TARTU ÜLIKOOL
LOODUS- JA TÄPPISTEADUSTE VALDKOND
Arvutiteaduse instituut
Informaatika õppekava

Kaspar Hollo

Programmeerimise e-kursusel osalejate küsimuste analüüs ja selle põhjal murelahendajate koostamine

Bakalaureusetöö (9 EAP)

Juhendajad: Eno Tõnisson, MSc
Tauno Palts, MA

Tartu 2016

Programmeerimise e-kursusel osalejate küsimuste analüüs ja selle põhjal murelahendajate koostamine

Lühikokkuvõte:

Tartu Ülikooli arvutiteaduse instituut korraldas 2015/2016. õppeaastal programmeerimise MOOCe (vaba ligipääsuga ja osalejate arvu poolest suuremahuline tasuta e-kursus), kus osalejatel oli e-kirjade vahendusel võimalik oma küsimustele kursuse korraldajatelt tagasisidet saada. Sügissemestril toimunud MOOCi „Programmeerimisest maalähedaselt“ osalejate poolt saadetud e-kirjad sisaldasid aga palju korduma kippuvaid küsimusi. Käesoleva bakalaureusetöö eesmärk on kevadsemestril toimuvate MOOCide osalejate korduma kippuvaid küsimusi ennetada ja sellega vähendada laekuvate e-kirjade arvu. Selleks analüüsiti esmalt osalejate e-kirjades sisalduvaid küsimusi ja saadud tulemused liigitati juhendajatega kooskõlastatud kategooriatesse. Liigitatud andmete analüüsi tulemusena selgitati välja osalejate poolt tehtavad korduvad vead. Väljaselgitatud korduvatest vigadest lähtuvalt koostati ülesannete kohta käivate vihjete kogumid ehk murelahendajad. Murelahendajad abistasid kevadsemestril toimuvates MOOCides vihjete kaudu osalejaid ning püüdsid ennetada neil tekkida võivaid küsimusi. Sügis- ja kevadsemestril toimunud MOOCide „Programmeerimisest maalähedaselt“ käigus laekunud e-kirjade arvu võrdlus näitas, et pärast murelahendajate kasutuselevõttu vähenes e-kirjade arv märkimisväärselt.

Võtmesõnad:

MOOC, murelahendaja, programmeerimine, kategoriseerimine

CERCS: P175, informaatika, süsteemiteooria

An analysis of questions asked by the participants of a programming e-course and the troubleshooters based on it

Abstract:

The Institute of Computer Science of the University of Tartu held programming MOOC-s (Massive Open Online Course) in the 2015/2016 learning year, where participants had the option to ask for feedback from the organizers of the course via e-mail. The e-mails sent by the participants of the autumn semester MOOC “About Programming” included many frequently asked questions. The objective of this bachelor’s thesis is to obviate the questions

of the following spring semester's MOOCs participants and thus reduce the number of e-mails received. To achieve this objective, the questions within the participants' e-mails were analysed and the results were classified into categories which had been co-ordinated with instructors. As a result of the data analysis process, the participants' frequently asked questions were extracted. Troubleshooters were created based on the extracted frequently asked questions. The troubleshooters were meant to help the participants of the spring-semesters' MOOCs through collections of hints and obviate questions those participants might have during aforementioned MOOCs. A comparison between the number of e-mails received in the autumn and spring semesters in the MOOC "About Programming" showed that after the implementation of troubleshooters the number of e-mails decreased significantly.

Keywords:

MOOC, troubleshooter, programming, categorization

CERCS: P175, informatics, systems theory

Sisukord

Sissejuhatus

Käesolevas bakalaureusetöös analüüsitakse programmeerimisteemalisel e-kursusel osalejate küsimusi ja analüüsi tulemuste põhjal koostatakse murelahendajaid. Töös analüüsitakse vaba ligipääsuga ja osalejate arvu poolest suuremahulise tasuta e-kursuse ehk MOOCi (ingl k *Massive Open Online Course*) „Programmeerimisest maalähedaselt“ e-kirjades olevaid küsimusi. Analüüsi tulemuste põhjal koostatakse ülesannete kohta käivad murelahendajad, mis on Murelahendaja keskkonna abil loodud vihjete komplektid.

Maailma erinevates ülikoolides on hakanud levima trend korraldada MOOCe. Selles töös kasutataksegi selliste e-kursuste kirjeldamiseks lühendit MOOC, kuna puudub sobiv eestikeelne lühend. Tartu Ülikooli arvutiteaduse instituut korraldas oma esimese MOOCi 2014/2015. õppeaastal – selleks oli eestikeelne programmeerimise MOOC nimega „Programmeerimisest maalähedaselt“. Eelmainitud MOOC ei jäänud aga viimaseks. Tartu Ülikooli arvutiteaduse instituut korraldas ka 2015/2016. õppeaasta sügis- ja kevadsemestril eestikeelseid programmeerimise MOOCe „Programmeerimisest maalähedaselt“ ja „Programmeerimise alused“. Osalejailt ei nõutud nendel MOOCidel eelnevaid programmeerimisalaseid teadmisi – tegemist oli sissejuhatavate kursustega programmeerimisse programmeerimiskeeles Python. Käesolevas bakalaureusetöös käsitletaksegi 2015/2016. õppeaasta sügissemestril toimunud MOOCi „Programmeerimisest maalähedaselt“ ning kevadsemestril toimunud MOOCe „Programmeerimisest maalähedaselt“ ja „Programmeerimise alused“.

Kursuslased pidid aine läbimiseks lahendama iganädalaseid programmeerimisülesandeid. Lahenduste kontrollimiseks kasutati automatiseeritud hindamissüsteemi ehk automaatkontrolli, mis eeldas püstitatud ülesande täpset lahendamist. Paratamatult tuli osalejatel ülesannete lahendamisel aeg-ajalt sisse mõni viga. Põhilised põhjused, miks programmi ei saadud nõutekohaselt tööle olid järgnevad:

- probleemid äsja omandatud teadmiste rakendamisega, sest õpitava teema materjal jäi lõplikult omandamata – nt raskused programmeerimiskeele süntaksi järgimisega;
- ei arvestatud automaatkontrolli nõudmistega – nt automaatkontrolli läbimiseks pidid defineeritavad funktsioonid omama kindlaid nimesid.

Kursusel osalejatel oli võimalik saata lahendamise käigus tekkinud probleemide kohta kursuse korraldajatele e-kirju. Sügissemestril toimunud MOOCi „Programmeerimisest

maalähedaselt“ vältel laekus 1534 osaleja poolt kokku umbes 1250 e-kirja, milles oli paratamatult ka mitmeid korduma kippuvaid küsimusi. Välja tuli mõelda korraldajatele sobiv lahendus e-kirjade hulga vähendamiseks, sest korduma kippuvaid küsimusi taheti sama õppeaasta kevadsemestril toimuvates MOOCides vältida.

Sügissemestril toimunud MOOCi „Programmeerimisest maalähedaselt“ e-kirjades sisalduvad küsimused otsustati probleemide täpsemaks väljaselgitamiseks esmalt kokku koguda ja siis saadud küsimusi analüüsida. Lisaks eelpool mainitud tegevustele otsustasid kursuste läbiviijad võtta kevadsemestril toimuvate MOOCide „Programmeerimisest maalähedaselt“ ja „Programmeerimise alused“ juures kasutusele abivahendi nimega Murelahendaja (ingl k *troubleshooter*). Murelahendajad annavad osalejatele ülesannete lahendamiseks suunavaid vihjeid. Iga vihje esitatakse esmalt küsimuse vormis ja hiljem vastatakse sellele küsimusele (vt Lisa I). Murelahendajate loomisel toetuti küsimuste analüüsist saadud tulemustele. Murelahendajate koostamise keskkonna programmeeris Vello Vaherpuu enda bakalaureusetöö raames (2016). Edaspidi kasutatakse bakalaureusetöös terminit murelahendaja, mis tähistab Murelahendaja keskkonna abil loodud konkreetse programmeerimisülesande kohta käivate vihjete komplekti.

Käesoleva bakalaureusetöö eesmärgiks on muuta 2015/2016. õppeaasta kevadsemestril toimuvad MOOCid „Programmeerimisest maalähedaselt“ ja „Programmeerimise alused“ võrreldes sügissemestril toimunud MOOCiga „Programmeerimisest maalähedaselt“ veelgi efektiivsemaks – ennetada ülesannete lahendamisel osalejatel tekkida võivaid probleeme ja sellega vähendada ka laekuvate e-kirjade hulka. Eesmärgini taheti jõuda kolme etapi läbimise tulemusel:

1. andmete kogumine – osalejate e-kirjade alusel koguda andmetabelisse kokku võimalikult kvaliteetsed andmed esinenud probleemide kohta;
2. andmetabeli analüüs – selgitada välja osalejate programmides esinevad korduvad vead;
3. andmetabeli analüüsist saadud tulemuste rakendamine murelahendajates.

Töö autor osales mainitud MOOCide läbiviimisel. Lisaks analüüsis osalejate e-kirjades olevaid küsimusi, märkis küsimuste analüüsi tulemused andmetabelisse ja seejärel analüüsis omakorda täidetud andmetabelit. Hiljem koostas murelahendajaid kevadsemestril toimunud MOOCide „Programmeerimise alused“ ja „Programmeerimisest maalähedaselt“ jaoks.

Lisaks sissejuhatusele, kokkuvõttele ja kasutatud kirjanduse loetelule on käesolevas bakalaureusetöös veel kolm peatükki. Esimeses peatükis antakse lühiülevaade erinevatest MOOCidest ja sellest, kuidas nendes MOOCides tagasisidet jagatakse. Samuti tutvustatakse Tartu Ülikooli arvutiteaduse instituudi poolt korraldatud MOOCe, murelahendajaga sarnast süsteemi ja seda, kuidas on programmeerimisvigu sarnases uurimuses liigitatud. Teises peatükis kirjeldatakse käesolevas bakalaureusetöös kasutatavat metoodikat. Kolmandas peatükis esitatakse bakalaureusetöö tulemused ja tehakse järeldusi. Bakalaureusetööl on 11 lisa, sealhulgas osalejate poolt tehtud vigu sisaldav andmetabel, nimekiri väljaselgitatud korduvalt esinenud vigadest ja viited koostatud murelahendajate juurde.

1. Ülevaade MOOCidest ja tagasisidest

1.1 MOOC ja tagasiside tüübid

1.1.1 Mis on MOOC?

MOOC tähistab vaba ligipääsuga ja suure osalejaskonnaga internetipõhist kursust, mis on kõikidele osalejatele tasuta (Vihavainen jt, 2012). Dhawal Shahi poolt kirjutatud artikli kohaselt korraldati 2015. aastal rohkem kui 500 erineva ülikooli poolt ligi 4000 MOOCi. Artikli andmetel kahekordistus 2015. aastal MOOCides osalevate inimeste arv võrreldes 2014. aastaga – kui 2014. aastal oli hinnanguline MOOCis osalejate arv 16-18 miljonit, siis 2015. aastal oli vastav näitaja 35 miljonit (Shah, 2015).

Maailma üheks populaarseimaks ja MOOCide hulga poolest suurimaks keskkonnaks on Coursera. Shahi andmetel (2015) pakuti 2015. aasta lõpus Coursera keskkonnas ligikaudu 1500 MOOCi, kus osales 17 miljonit inimest. Eeltoodud andmete järgi osales ühes Coursera MOOCis keskmiselt ligi 11 500 inimest. Coursera kursused on väga erinevatest valdkondadest ja osalemiseks on vaja ainult internetiühendust.

1.1.2 Programmeerimise MOOCid

Lisaks õpetatavale teemale võivad programmeerimise MOOCid erineda teistest MOOCidest ka osalejate lahenduste kontrollimise viisi poolest. Osalejate teadmisi kontrollitakse MOOCides enamasti valikvastustega testide abil (Ebner jt, 2014). Seevastu programmeerimise alaste MOOCide puhul kontrollitakse kursuse osalejate poolt koostatud programme automaatkontrolli abil (Vihavainen jt, 2012). Programmide kontrollimiseks kasutatakse automaatkontrolle, mis koosnevad erinevatest ühiktestidest (ingl k *unit test*). Ühiktestide abil saab kontrollida programmi erinevaid aspekte:

- käitumist – nt kas programmile antavate sisendite puhul saadakse oodatud vastused;
- struktuuri – nt kas programmis kasutatakse ettenähtud funktsioone.

Mõnedes programmeerimise MOOCides kasutatavad automaatkontrollid annavad osalejatele esinenud vigade kohta ka tagasisidet. Selliste MOOCide hulka kuuluvad näiteks MIT sissejuhatav programmeerimise MOOC[1] ja Helsingi Ülikooli objektorienteeritud program-

[1] https://www.edx.org/course/introduction-computer-science-mitx-6-00-1x-7

meerimist õpetav MOOC[2]. Selliste automaatkontrollide puuduseks on aga see, et jagatav tagasiside ei pruugi algajaist programmeerijatele alati arusaadav olla (Singh jt, 2013).

MOOCides võib esineda erinevaid probleeme. Üheks selliseks probleemiks on kursuse toimumise vältel osalejatele tagasiside andmine neil tekkinud küsimuste kohta. Järgmises kahes jaotises kirjeldatakse mõnda sellist tüüpi, mis erinevad üksteisest juhendajate poolt antava tagasiside poolest.

1.1.3 MOOCid, kus korraldajad ei andnud tagasisidet

MOOCidel, kus korraldajate poolt ei antud mingit tagasisidet, toodi peamiseks põhjuseks osalejate rohkus (Rohe, 2014; Kop jt, 2011). Õpetatavate suure hulga pärast kontrollitaksegi sellistes MOOCides osalejate teadmisi või oskusi automatiseeritult (Siemens, 2013). Kursuse korraldajad reeglina kellelegi eraldi tagasisidet andma ei hakka ja sellepärast võib osadele osalejatele arusaamatuks jääda, miks nende poolt esitatud lahendus oli väär või mida nad konkreetses olukorras valesti tegid (Rohe, 2014).

