TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Joosep Albre

Tartu Ülikooli kursuse “Programmeerimine” ülesannete automaatkontrollide uuendamine ja üleviimine keskkonda lahendus.ut.ee

Bakalaureusetöö (9 EAP)

Juhendaja: Tauno Palts, PhD

Tartu 2023

Tartu Ülikooli kursuse “Programmeerimine” ülesannete automaatkontrollide uuendamine ja üleviimine keskkonda lahendus.ut.ee

Lühikokkuvõte:

Tartu Ülikooli kursuse “Programmeerimine” osalejate arv on kasvutrendis ning seeläbi tõstab aine õppejõudude koormust kursuse ülesannete kontrollimise näol. Ülesannete ja nende kontrollide haldamise lihtsustamiseks koostati bakalaureusetöö raames kursuse ülesannetele automaatkontrollid. Koostamise jaoks kasutati lahendus.ut.ee keskkonna uut automaatkontollide defineerimise võimalust Test Specific Language (TSL). Nende koostamise protsess on ühtlasi ka esimene laiaulatuslikum TSL abil defineeritud automaatkontrollide kasutus - see on oluline selle võimaluse funktsionaalsusest ja valmidusest ülevaate saamiseks. Lisaks antakse töös ülevaade, miks on ülesannete lahendamine oluline ning kuidas automaatkontrollid saavad toetada programmeerimise õppimise ja õpetamise protsessi.

Võtmesõnad: programmeerimine, programmeerimise õppimine, programmeerimise kursus, automaatkontrollid

CERCS: P175 Informaatika, süsteemiteooria, S270 Pedagoogika ja didaktika

Modification of the automated assessment tests of the "Computer Programming" course at the University of Tartu and migration to lahendus.ut.ee environment

Abstract:

The number of students enrolled in the University of Tartu course "Computer Programming" is growing rapidly, thus increasing the workload of course staff in the form of assignment assessing. In order to simplify the management of the assignments and assessments, new automated assessment tests were created as a result of the thesis. Lahendus.ut.ee environment's new Test Specific Language (TSL) option of defining automated assessment tests was used for accomplishing this. This creation process was the first large-scale use of automated assignment tests defined in TSL and gave important insight into the functionality and readiness of the feature. Thesis also provides an overview of why task solving is important and how automated assessment tests can support the process of learning and teaching of programming.

Keywords: programming, learning programming, programming course, automated assessment tests

CERCS: P175 Informatics, systems theory; S270 Pedagogy and didactics

Sisukord

Sissejuhatus

Tänapäeval on programmeerimine ühel või teisel kujul esindatud kõikides eluvaldkondades. Selle tulemusena on programmeerimine ning selle õppimine muutunud populaarsemaks - seda trendi kinnitab ka Tartu Ülikooli kursust "Programmeerimine" läbivate üliõpilaste arvu pidev kasv [1-3]. Programmeerimisõppe nõudluse kasv põhjustab õppejõudude koormuse suurenemist, mille tulemusena tekib vajadus optimeerida kursuste ja selle ülesannete haldamist ja kontrollimist. Seda probleemi aitab lahendada kursusel automaatkontrollide kasutamine, mis vähendab õppejõudude koormust ülesannete kontrollimise ning hindamise arvelt [4].

Kursusel "Programmeerimine" seni kasutusel olnud automaatkontrollid on raskesti mõistetavad ning nende haldamine nõuab tehniliste oskuste olemasolu. Selle parandamiseks võetakse töös kasutusele automaatkontrollimise lahendus, kus testide defineerimiseks kasutatakse Test Specific Language[1] (edaspidi TSL) märgistuskeelt [5]. Selle kasutamine on toetatud Lahendus[2] keskkonnas, mis ühtlasi pakub ka lisavõimalusi kursuse hallatavuse parandamiseks. Bakalaureusetöö raames luuakse Lahendus keskkonda uus kursus ning defineeritakse TSL kasutades kursuse ülesannetele automaatkontrollid.

Ühtlasi on käesolev töö esimene laiaulatuslikum TSL abil defineeritud automaatkontrollide kasutus, mis annab ülevaate selle võimaluse funktsionaalsusest ning valmidusest laiemalt kasutusele võtmiseks. Automaatkontrollide loomisel leitud vead ja kitsaskohad on väärtuslik info TSL lahenduse arenguks ja töökindluse parandamiseks.

Töö on jaotatud neljaks suuremaks peatükiks. Esimene neist annab ülevaate teoreetilisest taustast - selgitatakse, miks on oluline programmeerima õppides lahendada ülesandeid ning kuidas automaatkontrollid toetavad programmeerimise õpingud. Järgmises peatükis kirjeldatakse automaatse kontrollimise võimalusi Tartu Ülikooli kursusel "Programmeerimine" ning süvenetakse eelmainitud TSL automaatkontrollide ülesehitusele. Metoodika peatükk kirjeldab Lahendus keskkonnas uue kursuse ning automaatkontrollide loomise protsessi ja selle käigus esinenud probleeme. Tulemuste peatükk annab ülevaate

[1] Deklaratiivne märgistuskeel, mis võimaldab kirjeldada automaatkontrollide olemust ja sisu.
[2] https://lahendus.ut.ee/

leitud vigadest, valminud kursusest ja automaatkontrollidest ning sõnastab võimalikud TSL edasiarendused.

1. Programmeerimise õpetamine kõrgkoolis

Selles peatükis antakse ülevaade programmeerimise olulisusest ning tutvustatakse Tartu Ülikooli kursust “Programmeerimine” ja selle ülesehitust. Lisaks selgitatakse automaatkontrollide olemust ning miks on mõistlik neid mainitud kursuse raames kasutada.

1.1 Programmeerimise õpetamise olulisus

Tehnoloogia kiire arengu tulemusena on programmeerimine ja selle õppimine muutunud aina olulisemaks. Seda mitte ainult arvutiteaduse ja informaatika kontekstis, vaid tänapäevaks on tehnoloogia ja programmeerimine põimunud pea kõikidesse eluvaldkondadesse nõnda suurel määral, et noortele tutvustakse programmeerimist ja selle konseptsioone juba enne kooli astumist [6].

Programmeerimise õppimine on väärtuslik, kuna arendab inimese kriitilist- ja loogilist mõtlemist. Maailma ühe edukaima tehnoloogiaettevõtte, Apple, kaasasutaja Steve Jobs on öelnud, et tema arvates peaks kõik inimesed õppima programmeerimist, kuna see õpetab neid mõtlema [7]. Tehnoloogiliste ülesannete puhul on tarvis esinenud probleem jagada väiksemateks ja kergemini hallatavateks tükkideks ning nende tükkide lahenduste kombineerimise tulemusena jõutakse lõpuks esialgse probleemi lahenduseni. Lisaks võib programmeerimise õppimist kõrvutada uue võõrkeele õppimisega - mitmed uuringud kinnitavad, et võõrkeele õppimine tõstab õpilaste akadeemilist võimekust [8].

Eesti tööturu kontekstis on IT- oskused kõrgelt hinnatud - Eesti Statistikaameti [9] andmetel on Eesti keskmine tarkvaraarendaja brutopalk 2022. aasta IV kvartali seisuga 3985 eurot kuus, kui samal ajal on üldine Eesti keskmine brutopalk sama perioodi lõikes 1775 eurot kuus [10]. Üle kahekordne palgaerinevus viitab, et programmeerimisoskuste omamine tõstab inimese väärtust tööturul.

