TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Infotehnoloogia õppekava

Siim Plangi

Automaatne programmeerimisülesannete kontrollija Tartu Ülikooli kursuse "Algoritmid ja andmestruktuurid" jaoks

Bakalaureusetöö (6 EAP)

Juhendajad: Anne Villems
Henri Lakk

Tartu 2014

Automaatne programmeerimisülesannete kontrollija Tartu Ülikooli kursuse “Algoritmid ja andmestruktuurid” jaoks

Lühikokkuvõte:

Käesolev töö põhineb 4 tudengi loodud programmil “Automaatne hindaja Tartu Ülikoolile” (AHUT), mis on veebipõhine rakendus tudengite programmeerimisülesannete automaatseks hindamiseks. Vajadus sellise programmi järele tekib siis, kui nii tudengeid kui ka neile antavaid ülesandeid on palju, aga tagasisidet soovitakse anda võimalikult kiirelt. Töös on kirjeldatud erinevaid varem loodud automaatseteid hindajaid, nende tööpõhimõtet ja struktuuri. Lisaks on põhjalikumalt tutvustatud AHUT programmi ennast, selle komponente ja kasutust ning viidud läbi testimine töökindluse hindamiseks ja võimalike vigade tuvastamiseks.

Võtmesõnad:

Programmeerimisülesanded, automaatkontroll, veebirakendus

Automatic programming exercise evaluation system for University of Tartu course “Algorithms and Data Structures”

Abstract:

This thesis is based on web application “Automaatne hindaja Tartu Ülikoolile” (AHUT) created by four students from University of Tartu. This application automatically checks programming solutions submitted by computer science students for specific exercises. The need for this kind of program was arisen due to the increasing number of students which means larger workload for teachers who check these programming solutions. This paper describes a variety of automatic programming evaluation systems developed earlier, introduces the AHUT program itself and its components and deducts testing of reliability of the AHUT system.

Keywords:

Programming exercises, automatic evaluation, web application

Sisukord

1 Sissejuhatus

Võrreldes aastaga 2003 on informaatika ja infotehnoloogia eriala tudengite arv Tartu Ülikoolis rohkem kui kahekordistunud [1]. See tähendab, et programmeerimise kursustel, nagu näiteks ainetel "Programmeerimine" ning "Algoritmid ja andmestruktuurid", on igal aastal järjest rohkem tudengeid ning nende ainete õppejõud on ülekoormatud erinevate programmeerimisülesannete kontrollimisega. See tõigi vajaduse luua süsteem, mis suudaks automaatselt kontrollida tudengite esitatud töid.

Käesolev bakalaureusetöö põhineb nelja tudengi (Tair Vaher, Johannes Ait, Taavo Salumaa ja Siim Plangi) loodud rakendusel "Automaatne hindaja Tartu Ülikoolile" (AHUT), mis valmis projektina Tartu Ülikooliaines "Tarkvaraprojekt". Kuna käesoleva töö autor oli AHUT väljatöötamisel ja arendamisel projektijuht ning soovis seda edasi arendada, oli heaks väljundiks seni loodud programm vormistada bakalaureusetööks.

AHUT on veebipõhine rakendus, kus õppejõud saavad defineerida enda kursusele ranges formaadis ülesandeid ning tudengid saavad oma programmeerimisülesandeid esitada, järgides õppejõu poolt ettemääratud reegleid. Rakendus käivitab esitatud programmeerimistööd ning kontrollib õppejõu poolt määratud sisenditega tudengi programmi väljundeid, kusjuures tudengid pole programmile etteantavatest sisenditest teadlikud.

Käesolev töö koosneb neljast peatükist. Esimene peatükk käsitleb automaatsete programmeerimisülesannete hindajate ajalugu ning annab lühiülevaate mõningatest automaatse hindaja (AH) programmidest. Teine peatükk käsitleb AHUT süsteemi arhitektuuri ning funktsionaalsust. Kolmas peatükk juhendab, kuidas AHUT süsteemi kasutada ning neljas peatükk kirjeldab AHUT süsteemi testimise tulemusi.

2 Automaatsete hindajate ülevaade

Süsteeme, mis kontrolliksid programmeerimisülesannete lahendusi, on loodud ja disainitud juba üle 50 aasta [2] ning praeguseks on arendatud palju eraldiseisvaid AH programme. See tekitab küsimuse, miks on neid nii palju ning miks neid pidevalt juurde luuakse? Laialt levinud valmisprogramme, mida saaks ülikool või muu üksus üle võtta, et oma töökoormust vähendada, on vähe [3]. Põhjus, miks levivad just programmid, mis pole kaugeltki veel küpsed, on see, et need programmid luuaksegi kas ainult teadusliku töö raames või ühe konkreetse kursuse jaoks. Antud hindajad disainitakse algusest peale lihtsateks ja oma tarbeks ning nendega ei jõutagi levitatava tooteni. On olemas AH programmide lähtekoodi, kuid sellega edasi minna ning luua midagi, mis vastab just spetsiifilise ülikooli vajadustega, on keeruline ning ajakulukas. Kuna nende programmide kvaliteet on kaheldav ja need on sageli vähe dokumenteeritud või lausa dokumenteerimata, siis on uue tarkvara loomine tihtipeale mõistlikum. Selliste programmidega, mis siiski jõuavad laiatarbekasutamiseni, on muud probleemid, näiteks tarkvara maksumus, vajaliku programmeerimiskeele mittetoetamine, mittepiisav kasutajaliides või tarkvaratoetuse lõpp. See loobki olukorra, kus iga ülikool loob oma programmi ainult enda tarbeks, kusjuures nende oma tarbeks loodud programmide põhiidee on sarnane.

Järgnevates punktides tutvustatakse Automaatsete hindajate ajalugu, varem loodud süsteeme ning kirjeldatakse täpsemalt üht AH süsteemi.

2.1 Automaatsete hindajate ajalugu

Automaatsete programmeerimisülesannete kontrollijate ajalugu ulatub sama kaugele kui programmeerimise õpetaminegi. Artikkel “Automatic Test-Based Assessment of Programming: A Review” [2] klassifitseerib automaatsed hindajad kolme erinevasse generatsiooni. Esimese generatsiooni hindajad võeti kasutusele paralleelselt arvutiteaduse õpetamisega. Need olid programmid, mille põhiülesandeks oli õpilaste lahenduste kontrollimine. Teise generatsiooni uuenduseks oli haldussüsteemide lisandumine. Kolmanda generatsiooni hindajaid kasutatakse praegusel ajal, neid iseloomustab veebitehnoloogia kasutuselevõtt. Järgnevates lõikudes tutvustatakse erinevaid AH rakenduste generatsioone.

Esimese generatsiooni AH programmid olid väga algelised, nende põhiülesandeks oli kontrollida õpilase ülesandele esitatud programm ning mõnel juhul koostasid need AH süsteemid lühikesi raporteid. Ühe esimestest I generatsiooni AH süsteemidest lõi J. Hollingsworth 1960. aastal [4]. Õpilased esitasid sellel ajal oma programmi koodi perfokaartidel. Õpilaste

lahenduste väljundeid võrreldi õpetajate programmi väljunditega. Lahendust hinnati kahe väljundiga, kas “Programm complete” või “Wrong answer”. Sellise automaatse hindajaga hoiti kokku õpetajate aega ja arvuti ressursse, mis võimaldas suuremal arvul õpilastel programmeerimist õppida. J. B. Hext ja J. W. Winnings’i programm võrdles salvestatud andmeid ja õpilase programmi poolt väljastatud tulemusi [5]. Pärast õpilase lahenduse käivitamist koostas testimise programm raporti tema lahenduse kohta. Nende programmi implementatsioon nõudis muudatusi kompilaatorites ning operatsioonisüsteemides.

Mõlema eelkirjeldatud programmi autorid toovad esile kaks võimalikku probleemi: süsteemi tahtlik rünnak ning õpilaste omavaheline koodide kopeerimine. Tähelepanu tuleb suunata hindaja turvalisuse tagamisele ning plagiarismi vältimisele. Esimese generatsiooni AH süsteeme oli väga raske luua, sest neid programmeerides pidi tundma assemblerkeelt ning osata laiendada operatsioonisüsteeme ja kompilaatoreid. Seetõttu said neid programme luua inimesed, kes olid programmeerimise eksperdid.

Teise generatsiooni AH programme iseloomustavad käsurea kasutajaliidesed ja skriptidega lahenduste kontrollimine. Skriptidega kontrollimisel pidi õpetaja iga ülesande jaoks kirjutama automaatse hindaja süsteemile eraldi kontrollimist kirjeldava skripti. Seejärel kompileerib automaatne hindaja, kasutades skripti kui juhist, õpilase lahenduse, käivitab selle ning proovib programmi väljundeid samm sammu haaval võrrelda õppejõu skriptis defineeritud testandmetega. Lisaks sellele oli II generatsiooni AH programmidel uute lahenduste seas esituste arvu piiramine, koodi ajamäära analüüs ning koodi keerukuse ja stiili analüüs. Teise generatsiooni automaatsed hindajad tõid kaasa lisaks programmeerimisülesannete kontrollimisele ka haldussüsteemid. Haldussüsteemid on AH programmidele lisavahendiks, millega lihtsustatakse automaatse hindamisega seotud toiminguid. Haldussüsteemidega sai õppejõud luua aruandeid, vaadata õpilaste tulemusi ja muud sarnast ning õpilane esitada oma lahendust ja näha iseseisvalt tagasisidet. Teise generatsiooni AH programmide loomine oli tunduvalt kergem, kui esimese generatsiooni AH programmide. Seda eelkõige seetõttu, et teise generatsiooni AH süsteemid loodi kõrgema taseme programmeerimiskeelte abiga, mis muutis nende loomise lihtsamaks ning võimaldas luua keerukamaid ja täiuslikumaid automaatseid hindajaid.

Kolmanda generatsiooni AH iseloomustab veebitehnoloogia kasutamine, keerukamad haldussüsteemid ja tõhusamad testimissüsteemid. Kõiki kolmanda generatsiooni AH süsteeme kasutatakse praegugi ja neid kireldatakse järgmises punktis.

2.2 Ülevaade kolmanda generatsiooni automaatsetest hindajatest

Käesolev punkt ja selle alapunktid põhinevad artiklil "Review of Recent Systems for Automatic Assessment of Programming Assignments" [3], kus käsitletakse erinevaid kolmanda generatsiooni AH süsteeme, mis on loodud peale 2005. aastat (nimekiri lisas I). Eelnimetatud uurimuse fookus oli AH funktsioonidel, mitte süsteemidel endil. Uuritavaid süsteeme oli selles uuringus palju ning neid eraldi ei käsitletud.

Vaadeldavaid AH süsteeme võrreldakse järgnevate tunnuste alustel:

1. programmeerimiskeelte toetus;
2. seos haldussüsteemidega;
3. ülesannete defineerimine;
4. õpilaste lahenduste esitamine;
5. manuaalne lahenduste hindamine;
6. turvapoliitika;
7. kättesaadavus ja levitamine.

Järgnevates punktides antakse ülevaade meetoditest ja viisidest, kuidas erinevad aspektid uuritavatel AH programmidel on lahendatud.

