TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Eva Luude

Kursuse “Objektorienteeritud programmeerimine” rühmatööde analüüs

Bakalaureusetöö (9 EAP)

Juhendaja: Marina Lepp, PhD

Tartu 2024

Kursuse “Objektorienteeritud programmeerimine” rühmatööde analüüs

Lühikokkuvõte:
Käesolevas bakalaureusetöös analüüsiti Tartu Ülikooli kursuse “Objektorienteeritud programmeerimine” osalejate koostatud rühmatöid ja nende kirjeldusi. Uurimistöö eesmärk oli leida rühmatööde tugevad ja nõrgad küljed ning pakkuda soovitusi kursuse korraldajatele õppematerjalide ja kursuse täiendamiseks. Rühmatööde ja nende kirjelduste analüüsist selgus, et üliõpilased koostasid enamasti mänge, üle poolte programmidest olid keskmisest keerukamad ning suur osa üliõpilastest jäid töö lõpptulemusega rahule. Peamised puudujäägid olid seotud graafilise kasutajaliidese, erindikäsitluse ning koodi kvaliteediga.

Võtmesõnad: objektorienteeritud programmeerimine, programmerimise rühmatöö, programmeerimise õpetamine
CERCS: P175 Informaatika, süsteemiteooria, S281 Arvuti õpiprogrammide kasutamise metoodika ja pedagoogika

Analysis of the Course “Object-Oriented Programming” Group Projects

Abstract:
This bachelor’s thesis analyzed the group projects and their descriptions created by the participants in the course “Object-Oriented Programming” at the University of Tartu. The research aimed to identify the strengths and weaknesses of the group projects and provide recommendations to the course organizers on improving the course and its materials. The analysis of the group projects and their descriptions revealed that the students primarily created games, over half of the programs were more complex than average, and most students were satisfied with their final results. The main weaknesses were related to the graphical user interface, exception handling, and code quality.

Keywords: object-oriented programming, programming group project, teaching programming
CERCS: P175 Informatics, systems theory, S281 Computer-assisted education

Sisukord

Sissejuhatus

Programmeerimine on protsess, mille käigus teisendatakse mõttes loodud ideed arvutile mõistetavaks ja teostatavaks vormiks [1]. Sellest on saanud 21. sajandi keskne oskus, mis on vajalik mitmesugustel erialadel ka väljaspool IT-sektorit [2]. Lisaks tehnilistele oskustele arendab programmeerimine ka probleemilahendusvõimet ja loovust [3]. Need mõlemad on olulised omadused, et tööturul konkurentsivõimeline püsida [2]. Programmeerimisoskuste omandamiseks kasutatakse sageli mitmesuguseid meetodeid, mis tihti omavahel kombineeritakse. Aktiivõpe, eriti koostööl põhinev, on osutunud üheks efektiivseks lähenemiseks [4-5]. Seda rakendatakse programmeerimise õppes näiteks rühmatööde kaudu, kus kaaslastega koostöös arendatakse tarkvara. Rühmatööd ei toeta aga üksnes tehniliste oskuste arengut, vaid võimaldavad arendada ka pehmeid oskusi nagu suhtlemine ja meeskonnatöö [6-8].

Tartu Ülikooli kursuse "Objektorienteeritud programmeerimine" raames tuleb üliõpilastel sooritada kaks rühmatööd, mille peamine eesmärk on kinnistada kursuse materjali. Kursusest osavõtjate arv on suur ja rühmadel on erinevad praktikumijuhendajad. Seetõttu puudub kursuse läbiviijatel hea ülevaade sellest, milliseid programme üliõpilased koostavad, kuidas nad saavad hakkama nõuete täitmisega ning milliseid puudujääke nende töödes esineb. Rühmatööd annavad hea ülevaate üliõpilaste omandatud praktilistest oskustest, mis on kursuse korraldajate jaoks oluline, et vajadusel korrigeerida kursuse materjale ja üldist korraldust. Seetõttu analüüsib autor aines tehtud rühmatöid ja nendega koos esitatud rühmatööde kirjeldusi, et tuvastada teemad, mis vajavad rohkem tähelepanu kursuse korralduses.

Selle bakalaureusetöö eesmärk on välja selgitada 2023. aasta kevadel toimunud kursuse "Objektorienteeritud programmeerimine" vältel tehtud rühmatööde peamised tugevused ja nõrgad kohad. Lisaks soovitakse teada, kuidas üliõpilased said hakkama rühmatöö nõuete täitmisega. Selle teabe põhjal soovib autor anda tagasisidet kursuse läbiviijatele rühmatööde kohta ning teha ettepanekuid kursuse materjalide täiustamiseks.

Eesmärgist lähtuvalt pani autor paika järgmised küsimused, millele uurimisel keskenduda:

1. Millistel teemadel rühmatöid koostati?
2. Milline oli üliõpilaste programmide ülesehitus ja keerukus?

3. Kuidas said üliõpilased hakkama rühmatöö programmi nõuete täitmisega?
4. Kuidas said üliõpilased hakkama rühmatöö kirjelduse nõuete täitmisega?
5. Millised olid rühmatööde peamised puudujäägid?

Uurimistöö on jaotatud kolmeks peatükiks. Töö algab ülevaatega kursusest "Objektorienteeritud programmeerimine", programmeerimise õpetamisest, rühmatöödest programmeerimise õppes, programmeerimise õppimisest ja programmerimisvigadest. Teine peatükk keskendub metoodika kirjeldamisele, tuues välja uurimuse konteksti, valimi ja protseduuri. Kolmandas peatükis analüüsitakse rühmatööde tulemusi ning antakse soovitusi edaspidiseks.

1. Ülevaade programmeerimise õpetamisest

Peatükk algab "Objektorienteeritud programmeerimise" kursuse üksikasjaliku tutvustusega, millele järgneb ülevaade erinevatest programmeerimise õpetamismeetoditest. Edasi uuritakse rühmatööde tähtsust ja edukate rühmatööde läbiviimise olulisi aspekte. Peatüki viimane osa keskendub programmeerimise õppimise protsessile ning programmeerimisvigadele.

1.1 Kursusest “Objektorienteeritud programmeerimine”

Objektorienteeritud programmeerimine (ainekoodiga LTAT.03.003) on 6 EAP-line aine, mis on eristava hindamisega. 6 EAP-d on võrdne 156 tunniga, millest antud kursusel moodustavad loengud ja praktikumid mõlemad 32 tundi ning iseseisev töö 92 tundi [9]. Kursuse “Objektorienteeritud programmeerimine” eesmärgiks on anda alusteadmised objekt-orienteeritud programmeerimise eripärast, oskused programmide koostamiseks ning esmased rühmatööoskused [10].

Kursuse õpiväljundid on Tartu Ülikooli õppeinfosüsteemi [9] kohaselt järgnevad:

- oskab selgitada objekt-orienteeritud paradigma põhimõisteid (kapseldus, abstraktsioon, pärimine, polümorfism, üledefineerimine, ülekate) ning analüüsida vastavaid programme;
- oskab kirjeldada erinevaid andmestruktuure (massiiv, ahel, magasin, järjekord, paisktabel) ja nende kasutusviise;
- oskab selgitada rakendusteekide väärtust ja olemust ning leida nendest vajalikku informatsiooni;
- oskab selgitada sündmuspõhise programmeerimise eripära ja erindite käitlemist ning tuua näiteid nende kasutamisest;
- oskab ühes objekt-orienteeritud programmeerimiskeeles kasutades integreeritud programmeerimiskeskkonda koostada, testida ja siluda programme, rakendades selleks eelmistes punktides loetletut;
- oskab kirjeldada isikliku kogemuse põhjal rühmaprojekti keskseid elemente.

Kursuse lõpphinne kujuneb loengute, praktikumide, kontrolltööde, rühmatööde, lisaülesannete ja eksami tulemusel [9]. Kuna üheks kursuse eesmärgiks on anda esmased rühmatöö oskused, on kursuse jooksul võimalik üliõpilastel sooritada kaks rühmatööd. Rühmatööde põhiline

ülesanne on kaaslasega koostöös kinnistada praktikumide materjali [11]. Rühmatööd toimuvad kaheliikmelistes tiimides, erandina võib rühmas olla ka kolm liiget. Mõlema rühmatöö eest on võimalik saada kuni viis punkti [10]. Rühmatöödel pole miinimum lävendit, aga praktikumide, kontrolltööde ja rühmatööde eest on vaja kokku saada vähemalt 28 punkti, et pääseda eksamile [9]. Rühmaliikmed saavad oma lahenduse eest võrdse hinde, kuid kui panus erineb märgatavalt võib hinne ka erineda [11]. Teine rühmatöö võib olla esimese edasiarendus või ka täiesti eraldiseisev projekt [10].

1.2 Programmeerimise õpetamisest

Programmeerimise õpetamise peamine eesmärk on arendada õppijates oskusi, mis võimaldaks neil luua tarkvara, et lahendada reaalseid probleeme [12]. Selle õpetamiseks on olemas mitmeid meetodeid, mille eesmärk on teha keerukad programmeerimise kontseptsioonid õppijate jaoks selgeks ning õppimisprotsess huvitavaks. Oluline on, et õpetajad valiksid õpetamismeetodid, mis vastaksid kõige paremini õppijate vajadustele ning motiveeriksid neid aktiivselt õppetöös osalema [13].

Programmeerimise õpetamises on kesksel kohal teooria teadmiste edasi andmine, mis traditsiooniliselt toimub loengute kaudu. Siiski jääb programmeerimise õppimises ainult teooriast väheks. Õppijatel on vajalik saada ka praktilist kogemust, et paremini mõista programmeerimise kontseptsioone [14]. Üks võimalus selleks on kombineerida teoreetilised loengud praktikumidega, mida rakendatakse ka kursusel “Objektorienteeritud programmeerimine”. Nii õpetajad kui õpilased on leidnud, et programmeerimist õpitakse rohkem praktikumides kui loengutes [14]. Praktikumides on võimalik õppijatel rakendada oma teoreetilisi oskusi praktikas ning seeläbi ka teooriat paremini kinnistada.

Digiajastu üliõpilastele, kes vajavad motiveerivamat ja kaasavamat õpikeskkonda, ei pruugi aga traditsioonilised õpetamismeetodid, mis keskenduvad peamiselt info edastamisele, olla piisavalt tõhusad [15]. Selle tulemusena on kasvanud mängupõhiste õpetamismeetodite ja visuaalsete õppevahendite populaarsus, mis aitavad õppijatel keskenduda ja säilitada huvi õppimise vastu [16-18]. Praktilisem õppeprotsess, mida näitlikustab loengute elavdamine põnevate lugude ja kohapeal lahendatavate praktiliste ülesannetega, on suurendanud üliõpilaste

kaasatust ja parandanud õpitulemusi [15]. Samuti on eakaaslaste õpetamine [4], paaris-programmeerimine [19-20] ja probleemipõhine õpe [21] osutunud efektiivseteks meetoditeks.

Paarisprogrammeerimise puhul on tegemist tarkvaraarenduse meetodiga, kus kaks programmeerijat töötavad ühiselt ühe arvuti taga – üks kirjutab koodi ning teine jälgib protsessi, et leida vigu. Programmeerijad vahetavad selle käigus regulaarselt rolle [19]. Paaris-programmeerimine võib vähendada koodi kirjutamisele kuluvat aega, tõsta koodi kvaliteeti ning parandada meeskonnatööd [20]. Samuti võib see parandada õppijate tulemusi ning suurendada nende rahulolu [19].

Probleemipõhine õpe keskendub probleemide analüüsimisele ja nende jagamisele loogilisteks sammudeks, mis aitab õppijatel paremini mõista programmeerimise põhimõtteid. Samuti soodustab probleemipõhine õpe õpilaste aktiivset osalemist ning julgustab neid probleemide lahendamisel loovalt ja strateegiliselt mõtlema [21].

1.3 Rühmatöödest programmeerimise õppes

Informaatika õppijate jaoks on rühmatööd sageli esmane kokkupuude tarkvara kollektiivse arendamisega. Need projektid pakuvad võimalust töötada suuremate ülesannete kallal ja aitavad arendada meeskonnatöö, kommunikatsiooni ja probleemide lahendamise oskusi [6-8]. Need oskused on tööturul, eriti just tarkvaraarenduses, kus meeskonnatöö on kesksel kohal, kõrgelt hinnatud [22-23]. Kohati on tarkvaraarendajad hinnanud pehmeid oskusi isegi olulisemaks kui tehnilisi [22]. Ometi just pehmetest oskustest nagu suhtlemine ja tiimitöö jääb aga informaatika lõpetanutel tihti vajaka [23-25]. See toob esile vajaduse keskenduda informaatika õppes nende oskuste arendamisele.

Kursusel “Objektorienteeritud programmeerimine” said üliõpilased ise moodustada rühmatöö grupid. See valikuvabadus võimaldas õppijatel valida grupiliikmeid, kelle tugevused ja nõrkused olid neile tuttavad, et moodustada sarnase motivatsiooni ja pühendumusega grupp. Kuigi tuttavatega ühes rühmas töötamine võib muuta ülesande lõbusamaks, võib see piirata koostööd erineva taustaga inimestega [26]. Teistpidi võib uute inimestega koos töötamine olla stressirohke, kuid aitab väljuda mugavustsoonist ja arendada oskusi mitmekesises meeskonnas töötamiseks [27]. Seega, kuigi sõpradega koos töötamine võib olla mugav, on üliõpilastel

tuleviku karjääri ja isikliku arengu seisukohast kasulik moodustada rühmi ka inimestega, kellega varem koostööd tehtud ei ole.

Rühmatööde edukus sõltub suuresti hoolikast eeltööst ja tõhusast meeskonnatöö korraldusest. Shamal Faily jt [27] täheldavad, et rühmatöid alustatakse tihti suure entusiasmiga, kuid see entusiasm võib projekti käigus järk-järgult kaduda. Seda peamiselt seetõttu, et projekti algusjärgus ei pöörata piisavalt tähelepanu vajaliku eeltöö tegemisele. Üks olulisemaid aspekte rühmatöö alguses on ülesannete efektiivne jaotamine, mis peaks arvestama iga grupiliikme varasemat kogemust ja eelistusi. See on hädavajalik, et vältida hilisemaid konflikte ja tagada projekti sujuv kulgemine [6, 27]. Liigne keskendumine juba tuttavatele ülesannetele võib aga viia varajase spetsialiseerumiseni, piirates sellega õppijate võimalusi arendada oskusi teistes valdkondades [6]. Seetõttu on oluline leida tasakaal õppurite eelistuste ja uute väljakutsete vahel, et tagada kõikide liikmete õppimine ja areng.

Lisaks tuleb arvestada, et edukad rühmatööd nõuavad pidevat koostööd ja omavahelist suhtlemist. Töö käigus omavahel suheldes on võimalik lahendada jooksvalt tekkinud probleeme ning hoida üksteist kursis töö käiguga. Samuti on olulised regulaarsed näost näkku kohtumised, et tagada rühmatöö edukus [27]. Need kohtumised pakuvad võimalust vaadata üle üksteise töö ning avastada vigu, mis võinuks muidu märkamata jääda.

1.4 Programmeerimise õppimisest

Programmeerimise õppimine on keeruline protsess, mis hõlmab uute teadmiste, strateegiate ja praktiliste oskuste omandamist. Benedict Du Boulay [28] järgi on tegemist mitmetahulise protsessiga, kus on oluline mõista nii programmeerimiskeele süntaksit ja semantikat kui ka arvuti töötamise põhimõtteid ja programmi eesmärke. Samuti on tähtsad praktilised oskused nagu planeerimine, arendamine, testimine ja koodi silumine.

