TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Carolin Kirotar

Programmeerimiskursuse „Tehnoloogia tarbijast loojaks“ projektide analüüs

Bakalaureusetöö (9 EAP)

Juhendaja: Marina Lepp, PhD

Tartu 2021

Programmeerimiskursuse „Tehnoloogia tarbijast loojaks“ projektide analüüs

Lühikokkuvõte:

Käesolevas bakalaureusetöös analüüsiti programmeerimiskursuse “Tehnoloogia tarbijast loojaks” osalejate koostatud projekte. Uurimuse eesmärgiks oli välja selgitada kursusel osalejate poolt loodud projektide tugevad ja nõrgad küljed ning selle põhjal anda kursuse korraldajatele soovitusi materjalide täiendamiseks ja kursuse läbiviimiseks. Töö käigus leiti vastused järgnevatele uurimisküsimustele: millised olid õpilaste struktuurilised eelistused projekti koodi kirjutamisel, millised olid õpilaste projektide peamised nõrgad kohad, millise hinnangu andsid kursusel osalejad projekti tegemise etapile.

Võtmesõnad:

MOOC noortele, programmeerimise projekt, programmeerimise õpetamine

CERCS: P175 Informaatika, süsteemiteooria**,** S280 Pedagoogika ja didaktika

Analysis of the Programming Course "Technology from Consumer to Creator" Projects

Abstract:

This bachelor's thesis analyzed the projects made by the participants of the programming course "Technology from Consumer to Creator". The purpose of the study was to find out the strengths and weaknesses of the projects created by the course participants and to give recommendations to the course organizers on supplementing the materials and conducting the course. As a result of this analysis, the following research questions were answered: which were the students' structural preferences when writing the project code, which were the main weaknesses of the student projects, and which valuation did the participants give to the project development phase.

Keywords:

MOOC for young, programming project, teaching programming

CERCS: P175 Informatics, systems theory, S280 Pedagogy and didactics

Sisukord

Sissejuhatus

Programmeerimine on oskus, mille teadmine tänapäevasel nutiajastul on väärtuslik juba pea igas valdkonnas. Selle tundmine avab uksi, mis muidu võivad tööturul suletuks jääda (Vallistu & Danilov, 2018; Kori et al., 2019). Palju tööd tehakse selle nimel, et programmeerimine või vähemalt informaatika üldisemalt leiaks erinevates kooliastmetes võrdväärse koha teiste ainetega (Juurak, 2018). Nii saaks varakult ettekujutuse antud valdkonnast ja teadmise, kas on soovi sellega kõrgkoolis jätkata (Kori et al., 2019). Aina enam pakutakse õppimise võimalust ka erinevate veebikursuste näol, et igas vanuses ja erineva taustaga inimestel oleks võimalik oma teadmisi laiendada. Informaatika didaktika töörühma andmetel hakati 2014. aastal Tartu Ülikoolis pakkuma MOOCi (ingl *Massive Open Online Course*) nimega "Programmeerimisest maalähedaselt". Kursus osutus nii populaarseks, et 2016. aastal tehti sellest põhjalikum edasiarendus "Programmeerimise alused" (Informaatika didaktika töörühm, s.a).

Käesolevas lõputöös vaatluse all olev kursus "Tehnoloogia tarbijast loojaks" loodi 2020. aastal eesmärgiga laiendada programmeerimise õppimise võimalust veelgi (Informaatika didaktika töörühm, s.a). See toimub paralleelselt nii koolides üle Eesti kui e-õppena. Kursus on justkui "Programmeerimise alused" järeltulija, eriti kuna nimetatud kursus toimus viimati 2018. aastal. See on mõeldud 16-26 aastastele noortele, kellel puudub varasem programmeerimise kogemus, kuid kellel on suur huvi seda omandada.

Jälgides viimaste aastate statistikat Tartu Ülikooli informaatika erialale sisseastujate kohta, võib öelda, et huvi valdkonna vastu vaid tõuseb. Kuid ka väljalangevus esimeselt aastalt ja samuti esmastelt programmeerimiskursustelt on jätkuvalt väga suur (Kori et al., 2019). Seetõttu on oluline, et kursused, mille algajad läbivad, tooksid sära silmadesse või vastupidi annaksid mõista, kas programmeerimine on valdkond, kus edasi pürgida. Autor valis oma bakalaureusetööks just selle teema, et panustada kursuse edukale edasiarendusele, nii et võimalikult paljud täiendaksid ennast programmeerimise alal edasi.

Programmeerimise kursustel on tavaks, et kursuse lõpuosas tehakse omandatud teadmiste põhjal suurem projekt, mis tuletaks õpitud teadmisi meelde, kinnistaks neid ja annaks võimaluse projekti käigus uusi oskusi juurde saada. Kursusel "Tehnoloogia tarbijast loojaks" pidid osalejad tegema toimiva digilahenduse arendusprojekti, mis vastaks etteantud nõuetele. Peale projekti edukat sooritamist paluti õppuritel vastata tagasiside küsitlusele, mis sisaldas osalejate poolseid hinnanguid tehtud tööle.

Käesoleva bakalaureusetöö eesmärk on välja selgitada 2020. aasta kevadel toimunud programmeerimiskursuse "Tehnoloogia tarbijast loojaks" vältel tehtud digilahenduse arendusprojektide tugevad ja nõrgad küljed ning selle põhjal anda soovitusi kursuse korraldajatele materjalide täiendamiseks ja kursuse läbiviimiseks. Eesmärgist lähtuvalt pani autor paika järgmised küsimused, millele uurimisel keskendub:

- Millised olid õpilaste struktuurilised eelistused projekti koodi kirjutamisel?
- Millised olid õpilaste projektide peamised nõrgad kohad?
- Millise hinnangu andsid kursusel osalejad projekti tegemise etapile?

Töö algab ülevaatega programmeerimist, selle erinevatest õpetamismeetoditest, ülesannete olulisusest, erinevusest rühmatöö ja individuaalse töö vahel ning algajate programmeerijate põhilistest vigadest. Järgmises peatükis tuuakse näiteid programmeerimise veebikursustest ja tutvustatakse lähemalt kursust "Tehnoloogia tarbijast loojaks". Kolmas peatükk on pühendatud metoodika kirjeldamisele. Neljandas peatükis esitatakse tulemused, arutletakse nende üle ja jagatakse soovitusi edaspidiseks.

1. Ülevaade programmeerimise õpetamisest

Peatükis antakse ülevaade programmeerimise õpetamise erinevatest meetoditest, ülesannete olulisusest, rühmatöödest ja algajate põhilistest vigadest.

1.1 Programmeerimise õpetamisest ja erinevatest meetoditest

Programmeerimise õpetamine võib olla keeruline protsess. Jazayeri (2015) põhjal teeb õpetamise keeruliseks asjaolu, et õpilased peavad lisaks programmeerimiskeele süntaksile ja semantikale omandama samal ajal ka mitmeid erinevaid uusi rakenduslikke oskusi, nagu põhilisi programmeerimistehnikaid, kõrgtasemelisi abstraheerimisvõimeid, disaini ja nende kontseptsioonide rakendamist probleemide lahendamisel. Välja on töötatud erinevaid õpetamismeetodeid, millel kõigil on omad eelised soovitud oskuste välja arendamise protsessis. Neist meetoditest tuntumad on projektipõhine, probleemipõhine ja oskusõpe (ingl *mastery learning*).

Projektipõhine õpe sobib hästi kõrgtasemeliste ja loovate oskuste õpetamiseks (Jazayeri, 2015). Õpilased koondatakse tavaliselt gruppidesse ja neile antakse uurimisprobleem, millele hakatakse lahendust otsima. Jazayeri (2015) järgi aitab meeskonnas töötamine arendada kaaslastega töötamist, võimaldab teistelt uusi teadmisi saada ja tõstab motivatsiooni. Kaaslastega töötades suudetakse enamasti luua oluliselt keerulisem programm kui üksinda, arendades seejuures enda oskusi edasi ning suurendades õpilaste saavutustaset (Jazayeri, 2015).

Luxton-Reilly jt (2018) kohaselt on probleemipõhine õpe sobilik keerulistele probleemidele lahenduste väljatöötamiseks. Õpilased kogevad seeläbi praktilisi probleeme, millega nad teoreetilise õppimise käigus kokku ei puutuks (Jazayeri, 2015). Mitmed uuringud on näidanud, et probleemipõhine õpe tõstab motivatsiooni ja suurendab sotsiaalset interaktiivsust, kuid samal ajal on vähe tõendeid sisuliste teadmiste paranemisest (Luxton-Reilly et al., 2018).

Oskusõpe sobib konkreetsete oskuste arendamiseks (Jazayeri, 2015). Luxton-Reilly jt (2018) kohaselt tuleb oskusõpe raames saavutada käsitletavas teemas teatud tase, enne kui saab teemasse põhjalikumalt süveneda või järgmise juurde edasi liikuda. See on seotud loogikaga, et programmeerimise õppimine nõuab enne keerukamate elementide juurde pöördumist süntaksi põhielementide mõistmist (Luxton-Reilly et al., 2018). On leitud tõendeid, et oskusõpe sobib paremini nõrgematele õpilastele, sest see aitab neil tugevamate õpilastega ühele ühtlasele tasemele jõuda (Luxton-Reilly et al., 2018).

Päriselus tihti kombineeritakse neid erinevaid meetodeid. Topalli ja Cagiltay (2018) toovad välja, et hea viis algkursuse läbiviimiseks on päriselul põhinevate probleemipõhiste mängude loomine, sest see parandab potentsiaalselt õppijate oskusi ja tõstab motivatsiooni. Mängude tegemine aitab keskenduda mitte ainult süntaksiprobleemidele, vaid õpetab nägema süsteemi terviklikku pilti, pannes rõhku nii algoritmide kui süsteemi arendamisele ja kujundamisele (Topalli & Cagiltay, 2018). Samas näitavad Černochová, Selcuki ja Černý (2020) uuringud, et õpilaste edu programmeerimisel sõltub nende huvist vastava teema kohta, stressivabast õpikeskkonnast, meeskonnaliikmetega koostööst ja vabadusest luua iseseisvalt, ilma õpetaja suunamiseta, uusi lahendusi. Jazayeri (2015) rõhutab, et projektipõhisest õppimisest kasu saamiseks peab õpilane olema kõigepealt ületanud paika pandud lävendi, kuna vaid seeläbi saavad kõik meeskonnaliikmed projekti võrdselt panustada. Neid andmeid võrreldes on näha, et toetatakse esmalt oskusõppe rakendamist ja seejärel projektipõhist lähenemist koos probleemipõhise õppega. Nii saavad õpilased ühtse meeskonnana erinevate reaalelul põhinevate ülesannetega oma programmeerimisoskusi harjutada.

1.2 Programmeerimise ülesannetest ja nende olulisusest

Programmeerimine ei koosne vaid teooriast, märgatava osa moodustavad vastupidi praktilised oskused. Heaks programmeerijaks ei saada üleöö, vaid tuleb järjepidevalt harjutada. Suurt rolli mängivad aga ülesanded, mis õpilastele antakse, sest vastavalt nende struktuurile, raskusastmele ja arusaadavusele tulevad õppijate oskused.

