ▪ Mälumudel motiveeris panustama ▪ Kohene tagasiside	▪ Palju tehnilisi vigu ▪ Süsteem vana, aegunud kasutajaliides ▪ Valesti vastamisel täiendav tagasiside vajalik ▪ Kompetentsidele lisamaterjal vajalik

4.3 Analüüs ja soovitused tulevikuks

ISC õppekeskkonnal on mitmeid positiivseid omadusi. Antud keskkond kasutab kompetentsipõhist õpet ehk kõik kontrollitavad oskused on lahti võetud väikesteks tükkideks. Tänu sellele on hea kontrollida tudengi teadmisi ka kõige detailsematest oskustest. Tudengite tagasiside kohaselt kinnistab kompetentsipõhine õppekeskkond nende teadmisi väga hästi.

Antud õppekeskkonnas õppimine on tudengeid väga palju motiveerinud. Tudengite motivatsioon püsib kõrgena tänu sellele, et ülesannetele vastamise katsete arv pole piiratud. Kuna süsteemis olev mälumudel ei hoia tudengi poolt saavutatud skoori stabiilsena, siis see motiveerib samuti pingutama maksimaalse skoori nimel ja rohkem ülesannetele vastama. Lisaks on hea prognoosida oma õppimist ette tänu nuppudele *NOW* ja *W16*. See näitab kõige paremini mälumudeli toimimist ning aitab seada fookust kompetentside omandamisele, mille skoor on veel madal.

ISC õppekeskkonda on kasutatud enamasti sidude, süsteemide ja signaalide aine tarbeks. Paraku on antud õppekeskkonnal ka mitmeid puudusi. Süsteem on vananenud ning kasutajaliides on üsna algelise ja minimalistliku disainiga. Tänu sellele on kogu õppekeskkond kohmakas ja väga tihti esineb tehnilisi vigu. Mitmed vead esinevad ülesannete lahendamisel. Näiteks oli mitmeid disainitud ülesandeid, kus tsüklis trükiti välja muutuja väärtus ning sellele järgnes tühik. Kuna ka viimasele elemendile lisandus tühik, siis sisestatud vastuses süsteem enam viimast tühikut arvesse ei võtnud. Lisaks

esines ka serverisiseseid vigu. Kõikide nende vigade parandamine nõudis pidevat hooldamist ning süsteemi ümber programmeerimist.

Kohati ebamugav oli ka ülesannete lisamine (Joonis 4) ja reahaaval korrigeerimine (Joonis 5). Selle tõttu oli disainitud materjalide õppekeskkonda lisamine üsna ajamahukas tegevus. Kindlasti on olemas paremaid ja mugavamaid viise, kuidas süsteemi ülesandeid lisada. Ülesannete lisamise formaat peaks olema lihtsam ja vähemate sammudega. Antud süsteemis tuli ülesanne lisamisel saata puhverserverisse ja seal seda korrigeerida. Peale selle ajas süsteem tihti sassi kompetentsi ja ülesande rea identifikaatorid, mistõttu lisandus veelgi käsitsi tööd, et ülesanne oleks formaadilt korrektsete andmetega.

Õppekeskkonna jaoks materjalide disainimine on raske ning ajamahukas tegevus. Kompetentsipõhise õppe puhul on keeruline igale eraldiseisvale kompetentsile ülesannet välja mõelda. Lihtsam on rakendada kompetentse koos ja neid omavahel kombineerida. Kõige kergem on mõelda välja mingi meetod, mis teeb ainult üht asja ja selle abil tudengite poolset sisendit koguda ning selle põhjal valikvastused genereerida. Lisaks on raske disainida materjale, mis sobiksid kõigile tudengitele, sest igaühe taust ning õppimisele eelnevate oskuste hulk on erinev. Üks olulisemaid ülesannete variantide tegemise viise ongi just tudengite poolsed sisendtekstid. Ülesannete variante saab suhteliselt lihtsalt genereerida ja tänu sellele puudub igasugune käsitsi töö ning ülesannete varieerimine on automatiseeritud. Sellisel viisil on võimalik väga suurel hulgal luua sarnaseid ülesandeid.