Leon E. Winslow [29] märgib, et algajate programmeerijate peamised väljakutsed tulenevad nende piiratud teadmistest ning raskusest neid teadmisi praktikas rakendada. Tihti eelistavad nad kasutada üldiseid probleemilahendusstrateegiaid, mitte spetsiifilisi probleemile keskendunud lähenemisviise. Objektorienteeritud programmeerimise puhul lisanduvad veel täiendavad raskused, mis on seotud objektorienteerituse mõistmise ja rakendamisega [30].

1.4.1 Programmeerimisvigadest

Programmerimisvigu saab liigitada kolme kategooriasse: süntaksivead, täitmisaegsed vead ja loogikavead. Süntaksivead, mida kompilaator tuvastab programmi koostamisel, hõlmavad vigu koodi kirjutamises, näiteks kirjavead või puuduvad semikoolonid [31].

Täitmisaegsed vead jagunevad Javas kaheks: vead ning erindid. Vead tähendavad tavaliselt tõsiseid probleeme programmis, näiteks mälu otsa saamine. Erindid on aga palju levinumad kui vead ning tekivad loogikavigade või täitmata eelduste tagajärjel. Täitmisaegseid vigu on võimalik suuresti ennetada rakendades korrektset erinditöötlust, näiteks püüdes erind kinni ning lahendades veaolukord [32].

Loogikavead tekivad, kui programm ei käitu oodatud viisil, kuid need ei põhjusta kompileerimisel ega täitmisel vigu [31]. Näiteks on loogikaveaks mittetühja meetodi välja kutsumine tühja väljundiga avaldises [33]. Loogikavead on kõige raskemini tuvastatavad, sest on sageli seotud programmi sisemise loogikaga. Yong Daniel Liang [31] toob välja, et üks võimalus nende leidmiseks on väljastada muutujate väärtusi ja programmi seisundeid käsureale. Kui tegu on aga keerukamate programmidega, siis on mõistlik kasutada integreeritud arenduskeskkondade poolt pakutavat silurit (*debugger*). Silur võimaldab läbi töötada erinevaid programmi osasid ning jälgida koodirea haaval, kas kõik töötab korrektselt.

Programmeerimise õpetamisel on oluline tähelepanu pöörata ka koodi kvaliteedile. Koodi kvaliteediga seotud probleemid pole küll otseselt vead, kuid võivad pikas perspektiivis põhjustada tõsiseid probleeme. Halb koodi kvaliteet mõjutab tarkvarasüsteemide hooldatavust, jõudlust ja turvalisust [34]. Uuringus, kus analüüsiti üle miljoni JetBrains Academy platvormil esitatud Java programmi, olid kõige populaarsemad kvaliteediprobleemid kasutamata lokaalsed muutujad ja *import*-laused ning puuduvad *break*-käsud *switch*-lausetes [35]. Ehkki koodianalüüsi tööriistad pakuvad kasulikku tagasisidet koodi kvaliteedi kohta, ei pruugi algajad programmeerijad neid juhiseid alati järgida ning koodi kvaliteeti parandada. See toob esile vajaduse õpetada koodi kvaliteedi põhimõtteid juba programmeerimise õppe alguses [34].

2. Metoodika

Antud peatükis antakse ülevaade uurimuse kontekstist, käsitletavast rühmatööst, kasutatud valimist ja andmete kogumisest.

2.1 Kontekst

Käesolevas bakalaureusetöös on vaatluse all 2023. aasta kevadel toimunud kursuse "Objektorienteeritud programmeerimine" teise rühmatöö projektid. Üliõpilastel oli valikuvõimalus, kas arendada edasi oma esimese rühmatöö projekte või alustada uue iseseisva projektiga. Valdav enamus otsustas esimese rühmatöö programme täiendada. Teise rühmatöö raames lisandusid keerukamad tehnilised aspektid nagu graafiline kasutajaliides, failist lugemine ja kirjutamine ning erindite käsitlemine, mida esimese rühmatöö raames ei nõutud [10]. Arvestades, et teine rühmatöö esindab projektide lõplikku ja viimistletud versiooni, osutusidki just need analüüsi jaoks kõige sobivamaks.

Teise rühmatöö ülesanne anti 8. nädalal ning esitati kas 12. või 14. nädalal. Kui üliõpilased esitasid töö 12. nädalal, siis oli neil võimalus saada praktikumijuhendajalt tagasiside oma tööle, vastavalt sellele tööd parandada ning seejärel uuesti esitada. Kui töö esitati 14. nädalal, siis enam töösse parandusi sisse viia ei saanud [11].

Rühmatöö põhieesmärk oli koos rühmakaaslastega eelnevate praktikumide materjali kinnistamine. Rühmatööd tehti kaheliikmelistes rühmades, kus mõlemad liikmed pidid kuuluma samasse praktikumirühma. Projekti hinne oli rühmaliikmetel võrdne, välja arvatud juhtudel, kui liikmete panus märgatavalt erines [11].

Järgnevalt on esitatud teise rühmatöö programmi nõuded otse rühmatöö juhendist [11] muudatusteta:

- Programm käsitleb mingit (inimlikku) tegevust, näiteks mängimist, kliendile vastamist vm. Temaatika võib, aga ei pruugi, olla sama, mis 1. rühmatöös.
- Suhtlemine kasutajaga peab olema realiseeritud graafilise kasutajaliidese abil. Programm peab töötlema nii hiire kui ka klaviatuuriga tekitatud sündmusi.
- Programmi akna suurust muutes peab kuvatu mõistlikult muutuma.

- Erinditöötluse abil tagada, et toimuks mõistlik reageerimine (vähemalt mõnedele) kasutaja ekslikele tegevustele (nt. sisestustele).
- Programm peab mingid andmed kirjutama faili ja neid failist ka lugema. Näiteks võib tekitada logifaili ja selle põhjal korraldada käikude tagasivõtmise.
- Programm peab koosnema mitmest klassist (sh. peaklass).
- Programm peaks olema kasutatav ilma eriliste eelteadmisteta. Küsimused peavad vajaliku info andma. Hea oleks, kui programm käivitamisel annab vajaliku üldtutvustava lühiinfo.
- Programm peab olema mõistlikult kommenteeritud.
- Programm peab olema rühmaliikmete endi kirjutatud.

Lisaks programmile pidid üliõpilased esitama ka rühmatöö kirjelduse. Järgnevalt on esitatud teise rühmatöö kirjelduse nõuded otse rühmatöö juhendist [11] muudatusteta:

- autorite nimed;
- projekti põhjalik kirjeldus, kus on kirjas programmi eesmärk ja selgitus programmi üldisest tööst, vajadusel lühike kasutusjuhis;
- iga klassi kohta eraldi selle eesmärk ja olulisemad meetodid;
- projekti tegemise protsessi kirjeldus (erinevad etapid ja rühmaliikmete osalemine neis);
- iga rühmaliikme panus (sh tehtud klassid/meetodid) ja ajakulu (orienteeruvalt);
- tegemise mured (nt millistest teadmistest/oskustest tundsite projekti tegemisel puudust);
- hinnang oma töö lõpptulemusele (millega saite hästi hakkama ja mis vajab arendamist);
- selgitus ja/või näited, kuidas programmi osi eraldi ja programmi tervikuna testisite ehk kuidas veendusite, et programm töötab korrektselt.

Teise rühmatöö eest oli üliõpilastel võimalik teenida kuni viis punkti [10]. Üksi töötada otsustanud üliõpilased said maksimaalselt pool ettenähtud punktisummast ehk 2,5 punkti, sest rühmatöö eeldas kahekesi tegutsemist. Kui paariline loobus programmi tegemise käigus, siis üksi projekti tegema pidanud liiget ei karistatud punktide vähendamisega. Järgnevalt on esitatud rühmatöö punktide jaotus:

- Graafiline kasutajaliides. Töötleb (hiire ja klaviatuuri) sündmusi. Akna suurust muutes peab kuvatu mõistlikult muutuma. 1p

- Erinditöötlus. 1p
- Programm kirjutab faili ja loeb failist. 1p
- Programm koosneb mitmest klassist. Programm on mõistlikult kommenteeritud. 1p
- Kirjeldus. 1p

2.2 Valim

2023. aasta kevadel 24.-39. õppenädalal toimunud kursusest "Objektorienteeritud programmeerimine" võttis osa 325 üliõpilast [9], kellest 31,7% ehk 103 olid tüdrukud ning 68,3% ehk 222 olid poisid. Teises rühmatöös osales 236 üliõpilast, mis on 72,6% kursusel osalenutest. Kokku esitati 122 rühmatööd. Viis rühmatööd jäeti arvestusest välja, kuna nendes programmides puudusid vajalikud klassid ja seetõttu programmid ei käivitunud. Järelikult kvalifitseerus analüüsimiseks 117 rühmatööd, mille panid kokku 225 üliõpilast. Kusjuures neist 79 ehk 35,1% olid tüdrukud ja 146 ehk 64,9% olid poisid. Lisaks rühmatöödele analüüsiti ka rühmatööde kirjeldusi. Viis rühma jätsid kirjelduse esitamata ehk valimisse võeti analüüsimiseks 112 rühmatöö kirjeldust.

2.3 Protseduur

Autor analüüsis kõiki esitatud rühmatöid ja nende kirjeldusi, mille järel sisestas andmed Google Spreadsheet tabelisse edasiseks analüüsiks. Analüüsimise järel kanti tabel üle Microsoft Excelisse jooniste loomiseks. Töö käigus keskenduti rühmatööde läbivaatamisel järgmistele aspektidele:

1. rühmatöö teema (lahtiselt);
2. ridade arv (arvuliselt);
3. klasside arv (arvuliselt);
4. üliõpilaste poolt loodud erindite arv (arvuliselt);
5. programmi nõuete täitmine (jah/ei ja milliseid);
6. kommentaarid töös (üldse mitte, vähe, mõistlikult);
7. failist lugemise ja kirjutamise rakendamine (üldse mitte, failist lugemine, faili kirjutamine, mõlemad);
8. hiire, klaviatuuri ja nupu sündmuste esinemine (üldse mitte, hiire, klaviatuuri, nupu, hiire ja klaviatuuri, hiire ja nupu, klaviatuuri ja nupu, kõik);

9. erindite püüdmine (ei püüta, püütakse aga ei teha midagi, püütakse ja tehakse, kombinatsioon);
10. missuguseid erindeid püüti (lahtiselt);
11. püütud erindite käsitlemine (lahtiselt);
12. lisamoodulite ja raamistike kasutamine (jah/ei ja milliseid);
13. vigade olemasolu (jah/ei);
14. vigade liik (süntaksivead/loogikavead/täitmisaegsed vead);
15. puudujäägid koodi ja programmi kvaliteedis (lahtiselt);
16. rühmatöö keerukus (lihtne, keskmisest lihtsam, keskmisest keerukam, keeruline).

Rühmatöö kirjelduste põhjal analüüsis autor veel järgmisi punkte:

1. kirjelduse nõuete täitmine (jah/ei ja milliseid);
2. rühma suurus (üheliikmeline/kaheliikmeline/kolmeliikmeline);
3. ajakulu (arvuliselt tundides);
4. rühmatöö tegemise mured (lahtiselt);
5. kuidas koordineeriti rühmatöö tegevust (lahtiselt);
6. üliõpilaste hinnang oma lõpp-projektile (pole üldse rahul, mitte eriti rahul, rahul, väga rahul);
7. kuidas üliõpilased veendusid oma programmi korrektsuses (lahtiselt).

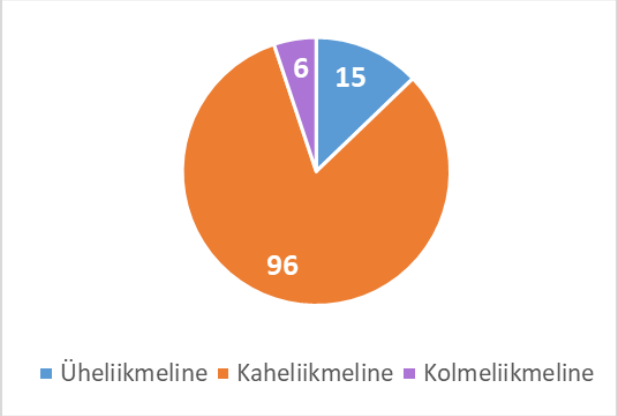
Ülaltoodud punkte hinnati kas kinniselt või lahtiselt. Kinniselt hinnatud punktide puhul on vastav skaala välja toodud punktide taga sulgudes. Lahtiselt analüüsitud punktide puhul pandi kirja märkmed ja tähelepanekud, mis hiljem grupeeriti kategooriateks, et hõlbustada edasist analüüsi ning jooniste loomist.

3. Tulemused ja arutelu

Antud peatükis esitatakse kogutud andmete põhjal saadud tulemused, tuginedes uurimisküsimustele. Lisaks tulemuste analüüsimisele pakutakse välja ka praktilisi soovitusi kursuse korraldajatele, et muuta rühmatööd tõhusamaks ja õppimiskogemus paremaks.

3.1 Rühma suuruse ja teema valik

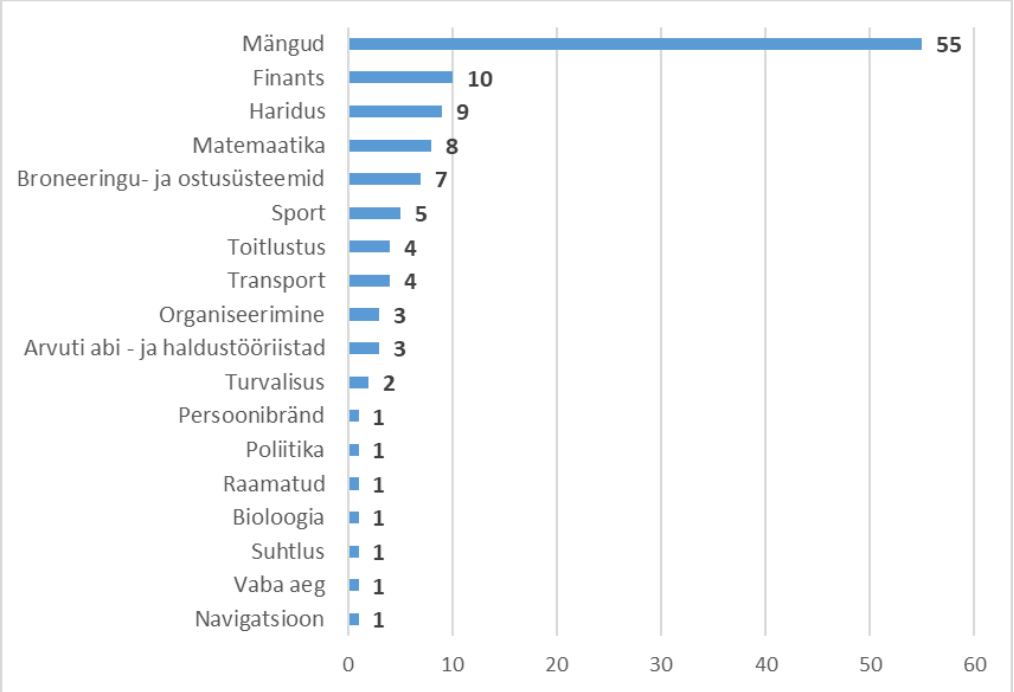
Vastavalt juhendile tuli rühmatöö sooritada kaheliikmelises rühmas [11]. Siiski leidus üliõpilasi, kes valisid projekti tegemiseks teistsuguse lähenemisviisi, tehes selle individuaalselt või kolmeliikmelises rühmas. Joonisel 1 on kujutatud rühmade jaotumine liikmete arvu poolest.