1.2 Programmeerimise õpetamine Tartu Ülikoolis

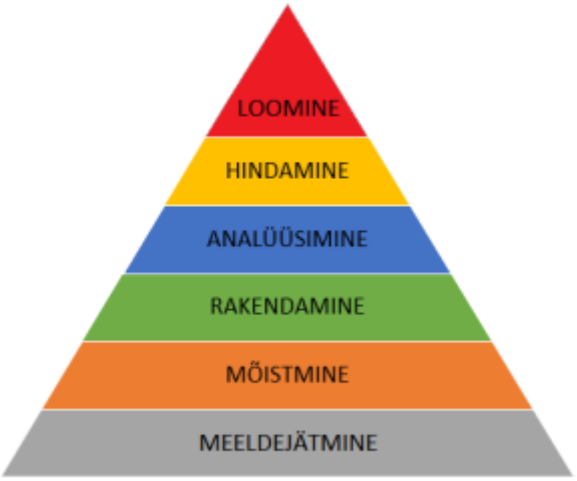
Tartu Ülikoolis on tihti üliõpilaste esimeseks kokkupuuteks programmeerimisega kursus “Programmeerimine” (LTAT.03.001) [1], kus õpitakse programmeerimiskeelt Python[3]. Aine maht on 6 ainepunkti, eristava hindamisega ning Tartu Ülikooli Õppeinfosüsteemi kohaselt on kursuse eesmärk “anda alusteadmised programmeerimise põhikonstruktsioonidest ning

[3] https://www.python.org/

esmased oskused algoritmide ja programmide koostamiseks". Kursus kestab 16 nädalat ning nende läbimise tulemusena üliõpilane tunneb ja oskab põhilisi programmeerimiskonstruktsioone, tunneb põhilisi andmetüüpe- ja struktuure ning oskab luua lihtsamale ülesandele vastavaid algoritme. Kursus "Programmeerimine" on eeldusaineks mitmetele programmeerimise kontekstis olulistele õppeainetele, nagu näiteks "Objektorienteeritud programmeerimine" ja "Programmeerimine II".

1.3 Ülesannete lahendamise olulisus programmeerimise õppimisel

Bloomi taksonoomia (ingl *Bloom's taxonomy*) on hierarhiline jaotus, mis koosneb õppimisega seotud kognitiivsetest protsessidest [11]. Mainitud taksonoomiat järgitakse tihti arvutiteaduses õppematerjalide koostamisel ja õpiväljundite defineerimisel [12]. Järjestatud madalamast tasemest on uuendatud taksonoomia osadeks: meeldejätmine, mõistmine, rakendamine, analüüsimine, hindamine ning loomine [13]. Joonisel 1 on kujutatud uuendatud Bloomi taksonoomia hierarhia, kus on näha kõik kuus kognitiivset protsessi ning nende paiknemine mainitud hierarhias.



Joonis 1. Bloomi taksonoomia kognitiivsete protsesside hierarhia [14]

Loomine on Bloomi hierarhias kõrgeimal kohal, kuna hõlmab endas kõiki teisi eelmainitud protsesse. Loomine seisneb õpitud teadmiste koondamisest ja nende kasutamisest uuel viisil, et luua uus funktsionaalne tervik. Bloomi taksonoomia loomise tasemele vastab programmeerimise õppimise kontekstis ülesannete lahendamine - tuleb hinnata probleemi ning rakendada olemasolevaid teadmiseid ja oskuseid, et luua töötav tervik ehk programm. Sellele toetudes saab järeldada, et programmeerimisülesannete lahendamine on efektiivne viis programmeerimise õppimiseks.

Bakalaureusetöö keskmes olev kursus “Programmeerimine” järgib Bloomi taksonoomia soovitusi, kuna kursuse peamine õppevorm on programmeerimisülesannete lahendamine. Järgmine peatükk annab ülevaate miks ja kuidas ülesannetele orienteeritud kursuste haldamist saab automaatkontrollide abil hõlbustada.

1.4 Automaatkontrollide olulisus kursusel “Programmeerimine”

Automaatkontrollid aitavad säästa õppejõudude aega ülesannete kontrollimise arvelt [4]. Tartu Ülikooli kursus “Programmeerimine” on väga populaarne õppeaine – Tartu Õppeinfosüsteemi andmetel oli mainitud kursuse põhiõppesse 2022. aasta sügisel registreerunud 467 õpilast [1]. Selle põhjal saab arvutada, et kui õppejõud peaksid 16 nädala materjali, kus iga nädal sisaldab umbes 6 ülesannet, käsitsi kontrollima, siis tuleks neil semestri jooksul manuaalselt kontrollida ligikaudu 45000 ülesannet - see on selgelt liialt ajakulukas ning ebatõhus lähenemine, mida automaatkontrollid aitavad vältida.

Oma tööst vigade leidmine ja nende parandamine on tõhus viis õpitu paremaks mõistmiseks ja kinnitamiseks [15]. Kodutööde ja praktikumide ülesannete puhul saavad õpilased automaatkontrollidelt kohest tagasisidet - töö esitamisel väljastab automaatkontroll tulemuse, andes teada, kas või kus esines lahenduses vigu. Tagasiside annab õpilastele mõista, millega eksiti ning millised teadmised vajaks täiendamist. Innustamaks õpilasi oma vigu otsima ning parandama, on neil võimalus teha lahendustes parandusi ning need uuesti hindamisele esitada.

Automaatkontrollid annavad õppejõule tagasisidet käsil oleva teema ja ülesannete kohta - kui mõni test osutub olema madala läbivusega, siis võib see vihjata, et ülesanne on liiga keeruline, midagi jäi ülesande püstituse juures segaseks või selle testi poolt kontrollitav aspekt on õpilaste jaoks segane. Selle info põhjal on õppejõul võimalik käituda vastavalt olukorrale - tehes parandused ülesande tekstis ja automaatkontrollis või näiteks praktikumis teemat põhjalikumalt korrates.

1.5 Automaatkontrollid

Automaatkontrollide all peetakse silmas tarkvara, mis võimaldab automaatselt hinnata konkreetse tarkvara, programmi või lähtekoodi kindlaid aspekte. Programmeerimise

õpetamise kontekstis kasutatakse automaatkontrolle, et kontrollida ning hinnata õpilaste estiatud programmide korrektsust.

Automaatkontrollide abil on võimalik kontrollida mitmeid erinevaid lahenduse aspekte - funktsionaalsust, lähtekoodi struktuuri ja programmi sooritust. Võimalik on kontrollida, kas koodi süntaks on korrektne ning kas kood kompileerub. Seda analüüsides on võimalik saada ülevaade koodi ülesehitusest ning probleemi lahendamisel kasutatud struktuuridest. Kui kood ei kompileeru, ei ole võimalik automaatkontrollil lahenduse funktsionaalsust testida. Funktsionaalsuse testimise juures kontrollitakse, et programm teeks täpselt seda, mida oodatakse. Funktsionaalsust testides on lisaks võimalik testida lahenduse sooritust, mõõtes lahenduse kiirust ja mälukasutust. [4, 16-17]

Programmi analüüsimise meetodite poolest jagunevad automaatkontrollid kaheks - staatiline analüüs ning dünaamiline analüüs**.** Staatilised automaatkontrollid testivad lahendust ilma seda kompleerimata ja jooksutamata - kontrollimine toimub analüüsides lähtekoodi. Kuna see ei vaja kompileerimist, saab staatiliselt analüüsida ka programme, mille funktsionaalsus on vigane või mis ei kompileeru. Enamjaolt kasutatakse kontrollimiseks tekstianalüüsi mudeleid, mis kontrollivad lähtekoodi süntaksit ja seal kasutatud võtmesõnu. Dünaamilised automaatkontrollid jooksutavad kontrollitavat programmi ning hindavad selle funktsionaalsust analüüsides programmi tulemust ja väljundit - kontrollitakse, kas programm lahendab püstitatud probleemi, ning kas teeb seda korrektselt. Enamjaolt tehakse seda analüüsides programmi funktsioonide poolt tagastatavaid väärtuseid ja väljundit ning võrreldes seda oodatud väärtuste ja väljundiga**.** [4, 16-17]

1.6 Automaatkontrollide puudujäägid

Automaatkontrollid säästavad programmeerimise õpetamisel aega ning vähendavad oluliselt õppejõudude koormust, kuid olukorras, kus on väike hulk ülesandeid ja oluline on koodi kvaliteet ja ülesehitus, võib manuaalne kontrollimine olla ratsionaalsem valik.