Antud eksperimendi jaoks oli sisendit koguvaid ülesandeid lihtsam disainida kui eraldiseisvate kompetentside staatilisi ülesandeid. Kuna sisendit koguv ülesanne koosneb meetodist, siis see annab võimaluse varieerida ja kombineerida erinevaid kompetentse. Staatiliste ülesannete puhul on kontrollitav oskus läbiv, muutuvad vaid muutujate väärtused või küsitakse mõne muu muutuja väärtust. Sisendit koguvate ülesannete puhul disainitavad ülesandeid peaksid sisaldama endas üht meetodit, mis teeb ainult üht asja. Tänu sellele on võimalik vältida mitmeti mõistetavust ja arusaamatusi. Sellised disainitud ülesanded on heaks näiteks puhtast koodist ja eeskujuks tudengitele, kuidas tulevikus programmeerida. Ülesande keerukus ei tohiks olla liiga suur, kuid ülesandeid lahendades võiks see ühtlaselt kasvada. Ülesanne peaks sisaldama ainult mõnda üksikut kompetentsi. Kui ülesandega on seotud liiga palju kompetentse, siis muutub ka vastuste töötlus keerulisemaks. Tulevikus on mõistlik ülesandeid disainida ka vastupidi. Tudengile võib

ette anda ülesande, milles on osa koodi eemaldatud ning tuleb täita lüngad. Lisaks on koodile olemas kirjeldus, mida antud koodilõik tegema peab. Sellised ülesanded arendaksid tudengi analüütilist mõtlemist.

Antud õppekeskkonnas olid nii *QUIZScan* (Joonis 3) kui ka *QUIZScanQuery* (Joonis 20) leheküljed eksperimentaalsed ning igapäevatöös neid ei ole kasutatud. Tänu nendele lehekülgedele oli võimalik lisatud ülesandeid paremini korrigeerida. Lisaks võimaldas *QUIZScanQuery* lehekülg tudengite poolt antud lahendusi hästi analüüsida. Antud töös kirjeldatud disainitud materjalide töötlus oli üsna primitiivne. Vastused loeti kas täielikult valeks või õigeks. Reeglina on ISC õppekeskkonnas ülesannete vastuste analüüs palju sügavam. Vastust analüüsitakse väga väikeste detailideni välja, sest tudeng ei pruugi täielikult valesti vastata. Võib ainult eeldada, et ta võis midagi sassi ajada või teisiti mõista. Seetõttu on kompetentsipõhisel õppel vastuste analüüsimine äärmiselt keeruline tegevus.

Antud keskkond pole oma tehniliste puuduste tõttu piisavalt sobilik vahend programmeerimisainete õpetamisel. Programmeerimisega seotud ülesandeid on seal võimalik küll hoida ja hallata, kuid õpetamise kvaliteet pole selleks piisav. Ülesannete vastuste töötlemine jääb liialt primitiivseks. Lisaks on tagasiside vastuse sisestamisel veidi puudulik. Kui ülesandele vastatakse valesti, siis tuleks tudengile anda vihje, mida tulevikus vältida, kui lahendada sarnaseid ülesandeid.

Järgnevalt on välja toodud kokkuvõtlik tabel ISC õppekeskkonna positiivsetest ja negatiivsetest omadustest.

Tabel 6. ISC õppekeskkonna positiivsed ja negatiivsed omadused

Positiivsed omadused	**Negatiivsed omadused**
▪ Kompetentsipõhine õpe ▪ Mälumudeli olemasolu ▪ Kinnistab teadmisi ▪ Annab motivatsiooni ▪ Lõpmatu vastamise katsete arv	▪ Vananenud kasutajaliides ▪ Palju tehnilisi vigu ▪ Serverisisesed vead ▪ Pidev hooldamine ▪ Ebamugav ülesannete lisamine ja korrigeerimine

ISC õppekeskkonnal on olemas ka uuem versioon, mis asub aadressil http://isc.ttu.ee/ [2]. Tulevikus tasub teha analüüs sellele õppekeskkonnale, et uurida, kas uuema disainiga süsteem on programmeerimise õpetamisel kasulikum. Käesolevas töös analüüsitud õppekeskkonnast uut keskkonda luua pole otstarbekas, kuna antud süsteemi arhitektuur on üpris keeruline. Kui hakata õppekeskkonda algusest peale looma, siis on vaja teha olemasolevale süsteemile põhjalik äri- ja süsteemianalüüs. Oluline on kaardistada ära süsteemi vajalik funktsionaalsus ning kõik ebavajalik eemaldada. Kuna süsteemi andmebaas on keeruka arhitektuuriga, siis tuleks see uuesti projekteerida. Kui hakata uut keskkonda looma, siis tuleks see ehitada mõne objekt-orienteeritud programmeerimiskeele abil, näiteks C# või Java. Kuna olemasoleva süsteemi tehnilisi vigu on vaja pidevalt käsitsi parandada, siis tuleks kirjutada süsteemile uus dokumentatsioon, kus on detailselt ära kirjeldatud süsteemi funktsionaalsus.