Joonis 1. Rühmade jaotumine liikmete arvu alusel

96 rühma ehk 82,1% olid kaheliikmelised, vastates rühmatöö juhendile. 15 üliõpilast sooritasid töö individuaalselt, kellest üks märkis, et põhjuseks oli raskus sobiva kaaslase leidmisel. Lisaks esines olukordi, kus rühmas oli nimeliselt kaks liiget, kuid üks neist ei andnud projekti sisulist panust ja sai rühmatöö eest null punkti. Sellisel juhul käsitles autor antud rühma üheliikmelisena. Kuus rühma olid kolmeliikmelised, ületades sellega ettenähtud kaheliikmelise rühma suuruse. Seetõttu eeldati neilt ka suurema mahuga projekte. Kuigi õppurid ei toonud otseselt kirjelduses välja, miks otsustati rühmatöö suuremas grupis teha, arvab autor, et sellel võis olla mitu põhjust. Näiteks ei pruukinud praktikumirühmas olla paarisarv osalejaid või sooviti rühmatöö koos sõpradega sooritada. Lisaks võisid mõned õppurid otsustada suurema rühma kasuks, et töökoormust paremini jaotada.

Üliõpilastele anti rühmatöö teema valikul vabad käed, tingimusel, et see keskendub mingile inimlikule tegevusele, näiteks nagu mängimine või kliendiga suhtlus [11]. Esimese rühmatöö juhendis soovitati aga valida teemaks midagi muud peale mängimise, näiteks erialaste andmete töötlus [36]. Autor jaotas õppurite tööd nende sisu põhjal 18 erinevasse kategooriasse, et saada parem ülevaade valitud teemadest. Joonis 2 näitab, et vaatamata antud soovitusele oli mängude loomine siiski kõige populaarsem, moodustades 47% kõikidest valitud teemadest.



Joonis 2. Rühmatööde teemad

Rühmatööde seas leidus üldtuntud mänge nagu Blackjack, trips-traps-trull, pokker, sudoku, male, kabe, rulett, kuldvillak, laevade pommitamine ja Yatzy. Samuti ka originaalseid mänge, sealhulgas detektiivimäng, suusahüppemäng, joonistamismäng ja pahalastega võitlemise mängud. Praktikumijuhendajad jälgisid tähelepanelikult, et esitatud mängud oleksid originaalsed, mitte internetist kopeeritud, kuna populaarsed mängud on internetist kergesti kättesaadavad.

Mängude loomine ja nende mängimine on õppijate jaoks lõbus ning kaasahaarav protsess. Uuringud on näidanud, et mänguteemaliste ülesannete kaasamine programmeerimise õppesse suurendab õppijate motivatsiooni ja huvi programmeerimise suhtes [15-16, 18] ning aitab parandada ka nende tulemusi [15, 17-18]. Seetõttu leiab autor, et õppijad võiksid ka edaspidi

mänge rühmatöö raames luua. Samas on arusaadav, et kursuse korraldajad ei soovi, et üliõpilased looksid mänge, mis on internetist kergesti kättesaadavad. Hetkel ei käsitle aga teise rühmatöö juhend selliste mängude sobimatust üldse ning esimene mainib seda põgusalt. See võib olla üks põhjus, miks mitmeid populaarseid mänge loodi. Seega leiab autor, et mõlema rühmatöö juhendis tuleb konkreetsemalt välja tuua, et loodud mängud peavad olema originaalsed. Lisaks peaksid esimese rühmatöö juhendis välja toodud näited olema loovamad ning ei tohiks sisaldada laialdaselt levinud mänge.

Populaarsuselt teine teema oli „finants“, mille alla liigitus 10 rühmatööd. Antud kategooria hõlmas rakendusi, mis simuleerisid pangatoiminguid, pakkusid investeerimisalast teavet ja aitasid finantse planeerida. Kuigi üliõpilased ei toonud konkreetselt välja, miks nad just antud teema valisid, on ilmne, et sellel oli praktiline väärtus. Näiteks said õppijad oma programme kasutada eelarve jälgimiseks või investeeringute tasuvuse hindamiseks. Haridus oli keskne teema üheksas rühmatöös ning matemaatika kaheksas. „Hariduse“ alla liigitusid näiteks programmid, mis hõlbustasid eriala valimist ja aitasid õppimist planeerida. „Matemaatika“ puhul tundus autorile üldiselt, et eesmärk oli luua rakendusi, mis lihtsustaksid matemaatika-õpet. Näiteks loodi programme tõenäosusteooria jaotuste arvutamiseks ja graafikute koostamiseks. Veel võib välja tuua „arvuti abi- ja haldustööriistad“, mis hõlmas praktilisi rakendusi arvuti taustapildi vahetamiseks, YouTube’ist muusika allalaadimiseks ja failide organiseerimiseks.

„Broneeringu- ja ostusüsteemide“ kategooria alla jaotusid seitse rühmatööd. Siia liigitusid programmid nagu kino ja lennupiletite ostusüsteemid, kassasüsteem, hotelli broneerimis-süsteem, raamatupoe tellimissüsteem ja baari krediidisüsteem. Erilist mainimist väärib baari krediidisüsteem, mille tegemise vajadus tulenes õppijatel elust enesest. Üliõpilased testisid seda programmi ka edukalt üritusel, kus said tagasisidet süsteemi toimimise ning probleemide kohta. Saadud tagasiside põhjal said õppijad oma programmi täiustada ning vigu parandada.

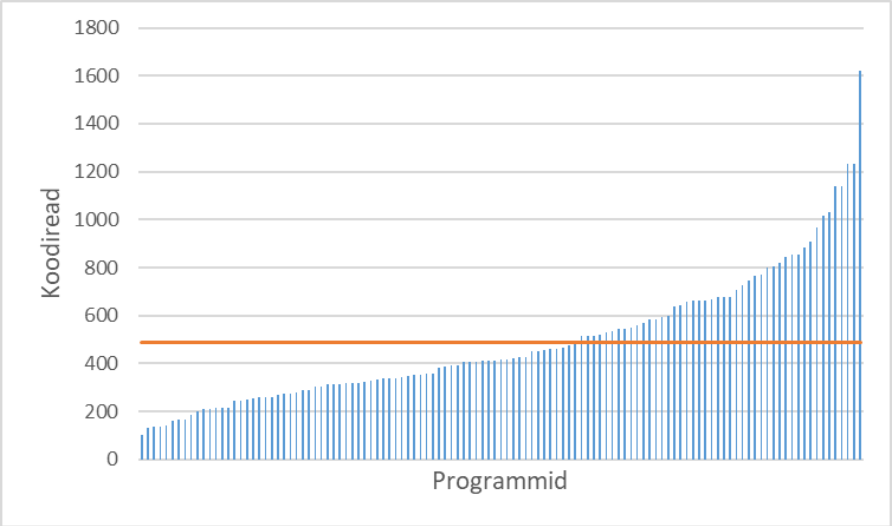
Praktiliste programmide loomine on programmeerimise õppes väga kasulik, sest võimaldab üliõpilastel lahendada konkreetseid probleeme erinevates eluvaldkondades. Kuna üliõpilased on ise antud programmide puhul lõppkasutajad, on ka testimisprotsess neile arusaadavam, sest teavad täpselt, mida programm tegema peaks. Selliste reaaleluliste probleemide lahendamine on tähtis, et säilitada õppijate huvi ja motivatsioon programmeerimise vastu [37]. Samuti võib

see aidata neil paremini mõista objektorienteerituse põhimõtteid [30]. Seetõttu leiab autor, et ka rühmatöö ülesande näidetesse võiks lisada rohkem praktilisi rakendusi, et julgustada õppijaid neid rohkem tegema.

Kuigi üliõpilased üldiselt oma teemavalikut ei põhjendanud, paistis et valik tehti isiklike huvide ja praktilise vajaduse põhjal. Autori hinnangul tehti rühmatöid eesmärgiga lihtsustada igapäevaelu, panna end proovile ning nautida programmi loomise protsessi ja selle lõpptulemust.

3.2 Programmide struktuur

Programmide struktuuri analüüsimisel keskenduti lähtekoodi ridade arvule, üliõpilaste poolt loodud klassidele ning kasutatud lisamoodulitele ja -raamistikele. Joonis 3 illustreerib programmide lähtekoodi ridade arvu, kus iga sinine joon tähistab ühte programmi.

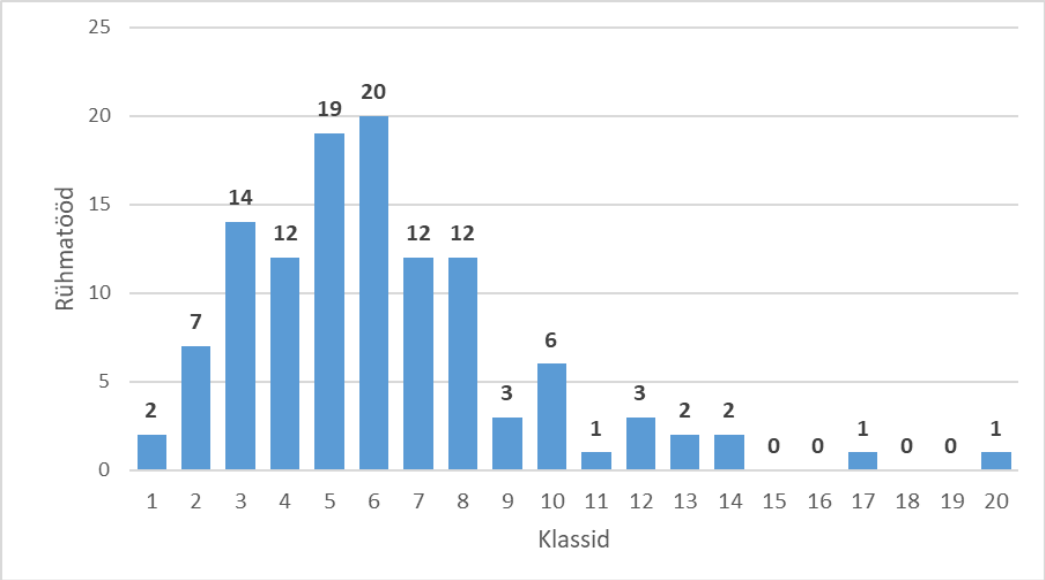


Joonis 3. Koodiridade arv programmides

Programmide keskmine ridade arv oli 489,3, mida illustreerib joonisel oranž joon. Mediaani väärtus oli 417, mis olles madalam kui keskmine koodiridade arv, viitab ridade arvu jaotuse kalduvusele väiksemate väärtuste poole. Standardhälve 271,4 ning koodiridade minimaalse arvu 104 ja maksimaalse 1619 suur erinevus kinnitavad märkimisväärseid varieeruvusi üliõpilaste programmide mahus. Arvestades, et antud rühmatöö nõuete piires oli võimalik luua väga erineva mahuga töid, oli selline varieeruvus ootuspärane. Samuti tuleb arvesse võtta, et

üliõpilaste varasem programmeerimiskogemus ning teadmised olid erineval tasemel. Seetõttu võisid mõned rühmad saavutada nõutud tulemuse väiksema koodiridade arvuga, samas kui teised kasutasid selleks rohkem koodi.

Joonisel 4 on esitatud üliõpilaste poolt loodud klasside arv. Rühmatöid analüüsides tuvastati, et enamik programme sisaldasid *module-info.java* faili. Kuna tegu on mooduli seadistuse määratlemiseks mõeldud failiga ning see ei olnud üliõpilaste endi loodud, siis antud kontekstis seda klasside arvu hulka ei arvestatud.



Joonis 4. Klasside arv rühmatöödes

Vastavalt rühmatöö nõuetele pidid üliõpilaste programmid koosnema vähemalt kahest klassist. Siiski leidus kaks tööd, mis ei vastanud antud tingimusele, sest need piirdusid vaid ühe klassiga. Maksimaalseks klasside arvuks osutus 20, mis esines ainult ühes töös. Keskmine klasside arv rühmatöö kohta oli 6,2, mis koos standardhälbe 3,2 ja mediaaniga 6 näitab, et enamik rühmi jäi mõõduka klasside arvu piiridesse, arvestades nende tööde mahtu. Samas oli klasside arv piisav, et tõhusalt rakendada objektorienteeritud disaini.

Lisamoodulite kasutamine ei olnud rühmatöödes kohustuslik, kuid autor tahtis siiski teada, kas ja milliseid üliõpilased rakendasid. Kõige populaarsemaks osutus JavaFX, mis on kõige uuem Java graafilise kasutajaliidese raamistik [31]. Seda kasutas oma programmides 102 rühma. Ülejäänud üliõpilased eelistasid graafilise kasutajaliidese loomiseks peamiselt Java standard-

teegi raamistikke AWT ja Swing. Peale JavaFX'i leidsid vaid viie rühmatöö puhul kasutust lisamoodulid. Nendeks olid: Next.js, Spring Boot, PostgreSQL, Gson, JNA, Log4j2, CommonMark ja MaterialFX. Arvestades Java standardteegi ning JavaFX'i pakutavat laialdast funktsionaalsust, pole üllatav, et lisamoodulite kasutamine polnud laialdasem. Java standardteeki kasutati aga kõikides programmides, eriti populaarseks osutusid *java.io* ja *java.util* paketid, mida kasutasid vastavalt 112 ja 109 rühma. *Java.io* [38] pakub süsteemile sisendit ja väljundit andmevoogude ja failisüsteemide kaudu ning *java.util* [39] sisaldab kogumike raamistikku, sündmuste mudelit ja kuupäeva ning kellaaja funktsioone.

3.3 Rühmatöö kirjelduse nõuete täitmine

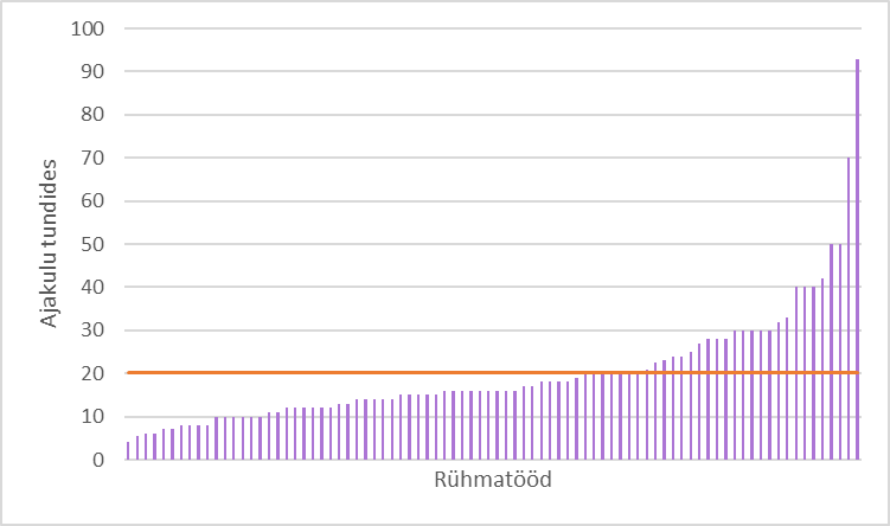
Programmi kirjelduse nõuded on esitatud "Metoodika" peatükis ning töö autor kontrollis nende täitmist. Joonisel 5 on välja toodud, kuidas üliõpilased said hakkama rühmatöö kirjelduse nõuete täitmisega ning millised nõuded jäid täitmata. 64 projektis, mis moodustab 57,1% kõikidest esitatud kirjeldustest, täideti kõiki kirjelduse nõudeid ning 48 töös jäi mõni nõue täitmata.

Joonis 5. Kirjelduse nõuete täitmine ja täitmata jäänud nõuded

26 rühma jätsid mainimata orienteeruva aja, mis kulus rühmatöö tegemisele. Üks rühm mainis ajakulu vaid koos esimese rühmatööga, kaks rühma ei osanud ajakulu hinnata, ühes töös oli kirjas vaid ühe etapi ajakulu ning teises toodi välja üksnes ühe liikme panus. Programmi korrektsuse tagamine jäi kirjeldamata 11 rühmatöös, mis võib osutada sellele, et üliõpilased kas ei testinud programmi põhjalikult või lihtsalt ei osanud seda protsessi selgitada. Joonis 5 toob esile ka teised täitmata jäänud nõuded. Nende põhjuseks võis lisaks oskamatusele olla ka

tähelepanematus või motivatsiooni vähesus. Eriti kui võtta arvesse, et tööprotsessi ja klasside kirjeldamine on ajakulukas protsess. Järgnevalt analüüsitakse antud nõudeid põhjalikumalt.