Programmeerimisülesanded on oluline osa õppimisprotsessist. Nende põhjal saab teada, kui hästi on õpilane teooria omandanud ja kui hästi seda kasutada oskab. Kuna programmeerimise õppimine ei ole lihtne, siis Konecki, Lovrenčić ja Kaniški (2016) sõnul võiksid ülesanded olla loogika mõistmiseks päriselul põhinevad ja tulemused praktilised. Nii saaksid õpilased aimu, millised ülesanded võivad neid tulevikus professionaalsetel töökohtadel ees oodata (Konecki et al., 2016).

Ülesanded võivad olla motivaatoriks või vastupidi negatiivseks mõjuks. Luxton-Reilly jt (2018) põhjal kasutatakse mängude loomist selleks, et õpilasi motiveerida programmeerimise õpingutel. Õpilased saavad niimoodi võimaluse kasutada rohkem oma loomingulist poolt, kui rangelt struktureeritud ülesandeid lahendades. Õpilastele endile on kasulik, kui nad saavad programmeerimist õppida materjalide abil, mis ei anna valmislahendusi ette, vaid sunnivad neid mõtlema ja toovad uusi küsimusi juurde (Černochová et al., 2020). Selle põhjal võib öelda,

et motivatsiooni säilitamiseks peab õpilastes huvi püsima või lausa kasvama. Seda saab omakorda teha, kui anda neile huvitavaid, parajalt väljakutsuvaid ja loomingulisi, näiteks mängudega seotud ülesandeid.

Samas vastavalt Cliburni, Milleri ja Bowringi (2010) uuringu tulemustele eelistab valdav enamus algajaid programmeerijaid kindlate kriteeriumitega etteantud ülesandeid. Kui programmeerimisoskus on alles arendamisel, siis on arusaadav, et kardetakse suuremaid väljakutseid ette võtta. Muidu võib juhtuda, et soov programmeerimist õppida kaob üldse ära. Tuleb tähelepanu pöörata ka sellele, et inimesed on erinevad ja seetõttu sobivad neile erinevad ülesanded. On loogiline, et kui õpilane ei saa ülesandega hakkama ning ta ei julge nõu küsida, siis tema motivatsioon langeb. Sellise olukorra vältimiseks tuleb õpilastesse sisendada, et eksimine on inimlik ja abi küsimises pole midagi halba.

Vastupidiselt kogu eelnevale toovad Luxton-Reilly jt (2018) välja, et kursuse kontekst pole üldse nii oluline kui hea ülesehitus ja hea tugi. See tähendab, et tublimad õpilased omandavad vajalikud oskused ka neile ebasümpaatsete ülesannetega. Oluline on vaid, et nad ülesandesse süveneksid, selle loogikat mõistaksid ja vajadusel juhendajalt nõuandeid saaksid.

1.3 Rühmatöödest

Reaalses elus nõuab uute programmeerimisalaste lahenduste loomine kogu meeskonna kooskõlastatud jõupingutusi. Eelnevalt selgus, et meeskonnaliikmed peavad olema oskustelt sarnasel tasemel kui soovivad ühepalju töösse panustada (Jazayeri, 2015). Sancho-Thomas, Fuentes-Fernández ja Fernández-Manjón (2009) kohaselt on oluline ka juba varakult omandada meeskonnatöö edukust toetavaid oskusi, näiteks suhtlemise, juhtimise ja läbirääkimisoskust. Nende võimete õppimine on väga oluline (Luxton-Reilly et al., 2018) ning nõuab suurt pühendumist (Sancho-Thomas et al., 2009), mida saab ideaalselt praktiseerida juba algaja programmeerijana rohkem rühmatöid tehes ja saavutatud tulemustest õppides.

Rühmatöö suureks eeliseks individuaalse töö ees peetakse teadmiste ja kogemuste jagamist meeskonnaliikmetega. Ghobadi, Campbelli ja Cleggi (2017) põhjal tõstavad ühised eesmärgid jagatavate teadmiste kvaliteeti, mis omakorda potentsiaalselt parandab töö kvaliteeti. Hannay, Dybå, Arisholm ja Sjøberg (2009) täheldasid soolo- ja paarisprogrammeerimist võrreldes tähtsa asjaoluna ka seda, et paarisprogrammeerimine võimaldab saavutada kvaliteetsema koodi lahenduse umbes poole ajaga, mis kuluks individuaalse programmeerimise korral. Lisaks saab

paarisprogrammeerimise käigus arendada meeskonnas töötamise oskust, vahetada teadmisi neid samal ka süvendades ja üldiselt kiiremini õppida (Hannay et al., 2009).

Ühest vastust küsimusele, kas õpilased eelistavad rühmatööd või individuaalset tööd, on raske leida. Kui õpilased saavad endale ise tiimiliikme valida, siis eelistatakse pigem seda varianti, sest nagu Monaghan jt (2014) välja toovad on hea läbisaamine tiimiliikmetega ja sarnane ideaal toimiva meeskonnatöö aluseks. Ka Luxton-Reilly jt (2018) rõhutavad, et õpilased naudivad koostööd nii sotsiaalse suhtlemise kui teiste eeliste, nagu enesekindluse suurenemise, püsivuse ja tulemuslikkuse pärast. Kogu eelnevat arvesse võttes tuleks kindlasti rühmatöid teha, sest nendest saab korraga rohkem kasu kui iseseisvast töötamisest.

Huvitava leiuna tuli Mülleri (2007) andmetel välja, et rühmas programmeerijad teevad umbkaudu sama palju vigu sooloprogrammeerijatega. Lisaks ei tule enamikest uurimustest välja, kumb programmeerimise variant on otstarbekam või kummaga on tulemused tunduvalt paremad. Seetõttu ei saa konkreetselt öelda, kas eelistada individuaalset või rühmatööd. Hannay jt (2009) järgi sõltub see programmeerijate enda oskustest ja iseloomust ning lahendatavate ülesannete keerukusest. Samas tuleb meeles pidada, et rühmatööd tehes saab lisaks programmeerimise oskuste arendamisele ka sotsiaalseid oskusi parandada.

1.4 Algajate põhilistest vigadest

Annamaa (s.a) kirjutatud Tartu Ülikooli programmeerimise algkursuse õpikus on toodud mõte, et programmeerimine pole põhjani õpitav, nagu näiteks luuletuse päheõppimine või muu lihtsa oskuse omandamine. Pythoni programmeerimise reeglite ühekaupa meeldejätmisest on vähe kasu, kui neid ei osata kombineerida. Lisaks tuleb osata näha ülesande sisemusse, lahendust esmalt ette kujutada ja seejärel see programmeerimiskeelde panna (Annamaa, s.a). Programmeerimine sisaldab endas nii palju erinevaid nüansse, mistõttu on vead lihtsad tulema.

Vead jaotatakse enamasti kolme kategooriasse: süntaksivead, täitmisaegsed vead ja loogika vead (Annamaa, s.a; Hristova, Misra, Rutter & Mercuri, 2003). Süntaksivead ilmnevad Hristova jt (2003) sõnul peamiselt vigases programmi õigekirjas, kirjavahemärkides või sõnade järjekorras ning nende tuvastamine on enamasti veateadete abil lihtne. Kuid õpilased ei pruugi krüptilistest veateadetest aru saada, mistõttu ei oska nad vigu parandada (Hristova et al., 2003). Täitmisaegsed vead, mida nimetatakse ka erinditeks, käsitlevad programmi täitmisel tekkivaid

vigu. Nende vigade puhul programm küll käivitub, kuid edasine käsu täitmine ei toimi (Annamaa, s.a). Ka nende puhul väljastatakse veateated, mis võiksid programmeerijaid aidata. Loogikavead on kõige üldisemad ja ühtlasi kõige raskemini avastatavad. Nende puhul programm ei pruugi veateadet anda, kuid vaatamata sellele ei tee programm täpselt seda, mida programmeerija sooviks (Annamaa, s.a). Loogikavigasid aitab leida rohke testimine. Algajate põhilisteks vigadeks on Fitzgerald jt (2008) andmetel näiteks valed omistuslaused, ebakorrektsed tsüklite kasutamised, võrdlemiseks ühe võrdusmärgi kasutamine kahe asemel nagu Pythoni puhul õige oleks, sulgude unustamine, valesti vormistatud tõeväärtuslaused ja muud taolised vead. Õnneks suudavad õpilased enamasti sellised vead iseseisvalt katse- ja eksitusmeetodil ära lahendada, kui nad need üles leiavad (Fitzgerald et al., 2008).

Vigade avastamiseks soovitatakse õpilastel oma koodi siluda (ingl *debugging*). Ko ja Myers (2005) defineerivad silumist kui tarkvara ja käitusvigade avastamist, mis toovad kaasa suurema käitustõrke ning koodi muutmist nende vigade likvideerimiseks. Kuid samas arvatakse, et algajate jaoks on see tegevus keeruline ja nad ei oska seda ära kasutada (Fitzgerald et al., 2008). Seetõttu tuleks kood kohe alguses kirjutada võimalikult puhtalt, korrektselt, jälgida elementaarseid reegleid ja iga koodilõiku testida.

2. Ülevaade programmeerimise veebikursustest

Programmeerimisest on tänapäeval saanud oskus, mille tundmine on kõigile kasulik. Sellel põhjusel on loodud erinevaid kursuseid, mis annavad baasteadmised põhilistest programmeerimisega seotud mõistetest ja oskustest. Alljärgnevalt käsitletakse eelkõige noortele suunatud veebipõhiseid programmeerimise algkursuseid ja antakse ülevaade töös põhiliselt vaadeldavast kursusest "Tehnoloogia tarbijast loojaks".

2.1 Noortele suunatud programmeerimise veebikursuste tutvustus

Enamikus Eesti gümnaasiumides on programmeerimine juba vähemalt valikainena valitav. See on Kori jt (2019) põhjal eeliseks neile õpilastele, kes plaanivad ülikooli informaatikat edasi õppima minna. Aine annab teada, kas valdkond pakuks üldse huvi. Samas selgub Puniste (2015) uuringust, et igas koolis õpetatakse erinevalt ja õpetamiseks kasutatakse suisa 14 erinevat keelt. Seega võib ebasobiv keele valik ja õpetamistehnika huviliste edasisi karjäärivalikuid mõjutada, sest algus ei tundu paljulubav.

Tartu Ülikooli arvutiteaduste instituut on läbi aegade pakkunud mitmeid erineva suunitlusega MOOCi kursuseid: „Programmeerimisest maalähedaselt", „Programmeerimise alused" ja „Programmeerimise alused II". Samuti on ülemaailmselt loodud eraldi veebikeskkondasid, kus pakutakse programmeerimise õppimise võimalust. Tuntumad neist on näiteks Codecademy, Coursera, edX, Code.org, Free Code Camp ja palju teised.

2.2 Kursuse „Tehnoloogia tarbijast loojaks" tutvustus

Informaatika didaktika töörühma korraldatud programmeerimiskursus „Tehnoloogia tarbijast loojaks" on loodud tutvustamaks noortele vanuses 16-26, kellel kokkupuude programmeerimisega puudub või on väga vähene, algoritmilist mõtlemist ja programmeerimise algtõdesid. Kõigil programmeerimishuviga noortel on võimalik 10 nädalat kestva algkursuse raames saada aimu, kui väga neile programmeerimise valdkond meeldiks. Kursus on edasiarendus gümnaasiumis pakutavast valikkursusest „Programmeerimine", kuid on mõeldud ka õppida soovijatele väljaspool kooli. Temaatiliselt on see sarnane samuti Tartu Ülikoolis pakutava e-kursusega „Programmeerimise alused". Kursuse raames tutvutakse programmeerimiskeelega Python.