Programmeerimiskeelte toetus

Java on väga paljude ülikoolide esimene õpetatav programmeerimiskeel, seega toetavad enamik uuritavatest AH süsteemidest just *Java* keelt. Lisaks *Java*'le on populaarsed järgmised programmeerimiskeeled: *C/C++*, *Python* ja *Pascal*. Lisaks kõrgema taseme programmeerimiskeeltele on automaatseid hindajaid loodud ka assembleri- ja skriptimiskeeltele. On ka süsteeme, mis on keelest sõltumatud. Sellised süsteemid kasutavad sisendväljund baasil kontrollimist, mis tähendab, et süsteem suudab kontrollida kõike, mida arvuti kompileerida ja käivitada suudab [3].

Sisendväljund (SV) baasil AH süsteemid annavad õppeasutusele palju suurema valikuvabaduse, kuna sel juhul saab AH süsteemi kasutada erinevates kursustes, mis kasutavad mitmesuguseid programmeerimiskeeli. Küll aga jääb sellistel AH programmidel vajaka kontrollimise täpsuses, sest tegelikult ei kontrollita õpilase lahenduse koodi, vaid selle tagastusväärtusi.

Automaatsete hindajate haldussüsteemid

Haldussüsteemid lihtsustavad AH programmide kasutamist, muudavad neid efektiivsemaks ja annavad parema ülevaate süsteemis toimuvast. Haldussüsteemid automatiseerivad ülesannete esitamise ning tagasiside andmise. Selle asemel, et õppejõud käsitsi lahendused AH süsteemi üles laevad, teevad seda hoopis õpilased ise ning süsteem annab automaatselt neile ka tagasiside. Lisaks saab AH haldussüsteemidele implementeerida lisasüsteeme ülesannete defineerimiseks, uute kursuste loomiseks, plagiarismi tuvastamiseks jpm. Seega on haldussüsteemid tänapäevaste AH-de lahutamatuks osaks.

Kuus artiklis vaadeldud AH programmi kasutavad juba eksisteerivat haldussüsteemi (nt *Moodle, Cascade*) ning AH-d on sellesse keskkonda integreeritud. Teised automaatsed hindajad kasutavad enda loodud keskkondi, et tagada õpilastele ja õpetajatele juurdepääs. Mõned AH rakendused ei oma üldse haldussüsteemi, vaid tegelevad ainult ülesannete kontrollimisega. Õppejõud peavad sel juhul ise kõik õpilaste lahendused AH süsteemi sisestama ning õpilastele hiljem tagasisidet käsitsi kirjutama.

Olemasolevate haldussüsteemide kasutamise juures on positiivne see, et ei pea implementeerima uut lahendust tulemuste ja tagasiside andmiseks. Lisaks sellele on tõenäoline, et ülikooli kogukond on varem haldussüsteemi (nt Moodle) kasutanud ning sellega tuttav. Negatiivse poole pealt võib välja tuua, et antud automaatsed kontrollijad peavad olema väga hästi turvatud, sest pistikühendusega programmid võivad anda ründaja(te)le ligipääsu tundlikele andmetele kasutatavas haldussüsteemis.

Ülesannete defineerimine ja kontrollimine

Ülesannete defineerimiseks ja kontrollimiseks on palju võimalusi. Kõige rohkem on vaadeldavatest AH süsteemidest kasutatud SV kontrolli. Sel juhul kirjeldatakse õpilasele, mida ta programmeerima peab ja defineeritakse ülesande sisendid ning oodatavad väljundid. Õpilase programm käivitatakse defineeritud sisenditega ning kontrollitakse programmi väljundeid. Vaadeldavatest AH süsteemidest oli teiste lahenduste hulgas skriptimine, millega saab õpilase esitatud lähtekoodi analüüsida erinevatel viisidel. Nende hulgas on SV kontroll, lähtekoodi ülesehituse ja struktuuri kontroll ning ka lähtekoodis sisalduvate elementide kontroll.

Vaadeldavate AH-de hulgas osutus kõige populaarsemaks implementatsiooniks SV baasil kontrollimine. SV meetodit on kõige lihtsam implementeerida ning ka õppejõududel on seda

kasutades kerge ülesandeid defineerida. Piisab ainult ülesande sisendväärtustest ja õigetest väljundväärtustest, et edukalt luua kontrollimise ülesanne.

Lahenduste esitamine ja taasesitamine

Õppimise seisukohalt on harjutamine tähtis ning seepärast on oluline, et õpilased saaksid ebaõnnestumise korral oma ülesande lahendust taasesitada. Samas tuleks vältida pidevaid katse-eksituse meetodil põhinevaid taasesitusi, kuna sel juhul õpilane ei õpi tehtud vigadest ning muudab oma programmi mõtlemata vigadele. Selle vältimiseks kasutati vaadeldavates AH süsteemides mitmeid lahendusi:

- Ajaline piiratus - õpilased saavad esitada lahendusi ülesannetele ainult määratud ajapiiri sees.
- Piirata esituste arvu - piiratakse õpilaste programmeerimislahenduste esitamiste arvu. See distsiplineerib õpilasi hoolikalt oma lahendused läbi mõtlema ning väldib katse-eksituse meetodil parandusi.
- Piirata tagasiside suurust - tagasiside suuruse piiramisel ei anta õpilasele mingit konkreetset vastust, kus tema programm on valesti koostatud. Siinkohal peab õpilane ise vea üles otsima ning katse-eksituse meetodil programmi muutmine pole tõhus viis oma programmist viga leida. See meetod võib aga kaasa tuua negatiivse suhtumise või arusaamatused AH süsteemi suhtes.
- Ajaline karistus - enne uut esitust peab iga õpilane ära ootama teatud ajavahemiku, et uuesti programmi esitada. Katse-eksituse meetod muutub sel juhul väga ajakulukaks meetodiks ülesanne õnnestunult lahendada.
- Muuta ülesande ülesehitust - muuta ülesannet teatud määral pärast igat ebaõnnestunud katset. Pole mingit mõtet teha programm katse-eksituse meetodil, kui pärast igat katset peab kasutaja uue programmi kirjutama.
- Muuta ülesannete esitamine õppijatele võistluseks - ülesande võistluseks muutmisel on täheldatud väga häid tulemusi õpilaste motiveerimisel ülesande edukaks sooritamiseks. See aga toob kaasa ka negatiivseid aspekte, nagu näiteks kuidas õpetada õpilasele tarkvara arendamist mõistliku ajakava alusel, kui ülesanded on fokusseeritud kiirusele ja mitte nii väga kvaliteedile.
- Hübriidlahendused - nii mõnigi vaadeldavatest AH süsteemidest kasutab kahte või enamat võimalust, et vältida katse-eksituse meetodi kasutamist.

Kõige enam kasutasid automaatsed hindajad ülesannete ajalist piiratust ja esitamise arvu piiramist, need on ehk ka kõige paremad ja kindlamad viisid toetamaks õpilasi enne esitamist oma koodi hoolikalt läbi mõtlema. Lisaks on neid kerge implementeerida igas süsteemis. Paraku toob esitamise arvu piiramine kaasa mure, mis saab nendest õpilastest, kes kõik vastamiskorrad ära kasutavad ning tulemust üldse kirja ei saa, kuigi nende programm mõningal määral töötas. Seetõttu on mõistlik implementeerida AH süsteemi rohkem kui üks eelloetletud taasesituse piiramise meetoditest ning lasta õppejõul teha valik, milliseid võimalusi tema oma ülesande kontekstis kasutab.

Manuaalne ülesannete hindamine

Kuigi AH süsteemid annavad õpilastele soorituse kohta tagasisidet, võib juhtuda, et õppejõududel on vaja ülesandeid manuaalselt hinnata. Uuritavates AH programmides leiti kaks erinevat võimalust manuaalse tagasiside andmise jaoks. Enamlevinud viis oli võimalus vaadata õpilase sooritust, mis võimaldab sama esitust hinnata ka manuaalselt. Selle lähenemisega ei saa muuta AH määratud tagasisidet, kuid vähemalt on õppejõul võimalus anda tagasisidet muid sidekanaleid kasutades. Teiseks variandiks oli vaadeldavates AH programmides see, et lisaks õpilase esituse vaatamisele saab ka tagasisidet ja/või hinnet muuta.

Turvapoliitika

Kuna automaatne programmeerimisülesannete kontrollimine nõuab õpilase esitatud programmide käivitamist, siis muudab see AH süsteemi rünnetele haavatavaks. Õpilane võib luua programmi, mis kahjustab antud AH keskkonda (nt kustutab süsteemi faile). Süsteemi turvalisuse tagamiseks oli vaadeldavates AH programmides kasutatud erinevaid lahendusi. Kõige populaarsemaks turvamise viisiks oli õpilase lahenduse käivitamine nn liivakastis (eraldi kaitstud keskkonnas) või virtuaalmasinas. Liivakastis käivitamisel pole õpilase programmil õigusi ega ressursse teha midagi, mis süsteemi halvaks. Virtuaalmasina puhul võib programm virtuaalmasina rikkuda, kuid selle asemel käivitatakse lihtsalt uus ning ründajast jääb sel juhul jälg maha. Liivakasti meetodi ja virtuaalmasinate jaoks on kasutatud näiteks *Linux*'i turvapoliitika moodulit [6] ja valmislahendust *Systrace* [7]. Äsja mainitud meetodid loovad võimaluse hinnata programmi efektiivsust, kui piiratakse protsessori ja vahemälu ressursse ning jälgitakse, et programm lõpetaks töö ajapiiri sees. Lisaks liivakasti ja virtuaalmasinate meetoditele saab kasutada ka staatilist koodi ülevaatust, et tuvastada kuritahtlikku koodi. Kindlasti ei tohiks AH programmi installeerida arvutile, kus on andmeid, mis ei tohiks lekkida ega hävida. AH programmeerimisel võiks kaaluda arhitektuurilist valikut,

kus tundegi lahendus käivitatakse sõltumatul arvutil, mis ongi spetsiifiliselt mõeldud ainult tundmatute programmide käivitamiseks.

Kättesaadavus ja levitamine

Üldiselt on automaatsete hindajate kättesaadavus raskendatud, paljud programmid on eraomandis või neid pole üldse levitatud. Lisaks sellele on enamik programme loodud prototüüpidena teadustöö raames ning neid ei arendata edasi ega levitata.

Eelnevates punktides käsitleti kolmanda generatsiooni AH programmides tehtud struktuurseid valikuid. Järgmises punktis võetakse vaatluse alla üks konkreetne süsteem ning kirjeldatakse lähemalt programmi loojate tehtud tarkvaralisi valikuid.

2.3 Helsingi Tehnoloogiaülikooli süsteem *Scheme-robo*

Helsingi Tehnoloogiaülikoolis loodi 1997. aastal programmeerimisülesannete automaatne kontrollija "*Scheme-robo*" [8], mis kontrollis üliõpilaste koduseid programmeerimisülesandeid *Scheme* programmeerimiskeeles. Vajadus automaatse kontrollija järele tekkis, kui ülikooli *Scheme*'i õpetavas aines tudengite arv kasvas ning koduste ülesannete käsitsi kontrollimine muutus õppejõududele liiga ajamahukaks. Ühe võimalusena prooviti ülesannete lahendamine muuta vabatahtlikuks, kuid see tõi kaasa üliõpilaste tulemuste langemise. Seega oligi vaja programmi, mis automatiseeriks ülesannete kontrollimise.