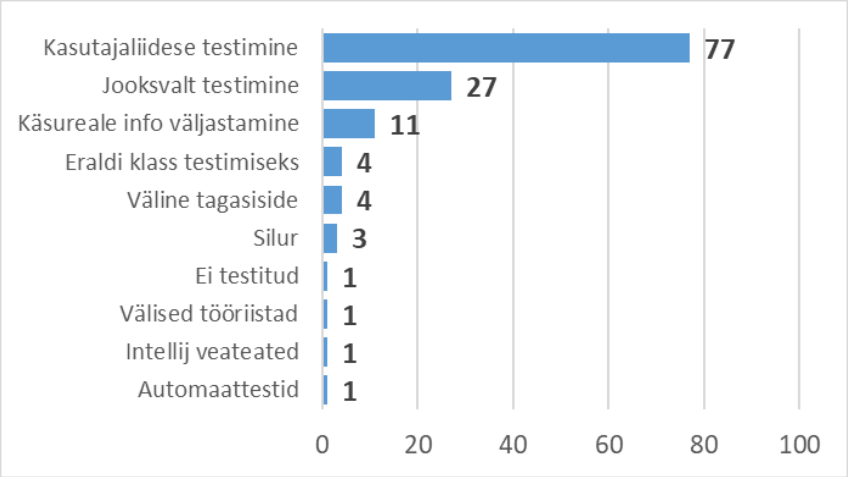
Joonis 6 illustreerib rühmatööde ajakulu jaotumist. Analüüsi kohaselt tõi 86 rühma kirjelduses välja projektile kulunud aja. Siiski kaks tööd ei täpsustanud ajakulu tundides, vaid viitasid ajaperioodidele nagu “3 päeva” ja “nädal”, mis eeldatavasti ei tähenda pidevat rühmatööga tegelemist. Seetõttu jäeti need tööd analüüsist välja.



Joonis 6. Ajakulu jaotumine tundide lõikes

Andmete analüüs näitas, et keskmine ajakulu oli 20,1 tundi rühma ning 10,3 tundi ühe liikme kohta. Mediaan oli vastavalt 16 ja 8 tundi. Rühma ajakulu varieerus 4 tunnist 93 tunnini, mis koos standardhälbe 13,86-ga viitas suurele kõikumisele. Sellise varieeruvuse võisid põhjustada mitmed tegurid, sealhulgas rühmade erinev suurus ning efektiivsus. Kuigi võiks eeldada, et suurem ajakulu tulenes mahukamast programmist ning rohketest funktsionaalsustest, siis ometi ei olnud see alati nii. Näiteks 93 tundi projekti teinud rühm tõi välja, et suur osa ajast kulus neil esimese rühmatöö graafiliseks muutmisele. Samuti tuleb arvesse võtta, et üliõpilased võisid oma ajakulu ka valesti hinnata – nii väiksemaks kui suuremaks, kui see tegelikult oli. Autor soovitab kursuse korraldajatel tutvustada üliõpilastele tööriistu, mis võimaldavad mõõta programmeerimisele kuluvat aega. See võib anda üliõpilastele täpsema ülevaate, milline oli nii nende kui paarilis(t)e tegelik ajakulu. Kuna selliseid tööriistu kasutatakse sageli tarkvaraarenduses, siis pakub see üliõpilastele hea võimaluse tutvuda nende vahenditega juba varakult.

Järgnevalt uuriti, kuidas üliõpilased veendusid oma programmi korrektselt töötamises, mille ülevaate annab joonis 7. Sageli kasutasid üliõpilased mitut testimismeetodit korraga, näiteks kontrolliti graafilise kasutajaliidese funktsionaalsusi ning jälgiti programmi seisundeid käsureal.



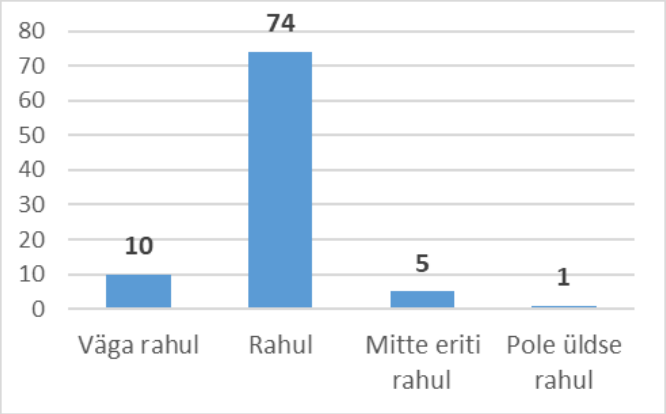
Joonis 7. Kuidas üliõpilased veendusid programmi korrektsuses

77 rühma veendusid oma programmi korrektsuses läbi graafilise kasutajaliidese testimise. Selleks käivitati programm ning asetati end kasutaja rolli, et katsetada läbi erinevaid toiminguid, mida kasutaja võib sooritada. See lähenemine võimaldas üliõpilastel kontrollida, kas programm töötab ettenähtud viisil. Sisestades kasutajaliidesesse tahtlikult valesid sisendeid, sai testida ka veateadete ja erinditöötluse korrektset tööd. Lisaks kasutati kasutajaliidest, et testida selle elementide mastabeeritavust (*scalability*) ning nuppude funktsionaalsusi. Arvestades, et programmid pidid sisaldama graafilist kasutajaliidest, oli selline testimisviis mõistlik. See võimaldas avastada vigu nii kasutajaliideses endas kui ka programmi taga olevas loogikas. Oluline on märkida, et antud testimisviisi juures on põhjalikkus eriti oluline, läbi tuleb testida erinevaid kasutaja poolt tehtavaid tegevusi. Üliõpilaste programmide analüüs näitas, et tihtilugu jäid mingid situatsioonid testimata, mis võis põhjustada vigu. Näiteks olukorrad, kus kasutajasisendile polnud piisavat kontrolli ning programm ei suutnud ebasobiva sisendi puhul tööd jätkata.

Lisaks tõid 27 rühma välja jooksvalt testimise, mis tähendab koodi pidevat testimist arenduse käigus. Selline lähenemine hõlmas näiteks iga uue meetodi katsetamist kohe pärast selle lisamist, veendumaks et see töötab korrektselt ja ei tekita probleeme teistes programmi osades.

Kuigi kõik rühmad ei maininud seda meetodit otseselt, on tõenäoline, et ka nemad rakendasid jooksvalt testimist mingil määral. See testimismeetod on vigade leidmiseks efektiivne, sest uute funktsionaalsuste põhjustatud probleemid on kiiremini ja kergemini tuvastatavad. 11 rühma kasutasid testimiseks käsurida, et väljastada teavet programmi seisundi kohta. Näiteks sai nii kontrollida, et muutuja väärtused vastaksid ootustele. Neljas rühmas kaasati testimisprotsessi ka sõpru ja tuttavaid, kelle tagasiside põhjal tehti programmis parandusi.

Üliõpilastele ei antud tööle lõpphinnangu andmise osas muud juhist kui hinnata lõpptulemust ning tuua esile millega saadi hästi hakkama ja mis vajaks täiendamist. Õppijate vastuste põhjal jaotas autor vastused järgmistesse kategooriatesse: “väga rahul”, “rahul”, “mitte eriti rahul” ja “pole üldse rahul”. 12 rühma ei väljendanud konkreetselt rahulolu või rahulolematust, vaid keskendusid rohkem sellele, mis rühmatöös hästi õnnestus, milliseid aspekte oleksid saanud paremini teha ja milliseid edasiarendusi võiksid tulevikus programmile veel teha. Üliõpilaste hinnang lõpptulemusele on esitatud joonisel 8.

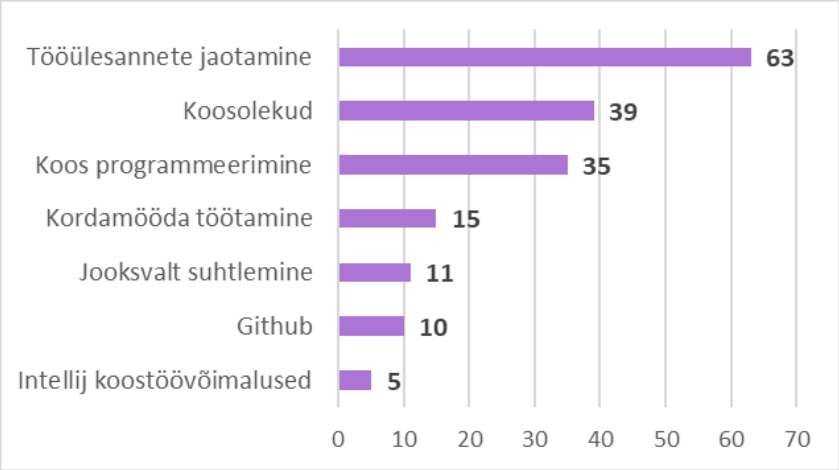


Joonis 8. Üliõpilaste hinnang oma töö lõpptulemusele

Andmetest selgub, et 75% kirjelduse esitanud rühmadest jäid oma lõpptulemusega rahule või väga rahule, mis on positiivne märk. Isegi rühmad, kes ei suutnud kõiki planeeritud eesmärke täita, olid enamasti lõpptulemusega rahul. Mitmes kirjelduses toodi välja, et kuna varasem kokkupuude graafilise kasutajaliidesega puudus, tunti uhkust, et suudeti sellega toime tulla. Viis rühma, kes ei jäänud tööga eriti rahule tõid kõik välja ajapuuduse ja/või aja halvasti planeerimise. Seetõttu ei jõutud teha selline programm nagu algselt oli plaanis, ei saadud tööle kõiki soovitud funktsionaalsusi ning ei jõutud koodi korrastada. Rühm, kes ei jäänud lõpptulemusega üldse rahule, tõi välja, et projekt osutus nende jaoks liiga keerukaks. Seda ei

osatud aga töö algul ette näha, sest see selgus alles rühmatöö tegemise käigus. Seetõttu on oluline, et juba töö algul seatakse paika selge plaan ja ajakava, mida saab jooksvalt jälgida ning vajadusel korrigeerida. Plaani koostamisel tuleb arvesse võtta ka liikmete varasemat kogemust, eelistusi ja logistikat, mis soodustavad plaani praktikas kasutusele võtmist [27].

Rühmatöö kirjelduses pidid üliõpilased lahti seletama oma tööprotsessi, tooma välja erinevad etapid ning rühmaliikmete osalemise nendes [11]. Selle põhjal analüüsis autor, kuidas toimus rühmatöö koordineerimine, mida illustreerib joonis 9. 93 projektis hõlmas tööprotsess individuaalset programmeerimist. Neist 15-s oli tegemist üheliikmeliste rühmadega, kus töö koosnes üksnes individuaalsest panusest. 63 rühma jagasid omavahel ära tööülesanded ning seejärel programmeeriti iseseisvalt. Ülejäänud 15 rühma ei maininud kirjelduses otseselt ülesannete jaotamist, vaid kirjeldasid hoopis kordamööda töötamist. Nimelt alustas üks liige programmeerimist ja tegi mingi osa, seejärel alustas tööd teine, kes vastavalt arendas, parandas või täiendas koodi. Seejärel läks programm vajadusel järjekordselt esimese liikme kätte.



Joonis 9. Rühmatöö tegevuse koordineerimine

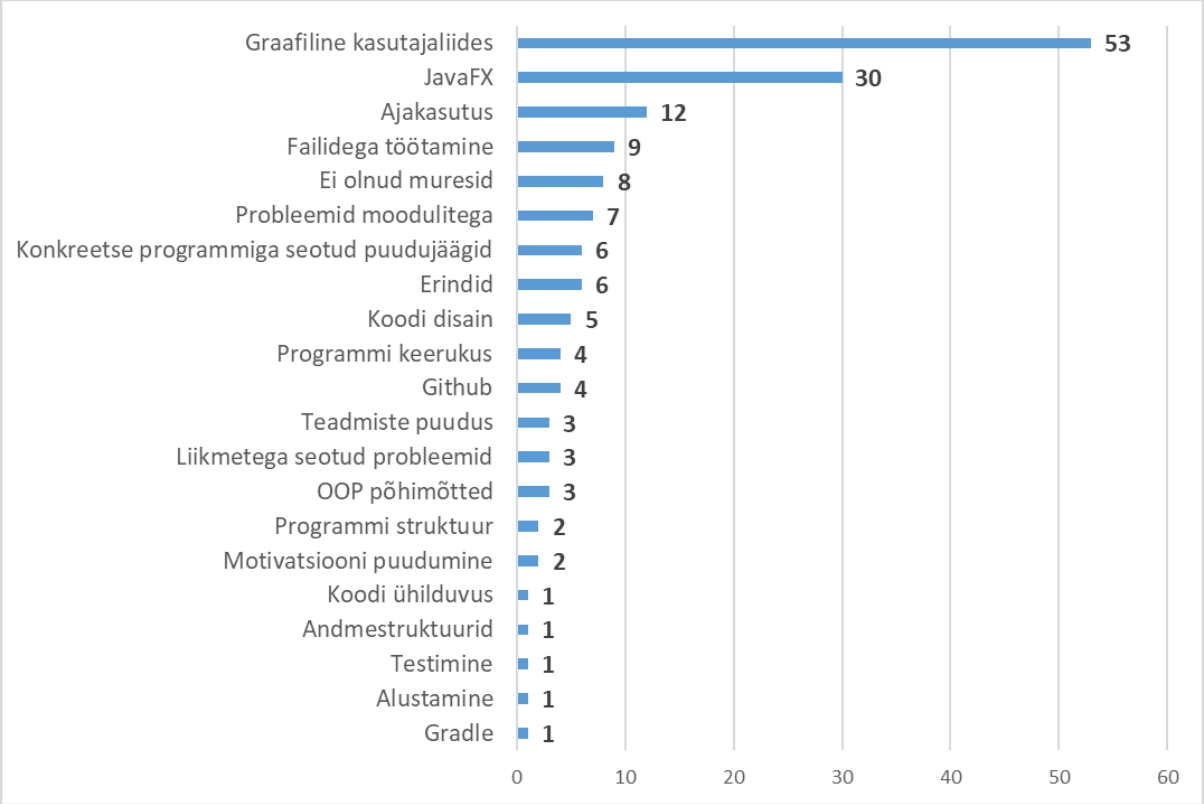
Koosolekute alla liigitas autor kohtumised, kus üliõpilased jagasid ideid ja panid paika üldise rühmatöö plaani. Samuti hõlmas see kategooria rühmatöö vältel tehtud kohtumisi, kus vaadati üle üksteise töö, jagati probleeme ja/või tehti üheskoos rühmatöö kirjeldus. Regulaarsed kohtumised on olulised, sest annavad liikmetele võimaluse jagada informatsiooni, teha ajurünnakuid, lahendada probleeme ning hoida üksteise tööl silma peal [27].

Koos programmeerimise alla liigitusid olukorrad, kus õppurid programmeerisid füüsiliselt koos olles, kas erinevatest arvutitest ja samal ajal arutledes või kasutades paarisprogrammeerimise tehnikat ühise arvuti taga. Paarisprogrammeerimise tehnikat on õppijatel kasulik rakendada, sest lisaks kõrgema kvaliteediga koodile soodustab see ka tõhusamat õppimist [20]. Seetõttu võivad praktikumijuhendajad julgustada üliõpilasi seda tehnikat isegi laialdasemalt kasutama.