ISC õppekeskkonnast on mõistlik teha raamistik, kuhu on võimalik paigutada mistahes tüüpi ülesandeid. Oluline on, et raamistikus säiliks kompetentsipõhine õpe. Tähtis on ära kaardistada valdkonnad, mis sobivad kompetentsipõhise õppega kokku. Teine oluline aspekt, mida tuleks alles hoida, on mälumudel. Tudeng unustab lühikese aja jooksul õpitud materjali üsna kiiresti. Et tulemust maksimaalsena hoida, on vajalik pidev ja järjepidev töö. Raamistikku üles ehitades on esmalt vajalik teha põhjalik äri- ja süsteemianalüüs, millesse on soovitatav kaasata tudengeid, sest nemad on õppekeskkonna sihtgrupiks. Peale analüüsi tuleb projekteerida andmebaas, mille süsteemi valik tuleb teha vastavalt arendustehnoloogiale. Analüüsi vajab ka kompetentsipõhisel õppel ülesannete vastuste tagasisidestamine. Kompetentsipõhisel õppel tuleb vastuseid analüüsida sügavuti, et välja uurida, kui täpse vastuse tudeng ülesandele andis. Primitiivne vastuste töötlus ei loo selget arusaama sellest, kas tudeng on kompetentsi omandanud või mitte. Lisaks on oluline analüüsida mälumudelit ning selle olemust ja kuidas seda välja arendada loodavale raamistikule. Igale valdkonnale, mis hakkab loodavat raamistikku kasutama, on vajalik koostada nimekiri kompetentsidest, mida raamistikus kontrollima hakatakse.

Järgnevalt on välja toodud kokkuvõtlik tabel soovitustest, mida ette võtta ISC õppekeskkonnaga. Soovitustena on lisatud uue õppekeskkonna arendamine ning raamistiku loomine. Mõlemale soovitusele on juurde toodud vastavad tegevused, mida läbi viia.

Tabel 7. Soovitused ISC õppekeskkonna osas

Soovitus	**Tegevused**
Uue õppekeskkonna arendamine	▪ Analüüsida ISC õppekeskkonna uuemat versiooni ▪ Ärianalüüs ▪ Süsteemianalüüs ▪ Ebavajalik funktsionaalsus eemaldada ▪ Andmebaasi projekteerimine ▪ Arendustehnoloogia valik
Raamistiku loomine	▪ Sobivate valdkondade kaardistamine ▪ Kompetentsipõhise õppe analüüs ▪ Mälumudeli analüüs ▪ Ärianalüüs ▪ Tudengite kaasamine arendusprotsessi ▪ Süsteemianalüüs ▪ Andmebaasi projekteerimine ▪ Arendustehnoloogia valik

ISC õppekeskkonna olemasoleva süsteemi analüüsiks on mitmeid arengukohti. Kui alustada analüüsiga nüüd, siis esmalt tuleks koostada äärmiselt detailne kompetentside nimekiri, mida kontrollida. Käesolevas töös on käsitletud vaid peamisi teemasid programmeerimises. Näiteks ei piisa ainuüksi *for* tsükli või massiivi kompetentsi kontrollimisest. Nendes kompetentsides ja teemades leidub detaile, mida saab käsitleda eraldiseisva kompetentsina.

Kui hakata koostatud kompetentsidele ülesandeid disainima, siis tuleks suuremat rõhku panna ülesannetele, milles sisaldub üks meetod, mis teeb ainult üht asja. Sellised ülesanded on headeks näideteks puhtast koodist. Kui tudeng alustab kontrolltööga, siis esmalt tuleks meetodit sisaldavate ülesannetega küsida tema sisendit. Saadud sisenditest

tuleks genereerida valikvastused. Saadud valikvastuste abil on võimalik tudengile näidata, kuidas on vastanud ülesandele kaastudengid ning milline on tegelik õige vastus. Meetodit sisaldavaid ülesandeid võiks disainida ka vastupidi. Tudengile antakse ette lünkadega koodilõik ning kirjeldus, mida antud koodilõik tegema peab. Tudengi ülesandeks on täita lüngad nii, et koodilõigu funktsionaalsus vastaks ette antud kirjeldusele. Kindlasti ei tohiks ära unustada ka staatilisi ülesandeid eraldiseisvatele kompetentsidele. Disainitud ülesandeid tulekski sel juhul nelja tüüpi: sisendit koguvad ülesanded, valikvastustega ülesanded, lünkade täitmisega seotud ülesanded ning staatilised ülesanded.