MOOCides, kus korraldajad ei jaga tagasisidet, soovitatakse abi saamiseks foorumi poole pöörduda (Bali, 2014). Osalejad saavad tekkinud probleemide ja ka teiste kursust puudutavate küsimuste üle foorumis aru pidada (Bali, 2014). Anant Agarwali väitel ei osale juhendajad foorumites asetleidvates vestlustes kindlal põhjusel. Agarwali sõnul viib kursuslaste omavaheline arutelu kiiremini õige lahenduseni, kui juhendajalt konkreetsele küsimusele vastust oodates (Duhring, 2013). Programmeerimise MOOCides võib sellisel juhul probleemiks osutuda õigete lahenduste foorumisse postitamine (Warren jt, 2014).

Juhendajatepoolse tagasisideta MOOCides on levinud ka teistsugused abistamise viisid. Courseras pakutakse MOOCe (nt California Ülikooli objektorienteeritud programmeerimist õpetav MOOC[3]), milles programmeerimist puudutavaid küsimusi üritatakse lisaks põhivideotele ka abivideotega lahendada. Põhivideod asendavad kirjalikke põhimaterjale sellistel kursustel, kus õpetatakse osalejatele mingit kindlat teemat (nt funktsioonid) ja tuuakse selle põhjal erinevaid näiteid. Abivideod on suunatud inimestele, kellele jäi põhivideotes teema kohta esitatud selgitustest väheseks. Abivideod üritavadki seda olukorda parandada, seletades läbitud teema detailsemalt ja uute näidete alusel lahti. Põhimaterjale

[2] http://mooc.fi/courses/2013/programming-part-1/

[3] https://www.coursera.org/learn/object-oriented-java

toetavad abivideod on kasutusel ka Tartu Ülikooli arvutiteaduse instituudi poolt korraldatud programmeerimise MOOCides.

Seda tüüpi MOOCide eelisteks on nii inim-, aja- kui ka raharessursi kokkuhoid. Puuduseks on juhendajate poolt saadava personaliseeritud tagasiside puudumine.

1.1.4 MOOCid, kus korraldajad andsid tagasisidet

Lisaks leidub programmeerimisalaseid MOOCe, kus korraldajad annavad osalejatele tagasisidet. Üheks selliseks oli Rice'i Ülikooli poolt 2013. aasta kevadel korraldatud graafilist interaktiivset programmeerimist õpetav MOOC, mille alusel viisid kursuse korraldajad läbi ka uurimuse. Warreni jt uurimuse üheks püstitatud eesmärgiks oli välja selgitada, kas personaliseeritud tagasiside andmine on sellise suurusjärguga kursusel võimalik. Osalejad kasutasid programmeerimiseks veebipõhist arenduskeskkonda CodeSkulptor (*sic*), mis võimaldas osalejatel oma programme URLi kaudu teistega jagada. Selline funktsionaalsus võimaldas osalejatel oma probleemseid programme lingi vahendusel MOOCi abiliinile (ingl k *help desk*) saata. Seega pääsesid juhendajad programmile vaevata ligi ja said osalejale kiiret tagasisidet anda. 9 nädala pikkuse MOOCi jooksul saatsid 1201 osalejat abiliinile kokku 2503 e-kirja. E-kirjadele vastas kogu MOOCi vältel kolm juhendajat. Tagasiside andmise peale kulus keskmiselt 45 tundi nädalas. Korraldajad veendusid MOOCi lõpus, et osalejatele personaliseeritud tagasiside andmine kogu MOOCi kestel on võimalik (Warren jt, 2014).

Seda tüüpi MOOCide eelisteks on juhendajate poolne osalejatele personaliseeritud tagasiside andmine. Puuduseks jällegi inim-, aja- kui ka raharessursi kulumine.

1.2 MOOCide „Programmeerimisest maalähedaselt" ja „Programmeerimise alused" üldinfo

2015/2016. õppeaasta kevadsemestril toimunud Tartu Ülikooli MOOCid „Programmeerimise alused" (tegu oli proovikursusega) ja „Programmeerimisest maalähedaselt" on kombineeritud versioonid viimases kahes jaotises mainitud tüüpidest. Neid kursuseid on üritatud võimalikult palju automatiseerida (kasutades nii automaatkontrolli kui ka murelahendajaid), kuid osalejatele on antud ka võimalus igal hetkel oma murega abiliini poole pöörduda. Abiliin on kasutajatugi, kus kursuse korraldajad vastavad osalejate poolt saadetud e-kirjadele. Selleks, et võimaldada kursusel osalejatel kursust efektiivsemat läbimist ning vähendada kursuse läbiviijate vajadust vastata pidevalt ühtedele ja samadele

esitatud küsimustele, mõeldi välja murelahendaja kontseptsioon. Murelahendajate koostamiseks tuleks eelnevalt võimalikud korduvad vead osalejate e-kirjade põhjal välja selgitada. Korduvate vigade väljaselgitamist võimaldas sügissemestril toimunud nelja nädala pikkune MOOC „Programmeerimisest maalähedaselt“, kus osales 1534 inimest. MOOCil „Programmeerimisest maalähedaselt“ oli automaatkontroll juba olemas ja selle peamiseks ülesandeks oli hinnata kursuslaste töid (arvestatud/mittearvestatud). Lisaks hindamisele andis automaatkontroll osalist tagasidet, mis võis mõningatel juhtudel osalejatele arusaamatuks jääda. Automaatkontrolli poolt saadud tagasisidest hoolimata otsustasid paljud kursuslased oma murest kirjutada abiliinile, kus kursuse korraldajad vastasid kokku umbes 1250 e-kirjale. Need e-kirjad jagunesid omakorda programmeerimis-põhisteks ja organisatoorseteks e-kirjadeks. Programmeerimispõhiste e-kirjade alusel selgitas töö autor välja korduvalt esinenud vead ja koostas saadud tulemusi arvesse võttes murelahendajad.

1.3 Varasemad murelahendajaga sarnased süsteemid

Varasemalt on kasutatud mitmeid murelahendajaga sarnaseid ülesannete lahendamist toetavaid süsteeme. Ühte sellist süsteemi kasutas ka Helsingi Ülikooli poolt korraldatud objektorienteeritud programmeerimist õpetav MOOC. Programmeerimiseks kasutati NetBeansi integreeritud arenduskeskkonda ehk IDEt (ingl k *integrated development environment*) koos NetBeansi pistikprogrammiga (ingl k *plugin*) Test My Code. Eeltoodud pistikprogramm laadis spetsiaalselt loodud kohustuslikud programmeerimisülesanded projektina alla ja lisas seejärel IDEsse. Kõik allalaaditud projektid sisaldasid ülesande-spetsiifilisi ühikteste, mille käivitamisel jagati IDEs vihjeid, kui kontrollitavas programmis esines mõni viga (Vihavainen jt, 2012).

Enne kui ülesanded anti MOOCis osalevatele inimestele lahendada, katsetati ülesanded mitmes erinevas faasis läbi, et saaks luua asjakohaseid teste ja vihjed. Esimeses faasis (alfatestimine) märkisid kursuse korraldajad võimalikke vigu ja teises faasis (beeta-testimine) saadi sama aine statsionaarset kursust läbivate üliõpilaste käest tagasisidet. Ülesandeid sai vajadusel parandada ka pärast nende avaldamist. MOOCis lahendati kokku 170 erinevat ülesannet, mille juurde kuulus ligikaudu 1300 testi koos üle 4000 vihjega (Vihavainen jt, 2012).

Tartu Ülikooli arvutiteaduse instituudi MOOCide ja Helsingi Ülikooli MOOCi süsteemide sarnasuseks võib tuua, et mõlemad annavad analüüsitud andmete põhjal ülesande-

spetsiifilisi vihjeid. Samuti kontrollitakse mõlemal juhul ülesandeid ühiktestidest koosneva automaatkontrolliga. Erinevuseks on see, et Helsingi Ülikooli poolt korraldatavas MOOCis antakse vihjeid kasutuses olevate testide alusel IDEs. Seevastu Tartu Ülikooli arvutiteaduse instituudi poolt korraldavates MOOCides antakse vihjeid murelahendajate ja automaatkontrolli vahendusel Moodle'i keskkonnas, mitte kasutatavas IDEs.

1.4 Programmeerimisvigade liigitamine

Algajaist programmeerijate aitamiseks on esmalt vaja nende probleemid välja selgitada. Sandy Garner jt viisid läbi uuringu, mille eesmärk oli õpilaste probleeme märkida ja kategoriseerida, et hiljem saaks kogutud andmeid uurida ja seeläbi tehtavate muudatuste alusel kursust paremaks muuta. Garneri jt väitel sõltub kogutud andmete kvaliteet võimalikke probleeme kirjeldavate kategooriate kirjelduste korrektsusest ja kategooriate kirjelduste rakendamise usaldusväärsusest. Artiklis väidetakse, et kategooriate loomisel tuleb valida täpsuse ja praktilisuse vahel. Kui moodustada palju erinevaid kategooriaid, mille kirjeldused on vähepaindlikud, siis saab tekkinud probleemidest põhjaliku ülevaate, kuid kategooriate rohkuse tõttu on neid raske järgida. Kui moodustada jällegi vähe kategooriaid, mille kirjeldused on paindlikud, siis on probleeme lihtsam liigitada, kuid andmete põhjal saadud ülevaade pole nii detailne (Garner jt, 2005).

Uurimuse aluseks võeti Otago Ülikooli poolt korraldatud sissejuhatav kursus programmeerimisse programmeerimiskeeles Java. Algsed veakategooriate kirjeldused loodi Garneri jt poolt juba uuringule eelnevatel aastatel. Toona loodud veakategooriate kirjeldused põhinesid praktikumides käsitlevatel teemadel ja kirjandusest leitud uuringutel. Praktikumi juhendajate sõnul oli veakategooriate kirjeldusi aga liiga palju, mis muutis nende kasutamise ebamugavaks. Seega otsustati konkreetse uuringu puhul praktikumi juhendajate ja teiste ekspertide abiga veakategooriate kirjeldusi vähendada (Garner jt, 2005).

Garneri jt uurimuses selgitas praktikumi juhendaja osalejal esinenud probleemid välja vahetu suhtluse teel. Nimelt käisid praktikumi juhendajad klassiruumis ringi ja aitasid abi küsinud tudengeid, märkides esinenud vead hiljem üles. Sellisel viisil sai praktikumi juhendaja täpselt välja uurida, milles osaleja probleem seisnes. Negatiivseks küljeks on jällegi asjaolu, et saadud tulemusi ei saa pärast kontrollida, mille tõttu ei saa olla kindel info usaldusväärsuses. Seetõttu käis praktikumides tavajuhendajatega kaasas antud teemas rohkemate kogemustega vanemjuhendaja, kes jälgis tudengi ja tavajuhendaja vahelist abiosutamise protsessi ja kategoriseeris esinenud vead etteantud juhendi alusel vastavatesse

veakategooriatesse. Hiljem võrreldi tavajuhendaja ja vanemjuhendaja märkeid (Garner jt, 2005).

Sarnaselt Sandy Garneri jt poolt koostatud uurimusele (2005) sooviti ka käesolevas bakalaureusetöös saada tekkinud probleemide kohta võimalikult palju kvaliteetseid andmeid. Bakalaureusetöö autor kasutas kategooriate kirjelduste loomisel ekspertarvamusi. Täpsemalt järgmises peatükis „Metoodika“.

2. Metoodika

2015/2016. õppeaasta sügissemestril toimunud MOOCi „Programmeerimisest maalähedaselt“ osalejad saatsid abiliinile ligikaudu 1250 e-kirja. Selleks, et sama õppeaasta kevadsemestril toimuvates MOOCides saaks osalejaid paremini aidata, koguti e-kirjades sisalduvad küsimused kokku ja seejärel analüüsiti saadud küsimusi. E-kirjades sisalduvate küsimuste analüüsimisel loodeti saada ülevaade programmeerimisel tekkinud probleemidest.

Probleemide väljaselgitamiseks koostati MOOCi „Programmeerimisest maalähedaselt“ toimumise vältel laekunud e-kirjade põhjal andmetabel (vt Lisa II), milles kajastuvad osalejate programmides esinenud programmeerimisvead ja organisatoorsed teavitused kategooriate kaupa. Kuigi andmetabel sisaldab ka organisatoorsete teavituste kohta käivaid andmeid (näiteks hilinemisest teatamine), keskendutakse selles bakalaureusetöös eelkõige e-kirjadele, mis sisaldasid küsimusi programmeerimise kohta.

Enne kui andmetabelit sai täitma hakata, tuli luua andmetabelis esinevad kategooriad koos kirjeldustega (vt Lisa III) ja andmetabeli täitmise juhend (vt Lisa IV). Bakalaureusetöö autor ei leidnud ühtegi uurimust, milles oleks programmeerimiskeele Python põhjal üritatud programmeerimisvigu käsitlevaid kategooriaid kirjeldada ja hiljem loodud kirjelduste alusel programmeerimisvigu kategoriseerida. Bakalaureusetöö autori meelest ei sobinud käesolevas töös rakendamiseks ka paljud Garneri jt uurimuses (2005) kasutavatest kategooriatest, kuna õpetatavad kursused erinevad üksteisest oluliselt (seda nii kursustel õpetavate teemade kui ka programmeerimiskeeltest tingitud erinevuste poolest). Näiteks ei käsitleta MOOCis „Programmeerimisest maalähedaselt“ objektorienteeritud programmeerimisele omaseid klasse. Bakalaureusetöös ei saanud kasutada ka Garneri jt uurimuses (2005) kasutatud andmete kogumise viisi, kuna siinses uurimuses ei suhtle töö autor ja osaleja vahetult, vaid e-kirjade vahendusel. Asjakohaste uurimuste ja eelnevate kategooriate kirjelduste puudumise tõttu otsustati bakalaureusetöös olevad veakategooriad koos kirjeldustega luua autori ja juhendajate arutelude tulemusena.