Esimese rühmatöö juhendis soovitati kasutada Githubi või mingit muud Git versiooni-haldustarkvara [36], kuid kõigest 10 rühma tõid kirjelduses välja selle kasutamise. Githubi kasutamine oleks olnud aga üliõpilastele väga kasulik, sest rühma liikmed oleksid saanud samaaegselt programmeerida ning ükski liige poleks pidanud teise järel ootama. Samuti oleks Githubi kasutamine aidanud praktikumijuhendajatel jälgida, et rühmaliikmete panused oleksid võrdsed. Töö autor soovitab praktikumijuhendajatel enne rühmatööga alustamist tutvustada lühidalt GitHubi kasutamist ja rääkida selle eelistest, et õppijad oleksid rohkem motiveeritud seda praktikas rakendama.

Analüüsist ilmnes, et enamasti ei piirdunud aga õppurid ainult ühe meetodiga töö koordineerimiseks, vaid kombineerisid neid. Näiteks pandi algselt üheskoos paika rühmatöö idee ja tööjaotus, seejärel programmeeriti iseseisvalt ning hoiti üksteist jooksvalt kursis. Samuti tehti vajaduse korral koos mingi keerukam programmi osa või töö viimistlus. Kuigi ainult 11 rühma mainisid jooksvalt läbi interneti suhtlemist sõnumite või video vahendusel, usub autor, et see number oli tegelikkuses palju suurem. Tõenäoliselt eeldasid üliõpilased, et pidev suhtlus rühmatöö vältel on iseenesestmõistetav ning ei hakanud seda tööprotsessi kirjeldades eraldi mainima.

Üliõpilased tõid esile mitmeid muresid, mis neil programmi koostamisel tekkisid, mille autor liigitas 21 kategooriasse. Oluline on märkida, et enamik rühmi tõid esile rohkem kui ühe puudujäägi. Joonis 10 pakub täpset ülevaadet oskustest ja teadmistest, millest üliõpilased rühmatöö loomisel puudust tundsid.



Joonis 10. Rühmatöö tegemise mured

Eriliseks kitsaskohaks osutus graafilise kasutajaliidese arendamine, mis oli üks rühmatöö oluline nõue [11]. Lausa 53 ehk 47,3% kirjelduse esitanud rühma tõid välja sellega seotud väljakutsed. Üliõpilased tundsid, et kursusel ei õpitud graafilise kasutajaliidese kohta piisavalt, mistõttu osutus selle ülesehitamine paljudele probleemseks. Õppurid rõhutasid muresid, mis puudutasid kasutajaliidese elementide paigutamist, akna suuruse paindlikku kohandamist, nuppude funktsionaalsust ning tõhusat suhtlust eri akende vahel. 30 rühma tõid esile konkreetsemalt JavaFX'iga seotud probleemid, kuna antud raamistikku kasutati peamiselt graafilise kasutajaliidese loomiseks. Oma olemasolevate projektide ülekandmine JavaFX'i osutus väljakutseks, sundides üliõpilasi programme oluliselt muutma või lausa nullist üles ehitama. Toodi välja, et kuna varasem kogemus JavaFX'iga oli minimaalne, põhjustas segadust selle koodi klassidesse jaotamine ning veateadete mõistmine. Seetõttu oleksid üliõpilased soovinud omada paremat ülevaadet JavaFX'i üldstruktuurist ja selle rakendamise põhimõtetest.

Tulemuste põhjal leiab autor, et arvestades graafilise kasutajaliidese keerukust, tuleks sellele kursuse sisus rohkem tähelepanu pöörata. Eriti oluline on välja tuua, kuidas elemente paigutada nii, et need reageeriksid mõistlikult akna suuruse muutusele ja kuidas akende vahelisi

üleminekuid teha sujuvamaks ja loogilisemaks. Lisaks kui võtta arvesse, et paljud üliõpilased arendavad edasi oma esimese rühmatöö programme, on vajalik anda täpsemaid juhiseid, kuidas neid JavaFX'iga integreerida. Samuti on kasulik juba esimese rühmatöö algul selgitada, et kui üliõpilased soovivad sama projektiga jätkata teises rühmatöös, peavad nad läbi mõtlema, kuidas on võimalik programmi graafilisele kujule viia. See on eriti tähtis, kuna mitmed õppijad olid sunnitud oma programme oluliselt muutma, et neid graafiliseks teha.

12 rühma tõid välja probleemid ajakasutusega, märkides, et ei suutnud aega efektiivselt planeerida, mis viis ajapuuduseni. Lisaks mainisid neli rühma programmi liigset keerukust, mis põhjustas raskusi. Autor soovitab nende probleemide lahendamiseks motiveerida üliõpilasi esitama töid varasemal tähtajal. See innustaks neid varem alustama ning aitaks õigeaegselt tuvastada, kui töö võib osutuda liigselt keerukaks.

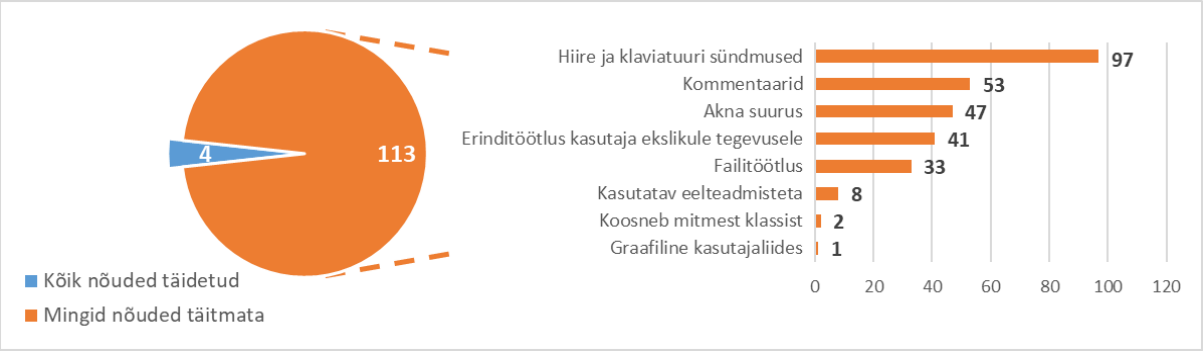
Autor soovib esile tuua ka motivatsioonipuuduse, kuigi seda mainisid otseselt ainult kaks rühma. Üks neist aga selgitas, et teise rühmatöö eest antav punktisumma on koguarvestuses väike, võrreldes tehtava töö mahuga, mis ei motiveerinud neid ka pingutama. See võib olla üks põhjus, miks 27,4% kursusel osalenutest ei sooritanud teist rühmatööd. Teine rühmatöö hõlmab aga mitmeid aspekte, näiteks nagu graafiline kasutajaliides ning failitöötlus, mida esimene ei käsitle. Samuti on rühmatööd programmeerimise õppes olulised, sest arendavad nii kommunikatsiooni kui ka meeskonnatööoskusi, millest jääb informaatika lõpetanutel tihtilugu vajaka [23-25]. Seetõttu leiab autor, et kursuse korraldajatel tuleb kaaluda rühmatöö eest saadava punktisumma suurendamist.

3.4 Rühmatöö programmi nõuete täitmine

Järgnevalt tahtis autor teada, kuidas üliõpilased said hakkama programmi nõuete täitmisega. Need nõuded on detailsemalt välja toodud "Metoodika" peatükis. Jooniselt 11 on näha, et 113 rühmatöös ehk 96,6% programmides jäid mingid nõuded täitmata ning kõigest neljas olid kõik täidetud. Joonis 11 illustreerib detailsemalt, millised nõuded täitmata jäid.

Üliõpilaste programmides osutusid kõige suuremaks puudujäägiks hiire ja klaviatuuri sündmused. Programmidele esitatud nõuete kohaselt pidid üliõpilased kasutama mõlemat sündmust, kuid 97 töös kasutati kas ainult ühte neist või ei kasutatud kumbagi. Kommentaare pidid rühmatööd sisaldama "mõistlikult", mis andis õppuritele laia tõlgendamisvõimaluse.

Autori hinnangul ei olnud 53 tööd piisavalt kommenteeritud – kaheksas programmis puudusid kommentaarid täielikult ja 45 programmis oli kommentaare kas vähe või need ei olnud asjalikud. 47 rühmatöös esines probleeme akna suuruse muutmisega, see põhjustas elementide nihkumise või kasutajaliidese aknast väljajäämise. Lisaks liigitusid siia programmid, kus akna suurust ei olnud võimalik muuta.



Joonis 11. Programmi nõuete täitmine ja täitmata jäänud nõuded

41 rühmatöös ei rakendatud erinditöötlust viisil, mis oleks pakkunud kasutajale asjakohast tagasisidet vähemalt mõnele tema ekslikule tegevusele. Ehkki 76 tööd vastasid antud nõudele, leidus neist 26-s mingeid kasutajasisestusi, mis kas jäid täiesti kontrollimata või ei andnud kasutajale täpseid veateateid. Seega leidis autor, et 67 programmis oleks pidanud sisendikontroll olema põhjalikum. Tundus, et üliõpilased ei mõelnud läbi kõiki variante, mida kasutaja võis sisestada ning seetõttu jäi kasutajasisendi kontrollimine paljudes töödes puudulikuks. Seda on aga väga oluline rakendada, sest ebapiisav kontroll võib põhjustada programmis nii loogikavigu kui ka täitmisaegseid vigu.

Kaheksa rühmatöö puhul leidis autor, et need ei olnud kasutatavad eelteadmisteta, sest programm ei pakkunud piisavalt juhendeid selle kasutamiseks. Näiteks eeldas programm, et kasutaja sisestab andmed teatud viisil, kuid ei pakkunud mingit informatsiooni, mida täpsemalt ootab.

Järgnevalt on põhjalikumalt analüüsitud programmi nõuete täitmist. Ainult 20 tööd sisaldas nii hiire kui ka klaviatuuri sündmusi, mis oli üliõpilastelt nõutud. Nende asemel eelistati JavaFX'i ActionEvent sündmusi, mida rakendati kasutajaliidese nuppude jaoks. Kuna ActionEvent sündmusi kasutati nuppude puhul väga tihti, siis otsustas autor neid edaspidi lihtsuse huvides

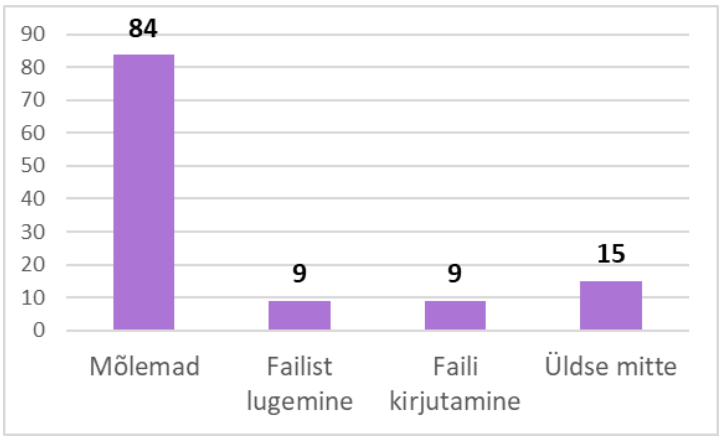
nimetada nupu sündmusteks. Tabel 1 toob välja hiire, klaviatuuri ja nupu sündmuste erinevad esinemissagedused programmides.

Tabel 1. Hiire, klaviatuuri ja nupu sündmuste kombinatsioonide esinemissagedused

Esinemissagedus	**Hiire sündmused**	**Klaviatuuri sündmused**	**Nupu sündmused**
46			x
19	x		x
15	x		
12	x	x	x
10		x	x
8	x	x	
5			
2		x	

Nupu sündmusi esines koguni 87 programmis ehk 74,4%-s kõigist analüüsitud projektidest. Kuna enamus programme sisaldas nuppe, võisid need sündmused osutuda üliõpilaste jaoks kõige mugavamaks. Hiire sündmusi esines 54 programmis ja klaviatuuri sündmusi kõige vähem – kõigest 32 programmis. Klaviatuuri sündmuste vähesus võis tuleneda sellest, et õppijad ei leidnud neile programmis rakendust. Näiteks tegid üliõpilased palju mänge, kus klaviatuuri kasutamine polnud vajalik. Lisaks leiab autor, et arvatavasti ei pidanud üliõpilased sündmuste nõuet väga oluliseks. Õppijad võisid eeldada, et kui nupu sündmused olid programmis olemas, siis polnud rohkem vaja lisada. Sellest tulenevalt soovitab autor, et kui kursuse korraldajad peavad oluliseks, et programmid sisaldaksid nii klaviatuuri kui ka hiire sündmusi, tuleb õppematerjalidesse lisada rohkem näiteid nende kasutamise kohta. Kui aga mõlema sündmuse esinemine pole nii oluline, võib seda nõuet ka vabamaks lasta. Sel juhul võib nõue olla, et programm peab töötlema sündmusi kasutajaliidese elementidel, kuid ei seata täpsemaid piiranguid.

Üliõpilaste programmid pidid sisaldama nii failist lugemist kui ka faili kirjutamist. Joonis 12 annab ülevaate, kuidas seda nõuet täideti.

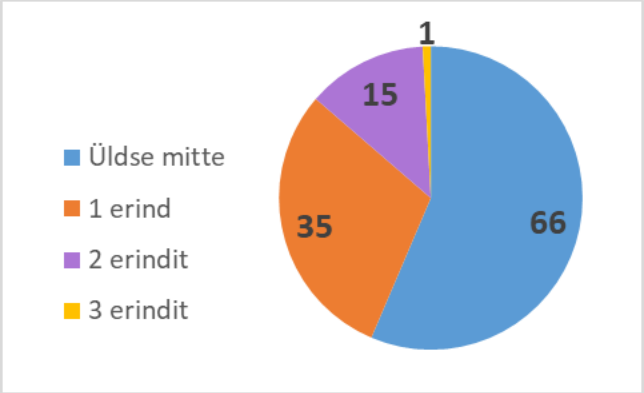


Joonis 12. Failitöötlus programmides

Failitöötlusega said üldiselt üliõpilased hästi hakkama ning 84 programmi ehk 71,8% kõigist rühmatöödest sisaldas nii failist lugemist kui ka kirjutamist. See näitab, et valdav enamus üliõpilastest olid omandanud failitöötluse põhioskused. 18 tööd sisaldas kas failist lugemist või kirjutamist ning 15 töös failitöötlus puudus. Üheksa gruppi tõid rühmatöö kirjelduses mure-kohana välja failitöötluse (vt joonis 10). Probleemseks osutus faili püsivalt andmete kirjutamine nii, et ka pärast programmi sulgemist jääksid andmed faili alles. Samuti valmistas raskusi failist andmete kättesaamine. Mitmetes programmides oli küll olemas kood failitöötluse jaoks, kuid kuna programm oli poolikuks jäänud, siis ei rakendatud seda.

Üliõpilased pidid erinditöötlust rakendama vähemalt mingile kasutaja ekslikule tegevusele. Töö autor otsustas aga analüüsida erinditöötlust laiemast perspektiivist ning mitte piirduda üksnes kasutaja eksimuste käsitlemisega. Vaadati kas ja mitu erindit lõid üliõpilased ise, kuidas erinditega ümber käidi ning milliseid erindeid püüti. Joonis 13 illustreerib üliõpilaste poolt loodud erindite arvu.

Programmi nõuete hulka ei kuulunud spetsiifiliste erindite loomise kohustus, ent 51 rühma otsustas sellele vaatamata seda siiski teha. Enda poolt loodud erindid aitavad pakkuda kasutajale täpsemaid veateateid ja kohandada erinditöötlust vastavalt konkreetsele vajadusele.