Automaatkontrollid kontrollivad lahenduse funktsionaalsust, kuid ei pööra tähelepanu koodi struktuurile ja loetavusele. Tarkvaraarenduses on lähtekoodi loetavus ja arusaadavus väga oluline faktor, mis võib mõjutada selle kvaliteeti ja hallatavust [18]. Seetõttu on oluline, et õpilased saaksid tagasisidet ka koodi loetavuse ja arusaadavuse kohta, kuid antud töö raames

koostatud automaatkontrollid seda ei võimalda. Selle kõrvalt on aga oluline märkida, et kursuse “Programmeerimine” eesmärk on selgitada programmeerimise põhitõdesid ning seetõttu koodi kvaliteedile ja loetavusele rõhumine ei ole prioriteet.

Automaatkontrollid säästavad kontrollimise arvelt küll õppejõudude aega, kuid nõuavad esmasel ülesseadmisel olulist ajalist panust**.** Eelneva seadistamise vajamine ühtlasi tähendab ka seda, et automaatkontrollide abil kontrollitavate lahenduste struktuur on osaliselt määratud, ehk õpilastele jääb vähem voli lahenduse isikupärastamiseks. Sel põhjusel pole võimalik automaatkontrolle kasutada näiteks isiklike projektide hindamiseks. Lisaks ajakulule on probleemiks ka asjaolu, et automaatkontrollide ülesseadmine võib olenevalt kasutatud raamistikust olla tehniliselt nõudlik ülesanne ning kõikidele õppejõududele ei pruugi see jõukohane olla. Koostatud automaatkontrollide loetavus ja hallatavus on üks oluline aspekt, mida käesoleva bakalaureusetöö väljundina parandatakse.

2. Automaatkontrollid Tartu Ülikooli kursusel “Programmeerimine”

Selles peatükis kirjeldatakse kursuse materjalide uude keskkonda üleviimise põhjuseid ning millised võimalused sellega ülesannete automaatseks kontrollimiseks kaasnevad. Tutvustatakse siiani kasutusel olnud automaatkontrollide ülesehitust ning süvenetakse uue lahenduse olemusele ja ülesehitusele.

2.1 Kursusel kasutatavad õppekeskkonnad

Töö koostamise hetkel on kursuse “Programmeerimine” materjalid jaotatud kahe erineva veebikeskkonna vahel - Courses[4] keskkonnas on kursuse teoreetiline õppematerjal ja programmeerimisülesannete püstitused ning Moodle[5] keskkonnas on ülesannete automaatkontrollid, teoreetilised testid, loengute materjalid ning hinnete ülevaade. Käesoleva töö raames viiakse ülesanded ning nende automaatkontrollid Lahendus keskkonda, kõrvaldades olukorra, kus ülesannete püstitused on ühes keskkonnas, kuid nende esitamine ning automaatkontrollid erinevas keskkonnas.

Peamine motivatsioon Lahendus keskkonda kolimiseks on soov regulaarselt uuendada ja vahetada kursusel “Programmeerimine” lahendatavaid ülesandeid. Nagu eelnevalt kirjeldatud, on hetkel ülesannete automaatkontrollide muutmine ja loomine tülikas protsess - selle leevendamiseks loodi Lahendus keskkonda uus automaatkontrollimist võimaldav moodul TSL (loe lähemalt alapeatükist 3.2), mis lihtsustab automaatkontrollide modifitseerimist ning loomist. Kursuse ülesannete välja vahetamisel on mõistlik ülesanne ja selle automaatkontroll salvestada, et oleks võimalus neid hiljem uuesti kasutada - hetkel aga puudub mugav lahendus selle saavutamiseks. Lahendus keskkond pakub võimaluse siduda ülesande ja selle automaatkontrolli üheks tervikuks ning talletada see kursuse materjalide väliselt ülesandekogusse, kus on see alati kättesaadav. Lisaks võimaldab see kasutada samu ülesandeid erinevate kursuste raames ilma, et peaks ülesande kirjeldusi või automaatkontrolle dubleerima.

[4] https://courses.cs.ut.ee/
[5] https://moodle.ut.ee/

2.2 Automaatkontrollide võimalused Lahendus keskkonnas

Töö koostamise hetkel on Lahendus keskkonnas toetatud järgnevad automaatkontrollide loomise võimalused:

1) Python Grader on 2014. aastal Tartu Ülikoolis loodud moodul, mille abil saab automaatselt kontrollida programmeerimisülesandeid [19]. Moodul arendati välja eesmärgiga lihtsustada ülesannete kontrollimsit ja hindamist sissejuhatavatel programmeerimiskursustel, mis kasutavad programmeerimiskeelt Python. Grader mooduli miinuseks on automaatkontrollide koostamiseks vajaminev tehniline oskus. Mooduli abil automaatkontrollide koostamiseks on tarvis piisavaid Pythoni teadmiseid ning omada arusaama Graderi võimalustest ja ülesehitusest. Automaatkontrollide ülesseadmine on kursuse õppejõudude jaoks raskendatud protsess, kuna nõuab aega ning vilumust.

2) Pildituvastuse moodul on Tartu Ülikoolis välja töötatud süsteem, mille abil on võimalik hinnata graafilise väljundiga ülesannete lahendusi. Esitatud lahenduse graafiline väljund koos kirjeldava märksõnaga saadetakse pildituvastusteenusele, mis tagastab tõenäosuse, et pildil kujutatu vastab antud märksõnale [20]. Bakalaureusetöö raames graafilise väljundi automaatset kontrollimist ei käsitleta, kuid selle olemasolu lisab põnevaid võimalusi kursuse ülesannete koostamisel.

3) TSL ehk Test Specific Language on Tartu Ülikoolis loodud deklaratiivne märgistuskeel[6], mille abil on võimalik defineerida automaatkontrolle [5]. TSL abil kirjeldatakse kontrollide sisu - antavad sisendid, oodatud väljundid, testi tüüp jt. parameetrid, ning selle kirjelduse põhjal genereeritakse automaatkontroll, mis lahendusi hindab. Töö koostamise hetkel pole TSL veel lõplikult valmis ning on pidevalt uuenemas.

Kursuse “Programmeerimine” senised automaatkontrollid on koostatud kasutades Python Grader moodulit. Bakalaureusetöö raames koostatud automaatkontrollid on loodud kasutades TSL märgistuskeelt. Järgnevalt tutvustatakse TSL automaatkontrollide täpsemat ülesehitust ja toimimist Lahendus keskkonnas.