Ülesandeid kontrollitakse SV baasil. Õppejõud defineerib ülesande, kirjutades testi sisendid ning oodatavad väljundid. Kui tudeng esitab oma koduse ülesande, käivitab *Scheme-robo* etteantud sisenditega ülesande. Selle asemel, et kontrollida standard väljundvoogu, kontrollib programm hoopiski meetodite tagastusväärtusi. See elimineerib probleemi, et tudengite vastuste väljatrükk peab olema rangelt reguleeritud, et AH väljatrükki õigeks loeks. Lisaks SV kontrollile suudab programm kontrollida ka lahenduste keerukusastet, struktuuri ning otsida plagiarismi erinevate lahenduste vahel.

Ülesannete esitamise ning tagasiside saamiseks pole erilist halduskeskkonda loodud. Tudengid saadavad oma harjutused manusena *Scheme-robo* e-maili aadressile ning *Scheme-robo* saadab tagasiside e-mailiga tagasi umbes 10 minuti pärast. Selline ajaviivitus on jäetud meelega, et takistada tudengitel pidevaid esitusi tegemast ning panna neid rohkem uurima oma programmi, kui lahendus kontrolli ei läbinud. Lisaks ülesannete esitustele saavad üliõpilased küsida läbi e-maili (kasutades spetsiifilisi parameetreid) oma programmikoodi koopiaid ja senimaani teenitud punktisummat.

Et takistada tudengitel või kõrvalistel isikutel e-mailiga pahavara saatmast, avatakse ja käivitatakse kõik lahendused turvalises liivakastis. Lisaks sellele loetakse ka erinevate arvutuste hulka, et vältida programmi jäämist lõpmatusse tsüklisse.

Tudengite tagasiside süsteemi kohta osutus väga positiivseks. 2000. aasta oktoobris läbi viidud küsitlusel, kus uuriti tagasisidet 350 programmeerimise aines osalenu käest, arvas 80% üliõpilastest, et *Scheme-robo* on hea või väga hea idee. 10% tudengeid arvas, et programm hindas ülesandeid halvasti, 41% mõõdukalt hästi, 44% hästi ja 6% väga hästi. Uurijatele ootuspäraselt olid *Scheme-robo*'s kõige negatiivsemaks küljeks tagasiside e-mailid, koguni 55% tudengitest arvas, et tagasiside e-mailid olid kehvad.

Ülalpool kirjeldati konkreetset kolmanda generatsiooni AH süsteemi. Tegemist oli eduka tarkvaraprojektiga Helsingi ülikoolis, mis lihtsustas oluliselt nii tudengite kui ka õppejõudude tööd. Järgmises peatükis võetakse vaatluse alla AHUT süsteem ning selles tehtud arhitektuurilised ja tarkvaralised valikud.

3 Ülesande püstitus

AHUT projekt lihtsustab ja kiirendab Tartu Ülikooli kursuse "Algoritmid ja andmestruktuurid" (A&A) õppejõudude tööd. AHUT loomise eesmärk on automatiseerida A&A õppeaine kodutööde hindamist, mida võib hiljem laiendada ka teistele õppeainetele. Kuna A&A kursusest võtab osa suur hulk üliõpilasi ning kodutöid on palju, siis kõikide tööde üksipulgi kontrollimine on õppejõududele ajamahukas ettevõtmine. AHUT on süsteem, kus üliõpilased saavad esitada oma töid ning programm kontrollib ning annab tagasisidet selle töö kohta. Samuti kuvatakse õppejõule iga ülesande puhul nimekiri tudengitest, kes on vastava ülesande ära lahendanud. Programm on kasulik ka üliõpilastele, sest nad saavad tagasisidet oluliselt kiiremini kui nad esitaksid tööd õppejõule manuaalseks hindamiseks.

Järgnevates punktides loetletakse AHUT funktsionaalseid ja mittefunktsionaalseid nõudeid, kirjeldatakse süsteemi arhitektuuri põhikomponente ning võrreldakse AHUT-d eelmises peatükis vaadeldud süsteemidega.

3.1 Funktsionaalsed nõuded

AHUT projekti alguses määrasid programmi autorid koos A&A kursuse juhendaja Henri Lakiga minimaalsed funktsionaalsed nõuded, mis on määratletud allpool [9].

1. Programm peab automaatselt kontrollima üles laetud kodutöid/proovilahendusi, kasutades standardset sisend- ja väljundvoogu.
2. Programm peab automaatselt võrdlema tudengite lähtekoode ning otsima plagiarismikahtlusega lahendusi.
3. Programm peab tuvastama rakendust kasutatava isiku.
4. Programm peab eristama õppejõude, tudengeid ning administraatorit:
    a. Administraatori alamkasutaja on õppejõud;
    b. Õppejõu alamkasutaja on tudeng;
    c. Kõik alamkasutajate võimalused peavad olema ka ülemkasutajatel.
5. Kõigil kasutajatel peab olema võimalik:
    a. Muuta oma konto parooli;
    b. Taastada oma parool;
    c. Üles laadida kodutöö/proovilahendus.
6. Tudengitel peab olema võimalik:
    a. Vaadata tagasisidet oma esitatud tööle;

    b. Vaadata temale määratud kursuse ülesannete nimekirja.
7. Õppejõududel peab olema võimalik lisaks tudengite funktsioonidele:
    a. Vaadata temale määratud kursuse ülesannete nimekirja;
    b. Vaadata kodutööde tulemusi ülesannete kaupa;
    c. Alla laadida tudengi esitatud lähtekoodi;
    d. Vaadata tudengi esitatud lähtekoodi;
    e. Registreerida kasutajaid enda õpetatavale ainele CSV mooduli abil;
    f. Registreerida kasutajaid enda õpetatavale ainele manuaalselt;
    g. Lisada ülesandeid;
    h. Ülesandeid arhiveerida;
    i. Kursust arhiveerida;
    j. Ülesannet muuta;
    k. Otsida tudengeid nime järgi.
8. Administraatoril peab olema võimalik lisaks õppejõudude funktsioonidele:
    a. Avada uusi kursusi;
    b. Määrata kursustele õppejõude;
    c. Vaadata logifaile.

3.2 Mittefunktsionaalsed nõuded

Järgnevad AHUT mittefunktsionaalsed nõuded koostati projekti alguses koos Henri Lakiga [10].

1. Minutis on võimalik maksimaalselt teha 100 päringut.
2. Tudengi lahenduse kontrollimiseks peab saama määrata maksimaalse ajamäära. Kui ajamäär ületatakse, lõpetatakse tudengi lahenduse kontroll. Ajamäär peab olema muudetav konfiguratsioonifaili abil.
3. Tööde arv, mida suudetakse korraga kontrollida on 1. Kontrollimisel lähtutakse põhimõttest, kes kõige enne lahenduse esitab, selle tööd ka varem kontrollitakse.
4. Lahenduse veebikeskkond peab toetama järgmiste veebibrauserite viimaseid versioone: *Google Chrome*, *Mozilla Firefox*, *Internet Explorer* ja *Safari*.
5. Tudengite lahenduste kontrollija moodul peab realiseerima mingit võimalikku liivakasti tehnoloogiat.
6. Süsteem peab kontrollima teatud ajavahemiku tagant, kas on uusi ülesandeid, mida kontrollida. Ajamäär peab olema muudetav konfiguratsioonifaili abil.

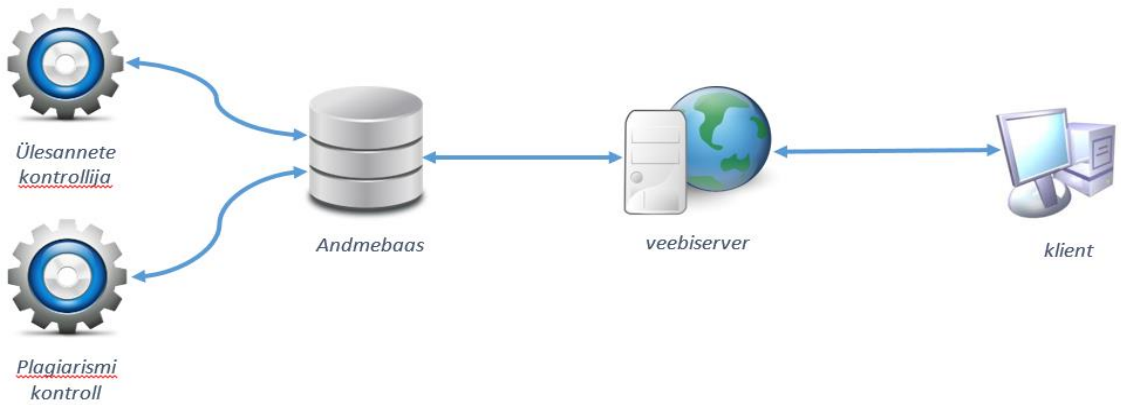
7. Lahendus kasutab kliendikihis *HTML*, *CSS* ja *Javascript* tehnoloogiaid.
8. Ülesande üleslaadimine ei tohiks võtta rohkem kui 30 sekundit.
9. Esmakasutajal ei kulu ülesande üleslaadimiseks üle 5 minuti.
10. Rakendus peab suutma programmi väljundi õigsust kontrollida.
11. Rakendus peab alles hoidma viimast esitatud kodutööd antud ülesandele.
12. Rakenduse kaudu peab saama uusi kasutajaid registreerida.
13. Rakendus peab toetama *timeout*'i (ülesande kontrollimine ei tohi jääda lõpmatusse tsüklisse).
14. Rakendus peab suutma kompileerida programme, mis on kirjutatud keeltes:
    a. *Python* 2 alates versioonist 2.6;
    b. *Python* 3 alates versioonist 3.0.

3.3 Süsteemi arhitektuur

Automaatse hindaja süsteem koosneb järgnevatest komponentidest (Joonis 1):

- Veebiserver;
- Andmebaas;
- Ülesannete kontrollija (ÜK);
- Plagiarismi kontroll (PK);
- Klient.

Täpsemalt kirjutatakse nendest komponentidest järgnevates punktides. Seletatakse, mis on nende komponentide täpsem eesmärk, ülesehitus, roll ja kuidas need toimivad.



Joonis 1. Automaatse hindaja arhitektuur.

Kõik eelmainitud komponendid (va. Klient) saab paigaldada nii ühte arvutisse kui ka kõik

komponendid paigaldada erinevatesse arvutitesse. ÜK moodulit paigaldatakse ainult *Linux*'i operatsioonisüsteemile. Ülejäänud komponente (va. Klient) saab teoreetiliselt installeerida ka teistesse operatsioonisüsteemidesse, aga hetke realisatsioon nõuab *Linux'*i operatsioonisüsteemi. Installeerimist on katsetatud *Linux*'i *Red Hat* distributsioonil, kuid ka muud viimased *Linux'*i distributsioonid peaksid sobima, kui nendele installeerida vajalikud tarkvarapaketid. Automaatse hindaja installeerimist kirjeldatakse punktis 4.1.