Joonis 13. Üliõpilaste poolt loodud erindite arv

Üliõpilased rakendasid ise loodud erindeid peamiselt selleks, et teavitada kasutajaid ebasobivast sisestusest ja failidega seotud probleemidest ning et ennetada loogikavigu. Näiteks kontrolliti, et lõppkuupäev ei jääks alguskuupäevast varasemaks ja et kulud ei ületaks tulusid. Kuna erinditöötluse puhul kehtib reegel, et mida spetsiifilisem, seda parem, siis on enda erindite loomine väga kasulik. Ise erindite loomine võimaldab kirjutada püüniseid, mis püüavad ainult enda loodud erindit, mistõttu ei pea kartma, et püütakse kogemata vale erind [32].

Enamik rühmi puutus oma programmides kokku mitmesuguste erinditüüpidega. See tõi kaasa ka varieeruvuse nende käsitlemisviisides programmi siseselt. Tabelis 2 on toodud täpsem ülevaade erinevatest võimalustest, kuidas üliõpilased erindite püüdmisele ja nende käsitlemisele lähenesid.

53 töös leidus erindeid, mida ei püütud ning kaheksas programmis ei püütud ühtegi erindit. Autor määratles erindite mittepüüdmise olukorrana, kus meetod viskas erindi, mida ei püütud kinni ei antud meetodis ega ka ülemistes meetodi väljakutsetes. Erindite püüdmine või püüdmata jätmine sõltub konkreetsest olukorrast ja erindi tüübist. Kui on võimalik erindist tulenev probleem lahendada, on soovitatav erind kinni püüda. Vastasel juhul on mõistlikum erind edasi suunata [32].

Tabel 2. Erindite püüdmine ja nende käsitlemine

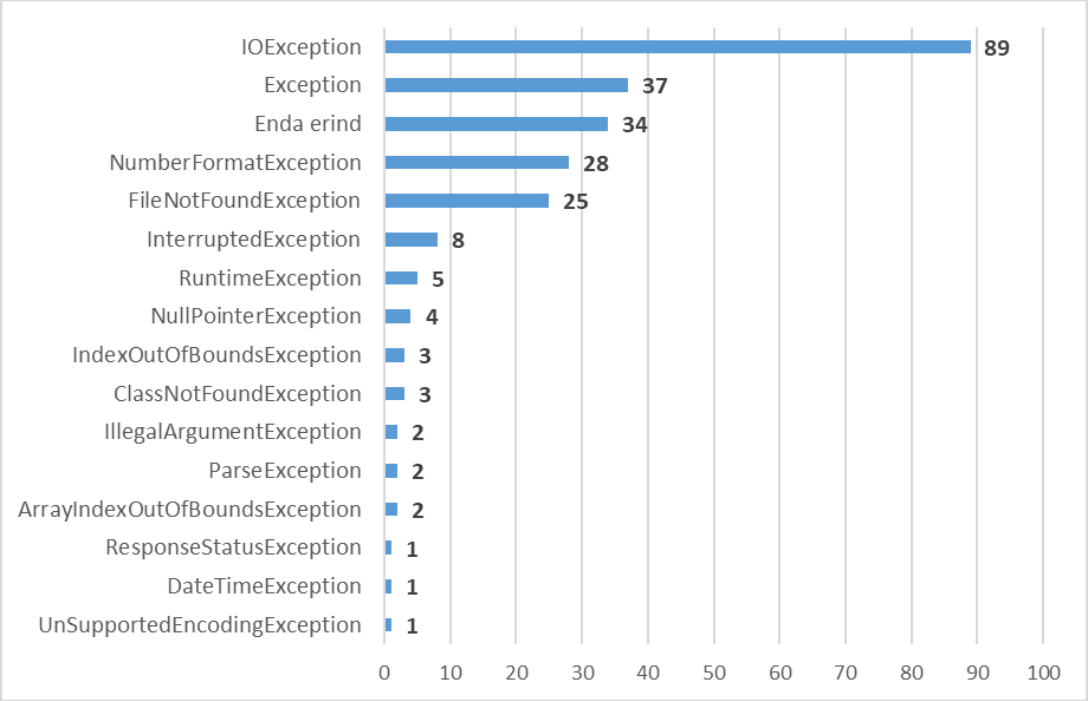
Esinemissagedus	Erindeid ei püüta	Erindid püütakse, aga ei teha midagi	Erindid püütakse ja tehakse
52			x
41	x		x
12		x	x
8	x		
4	x	x	x
0		x	
0	x	x	

Peamiselt jäid püüdmata erindid, mis on seotud failide lugemise ja kirjutamisega, eelkõige FileNotFoundException ja IOException. Nende erindite korrektne käsitlemine aitab ennetada täitmisaegseid vigu, mis võivad põhjustada programmi töö peatumise või isegi kokkujooksmise. 16 rühmatöös erindeid küll püüti, kuid nendega edasi mingit käsitlemist ei toimunud. Selline käitumine võib varjata vigu, mistõttu oleks antud olukorras olnud mõistlikum erindid püüdmata jätta ja need hoopis edasi suunata.

Tervelt 109 töös aga erindeid püüti ning joonis 14 näitab lähemalt milliseid. 76,1% rühmatöödes püüti IOExceptionit. See on mõistetav, arvestades et IOException võib esineda failide lugemise ja kirjutamise käigus ning failitöötlus oli üks programmi nõuetest. IOExceptioni püüdmisel visati 53 töös RuntimeException ning 44 rühma väljastas erindi teabe käsureale. RuntimeExceptioni viskamine oli antud olukorras mõistlik, kui meetodi signatuuri ei saanud lisada *throws* deklaratsiooni erindi edasisuunamiseks [32].

Populaarsuselt teiseks osutus üldise Exception klassi püüdmine, mida tehti 37 töös. Üldise Exception klassi püüdmine, mis haarab endasse nii kontrollitavad kui ka kontrollimatud erindid, ei pruugi olla parim lahendus. See muudab konkreetsete probleemide tuvastamise keeruliseks, sest teadmata milliseid erindeid täpselt püütakse, ei ole võimalik rakendada mõistlikku erinditöötlust. Analüüs näitas, et Exceptionit püüti siis, kui konkreetsema erindi määramine tundus keerukas või arvati, et teatud koodijupp võib probleeme põhjustada. Sageli

kasutati üldise Exceptioni püüdmist ka sobimatu kasutajasisendi korral. Parem lähenemine oleks olnud sisendi hoolikam valideerimine või spetsiifilisemate erindite, näiteks NumberFormatException, püüdmine. Samuti oleks mõistlik olnud luua konkreetsete olukordade jaoks hoopis oma erindi klass. Exceptioni käsitlemises oli kõige levinum praktika RuntimeExceptioni viskamine, mida täheldati 21 programmis. Lisaks väljastati 16 töös erindi teave otse käsureale, millest viie puhul kasutati selleks *e.getStackTrace()* meetodit. Seda pakuvad integreeritud arenduskeskkonnad välja kontrollitavate erindite lahendamiseks. Selline lähenemine ei aita aga parandada vea põhjust, mistõttu ei peeta seda ka sobivaks veakäsitluseks [32].



Joonis 14. Püütud erindid

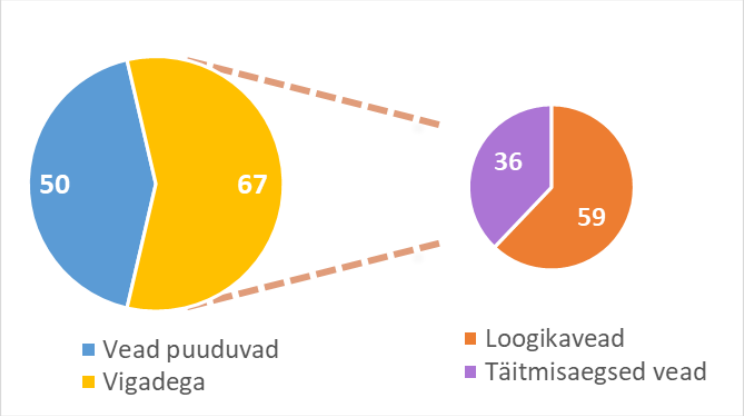
34 projektis püüti rühmatöö autori(te) poolt loodud erindeid, mis olid peamiselt mõeldud ebakorrektse sisendi tuvastamiseks. Nende erindite käsitlemisel eelistati enim kasutajaliidese kaudu veast teavitamist – 15 töös tekstikujul ja 11-s esitati veateade eraldi hüpikaknas. Samuti NumberFormatExceptioni käsitlemisel domineeris veast teavitamine kasutajaliideses, kusjuures üheksal juhul tekstikujul ja 11 töös eraldi veateatena hüpikaknas. Selline lähenemine on konteksti arvestades väga mõistlik. See võimaldas kasutajal saada kasutajaliideses koheselt tagasisidet probleemi kohta ning pakkus võimalust sisestust korrigeerida. Erindeid nagu NullPointerException, IndexOutOfBoundsException, ArrayIndexOutOfBoundsException

ja IllegalArgumentException oleks võinud püüdmise asemel hoopis ennetada. Näiteks NullPointerExceptioni on võimalik enamasti ära hoida, kui kontrollida *if*-lausete abil, et objekt ei oleks väärtusega *null*.

Kuigi vaid kuus rühma tõid kirjelduses esile erinditega seotud probleemid, ilmnes tööde ülevaatamisel, et tegemist oli laialdasema murekohaga. Tundus, et paljud üliõpilased ei olnud täielikult teadlikud, kuidas erindeid asjakohaselt käsitleda. See väljendus nii erindite püüdmises ilma edasiste toiminguteta kui ka juhtudel, kus püüti erindeid, mille püüdmine polnud mõistlik. Eriti märkimisväärne oli üldise Exception klassi laialdane püüdmine, mille asemel oleks tulnud püüda spetsiifilisemaid erindeid. Erindi püüdmine on mõistlik ainult juhul kui suudetakse lahendada erindi poolt viidatud probleem. Sageli piirdusid üliõpilased aga erindi kinnipüüdmise ja selle väljastamisega käsureale, mis ei aidanud lahendada tegelikku probleemi. Sellest tulenevalt soovitab autor kursuse materjalides erinditöötluse osa täiendada. Tuleb selgitada täpsemalt, kuidas käsitleda püütud erindeid ja milliseid erindeid ei peaks püüdma. Kasulik tööriist võib olla ülevaatlik tabel, mis kategoriseerib enimlevinud erinditüübid ja nende soovitatavad käsitlemisviisid, pakkudes õppijatele kiiret ülevaadet korrektsest erindikäsitlusest.

3.5 Vead ja puudujäägid

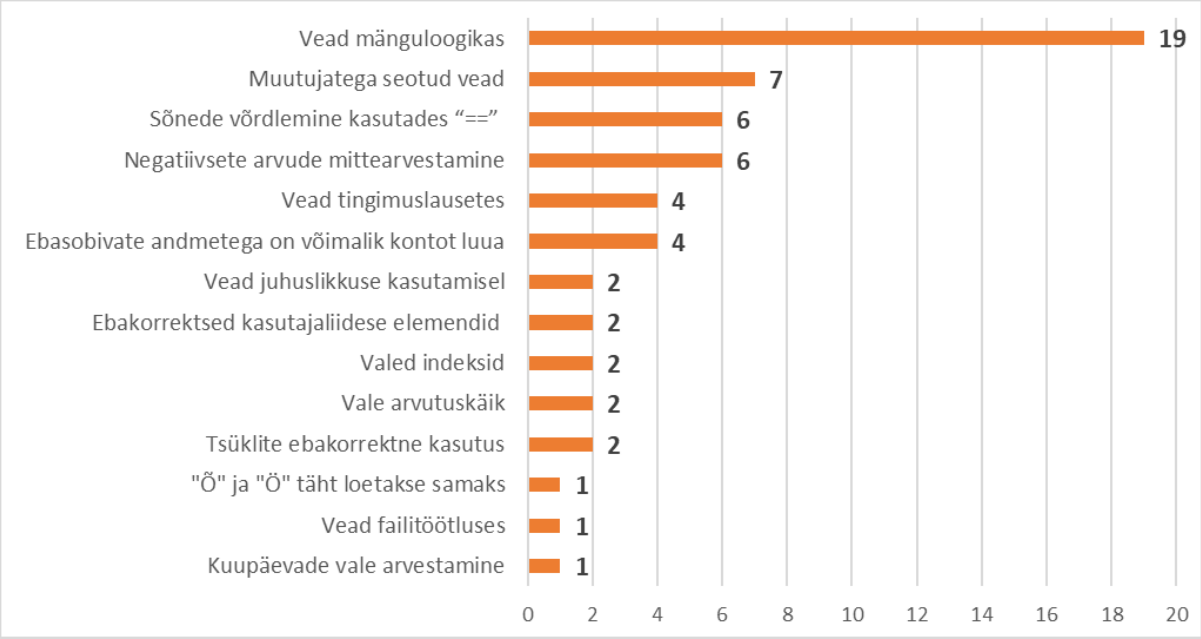
Antud peatükk keskendub sellele, milliseid vigu üliõpilased oma programmides tegid ning milliseid puudujääke esines programmi ja koodi kvaliteedis. Joonis 15 illustreerib kui palju ja milliseid vigu programmides esines.



Joonis 15. Vigade esinemine programmides

Programmide analüüs näitas, et 57,3% rühmatöödest sisaldasid vigu. Antud vead jagunesid kaheks: loogikavead, mida leiti kokku 59 ning täitmisaegsed vead, mida tuvastati 36. Ühes programmis võisid esineda kas ainult loogikavead, ainult täitmisaegsed vead või mõlemad. Ainult loogikavigu leiti 38 programmis, ainult täitmisaegseid 19-s ja mõlemat tüüpi vigu 10 programmis. Süntaksivigu üheski rühmatöös ei esinenud. See oli ootuspärane, arvestades et süntaksivead on kõige kergemini õppijatele avastatavad, kuna ei lase programmil kompileerida.

Joonis 16 illustreerib, milliseid loogikavigu üliõpilased enim tegid. Loogikavigade seas osutusid kõige levinumaks mänguloogikaga seotud vead, mis moodustasid 32,2% kõikidest loogikavigadest. Arvestades et ligikaudu pooled loodud programmidest olid mängud (vt joonis 2), on arusaadav, et just selles kategoorias esines kõige rohkem vigu. Antud kategooria hõlmas näiteks valesti rakendatud võidu, kaotuse ja viigi loogikat, ebatäpset punktiarvestust ja ebasobivate käikude lubamist. Sellised vead võisid tekkida, kuna üliõpilased ei pruukinud kõiki olukordi mängudes läbi testida, mistõttu jäid mingid vead avastamata ja seetõttu ka parandamata.