Kursuse pikkus on 10 nädalat, mille vältel keskendutakse igal nädalal erinevale teemale. Iga teemaplokk on omakorda jaotatud mitmeks alateemaks. Seda on tehtud selleks, et tutvustada

õpilastele põhjalikult ühe teemaga seonduvat ja lihtsustada nende õpinguid. Kursuse kava nädalate kaupa on järgmine (Informaatika didaktika töörühm, s.a):

1. Sissejuhatus
2. Tingimuslause
3. Tsükkel
4. Sõnad. Graafika
5. Järjend
6. Funktsioon
7. Andmevahetus. Lihtne kasutajaliides
8. Digilahenduse arendusprojekt
9. Kordamine
10. Arvestustöö

Kursusega seotud materjalid asuvad kolmes keskkonnas (Informaatika didaktika töörühm, s.a). Kursus põhineb gümnaasiumi valikkursustega „Programmeerimine“ ja “Tarkvaraarendus” samadel õpikul, mille autoriteks on Eno Tõnisson, Tauno Palts, Merilin Säde, Kaarel Tõnisson ja teised (Tõnisson et al., 2019a; Tõnisson et al., 2019b). Esimesest neist leiab materjale iganädalaste teemade kohta, mille läbitöötamine hõlbustab kohustuslike kontrollülesannete lahendamist. Lisaks põhilistele õppematerjalidele võib õpikutest leida ka silmaringi laiendavaid materjale ning näiteprogramme.

Kursuse enda materjalid ja õpingute vahearvestused paiknevad Tartu Ülikooli Moodle õpikeskkonnas. Esimesel seitsmel õpinädalal tuleb digiõpikust uus vajalik materjal läbi töötada ja lahendada enesekontrolliülesanded. Iga kord tuleb lahendada kaks kuni viis iseseisvat programmeerimisülesannet ning sooritada nädalatest. Test kontrollib teooriast arusaamist ning väikeste ülesannete loogika mõistmist. Ülesannete korrektsust hindab automaatkontroll. Oma programmeerimisülesannete lahendustele tuleb automaatkontrolli teostada läbi keskkonna Lahendus. Kursuse lõpuosa keskendub mahukamatele ülesannetele, milleks on digilahenduse arendusprojekt, kordamine ja arvestustöö. Digilahenduse projekt tehakse individuaalselt või 2-3-liikmelistes rühmades ning see võtab kokku põhilise kursusel õpitu. Projekte vaatavad üle ja hindavad arvestatuks mentorid-õpetajad, kellel on vastav ettevalmistus ja kes on esitanud soovi olla an-

tud kursusel õppijate nõuandjaks. Õpilased, kes sooritavad arvestustöö väga edukalt, tulemusega vähemalt 90%, saavad võimaluse eritingimustel asuda õppima Tartu Ülikooli Informaatika bakalaureuseõppesse (Informaatika didaktika töörühm, s.a).

3. Metoodika

Antud peatükis antakse ülevaade uurimuses käsitletavast projektist, kasutatud valimist ja analüüsiks rakendatud meetoditest.

3.1 Projekti kirjeldus

Käesolevas töös on vaatluse all 2020. aasta kevadel toimunud programmeerimiskursuse "Tehnoloogia tarbijast loojaks" raames teostatud digilahenduse arendusprojekt. Projektiks loetakse õpilase enda püstitatud ülesannet, mis hõlmab kogu kursusel läbitud materjale. Kuna projekt on mõeldud kursuse vältel õpitut tervikuks kokku võtma, on see mahukam ja keerulisem kui varasemalt õpilastele antud ülesanded. Autor valis analüüsimiseks just selle ülesande, et välja selgitada, mida saaks kursuse korraldusliku poole pealt paremini teha. Järgnevalt on välja toodud digilahenduse arendusprojekti juhend, mis on esitatud samasugusel kujul nagu kursuse materjalides.

Seni on kursusel enamasti pidanud lahendama etteantud ülesandeid. Nüüd on võimalus ja isegi kohustus lahendada ülesanne, mille olete ise püstitanud. Ega väga suurt programmi ei saa selle kursuse materjalide põhjal teha, aga midagi huvitavat ja kasulikku siiski saab. Nimetame seda ülesannet (digilahenduse arendus)projektiks.
Projekt tehakse individuaalselt või 2-3-liikmelistes rühmades. Orienteeruv töömaht ühe rühmaliikme kohta on 8 tundi.

Nõuded

- Lahendus peab olema enda tehtud ja uus.
- Programmeerimiskeel peab olema Python.
- Lahenduse failis peab kommentaarina sisalduma ka püstitatud ülesande tekst ja autorite nimed.
- Lahendus ise peab olema mõistlikult kommenteeritud.
- Ülesande lahendus peab sisaldama vähemalt ühte endakirjutatud funktsiooni ja selle rakendamist.
- Ülesande lahenduses peab olema vähemalt üks tingimuslause.
- Ülesande lahenduses peab olema vähemalt üks tsükkel.

- Ülesande lahenduses tuleb failist lugeda või/ja kasutajalt andmeid küsida.
- Ülesande lahenduses tuleb faili või/ja ekraanile väljastada.
- Ülesande lahendus peab sisaldama vähemalt ühte järjendit.

Projektis võib kasutada vahendeid, mida meie kursuse põhiosas ei õpetata. Toome välja paar moodulit (link), millest võib kasu olla.

Etapid

1. Ülesande püstituse ja rühma koosseisu esitamine (Moodle'is)
2. Programmi esitamine (Moodle'is)
    a. (Programmi esitlemine klassis)
3. Aruandevormi täitmine (Moodle'is)

3.2 Valim

2020. aasta kevadel, 9. märtsist kuni 24. maini toimunud programmeerimiskursusel osales kokku 469 õpilast, kellest 134 olid naised ja 335 mehed. Osalejate vanus ulatus 14. kuni 34. eluaastani. Antud töö valimiks on need osalejad, kes sooritasid kursuse raames tehtava digilahenduse arendusprojekti hindele arvestatud. Nendeks oli 225 õppurit, kellest 50 olid naissoost ja 175 meessoost. Ka nende vanus jäi vahemikku 14 kuni 34 eluaastat, kusjuures keskmine vanus oli 19,25 ja standardhälve oli 2,69. Kursuse lõpus paluti õpilastel täita tagasiside küsitlus, millele olid kohustatud vastama kõik projekti teostanud õpilased. Ankeetküsitlusele vastas 219 õpilast, nii et kuus õpilast jätsid mingil põhjusel vastamata. Kursuse sooritas edukalt 202 osalejat ehk osad, kes projektist küll läbi said, ei lõpetanud kursust.

3.3 Protseduur

Andmete kogumiseks kasutati kursusel osalejate endi koostatud projektide lahendusi ja korraldajate koostatud aruandevormi. Projekti tegemine toimus kaheksandal ja üheksandal nädalal, mil kogu teooria osa oli läbitud. Tagasisideküsitlus viidi läbi pärast projektide valmimist. Küsitlus andis infot projektile kulunud aja, protsessi etappide, puudujäänud teadmiste või oskuste ja üldise arvamuse kohta. Projektidest ülevaate saamiseks vaadati läbi kõik 197 lahendust, mille andmed pandi statistika koostamiseks ja analüüsimiseks kirja andmetabelisse (vt lisa I).

Andmetabelisse märkis autor üles järgmised punktid, mida antud töös hakatakse põhjalikumalt analüüsima:

- rühmatöö suuruse valik (üheliikmeline/kaheliikmeline/kolmeliikmeline)
- nõuete täitmine (jah/ei),
- vigade olemasolu (täiesti vigadeta/pisikeste vigadega/suurte vigadega/ei tööta üldse ja kirjas millised),
- üleliigsed asjad (jah/ei ja kirjas millised),
- kordused (jah/ei ja kirjas millised),
- muutujate ja funktsioonide nimede asjakohasus (asjakohased/ebatäpsed/ei sobi) ning keel (eesti/inglise/mõlemad),
- *for*- või *while*-tsükli valik (*for*/*while*/mõlema kasutamine),
- kommentaaride põhjalikkus (väga põhjalik/põhjalik/kommenteeritud/kommenteerimata),
- lisamoodulite kasutamine (jah/ei ja kirjas milliste),
- graafilise liidese olemasolu (jah/ei ja kirjas millise),
- ridade ja funktsioonide arv (arvuliselt),
- programmi keerukus (keeruline/keskmine/lihtne/liiga lihtne),
- mõistetavus teistele (jah/ei ja põhjendatud miks),
- projekti teemavalik (lahtiselt),
- inspiratsioon näidisülesannetest (palju/veidi/pole ja kirjas millistest ülesannetest).

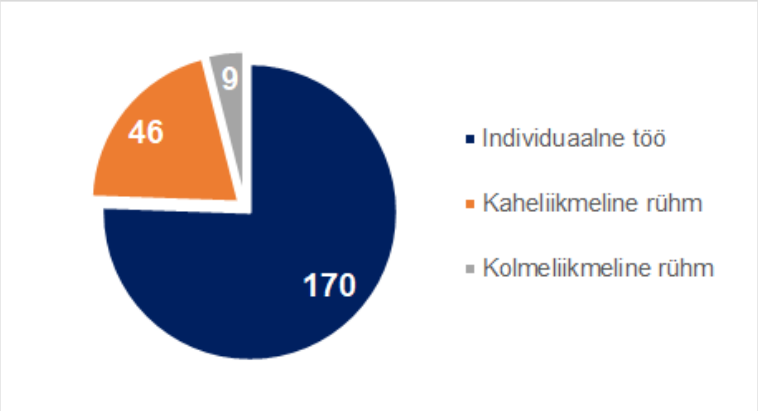
Peamiselt hinnati punkte kinniselt, kas paari või mitme palli skaalal. Need on märgitud ülaltoodud punktide taga sulgudes. Osasid punkte analüüsiti lahtiselt: parandusi kirja pannes, soovitusi andes või tähelepanekuid üles märkides.

4. Tulemused ja soovitused

Antud peatükis tuuakse välja analüüsitud projektide ja tagasiside küsimustiku statistika. Autor annab ka omapoolseid soovitusi kursuse läbiviijatele.

4.1 Rühma suuruse valik ja nõuete täitmine

Enne projektiga alustamist anti õpilastele valikuvõimalus, kas nad soovivad üksinda või meeskonnana töötada. Joonisel 1 on kujutatud kursusel osalejate valik, kas teha projekt kahe- või kolmeliikmelise rühmana või hoopis individuaalselt.