AHUT programm on arendatud kasutades järgnevaid tehnoloogiaid:

- *Java*;
- *Javascript*;
- *Python*;
- *CSS*;
- *HTML*.

AHUT autorid otsustasid kasutada ülalolevaid tehnoloogiaid, sest kogemusi arvestades valiti need, millega varem on kokkupuude olnud.

Selles peatükis selgitati süsteemi üldarhitektuuri. Järgnevates punktides kirjeldatakse süsteemi komponente põhjalikumalt.

Veebiserver

Veebiserveri tarkvarana on AHUT rakendusel kasutatud viimast *Apache Tomcat'*i seitsmendat versiooni. Automaatse hindaja rakenduses on valitud *Tomcat*, sest see on vabavaraline tarkvara, toetab mitmeid operatsioonisüsteeme ning kasutab serverikeelena *Java* keelt, mille abil on AHUT arendatud.

Klient ühendab arvuti veebiserveriga krüpteeritud ühendusega ehk *SSL*-iga, et tõsta rakenduse turvalisust. Veebiserver pakub kliendile nii staatilist kui ka dünaamilist veebisisu. Veebisisuks on AHUT haldussüsteem, kus külastaja navigeerib ning saab läbi viia erinevaid toiminguid vastavalt tema õigustele.

Veebiserver on pidevas suhtluses andmebaasiga. Veebiserver kasutab andmebaasi järgnevate andmete lugemiseks ja salvestamiseks:

- Kasutajad ja nende rollid;
- Kursused;
- Ülesanded;

- Programmeerimisülesannete lahendused;
- Logid.

Nende andmete lugemisel valmistatakse ette veebisisu, mida esitatakse kliendile. Andmete kirjutamisel on kaks eesmärki: (1) andmete salvestamine, et neid tulevikus lugeda saaks; (2) edastada kasutajate ülesannete lahendusi ÜK moodulile. Veebiserver salvestab kasutaja lähtekoodi andmebaasi ning omistab sellele seisundi "kontrollimata", mille abil ÜK hiljem kontrollimata lahendused üles leiab. ÜK ja veebiserveri vaheline suhtlus toimub asünkroonselt.

Andmebaas

Andmebaasi tarkvaraks on valitud *MySQL,* seda samasugustel põhjustel, nagu osutus valituks *Apache Tomcat* (vabavaraline, toetab mitmeid operatsioonisüsteeme). Andmebaas täidab AHUT rakenduses andmekandja ja suhtlusvahendi rolli. Andmebaas vahendab suhtlust veebiserveri, ÜK ja PK moodulite vahel. AHUT andmebaasi mudelis on 13 erinevat olemitüüpi (Joonis 2). Järgnevates lõikudes kirjeldatakse lühidalt iga olemitüübi eesmärki.

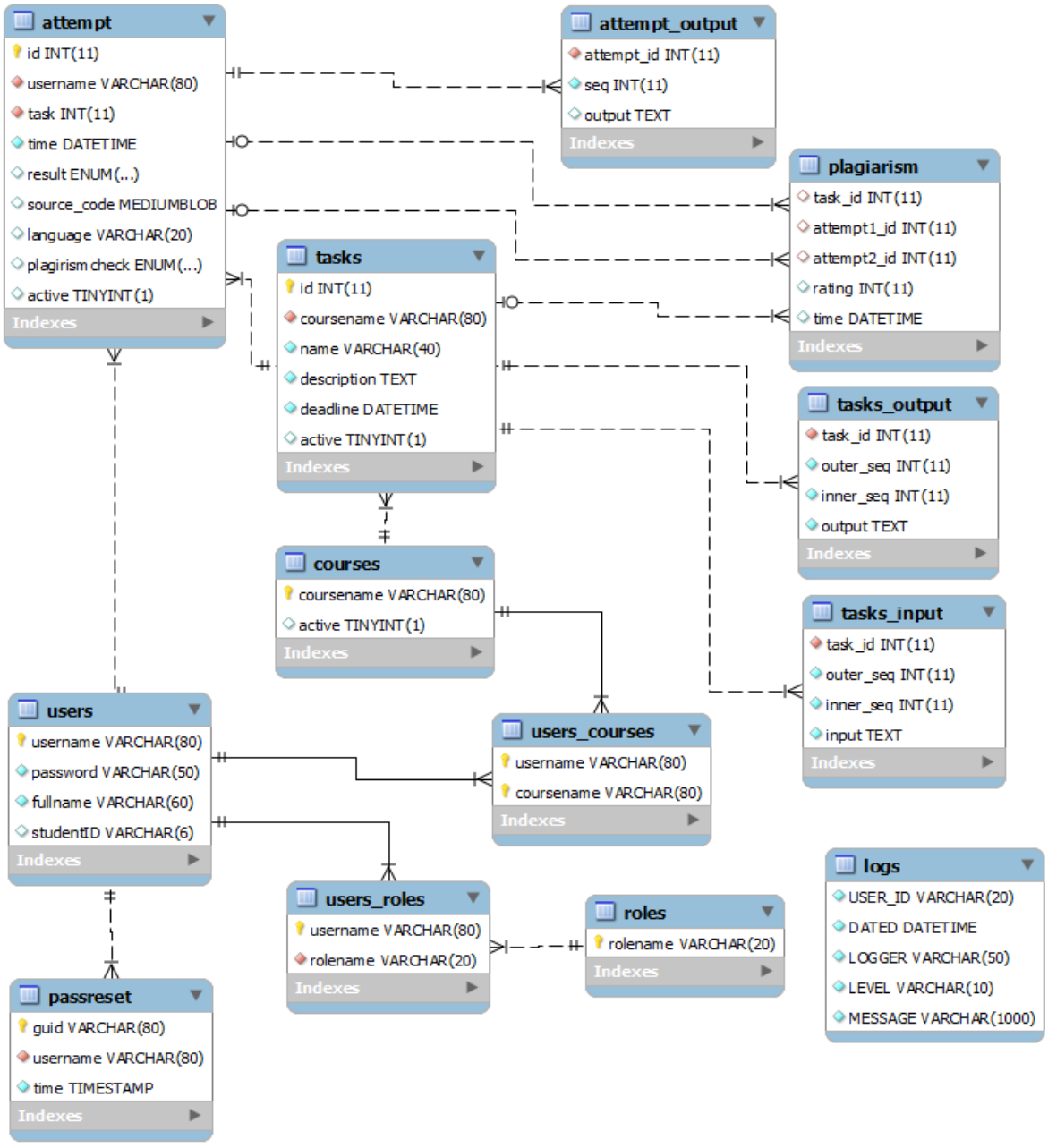
Käesolev lõik tutvustab olemitüüpe, mis on enamasti seotud kasutajate andmetega. Kasutajate andmete hoiustamiseks on tabel "*users*", kus hoitakse kasutajanime, parooli, matriklinumbrit ja pärisnime. Kasutajate rolle kirjeldab tabel "*users_roles*", kus hoitakse kasutaja rolle AHUT rakenduses. Tabelis "*roles*" on kirjeldatud kasutajate rollid, milleks on "*administaator*", "*õppejõud*" ja "*tudeng*". Tabel "*users_roles*" ja „*roles"* on vajalik *Tomcat*'i süsteemi jaoks, et tuvastada sisse logitud kasutaja roll veebirakenduses. Tabelis "*users_courses*" on kirjeldatud kasutajate kuuluvust erinevatesse kursustesse. Kasutajale määratud kursuste arv ei ole piiratud. Olemitüüpi "*passreset*" kasutatakse kasutaja parooli taastamiseks.

Ülesande haldusega on kõige rohkem seotud tabelid "*tasks*", "*tasks_output*", "*tasks_input*". Tabelis "*tasks*" hoitakse erinevate ülesannete andmeid, näiteks ülesande kirjeldus, tähtaeg, nimi ja seonduv kursus. Kuna ülesandel võib olla mitmeid sisendi- ja väljundikomplekte, siis on vajalik salvestada sisendid ja väljundid erinevatesse tabelitesse ning siduda need seoste abil ülesandega. Siinkohal tulevadki kasutusse "*tasks_output*" ja "*tasks_input*" tabelid, kus hoitakse vastavalt ülesande sisendeid ja väljundeid.

Kasutajate ülesannete lahendused hoitakse tabelis "*attempt*". Sinna salvestatakse kasutajanimi, ülesande ID, tulemus, plagieerimiskontrolli staatus, lähtekood, lähtekoodi programmeerimiskeel ning arhiveerimistunnus. Kui kasutaja lahendus on käivitatud, siis olenemata

tulemusest salvestatakse kasutaja programmi väljund tabelisse "*attempt_output*". Eraldi tabelid väljundite jaoks on vaja selleks, et tudengi lahendust käivitatakse niipalju kordi kui on ülesandel sisend- ja väljundkomplekte. Iga SV komplekti jaoks on vaja salvestada ka tudengi programmi väljund. Lahendustega on seotud ka tabel "*plagiarism*", kuhu PK moodul salvestab tulemused, mis leiti kahe lähtekoodi võrdlusel.

Erinevaid kursusi hoitakse tabelis "*courses*". Sinna salvestatakse kursuse nimi ja arhiveerimistunnus. Veel üheks eraldiseisvaks tabeliks on "*logs*", kuhu salvestatakse erinevaid logikirjeid hilisemaks analüüsimiseks.



Joonis 2. AHUT andmebaasi mudel.

AHUT programmis omab andmebaas väga tähtsat rolli, sest see tagab andmete säilitamise ja suhtluse erinevate komponentide vahel.

Plagiarismi kontroll

PK on moodul, mis kasutab Vineet Nairi loodud "*Detect plagiarism*" [11] programmi lähtekoodi. PK on kirjutatud programmeerimiskeeles *Python* 2.

PK tööpõhimõte seisneb selles, et alamstringide meetodil võrreldakse kaht erinevat lähtekoodi. Vastavalt sellele, kui palju on sarnaseid alamstringe (nt kolm järjestikku sõna on mõlemas töös), arvutab PK nendele kahele lähtekoodile plagiarismiskoori (hinde), mis on vahemikus 0-100. Skoor on vastavuses sellega, kui tõenäoline on, et ühe ülesande lahendus on teise pealt kopeeritud. Skoor 100 näitab, et kõik alamstringid kattusid, skoor 0 näitab, et ükski alamstring ei kattunud.

PK käivitatakse koos teiste komponentidega ning see jääb pidevalt kontrollima andmebaasi, kas on lisatud uusi ülesandeid, millel on plagieerimiskontrolli staatus "kontrollimata". Ajavahemikku, mille tagant andmebaasi kontrollitakse, saab muuta konfiguratsioonifailis. Kui leidub kontrollimata ülesanne, siis alustab PK moodul võrdlust sama ülesande kõikide teiste tudengite esitatud lähtekoodidega. Juhul kui plagiarismiskoor ületab rakenduse seadistatud väärtust, salvestatakse antud skoor koos viidetega lahendustele andmebaasi plagiarismi tabelisse. Kuna PK mooduli tööhulk on ruutvõrdelises seoses lahenduste koguarvuga ülesandel, siis saab konfiguratsioonifailis määrata ülempiiri, mitut tööd võrreldakse äsja esitatud lähtekoodiga. Kui ülesandele esitatud lahenduste arv ületab eelmainitud ülempiiri, siis valitakse konfigureeritud arv viimati esitatud tudengite töödest.