Joonis 16. Loogikavead programmides

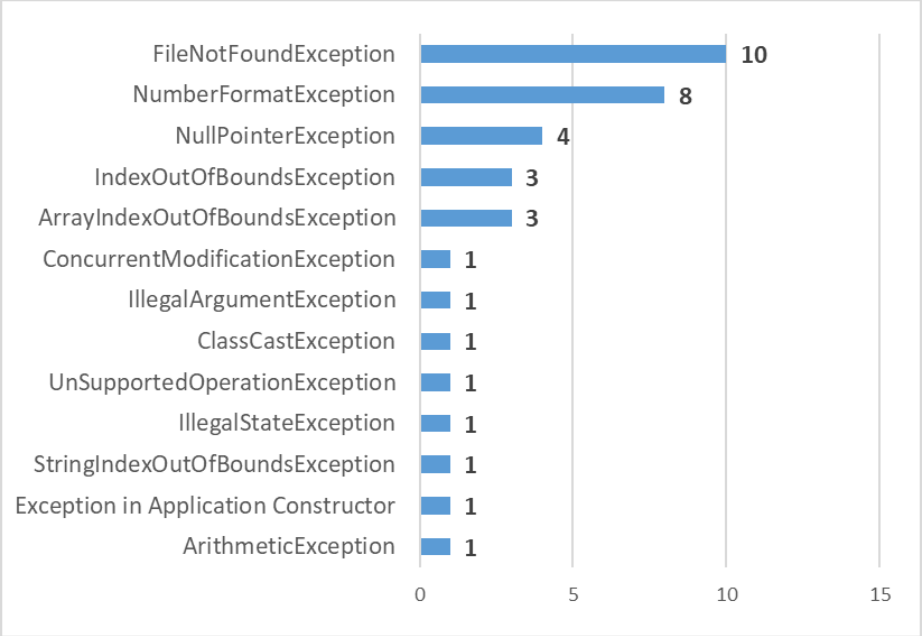
Seitsmes töös leiti muutujatega seotud vigu, mis tihti ilmnesid sarnaste nimedega muutujate segiajamisest või vale muutuja kasutamisest. Tundus, et need olid tekkinud tähelepanematusest

ning jäid seetõttu õppijatel märkamata. Sõnede võrdlemine võrdusmärkidega on tüüpiline algajate viga Javas, see esines kuues programmis. Õige lähenemine oleks olnud sõnede võrdlemisel kasutada meetodit *equals*. Antud vea toovad välja ka integreeritud arenduskeskkonnad, mis näitab, et üliõpilased ei pööranud liigset tähelepanu selle märkustele. Kuues rühmatöös ei arvestatud sisendi kontrollimisel negatiivsete väärtustega. Näiteks oli lubatud lisada pangakontole negatiivne summa või ei arvestatud negatiivseid arve liidetavatena. Kahes programmis esines vigu juhuslikkuse kasutamisel. Mõlemas neist oli valesti määratud *java.Lang.Math.random()* funktsiooni vahemik. Ühel juhul oli põhjuseks sulgude vale järjekord ning teises oli vahemiku ülemine piir väiksem kui loogika järgi oleks pidanud.

Loogikavigade vältimiseks oleksid üliõpilased pidanud oma programme põhjalikumalt testima. Näiteks oleksid nad võinud kasutada probleemsete meetodite läbitöötamiseks silurit. Praegu puuduvad õppematerjalis juhendid siluri kasutamise kohta, kuid nende lisamine oleks kasulik. Eriti seetõttu, et keerukamate programmide puhul lihtsustab siluri kasutamine oluliselt vigade leidmist [31]. Samuti tuleb julgustada õppijaid pöörama rohkem tähelepanu integreeritud arenduskeskkonna märkustele, mis aitavad ennetada mitmeid loogikavigu.

Üliõpilaste töödes esinesid täitmisaegsetest vigadest ainult erindid. Joonisel 17 on välja toodud, millised erindeid üliõpilaste programmid viskasid. Kõige sagedamini, ehk 10 töös esines FileNotFoundException, mis tulenes ebatäpsest failitee määramisest. Mitmes töös oli failitee spetsiifiline üliõpilase arvutile, mistõttu programmi kellegi teise arvutis jooksutades faili ei leitud. Kuna üliõpilaste enda arvutis antud viga ette ei löönud, võis see neile märkamatuks jääda. Enamik FileNotFoundExceptionit visanud programme andis veateate kohe käivitamisel, hoolimata sellest, et mitmed neist oleksid võinud töötada ka ilma antud failita.

Teine levinum viga oli NumberFormatException, mida viskasid kaheksa programmi. See esines peamiselt seetõttu, et kasutaja sisestatud andmed ei ühtinud programmi ootustega. Näiteks kui programm ootas numbrilist väärtust, võis kasutaja sisestada teksti või jätta sisendi tühjaks.

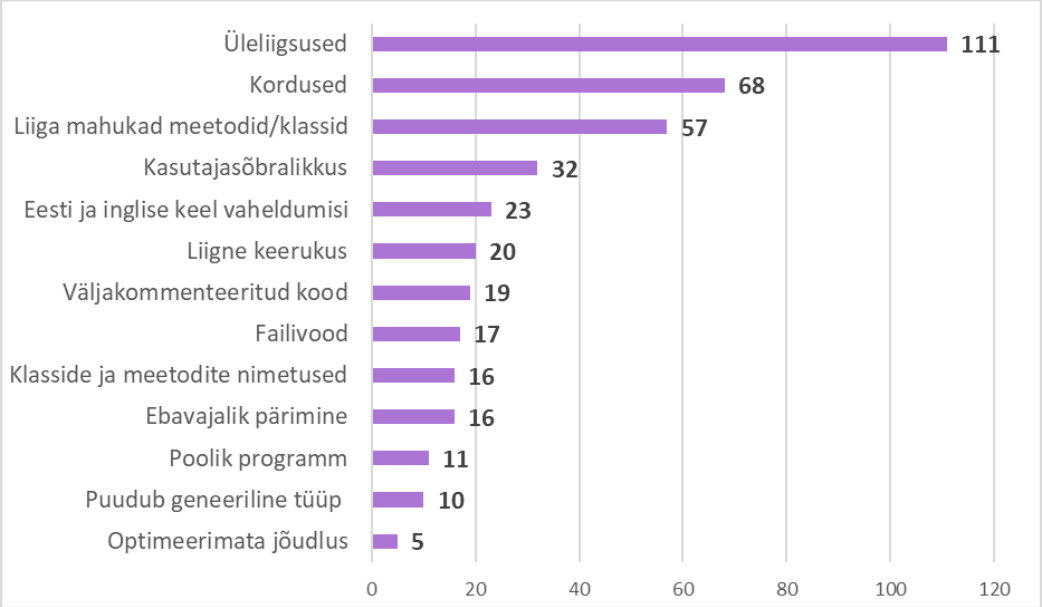


Joonis 17. Täitmisaegsed vead programmides

Ka ArrayIndexOutOfBoundsException, mis esines kolmes rühmatöös, oli tingitud ebapiisavast sisendikontrollist, mis lubas kasutajal olulisi välju tühjaks jätta. Kui programm taaskäivitati, esines antud erind failist lugemisel, sest andmed polnud failis oodataval kujul. Nii NumberFormatException kui ka ArrayIndexOutOfBoundsException oleksid olnud välditavad kui õppijad oleksid kasutajasisendi kontrollile rohkem tähelepanu pööranud. See on eriti oluline programmide puhul, mis salvestavad andmeid faili, kust tuleb neid ka hiljem korrektselt lugeda.

Erindeid oleks olnud suuresti võimalik ennetada, kui neile oleks rakendatud korrektset erinditöötlust või oleks kasutatud *if*-lauseid vigade ja kasutajasisendi kontrollimiseks. See kinnitab järjekordselt, et erinditöötluse osa õppematerjalides peab olla põhjalikum.

Lisaks vigadele analüüsiti ka teisi tegureid, mis ei pruukinud otseselt programmi funktsionaalsust mõjutada, kuid millel oli mõju koodi ja programmi kvaliteedile. Joonis 18 annab täpsema ülevaate antud puudujääkidest.



Joonis 18. Programmi ja koodi kvaliteediga seotud puudujäägid

Peaaegu igas rühmatöös esines mingeid üleliigsusi, sealhulgas üleliigsed *import*-laused, kasutamata muutujad, meetodid ja klassid ning üleliigsed semikoolonid ja muutuja väärtustamised. Need üleliigsused tegid koodi ebavajalikult pikaks ja ebaselgeks. Lisaks leiti enam kui pooltes töödes korduvat koodi, mille alla liigitati nii identsed kui lihtsalt korduvat loogikat kasutavad koodijupid. Korduva koodi vältimiseks on mõistlik koondada korduv funktsionaalsus eraldi meetoditesse, mis võimaldab funktsionaalsust taaskasutada. Ka kordused ja üleliigsused olid tihti integreeritud arenduskeskkonna poolt märgitud, kuid õppijad jätsid need tähelepanuta.

57 töös täheldati liiga mahukaid meetodeid ja klasse, mis ei järginud üksikvastutuse põhimõtet *(single responsibility principle)*. Antud põhimõtte kohaselt peab üks meetod või klass tegelema ainult ühe funktsionaalsusega [40]. Eriti probleemseks osutus *start*-meetod, mis JavaFX'is loob kasutajaliidese peamise akna. Sageli sisaldas antud meetod enamikku koodist, ulatudes mõnel juhul isegi 1000 reani. Paar üliõpilast tõi ka rühmatöö kirjelduses välja, et ei osanud JavaFX'i elemente efektiivselt meetoditesse ja klassidesse jaotada, mis võis olla ka niivõrd mahukate *start*-meetodite põhjustajaks. Liiga mahukad meetodid muudavad koodi aga raskemini loetavaks, raskendavad vigade leidmist ning vähendavad koodi taaskasutatavust.

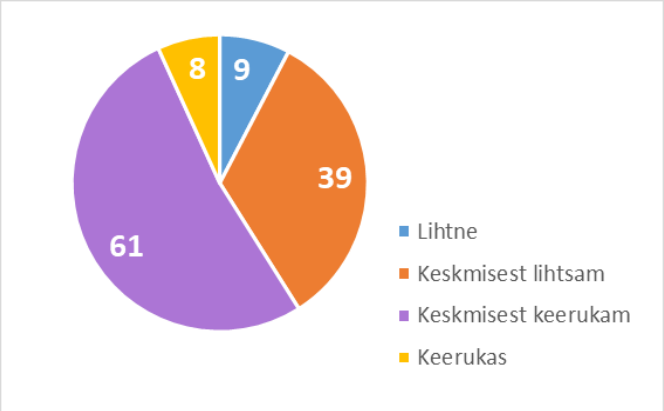
Kategooria “Kasutajasõbralikkus” hõlmas töid, mis põhjustasid kasutajatele mingeid ebamugavusi kasutajakogemuses. Sellesse kuulusid programmid, mis olid tõusutundlikud, kus mingi osa töötas käsureal või kus esines probleeme kasutajaliidese elementidega. Samuti liigitati siia programmid, kus loodi kasutajaliides kasutades nii JavaFX’i kui ka Java standardteeki Swing. Tegemist pole tavapärase lähenemisega, sest mõlemad raamistikud haldavad sündmusi ja ressursse erinevalt. See põhjustas ebaühtsusi kasutajaliideses, kus näiteks üks kasutajaliidese aken nägi ühtemoodi välja ning teine oli teistsugune. Autori hinnangul esinesid kasutajasõbralikkuse probleemid peamiselt seetõttu, et üliõpilastel jäi ajast puudu. Seetõttu ei jõudnud nad pühenduda kasutajakogemuse parandamisele. Autor soovitab õppijatele selgitada, et kasutajaliidese loomisel on mõistlik kasutada ühte raamistikku ja selle komponente. Vastasel juhul võib kasutajaliideses esineda ebaühtsusi.

Veel võib välja tuua ebavajaliku pärimise klasside vahel. Selles kategoorias osutus suurimaks probleemiks mitme klassi poolt Application klassi pärimine. Application klass on abstraktne klass, mis määratleb põhilise raamistiku JavaFX’i programmide kirjutamiseks [31]. Kui mitu klassi pärisid Application klassi, tekkis programmil mitu sisenemispunkti ehk mitu käivitatavat klassi. See tekitas segadust, sest polnud selge, milline klass oli mõeldud peamiseks käivituspunktiks.

Kuigi puudujäägid koodi kvaliteedis ei ole nii tõsised probleemid nagu vead, mõjutavad need siiski programmi hooldatavust ja jõudlust [34]. Ka õppijate kood oli kvaliteediprobleemide tõttu tihtilugu raskesti jälgitav ning programmid polnud kasutajasõbralikud. Seega oleks kasulik lisada õppematerjalidesse peatükk, mis käsitleb koodi kvaliteedi parimaid praktikaid. Samuti on üks idee määrata koodi kvaliteedi eest õppijatele eraldi hinne, et motiveerida neid sellele rohkem tähelepanu pöörama [35]. Autor leiab, et isegi motiveerivam võib olla õppijate premeerimine boonuspunktidega kvaliteetselt kirjutatud koodi eest. Seda võib rakendada nii rühmatöös kui ka teistes hindelistes programmerimisülesannetes.

3.6 Programmide keerukus

Keerukuse hindamisel võttis autor arvesse koodiridade arvu, klasside hulka, lisamoodulite kasutust, nõuetele vastavust ja programmi üldist funktsionaalsust. Nende tegurite põhjal jaotati rühmatööd keerukuse alusel ning seda jaotust illustreerib joonis 19.



Joonis 19. Programmide keerukus

Programmid, mille maht oli väike, kus mitmed nõuded olid täitmata ja funktsionaalsused osutusid puudulikuks, määratles autor hinnanguga "lihtne". Selliseks osutus üheksa rühmatööd. Kaks rühma tõid välja, et töö jäi poolikuks ajapuuduse tõttu ning ühes rühmas jäi programm lõpetamata, kuna paariline ei teinud enda osa. Hinnang “keskmisest lihtsam” anti töödele, mis polnud niivõrd poolikud kui hinnangu “lihtne” saanud programmid, kuid mis siiski jätsid mitmed nõuded täitmata ja polnud eriti mahukad. Enamik rühmatöid, täpsemalt 52,1%, klassifitseeriti “keskmisest keerukamaks”, kuna need vastasid suuresti nõuetele ning olid mahult keskmised või suuremad. “Keerukaks" hinnati kaheksa rühmatööd, millest kõik kas hõlmasid mitmeid lisamooduleid või olid märkimisväärse koodimahu ja funktsionaalsustega. Näiteks liigitus siia töö, mis integreeris Next.js, Spring Boot ja PostgreSQL andmebaasi. Kuigi antud programm polnud koodiridade arvult väga mahukas, eristus see märgatavalt tänu mitmetele kasutatud lisamoodulitele, mis lisasid sellele keerukust.

Järgnevalt on tabelis 3 esitatud ülevaade koodiridade, klasside arvu ning keerukuse kohta. Samuti on välja toodud erinevused ühe-, kahe- ja kolmeliikmeliste rühmade vahel. Keerukuse teisendas autor neljapalliskaalale, kus 1 tähistab hinnangut “lihtne” ning 4 “keeruline”.

Rühma suuruse ja programmi keerukuse osas valitses ootamatu seos: mida suurem oli rühm, seda madalam oli nende klasside arv ja keerukus. Kuigi kolmeliikmelistelt rühmadelt oodati mahukamaid programme, näitasid andmed, et nende keskmine ridade ja klasside arv ning keerukus jäid teistele rühmadele märkimisväärselt alla. See on vastuolus eeldusega, et suurema liikmete arvuga rühmad võiksid luua keerukamaid programme. Need tulemused võivad viidata võimalusele, et mõned üliõpilased valisid kolmeliikmelise rühma, et vähendada töökoormust,

mitte et luua keerukam projekt. Kolmeliikmelised rühmad, kelle esitatud tööd olid oodatust väiksema mahuga, said vastavalt ka madalamad hinded.

Tabel 3. Koodiridu, klasside arvu ja keerukust kirjeldav statistika

	Ridade arv	Klasside arv	Keerukus
Miinimum	104	1	1
Maksimum	1619	20	4
Keskmine	489,3	6,2	2,6
Mediaan	417	6	3
Standardhälve	271,4	3,2	0,7
Individuaaltööde keskmine	479,6	6,3	2,7
Kaheliikmeliste rühmade keskmine	497,7	6,2	2,6
Kolmeliikmeliste rühmade keskmine	379,5	5,7	2,4

Kuigi kaheliikmeliste rühmade keskmine koodiridade arv oli suurem kui individuaalselt tegutsenud õppijatel, jäi nende klasside arv ja keerukus siiski väiksemaks. Antud tulemuste puhul tuleb arvesse võtta, et rühmade suurus jaotus ebavõrdselt – individuaaltöid oli 15, kaheliikmelisi 96 ning kolmeliikmelisi kõigest kuus, mis avaldas suurt mõju antud tulemustele.