E-kirjade esimesel läbitöötamisel loodi nii kategooriate kirjeldused kui ka andmetabeli täitmise juhend. Kategooriate kirjelduste koostamisel arvestati peamiselt kursusel käsitletavate programmeerimiskonstruktsioonide ja automaatkontrolli poolt seatud nõudmistega. Koostatud kategooriad ei ole kindlasti veel optimaalsed ja neid saab tulevaste uuringute jaoks täiustada. Hiljem kooskõlastasid töö autor ja juhendajad ka andmetabeli täitmise

juhendi. Saamaks esinenud programmeerimisvigadest võimalikult täpse ülevaate, otsustati koostada pigem palju kategooriaid.

Pärast eelmainitud juhendi ja kategooriate kirjelduste valmimist hakati andmetabelit täitma. Täitmise käigus jaotati andmetabelis leiduvad veakategooriad omakorda ülesannete alla, kus neid vigu tehti. Nii oli võimalik esinenud probleeme ülesannete kaupa tuvastada. Vigade vältimiseks kontrollis bakalaureusetöö autor andmetabelis olevad märked koostatud ees-kirjade järgi kaks korda üle ja viis vajadusel sisse muudatused.

Veendumaks andmetabeli täitmise juhendi ja kategooriate kirjelduste arusaadavuses, otsustati nende arusaadavust kontrollida. Juhendi arusaadavuse kontrollimiseks kasutati kahe eksperdi abi, kes tegelevad igapäevaselt programmeerimise õpetamisega. Eksperdid tegid eeltoodud juhendi ja 30 juhuslikult valitud pileti (ingl k *ticket*) põhjal märkeid teise, kuid samu kategooriaid sisaldavasse andmetabelisse. Pileti puhul on tegu kirjade ahelaga, mis sisaldab vähemalt ühte osaleja poolt saadetud e-kirja. Igas e-kirjas on omakorda vähemalt üks organisatoorne teade (nt hilinemisest teatamine) või programmeerimisvigu sisaldav programm.

Ekspertide poolt kontrollitud piletitest käsitlesid 18 piletit organisatoorseid teavitusi ja 12 piletit programmeerimisvigu sisaldavaid programme. Organisatoorsete teavituste kategoriseerimisel langesid kõik bakalaureusetöö autori ja ekspertide märked kokku. Selle uurimuse jaoks on aga palju olulisemad piletid, mis käsitlesid programmeerimisvigu sisaldavaid programme. Andmetabeli täitmise juhendi arusaadavus tuletatigi bakalaureusetöö autori ja ekspertide poolt tehtud programmeerimisvigu tähistavate märgete (vt Lisa V) kattuvusprotsentide alusel.

Bakalaureusetöö autori ja esimese eksperdi poolt tehtud märgete kattuvusprotsendiks tuli ligikaudu 83,1% ning bakalaureusetöö autori ja teise eksperdi poolt tehtud märgete kattuvusprotsendiks 80,3%. Kattuvusprotsendi arvutamise näite leiab lisast VI (vt Lisa VI). Üle 80% kattuvus on kõrge ja näitab, et kategooriad ja nende kirjeldused on üldjuhul arusaadavad ja tulemused suures osas üheti mõistetavad.

Ekspertide hinnangul suurendas kategooriate rohkus sisseelamiseks kuluvat aega ja võis olla põhjuseks, miks tekkisid üksikud erinevused. Samuti võis erinevus tekkida kirjade ahelas samasuguse vea korduval esinemisel. Seega ühe võimaliku parandusena võiks tulevaste uuringute jaoks kaaluda kategooriate mõningast vähendamist ja vigade lugemist kirjade ahelas ühekordselt.

Pärast andmetabeli täitmise juhendi arusaadavuse kontrollimist analüüsiti täidetud andmetabelit. Analüüsi peamiseks eesmärgiks oli korduvalt esinevate vigade (vt Lisa VII) väljaselgitamine. Korduvalt esinenud vigade hulka lisati ainult selliseid vigu, mida osalejad tegid kohustuslike programmeerimisülesannete lahendamisel ja kui viga oli esinenud vähemalt kolmel erineval inimesel. Nii üritati korduvate vigade hulgas vältida eriskummalisi ja harva esinevaid vigu. Korduvate vigade vältimiseks loodi murelahendajad, mis annavad osalejatele programmeerimisülesannete vihjeid.

Bakalaureusetöö autor ei leidnud kirjandusest, et mõnes muus MOOCis oleks murelahendajatega samamoodi vihjeid jagatud. Seega tuli bakalaureusetöö autoril juhend ise kavandada. Sarnaselt andmetabeli täitmise juhendile, valmis ka murelahendaja koostamise juhend (vt Lisa VIII) töö käigus. Lõplik murelahendaja koostamise juhend kujunes välja bakalaureusetöö autori ja juhendajate arutelude tulemusena. Iga ülesande kohta käiva murelahendaja koostamisel toetuti otseselt andmetabeli analüüsi käigus saadud tulemustele. Konkreetse ülesande murelahendaja valmimisel pandi see kursuse materjalides ülesande kirjelduse alla osalejatele kasutamiseks. Osalejate uute e-kirjade ja Murelahendaja keskkonna poolt kogutud statistika järgi parandati murelahendajaid jooksvalt. Antud bakalaureusetöö lisast IX leiab metoodika rakendamise järjekorra ka kokkuvõtlikul kujul (vt Lisa IX).

3. Tulemused

3.1 Andmete kogumine

Probleemide väljaselgitamiseks koguti ja analüüsiti kursuse „Programmeerimisest maalähedaselt“ osalejate poolt saaadetud küsimusi. Kursuslastel oli oma programmide kohta käivaid küsimusi võimalik saata abiliinile – sügissemestril toimunud 4 nädala pikkuse MOOCi „Programmeerimisest maalähedaselt“ jooksul tegid seda 398 osalejat 1534 osaleja hulgast. Abiliini poole pöördunud osalejatelt laekus 700 piletit, mis omakorda sisaldasid umbes 1250 saabunud e-kirja. Käesolevas uurimuses toetutakse eelkõige e-kirjadele, mis sisaldasid programmeerimisvigadega programme.

Järgnevalt on esitatud sügissemestril toimunud MOOCi „Programmeerimisest maalähedaselt“ käigus saadetud e-kirja näide. Laekunud e-kiri käis 2. nädalal toimunud III kontrollülesande (III kontrollülesanne, 2015) kohta, mille teemaks oli tingimuslause.

„Tere
Taas olen jännis, ma ei saa pihta mida teha, et teine osa töötaks. Isikukoodi "he" osa saan käima ja omast arust teen nagu õiget asja...
Tervitades
XXX“
E-kirjaga kaasa pandud programmilõik nägi selline välja:

```
print("sisesta isikukood:")
sisestatud_isikukood = input()

if sisestatud_isikukood == "1" or "3" or "5":
    print("he")
elif sisestatud_isikukood == "2" or "4" or "6":
    print("she")
```

Legendi ja andmetabeli täitmise juhendi alusel tuleb andmetabelisse (vt joonis 1) teha III kontrollülesande all olevatesse kategooriatesse kokku kaks märget. Esimene märge läheb kategooria „Tsükkel või tingimuslause“ alla ja teine märge kategooria „Esimese sümboli

valimine“ alla. Mõlemad märgitud vead kuuluvad ka korduvalt esinenud vigade hulka, sest mõlemal puhul on täpselt sellise vea teinud rohkem kui kolm erinevat inimest.

	A	Y	Z	AA	AB	AC	AD	AE
1		III kontrollülesanne						
2	Osalejad / kategooriad	Tsükkel või tingimuslause	Esimese sümboli valimine	Muutuja viga	Süntaks ja treppimine	Inputi viga	Tüübi viga	Väljastamine
59	XXX							
60	XXX							
61	XXX	I	I				I	
62	XXX							
63	XXX							
64	XXX							
65	XXX	IIII			II	I		
66	XXX							
67	XXX	I	I				I	
68	XXX	II			I	I		I

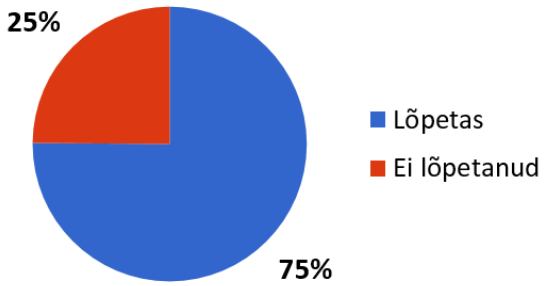
Joonis 1. Andmetabeli III kontrollülesande väljalõige.

Kõik laekunud e-kirjad kontrolliti mitmeid kordi üle, et andmetabelis oleks võimalikult vähe vigu. Kogu protsess võttis aega kümneid tunde.

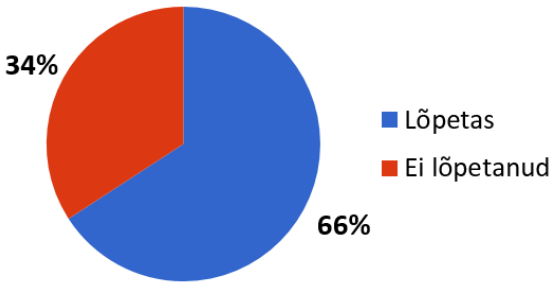
3.2 Andmetabeli analüüs

Täidetud andmetabeli põhjal saab teha järeldusi nii kontrollülesannete püstituste, loodud materjalide kui ka automaatkontrolli kohta. Andmetabelisse tehti kokku 1327 märget, millest 788 käisid kohustuslike kontrollülesannete kohta. Ülejäänud 539 märget jaotusid nädala testide, lisaülesannete, korralduslike küsimuste jms kategooriate vahel. Selles bakalaureusetöös tehakse järeldusi eelkõige kohustuslike programmeerimisülesannete alusel.

Saadud tulemuste põhjal võib väita, et abiliinist on kasu olnud. Andmetabelist selgub, et sügissemestril toimunud MOOCi „Programmeerimisest maalähedaselt” jooksul abi küsinud inimestest lõpetas 75,1% (vt joonis 2). Abi mitteküsinud inimeste puhul on sama näitaja aga 62,6% (vt joonis 3). Võib oletada, et ilma abiliini poolt pakutava toeta oleks lõpetajate osakaal olnud abi küsinud inimeste seas märgatavalt väiksem, sest tegu oli eeldatavalt väiksemate oskustega osalejatega. See oleks tõenäoliselt omakorda viinud alla ka üldise lõpetajate protsendi (65,8%).



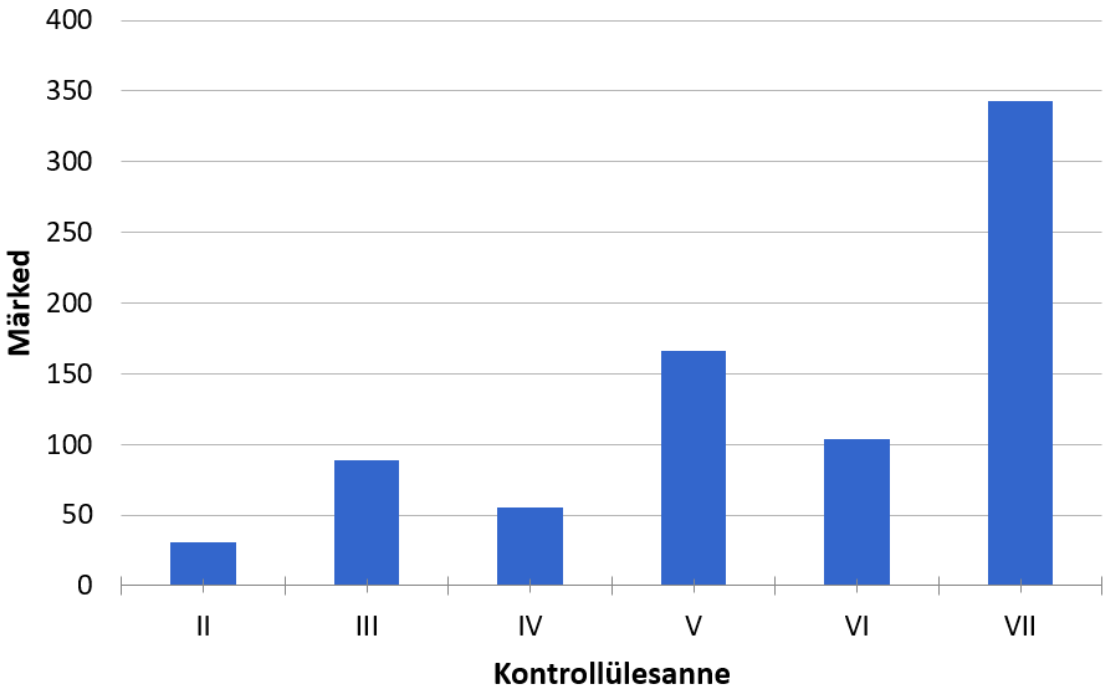
Joonis 2. Abi küsinud inimeste lõpetamisprotsent.



Joonis 3. Abi mitteküsinud inimeste lõpetamisprotsent.