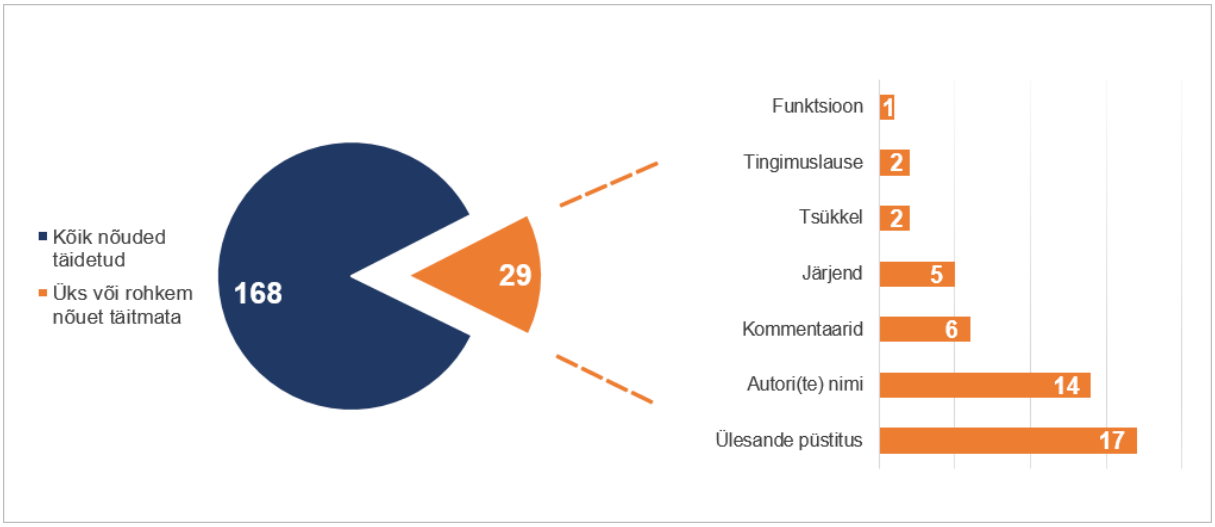


Joonis 1. Osalejate arv vastavalt rühmadele.

Tulemustest selgus, et lausa 76% õpilastest otsustas tööd teha üksinda. See võis tingitud olla sellest, et 2020. aasta kevadel valitses pandeemia, mis sundis inimesi koduseinte vahel üksinda töötama ega võimaldanud kokkusaamisi. Mõjutada võis ka see, et kursusele kogunesid inimesed, kes tõenäoliselt varem teineteist ei tundnud ja kui antakse valida, eelistatakse rühmatööd teha sõbra või tuttavaga (Monaghan et al., 2014). Jooniselt 1 on näha, et kui valiti rühmatöö, siis tehti seda peamiselt kaheliikmelistes meeskondades. Neid rühmi oli kokku 24. Joonisega võrreldes võib märgata õpilaste arvus erinevust, sest kahe liikme paarilised ei osalenud aktiivselt antud kursusel ja seetõttu neid ei arvestatud. Eelistus võis tuleneda sellest, et paaris on lihtsam ülesandeid ära jaotada kui suuremates rühmades. Lisaks ei nõutud õpilastelt selle projekti raames midagi eriti mahukat, mis tähendab, et kaks liiget meeskonnas oli ideaalne tiimi suurus kiireks ja korralikuks tööks. Kursuse korraldajatele võib küll soovitada enam-vähem ühtlase tasemega õpilaste ranget paaripanemist, et nad saaksid teha veidi mahukama programmi ja õpiksid meeskonnas programmeerimist. Rühmatöö käigus õpiksid nad tõenäoliselt programmeerimist kiiremini (Hannay et al., 2009) ja lisaks omandaksid ka sotsiaalseid oskusi

(Sancho-Thomas et al., 2009), mis annavad ainult lisakasu juurde. Kuid kuna kursus on mõeldud sissejuhatusena programmeerimisse, selles osalevad väga erineva taustaga inimesed, see on vabatahtlik ja tehakse tüüpiliselt muu õppimise või töötamise kõrvalt, siis on oht, et sundimine viib motivatsiooni alla ja lisaks võtab ära soovi jätkata informaatika valdkonnas, nagu oli leitud Černochová, Selcuki ja Černý (2020) uuringutes.

Projektist arvestatud hinde saamiseks pidi töö vastama etteantud nõuetele (vt alapeatükk Projekti kirjeldus). Joonisel 2 on näidatud õpilaste tööde vastavus nõuetele ja on toodud täpsemalt, mis hulgal ja mis nõuded olid täitmata.

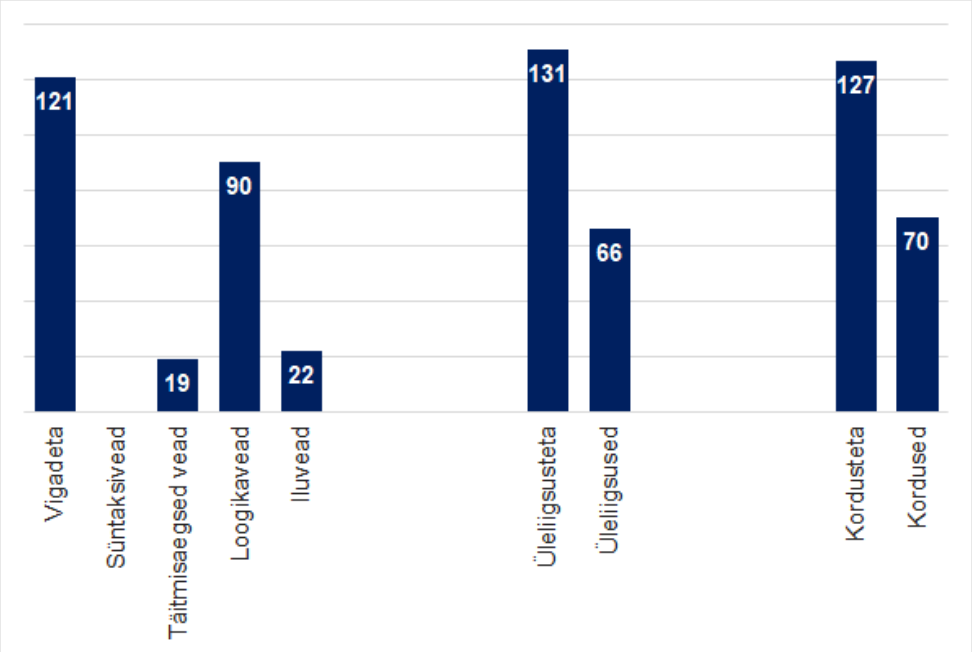


Joonis 2. Õpilaste projektide vastavus etteantud nõuetele ja puuduvad nõuded.

Valdavas enamuses, 85% programmides oli selle punkti täitmisega hästi hakkama saadud. Puudujäägid esinesid 29 programmis, kus puuduvaid nõudeid oli kokku 47 ehk mõnes programmis ei olnud mitut vajalikku osa. Arvatavasti nõuetesse vähesest süvenemisest olid nendes projektides peamiselt puudu kõige lihtsamad kriteeriumid, nagu kommentaarina programmi kirjutatud ülesande püstitus ja autori või autorite nimed. Kuues programmis olid puudu kommentaarid ja viies järjend. See on juba suurem viga, mis näitab õpilaste hajameelsust nõuete järgimisel, unustamist või oskamatust. Vaid viiel olid puudu muud kriteeriumid, nii et üldiselt võib öelda, et õpilased jälgisid tähelepanelikult ettekirjutatud tingimusi.

4.2 Vead, üleliigsused ning kordused

Järgmisena otsiti, kas programmides leidub vigu ja kui leidub, siis mis sorti need on. Vaadati ka seda, kas programmides on mõni asi üleliigne ja kas mõni koodilõik kordub. Joonisel 3 on esitatud projektides esinenud nelja erinevat sorti vead, üleliigsused ja kordused.



Joonis 3. Projektis esinenud vead, üleliigsused ja kordused.

61% programmidest toimisid täiesti korrektselt. Osalejate vähese kogemuse tõttu olid vead aga paratamatud, mistõttu täpselt 76 programmis leidus vigu. Nendest neljas olid nii rängad vead, et said hinnangu "ei tööta". 15 programm sisaldasid endas suuri vigu, kuid töötasid siiski teatud juhtudel. Tervelt 57 küll töötasid, kuid ikkagi pisikeste vigadega. Ette tuli põhiliselt kolme sorti vigu, neid oli kokku 131. Järelikult 55 projektis esines rohkem kui üks viga. Süntaksivead moodustasid koguarvust 0%, täitmisaegsed vead 14,5% ja loogikavead tervelt 68,7%. Autor lisas omalt poolt eraldi kategooria niinimetatud iluvigade jaoks, kuhu liigitas vead, mis raskendasid programmist arusaamist, ei mõjunud koodi puhtuse seisukohalt hästi ja mida õppurid kergesti parandada saaksid. Iluvead moodustasid vigade koguarvust 16,8%. Lisamärkusena tuleb meeles pidada, et õpetajad ja mentorid, kes kursusel osalejate projekte hindasid, võisid osasid vigu (nt iluvigu) üldse mitte või vähem olulisemaks lugeda kui lõputöö autor.

Süntaksivigu projektides ei esinenud, mis näitab, et kõik esitatud programmid kompileerusid ja läksid tööle. Täitmisaegsete vigadena (vt lisa II) leidis autor näiteks defineerimata muutuja, ebakorrektse *randint*'i kasutamise, tühja faili puhul kokkujooksmise. Peamiselt esines eri sorti

loogikavigu (vt lisa III), millest enim jäid silma probleemsed tsüklite, funktsioonide ja tingimuslausete kasutamised. Iluvigadena (vt lisa IV) tulid esile ebaselged teksti väljastamised, kirjavead, pidevad vahetused kasutajaliidese ja käsurea vahel.

Vead võisid tuleneda sellest, et õpilastel polnud etteantud ülesannet, kus oleks välja toodud, mida programm täpselt tegema peab. Seetõttu õpilased lihtsalt ei pannud tähele, et kõik ei toimi päris nii nagu peaks. Või nad küll teadsid, et nende programmis on vead, kuid kogenematusest ei osanud neid parandada. Sellest saab järeldada, et õpilased ei silunud eriti oma koodi ega testinud piisavalt. Kuid nagu Fitzgerald jt (2008) leidsid, võib silumine osutuda algajatele liiga keeruliseks. Testimise valdkond, sealhulgas silumine, on aga oluline programmeerimise osa ja mida varem selle poolega tegelema hakata, seda käepärasemaks ja selgemaks see saab. Seetõttu autor siiski arvab, et kursuse õpetajad-mentorid võiksid õpilastele kohe alguses näidata silumise võimalust, selgitada veateateid ja õpetada testimist. Ühtlasi tuleb selgeks teha, et mõistlik on pärast iga väikese koodilõigu kirjutamist testida, sest nii on kõige lihtsam vigu avastada ja kõige mugavam neid parandada (Annamaa, s.a). Veel ühe võimalusena saavad kursuse korraldajad osalejatele õpetada funktsionaalsuste ise sõnastamist, et nad paremini mõistaksid, kuidas programmi panna tegema seda, mida soovitakse.