Kokkuvõte

Antud bakalaureusetöö eesmärk oli analüüsida Tartu Ülikooli kursuse "Objektorienteeritud programmeerimine" rühmatööde peamiseid tugevusi ja nõrkusi ning rühmatöö nõuete täitmist. Nende põhjal sooviti pakkuda kursuse korraldajatele tagasisidet ja soovitusi kursuse materjalide täiendamiseks. Andmete kogumiseks vaadati läbi 117 rühmatööd ning 112 rühmatöö kirjeldust, mille põhjal täideti andmetabel. Eesmärgist lähtuvalt püstitati ka viis uurimusküsimust.

Esimeses uurimusküsimuses tahtis autor teada, millistel teemadel rühmatöid koostati. Tulemuste põhjal jaotas autor teemad 18 kategooriasse, millest ülekaalukalt populaarseim oli mängude loomine, mida tegid ligi pooled rühmadest. Loodi nii klassikalisi mänge nagu erinevad sõna- ja kaardimängud kui ka originaalsemaid mänge. Veel olid populaarsed finantsi ja haridusega seotud teemad.

Teises küsimuses uuriti, milline oli üliõpilaste programmide ülesehitus ning keerukus. Koodiridade arvu suur variatsioon (keskmine 489,3 ja standardhälve 271,4) oli ootuspärane, arvestades, et rühmatöö nõuete raames oli võimalik luua erineva mahuga programme ning üliõpilaste programmeerimise oskus on erinev. Kuna programmides nõuti graafilise kasutajaliidese olemasolu, valis 102 rühma selle loomiseks JavaFX'i. Kui JavaFX välja jätta, kasutas lisamooduleid vaid viis rühma. Enamik rühmatöid hinnati "keskmisest keerukamaks", sest täitsid suures osas nõudeid ning olid keskmise või suurema mahuga. Üllatuslikult olid kolmeliikmelistel rühmadel nii ridade ja klasside arv kui keerukus väiksem kui oli eeldatud.

Kolmandale küsimusele vastates tahtis autor teada, kuidas üliõpilased said hakkama programmi nõuete täitmisega. Üllatavalt täitsid kõiki nõudeid vaid neli rühma ning lausa 113 töös jäid mingid nõuded täitmata. Kõige enam puudusid programmidest hiire ja/või klaviatuuri sündmused. Nende asemel kasutati JavaFX'i ActionEvent sündmust kasutajaliidese nuppude jaoks. Autori arvates polnud 53 tööd mõistlikult kommenteeritud – kommentaarid kas puudusid või neid oli vähe. Veel esines 47 töös probleeme kasutajaliidese akna suuruse mõistliku muutumisega ning 41 töös oli puudulik erindikäsitlus kasutaja ekslikule tegevusele. Failist lugemise ja kirjutamise nõue polnud täidetud 33 programmis. Mitmes neist oli kood küll olemas, kuid programmi poolikuse tõttu oli see jäänud rakendamata.

Neljas küsimus otsis vastust, kuidas üliõpilased täitsid rühmatöö kirjelduse nõudeid. Tulemused olid paremad kui programmi nõuete täitmisel ning 64 rühma puhul olid kõik kirjelduse nõuded täidetud. Enim jäeti kirjeldusest välja ajakulu, mida ei maininud 26 rühma, tõenäoliselt seetõttu, et seda on keeruline hinnata. Keskmine ajakulu oli rühma kohta 20,1 tundi ning 10,3 tundi liikme kohta. Rühmad veendusid oma programmide korrektsuses kasutades erinevaid testimismeetodeid, kusjuures kõige populaarsemaks osutus kasutajaliidese testimine, mida rakendas 77 rühma. 84 rühma jäid oma lõpptulemusega rahule või väga rahule, viis rühma, kes eriti rahule ei jäänud, tõid kõik välja ajapuuduse. Protsessi kirjeldustest selgus, et enamik rühmi otsustas ülesanded omavahel jagada ning suure osa tööst individuaalselt teha.

Viimasele küsimusele vastates tahtis autor teada, millised olid rühmatööde peamised puudujäägid. Tulemustest selgus, et vigu tehti 67 töös. Peamised loogikavead esinesid mänguloogikas, kus oli probleeme võidu ja kaotuse rakendamisega. Täitmisaegsetest vigadest osutusid levinumaks FileNotFoundException ning NumberFormatException, mis tekkisid vale failitee kasutamise ja sobimatu sisendi lubamise tõttu. Samuti osutus koodi kvaliteet suureks probleemiks – 111 töös esines üleliigsusi, 68-s kordusi ning 57-s liiga mahukaid meetodeid ja klasse, mis raskendasid koodi mõistmist. Üliõpilased tõid ise välja peamise murekohana graafilise kasutajaliidese. Oli näha, et probleeme valmistasid nii sündmused, akna suuruse paindlik kohandamine kui ka elementide paigutus. Ka erinditöötlus oli õppijatel probleemne. Näiteks püüti erindeid, mida poleks pidanud ning jäeti käsitlemata need, mida oleks vaja.

Rühmatööde analüüsi põhjal pandi kirja soovitusi materjalide täiendamiseks. Analüüsi tulemused näitasid, et kõige rohkem vajavad kursuse materjalides tähelepanu graafilise kasutajaliidese, erinditöötluse ning koodi kvaliteedi teemad.

Edasistes uuringutes oleks huvitav võrrelda tulemusi mitme aasta lõikes, et märgata seaduspärasid või probleemseid kohti, mis on kas läbi aastate samaks jäänud või hoopis viimastel aastatel esile kerkinud. Kursuse läbiviijad saavad antud lõputöös leitud tulemuste põhjal täiendada “Objektorienteeritud programmeerimise” kursust. Tööd saab kasutada ka täiendavate uurimuste tegemisel objektorienteeritud programmeerimise kursuste jaoks.

Viidatud kirjandus

[1] Hoc J.-M., Nguyen-Xuan A. Chapter 2.3 - Language Semantics, Mental Models and Analogy. D. J. Gilmore, J.-M. Hoc, R. Samurçay, T. R. G. Green. (Eds.), *Psychology of Programming*. London: Academic Press, 1990, 139-156. https://doi.org/10.1016/B978-0-12-350772-3.50014-8.

[2] Mahmud M. M., Wong S. F. Digital age: The importance of 21st century skills among the undergraduates. *Frontiers in Education*, 2022, 7. https://doi.org/10.3389/feduc.2022.950553.

[3] Cheung, H. Y., Wong, G. K. W. Exploring children's perceptions of developing twenty-first century skills through computational thinking and programming. *Interactive Learning Environments*, 2020, 28(4), 438–450. https://doi.org/10.1080/10494820.2018.1534245.

[4] Berssanette J. H., Francisco A. Active Learning in the Context of the Teaching/Learning of Computer Programming: A Systematic Review. Journal of Information Technology Education: Research, 2021, 20, 201-220. https://doi.org/10.28945/4767.

[5] Dori Y. J., Lavi R., Tal M. Perceptions of STEM alumni and students on developing 21st century skills through methods of teaching and learning. *Studies in Educational Evaluation*, 2021, 70. https://doi.org/10.1016/j.stueduc.2021.101002.

[6] Aivaloglou E., Van der Meulen A. Who Does What? Work Division and Allocation Strategies of Computer Science Student Teams. *IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training [ICSE-SEET]*, 2021, 273-282. https://doi.org/10.1109/ICSE-SEET52601.2021.00037.

[7] Christensen E. L., Paasivaara M. Learning Soft Skills through Distributed Software Development. *In Proceedings of the International Conference on Software and System Processes and International Conference on Global Software Engineering (ICSSP'22)*, 2022, 93–103. https://doi.org/10.1145/3529320.3529331.

[8] Bourgeois A. G., Metzler M., Sunderraman R., Younis A. A. Case Study: Using Project Based Learning to Develop Parallel Programing and Soft Skills. *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2019, 304-311. https://doi.org/10.1109/IPDPSW.2019.00059.

[9] Tartu Ülikooli Õppeinfosüsteem. Objektorienteeritud programmeerimine (6 EAP). https://ois2.ut.ee/#/courses/LTAT.03.003/version/5fcd37f8-4721-56f8-a8e6-39786150e413/details (14.05.2024).

[10] Tartu Ülikooli arvutiteaduse instituudi kursuste veebileht. Objektorienteeritud programmeerimine. *Kursuse Korraldus*. https://courses.cs.ut.ee/2023/OOP/spring/Main/KursuseKorraldus (14.05.2024).

[11] Tartu Ülikooli arvutiteaduse instituudi kursuste veebileht. Objektorienteeritud programmeerimine. *2. rühmatöö*. https://courses.cs.ut.ee/2023/OOP/spring/Main/Ruhm2 (14.05.2024).

[12] Figueiredo J., García-Peñalvo F. J. Teaching and learning strategies of programming for university courses. *Proceedings of the Seventh International Conference on Technological Ecosystems for Enhancing Multiculturality (TEEM'19)*, 2019, 1020-1027. https://doi.org/10.1145/3362789.3362926.

[13] Mohorovicic S., Strcic V. An Overview of Computer Programming Teaching Methods. *Central European Conference on Information and Intelligent Systems*, 2011, 47-52. https://api.semanticscholar.org/CorpusID:56241852 (14.05.2024).

[14] Ala-Mutka K., Järvinen, H.-M., Lahtinen E. A study of the difficulties of novice programmers. *ACM SIGCSE Bulletin*, 2005, 37(3), 14-18. https://doi.org/10.1145/1067445.1067453.

[15] Bogdanovych A., Trescak T. Teaching Programming Fundamentals to Modern University Students. *In Proceedings of the 8th International Conference on Computer Supported Education - Volume 2*, 2016, 308-317. https://doi.org/10.5220/0005809403080317.

[16] Prensky, M. Digital Game-Based Learning. New York: Paragon House Publishers. 2007.

[17] Airaksinen, J., Vihavainen, A., Watson C. A Systematic Review of Approaches for Teaching Introductory Programming and Their Influence on Success. *In Proceedings of the tenth annual conference on International computing education research (ICER '14)*, 2014, 19-26. https://doi.org/10.1145/2632320.2632349.

[18] Angotti R. L., Goldstein D. S., Hillyard C., Nordlinger J., Panitz M. W., Sung K. Game-Themed Programming Assignment Modules: A Pathway for Gradual Integration of Gaming Context Into Existing Introductory Programming Courses. *IEEE Transactions on Education*, 2011, 54(3), 416-427. https://doi.org/10.1109/TE.2010.2064315.

[19] Ferzli M., Miller C., Wiebe E., Williams L., Yang K. In support of pair programming in the introductory computer science course. *Computer Science Education*, 2002, 2(3), 197–212. https://doi.org/10.1076/csed.12.3.197.8618.

[20] Arisholm E., Dybå T., Hannay J. E., Sjøberg D. I. K. The effectiveness of pair programming: A meta-analysis. *Information and Software Technology*, 2009, 51(7), 1110-1122. https://doi.org/10.1016/j.infsof.2009.02.001.

[21] Nuutila E., Malmi L., Törmä S. PBL and Computer Programming — The Seven Steps Method with Adaptations. *Computer Science Education*, 2005, 15(2), 123-142. https://doi.org/10.1080/08993400500150788.

[22] Guzdial M., Hewner M. What game developers look for in a new graduate: interviews and surveys at one game company. *In Proceedings of the 41st ACM technical symposium on Computer science education (SIGCSE '10)*, 2010, 275–279. https://doi.org/10.1145/1734263.1734359.

[23] Begel A., Simon B. Struggles of new college graduates in their first software development job. *In Proceedings of the 39th SIGCSE technical symposium on Computer science education (SIGCSE '08)*, 2008, 40(1), 226–230. https://doi.org/10.1145/1352322.1352218.

[24] Radermacher A., Walia G. Gaps between industry expectations and the abilities of graduates. *In Proceedings of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE'13)*, 2013, 525–530. https://doi.org/10.1145/2445196.2445351.

[25] Begel A., Simon B. Novice software developers, all over again. *In Proceedings of the Workshop on Computing Education Research*, 2008, 3-14. https://doi.org/10.1145/1404520.1404522.

[26] Aivaloglou E., Meulen A. An Empirical Study of Students' Perceptions on the Setup and Grading of Group Programming Assignments. *ACM Transactions on Computing Education*, 2021, 21(3), 1-22. https://doi.org/10.1145/3440994.

[27] Faily S., Iacob C. Exploring the gap between the student expectations and the reality of teamwork in undergraduate software engineering group projects. *The Journal of Systems and Software*, 2019, 157, 16-17. https://doi.org/10.1016/j.jss.2019.110393.

[28] Du Boulay B. Some Difficulties of Learning to Program. J*ournal of Educational Computing Research*, 1986, 2(1), 57-73. https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9.

[29] Winslow L. E. Programming pedagogy—a psychological overview. *SIGCSE Bull*, 1996, 28(3), 17–22. https://doi.org/10.1145/234867.234872.

[30] Guerrero C. A., Gutiérrez L. E., López-Ospina H. A. Ranking of problems and solutions in the teaching and learning of object-oriented programming. *Educ Inf Technol*, 2022, 27, 7205–7239. https://doi.org/10.1007/s10639-022-10929-5.

[31] Liang Y. D. Introduction to Java Programming, Comprehensive (10th Edition). New Jersey: Pearson Education. 2014.

[32] Tartu Ülikooli arvutiteaduse instituudi kursuste veebileht. Objektorienteeritud programmeerimine. *Nädal 10 Veahaldus.* https://courses.cs.ut.ee/2023/OOP/spring/Main/Practice10 (14.05.2024).

[33] Hristova M., Mercuri R., Misra A., Rutter M. Identifying and correcting Java programming errors for introductory computer science students. *In Proceedings of the 34th SIGCSE technical symposium on Computer science education (SIGCSE '03)*, 2003, 153–156. https://doi.org/10.1145/611892.611956.

[34] Heeren B., Jeuring J., Keuning H. Code Quality Issues in Student Programs. *In Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '17)*, 2017, 110–115. https://doi.org/10.1145/3059009.3059061.

[35] Birillo A., Bryksin T., Golubev Y., Keuning H., Tigina M., Vyahhi N. Analyzing the Quality of Submissions in Online Programming Courses. *In Proceedings of the 45th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET '23)*, 2023, 271–282. https://doi.org/10.1109/ICSE-SEET58685.2023.00031.

[36] Tartu Ülikooli arvutiteaduse instituudi kursuste veebileht. Objektorienteeritud programmeerimine. *1. rühmatöö*. https://courses.cs.ut.ee/2023/OOP/spring/Main/Ruhm1 (14.05.2024).

[37] Gomes A., Mendes A.J. Learning to program — difficulties and solutions. *International Conference on Engineering Education*, 2007, 283-287. http://icee2007.dei.uc.pt/proceedings/papers/411.pdf (14.05.2024).

[38] Java™ Platform Standard Edition 8. Package java.io. https://docs.oracle.com/javase/8/docs/api/index.html?java/io/package-summary.html (14.05.2024).

[39] Java™ Platform Standard Edition 8. Package java.util. https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html (14.05.2024).

[40] Martin R. C. Agile Software Development: Principles, Patterns, and Practices. USA: Prentice Hall PTR. 2003.

Lisad

I. Rühmatööde andmetabel

Järgnevalt veebilehelt leiab rühmatööde analüüsi andmetabeli:

https://docs.google.com/spreadsheets/d/1O-DaJj8ZIMqcDJTE9ZtgVp0CPf-Wv4nTZHJpQQUhrmA/edit?u3sp=sharing

II. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, **Eva Luude,**
(autori nimi)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

Kursuse "Objektorienteeritud programmeerimine" rühmatööde analüüs,
(lõputöö pealkiri)

mille juhendaja on **Marina Lepp**,
(juhendaja nimi)

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 4.0, 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.

Eva Luude
14.05.2024