[6] Dokumendisiseste vormingusiltide süsteem sisu struktuuri ja esitusviisi määramiseks [21]

2.3 TSL automaatkontrollide ülesehitus

TSL abil koostatud automaatkontrollid on defineeritud failis, kus YAML[7] formaati järgides kirjeldatakse, millistest testidest automaatkontroll koosneb. Selle kirjelduse põhjal genereeritakse jooksutatavad testid programmeerimiskeeles Python, mis teostavad lõpliku kontrolli [5]. TSL abil koostatud automaatkontrollide toimimiseks on Lahendus keskkonna redigeerimisaknas neli faili: (a) *tsl.yaml,* (b) *generated_0.py,* (c) *evaluate.sh* ja (d) *meta.txt*. Nimetatud failide olemuste kirjeldused on järgmised:

a) *tsl.yaml* - YAML fail, kus on defineeritud automaatkontrolli testide olemus ja sisu.
b) *generated_0.py* - Pythoni fail, mis sisaldab *tsl.yaml* põhjal genereeritud automaatkontrolli teste.
c) *evaluate.sh* - skript, mis käivitab automaatkontrolli.
d) *meta.txt* - tekstifail, mis sisaldab metaandmeid genereeritud testide kohta.

Automaatkontrollide koostamisel peab redigeerima vaid faili *tsl.yaml* - ülejäänud failid on vajalikul kujul initsialiseeritud või automaatselt genereeritud *tsl.yaml* põhjal.

Redigeeritav fail *tsl.yaml* koosneb kahest osast - metaandmetest ja testidest. Faili alguses kirjeldatud metaandmed sisaldavad erinevaid parameetreid antud automaatkontrolli kohta, sealhulgas milliseid Pythoni ja TSL versioone kasutatakse testide genereerimisel. Automaatkontroll võib sisaldada palju erinevaid teste, mille abil on võimalik katta kõigi vajalike funktsionaalsuste ja kasutusjuhtumite kontrollid. Joonisel 2 on toodud näide ülesande automaatkontrollist, kus on näha faili metaandmed ning defineeritud testid.

[7] inimloetav andmete jadastuse keel [21]

```
language: python3
validateFiles: true
tslVersion: 1.0.0
requiredFiles:
  - submission.py

tests:
  - type: program_execution_test
    name: Aastatulu 14400 ja 25200 euro vahemikus
    id: 1
    points: 15
    standardInputData:
      - 16825
    genericChecks:
      - checkType: ALL_OF_THESE
        expectedValue:
          - 6085.83
        beforeMessage: Kontrollib, kas programm väljastab korrektse
maksuvaba tulu
        passedMessage: Programm väljastas korrektse maksuvaba tulu
        failedMessage: Programm ei väljastanud korrektset maksuvaba tulu
    exceptionCheck:
      mustNotThrowException: true
      beforeMessage: Kontrollib, et programm ei viskaks erindit.
      passedMessage: Programm ei visanud erindit.
      failedMessage: Programm viskas erindi.
```

Joonis 2. Näide failist *tsl.yaml* ning seal sisalduva testi ülesehitusest.

Selgitamise eesmärgil on joonisel 2 välja toodud vaid üks test - Lahendus keskkonnas sisaldab kujutatud automaatkontroll hulga rohkem teste. Järgnev peatükk selgitab lähemalt milline on testide ülesehitus ja mida need sisaldavad.

2.4 TSL testide ülesehitus

Kõik TSL abil kirjeldatud testid on defineeritud järgnevate väljade abil:

a) Testi tüüp
b) Testi nimi
c) Testi id
d) Testi eest saadavad punktid
e) Testi tüübi kohustuslikud väljad
f) Mittekohustuslikud väljad

Igale testile on kohustuslik anda tüüp, mis määrab selle testi poolt kontrollitava aspekti. Testi tüüp määratakse võtmesõnage *type*, ning jagunevad kontrollitava skoobi alusel kaheks -

tüübid, mis kontrollivad kogu programmi tööd ning tüübid, mis kontrollivad kindlat funktsiooni ja selle tagastatavat väärtust. Käesoleva töö koostamise hetkel on 19 erinevat tüüpi teste, mis on kõik loetletud tabelis 1.

Testidele on kohustuslik määrata ka nimi, id ja testi eest saadavate punktide arv. Need määratakse vastavalt võtmesõnadega *name*, *id* ja *points*. Nimi kuvatakse kontrollimise tulemuse väljastamisel õpilasele ning viitab, millist lahenduse aspekti test kontrollib. Id on unikaalne identifikaator, mille alusel kasutajaliides teste eristab. Iga testi juures määratakse punktisumma, mis selle testi edukas läbimine annab - ühe automaatkontrolli testide koondsumma on 100 punkti.

Olenevalt testi tüübist kaasnevad mõned väljad, mida antud tüübi juures on võimalik väärtustada. Mõned neist on kohustuslikud - funktsiooni kontrollival testi tüübil võib selleks olla näiteks funktsiooni nimi, ning mõned mittekohustuslikud - programmile või funktsioonile sisendite andmisel on näiteks sisendfaili määramine mittekohustuslik, kuna antud funktsioon või programm ei pruugi failist lugemist kasutada.

Tabel 1. TSL poolt toetatud testide tüübid

	Testi kirjeldus	Funktsiooni skoobiga testi tüüp	Programmi skoobiga testi tüüp
1.	Kontrollib, kas programm/funktsioon impordib mooduli(d).	*function imports module test*	*program imports module test*
2.	Kontrollib, kas programm/funktsioon kutsub välja funktsiooni(d).	*function calls function test*	*program calls function test*
3.	Kontrollib, kas programmis/funktsioonis sisaldub tsükkel.	*function contains loop test*	*program contains loop test*
4.	Kontrollib, kas programmis/funktsioonis sisaldub try/except plokk.	*function contains try except test*	*program contains try except test*
5.	Kontrollib, kas programm/funktsioon kutsub välja “print” käsu.	*function calls print test*	*program calls print test*
6.	Kontrollib, kas programmis/funktsioonis on defineeritud antud funktsioon(id).	*function defines function test*	*program defines function test*
7.	Kontrollib, kas programmis/funktsioonis sisaldub märksõna / sisalduvad märksõnad.	*function contains keyword test*	*program contains keyword test*
8.	Kontrollib, kas programm/funktsioon annab ette antud sisenditega käivitades soovitud väljundi(d).	*function execution test*	*program execution test*
9.	Kontrollib, kas funktsioon kasutab ainult lokaalseid muutujaid.	*function is pure test*	
10.	Kontrollib, kas funktsioon on rekursiivne.	*function is recursive test*	
11.	Kontrollib, kas funktsioonis sisaldub “return” käsk.	*function contains return test*	

Tabelist 1 enim kasutatud testide tüübid on *function execution test* ja *program execution test,* mis aitavad kontrollida lahenduseks esitatud programmi või funktsiooni väljundit ette antud sisendite korral. Ühtlasi on need ainukesed dünaamilised testi tüübid - ülejäänud 17 testi tegelevad lahenduse staatilise analüüsiga.