Ülesannete kontrollija

ÜK moodul on kirjutatud sarnaselt PK moodulile *Python* 2 programmeerimiskeeles. ÜK ülesandeks on kontrollida kasutajate esitatud programme. ÜK moodul kasutab ülesannete kontrollimiseks standardset sisend- ja väljundvoogu. Sisendvooga antakse tudengi programmile ette ülesande sisendid ning hiljem kontrollitakse, kas üliõpilase programmi väljundvoog vastas oodatavatele väljunditele.

Moodul töötab järgneva protseduuri alusel:

1. Moodul otsib andmebaasist kasutajate üles laetud lahendusi, mis on kontrollimata:
    a. kui leidub selline, siis moodul liigub punkti 2.

b. kui sellist ei leidu, siis oodatakse konfiguratsioonifailis ettemääratud aeg ning minnakse tagasi punkti 1.

2. Laetakse alla andmebaasist lähtekood koos vastava ülesande sisendite ja väljunditega.
3. Lähtekood kompileeritakse. Kui tegevus ebaõnnestub, kirjutatakse andmebaasi tulemus "kompileerimise viga" ja minnakse tagasi punkti 1.
4. Käivitatakse programm (järgneva) sisendikomplektiga ning kontrollitakse, kas programmi väljund vastab väljundkomplektile. Kui
    a. programmi väljund vastab oodatavale väljundkomplektile ja rohkem SV komplekte ei leidu, siis kirjutatakse tulemus "OK" ning minnakse punkti 1.
    b. programmi väljund vastab oodatavale väljundikomplektile, aga leidub rohkem SV komplekte, siis minnakse tagasi punkti 4.
    c. programmi väljund ei vasta oodatavale väljundkomplektile, siis kirjutatakse andmebaasi tulemus "Vale vastus" ning minnakse punkti 1. Järgnevaid SV komplekte ei kontrollita.
    d. programm ei lõpetanud oma tööd konfiguratsioonifailis määratud ajapiiri sees, siis kirjutatakse andmebaasi tulemus "*timeout*" ning minnakse punkti 1. Järgnevaid SV komplekte ei kontrollita.

ÜK moodul töötab lõpmatus tsüklis eeloleva protseduuri alusel. ÜK moodul hindab ülesande lahendust tervikuna ning eraldi SV komplektidele tagasisidet ei anna.

Klient

Kliendiks on AHUT süsteemis kasutaja, kes on veebilehitseja abiga navigeerinud AHUT veebilehele. AHUT süsteemi poolt toetatud veebilehitsejad on eelnevalt kirjeldatud mittefunktsionaalsetes nõuetes.

Eelnevates punktides vaadeldi AHUT süsteemi arhitektuuri ja tarkvaralisi valikuid. Järgnevalt kirjeldatakse AHUT programmi funktsionaalsust ja automaatse hindamise põhimõtteid.

3.4 Automaatse hindamise põhimõtted

Järgnevates punktides kirjeldatakse AHUT süsteemi funktsionaalsust samade tunnuste alusel nagu punktis 2.2 vaadeldud AH süsteeme.

Programmeerimiskeelte toetus

AHUT toetab hetkel *Python* 2 alates versioonist 2.6 ning *Python* 3 alates versioonist 3.0. Tartu Ülikoolis õpetatavad ained "Programmeerimine" ning "Algoritmid ja andmestruktuurid" kasutavad *Python*'i keelt programmeerimise õpetamiseks ning just nende kursuste jaoks on AHUT loodud.

Kuna lahenduste kontrollimine toimub SV baasil, siis ei ole raske toetatavate keelte hulka lisada teisi programmeerimiskeeli. Vähese programmeerimiskoodi lisamisega saab tuua AHUT programmi need programmeerimiskeeled, mis on installeeritud serverile ning toetavad standardset SV voogu.

Haldussüsteem

AHUT süsteem kasutab enda loodud veebipõhist haldussüsteemi, seega igaüks, kellel on ligipääs arvutile ja internetiühendusele, saab AHUT süsteemi kasutada. Eristatakse kolme erinevat kasutajaklassi, milleks on õppejõud, tudengid ning administraator. Igale klassile on omistatud omad kasutusõigused. Haldussüsteemis on mitmeid võimalusi:

- kasutajahaldus kasutajate loomiseks, nendele õiguste andmiseks ning kursustele kaasamiseks;
- ülesannete haldus ülesannete loomiseks, arhiveerimiseks, kopeerimiseks ning kustutamiseks;
- kursuste haldus kursuste loomiseks ja arhiveerimiseks;
- tudengite tulemuste vaatamine ning lahenduste uurimine;
- plagiarismikontroll.

Haldussüsteem on AHUT lahutamatu osa, sest ainult ülesandeid kontrollides ei annaks AHUT suurt ajavõitu võrreldes manuaalse hindamisega. Vastasel korral peaksid õppejõud ikkagi manuaalselt üliõpilaste esitusi AHUT süsteemi sisestama ning tagasisidesid kirjutama. Kuna *Moodle* pistikprogrammide arendamisega polnud projekti autorid kursis, siis alustatigi oma süsteemi loomist nullist. Tulemuseks on suhteliselt keerukas halduskeskkond, mis peaks rahuldama kõigi soovijate vajadusi.

Ülesannete defineerimine ja kontrollimine

Ülesannete kontrollimiseks ja defineerimiseks on kasutatud SV meetodit. Seda põhjusel, et piiratud ajaga rakenduse loomisel oli selle meetodi kasutamine kõige kergem ja samas ka kõige kiirem variant. Lisaks on sellisel juhul ka ülesannete defineerimine kergem ning hiljem on lihtne juurde lisada teiste keelte tugesid. Negatiivse näitena võib välja tuua, et programm kontrollib ainult tudengi sisendeid ja väljundeid, tegelikku koodi üle ei vaadata. Et kontrolli väärtust tõsta, on õppejõul võimalus defineerida ühele ülesandele mitu kontrolli. Selle võimalusega käivitatakse tudengi programm mitu korda erinevate sisenditega ja kontrollitakse saadud väljundeid. See vähendab võimalust, et tudeng saab ebakorrektse lahendusega ülesande läbitud. Kui lahendus ühe sisendiga ei anna korrektset väljundit, loetakse lahendus üheselt valeks, sest selles olukorras ei töötanud programm korrektselt.

Tudengite lahendused peavad olema vormistatud spetsiifiliselt, et AHUT programm neid hinnata saaks. Etteantavad sisendid peavad tudengite programmid lugema sisse ülesandes kirjeldatud viisil (nt *Python* 3 "*input()*" meetodiga ühe pika stringina, kus erinevad väärtused on eraldatud komaga). Kontrollitavad programmid ei tohi väljastada mitte midagi muud peale ülesandes kirjeldatud väljundi (nt "print" käsku kasutades väljastades ainult leitud vastuse). Näide ülesandest ning seal kirjeldatud sisendite ja väljundite kohta leiab jooniselt 6 leheküljel 27.

Tudengi lahendusele etteantavad sisendid on tema jaoks peidetud. Vastasel juhul saaks tudeng luua programmi, mis väljastab ainult õige vastuse, ilma tegelikku lahenduseni jõudmata (nt kasutab internetis olevat rakendust õigete vastuste saamiseks).

Ülesannete esitamine

Automaatse hindaja rakenduses on töö esitamise aeg piiratud. Ülesandeid võib esitada ka pärast tähtaega, kuigi sellest jääb märge maha nii tudengile endale kui ka õppejõule. Taasesitamiste arv ei ole piiratud.

Tagasiside sõnumid lahendustele on:

- "Kompileerimise viga" - AHUT saab tudengi programmi käivitamisel kompileerimise vea.
- "Vale vastus" - tudengi programmi vähemalt üks väljund ei vasta oodatavatele väljunditele.
- "OK" - tudeng on ülesande sooritanud.

- "*Timeout*" - tudengi programm ei lõpetanud tööd ettenähtud ajapiiri sees.
- "Kontrollimisel" - tudengi programm on kontrollimisel.
- "Kontrollimata" - tudengi programmi pole veel kontrollima hakatud.

Põhjus, miks tagasiside pikem ei ole, on selles, et programm ei analüüsi koodi, vaid ainult väljundit. Seega on ka tagasiside limiteeritud. Tagasiside piiratus võiks suurendada tudengite eneseanalüüsi oma koostatud programmi üle ning panna rohkem mõtlema ülesande õige lahenduse peale.

Manuaalne ülesannete hindamine

Manuaalselt on võimalik üliõpilasi hinnata käsitsi. Üliõpilaste lahenduste lähtekoode saab nii veebis üle vaadata kui ka failina alla laadida ning see siis oma arvutis käivitada. Lähtekoodi manuaalne ülevaatus lubab õppejõul kontrollida lahendust käsitsi, et teha kindlaks, kas tudengi lahendus on korrektne. Samuti annab see õppejõule võimaluse võrrelda lähtekoode kahe töö plagiarismikahtluse korral. Lisaks lähtekoodi ülevaatusele on õppejõul võimalus vaadata tudengi lahenduse väljundeid. See loob võimaluse vajaduse korral tudengitele juhiseid anda.

Turvapoliitika

Automaatse hindaja rakendus kasutab tudengite ülesannete turvaliseks käivitamiseks *Linux*'i kasutajaõiguste süsteemi. *Linux* operatsioonisüsteemi on loodud uus kasutaja, kellel on õigus lugeda, kirjutada ja käivitada ainult tudengi lähtekoodiga faili. Lisaks on võetud kasutajalt õigus kasutada võrku. See peaks minimeerima võimalused, et tudeng saaks kas tahtlikult või kogemata süsteemi halvata.

Kättesaadavus ja levitamine

Antud süsteem on Tartu Ülikooli omand ning edasise leviku ja kättesaadavuse eest vastutab ja otsustab Tartu Ülikool. Hetkel on see vabalt kättesaadav *GitHub*'i repositooriumist nimega "automaatnehindaja" [12].

Eelnevates lõikudes kirjeldati AHUT programmi funktsionaalsust. Järgnevalt tutvustatakse, kuidas AHUT programmi kasutada.