Olemasoleva õppekeskkonna analüüsi uuesti alustades võiks ülesannete vastuste töötlus olla rohkem sügavam. Käesolevas töös olnud eksperimendis loeti ülesannaete vastuseid täielikult õigeks või täielikult valeks. Enne vastuste töötlemist tuleks ära kaardistada iga kompetentsi tüüpvead ning vastavalt sellele hinnata tudengi poolt antud vastust. Olukordi, kus tudeng ülesandele täielikult valesti vastab, juhtub pigem harva ja seetõttu on vastuste põhjalikum töötlemine äärmiselt vajalik. Läbi sügavama vastuste töötlemise on võimalik tudengi teadmisi ja oskusi vastavate kompetentside kohta adekvaatsemalt hinnata.

5 Kokkuvõte

Käesoleva lõputöö eesmärgiks oli analüüsida ISC õppekeskkonna sobivust programmeerimisainete õpetamisel. Kuna tudengitel pole palju kokkupuudet kompetentsipõhiste õppekeskkondadega, siis läbi äriinfotehnoloogia esimese kursuse tudengite seas läbi viidud eksperimendi tulemuste oli eesmärk välja uurida, kas on mõistlik sellistesse õppekeskkondade arendusse panustada.

Iga disainitud ülesandega lahendamisel kontrolliti ühe või mitme kompetentsi omandamist. Kõige tähtsam kompetents, mille omandamist käesolevas töös kontrolliti, oli programmeerimiskeele C# fundamentaalsete teadmiste omamise oskus.

Disainitud materjale oli kaht tüüpi. Esimese tüübi moodustasid staatilised ülesanded, millel on kindel küsimus ja üks konkreetne vastus. Antud asjaolu välistas mitme paralleelse õige vastuse olemasolu. Teist tüüpi ülesannetega paluti tudengil kirjeldada oma sõnadega, mida antud koodilõik teeb. Sellega koguti tudengi poolt sisendit, et saadud vastustest genereerida uusi valikvastustega ülesandeid.

Eksperimendi käigus selgus, et ISC õppekeskkond pole piisavalt sobilik programmeerimisainete õpetamisel. Mõistlik on teha antud õppekeskkonnast raamistik, kuhu on võimalik mistahes ülesandeid lisada. Oluline on raamistikus säilitada kompetentsipõhine õpe.

Kasutatud kirjandus

[1] Tallinna Tehnikaülikool, „Aine koduleht: ÕIS,“ 2017. [Võrgumaterjal]. Saadaval: http://ois.ttu.ee/aine/ITB1701. [Kasutatud 2. aprill 2018].

[2] Tallinna Tehnikaülikool, „ISC,“ 2017. [Võrgumaterjal]. Saadaval: http://isc.ttu.ee/et/userGuide. [Kasutatud 2. aprill 2018].

[3] V. Kukk, K. Umbleja ja M. Jaanus, „Two-dimensional knowledge model for learning control and competence mapping,“ 2015.

[4] Microsoft, „Learn C# for Beginners - Microsoft Virtual Academy,“ 2016. [Võrgumaterjal]. Available: https://mva.microsoft.com/en-US/training-courses/c-fundamentals-for-absolute-beginners-16169?l=vE6GqMQIC_506218949. [Kasutatud 6. aprill 2018].

[5] B. Meyer, Touch of Class: Learning to Program Well with Objects and Contracts, Springer Science & Business Media, 2009.

[6] V. Kukk, „Course Implementation: Value-Added Mode,“ 2016.

[7] „ASCII Table,“ 2010. [Võrgumaterjal]. Saadaval: https://www.asciitable.com/. [Kasutatud 6. aprill 2018].