Andmetabeli analüüsi peamine eesmärk oli korduvalt esinenud vigade väljaselgitamine (vt Lisa VI). Korduvalt esinenud vigade abil sooviti saada teemade kaupa ülevaadet, millele tuleks tulevaste MOOCide jaoks loodavate murelahendajate küsimuste ja vastuste koostamisel rohkem tähelepanu pöörata. Käsitlevate teemade seas olid muutujad, andmetüübid, tingimuslause, sõned, tsükkel, regulaaravaldis ja funktsioonid.

Andmetabeli alusel saab välja tuua ka muid asju. Näiteks milline ülesanne või teema valmistas osalejatele kõige rohkem raskusi. Kogutud andmete kohaselt (vt joonis 4) valmistas ülekaalukalt kõige rohkem probleeme VII kontrollülesanne (VII kontrollülesanne, 2015), mille läbivaks teemaks olid funktsioonid. Kõigist 788 kohustuslike programmeerimisülesannete kohta tehtud märkest tehti selle ülesande juures 343 märget. See moodustab ligikaudu 44% kõikidest kontrollülesannete kohta tehtud märgetest.



Joonis 4. Kontrollülesannete kohta tehtud märgete arv.

Järgnevalt uuritakse VII kontrollülesande kohta koostatud märgete tabelit (vt tabel 1). Lisaks üritatakse tuua põhjuseid, miks selle kontrollülesande lahendamisel nii palju vigu esines. Tabelist eristuvad märgete rohkuse poolest 3 kategooriat: funktsiooni väljakutsumine (64 märget), muutuja viga (74 märget) ja ümardamine (51 märget).

Tabel 1. VII kontrollülesande märgete arv kategooriate kaupa.

VII kontrollülesanne	
Kategooria	Märkeid
Funktsiooni argument puudu või kasutamata	38
Funktsiooni väljakutse	64
Ümardamine	51
Muutuja viga	74
Tüübi viga	31
Süntaks ja treppimine	23
Arvutusviga	19
Inputi viga	31

Funktsiooni väljakutsumisega seotud eksimuste suur arv näitab, et tegemist on keerulise teemaga. Kursuse materjalides funktsiooni väljakutsumist käsitlevat jaotist tuleks üritada pikemalt lahti seletada ja täiendavate näideta abil illustreerida.

Kõige rohkem esines selles kontrollülesandes muutuja vigu – 74 märget. Automaatkontrolli ajas segadusse, kui osaleja pani defineeritavale funktsioonile ja selles funktsioonis kasutatavale muutujale (lokaalsele muutujale) sama nime. Pythonis pole selline nimetamine aga otseselt keelatud ja seega muudeti kevadsemestril toimunud MOOCide automaatkontrolle nii, et sellist asja ei loeta enam veaks.

Ümardamise puhul on ilmselt probleemiks inimeste tähelepanematus. Ülesande püstituse kohaselt tuleb defineeritud funktsiooni sees ümardada. Paljud inimesed ümardasid aga defineeritud funktsiooni väliselt ja seega ei lugenud automaatkontroll esitatud lahendusi õigeks. Kevadsemestril toimunud MOOCis „Programmeerimisest maalähedaselt“ vahetati VII kontrollülesanne sarnase ülesande vastu välja ja seal üritati sõnastuses rõhutada, et ümardada tuleb juba funktsiooni sees.

3.3 Andmetabeli analüüsi tulemuste rakendamine murelahendajates

Kevadsemestril toimunud MOOCide „Programmeerimisest maalähedaselt“ ja „Programmeerimise alused“ jaoks koostas bakalaureusetöö autor kokku 41 murelahendajat (vt Lisa X). Iga murelahendaja koostamise peale kulus keskmiselt kaks tundi aega. Murelahendajate loomisel toetuti andmetabeli analüüsi käigus saadud tulemustele, peamiselt korduvalt esinenud vigadele (vt Lisa VII).

3.3.1 Murelahendaja loomise keskkond

Murelahendajate loomisel ei saa mainimata jätta ka Murelahendaja keskkonda – rakendus, mille abil luuakse murelahendajaid ning kogutakse statistikat iga murelahendaja kohta. Murelahendaja keskkonda hakati arendama 2015/2016. õppeaasta sügissemestri lõpus Vello Vaherpuu poolt (2016) ning arendus lõppes sama õppeaasta kevadsemestri lõpus. Keskkond sai MOOCi „Programmeerimise alused“ alguseks piisavalt valmis, et kursuse jaoks saaks hakata murelahendajaid koostama. Esimestest kursuse „Programmeerimise alused“ murelahendajatest on selgelt näha, et bakalaureusetöö autor pidi mõne keskkonna funktsionaalsuse valmimiseni veidi improviseerima. Näiteks ei olnud esimeste murelahendajate koostamisel võimalik määrata küsimustele ja vastustele järjekorranumbreid. Aja möödudes

lisandusid keskkonda erinevad funktsionaalsused, mis muutsid murelahendajate koostamise kiireks ja mugavaks.

3.3.2 Näide murelahendajast

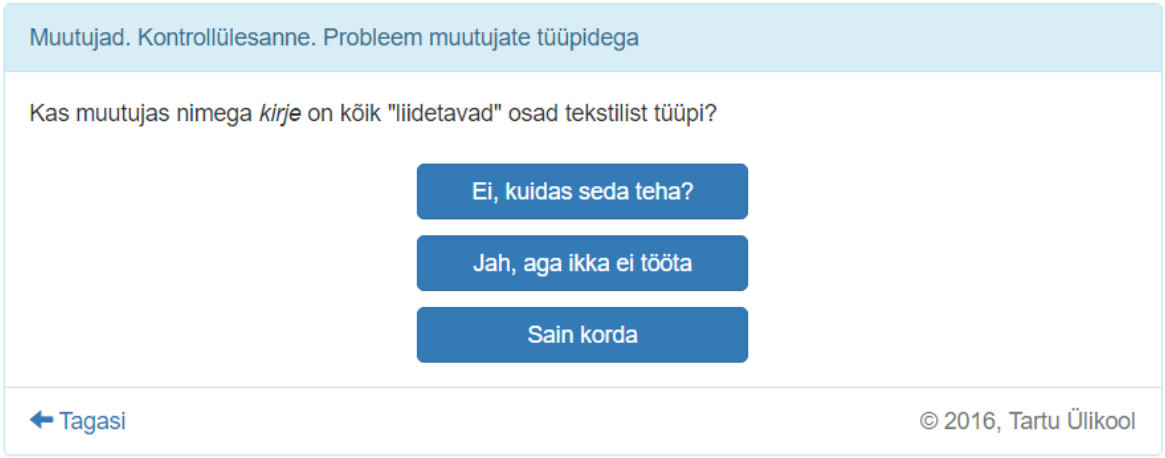
Järgmisena tuuakse näide murelahendaja kaudu antud vihje kohta (küsimus ja vastus). Näide põhineb kevadsemestril toimunud MOOCi „Programmeerimisest maalähedaselt“ 1. nädala II kontrollülesandel (II kontrollülesanne, 2016), mille teemaks oli muutujad ja andmetüübid. Murelahendaja kaudu jagatava vihje koostamisel oli otseselt abi järgnevast korduvalt esinenud veast (vt joonis 5).

Erinevat tüüpi liikmeid üritatakse üheks sõneks kokku liita. Näiteks:

```
aasta = 2015
loom = "metssiga"
print(aasta)
print(loom)
print(aasta + ". loomaks on " + loom)
```

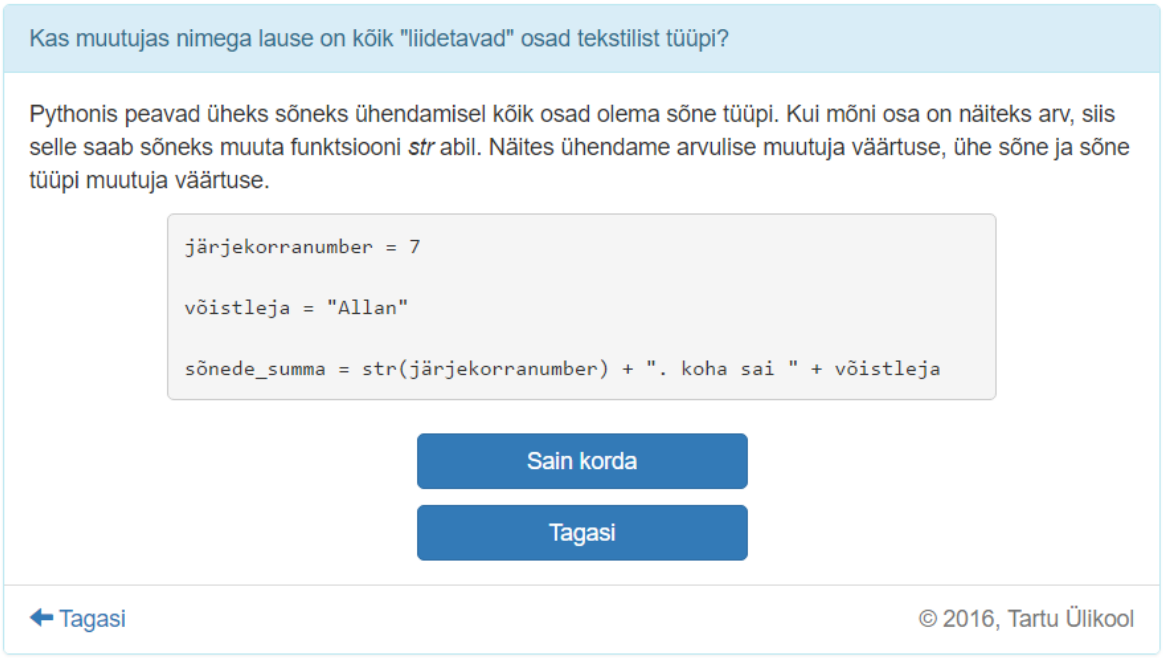
Joonis 5. Teema „Muutujad ja andmetüübid“ puhul korduvalt esinenenud viga

Vihje kohta käiva küsimuse eesmärk on suunata inimest oma programmi esitatud küsimuse alusel kontrollima. Konkreetse küsimuse puhul palutakse osalejal muutuja kirje väärtuseks olevate liidetavate osade tüübid üle kontrollida (vt joonist 6).



Joonis 6. II kontrollülesande kohta käiv murelahendaja küsimus.

Kui inimene ei saa küsimuse kujul olevast vihjest aru, siis vihje pikem selgitus koos näitega leidub vastuse all. Eelneva küsimuse (vt joonis 6) juurde käiv vastus selgitab, kuidas saab mingit muud tüüpi osad sõneks muuta. Lehekülje all leidub ka koodi kujul olev näidis (vt joonis 7).



Joonis 7. II kontrollülesande kohta käiv murelahendaja vastus.

Sarnasel viisil koostati ka ülejäänud murelahendajates olevad vihjed. Murelahendajate koostamise täpsema juhendi leiab lisast VIII (vt lisa VIII).

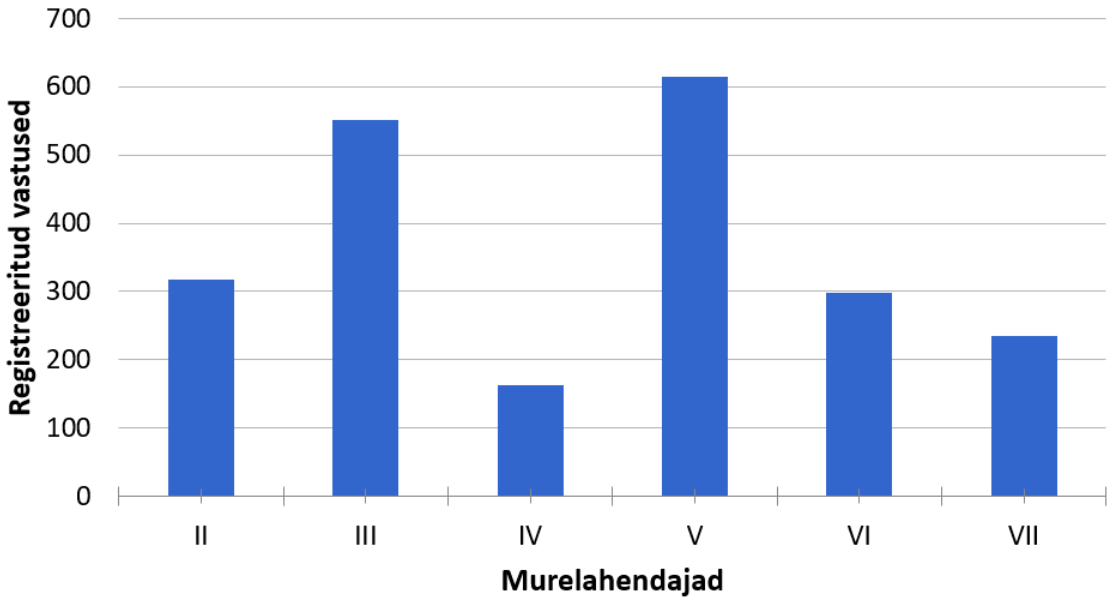
3.3.3 Murelahendajate erinevused

Kõikide MOOCi „Programmeerimise alused“ kontrollülesannete murelahendajate koostamisel ei saanud otseselt lähtuda väljaselgitatud korduvalt esinenud vigadest ja teistest analüüsi tulemustest, kuna vead märgiti andmetabelisse ainult sügissemestril toimunud MOOCi „Programmeerimisest maalähedaselt“ alusel. MOOCi „Programmeerimisest maalähedaselt“ raames läbiti vähem teemasid (MOOCide võrdlus, 2016) ja teemasid ei töötatud ka nii põhjalikult läbi kui MOOCis „Programmeerimise alused“. Kevadsemestril toimunud MOOCi „Programmeerimisest maalähedaselt“ murelahendajaid sai seevastu korduvalt esinenud vigade ja teiste analüüsitulemuste põhjal hästi koostada, kuna sügis- ja kevadsemestril toimunud kursuste teemad ning enamik programmeerimisülesannetestki jäid samadeks.