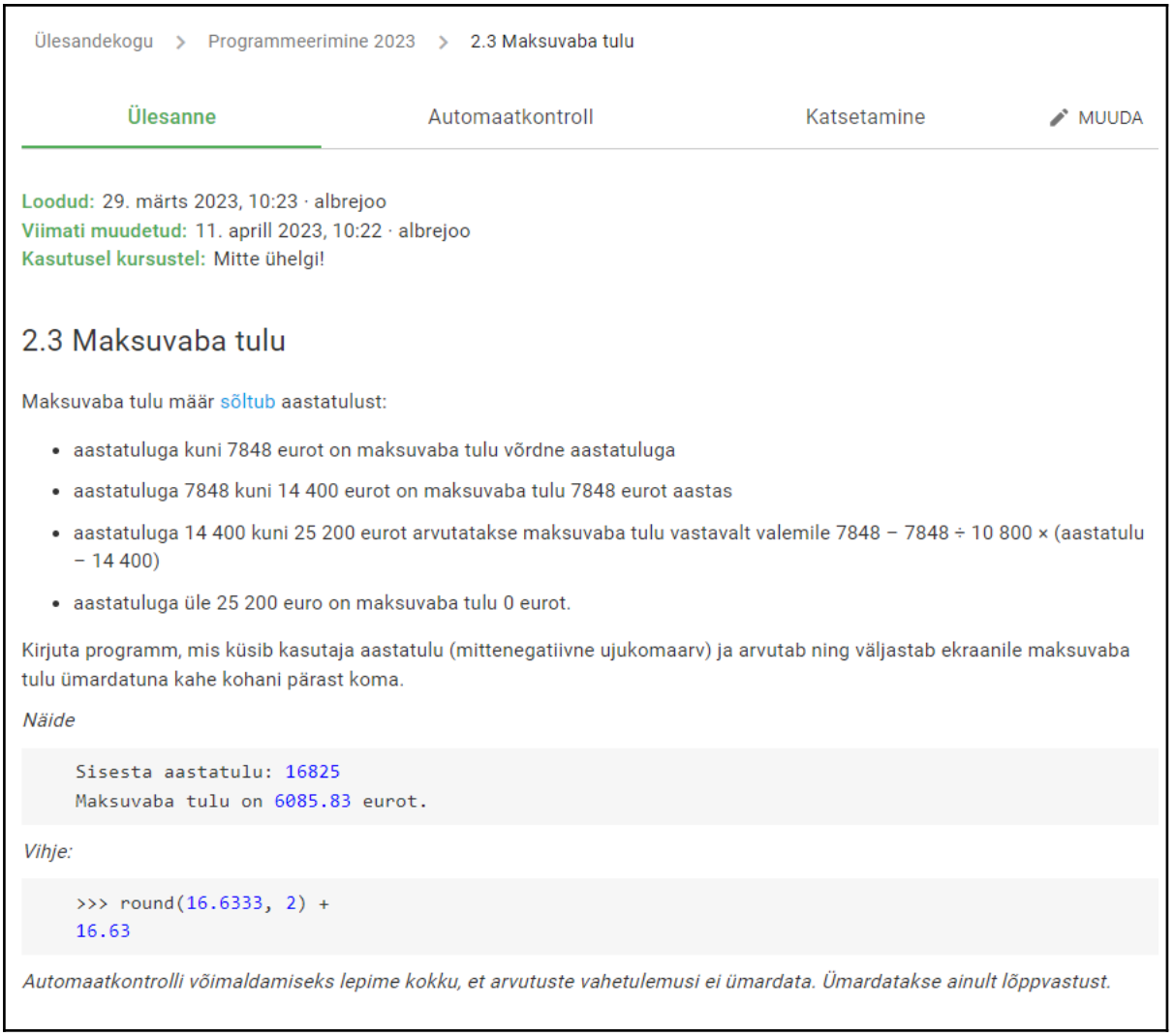
3. Metoodika

Bakalaureusetöö raames loodi Lahendus keskkonda uus kursus. Kursus koosneb üksnes programmeerimisülesannetest ning nende jaoks loodud automaatkontrollidest. Selles peatükis kirjeldatakse, milline nägi välja kursuse, ülesannete ja automaatkontrollide koostamise protsess tuginedes eelpool kirjeldatud teadmistele.

3.1 Ülesannete üleviimine Lahendus keskkonda

Kursusel kasutatud ülesanded on võetud arvutiteaduse instituudi Courses keskkonnast sügiseti toimuva kursuse "Programmeerimine" materjalidest [22]. Ülesannete ülekandmisel Lahendus keskkonda jäi nende sisuline pool üldiselt puutumata. Sisu muudeti juhul, kui see vajas kaasajastamist või oli ebakorrektne. Näiteks on viimase aasta jooksul maksuvaba tulu hõlmavad seadused muutunud ning seetõttu tehti muudatused maksuvaba tulu käsitleva ülesande püstituses. Ülesannete kirjeldused ja näited on vormindatud kasutades AsciiDoc[8] märgistuskeelt, et tekstid oleksid struktureeritud ning lihtsamini loetavad. Joonisel 3 on näha ülesannete teksti vormistus ja vorming Lahendus keskkonnas. Ülesande püstitustes on funktsioonide nimed märgitud kaldkirjas ja loetelud toodud välja punktidena, et lugejal oleks teksti lihtsam jälgida. Lisaks on kõigi ülesannete juures välja toodud ka näide programmi tööst, et lugejal oleks parem arusaam, milline valmis programm välja nägema peaks. Automaatkontrollide toimimiseks on tihti ülesande juures ka välja toodud täpsustused, mida lahendamisel peaks jälgima, et kontroll töötaks ootuspäraselt.

[8] https://asciidoc.org/



Joonis 3. Ülesande kirjelduse vaade Lahendus keskkonnas

3.2 Automaatkontrollide koostamine

Kursuse "Programmeerimine 2023" ülesannetele loodi automaatkontrollid, mis kontrollivad lahenduste korrektsust ning vigade puhul annavad tagasisidet, millise testi jooksutamisel ebakõla avastati. Automaatkontrollide redigeerimine ja loomine leidis aset otse ülesande redigeerimisaknas, kuna seal olev kompilaator teavitab koheselt, kui testide koostamise ajal esineb süntaksivigu. Redigeerimisaken on leitav ülesande juures asuvast menüüst, kuid sellele ligi pääsemiseks on tarvis vastavaid Lahendus keskkonna õiguseid - õpilastel ligipääs automaatkontrollide failidele ning nende redigeerimisele puudub.

Automaatkontrollide koostamisel oli esimene samm kindlaks määrata, mida täpsemalt on tarvis antud ülesande puhul kontrollida. See selgus uurides ülesande sisu ja milliste struktuuride kasutamist seal oodatakse. Selle alusel valitakse välja testide tüübid, mille abil

vajalikke aspekte kontrollida saab. Tulenevalt ülesande sisust valitakse vajalikud staatilised testid, mis kontrollivad, et kasutusel oleks kindlad teegid või välja kutsutud kindlad meetodid. Suurema osa automaatkontrollidest moodustasid dünaamilised testid, mis andsid programmile ette sisendi ning ootasid, et lahendus tagastaks õige väljundi. Et automaatkontroll saaks anda võimalikult täpset tagasisidet oli oluline testidele lisada asjakohased sõnumid, mis väljastatakse automaatkontrolli käitamise ajal. Testide koostamisel tugineti mitteavalikule juhendile pealkirjaga "TSL loomise juhend", mida haldavad TSL välja töötanud isikud. Mainitud juhend on üleval Google Docs keskkonnas ning on sarnaselt TSL-ga pidevalt täienemisel.

3.3 Automaatkontrollide testimine

Automaatkontrolle koostades peab omakorda veenduma, et nende enda funktsionaalsus oleks korrektne ja ootuspärane. Selles veendumiseks lahendati automaatkontrolli valmides vastav ülesanne ning esitati kood lahendusena automaatkontrolli. Esitatud lahendust kontrollitakse jooksutatades automaatkontrolli teste ning selle tulemusena väljastatakse tagasiside, millised testid läbiti edukalt ning millised mitte. Ebaedukate testide puhul tuli esmalt veenduda, et esitatud lahendus oleks korrektne - korrektse koodi korral viitab testi ebaõnnestumine aga hoopis veale testis endas. Säärane automaatkontrollide testimine aitas välja tuua murekohti ning kõrvaldada hulga automaatkontrollide koostamisel tekkinud vigu.