Lisa 1 – Staatilised ülesanded

```
for (int i = 0; i < 10; i++)
{
    Console.Write("{0} ", i);
}
```

```
string str1 = "isc.ttu.ee";
string str2 = "ttu";
bool g = str1.Contains(str2);
```

```
string str = "Visual Studio";
int k = 0;
while (k < str.Length)
{
    Console.Write("{0}_", str[k]);
    k++;
}
```

```
string str = "Visual Studio";
int k = str.Length - 1;
while (k >= 0)
{
    Console.Write("{0} ", str[k]);
    k--;
}
```

```
int[] arr = new[] {1, 3, 5, 7, 9};
for (int i = 0; i < 5; i++)
{
    Console.Write("{0} ", arr[i] * arr[i]);
}
```

```
int[] arr = {2, 4, 6, 8, 10};
for (int i = 4; i >= 0; i--)
{
    Console.Write("{0} ", arr[i]);
}
```

```
int[] numbers = new[] {3, 9, 7, 8};
int g = numbers.Sum();


int[] numbers = new[] {3, 9, 7, 8};
int c = numbers.Length;


int[] data = {1, 2, 3};
int i = 1;
data[i++] = data[i] + 10;
Console.WriteLine(String.Join(",", data));


int[] numbrid = new int[] {10, 23, 33, 36, 45};
Console.WriteLine(numbrid[3])


for (byte b = Byte.MinValue; b < Byte.MaxValue; b++)
{
    if (b == 0x90)
    {
        Console.WriteLine("Welcome to Visual Studio");
    }
}


for (int i = 0; i < 10; i++)
{
    Console.Write(i + ' ');
}


bool b1 = true, b2 = false;
if ((b2 = true) | (b1 ^ b2))
{
    Console.WriteLine("success");
}


for (int i = 1; i < 3; i++)
{
    Console.Write("{0} ", i);
}


int tulemus = 67;
if (tulemus >= 91 && tulemus <= 100)
    Console.Write("Hinne on 5");
else if (tulemus >= 81 && tulemus <= 90)
    Console.Write("Hinne on 4");
```

```
else if (tulemus >= 71 && tulemus <= 80)
    Console.Write("Hinne on 3");
else if (tulemus >= 61 && tulemus <= 70)
    Console.Write("Hinne on 2");
else if (tulemus >= 51 && tulemus <= 60)
    Console.Write("Hinne on 1");
else if (tulemus < 51)
    Console.Write("Mittesooritatud");


int aasta = 2017;
if ((aasta % 4 == 0 && aasta % 100 != 0) || aasta % 400
== 0)
{
    Console.WriteLine("{0} on liigaasta", aasta);
}
else
{
    Console.WriteLine("{0} pole liigaasta", aasta);
}


string str = "Tere hommikust! Ilusat päeva jätku";
int i, k;
k = 0;
i = 1;
while (k <= str.Length - 1)
{
    if (str[k] == ' ')
    {
        i++;
    }
    k++;
}


int[] n = new int[] {10, 23, 33, 36, 45};
Console.WriteLine(n[3]);


int[] arr = new[] {2, 4, 6, 8, 10};
for (int i = 4; i >= 0; i--)
{
    Console.Write("{0} ", arr[i]);
}


int[] numbers = new[] {3, 9, 7, 8};
int c = numbers.Count();
```

```
int[] numbers = new[] {3, 9, 7, 8};
int c = numbers.Rank;
```

```
int[] numbers = new[] {3, 9, 7, 8};
bool c = numbers.IsFixedSize;
```

```
int[] numbers = new[] {3, 9, 7, 8};
bool c = numbers.IsReadOnly;
```

```
int[] numbers = new[] {3, 9, 7, 8};
bool c = numbers.IsSynchronized;
```

```
int[] arr = {2, 4, 6, 8, 10};
for (int i = 3; i >= 0; i--)
{
    Console.Write("{0}", arr[i]);
}
```

```
int[] arr = {2, 4, 6, 8, 10};
for (int i = 3; i < >= 0; i++)