4 AHUT kasutamine

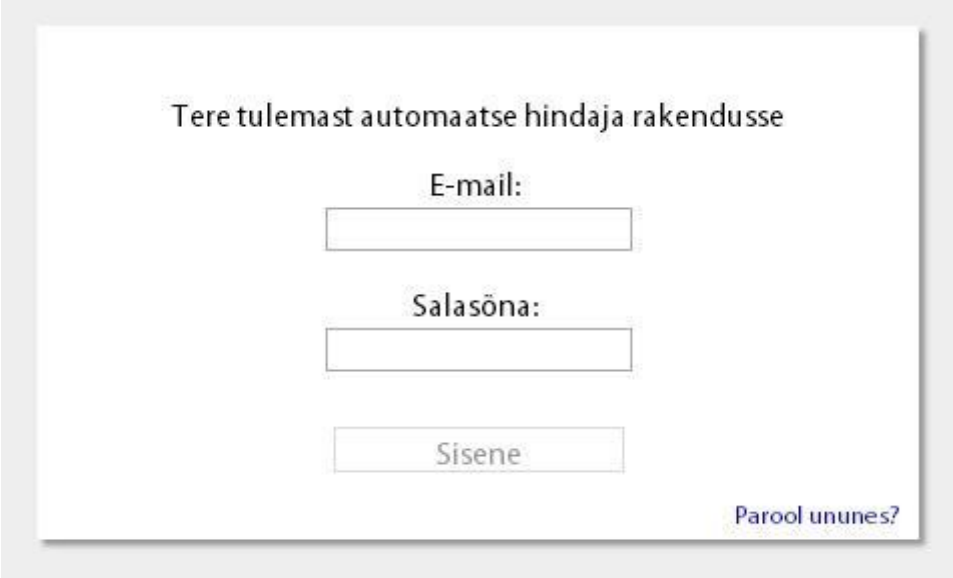
Selles peatükis tutvustatakse AHUT kasutamist, esitletakse AHUT kasutajaliidest ning antakse näpunäiteid selle kasutamiseks.

4.1 Installeerimisjuhend

AHUT installatsioon ei ole automatiseeritud ning seetõttu on AHUT installeerimiseks vaja osata kasutada *Linux*'i käsurida. Installatsioonijuhend *Linux Red Hat* distributsioonile on saadaval *Github* automaatse hindaja repositooriumi vikis [13].

4.2 Ülevaade AHUT kasutajaliidesest

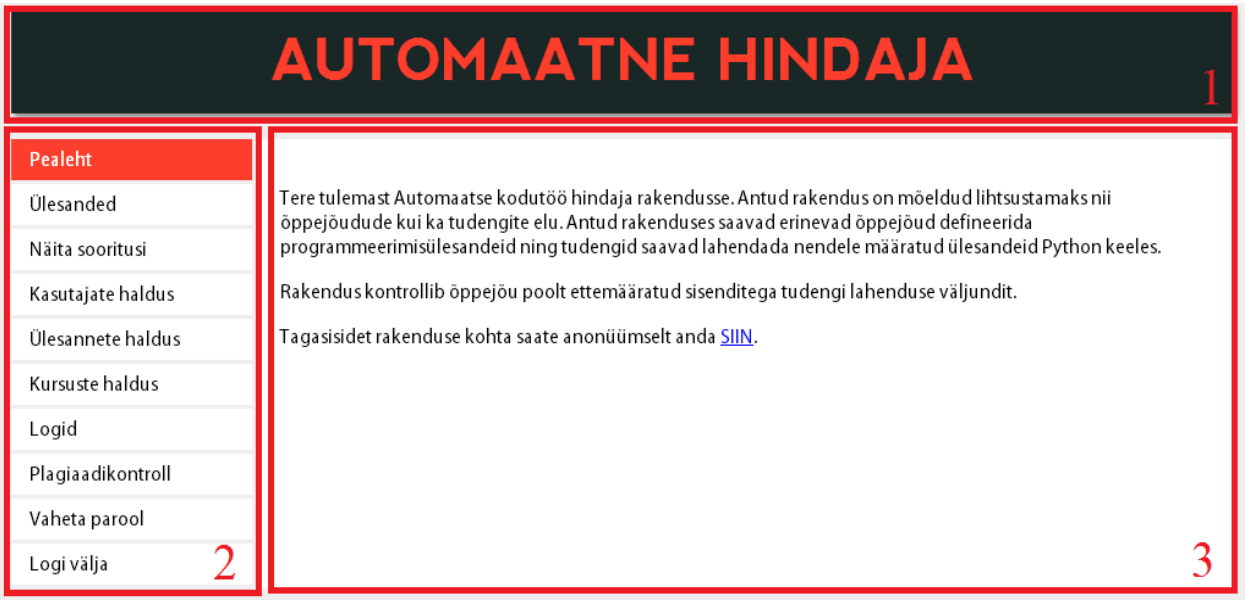
AHUT sisselogimise vaade võimaldab kasutajal end süsteemis identifitseerida (Joonis 3). Kasutaja peab sisestama oma e-maili ja parooli. Kui kasutajal on parool meelest läinud, siis saab ta taastada parooli kasutades sisselogimise vaate viita "Parool ununes?". Parooli ununemisel saadetakse kasutaja e-mailile link, kus ta saab oma parooli vahetada ilma, et ta peaks sisestama vana parooli. Kasutajatunnuse ja parooli edastamiseks süsteemi kasutatakse krüpteeritud ühendust.



Joonis 3. AHUT sisselogimise vaade.

AHUT rakendusel on veebipõhine kasutajaliides. Erinevad kasutajatasemed määravad ära kasutajaliidesel kuvatavad võimalused, kuid ülddisain on kõigil kasutajatasemetel sarnane. Allpool on kuvatud joonis AHUT kasutajaliidesest (Joonis 4) ning punase kastiga on näidatud erinevad elemendid AHUT kasutajaliideses.

1. AHUT kasutajaliidese päis. Tegemist on ilma funktsioonita disainielemendiga.
2. AHUT peamenüü. Siin saab kasutaja valida erinevate funktsioonide vahel.
3. AHUT sisu. Siia kuvatakse peamenüüst valitud sisu.



Joonis 4. AHUT kasutajaliides administraatori vaates.

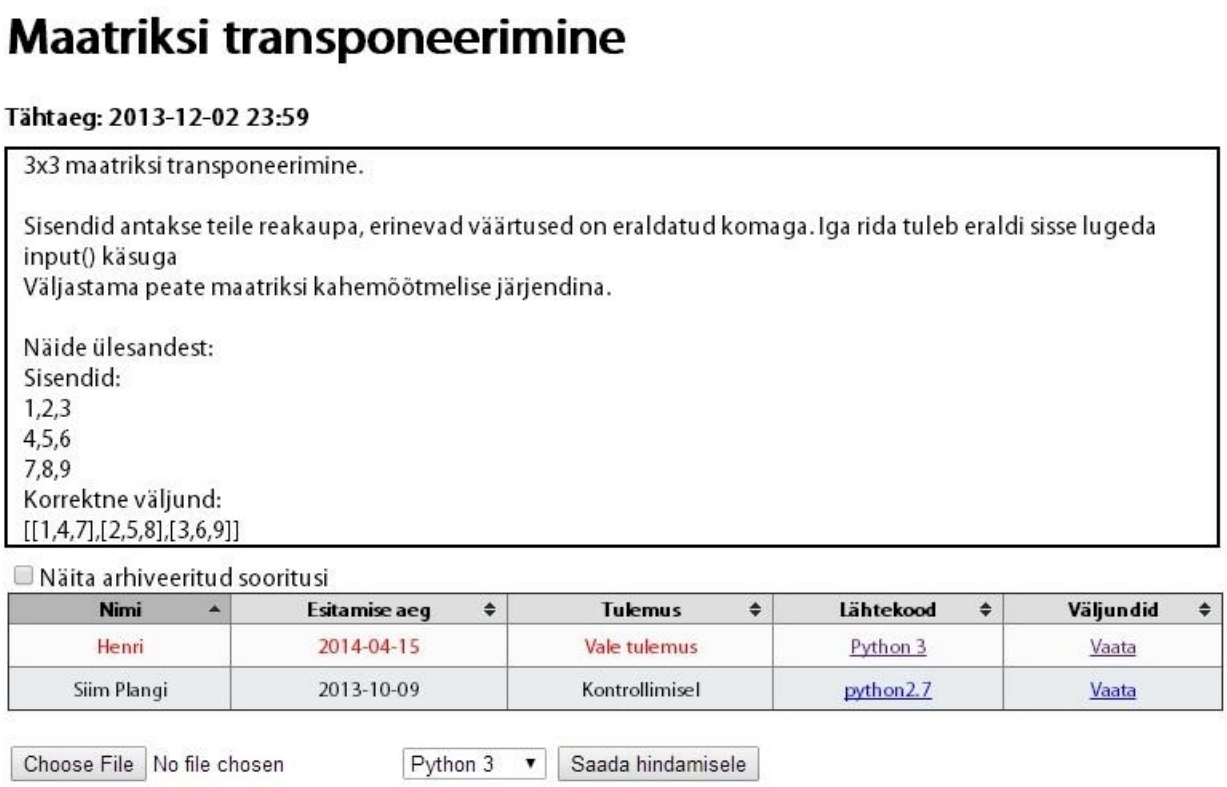
Järgnevalt tutvustatakse AHUT rakenduse peamenüüs olevaid menüüelemente, nende võimalusi ja sisu.

Ülesanded

Peamenüüs olev element "Ülesanded" on nähtav kõikidele kasutajatasemetele. Ülesannete kuval näidatakse kõiki valitud kursuse ülesandeid (Joonis 5).

Joonis 5. AHUT ülesannete rubriigi sisuvaade.

Ülesandel vajutades kuvatakse ülesande vaade (Joonis 6), kus näidatakse ülesande pealkirja, tähtaega ja kirjeldust.



Joonis 6. AHUT ülesande vaade administraatorina.

Lisaks sellele näidatakse sama ülesande varasemaid sooritusi. Kui kasutaja on tudengi staatuses, siis näidatakse talle ainult tema enda sooritusi, kui kasutaja on õppejõu või administraatori staatuses, siis näidatakse talle kõiki kasutajaid, kes antud ülesannet sooritanud on. Soorituste tabelis saab vaadata ka esitatud lähtekoodi ning õppejõud ja administraator ka soorituse väljundeid.

Näita sooritusi

Rubriigis "Näita sooritusi" saab tudeng näha kõiki oma sooritusi antud kursusel ja õppejõud/administraator näeb kõikide kasutajate sooritusi valitud kursusel (Joonis 7).

Joonis 7. AHUT „Näita sooritusi" sisu administraatori vaates.

Antud rubriik annab tudengile ja õppejõule ülevaate kursusel toimuvast.