3.4 Esinenud probleemid

Automaatkontrollide loomiseks kasutatud märgistuskeel TSL pole käesoleva töö koostamise hetkel lõplikult valmis ning on pidevalt täiendamisel. Töö raames koostatud automaatkontrolle võib pidada esimeseks laiaulatuslikumaks TSL funktsionaalsuse katsetamiseks ning selle tulemusena esines testide koostamisel tehnilisi tagasilööke. Enamik tagasilöökidest olid väikesed ja ei takistanud testide kirjutamist - näiteks kui kasutajaliides kuvas automaatkontrolli tulemusi valel kujul. Kuid avastati ka loogikamuresid - näiteks selgus, et funktsioon, mis kontrollib, et väljundis puuduksid kindlad elemendid, ei teinud seda. Probleemide esinemise korral edastati info vastutavatele arendajatele, kes aegsasti neile lahendused leidsid ja parandused sisse viisid. Olulisemad leitud vead on loetletud tulemuste peatükis.

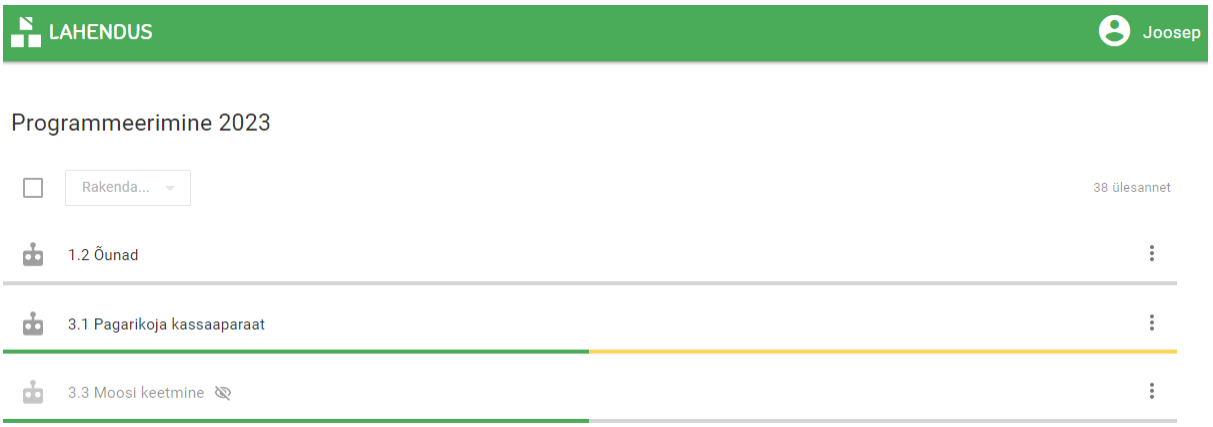
Kursuse 14. ja 15. õppenädal keskenduvad objektorienteeritud programmeerimise tutvustamisele. Käesoleva töö koostamise hetkel ei toeta TSL objektorienteeritud programmeerimise põhimõtetel koostatud ülesannete kontrollimist. On teada, et mainitud funktsionaalsus on arendamisel, kuid pole täpselt selge, millal see valmib. Selle tõttu jäeti 14. ja 15. õppenädala ülesanded bakalaureusetöö skoobist välja.

4. Tulemused

Selles peatükis kirjeldatakse lähemalt bakalaureusetöö tulemusena valminud kursust ja automaatkontrolle ning tuuakse võrdlus varasemate automaatkontrollidega. Lisaks loetletakse olulisemad töö käigus avastatud vead ning pakutakse välja tuleviku edasiarenduse võimalusi.

4.1 Valminud kursus

Bakalaureusetöö raames loodi Lahendus keskkonda uus kursus "Programmeerimine 2023". Kursus koosneb üksnes programmeerimisülesannetest ning nende jaoks loodud automaatkontrollidest. Õppematerjal on jaotatud nädalate kaupa ning sisaldab 38 ülesannet, millest 2 on praktikumide ülesanded, 4 on näidiskontrolltööde ülesanded ning 32 on kodutööde ülesanded. Joonis 4 näitab, mida kuvatakse "Programmeerimine 2023" kursusel õpetaja rollis olevale kasutajale.



Joonis 4. Kuvatõmmis "Programmeerimine 2023" kursuse ülesannete loendist

Õpetajatel on kursuse ülesannete loetelus lisandina riba, mis annab hinnangulise ülevaate automaatkontrollide läbimise kohta. Joonisel 4 ülesande 1.2 all nähtav hall riba viitab, et ülesande automaatkontrolli pole esitatud ühtegi lahendust. Roheline ja kollane riba värvus kirjeldavad vastavalt edukalt ning ebaedukalt läbitud automaatkontrollide osakaalu - 3.1 ülesande juures on hinnanguliselt pooled lahendused automaatkontrolli läbinud edukalt ning ülejäänud esitustes esines vigu. Samuti on õpetajatel võimalik kursuselt ülesandeid peita - joonisel 4 on ülesanne 3.3 õpilaste jaoks peidetud ning sellele viitab teksti hall värvus ja läbikriipsutatud silma ikoon.

4.2 Valminud automaatkontrollid

Selles peatükis võrreldakse varasemalt kasutusel olnud automaatkontrollide ning töö raames koostatud automaatkontrollide ülesehituste erinevust. Erinevust näidatakse võrreldes ühe kindla ülesande Python Grader automaatkontolli TSL abil defineeritud automaatkontrolliga.

Võrdluses kasutatud ülesanne on võetud 3. nädala materjalidest, kus õpitakse Pythoni funktsioone. Ülesande eesmärk on koostada funktsioon, millele antakse argumentideks suurte karpide arv, väikeste karpide arv ning moosi kogus ja funktsioon peab selgitama, kas moosi kogus mahub täpselt karpidesse ning tagastama, mitu karpi selleks tarvis on. Suur karp mahutab 5kg ja väike 1kg moosi.

Peale vaadates on TSL abil defineeritud ning Grader mooduli abil koostatud automaatkontrollide märgatavaks erinevuseks automaatkontrolli faili ridade arv - identsete kasutusjuhtumite testimise puhul koosneb TSL fail 183st ning Graderi oma 78st koodireast. See on tingitud erinevusest, kus TSL puhul on iga test eraldiseisev ning vajab konkreetset ülesehitust, võtmesõnu ja nende väärtustamist. Graderi puhul kutsutakse aga välja failis koostatud kontrollfunktsioon andes ette kontrollitava funktsiooni sisendid. Selle erinevuse illustreerimiseks on joonisel 5 ja joonisel 6 välja toodud sama sisend-väljund kombinatsiooni kontroll mõlema automaatkontrollimise võimaluse näitel.

```
- type: function_execution_test
    name: 2 suurt karpi, 6 väikest karpi, 14kg moosi
    id: 3
    points: 15
    functionName: moos
    arguments:
      - 2
      - 6
      - 14
    returnValue: 6
```

Joonis 5. TSL abil koostatud moosi ülesande test sisenditega 2, 6 ja 4

```
def test1(): check(2, 6, 14)
```