{
    Console.Write("{0} ", arr[i]);
}
```

```
int[] arr = new[] {1, 3, 5, 7, 9};
for (int i = 1; i < 5; i++)
{
    Console.Write("{0} ", arr[i] * arr[i]);
}
```

```
int[] numbers = new[] {3, 9, 7, 8};
int c = numbers.Rank;
c = c * c;
```

```
int[] numbers = new[] {3, 9, 7, 8, 3};
int c = numbers.Sum();
c = c / numbers.Length;
```

```
int[] numbers = new[] {3, 9, 7, 8};
int c = numbers.Length;
c = 2 * c;
```

```
int[] numbers = new[] {3, 9, 7, 8};
int c = numbers.Sum();
c = c % 4;


for (int i = 1; i < 5; i++)
{
    Console.Write("{0}", i);
}


int[] numbers = new[] {1, 2, 3, 4};
int c = numbers.Rank;
c = c % 2;


int[] numbers = new[] {3, 9, 7, 8};
int c = numbers.Length - 1;


int[] numbers = new[] {3, 9, 7, 8};
Array.Reverse(numbers) ;
int c = numbers.Length;


for (int i = 9 ; i < 10; i++)
{
    Console.Write("{0}", i);
}


string str = "Visual Studio";
int k = 0;
while (k <= str.Length - 1);
{
    Console.Write("{0} ", str[k]);
    k++;
}


int[] numbers = new[] {7, 8};
int c = numbers.Length;


int[] arr = {2, 4, 6, 8};
for (int i = 3; i >= 0; i--)
{
    Console.Write("{0}", arr[i]);
}
```

```
for (int i = 9; i < 10; i++)
{
    Console.Write("{0}-", i);
}


int[] arr = {2, 4, 6, 8, 10};
for (int i = 4; i >= 0; i--)
{
    Console.Write("{0}", arr[i]);
}


for (int i = 1; i < 5; i++)
{
    Console.Write(' ' * i);
}


for (int i = 4; i < 6; i++)
{
    Console.Write("{0}", i);
}


int[] arr = {2, 4, 6, 8, 10};
for (int i = 2; i >= 0; i--)
{
    Console.Write("{0}", arr[i]);
}


for (int i = 0; i < 10; i++)
{
    Console.Write("{0}", (i%5).ToString());
}


for (int i = 19; i < 20; i++)
{
    Console.Write("{0}", i.GetType());
}


for (int i = 19; i < 20; i++)
{
    Console.Write("{0}, {1}", i, i.Equals(i % 2));
}
```

```
int[] numbers = new[] {634, 724, 72, 8, 99, 3454};
int c = numbers.GetLowerBound(0);


for (int i = 1; i < 2; i++)
{
    Console.Write("{0}", i);
}


for (int i = 8; i < 10; i++)
{
    Console.Write("{0},", i);
}


for (int i = 1; i < 10; i++)
{
    Console.Write("{0}", i * i % i);
}


for (int i = 16; i < 25; i++)
{
    Console.Write("{0}", i % 3 + 5);
}


int[] testInts = new[] {4, 8, 15, 16, 23, 39};
int x = testInts.Sum();
foreach (int test in testInts)
{
    Console.Write("{0}, ", test);
}
Console.Write("The sum: {0}", x);


for (int i = 1; i < 10; i++)
{
    Console.Write("{0}", i % 2 + i);
}


for (int i = 1; i < 10; i++)
{
    Console.Write("{0}", i % 2);
}


int[] numInts = new[] {12, 34, 45, 2};
int a = numInts.Max();
```

```
for (int i = 1; i < 10; i++)
{
    Console.Write("{0}", i * 10);
}


string t = "Main", s = "Void";
var d = string.Concat(t, s);


for (int i = 3; i < 6; i++)
{
    Console.Write("{0} ", i);
}


int opt = Byte.MinValue, num1 = 6, num2 = 2;
switch (opt)
{
    case 0:
        Console.Write(num1 + num2);
        break;
    case 1:
        Console.Write(num1 - num2);