3.4 Järeldused

Kevadsemestri MOOCideks koostatud murelahendajate kasutegurit saab hinnata, sest Murelahendaja keskkond kogub andmeid selle kohta, kui palju inimesi murelahendaja kaudu esitatud vihjete abil lahenduseni jõudsid. Kui abi otsinud inimene sai jagatud vihje abil probleemi lahendatud, vajutas ta loodetavasti nupule „Sain korda“, mille peale Murelahendaja keskkond registreeris antud nupuvajutuse.

Alljärgnevalt tehakse järeldusi ainult MOOCi „Programmeerimisest maalähedaselt“ jaoks loodud murelahendajate tulemuslikkusest, sest nende loomisel sai täiel määral lähtuda andmetabeli analüüsi käigus saadud tulemustest. Murelahendaja keskkonna poolt saadud andmete põhjal (vt joonis 8) selgub, et kõige rohkem saadi abi V kontrollülesande murelahendaja poolt esitatud vihjetest (615 registreeritud vastust). Kõige vähem saadi abi IV kontrollülesande murelahendaja poolt esitatud vihjetest (163 registreeritud vastust). Kõikide MOOCi „Programmeerimisest maalähedaselt” jaoks loodud murelahendajate kaudu jagatavatest vihjetetest saadi abi 2180 korral.



Joonis 8. Murelahendajate tulemuslikkus kontrollülesannete kaupa.

Lisaks Murelahendaja keskkonna poolt kogutud andmetele on võimalik järeldusi teha ka sügis- ja kevadsemestril toimunud MOOCide „Programmeerimisest maalähedaselt“ e-kirjade hulga erisuste põhjal. Kuna mõlemal MOOCi osalejate arv oli võrreldav (1534 osalejat sügissemestril, kellest lõpetas 1010, ja 1430 osalejat kevadsemestril, kellest lõpetas 882), siis on saabunud e-kirjade arvu vähenemine pärast murelahendajate kasutuselevõttu

statistiliselt märkimisväärne. Sügissemestril laekus umbes 1250 e-kirja ja kevadsemestril umbes 750 e-kirja. Ei saa välistada, et e-kirjade hulga erisus tulenes osalejate individuaalsetest eripäradest (varasem programmeerimiskogemus, julgus abiliini poole pöörduda jms), sest kursusel osalejaid ei võrreldud. Siiski muudavad osalejate valimite suurused tõenäolisemaks, et murelahendajatest on kasu olnud. Selle alusel võib väita, et murelahendajad muutsid MOOCid efektiivsemaks.

Täpsemate tulemuste saamiseks tuleb ka 2015/2016. õppeaasta kevadsemestril toimunud MOOCi „Programmeerimisest maalähedaselt“ probleemid analoogselt e-kirjade alusel esmalt välja selgitada ja siis saadud andmeid analüüsida. Äsja mainitud MOOCi hilise lõppemisaja tõttu otsustati see antud uurimusest välja jätta.

4. Kokkuvõte

Inimesed harivad ennast erinevate MOOCide abil üha enam. Käesoleva bakalaureusetöö alguses kirjeldati kirjandusele toetudes MOOCide olemust ja seda, kuidas sellistes MOOCides tagasisidet jagatakse. Selgus, et korraldajad ei jaga enamike MOOCide käigus osalejatele personaliseeritud tagasisidet, kuid osades programmeerimist õpetavates MOOCides on kasutusele võetud automaatkontrollid, mis annavad lisaks hindamisele ka tagasisidet. Automaatkontrolli poolt antav tagasiside ei pruugi aga olla osalejatele piisavalt arusaadav.

2015/2016. õppeaastal Tartu Ülikooli arvutiteaduse instituudi poolt korraldatud programmeerimise MOOCides said osalejad lisaks automaatkontrolli poolt jagatavale tagasisidele küsida korraldajate käest e-kirjade vahendusel ka personaliseeritud tagasisidet. Korduma kippuvate küsimuste rohkuse tõttu otsustati selle vastu midagi ette võtta.

Käesoleva bakalaureusetöö eesmärgiks oli muuta 2015/2016. õppeaasta kevadsemestril toimuvad MOOCid „Programmeerimisest maalähedaselt“ ja „Programmeerimise alused“ võrreldes sügissemestril toimunud MOOCiga „Programmeerimisest maalähedaselt“ veelgi efektiivsemaks – ennetada ülesannete lahendamisel osalejatel tekkida võivaid probleeme ja ühtlasi vähendada ka laekuvate e-kirjade hulka.

Eesmärgini taheti jõuda kolme etapi läbimise tulemusena. Esimeses etapis sooviti 2015/2016. õppeaasta sügissemestril toimunud MOOCi „Programmeerimisest maalähedaselt“ osalejate poolt saadetud e-kirjades olevate küsimuste alusel koguda andmetabelisse võimalikult kvaliteetseid andmeid esinenud probleemide (programmeerimisvead või organisatoorsed teavitused) kohta. Juhend ja kategooriate kirjeldused, mille alusel probleemid andmetabelisse märgiti, loodi bakalaureusetöö autori ja juhendajate arutelude tulemusel. Bakalaureusetöös kirjeldatud Sandy Garneri jt uurimuses (Garner jt, 2005) koguti andmeid osalejate probleemide kohta sarnasel viisil, kuid siinses lõputöös ei saanud Garneri jt uurimuses kasutatud kategooriaid ja andmete kogumise meetodit õpetavate kursuste erinevuste tõttu kasutada. Bakalaureusetöö autori poolt loodud andmetabeli täitmise juhendi ja kategooriate kirjelduste arusaadavust kontrollisid eksperdid juhuslikult valitud piletite alusel. Kontrolli tulemusel selgus, et nii uurimuses kasutatav andmetabeli täitmise juhend kui ka kategooriate kirjeldused on arusaadavad. Seega võis järeldada, et ka saadud tulemused on suures osas üheti mõistetavad.

Teises etapis analüüsiti andmetabelisse kogutud andmeid. Andmetabeli analüüsi käigus selgitati välja osalejate programmides korduvalt esinevad vead. Väljaselgitatud korduvalt esinenud vigade alusel oli teemade kaupa võimalik ette näha, milliseid vigu osalejad tulevastes MOOCides kõige sagedamini teevad. Selline teadmine tuli kasuks murelahendajate loomisel. Peale korduvalt esinenud vigade väljaselgitamise vaadeldi siinses uurimuses lähemalt, milline sügissemestril toimunud MOOCi „Programmeerimisest maalähedaselt“ kontrollülesanne valmistas osalejatele kõige rohkem raskusi ja millised olid põhjused.

Kolmandas etapis kasutati andmetabeli analüüsi tulemusi MOOCide „Programmeerimisest maalähedaselt“ ja „Programmeerimise alused“ murelahendajate loomisel. Bakalaureusetöö autor koostas 41 erinevate murelahendajat. Murelahendajate koostamise juhend valmis bakalaureusetöö autori ja juhendajate arutelude tulemusena, kuna bakalaureusetöö autor ei leidnud ühtegi teist sellist MOOCi, kus oleks murelahendajaga sarnasel viisil vihjeid jagatud. Sarnaseimat süsteemi pakkus Helsingi Ülikooli poolt korraldatud objekt-orienteeritud programmeerimist õpetav MOOC, kus automaatkontroll jagas hoolikalt analüüsitud andmete põhjal vihjeid (Vihavainen jt, 2012). Murelahendajad koostati Vello Vaherpuu poolt tema bakalaureusetöö raames (2016) valmistatud Murelahendaja keskkonnas. Murelahendajate loomisel toetuti analüüsi käigus saadud tulemustele, eelkõige osalejate lahendustes esinenud korduvatele vigadele. MOOCi „Programmeerimise alused“ murelahendajate loomisel sai MOOCi „Programmeerimisest maalähedaselt“ käigus väljaselgitatud korduvalt esinenud vigadele toetuda vaid osaliselt, sest MOOCis „Programmeerimisest maalähedaselt“ läbiti vähem teemasid.

Kokkuvõttes võib bakalaureusetöö eesmärki täidetuks pidada. Murelahendajate lisamisega kevadsemestril toimunud MOOCile „Programmeerimisest maalähedaselt“ vähenes abiliinile tulnud e-kirjade hulk märgatavalt – sügissemestril saadeti abiliinile 1250 e-kirja ja kevadsemestril 750 e-kirja. Asjaolu, et murelahendajatest oli abi, näitavad ka Murelahendaja keskkonna kaudu kogutud sellekohased andmed, mille kohaselt saadi murelahendajate vihjetest abi 2180 korral. Eeltoodust tulenevalt võib järeldada, et murelahendajate kasutuselevõtt MOOCides on olnud vajalik ja kasulik, sellest on olnud osalejatele probleemide ennetamisel abi ja murelahendajad on muutnud MOOCid efektiivsemaks.

5. Kasutatud materjalid

Bali, M. (2014). MOOC pedagogy: gleaning good practice from existing MOOCs. *Journal of Online Learning and Teaching*, 10(1), 44–56.

Duhring, J. (14.06.2013). Effective habits of power users: A look at recent MOOC research. *MOOC News & Reviews*: http://www.moocnewsandreviews.com/effective-habits-of-power-users-a-look-at-recent-mooc-research/ (20.02.2016)

Ebner, M., Lackner, E., & Kopp, M. (2014). How to MOOC? – A pedagogical guideline for practitioners. Roceanu, I. (ed.). Proceedings of the 10th International Scientific Conference "eLearning and Software for Education" Bucharest, April 24 - 25, 2014. Publisher: Editura Universitatii Nationale de Aparare "Carol I" http://www.openeducationeuropa.eu/sites/default/files/asset/How-to-MOOC-A-pedagogical-guideline-for-practitioners.pdf (02.05.2016)

II kontrollülesanne. (2016). Kursus: „Programmeerimisest maalähedaselt": https://courses.cs.ut.ee/2016/progmaa/Main/PARTIIYlesanne (29.04.2016)

III kontrollülesanne. (2015). Kursus: „Programmeerimisest maalähedaselt": https://courses.cs.ut.ee/2015/progmaa/fall/Main/PARTIIIYlesanne (29.04.2016)

Kop, R., Fournier, H., & Mak, J. S. F. (2011). A pedagogy of abundance or a pedagogy to support human beings? Participant support on massive open online courses. The International Review Of Research In Open And Distributed Learning (IRRODL); Vol 12, No 7, pp. 74–93.

MOOCide võrdlus. (2016). Kursus: „Programmeerimisest maalähedaselt": https://courses.cs.ut.ee/2016/progmaa/spring/Main/Maalahealusedvordlus (29.04.2016)

Rohe, B., Literacy, E. M. MOOCs: Benefits, Costs, Ethics, Out comes and Popular Opinion. Educational Technology Portal, School of Education, University of Delaware. www.udel.edu/edtech/gallery/examples/Ben-Rohe-MOOCs.docx (02.05.2016)

Shah, D. (2015). By The Numbers: MOOCS in 2015. Class Central: https://www.class-central.com/report/moocs-2015-stats/ (26.02.2016)

Siemens, G. (2013). Massive open online courses: Innovation in education. In *Open educational resources: Innovation, research and practice*, 5. Ed.: R. McGreal, W. Kinuthia, S. Marshall. Published by Commonwealth of Learning and Athabasca University,

Vancouver. http://web.iaincirebon.ac.id/ebook/Indrya/Bandura/inovasi/pub_PS_OER-IRP_web.pdf#page=31 (02.05.2016)

Singh, R., Gulwani, S., Solar-Lezama, A. (2013). Automated feedback generation for introductory programming assignments. In ACM SIGPLAN Notices. Vol. 48, No. 6, pp. 15–26.

Warren, J., Rixner, S., Greiner, J., Wong, S. (2014). Facilitating human interaction in an online programming course. In *Proceedings of the 45th ACM technical symposium on Computer science education.* ACM. pp. 665–670.

Vaherpuu, V. (2016). Murelahendaja loomise keskkond. Tartu Ülikooli arvutiteaduse instituut, bakalaureusetöö.

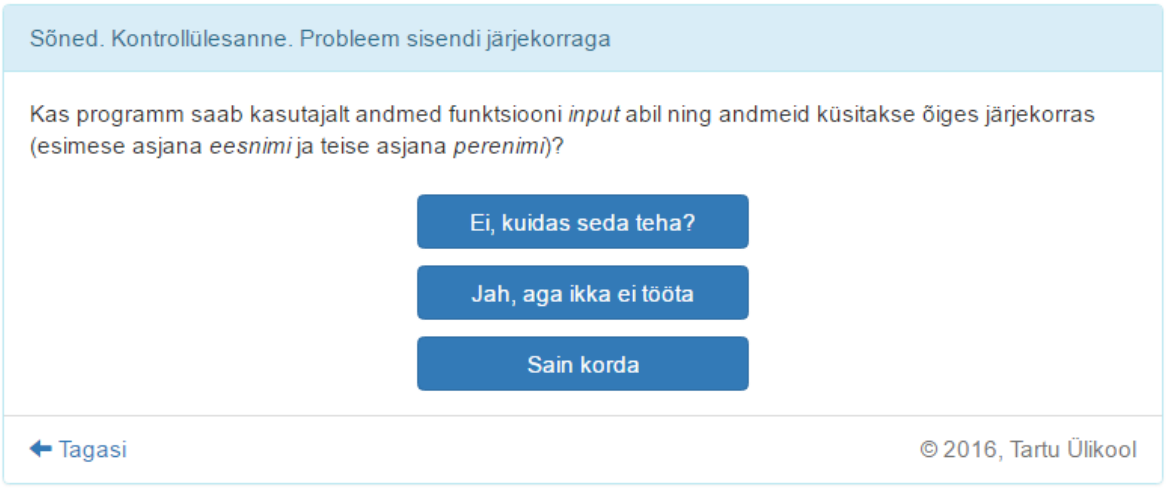
Vihavainen, A., Luukkainen, M., Kurhila, J. (2012). Multi-faceted support for MOOC in programming. In *Proceedings of the 13th annual conference on Information technology education*. ACM. pp. 171–176.