Kasutajate haldus

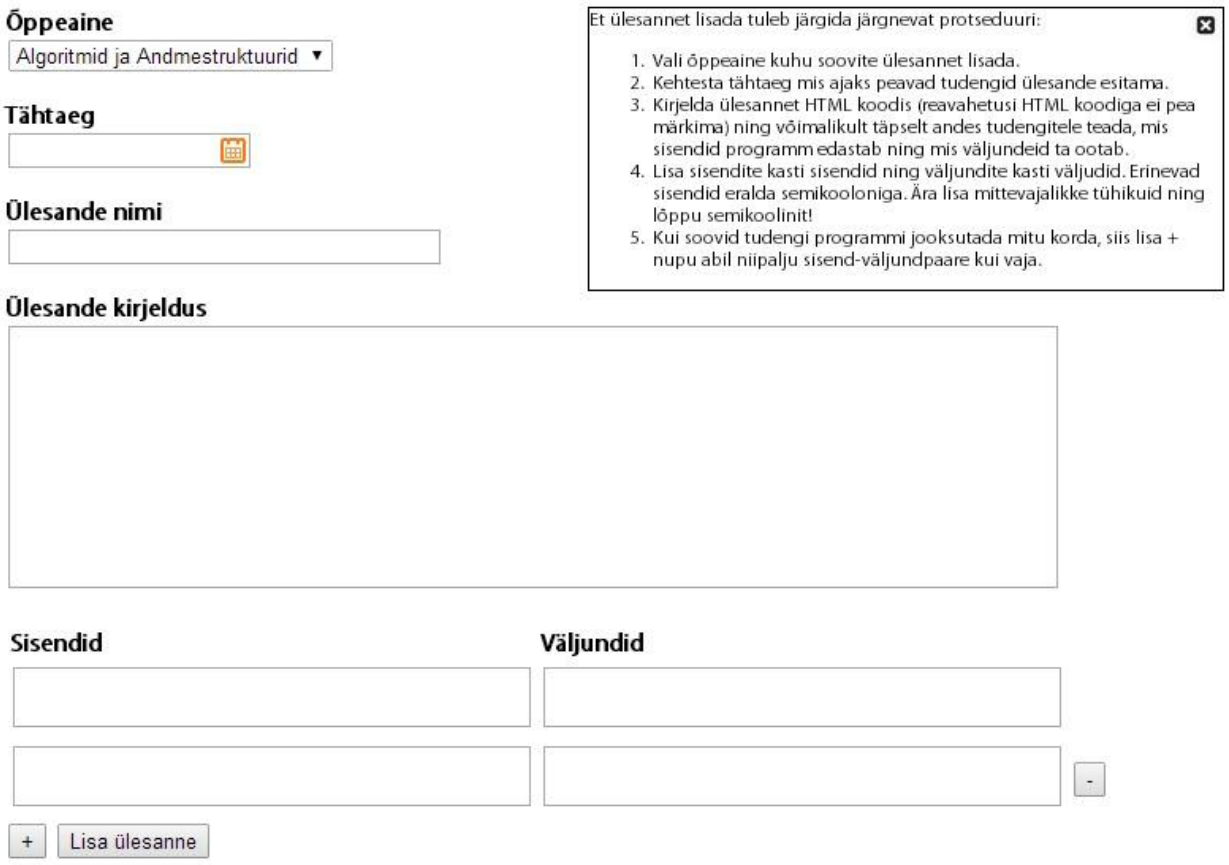
Kasutajate halduse menüüelement on nähtav õppejõule ja administraatorile. Selles menüüs on kolm alamelementi (Joonis 8), milleks on: (1) “Kasutaja lisamine”; (2) “Automaatne lisamine” ja (3) “Lisa kasutaja kursusele”.

Joonis 8. AHUT kasutajate haldusmenüü.

Kasutajate lisamise vaates saab kasutajaid lisada süsteemi käsitsi. Automaatse lisamisega saab kasutajaid lisada automaatselt Microsoft Office Exceli failiga. Selleks on vaja eksportida tudengite nimekiri ülikooli Õppeinfosüsteemist (ÕIS) Exceli failina ning seejärel see süsteemi üles laadida. Menüü “Lisa kasutaja kursusele” abil saab olemasolevaid kasutajaid siduda olemasolevate kursustega.

Ülesannete haldus

Ülesannete haldus on nähtav õppejõule ja administraatorile. Selles menüüs on neli allelementi, milleks on: (1) “Lisa ülesanne”; (2) “Muuda ülesanne”; (3) “Soorituste arhiveerimine” ja (4) “Ülesannete arhiveerimine”. “Lisa ülesanne” rubriik on mõeldud uute ülesannete lisamiseks (Joonis 9). Ülesande lisamiseks tuleb täita kõik väljad, õpetus ülesande lisamiseks on lisatud kuva paremasse ülemisse äärde.



Joonis 9. AHUT „Lisa ülesanne“ vaade.

“Muuda ülesanne” vaade on sarnane “Lisa ülesanne” vaatega, ainult et nüüd täidetakse kõik lahtrid eelnevalt sisestatud ülesande andmetega. “Soorituste arhiveerimise” rubriigis saab kõik tudengite esitatud tööd arhiveerida, jättes ülesande ise aktiivseks. “Ülesande arhiveerimise” vaates saab ülesannet arhiveerida, kusjuures sel juhul arhiveeritakse ka kõik sooritused.

Kursuste haldus

“Kursuse halduse” menüüelementi näeb ainult administraator ning ainult temal on lubatud kursuseid hallata. Sellel menüüelemendil on kaks allelementi, milleks on “Lisa kursus” ja “Kursuse sulgemine”. “Lisa kursuse” vaates saab lisada kursuseid, täites ära kursuse nime lahtri ning seejärel vajutades “Lisa kursus” nuppu. “Sulge kursus” on mõeldud kursuste arhiveerimiseks, kusjuures arhiveerimisel arhiveeritakse ka kõik selle kursuse ülesanded ning nende ülesannete sooritused.

Logid

Logide vaade on nähtav ainult administraatorile, selles näeb erinevaid toiminguid, mida

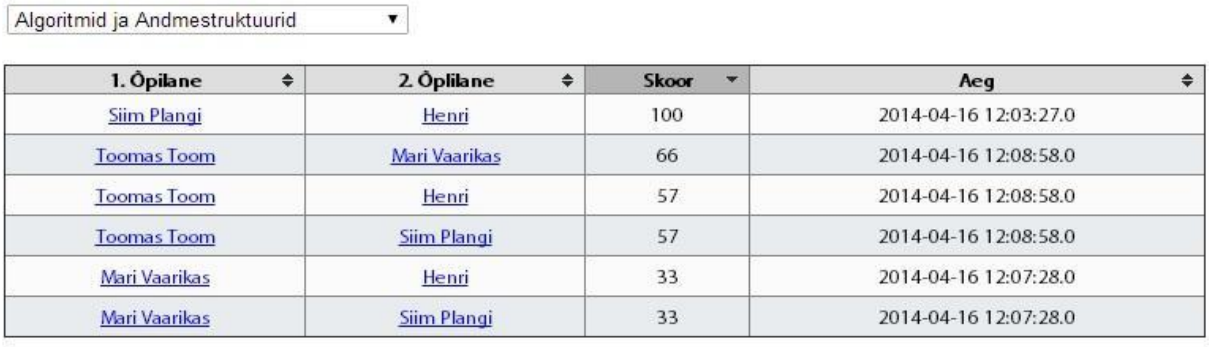
veebilehel sooritatakse (Joonis 10). Näiteks logitakse lahenduste üleslaadimisi, nende kontrollimise alustamist ja lõpetamist. Lisaks logitakse veel serveri sisemisi vigu. Kui tudeng esitab programmi, mis võiks tahta süsteemi halvata, siis jääb sellest logifaili märge ning hiljem on võimalik isik tuvastada.

Joonis 10. AHUT logide vaade.

Logid annavad parema võimaluse leida süsteemist vigu ning neid ka kiiremini parandada.

Plagiaadikontroll

"Plagiaadikontroll" on nähtav nii administraatorile kui ka õppejõule. Plagiaadikontrolli vaade on realiseeritud tabelina (Joonis 11), kus on mõlema võrreldava töö esitaja, plagiaadiskoor ja võrdluse aeg.



Joonis 11. AHUT plagiarismikontrolli vaade.

Esimeses ja teises tulbas on näha kahe võrreldava tudengi nimi. Kolmandas tulbas on näha plagiarismiskoor ning neljandas tulbas võrdluse aeg. Vajutades nimele saab näha esituse lähtekoodi ning nii on võimalik teha ka manuaalne ülevaade, otsustamaks kas tegemist on plagiarismiga või mitte.

Parooli vahetamine

Viimaseks vaateks AHUT menüüs on parooli vahetamine. Parooli vahetamiseks peab kasutaja sisestama on praeguse parooli ning kaks korda uue parooli. Uute paroolide mittekattumisel katkestatakse parooli vahetus.

Eelnevates punktides näidati kuidas AHUT programmi kasutada. Järgmises peatükis tutvustatakse AHUT süsteemi testimise tulemusi.

5 Süsteemi testimine

Selles peatükis tutvustatakse süsteemi testimisel saadud tulemusi. Süsteemi testiti Tartu Ülikooli ainete "Programmeerimise alused I" ja A&A ülesannete lahendustega, mille tudengid olid õppejõule esitanud. "Programmeerimise alused I" on programmeerimise algkursus, kus tudengid õpivad programmeerima *Python* keeles. Mõlemad ainekursused toimusid enne käesolevat testimist ning seega saadi tudengite lahendused vastavate ainete õppejõudude käest. Testimise käigus käsitleti tudengite lahendusi anonüümsetena.

Süsteemi testimiseks valiti välja kaks ülesannet ainest "Programmeerimise alused I" ja üks keerulisem ülesanne A&A kursuselt. Edasi valiti juhusliku valiku alusel iga ülesande kohta tudengite lahendused. Ülesanded ja väljavalitud lahenduste arvud olid järgnevad:

- Aastaarvu järgi liigaasta tuvastamine (ülesanne 1) - 10 lahendust
- 0-99 arvu väljatrükk eesti keele kirjapildis (ülesanne 2) - 10 lahendust
- *Infix*-i teisendus *Postfix*-i (ülesanne 3) - 5 lahendust

Ükski valitud lahendustest polnud valmistatud AHUT kontrollimiseks, seega tuli lahendusi teataval hulgal modifitseerida, et AHUT neid lahendusi korrektselt kontrollida saaks. Lahenduste modifitseering piirdus üleliigsete väljundite eemaldamisega, sisendite lugemiseks lisati "*input()*" käsud ning lisati lahendi väljastus vajalikul kujul.

5.1 Testimise protseduur

AHUT programmi testitakse järgneva protseduuri alusel. Alustuseks vaadatakse valitud tudengite esitused läbi manuaalselt ning kontrollitakse nende toimimist valitud sisenditega. Kui kõik väljundid on võrdsed oodatavate väljunditega, antakse tudengi lahendusele hinnang "OK". Valede vastuste ilmnemisel "Vale tulemus" ja kompileerimisvea korral "Kompileerimise viga". Seejärel sisestatakse tudengite lahendused AHUT programmi (AHUT-s on defineeritud ülesanne samade sisendite ja oodatavate väljunditega) ning AHUT programmi hinnanguid võrreldakse manuaalsete hinnangute väärtustega. Sisendid on valitud juhuslikult, oluline pole mitte testida tudengite lahendusi, vaid seda, kas AHUT suudab hinnata lahendusi samamoodi nagu manuaalne hindamine.

5.2 Ülesande 1 testimise tulemused

Tegemist on "Programmeerimise Alused I" aine raames lahendatava ülesandega. Ülesande

idee on lihtne, tuleb leida, kas sisendina antud aastaarv on liigaasta või mitte. Testimiseks käivitati iga tudengi lahendust mitu korda, iga kord anti lahendusele erinev sisend. Allolevas tabelis (Tabel 1) on näidatud sisendid, mis tudengi programmidele anti ja tudengi programmide oodatavad väljundid.