Joonis 6. Python Graderi abil koostatud test sisenditega 2, 6 ja 4

Joonisel 5 ja joonisel 6 olevad testid kontrollivad täpselt sama aspekti ning selle võrdluse puhul paistab Graderi abil koostatud test oluliselt lihtsama ülesehitusega. Sellisel kujul

toodud näide on aga eksitav. Nagu eelnevalt mainitud, kutsub Grader välja failis koostatud kontrollfunktsiooni. Joonisel 7 on välja toodud Graderi moosi ülesande kontrollfunktsioon nimega *check*.

```
10  def check(a, b, c):
11
12      programminimi = "kodu3.py"
13      funktsiooninimi = "moos"
14
15      progolemasolu(programminimi)
16
17      with Plan(f"Katsetan funktsiooni '{funktsiooninimi}' järgmiste sisenditega:\n" \
18                + f"suurte karpide arv = {a},\n" \
19                + f"väikeste karpide arv = {b},\n" \
20                + f"moosi kogus = {c}."):
21
22          # Programmi käivitamine
23
24          näitesisend = [a, b, c] + [''] * 20
25          sisendid = deepcopy(näitesisend)
26          globs, cap = run_script(programminimi, sisendid)
27
28          ### --- Funktsiooni testimine ---
29
30          # Funktsiooni signatuuri kontroll
31
32          oodatav_parameetrite_arv = len(signature(õige_vastus).parameters)
33          veatekst = "Funktsiooni parameetrid peavad olema suurte karpide arv, "\
34                     + "väikeste karpide arv ja moosi kogus."
35          funksignatuur(globs, funktsiooninimi, oodatav_parameetrite_arv, veatekst)
36
37          # Funktsiooni tagastatud väärtus
38
39          funktsioon = globs[funktsiooninimi]
40          argumendid = [a, b, c]
41          sisendid = deepcopy(näitesisend)
42          oodatav_sisendite_arv = 0
43
44          tegelik_tulemus, cap = täidafunk(funktsioon, argumendid, sisendid, oodatav_sisendite_arv)
45
46          # Funktsioon tagastas väärtuse
47
48          if tegelik_tulemus == None:
49              tühiprintteade(programminimi, funktsiooninimi)
50
51          # Funktsiooni tagastatud väärtus on õiget tüüpi
52
53          funktüüp(funktsiooninimi, tegelik_tulemus, int)
54
55          # Funktsiooni tagastatud väärtus on õige
56
57          oodatav_tulemus = õige_vastus(a, b, c)
58
59          if tegelik_tulemus != oodatav_tulemus:
60              teade = f"Funktsiooni '{funktsiooninimi}' tagastatud tulemus ei ole õige.\n"
61              teade += f"Ootasin, et funktsioon tagastab {oodatav_tulemus}, " + \
62                       f"aga funktsioon tagastas {tegelik_tulemus}.\n"
63              lõpeta(teade)
```

Joonis 7. Kuvatõmmis Python Graderis koostatud moosi ülesande kontrollfunktsioon

Joonisel 7 kuvatud kontrollfunktsioon näitab, kust tuleb mängu varasemalt töös mainitud Grader mooduli automaatkontrollide haldamise keerukus - automaatkontrolli koostamisel tuleb defineerida ka kontrollfunktsioon.

Kirjeldatud näide illustreerib TSL abil defineeritud automaatkontrollide loetavuse ning loomise olemust. TSL automaatkontrolli fail võib olla küll koodiridade poolest mahukam,

kuid iga testi puhul on lihtsasti mõistetaval kujul kirjas, mida see test täpselt kontrollib. Testi lihtsasti mõistetav ülesehitus on peamiseks faktoriks, mis kergendab TSL abil koostatud testide haldamist.

4.3 Leitud vead

Valminud bakalaureusetöö üks oluline eesmärk oli testida TSL abil automaatkontrollide koostamise võimalusi ning valmisolekut nende kasutusele võtmiseks programmeerimise kursustel. Protsessi käigus avastatud vead ja tehnilised probleemid on väga väärtuslik tagasiside selle hindamiseks. Vigade esinemise kogus ning nende mõju on indikaatorid tarkvara valmisolekust - mida vähem esineb laiema kasutuse juures tõsiseid vigu, seda lõpetatumaks võib tarkvara arendusprotsessi pidada.

Ülevaade automaatkontrollide koostamisel avastatud olulisematest tehnilistest probleemidest:

a) Dünaamiliste testide puhul on *checkType: NONE_OF_THESE* väärtuse abil võimalus defineerida test, mis kontrolliks, et programmi väljundist puuduksid kindlad elemendid. Selgus, et see funktsionaalsus on vigane ning oodatud kontrolli ei toimunud.

b) *Function_execution_test* tüüpi testi abil saab kontrollib lahenduse ühte kindlat funktsiooni. Kui funktsioon ootab sisendiks argumente, siis tuleb need testi defineerides väärtustada. Selgus, et kui argumentideks antavad väärtused erinevad andmetüübi poolest, näiteks sõne ja täisarv, siis visatakse testi jooksutamisel erind ning kontrolli ei toimu.

c) Juhul kui lahendusena esitatud programm ootab kasutajasisendit ja automaatkontrollis on kasutatud *function_execution_test* tüüpi testi, siis funktsiooni testimisel väljastati veateade: “programm küsib rohkem sisendeid, kui on antud”. Test, mis kontrollib kindla funktsiooni tööd, on mõjutatud funktsioonivälistest aspektidest - see pole oodatud käitumine.

d) Dünaamiliste testide puhul on võimalus lisada kolm täpsustavat sõnumit - testi olemust kirjeldav sõnum, eduka testi sõnum ning ebaeduka testi sõnum. Nimetatud sõnumid väljastatakse lahendust kontrollides kasutajaliideses. Selgus, et testi olemust kirjeldava sõnumi puudumisel ei kuvanud kasutajaliides automaatkontrolli tulemusi oodatud kujul - tulemus väljastati vormindamata JSON kujul.

e) *Function_execution_test* tüüpi testiga funktsiooni tagastusväärtuse kontrollimisel puudub võimalus lisada tagasisidet andvaid sõnumeid. Sõnumite olemasolu on oluline, et edastada lõppkasutajale täpsustavat infot testi sisu ja tulemi kohta.

Loetletud vigadest on töö esitamise hetkeks lahenduseta c) ja e) punktis mainitud probleemid. TSL arendajad on mõlemast teadlikud, ning tegelevad lahenduste leidmisega. Teised loetelus mainitud probleemid on lahendatud ning toimivad ootuspäraselt.

4.4 Edasiarenduse võimalused

TSL on veel arendamisel ning töö autoril puudub täielik ülevaade, milliste funktsionaalsuste ja uute võimaluste lisamine on juba kavas. Järgnevalt on välja toodud mõned ideed, mida võiks TSL arenduse juures kaaluda:

a) Objektorienteeritud programmide kontrollimise võimekus. TSL arendajad on selle puudusest teadlikud ning tegelevad lahenduse väljatöötamisega.
b) Dünaamiliste testide loomisel peab manuaalselt defineerima oodatava väljundi või tagastusväärtuse. Testide loomise vaatepunktist oleks oluliselt mugavam, kui peaks defineerima vaid sisendid ning oodatava väärtuse arvutatab TSL automaatselt. See eemaldaks testide loomise juurest ühe tüütu osa - oodatud vastuste manuaalse arvutamise ning seeläbi muudaks automaatkontrolli loomise protsessi kasutajasõbralikumaks.
c) Test, mis kontrollib programmi väljundi korrektsust, otsib kogu väljundi seest konkreetse oodatava väärtuse olemasolu. Näiteks, kui test ootab, et programm väljastab väärtuse “5”, siis väljundit “0123456789” peetakse korrektseks väljundiks, kuna seal sisaldub oodatud väärtus “5”. Seda teadmist ära kasutades on võimalik automaatkontrolli petta. Käesolevas töös lisati selle vältimiseks lihtsamate väljunditega ülesannete automaatkontrollidesse testid, mis kontrollivad, et väljund ei sisaldaks üleliigseid väärtuseid. Eelmainitud näite puhul tähendab see, et kontrollitakse, et väljund ei sisaldaks väärtuseid 0 - 4 ja 6 - 9. Üks versioon selle leevendamiseks oleks võimalus, kus ühe testi juures saab defineerida nii väärtused, mida oodatakse kui ka väärtused, mida vastus sisalda ei tohiks. Töökindlam lahendus oleks asendada kogu väljundist vastuse otsimine mõne muu väljundi tuvastamise meetodiga, kuid konkreetsed ideed ja ettepanekud selle saavutamiseks puuduvad.