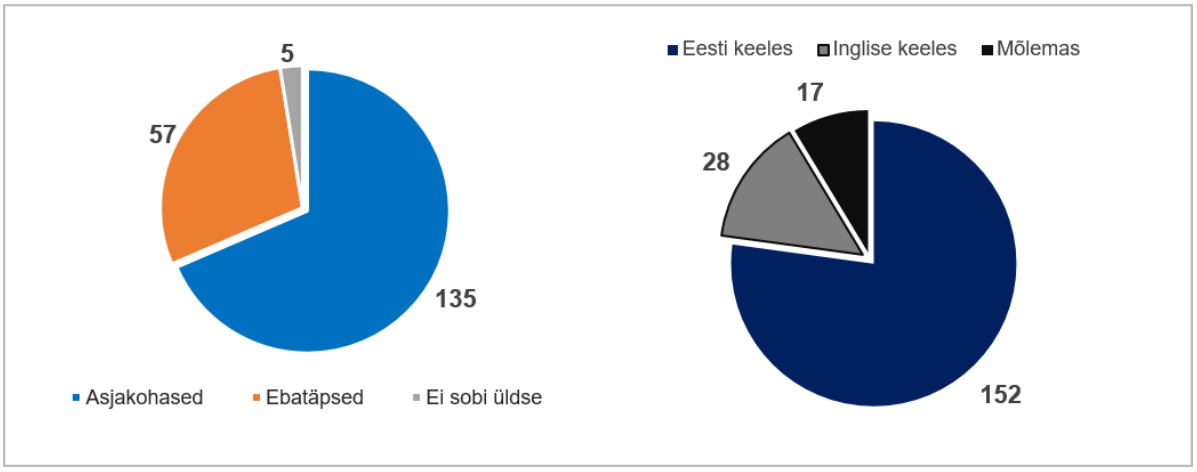
Koodis vaadati veel üleliigseid asju ja kordusi. Autori arvates pingutasid õppurid kohati üle ja tahtsid programmi pikemaks teha, lisades sinna tegelikult üleliigseid asju juurde. Kuid asju, mida ei kasutata, ei ole vaja. Enim tulid esile järgmised liigsused: üherealised oma ülesannet mittetäitvad funktsioonid teiste kasulike funktsioonide kõrval, kasutamata muutujate defineerimine, tsüklile järgnev samasuguse tingimusega *if*-lause ja muud taolised. Üleliigsustega programme oli 34% kogu projektidest. Programmid, mis sisaldavad kõike vajalikku ega midagi rohkemat, on kõige paremini loetavad ja arusaadavamad nii endale kui teistele. Korraldajad võiksid püüelda selle poole, et näiteks teeksid õpikusse koodi disaini alase peatüki, kus näitavad vahet programmide vahel, mis on küll ilma vigadeta, kuid mõttetute liialdustega, ja mis koodi kujunduse poolest pole kõige loetavamad. Nii näeksid õpilased erinevust ja saaksid enda koodi kirjutamisel seda meeles pidada.

Autori üllatuseks tuli ette päris palju kordusi, mida oleks saanud funktsioonidena lihtsalt ja mugavalt vältida. Kordusi leidus peamiselt korduvate koodijuppide, kontrollide, mitmekordselt defineeritud muutujate näol. Siit võib välja lugeda, et õpilased pole hästi aru saanud funktsioo-

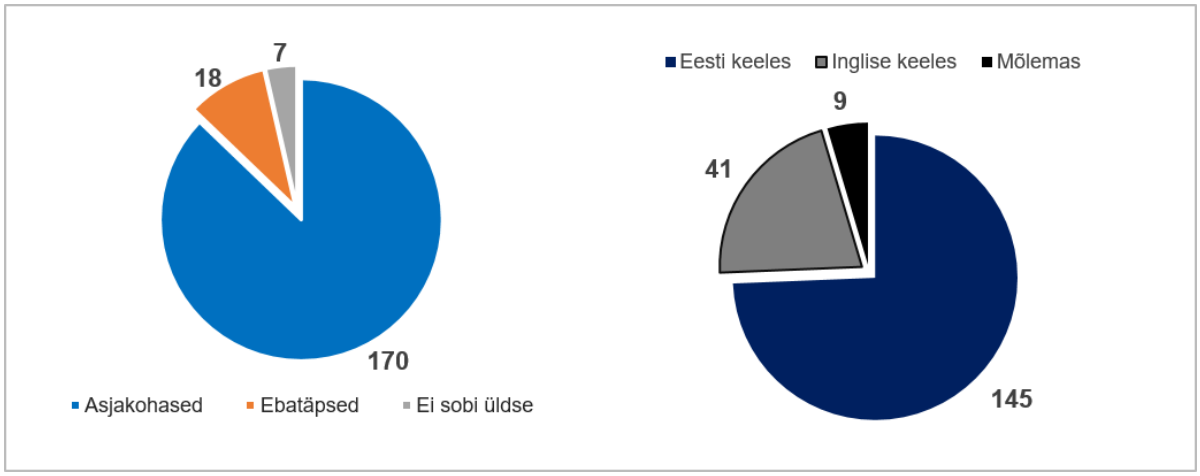
nide kasutamise põhimõttest, mis omakorda tähendab, et funktsioonide loogikat tuleb veel rohkem toonitada. Kordustega programme oli 36% projektide koguarvust. Tuleb mõista, et algajad õppurid ei oska veel programmi ülesehitusele nii suurt tähelepanu pöörata ja nende jaoks on olulisem, et programm töötaks. Siit järeldus, et juba varakult tuleks õpilastele rõhutada puhta koodi olulisust ja panna neid märkama koodi struktuurilisi nüansse. Korduste vigu saab samuti näidata disaini alases peatükis, mida varasemalt kursuse loojatele soovitati.

4.3 Muutujate ja funktsioonide nimed

Autor hindas muutujate ja funktsioonide nimede asjakohasust ning märkis üles nende keele. Joonisel 4 on esitatud muutujatele antud nimede sobivus ja keelevalikud. Joonisel 5 on sama informatsioon funktsiooni nimede kohta.



Joonis 4. Muutujate nimede asjakohasus ja keelevalikud.



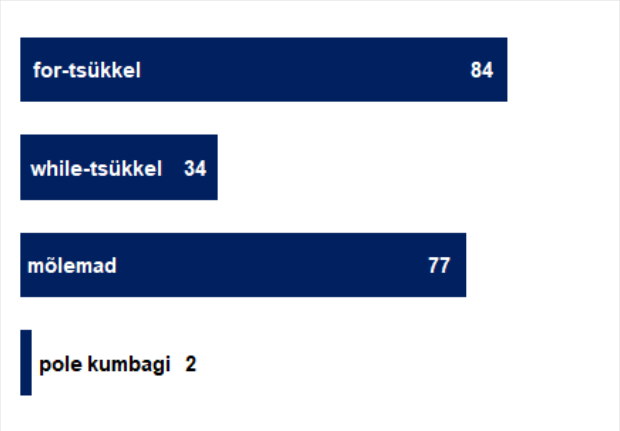
Joonis 5. Funktsioonidele antud nimede asjakohasus ja keelevalikud.

Nagu joonistelt 4 ja 5 on näha, oskasid õpilased funktsioonidele küllaltki täpseid ja häid nimesid panna, kuid muutujaid defineeriti tihti tähtede või õpilastele endile meelepäraste nimedega, millest programmi teistel lugejatel oli keeruline aru saada. Hinnangu "ebatäpne" said nimed, mis ei andnud oma kasutuspõhimõtet ja ideed edasi, nii et nendel nimedel pidi mõistmiseks pikemalt peatuma. Vaatamata sellele jäi kood nende puhul loetavaks, mistõttu pole need nimed liigitatud täiesti sobimatuteks. Sellised muutujatele antud nimed, mis polnud piisavalt head, tulid esile 29% projektides. Palju tuli ette korduvaid nimetusi, mis olid lihtsalt kas suurte tähtedega kirjutatud või numbriliselt eristatud. Funktsioonidele antud nimede sama näitaja oli vaid 9%. Samas funktsioonidele oli antud kaks sellist nime rohkem, mis kohe üldse ei sobinud ega andnud edasi, milleks seda funktsiooni kasutatakse. Hinnanguga "üldse ei sobi" märgiti nimed, mis näisid täiesti suvalised ja mis koodi loetavuse seisukohalt olid halvasti valitud. Näiteks tuli selles kategoorias ette samanimelisi, "funktsioon" nimelisi ja arusaamatute lühenditega funktsioone. Lisaks oli ühel osalejal funktsiooni nõue täitmata ja teisel õpilasel oli projekt lahendanud *lambda* funktsiooni kasutades, mistõttu tema funktsioonil puudus konkreetne nimi ja ka keel. Hea tava kohaselt võiksid nimed anda võimalikult täpselt edasi seda, millega nad on seotud või milleks neid kasutatakse. Seda võiksid kursuse korraldajad õppuritele kohe varakult üle seletada.

Keeleliselt eelistati ikkagi rohkem eesti keelt, täpsemalt 77% muutujatest ja 74% funktsioonidest olid eesti keeles. Leidus ka õpilasi, kes olid töö teinud inglise keeles. See on kasulik tuleviku tarbeks harjutamiseks, kuna valdavalt kirjutatakse koodi just inglise keeles, sõltumata sellest, kas töö käib Eesti või välismaa ettevõtetes. Mõned õpilased olid teinud programme kasutades nii eesti kui inglise keelt vaheldumisi. Vastavad näitajad oli järgmised: muutujate puhul kasutasid 9% eesti ja inglise keelt, funktsioonide puhul 5% õppuritest. Lõputöö autori hinnangul ei jätnud see head muljet, kuna polnud ühtsust ja koodi sisu hakkas seetõttu kannatama. Tuleks valida üks kindel keel, mille põhjal kõik muutujad ja funktsioonid defineerida.

4.4 Veel koodi struktuuri elemente

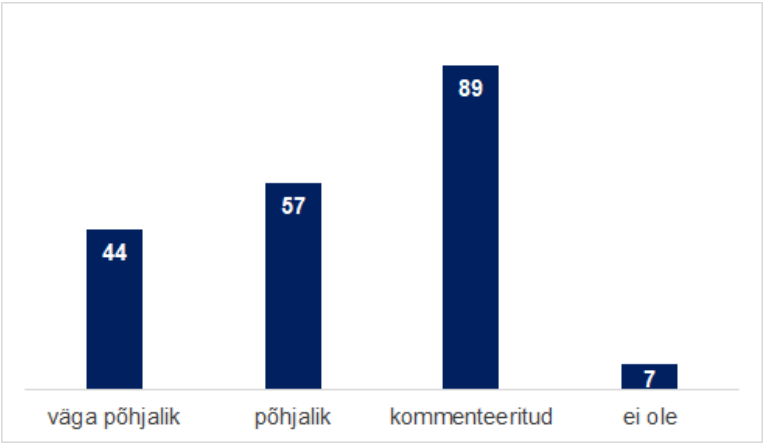
Autor võrdles *for*- ja *while*-tsükli kasutamist. Joonisel 6 on välja toodud kursusel osalejate projektid, mis sisaldasid ainult *for*-tsüklit, ainult *while*-lauset, mõlemat koos või mitte kumbagi.



Joonis 6. *For*-tsükli versus *while*-tsükli kasutamine digiprojektides.

Selgus, et 42% projektides tuli ette *for*-tsükkel ja 18% sisaldasid *while*-tsüklit. Niisiis kasutati tsükliteks üldiselt rohkem *for*-lauset. Osades programmides, täpsemalt 77, leidusid mõlemad tsüklid. Kumba tsüklit kasutati ei olnud seotud vaid isikliku eelistusega, vaid ka sellega, kumb koodi paremini sobis. Kõiki tegevusi ei saa näiteks *for*-lausega teha ja tahes-tahtmata tuleb *while*-lause kasutusele võtta. Aga on ka vastupidi, et näiteks failist lugemisel on *for*-lause efektiivsem. Nõuetes oli valik tegeleda failist lugemise või faili kirjutamisega ja see võis mõjutada *for*-tsüklite kasutamise suurt arvu. Samas on üldpildis ikkagi näha, et *for*-tsükkel kui lihtsam ja mugavam variant on eelistatum *while*-tsüklile, mis on iseloomu poolest paindlik aga tülikas.

Kommentaaride sügavust hinnati 3-punkti skaalal. Joonisel 7 on toodud õpilaste kirjutatud kommentaaride põhjalikkus.