Tabel 1. Testimisel kasutatud sisendid ja oodatavad väljundid ülesandel 1.

Sisend	**Oodatav väljund**
1900	mitte liigaasta
1904	liigaasta
36	liigaasta
3	mitte liigaasta
2000	liigaasta

Manuaalsel läbivaatusel selgus, et kümnest juhuslikult valitud lahendusest väljastasid oodatavad väljundid kaheksa. Kahel juhul oli tegemist vale vastusega, sest ei olnud arvestatud erandeid liigaasta tuvastamisel. AHUT süsteem suutis tuvastada kõik kaheksa korrektset lahendust ning kaks ebakorrektset lahendust, seega läbis testi 100% edukusega.

5.3 Ülesande 2 testimise tulemused

Sarnaselt eelnevas punktis mainitud ülesandele oli ka see "Programmeerimise alused I" raames käsitletud ülesanne. Tudengi programmile antakse sisendiks arv 0-99 ning tudengi programm peab trükkima välja selle arvu eestikeelse kirjapildi. Allolevas tabelis (Tabel 2) on näidatud manuaalsel läbivaatusel ja AHUT-s defineeritud sisendid ja oodatavad väljundid.

Tabel 2. Testimisel kasutatud sisendid ja oodatavad väljundid ülesandel 2.

Sisend	**Oodatav väljund**
99	üheksakümmend üheksa

1	üks
9	üheksa
10	kümme
15	viisteist
20	kakskümmend
25	kakskümmend viis
80	kaheksakümmend
0	null

Manuaalsel läbivaatamisel selgus, et kümnest tudengite lahendusest olid korrektsed neli, kolm neist said hinnangu "vale tulemus", sest kahel juhul ei käsitletud arvu 0 väljastamist ja ühel juhul oli programmis loogika viga. Kolmel juhul oli implementeeritud ainult arvude 0-9 eesti keeles väljastamine, seega sel juhul antakse manuaalsel läbivaatlusel hinnang "vale tulemus". AHUT programm suutis ka sellel korral anda 100% samad hinnangud tudengite lahendustele.

5.4 Ülesande 3 testimise tulemused

Antud ülesanne on võetud 2013. aasta Tartu Ülikooli sügissemestri ainest "Algoritmid ja andmestruktuurid". Tegemist on teisendamisülesandega, mille eesmärgiks on teisendada *infix* kujul avaldis (nt 2+3*4) *postfix* kujule (nt. 234*+). Antud ülesande skoobist on välja jäänud kahekohalised numbrid, seega kahekohaliste numbritega sisendeid selles testimises ei kasutata. Sisendid ja oodatavad väljundid on näidatud järgnevas tabelis (Tabel 3).

Tabel 3. Testimisel kasutatud sisendid ja oodatavad väljundid ülesandel 3.

Sisend	**Oodatav väljund**
9*8/2+3	98*2/3+

3+5*(5+9)	3559+*+
(2*1)+3-8	21*3+8-
(3+6-1)/4	36+1-4/
3+3+3-3	33+3+3-

Ka seekord kattusid AHUT ja manuaalsel läbivaatusel antud hinnangud. Kaks tudengite lahendust läbisid testid vigadeta, kaks tükki ei andnud ootuspäraseid väljundeid ning üks lõpetas töö kompileerimisveaga. Kompileerimisviga tulenes võõraste *Python*'i moodulite kasutamisest, mida pole lubatud teha. Kõik A&A kursuses olevad ülesanded tuleb läbida ilma kolmanda osapoolte mooduleid kasutamata.

5.5 Testimise kokkuvõte

Testimine näitas, et AHUT süsteem toimib tudengite lahenduste kontrollimise osas korrektselt. Küll aga ilmnesid testimisel kaks juba varem käesoleva töö autorile teada olevat probleemi. Esimeseks neist on programmi jäikus kontrollimisel ehk programmis pole lubatud muid väljundeid peale tulemuste. Kui programm väljastab inimesele mingisugust teksti, loeb AHUT süsteem tulemuse valeks ning lõpetab kontrollimise. Teiseks probleemiks on AHUT programmi kontrollimise lõpetamine, juhul kui ilmneb esimene vale tulemus. Selle asemel antakse tudengi lahendusele hinnang "Vale tulemus" ning tudengil puudub arusaam, kas tema lahendus ei tööta üldse või ei tööta ainult mõningate sisenditega. Plagiarismituvastajat käesolevas töös ei testitud.

6 Kokkuvõte

Käesolev bakalaurusetöö põhineb Tartu Ülikooli Matemaatika-informaatikateaduskonna õppeaines "Tarkvaraprojekt" loodud programmil "Automaatne hindaja Tartu Ülikoolile". AHUT on rakendus, mis tulevikus võiks lihtsustada Tartu Ülikooli aine „Algoritmid ja andmestruktuurid" tudengite programmeerimislahenduste kontrollimist. Antud rakendust võib tulevikus laiendada ka teistele õppeainetele, kus tuleb töötada suure hulga tudengite ja nende esitatud lahendustega programmeerimisülesannetele.

AHUT on veebipõhine rakendus, milles õppejõud loovad tudengitele ülesandeid ning viimased esitavad oma programmeerimislahendused. AHUT kontrollib ülesandeid automaatselt ja hindab töid. AHUT koosneb viiest komponendist, milleks on: (1) veebiserver kasutajale sisu edastamiseks; (2) ülesannete kontrollija tudengite lahenduste kontrollimiseks; (3) plagiarismi kontrollija tudengite töödest plagiarismi tuvastamiseks; (4) andmebaas andmete talletamiseks; (5) klient, kes AHUT programmiga töötab.

AHUT töökindluse hindamiseks viidi läbi kontrolltestimine. Selleks koguti kokku erinevate tudengite lahendused konkreetsetele ülesannetele varasematest kursustest ning modifitseeriti neid lahendusi, et AHUT süsteem neid kontrollida saaks. Tulemused näitasid, et AHUT süsteem hindab üliõpilaste lahendusi korrektselt. Siiski ilmnesid testimisel programmi puudused. Nendeks puudusteks olid süsteemi range sisendväljund kontroll, mis muudab tudengite lahenduste sisendite ja väljundite käsitlemise tugevalt reguleerituks. Teiseks puuduseks oli süsteemi kontrollimise lõpetamine kohe kui tudengi programm väljastas esimese ebakorrektse väljundi.

Esimese probleemi lahenduseks pakub autor välja kaks võimalust. Üheks neist võiks olla tudengi lahenduses olevate funktsioonide käivitamine. Väljundvoo lugemise asemel käivitaks AHUT süsteem tudengi programmis olevaid funktsioone ning kontrolliks nende tagastatavaid väärtusi. Teiseks võimaluseks pakub autor välja väljundite erilise märgistamise, mille abil AHUT süsteem saaks aru, millisest kohast algab programmi vastus ning kus see lõpeb.

Teise probleemi lahendamiseks võiks süsteem käivitada kõik testid ning anda tudengile tulemuseks läbitud testide arvu suhte testide koguarvuga. Sel viisil näeks tudeng, kas tema programmis on tõsine viga või ainult mõnel juhul annab tema programm vale tulemuse.

7 Kasutatud materjalid

[1] Tartu Ülikool, „Tartu Ülikooli Õpilased õppekavati 2003-2013," [Võrgumaterjal]. Available: http://www.ut.ee/sites/default/files/www_ut/oppimine/yliopilased_oppekavati_2003_13.xlsx. [Kasutatud 07 05 2014].

[2] C. Douce, D. Livingstone ja J. Orwell, „Automatic Test-Based Assessment of Programming: A Review," *Journal of Educational Resources in Computing,* kd. 5, nr 2, 2005.

[3] P. Ihantola, T. Ahoniemi, V. Karavirta ja O. Seppälä, „Review of Recent Systems for Automatic Assessment of Programming Assignments," *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, 2010.

[4] J. Hollingsworth, „Automatic graders for programming classes," *Communications of the ACM,* kd. 3, nr 10, pp. 528-529 , 1960.

[5] J. B. Hext ja J. W. Winings, „An automatic grading scheme for simple programming exercises," *Communications of the ACM,* kd. 12, nr 5, pp. 272-275 , 1969.

[6] B. Merry, „Using a Linux Security Module for Contest Security," *Olympiads in Informatics,* kd. 3, pp. 67-73, 2009.

[7] M. Amelung, P. Forbrig ja D. Rösner, „Towards generic and flexible web services for e-assessment," %1 *Proceedings of the 13th annual Conf. on Innovation and technology in computer science education*, 2008.

[8] R. Saikkonen, L. Malmi ja A. Korhonen, „Fully Automatic Assessment of Programming Exercises," %1 *Proceedings of the 6th annual conference on Innovation and technology in computer science education*, 2001.

[9] S. Plangi, T. Vaher, J. Ait ja T. Salumaa, „AHUT funktsionaalsed nõuded," 2013. [Võrgumaterjal]. Available: https://github.com/splangi/automaatnehindaja/wiki/Funktsionaalsed-n%C3%B5uded. [Kasutatud 07 05 2014].

[10] S. Plangi, T. Vaher, J. Ait ja T. Salumaa, „AHUT mittefunktsionaalsed nõuded,“ 2013. [Võrgumaterjal]. Available: https://github.com/splangi/automaatnehindaja/wiki/Mittefunktsionaalsed-n%C3%B5uded. [Kasutatud 07 05 2014].

[11] V. Nair, „Plagiarism detector in python,“ 30 01 2012. [Võrgumaterjal]. Available: https://github.com/nairv/detect_plagiarism/blob/master/detect_plagiarism.py. [Kasutatud 07 05 2014].

[12] S. Plangi, T. Vaher, J. Ait ja T. Salumaa, „AHUT github repositoorium,“ 2013. [Võrgumaterjal]. Available: https://github.com/splangi/automaatnehindaja. [Kasutatud 07 05 2014].

[13] S. Plangi, T. V. J. Ait ja T. Salumaa, „AHUT installeerimisjuhend,“ 13 05 2014. [Võrgumaterjal]. Available: https://github.com/splangi/automaatnehindaja/wiki/Insalleerimisjuhend. [Kasutatud 13 05 2013].

Lisad

I. Nimikiri AH süsteemidest, mida käsitleti artiklis „Review of Recent Systems for Automatic Assessment of Programming Assignments“

1. Autograder
2. AWAT
3. CTPracticals
4. EasyAccept
5. EduComponents
6. Linuxgym
7. Moe
8. Mooshak
9. Peach
10. ProtoAPOGEE
11. Resolver
12. RoboCode
13. USACO’s
14. UVA Online Judge
15. VERKKOKE
16. Web-Cat
17. WeBWorK-JAG

II. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina **Siim Plangi** (sünnikuupäev: 16.06.1992)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose **Automaatne programmeerimisülesannete kontrollija Tartu Ülikooli kursuse "Algoritmid ja andmestruktuurid" jaoks**, mille juhendaja on Anne Villems ja Henri Lakk,

   1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is 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, **14.05.2014**