VII kontrollülesanne. (2015). Kursus: „Programmeerimisest maalähedaselt“: https://courses.cs.ut.ee/2015/progmaa/fall/Main/PARTVIIYlesanne (29.04.2016)

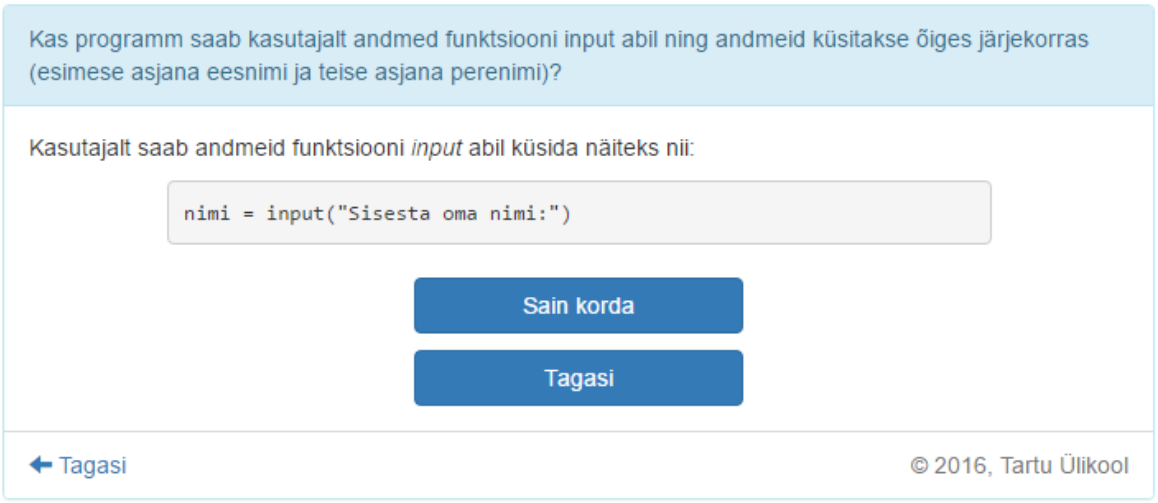
Lisad

I. Murelahendaja küsimus ja vastus

Murelahendaja küsimuse ja vastuse ekraanitõmmised (vt joonis 9 ja 10).



Joonis 9. IV kontrollülesande kohta käiv murelahendaja küsimus.



Joonis 10. IV kontrollülesande kohta käiv murelahendaja vastus.

II. Andmetabel

Järgneb lühike andmetabeli kirjeldus. Veerus A olid algselt osalejate nimed, kelle kohta märkeid andmetabelisse tehti. Osalejate anonüümsuse tagamiseks asendati kõik andmetabelis olevad nimed kirjetega „XXX“. Kategooriate pealkirjad kajastuvad andmetabeli teises reas. Andmetabeli veergudesse B kuni J tehakse märkeid organisatoorsete teavituste kohta ja alates veerust K tehakse märkeid programmeerimisvigade kohta. Programmeerimisvigade kohta käivad veakategooriad on omakorda jaotatud ülesannete alla, saamaks ülesannete kaupa parema ülevaate, milliseid vigu vastavas ülesandes täpsemalt tehti. Andmetabelis olevate märgete „I“ hulk ühes lahtris tähistab seda, kui mitu seda liiki viga (olenevalt millise kategooria alla märge tehtud on) on töö autor konkreetse osaleja programmidest leidnud.

Viide andmetabelile:

https://docs.google.com/spreadsheets/d/1NOj5mGcbSueRn8r0ccPkZKtFTuoI_M9jlVVbuW0VIZw/edit?usp=sharing

III. Kategooriate kirjeldused

Järgnevalt esitatakse kategooriate kirjelduste pikemad seletused:

1. **Probleemid sisselogimisega** – siia alla ei kuulu kirjad, mis kurdavad seda probleemi enne 5. oktoobrit 2015. Need kirjad lähevad kategooria „Korralduslikud küsimused“ alla.
2. **Nimi valesti** - kui inimene on nime muutnud või nimi on valesti.
3. **Korralduslikud küsimused** – nt millal hakkab kursus? Kas kursust lõpetades saab ka mingi tõendi?
4. **Ajapikenduse küsimine.**
5. **Kursusest loobumine.**
6. **Probleemid tarkvara ja riistvaraga** – nt inimesel kadusid kõik failid arvutist ära või kui Thonny jooksis kokku jne.
7. **Thonny installeerimine.**
8. **Muud probleemid** – nt teemavälised küsimused programmeerimise kohta.
9. **Moodle’ga seotud küsimused** – nt linnukesed, logifailid, vale faili esitamine.
10. **Inputi viga** – erinevad funktsiooni *input* kasutamisest tingitud vead.
    - 10.1. II kontrollülesanne. Nt automaatkontroll keelab siin ülesandes *input*’i kasutamise.
    - 10.2. III kontrollülesanne. Nt *input*’i pole programmis üldse kasutatud.
    - 10.3. VII kontrollülesanne. Nt ei teata funktsiooni skoobist – globaalse muutuja kasutamine funktsiooni sees.
11. **Tüübi viga.**
    - 11.1. II kontrollülesanne. Nt arvulisi ja tekstilisi osi on üritatud liita (arvule pole funktsiooni *str* rakendatud).
    - 11.2. III kontrollülesanne. Nt tingimuslause tingimuses võrreldakse sõne ja arvu.
    - 11.3. V kontrollülesanne. Nt kasutaja sisestusest saadud arv on muudetud ujukomaarvuks, kuid automaatkontroll nõuab seal täisarvu.
    - 11.4. VII kontrollülesanne. Nt sisendit pole arvuks muudetud, kui seda tahetakse tingimuslause päises teise arvuga võrrelda.
12. **Muutuja viga.**
    - 12.1. Kui programmis esineb defineerimata muutuja.

12.2. II kontrollülesanne. Nt muutujatel ei ole automaatkontrolli poolt nõutavad nimed või mitmel erineval muutujatel on samad nimed (üledefineeritud) jne.

12.3. VII kontrollülesanne. Nt kui funktsioonil ja lokaalsel muutujal on sama nimi.

13. **Süntaks ja treppimine –** puuduvad koolonid, sulud, punktide asemel on arvudes komad, valed jutumärgid jne + valed taanded.
14. **Väljastamine** – nt kui väljastatakse midagi muud kui automaatkontroll soovib või väljastamine on puudu.

14.1. VI kontrollülesanne. Väljastamine nõuab täpset sõnastust, „On Eesti Isikukood“ või „Ei ole Eesti isikukood“.

14.2. Teised kontrollülesanded ei nõua nii täpset väljastust. Nad nõuavad lihtsalt, et väljastus sisaldaks õiget vastust. Väljastuses võib esineda ka muid sõnu.

15. **Ümardamine** – igasugused ümardamisega seotud vead.

15.1. Nt VII kontrollülesandes tuli funktsiooni sees ümardada. Siia kuulub ka meetodi *ceil* rakendamine.

16. **Tsükkel või tingimuslause** – kui tingimuslause või tsükli päises on loogika kuidagi valesti (siia ei kuulu erinevate tüüpide võrdlemine – see on tüübiviga). Nt avaldises kasutatakse valet loogilist operaatorit.

16.1. III kontrollülesande tingimuslauses olev tingimus on poolikult kirja pandud.

16.2. V kontrollülesanne. Tsükli piiramine suvalise suure arvuga.

16.3. VI kontrollülesanne. Tingimuslause päises ei osata regulaaravaldist õigesti kasutada.

17. **Esimese sümboli valimine**

17.1. III kontrollülesandes ei osatud isikukoodi esimest sümbolit [0] abil valida.

18. **Meetodi kasutamine** – ülesande jaoks vajaliku meetodi kasutamata jätmine või selle valesti kasutamine.

18.1. IV kontrollülesanne. Nt kasutatakse meedoit *capitalize* valesti (nt ilma sulgudeta).

19. **Sisendi vale järjekord** – automaatkontroll nõuab, et kasutaja käest küsitaks andmeid kindlas järjekorras.
20. **Arvutusviga** – erinevad arvutusvead.
    20.1. Nt tehete järjekord (siia ei kuulu ümardamine ega tsükliga seotud arvutused).
    20.2. Nt VII kontrollülesandes pole funktsiooni sees arvutatud.
21. **Loenduri suurendamine** – nt tsüklis on loendurit vale sammu võrra suurendatud või loendurile on pandud vale algväärtus.
22. **Summa arvutamine** – nt V kontrollülesandes ei osatud tsüklit kasutades arvutada 1 + 2 + 3 + 4 + 5.
23. **Loenduri ja summa vastupidine järjekord.**
24. **Regulaaravaldis valesti** – nt VI kontrollülesanne. Kui regulaaravaldise loogika on paigast ära.
25. **Regulaaravaldis puudu** – nt VI kontrollülesandes pole regulaaravaldist üldse kasutatud.
26. **Vajalik moodul importimata** – nt VI kontrollülesandes on moodul „Re“ importimata.
27. **Funktsiooni argument puudub või seda pole funktsioonis kasutatud.**
28. **Funktsiooni ei kutsuta välja või seda tehakse valesti**.
    28.1. Nt VII kontrollülesandes kutsutakse funktsioon välja ilma sulgudeta või vajaliku argumendita.
29. **Uue faili tegemine.**
30. **Avastatud vead –** kui osalejad leidsid materjalidest vigu.

IV. Juhend andmetabeli täitmiseks

Siin on juhend, mis koosneb punktidest, mida tuleks programmerimisvigade ja organisatoorsete teavituste andmetabelisse märkimiseks järgida. Juhend koosneb üheksast punktist. Soovituslik oleks kasutada kasutajatoe keskkonda Freshdesk, mis koondab kirjade ahela ühe pileti (ingl k *ticket*) alla.

Järgnevalt on loetletud sammud, millest peaks andmetabeli koostamisel lähtuma:

1. Läbi tuleb vaadata kõik kontrollitava pileti all olevad e-kirjad.
2. Iga programmeerimise vea ja organisatoorse teavituse kohta e-kirjas tuleb teha andmetabeli õigesse lahtrisse (kategooriasse) märge „I“. Kategooriate pikema seletuse leiab lisadest (vt Lisa III).
3. Kui programm on failina kaasa pandud või e-kirja kopeeritud, siis tuleb kõik vead üles märkida, olenemata sellest, kas abistaja (kirjale vastaja) on antud veale osutanud.
4. Kui e-kiri ei sisalda kaasa pandud programmi, siis tuleb vead vastavalt juhendaja poolt antud vihjetele märkida.
5. Kui inimesel on programmis mingid osad puudu, mida automaatkontroll kindlasti nõuab, et ülesanne arvestatud saaks (näiteks väljastamine on puudu), siis tuleb ka see üles märkida.
6. Kui samalaadi viga on ühe e-kirja jooksul tehtud mitu korda, siis see läheb kirja ikkagi ühe veana. Ühes piletis võib sama laadi viga aga esineda mitu korda, sest ühes piletis võib olla mitu e-kirja.
7. Nädala lõputestis esinevad vead tuleb kategoriseerida ainult vastava testi küsimuse-numbri järgi.
8. Süntaksivigade märkimise korral tuleb lugeda, millele abistaja on osutanud. Põhjuseks on see, et mõned osalejad kopeerisid oma programmikoodi e-kirja ja sellise kopeerimise tõttu võivad erinevad vead sisse tulla (näiteks treppimine kaob ära).
9. Mingit konkreetset probleemi lahendades tuleks võtta ette ka ülesande tekst, et ükski täpset sõnastust vajav viga (näiteks täpset sõnastust vajav ekraanile väljastamine) ei jääks avastamata.

V. Programmeerimisvigade kohta käivate märgete võrdlus

Töö autor ja eksperdid tegid programmeerimisvigade kategoriseerimisel andmetabeli lahtritesse märkeid. Antud tabelites asendati tabelite parema loetavuse huvides konkreetse tabeli lahtritesse tehtud märked nende märgete hulgale vastava arvuga.

Programmeerimisvigade kohta käivate märgete võrdluse tabeli leiab järgneva lingi alt:

https://docs.google.com/spreadsheets/d/1NOj5mGcbSueRn8r0ccPkZKtFTuoI_M9jlVVbuW0VIZw/edit#gid=1052187452

VI. Kattuvusprotsendi arvutamine

Järgnevalt selgitatakse bakalaureusetöös kasutatud kattuvusprotsendi arvutamise meetodit. Esmalt tuuakse näide, mille alusel loeti käesolevas töös bakalaureusetöö autori ja ekspertide poolt programmeerimisvigu tähistavad märked kattuvateks ja mille alusel mittekattuvateks.

Töö autor ja eksperdid tegid programmeerimisvigade kategoriseerimisel andmetabeli lahtritesse märkeid. Antud näites (vt tabel 2) asendati tabeli parema loetavuse huvides selle tabeli lahtritesse tehtud märked nende märgete hulgale vastava arvuga. Siin toodud näide on töö autori poolt välja mõeldud.

Tabel 2. Märkijate poolt tehtud III kontrollülesande märgete võrdlus,

III kontrollülesanne							
	Tsükkel või tingimuslause	Esimese sümboli valimine	Muutuja viga	Süntaks ja treppimine	Inputi viga	Tüübi viga	Väljastamine
Esimene märkija		2					2
Teine märkija		1	1				2

Kõiki tehtud märkeid vaadeldi eraldi ja ainult selle kategooria raames, kus see konkreetne märge asetseb. Märge loeti kattuvaks, kui mõlemad märkijad olid selle märke samamoodi kategoriseerinud. Vastasel juhul loeti märge mittekattuvaks.