        break;
}


int[] numbers = new[] {73, 81, 454, 790};
int g = numbers.GetUpperBound(0);


int[] numbers = new[] {54, 45, 546, 763, 47368};
int g = numbers.Rank;
g = g * numbers.Length;


int i = 1;
do
{
    Console.Write("tekst");
    i++;
} while (i < 1);


string s = "isc.ttu.ee system";
s = s.Remove(10);
```

```
int[] numbers = new[] {13, 85, 21, 0};
int g = numbers.Sum();


string sw = "suvaline tekst";
char c = (char) 115;
var a = sw.Contains(c);


int x = 7;
int y = 3;
int z = x + y;
z = y / x;
x = z - 2;
Console.Write("x = {0}, y = {1}, z = {2}", x, y, z);


byte b = Byte.MaxValue;
int c = b + Byte.MinValue;


int i = Int32.MaxValue;
i = i + 1;


var q = 13
q = q - 3;


float f = float.MinValue;
f = f - 1;


double d = 60, n = 0;
Console.Write(d / n);


bool b = (32 == ' ');


bool c = (3 == '3');


int x = 6, result;
for (int i = 12; i < 12; i++)
{
    result = x * i;
    Console.Write("{0} x {1} = {2}", x, i, result);
}
```

```
int a = 23, b = 6, c = 14, d = 7;
Console.Write(a + b / c * d);
```

```
int[] numInts = new[] {3, 4, 9, 6, 0};
var t = numInts[0];
for (int i = 0; i < numInts.Length - 1; i++)
{
    numInts[i] = numInts[i + 1];
}
numInts[numInts.Length - 1] = t;
Console.Write("{0}", String.Join(" ", numInts));
```

```
decimal d = Decimal.MaxValue;
double db = Double.MaxValue;
bool s = ((double) d > db);
```

```
int i = 100;
while (i < 110)
{
    Console.Write((char)i + " ");
    i++;
}
```

```
int i = 100;
while (i < 110)
{
    Console.Write(i + " ");
    i++;
}
```

```
int i = 0;
do
{
    Console.Write(i + " ");
    i++;
} while (i < 10);
```

```
int i = 0;
do
{
    Console.Write(i + ' ');
    i++;
} while (i < 10);
```

```
for (int i = 0; i < 20; i++)
{
    if (i % 3 == 0)
    {
        Console.Clear();
    }
    Console.Write(i);
}


for (int i = 0; i < 5; i++)
{
    Console.Write(i % 2 * i);
}


List<int> numList = new List<int> {43, 25, 78};
numList.Sort();
foreach (int i in numList)
{
    Console.Write(i);
}


int opt = 3;
switch (opt)
{
    case 0:
        Console.Write("zero");
        break;
    case 1:
        Console.Write("one");
        break;
    default:
        Console.Write("default");
        break;
}

var opt = "opt";
switch (opt)
{
    case "1":
        Console.Write("zero");
        break;
    case "case":
        Console.Write("one");
        break;
    default:
        Console.Write("Do nothing");
        break;
}
```

```
string[] vs = { "one", "two", "three" };
foreach (string str in vs)
{
    Console.Write("{0},", str.Length);
}
```

```
string[] vs = { "i", "s", "c" };
foreach (string str in vs)
{
    Console.Write("{0}", str.ToUpper());
}
```

```
List<string> vs = new List<string> { "i", "s", "c" };
vs.Add("x");
Console.Write(vs.Capacity);
```

```
List<string> vs = new List<string> { "i", "s", "c" };
Console.Write(vs.IndexOf("s"));
```