Oluline on märkida, et TSL arendamise keskmes on eesmärk teha automaatkontrollide loomine võimalikult lihtsaks ning madala ajakuluga protsessiks. Selle saavutamiseks on hetkel arendamisel kasutajaliides, mille abil on võimalik automaatkontrolle koostada üksnes rippmenüüst väärtuseid valides ning lünkasid täites. Käesoleva bakalaureusetöö raames koostatud automaatkontrolle võib pidada vahesammuks selle eesmärgi suunas liikumisel.

Kokkuvõte

Bakalaureusetöö eesmärk oli uuendada Tartu Ülikooli kursuse “Programmeerimine” ülesannete automaatkontrolle ning viia need üle Lahendus keskkonda, et parandada automaatkontrollide hallatavust ning seeläbi vähendada õppejõudude koormust. Töö raames valmis Lahendus keskkonda kursus “Programmeerimine 2023”, mis koosneb kursuse ülesannetest ja valmistatud automaatkontrollidest.

Automaatkontrollide loomine oli ühtlasi selle jaoks kasutatud TSL võimaluse esimene laialdasem kasutus. Töö käigus leiti tehnilisi vigu, mis on nüüdseks parandatud, kuid ka vigu, mis endiselt lahendamist ootavad - avastatud vead on oluline tagasiside TSL funktsionaalsuse kontrollimiseks ning täiendamiseks. Automaatkontrollide koostamise protsessi edaspidiseks hõlbsustamiseks pakuti välja mõned võimalikud TSL edasiarenduse ideed.

Viidatud kirjandus

[1] Tartu Ülikooli õppeinfosüsteem ÕIS. Programmeerimine (6 EAP). https://ois2.ut.ee/#/courses/LTAT.03.001/version/5ce5cc23-3331-cfb9-2784-3f820084cb31/details (Kasutatud 02.04.2023)

[2] Tartu Ülikooli õppeinfosüsteem ÕIS. Programmeerimine (6 EAP). https://ois2.ut.ee/#/courses/LTAT.03.001/version/37ca53ac-440c-032f-1c5e-125c983bb04b/details (Kasutatud 08.05.2023)

[3] Tartu Ülikooli õppeinfosüsteem ÕIS. Programmeerimine (6 EAP). https://ois2.ut.ee/#/courses/LTAT.03.001/version/8a83ddcb-cbd3-1ae5-c3f4-b4ec8196a4ab/details (Kasutatud 08.05.2023)

[4] Carter, J., Ala-Mutka, K., Fuller, U., Dick, M., English, J., Fone, W., Sheard, J. (2003). How shall we assess this?. SIGCSE Bulletin. 35. 107-123, doi: 10.1145/960492.960539.

[5] Muuli, E., Palm, R., Lepp, M. (2023). Simplifying the creation and maintenance of automated assessments of programming tasks via Test Specific Language.. https://doi.org/10.1145/3578837.3578840

[6] Toome, E. (2021). A program that helps Estonian children prepare for the future. Education Estonia. https://www.educationestonia.org/progetiger-programming-school-kindergarten/ (Kasutatud 05.05.2023)

[7] Furnace, Public Brodcasting Service, Oregon Public Brodcasting, John Gau Productions (Tootjad), Sen P. (Rešissöör). (2012). Steve Jobs: The Lost Interview

[8] Wei, L. (2019) Cognitive Benefits of Language Learning: Broadening our perspectives. https://www.thebritishacademy.ac.uk/documents/287/Cognitive-Benefits-Language-Learning-Final-Report.pdf (Kasutatud 28.04.2023)

[9] Eesti Statistikaamet (2022). Palgarakendus. https://palgad.stat.ee/# (Kasutatud 03.05.2023)

[10] Eesti Statistikaameti koduleheküIg (2022). https://www.stat.ee/et/avasta-statistikat/valdkonnad/tooelu/palk-ja-toojoukulu/keskmine-brutokuupalk (Kasutatud 03.05.2023)

[11] B.S. Bloom, M.D. Engelhart, E.J. Furst, W.H. Hill, and D.R. Krathwohl. 1956. Taxonomy of Educational Objectives: Handbook I: Cognitive Domain, Longmans Group.

[12] Masapanta-Carrión, S., Velázquez-Iturbide, J. (2018). A Systematic Review of the Use of Bloom's Taxonomy in Computer Science Education. https://doi.org/10.1145/3159450.3159491

[13] Conklin, J. (2005). Review of A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom’s Taxonomy of Educational Objectives. Educational Horizons, 83(3), 154–159. http://www.jstor.org/stable/42926529 (Kasutatud 25.04.2023)

[14] Kahe õppedisaini mudeli võrdlus. https://retiht.files.wordpress.com/2021/01/bloom-1.png?w=333 (Kasutatud 01.05.2023)

[15] Absalomov, H., Esanov, S., & Hakimov, H. (2022). The importance of error correction in foreign language learning. Общество и инновации. https://hrcak.srce.hr/file/349678 (Kasutatud 25.04.2023)

[16] Combéfis, S. (2022) Automated Code Assessment for Education: Review, Classification and Perspectives on Techniques and Tools. Software 2022, 1, 3-30. https://doi.org/10.3390/software1010002

[17] Hao, Q., Smith IV, D., Ding, L., Ko, A., Ottaway, C., Wilson, J., Arakawa, K., Turcan, A., Poehlman, T., Greer, T. (2022) Towards understanding the effective design of automated formative feedback for programming assignments. Computer Science Education 32:1, 105-127.

[18] Setiawan, I., Maryono, D., & Basori, B. (2019). The Analysis of Software Source Code Readability: Case Study at Education of Informatics and Computer Engineering Study Program of Sebelas Maret University. Journal of Informatics and Vocational Education.

[19] Puulmann, K. (2014) Python module for automatic testing of programming assignments. TÜ arvutiteaduse instituudi bakalaureusetöö. http://macobo.github.io/python-grader/thesis.pdf (Kasutatud 06.03.2023)

[20] Muuli, E., Tõnisson, E., Lepp, M. et al. (2020). Using image recognition to automatically assess programming tasks with graphical output. Educ Inf Technol 25, 5185–5203. https://doi.org/10.1007/s10639-020-10218-z

[21] Andmekaitse ja infoturbe leksikon. https://akit.cyber.ee/ (Kasutatud 01.05.2023)

[22] Tartu Ülikool. Arvutiteaduse instituut. Arvutiteaduse instituudi kursused 2022/23 kevad. https://courses.cs.ut.ee/2021/prog-alused/spring/Main/HomePage (Kasutatud 20.01.2023)

Lisad

I. Loodud materjalid

Lõputöö raames loodud kursus ja selle ülesanded ning automaatkontrollid on kättesaadavad aadressil: https://dev.lahendus.ut.ee/library/dir/272/Programmeerimine-2023**.** Materjalidele juurdepääsemiseks tuleb ligipääsu küsida Tartu Ülikooli kursuse “Programmeerimine” (LTAT.03.001) vastutavalt õppejõult.

II. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Joosep Albre,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose **Tartu Ülikooli kursuse "Programmeerimine" ülesannete automaatkontrollide uuendamine ja üleviimine keskkonda lahendus.ut.ee**, mille juhendaja on Tauno Palts, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Joosep Albre
09.05.2023