Eeltoodud tabelis on märkeid tehtud kolme kategooriasse. Vaatleme esmalt kategooriat „Esimese sümboli valimine“. Esimene märkija tegi selle kategooria alla kaks märget ja teine märkija ühe märke. Selle kategooria puhul loetakse üks märge kattuvaks ja üks märge mittekattuvaks. Mittekattuv märge tuleb siit selle pärast, et esimene märkija on vastavasse kategooriasse ühe märke rohkem teinud kui teine märkija. Seega puudub esimese märkija poolt tehtud märkel nii-öelda teise märkija sama kategooria lahtrist paariline.

Järgmisena vaadeldakse kategooriat „Muutuja viga“. Esimene märkija ei teinud selle kategooria alla ühtegi märget ja teine märkija tegi ühe märke. Tehtud märge loetakse mittekattuvaks.

Viimasena vaadeldakse kategooriat „Väljastamine“. Mõlemad märkijad tegid siin kaks märget. Tehtud märkeid loetakse samuti kattuvateks.

Kattuvusprotsendi arvutamiseks liideti kattuvad märked kokku ning jagati kattuvate ja mittekattuvate märgete summaga. Selle näite puhul kattusid kolm märget ning kaks märget ei kattunud. Seega on selle tabeli kattuvusprotsendiks 3/5 ehk 60%.

VII. Korduvalt esinenud vead

Järgnevalt esitatakse teemade kaupa korduvalt esinenud vead.

Üldised vead

1) Treppimine on paigast ära.
2) Valede sulgude kasutamine #255.

Muutujad, andmetüübid (kontrollülesanne II)

1) Kasutatakse valesid muutujanimesid:
   a) Automaatkontroll nõuab muutujaid „aasta“ ja „loom“, kuid inimene on kasutanud teistsuguse nimega muutujaid #69.
2) Automaatkontroll nõuab, et väljastatavas (rakendatakse funktsiooni *print*) lauses kasutataks muutujaid „aasta“ ja „loom“. Leidus mitu näidet, kus inimene oli loonud mingi uue muutuja ja pannud seal õigeid muutujaid kasutades kirja kohe terve lause.
   a) #131

   aasta = 2015
   loom = “metssiga”
   lause = str(aasta) + “. aasta loomaks on” + loom
   print(lause)

3) Kasutatakse funktsiooni *input*, kuigi automaatkontroll ei luba seda konkreetses ülesandes kasutada. #47
4) Erinevat tüüpi liikmeid üritatakse üheks sõneks kokku liita:
   a) #47

   aasta = 2015
   loom = "metssiga"
   print(aasta)
   print(loom)
   print(aasta + ". loomaks on " + loom)

5) Tüübi viga väljastamisel:
   a) #47

   aasta = 2015

loom = “metssiga”

print(aasta + “. aasta loomaks on” + metssiga)

6) Topelt viitamisega muutujad:
   a) #50, #85

a = aasta = 2015

b = loom = “metssiga”

7) Samanimelised muutujad:
   a) #53

a = “aasta”

a = str(2015)

b = “loom”

b = “metssiga”

8) Muutujad on jutumärkides.
9) Muutujaid pole üldse kasutatud #83.
10) Väljastamine on täiesti puudu #105.
11) Väljastamisel kasutatakse defineerimata muutujaid #105.
12) Ei osata kasutada funktsiooni *print*:
   a) #105

print = (aasta) #pole võrdusmärki ehk ei omastata midagi.

Valikulause/tingimuslause (kontrollülesanne III)

1) Tingimuslause päises võrreldakse erinevaid tüüpe:
   a) #179

```
id = input("Sisesta isikukood:  ")
arv = id[1]
if arv == 1 or arv == 3 or arv == 5 or arv == 7 :
        print("he")
```

2) Kui inputist saadud sõnest on ainult esimene sümbol vaja välja pookida, siis alustatakse loendamist 1-st, mitte 0-st #179.
3) Tingimuselause päise lõpust puudub koolon #225.

4) Tingimuslause päises olevate elementide võrdlemiseks on kasutatud ainult ühte võrdusmärki:
   a) #225

```
if ik[0] = “1” or ik[0] = “3”:
    print(“He”)
```

5) Tingimuslause päises on kasutatud valet operaatorit:
   a) #226

```
if ik[0] == “1” and ik[0] == “3”:
    print(“He”)
```

6) Tingimuslause päises esineb ainult üks pikalt väljakirjutatud tingimus:
   a) #226, #204

```
if ik[0] == “1” or “3”:
    print(“He”)
```

Sõned (kontrollülesanne IV)

1) Meetodi rakendamine sellele järgnevate sulgudeta:
   a) #225

```
“eesnimi”.capitalize
“perenimi”.capitalize
```

2) Meetodi rakendamata jätmine (IV kontrollülesande puhul oli selleks meetodiks *capitalize*):
   a) #265

```
eesnimi = “Oskar”
perenimi =”Ohakas”
print(eesnimi + “ “ + perenimi)
```

3) Ühe meetodi samaaegne rakendamine mitmele erinevale muutujale:
   a) #228 – meetod mõjub siin ainult esimesele muutujale

```
print(“Tere, ”, (eesnimi + “ “ + perekonnanimi).capitalize())
```

Tsükkel (kontrollülesanne V)

1) Tsükli kehas ei suurendata loendurit #390.
2) Tsükli tingimuses võrreldakse omavahel arvu ja sõne:

```
arv = input("Sisesta arv:")
i = 0
summa = 0
while i <= arv:
    summa += i
    i += 1
```

3) Summa salvestamine defineerimata muutujasse:
   a) #559

```
arv = int(input("Sisesta arv:"))
kordi = 0
while kordi < arv:
        vastus = arv + kordi
        kordi = kordi + 1
```

4) Tsükli kehas on vale tehete järjekord (nt esimesena peaks toimuma summa arvutamine ja alles siis loenduri suurendamine).
5) Tsükli päises kasutatakse valesid muutujaid:
   a) #473

```
arv = input()
summa = 0
i = 1
while i <= summa:
```

6) Loenduril on suvaline algväärtus:
   a) #400

```
i = 5
while i < 16:
        print(i)
        i = i + 1
```

7) Tsükli päises olevas tingimuses on suvalise suurusega piirang #400 (vaata eelmist näidet).
8) Tsüklis kehas olevat loendurit suurendatakse rohkem kui ühe sammu võrra korraga:
   a) #404

```
i=0
while i != sisend:
    i += 2
    summa = summa + i
print(summa)
```

Regulaaravaldis (kontrollülesanne VI)

1) Koostatud regulaaravaldised sisaldavad ettemääratust vähem või rohkem sümboleid:
   a) #390

```
if re.match("^[1-6][0-9]{1}$", isikukood):
print("On Eesti isikukood")
```

2) Isikukoodi esimene number peab jääma vahemikku 1–6:
   a) #351

```
klapib = re.match("[0-9]{11}", isikukood)
if klapib:
    print("On Eesti isikukood")
```

3) Isikukoodi 10 viimast numbrit peavad jääma vahemikus 0–9:
   a) #404

```
if re.search("[1-6]{11}" ("sisesta isikukood")):
        print("On Eesti isikukood")
```

4) Funktsiooni *search* kasutades peab regulaaravaldis sisaldama sümboleid ^ ja $, vastasel juhul loetakse rohkem kui 11 sümbolit sisaldav isikukood õigeks. Vaata eelmist näidet.
5) Funktsiooni *search* argumente pole komaga eraldatud ja teine argument on midagi muud kui regulaaravaldisega võrreldav isikukood. Vaata üle-eelmist näidet.
6) Õige moodul on importimata.

Funktsioonid (kontrollülesanne VII)

1) Funktsiooni ei osata välja kutsuda – väljastatakse lihtsalt funktsiooni nimi, jättes kasutamata nii sulud kui ka argumendi:
   a) #580

```
def jalanõude_suurus(suurus):    #Sellise funktsiooni puhul
...
print(str(jalanõude_suurus))  #Selline väljakutse
```

2) Programmis on nii samanimeline funktsioon kui ka muutuja:
   a) #580

```
def jalanõude_suurus(suurus):
jalanõude_suurus = round((suurus + 1,5) * 3 / 2)
        return jalanõude_suurus
```

3) Ei olda teadlikud funktsiooni skoobist – funktsiooni väliselt üritatakse funktsiooni sisse midagi anda (mitte funktsiooni argumendi kaudu):
   a) #613

```
def jalanõude_suurus(suurus):
valem = ((round(jalalaba_pikkus_cm + 1.5) * 3 / 2))
    return valem
print("Sisesta jalalaba_pikkus_cm")
jalalaba_pikkus = float(input())
```

4) Funktsiooni sees ei ümardata:
   a) #540

```
def jalanõude_suurus(suurus):
        j_suurus = ((suurus + 1.5) * 3 / 2)
        return j_suurus
suurus = float(input("Sisesta suurus: "))
print(round(jalanõude_suurus(suurus)))
```

5) Funktsiooni argumenti ei kasutata kuskil:

a) #613

```
def jalanõude_suurus(suurus):
        valem = ((round(jalalaba_pikkus_cm + 1.5) * 3 / 2))
return valem
```

VIII. Juhend murelahendaja koostamiseks

Siin on juhend, mis koosneb punktidest, mida tuleks murelahendaja koostamisel MOOCide „Programmeerimise alused“ ja „Programmeerimisest maalähedaselt“ puhul järgida. Juhendi teisest poolest leiab ka murelahendaja ülesehituse kohta käivat teavet.

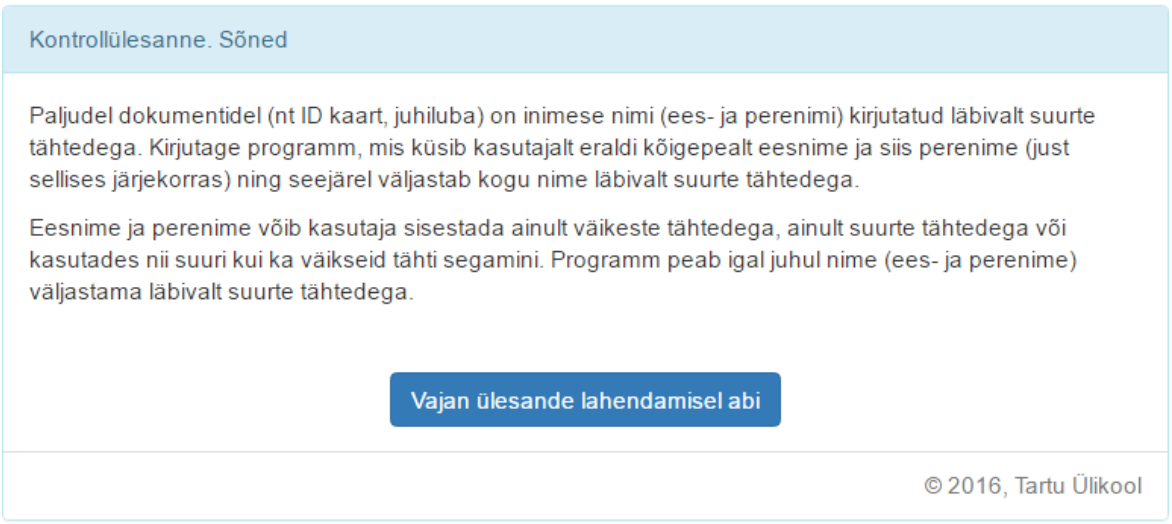
Soovitused murelahendaja koostamiseks:

1. Lahendada ülesanne tuginedes materjalides õpetatavale.
2. Jaotada koostatud programm mõtteliselt väiksemateks hallatavateks osadeks.
3. Vihjed esitatakse järk-järgult, sest nii on neid koostajatel mugavam hallata ja tekst on lugejatele paremini hoomatav.
4. Iga mõttelise osa kohta tuleb koostada vähemalt üks küsimus ja selle küsimuse juurde käiv vastus, kasutades Murelahendaja keskkonda. Küsimused ja vastused lähtuvad järgmistest põhimõtetest:
   a) küsimuste ja vastuste loomisel tuleb toetuda analüüsi käigus selgunud korduvalt esinenud vigadele (vt Lisa VII)
   b) funktsiooni, meetodi või mõne muu programmeerimiskonstruktsiooni kohta käiva vihje puhul tuleb lisaks tekstilisele seletusele vastuse leheküljele lisada ka koodi kujul olev näide.
5. Küsimustele ja vastustele tuleb panna järjekorranumbrid, et neid oleks tihedas Murelahendaja keskkonnas lihtsam eristada ja vajadusel ka ümber tõsta.
6. Murelahendaja keskkonna poolt loodud link tuleb lisada kursuse materjalide (näiteks iganädalaste kohustuslike kontrollülesannete) juurde.
7. Kursuse käigus laekunud e-kirjade alusel tuleb parandada/muuta murelahendajas esinevaid küsimusi või vastuseid.

Murelahendaja ülesehitus

Järgnevalt kirjeldatakse murelahendajate üldist ülesehitust.