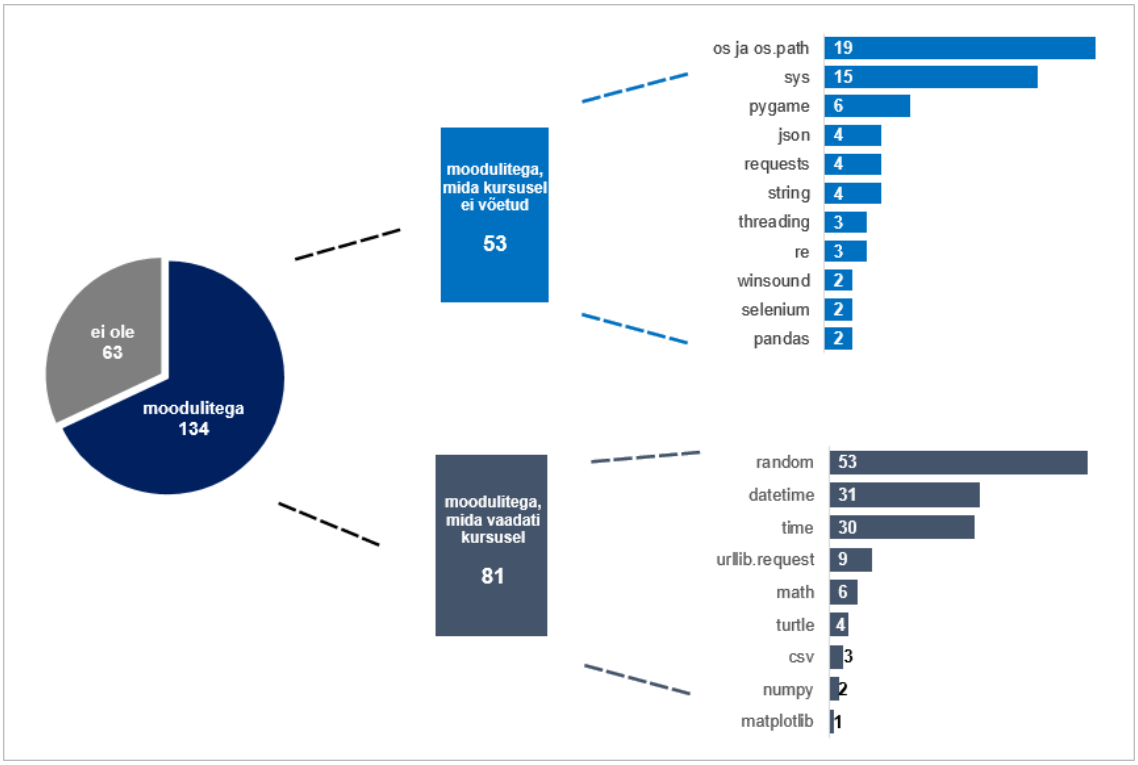


Joonis 7. Kursusel osalejate programmides esinenud kommentaaride põhjalikkus.

Siin oli väga suur varieeruvus, mis võis tulla sellest, et inimesed kirjutasid koodi rohkem enda jaoks ja ei mõelnud teistele lugejatele. Õpilaste vastustest ankeetküsimustikule tuli välja, et

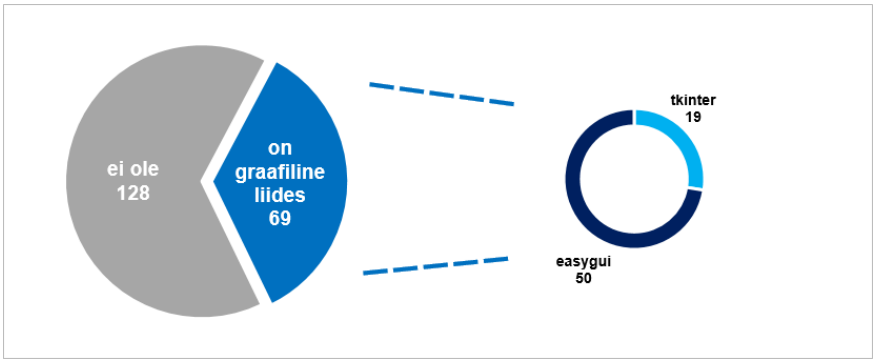
päris mitmed ei saanud aru, miks üldse kommenteerima peab või kuidas iga olulist tegevust kommenteerida. Projekti püstituses oli kirjas, et lahendus peab olema mõistlikult kommenteeritud, kuid polnud seletatud, mida "mõistlikult" tähendab ja see võiski põhjustada õppurites arusaamatusi. Väga põhjalikult olid kommenteeritud 22% programmidest. Põhjalikult kommenteeritud projektid moodustasid 29% ja kommenteeritud 45% projektide koguarvust. 4% oli kommenteerimise kas ära unustanud või meelega ära jätnud. Samas pole üleliia kommentaare vaja, sest muidu hakkavad need segama koodi üldilmet. Autori arvates olid kõige mõistlikumalt ja paremini kommenteeritud need tööd, mis said hinnangu "põhjalik". Eriti meeldisid tööd, kus olid kommentaarid iga koodisektsiooni juures selgelt struktureeritud ja ühtse disainiga, sest see jättis väga korrektse mulje. Silma jäid ka need tööd, mis olid teistest keerulisemad ja vajasidki arusaamiseks põhjalikemaid seletusi. Üldiselt oli kommentaare päris mõistlikult ja nende kallal väga kritiseerida ei saa.

Kuigi moodulite ja kasutajaliidese kasutamine polnud projekti juhendis nõutud, vaadati huvi eesmärgil, mis lisamooduleid kasutati ja kas graafilist liidest oli programmi kombineeritud. Joonisel 8 on näidatud, kui palju erinevaid mooduleid üldse kasutati ja mis mooduleid lisaks kursusel mainitutele õpilased veel kasutasid. Joonisel 9 on näidatud, kui palju kasutati graafilist liidest ja kumba varianti kursusel tutvustatutest eelistati.



Joonis 8. Moodulite kasutamine programmides.

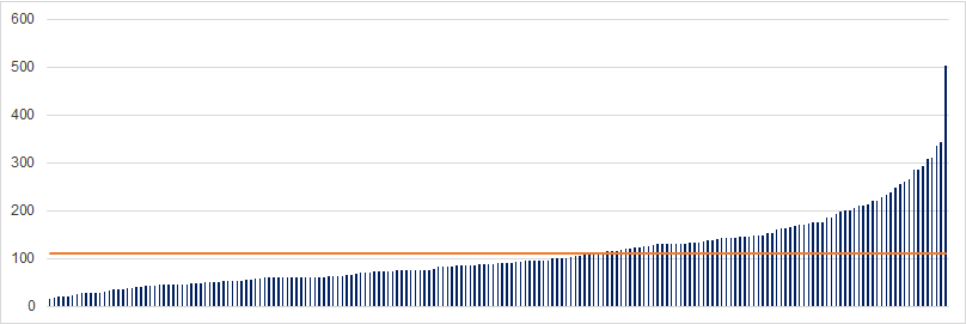
Nagu jooniselt 8 on näha, sisaldasid 68% programmidest ühte või enamat moodulit. Tuleb ka välja, et kasutati päris suurel hulgal mooduleid, millest kursusel juttu ei olnud. 53 programmis tulid ette taolised võõrad moodulid, sealjuures leidus 47 erinevat moodulit ja koguarv ulatus 100ni. Seda on hea tõdeda, sest järelikult uurisid õpilased juurde ega jäänud vaid kursuse piiridesse. Juurde otsitud moodulitest kasutati kõige enam järgnevaid: *os*, *os.path*, *sys*, *pygame*, *json*, *requests*, *string* ja teisi. Neist kolme esimest soovitatakse kursuse juhendajatel õppuritele kursuse vältel tutvustada, kuna neid kasutati päris suurel määral ja need annavad juurde võimalusi laiemate ideede elluviimiseks. 81 programmi sisaldasid mooduleid, mida kursusel tutvustati lühidalt või millest oli veidi juttu õpikus. Põhiliselt kasutati neid mooduleid: *random*, *datetime*, *time*. Tegemist on kõige lihtsamate ja üldistemate moodulitega, mida tavapäraselt tutvustataksegi programmeerimiskursuste algusfaasis. Autor arvab, et moodulite kasutamise võiks ka edaspidi jätta vabatahtlikuks, kuna kohustus lisamooduleid kaasata ajaks projekti keerulisemaks ja võib liiga suure surve peale panna. Selle tulemusel võib omakorda ideede väljamõtlemine muutuda veel raskemaks ja õpilaste motivatsioon kannatada.



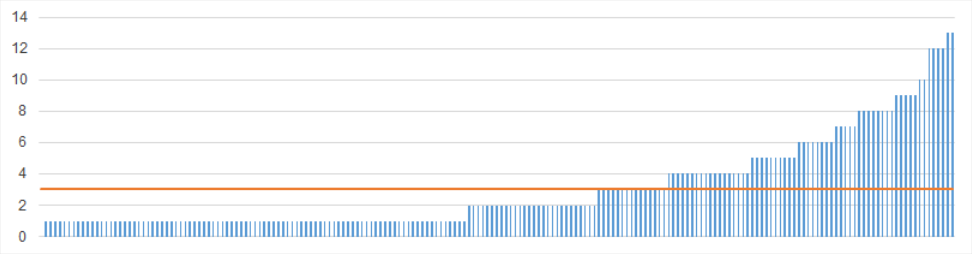
Joonis 9. Kursusel osalejate graafilise liidese kasutamine.

Graafilist liidest sisaldasid ligi 36% projektidest. See näitab, et õpilastel on huvi lisaks nõuete täitmisele arendada oma oskusi edasi ja katsetada ka visuaalidega. Üldiselt kasutatakse graafilist liidest arenduse poole peal päris palju, nii et see on hea kui juba varakult ka graafilist poolt tundma õpitakse. Nagu jooniselt 9 on näha, eelistati *easygui* kasutajaliidest tugevalt üle *tkinter'i*. Selle põhjuseks võis olla see, et *easygui* on kõige lihtsam kasutajaliides, mida Python pakub. *Tkinter'i* puhul tuleb selgeks teha rohkem komponente, näiteks aru saada raami ja tahvli tegemisest, kuid *easygui* puhul piisab põhiteadmistest. Õppuritele endale oleks hästi kasulik, kui kursuse läbiviijad muudaksid liidese kasutamise projekti juures kohustuslikuks. See õpetaks kasutajaliidese kasutamist kõigile, mitte vaid vabatahtlikele soovijatele.

Samuti uuriti, mitmest reast programmid koosnesid ja mitu funktsiooni keskmiselt programmides leidus. Siin olid suured erinevused, mistõttu on joonisel 10 ära toodud kõik võimalikud ridade arvu ja joonisel 11 funktsioonide arvu variandid.



Joonis 10. Ridade arv kursusel osalejate projektides.

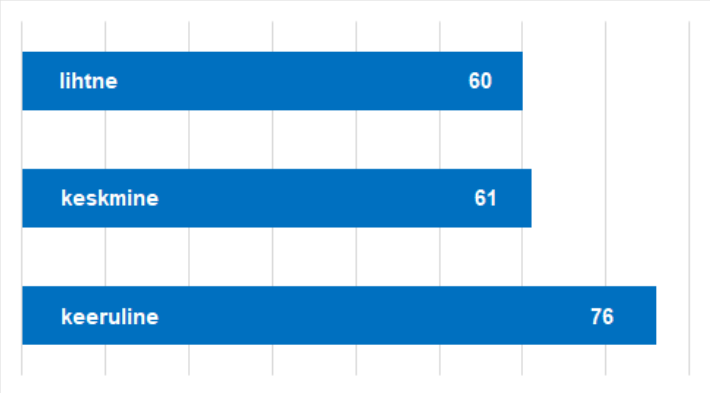


Joonis 11. Funktsioonide arv kursusel osalejate projektides.