Lisa 2 – Sisendit koguvad ülesanded

```
public static int fn(int number)
{
    int result = 0;
    for (int i = 1; i < number; i++)
    {
        if ((i * i * i) <= number)
        {
            result = i;
        }
    }
    return result;
}
```

```
public static int fn(int number)
{
    int result = 0;
    for (int i = 1; i < number; i++)
    {
        if ((i * i) <= number)
        {
            result = i;
        }
    }
    return result;
}
```

```
public static int fn(int number)
{
    int result = 0;
    for (int i = 1; i < 10; i++)
    {
        if ((i * i) <= number)
        {
            result = i;
        }
    }
    return result;
}
```

```
public static int fn(int number)
{
    int result = 0;
    for (int i = 1; i < 8; i++)
    {
        if ((i * i) <= number)
        {
            result = i;
        }
    }
    return result;
}
```

```
public static int fn(int number)
{
    int result = 0;
    for (int i = 1; i < 10; i++)
    {
        if ((i * i) <= number)
        {
            result = i - 1;
        }
    }
    return result;
}
```

```
public static int fn(int number)
{
    int result = 1;
    for (int i = 1; i < 10; i++)
    {
        if ((i * i) <= number)
        {
            result = i - 1;
        }
    }
    return result;
}
```

```
public static int fn(int number)
{
    int result = 0;
    for (int i = 1; i < number; i++)
    {
        if ((2 * i) <= number)
        {
            result = i;
        }
    }
    return result;
}


public static int fn(int number)
{
    int result = 0;
    for (int i = 1; i < number; i++)
    {
        if ((i * i) <= 5)
        {
            result = i;
        }
    }
    return result;
}


public static string f(int[] arr)
{
    string B = "";
    for (int i = 2; i >= 0; i--)
    {
        string c = arr[i].ToString();
        B = B + c;
    }
    return B;
}


public static string f(bool x1, bool x2)
{
    string B = "not OK";
    if (x1 ^ x2)
    {
        B = "OK";
    }
    return B;
}
```

```
public static string f(bool x1, bool x2)
{
    string B = "OK";
    if (x1 ^ x2)
    {
        B = "not OK";
    }
    return B;
}


public static string f(bool x1, bool x2)
{
    string B = "not OK";
    if (x1 | x2)
    {
        B = "OK";
    }
    return B;
}


public static string f(bool x1, bool x2)
{
    string B = "not OK";
    if (x1 & x2)
    {
        B = "OK";
    }
    return B;
}


public static string f(bool x1, bool x2)
{
    string B = "not OK";
    if (x1 ^ x1)
    {
        B = "OK";
    }
    return B;
}
```

```
static int Bit(int n)
{
    int c = 0;
    while (n != 0)
    {
        c++;
        n &= (n - 1);
    }
    return c;
}


static int ItBit(int n)
{
    int test = n;
    int c = 0;
    while (test != 0)
    {
        if ((test & 1) == 1)
        {
            c++;
        }
        test >>= 1;
    }
    return c;
}


static string RemCh(string key)
{
    string table = "";
    string result = "";
    foreach (char value in key)
    {
        if (table.IndexOf(value) == -1)
        {
            table += value;
            result += value;
        }
    }
    return result;
}
```

```
static void Test(int[] array)
{
    if (array != null && array.Length > 0)
    {
        int first = array[0];
        Console.Write(first);
    }
}


static int seq(int[] dataset, int target, int n)
{
    int found = 0;
    int pos = -1;
    for (int i = 0; i < n && found != 1; i++)
    {
        if (target == dataset[i])
        {
            pos = i;
            found = 1;
        }
    }
    return pos;
}


static void sortDesc(int[] dataset, int n)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = n - 1; j > i; j--)
        {
            if (dataset[j] > dataset[j - 1])
            {
                int temp = dataset[j];
                dataset[j] = dataset[j - 1];
                dataset[j - 1] = temp;
            }
        }
    }
}
```

```
static void sortAsc(int[] dataset, int n)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = n - 1; j > i; j--)
        {
            if (dataset[j] < dataset[j - 1])
            {
                int temp = dataset[j];
                dataset[j] = dataset[j - 1];
                dataset[j - 1] = temp;
            }
        }
    }
}
```

```
public static bool IsPal(string text)
{
    text = text.ToUpper();
    int j = text.Length - 1;
    for (int i = 0; i < j; i++)
    {
        if (text[i] != text[j])
        {
            return false;
        }
        j--;
    }
    return true;
}
```

```
public static int Count(string text)
{
    int count = 0;
    string letter;
    for (int i = 0; i < text.Length; i++)
    {
        letter = text.Substring(i, 1);
        if (letter == " ")
            count++;
    }
    return count;
}
```

```
public static int Pow(int num, int exp)
{
    int result = 1;
    for (int i = 0; i < exp; i++)
    {
        result *= num;
    }
    return result;
}


public static bool Prime(int n)
{
    for (int i = 2; i < n; i++)
    {
        if (n % i == 0)
        {
            return false;
        }
    }
    return true;
}


public static int SumDig(int n)
{
    string number = Convert.ToString(n);
    int sum = 0;
    for (int i = 0; i < number.Length; i++)
    {
        sum += int.Parse(number.Substring(i, 1));
    }
    return sum;
}


public static int Fact(int num)
{
    if (num == 0)
    {
        return 1;
    }
    else
    {
        return num * Fact(num - 1);
    }
}
```

```
static float Max(float[] list)
{
    float max = -99999999.00f;
    for (int i = 0; i < list.Length; i++)
    {
        if (i == 0)
        {
            max = list[i];
        }
        else
        {
            max = max < list[i] ? list[i] : max;
        }
    }
    return max;
}


public static string Change(string text)
{
    if (text.Length <= 1)
    {
        return text;
    }
    char first = text[0];
    string rest = text.Substring(1);
    return Change(rest) + first;
}


public static int Sq(int num, int min, int max)
{
    int result = 0;
    for (int i = min; i <= max; i++)
    {
        if ((i * i) <= num)
        {
            result = i;
        }
    }
    return result;
}
```