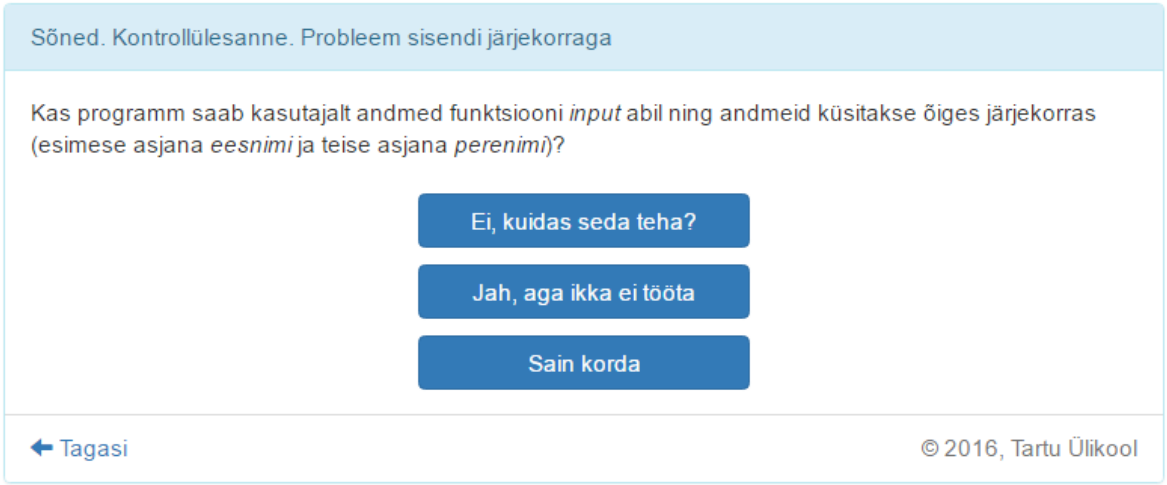
Murelahendaja esimesel leheküljel (vt joonis 11) on konkreetse ülesande kirjeldus ja esimese vihjeni viiv nupp „Vajan ülesande lahendamisel abi“.



Joonis 11. IV kontrollülesande kirjeldus

Iga murelahendaja küsimuse juures (vt joonis 12) on 3 valikuvarianti:

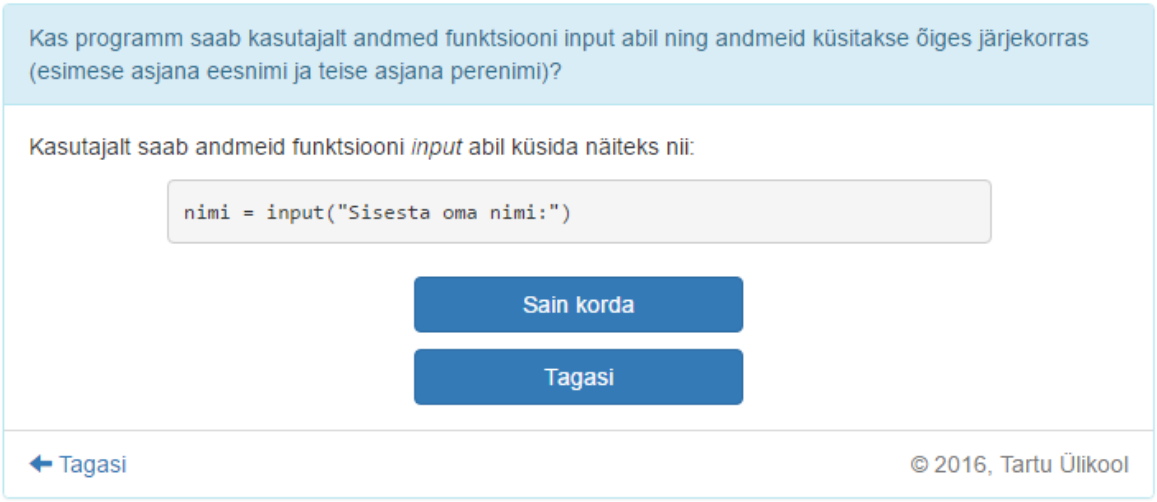
- „Ei, kuidas seda teha?“;
- „Jah, aga ikka ei tööta“;
- „Sain korda“.



Joonis 12. IV kontrollülesande murelahendaja küsimus.

Klõpsates nupule „Ei, kuidas seda teha?“ avaneb uus lehekülg, kus on küsimusele vastatatud (vt joonis 13) programminäidet kasutades. Programminäide erineb kuruse kodulehel olevatest põhimaterjalidest, et abiotsijal oleks võimalik näha rohkem näiteid. Lisaks vastusele on sellel leheküljel veel kaks nuppu – „Sain korda“ ja „Tagasi“. „Tagasi“-nupule

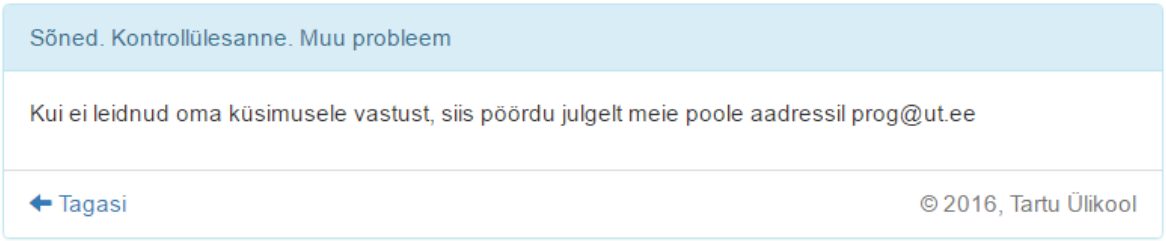
vajutades suunatakse inimene küsimusevaate juurde tagasi. Nupust „Sain korda“ räägitakse allpool olevas lõigus.



Joonis 13. IV kontrollülesande vastus.

Klõpsates küsimuse juures nupule „Jah, aga ikka ei tööta“ antakse osalejale ette uus küsimus, mis vihjab uuele võimalikule probleemile.

Kui murelahendaja ei suuda osaleja probleemi lahendada, siis iga murelahendaja viimasel leheküljel julgustatakse osalejaid e-kirja vahendusel korraldajate poole pöörduma (vt joonis 14). Soovitus korraldajatele kirjutada antakse alles murelahendaja viimasel leheküljel, et probleemi käes vaevlevad inimesed läbiksid esimese asjana murelahendaja ja pöörduksid korraldajate poole alles viimases hädas.



Joonis 14. Soovitus korraldajate poole pöörduda.

Murelahendaja keskkond kogub lehekülje külastamise kohta andmeid. Selleks, et saada murelahendaja kasulikkuse kohta informatsiooni, on iga küsimuse ja vastuse juures nupp „Sain korda“. Vastav nupp on lisatud iga küsimuse ja vastuse juurde eelkõige seetõttu, et inimene peaks kursuste korraldajatele tagasiside andmiseks võimalikult vähe lisaliigutusi tegema.

IX. Metoodika rakendamise lühikokkuvõte

Metoodika rakendamise järjekord kokkuvõtlikul kujul:

1. kirjade esmane läbivaatamine ja veakategooriate kirjelduste loomine;
2. andmetabeli täitmise juhendi loomine;
3. andmetabeli täitmise juhendi ja kategooriate kirjelduste kooskõlastamine juhendajatega;
4. andmetabeli loomine ja täitmine;
5. andmetabelis olevate andmete kontrollimine;
6. andmetabeli täitmise juhendi ja kategooriate kirjelduste arusaadavuse kontrollimine;
7. korduvalt esinenud vigade väljaselgitamine andmetabeli põhjal;
8. kontrollülesannete kohta käiva statistika uurimine;
9. murelahendaja koostamise juhendi loomine;
10. murelahendaja koostamise juhendi kooskõlastamine juhendajatega;
11. murelahendajate loomine;
12. murelahendajate linkide postitamine e-kursuste materjalide alla;
13. murelahendajate täiendamine Murelahendaja keskkonna poolt kogutud statistika ja abiliini laekunud e-kirjade alusel.

X. Murelahendajad

Siit leiab bakalaureusetöö autori poolt koostatud murelahendajad:

MOOCi „Programmeerimisest maalähedaselt“ murelahendajad

II kontrollülesanne: http://progtugi.cs.ut.ee#/ts/56d9fa47c1bd5ad72a086d81/

III kontrollülesanne: http://progtugi.cs.ut.ee#/ts/56e354907d82334e4a72fd00/

IV kontrollülesanne: http://progtugi.cs.ut.ee#/ts/56e3e65e7d82334e4a72fe4a/

V kontrollülesanne: http://progtugi.cs.ut.ee#/ts/56ec34d67d82334e4a735daf/

VI kontrollülesanne: http://progtugi.cs.ut.ee#/ts/56ec6be37d82334e4a736063/

VII kontrollülesanne: http://progtugi.cs.ut.ee#/ts/56f6382aa2b8b3bf6e8b968f/

MOOCi „Programmeerimise alused“ murelahendajad

Ülesanne 1.2. Rasvatihane: http://progtugi.cs.ut.ee#/ts/568b994056213e0300c83c8d/

Ülesanne 1.3. Astendamine: http://progtugi.cs.ut.ee#/ts/56912c301224608073f64398/

Ülesanne 1.4a. Nädala ajakulu: http://progtugi.cs.ut.ee#/ts/56913e1e1224608073f6444c/

Ülesanne 1.4b. Kiiruseületamise trahv:
http://progtugi.cs.ut.ee#/ts/569170a6b1e1196c0a50f6b3/

Ülesanne 2.1. Jäätumine: http://progtugi.cs.ut.ee#/ts/56997ce0b2d45de61a379f9b/

Ülesanne 2.2. Tribüün: http://progtugi.cs.ut.ee#/ts/5699f936b2d45de61a379feb/

Ülesanne 2.3. Allveelaev: http://progtugi.cs.ut.ee#/ts/569d5e26aeb9998220318b12/

Ülesanne 2.4. Leedu perenimed: http://progtugi.cs.ut.ee#/ts/569ad7a76f7f2b261ee618cb/

Ülesanne 2.5a. Pulss: http://progtugi.cs.ut.ee#/ts/569c1779aeb9998220318a51/

Ülesanne 2.5b. Buss: http://progtugi.cs.ut.ee#/ts/569c2406aeb9998220318a78/

Ülesanne 3.1. Suured tähed: http://progtugi.cs.ut.ee#/ts/56a0b1181f8cb8a227e43015/

Ülesanne 3.2. Paaritute arvude summa:
http://progtugi.cs.ut.ee#/ts/56a160cf1f8cb8a227e4342a/

Ülesanne 3.3. Täring: http://progtugi.cs.ut.ee#/ts/56a20d071f8cb8a227e434e5/

Ülesanne 3.4a. Summa: http://progtugi.cs.ut.ee#/ts/56a212341f8cb8a227e43537/

Ülesanne 3.4b. Vabavisked: http://progtugi.cs.ut.ee#/ts/56a222e01f8cb8a227e43586/

Ülesanne 3.4c. Male: http://progtugi.cs.ut.ee#/ts/56a4bd331f8cb8a227e43ec1/

Ülesanne 4.1. Mopeedid: http://progtugi.cs.ut.ee#/ts/56acee1b5a7c5f2b396374ed/

Ülesanne 4.2. Paaritute arvude summa: http://progtugi.cs.ut.ee#/ts/56acfe6c5a7c5f2b39637523/

Ülesanne 4.3. Konto väljavõte: http://progtugi.cs.ut.ee#/ts/56ad4c065a7c5f2b39637684/

Ülesanne 4.4a. Viktoriin: http://progtugi.cs.ut.ee#/ts/56addfb75a7c5f2b3963772b/

Ülesanne 4.4b. Kümnevõistlus: http://progtugi.cs.ut.ee#/ts/56ae1af05a7c5f2b396379de/

Ülesanne 4.4c. Tahvli juurde: http://progtugi.cs.ut.ee#/ts/56ae36f65a7c5f2b39637b7c/

Ülesanne 5.1. Tervitused: http://progtugi.cs.ut.ee#/ts/56b486e15a7c5f2b39639aaf/

Ülesanne 5.2. Teleri suurus: http://progtugi.cs.ut.ee#/ts/56b4a1b55a7c5f2b39639bd4/

Ülesanne 5.3. Müügiautomaat: http://progtugi.cs.ut.ee#/ts/56b4c37c5a7c5f2b39639cc1/

Ülesanne 5.4a. Kuupäev: http://progtugi.cs.ut.ee#/ts/56b5bdd25a7c5f2b3963a0a6/

Ülesanne 5.4b. Tervitused mõtisklustega: http://progtugi.cs.ut.ee#/ts/57232847086077310ae8f6c8/

Ülesanne 5.4c. Mündid: http://progtugi.cs.ut.ee#/ts/56b64f985a7c5f2b3963a5c1/

Ülesanne 6.1. Salakiri: http://progtugi.cs.ut.ee#/ts/56be48955a7c5f2b3963d39c/

Ülesanne 6.2. Päevik: http://progtugi.cs.ut.ee#/ts/56be48b35a7c5f2b3963d39d/

Ülesanne 6.3. Kalkulaator: http://progtugi.cs.ut.ee#/ts/56c11b4f5a7c5f2b3963e9a8/

Ülesanne 6.4a. Nimepäev: http://progtugi.cs.ut.ee#/ts/56bf02775a7c5f2b3963d533/

Ülesanne 7.1. Lipp või liiklusmärk: http://progtugi.cs.ut.ee#/ts/56c775742b32fb750a414552/

Ülesanne 7.2. Seadme paneel: http://progtugi.cs.ut.ee#/ts/56c782852b32fb750a4145c5/

Ülesanne 8.1a. Arvestusülesanne: http://progtugi.cs.ut.ee#/ts/56d44b0ec1bd5ad72a085750/

XI. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Kaspar Hollo** (sünnikuupäev: 21.11.1992),
(*autori nimi*)

1. Annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
   Programmeerimise e-kursusel osalejate küsimuste analüüs ja selle põhjal murelahendajate koostamine,
   (*lõputöö pealkiri*)

   mille juhendajad on Eno Tõnisson ja Tauno Palts
   (*juhendaja nimi*)

   1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace'i lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **09.05.2016**