Minimaalne ridade arv oli 14 ja maksimaalne 503. Keskmine ridade arv oli 109, joonisel 10 on see tähistatud oranži värviga. Funktsioonide puhul varieerus arv nullist kuni 13ni. Keskmine funktsioonide arv oli 3, mis on joonisel 11 tähistatud oranžiga.

4.5 Programmi keerukus ja mõistetavus teistele

Kõiki eelnevalt analüüsitud punkte arvesse võttes hindas autor programmi keerukust 3-palli süsteemis, kus hinnang 3 tähistas “keeruline sellise projekti jaoks” ja 1 tähistas “lihtne”. “Keeruline” tähendas, et projekt sisaldas palju elemente, mida kursusel ei õpitud või mida projekti juhendis polnud nõutud. “Keskmine” tähistas, et projekt oli sisukas ja sisaldas veidi rohkemat kui nõutud, näiteks graafilist liidest. “Lihtsana” märkis autor projektid, mis vastasid nõuetele, kuid olid teistega võrreldes lühemad ja väiksema funktsionaalsusega. Joonisel 12 on toodud programmi keerukuse hinnangud.



Joonis 12. Programmi keerukuse hinnang.

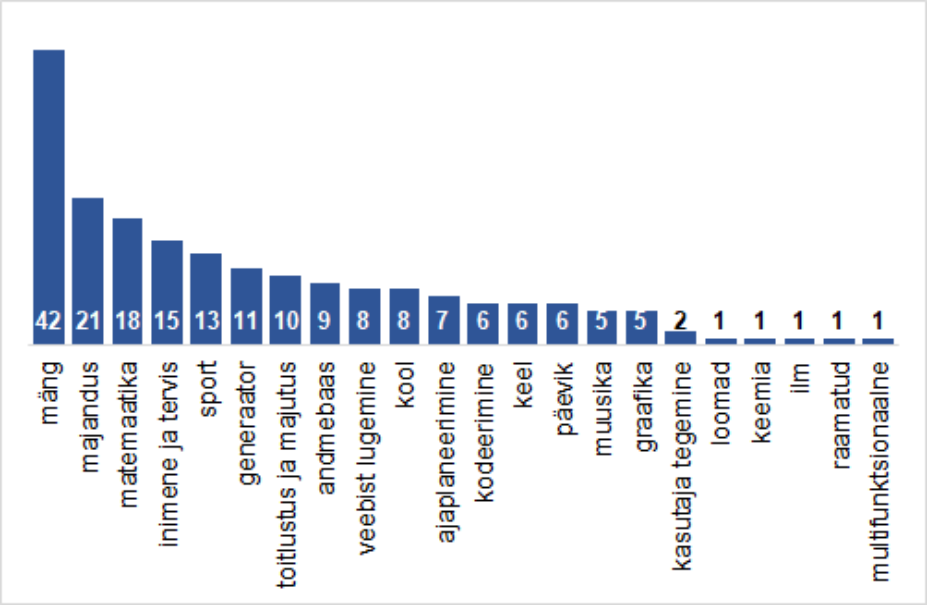
39% projektides tehti rohkemat, kui kursusel oli nõutud. Niimoodi said osalejad kinnistada kursusel õpitut ja omandada veel uusi teadmisi juurde. Nad said kogeda, et programmid võivad päris pikaks minna, võrreldes varasemalt tehtud lühikeste ülesannetega oli see märkimisväärne erinevus. Nende programmid sisaldasid keerulisemaid konstruktsioone, nagu näiteks klasside süsteemi, globaalseid muutujaid ja rohket funktsioonide kasutamist, koos põhiteadmistest väljas olevate moodulitega. Tõenäoliselt said need õppurid päris palju interneti abi kasutada, mis andis neile väärt kogemuse juurde, sest ka edasistes programmeerimise õpingutes tuleb tihti osata otsingumootoritest oma probleemidele lahendusi leida. Keerulistele programmidele järgnesid keskmise keerukusega projektid, mis moodustasid järgmise suure grupi (31%) ja tihedalt järgnesid lihtsad projektid (30%). Nende lihtsate seas oli palju ka nii-öelda lihtsama vastupanu teed minejaid, kes tegid ära vaid selle, mis neilt nõuti. See on aga vale suhtumine, sest algkursus on koht, kus tuleb kohe pingutada. Muidu ei jää värskelt õpitud teooria meelde ja jäävad lüngad.

Keerukuse juures edasi liikudes hinnati programmi mõistetavust teistele, et kas näiteks teine kursusel osaleja saaks programmist aru. Autor hindas, et 79% programmidest olid mõistetavad ja kenasti tehtud. Aga järelikult oli ikkagi 21% projektidest raskesti arusaadavad. Raskusi arusaamisest võisid valmistada arusaamatud muutujate nimed, halb koodi ülesehitus, vähesed kommentaarid või loogikavead. Ülesehituse poole pealt leidus palju programme, kus funktsioonid olid kuskil koodilõikude vahel ja mitte kõige ees, nagu hea kombe kohaselt peaks olema. Oli programme, mis sisaldasid häirivalt palju tühikuid või vastupidi, mis ei sisaldanud ühtegi ja kogu kood oli üksteise otsa kirjutatud. Keeruliste programmide juures puudusid seletavad kommentaarid, mistõttu võis mõte segaseks jääda. Siin saavad juhendajad teha juba eel-

nevalt mainitud asju, nagu näiteks õpetada koodi ülesehitust paremini jälgima, seletada millised nimed sobivad muutujatele ja funktsioonidele, panna õpilasi rohkem koodi testima ja sellise pilguga vaatama, et kui nad oleksid keegi teine, kes ei tea selle koodi loomisest midagi, kas nad saaksid sellest aru. Hea võimalus oleks peale tööde esitamist ja enne mentorite ranget ülevaatamist lasta kaasõpilasel koodi hinnata, et teine samal tasemel programmeerija oskaks vaadata, mis vigu tema leiab või kui hästi koodist aru saab. See õpetaks teistmoodi koodi vaatama ja lugema, pannes märkama asju, millele vaid enda koodi kirjutamisel üldse ei mõtlekski.

4.6 Teemavalik ja inspiratsioon

Autor uuris samuti, mis teemadel kursusel osalejad projekte koostasid. Joonisel 13 on toodud kõik võimalikud teemaplokid, millest osalejad programme koostasid. Märkusena toob autor välja, et mõned programmid oleksid sobinud mitme teemaploki alla, kuid lõpuks määrati need kategooriatesse selle alusel, millele programm peamiselt keskendub.



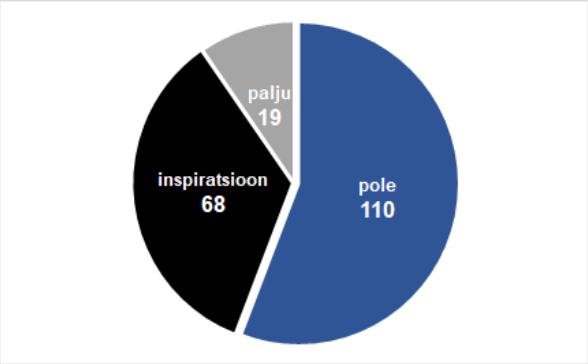
Joonis 13. Teemad, mille põhjal kursusel osalejad programme koostasid.

Enim kaldusid programmid olema mängutemaatilised. Jooniselt 13 on näha, et lausa 21% otsustasid selle valdkonna kasuks. Nagu Topalli ja Cagiltay (2018) ning Luxton-Reilly jt (2018) andmetel välja tuli, on mängude koostamine programmeerimise õpingute algusfaasis üks parimaid variante, sest need õpetavad nägema terve süsteemi toimimist, sealjuures kuidas erinevad süsteemi arendamise etapid moodustavad lõpuks terviku. Peale selle tõstavad mängud enamasti õpilaste motivatsiooni pingutada (Topalli & Cagiltay, 2018; Luxton-Reilly et al., 2018). Män-

gudele järgnesid majanduse, matemaatika, inimese ja tervise ning spordi teemalised programmid. Põnevaid programme tehti veel teemadel nagu generaator, toitlustus ja majutus, andmebaasid, kool, keel, päevik ja paljudel muudel.

Konkreetselt ei saa öelda, miks just sellised valikud tehti. Võib oletada, et kõige populaarsemate teemaplokkide juures sai otsustavaks võimalus tegeleda arvudega ja teha tehteid. Palju tehti elulisi programme, mis aitaksid lihtsustada mõnda igapäevast toimetust. See oli hea valik, kuna IT-alastel töökohtadel käib samuti töö eluliste probleemide ümber (Konecki et al., 2016). Kindlasti tehti valik ka vastavalt sellele, mis ennast huvitab ja motiveeriks projektiga vaeva nägema.

Eelnevalt analüüsitud punkte ja teemavalikut kokku võttes saab projekte võrrelda ka varasemalt lahendatud ülesannete ja õpikus toodud näidisülesannetega. Joonisel 14 on esitatud ülevõtmised kursusel tehtud või näidatud ülesannetest.



Joonis 14. Inspiratsioon kursusel lahendatud või õpikus näidatud ülesannetest.

Jooniselt 14 on näha, et näidisülesannetelt saadi vähemal või rohkemal määral inspiratsiooni. 10% programmidest koosnesid paljudest ülevõtmistest, nii et nende koodid ja teemavalikud olid kursuse ülesannetega ligilähedased. Enim ülevõtmisi oli siinjuures õpikust peatüki “Faili kirjutamine“ juures näidisena toodud programmist “Šifreeritud kirjutamine”, õpikust peatükis “Tsüklite rakendamine” näidatud programmist “Arvamismäng” ja osalejate läbilahendatud ülesandest “2.1 Pilved”. “Šifreeritud kirjutamisest” oli konkreetselt üle võetud šifreerimise funktsiooni loogika. Ülesannetest “Arvamismäng” ja “Pilved” oli üle võetud see, et vastavalt sisestatud arvule väljastatakse, kas see on suurem või väiksem tingimusest. 35% said põhiliselt lihtsalt inspiratsiooni ehk nende programmid olid sarnased, kuid ka piisavalt erinevad. Kõige roh-

kem sarnasusi siit kategooriast leidus kursusel läbilahendatud programmiga nimega “2.2 Sisseastumine”. Sellest ülesandest võeti üle loogika, et on palju lihtsaid tingimuslauseid, mille põhjal väljastatakse mingi lause. 56% programmidest olid puhas omalooming ja ei näinud ühegi näidiskoodiga sarnased välja.

Ülevõtmisi ei saa vältida, kuna programmeerimisega alles alustavad õpilased ei oska veel ise häid programme välja mõelda ega ole endas nii kindlad, et totaaselt midagi uut luua. Cliburni jt (2010) väitsid samuti, et algajad tahavad etteantud ülesandeid. See on parem, kui õpilased pigem teevad midagi, mis on juba sarnasel kujul tehtud ja kinnistavad oma teadmisi. Siinkohal ei saa kursuse läbiviijatele etteheiteid teha ja raske on ka soovitusi anda, kuna näidisülesannete etteandmine on vajalik ja nendelt maha tegemist ei ole võimalik keelata. Ka tuleb aru saada, et palju õpilased õpivad näidiste järgi ja omandavad programmeerimisoskused just nende põhjal.

4.7 Tagasiside küsimustiku vastused

Kursusel osalejatel lasti peale projektide esitamist vastata tagasiside vormile, kus nad märkisid kaua projekti tegemine aega võttis, millised olid täpsed töö tegemise etapid, mis oskustest või teadmistest jäi vajaka ja kas projekt sobis nende arvates kursusele või mitte. Nagu eelnevalt ka välja toodud, vastas ankeetküsitlusele 219 õpilast ehk kuus õpilast jätsid vastamata.

Õpilastel kulus keskmiselt töö tegemiseks 8,14 tundi, mis on ligilähedane nõuetes ära toodud arvuga. Kuid siin tuleb arvesse võtta, et üks õpilane oli pannud enda tundide arvuks 200. Kui seda mitte arvestada, tuleb keskmine hoopis 7,2 tunni ringis.

Autor otsis seost rühmatööna ja individuaalselt tehtud tööde tundide arvu, ridade arvu ja keerukuse vahel, sest Hannay jt (2009) leidude kohaselt peaks grupitööd olema kiiremini, kvaliteetsemalt ja samal ajal mahukamalt tehtud. Nende näitajate seost kontrolliti Kruskal-Wallise testiga, mille tulemusena selgus, et statistiline erinevus leidus ainult keerukuse näitaja puhul (hii-ruut = 7,7599, $p < 0,05$). Kasutades Mann-Whitney U testi leiti statistiline erinevus individuaalse ja paaristöö ($U = 2520$, $p < 0,05$) ning kahe- ja kolmeliikmeliste rühmade vahel ($U = 63$, $p < 0,05$). Ridade arvu ja tundide arvu näitajate vahel statistilist erinevust ei tuvastatud (hii-ruut vastavalt 3,9746 ja 1,1693, $p > 0,05$). Tabelis 1 on toodud individuaalse, kaheliikmelise ja kolmeliikmelise rühma programmi keerukuse, ridade arvu ja tööks kulunud keskmise tundide arvu võrdlus.

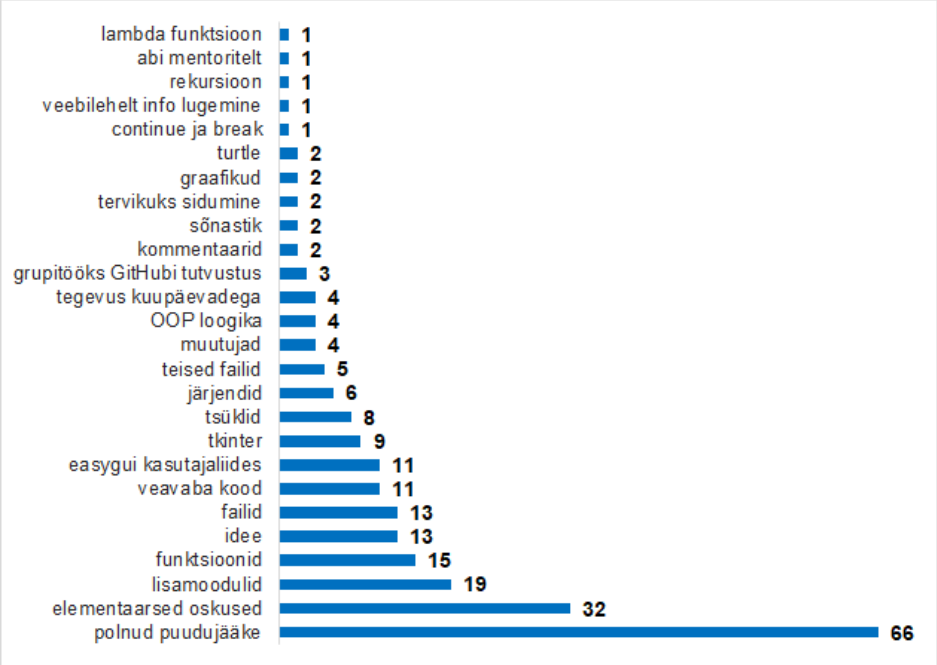
Selle kursuse andmete põhjal tuli välja, et kolmeliikmelistes rühmades tehtud tööd olid individuaalsete ja ka paaristöödega võrreldes keskmiselt keerulisemad, sisaldasid endas rohkem koodiridu ja nende peale kulus indiviidi kohta ligikaudu sama palju aega. Tulemus vastab eelnevalt välja toodud teooriale. Samas oli kolmeliikmelisi rühmi päris väike hulk, mistõttu võivad rohkemate andmetega olla tulemused erinevad. Paaristöid individuaalsetega kõrvutades kulus küll keskmiselt vähem aega, kuid keskmine ridade arv ja keerukus olid hoopis väiksemad.

Tabel 1. Individuaalse, kaheliikmelise ja kolmeliikmelise rühma võrdlus.

	Programmide arv	Programmi keerukus		Programmi ridade arv		Õpilaste arv	Keskmine tundide arv	
		Keskmine	Standardhälve	Keskmine	Standardhälve		Keskmine	Standardhälve
Individuaalne töö	170	2,11	0,82	111,86	75,89	170	7,49	5,23
Kaheliikmeline rühm	24	1,75	0,85	84,38	47,34	46	6,29	2,77
Kolmeliikmeline rühm	3	3	0	153,33	63,61	9	7,44	3,24

Töö tegemise etapid olid suures pildis sarnased. Alustati programmi idee välja mõtlemisega, millele järgnes töö planeerimine. Kui need sammud olid paika pandud, siis liiguti põhilise tegevuse ehk koodi kirjutamise juurde. Vaid väike osa mainis koodi katsetamist, testimist ja vigade parandamist. Niisiis tundub, et sellele ei pandud nii suurt rõhku või lihtsalt ei pandud seda eraldi kirja.

Õpilastelt küsiti veel, et millistest teadmistest või oskustest tunti puudust. Joonisel 15 on toodud oskused ja teadmised, mille järele tundsid õpilased endi sõnul töö käigus vajadust.



Joonis 15. Oskused ja/või teadmised, millest kursusel osalejate sõnul oli vajaka.

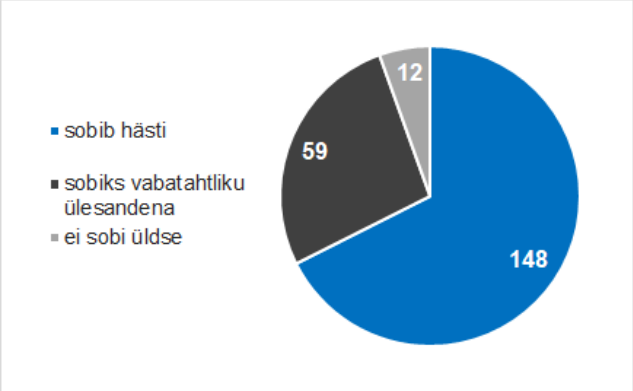
28% õppuritest tundsid, et neil ei jäänud ühestki oskusest ega teadmisest puudu. See on ühtpidi tore, kuid teistpidi näitab, et need õpilased võibolla ei pingutanud oma võimetele vastavalt ja oleksid saanud keerukamaid programme teha, kus oleks ka lisamaterjali tarvis läinud. Siin võib ka kahelda, sest äkki nad otsisid lisamaterjali, kuid lihtsalt ei kaasanud seda oma programmidesse. Üldiselt oli väga erinevaid vastuseid, aga põhiprobleemiks olid vähesed või puudulikud elementaarsed ja tehnilised oskused, millel võiks õpilaste meelest kas kauem peatuda või millest rohkem rääkida. Tehnilistest oskustest nimetati järgnevaid asju, mille tundmisest jäi puudu: lisamoodulite info, tekstifailidega toimetamised, tekstifailide asemel teiste failidega toimetamine, funktsioonide töö põhimõte, veavaba koodi kirjutamine, *tkinter*'i või *easygui* kasutamine ja muud sarnased oskused. Nii mõnedki õpilased olid enda suhtes kriitilised ja tundsid, et hetkel teevad nad lihtsaid asju palju keerulisemalt ning tahavad õppida efektiivsemat koodi kirjutamist. Nad mõistavad, et on programmeerimise õpingutel alles alguses ja näevad, et arenguruumi on kõvasti. Nendeks õpilasteks on need 32, kes mainisid, et elementaarsetest oskustest jääb puudu. Kahtlemata võib tunduda, et äkki nimetatud õppurid hoopis ei pingutanud piisavalt ja ei tahtnud eriti õppida, kuid nende tagasisidest jäi ikkagi mulje, et nad juba ootavad, et nende oskused kogemustega paraneksid. See suurendab autori arvates usku nendesse õpilastesse, kuna nad ei tee tööd vaid ühe projekti jaoks, vaid enda arengule mõeldes. Kursuse õpetajad-mentorid võiksid tunnustada õpilasi ja parimaid programme esile tuua, sest see kindlasti innustaks õpinguid jätkama.

Teise suure murekohana leidsid õppurid, et kursuse alguses õpitud teadmised olid vahepeal meelest läinud, mistõttu kulus palju aega uuesti meelde tuletamisele. Selle probleemi juures ei saa kursuse korraldajad väga midagi lisaks teha, kuna projekt ongi mõeldud õpitut meelde tuletama ja kinnistama. Tundub, et õpilased pidasid õpiku kasutamist halvaks märgiks, kuid tegelikult pole selles midagi halba, kui lugeda materjali töö tegemise ajal üle. Kui materjalidest ei leia mõnele küsimusele vastust, on alati võimalik *Google'*st otsida, mida paljud tagasisides kirjutasid, et ka tegid. Kahjuks jäi õpilaste vastustest küsitlusele mulje, et vähesed kasutasid võimalust küsida abi mentoritelt, mis oleks olnud üks kiiremaid ja parimaid lahendusi.

Veel kujunes probleemseks kohaks projekti idee väljamõtlemine. Oldi liiga ambitsioonikad ja taheti luua programme, mis olid üle oma võimete. See on küll tore, et õpilased tahavad rohkemat teha, kuid nad ei mõelnud sellele, et on alles esimesel programmeerimise kursusel. Niisiis kursuse kavast kõvasti keerulisemate programmidega kaugele ei jõutud. Motivatsioon seetõttu langes ja õpilased pidid ennast kokku võtma, et mõne uue hea idee peale tulla ja projekt lõpule viia. Siinkohal oleksid õpilased võinud tuge otsida mentoritelt, kes oma teadmiste ja kogemustega oleksid saanud nõu anda ning õigele teele suunata. Kursuse korraldajad küll julgustavad praegugi õppureid iga väiksema mure või probleemiga abi küsima, kuid nad ikkagi ei julge. Kuidagi peaks õpilastesse sisendama, et eksimine on inimlik ja abi küsimine ei näita mitte rumalust, vaid just soovi targemaks saada. Mentorid saaksid endast jätta võimalikult sõbraliku mulje, et õpilased tunneksid nagu nad küsiksid sõbralt nõu ja ei pelgaks suhelda.

Autor tõi välja, et taheti teha liiga suuri projekte oma oskuste kohta, kuid sama probleemi teine külg oli see, et õpilastel ei tulnud üldse ideid, millest projekti teha. Paljud kurtsid, et hästi palju aega kulus loomeprotsessi peale. Üks õppur tõi välja mõtte, et võiks teha juba enne projekti mõned ülesanded nii, et ise peab juhised välja mõtlema. See annaks aimu, kuidas oma koodile tähendus anda ja selle põhjal ülesanne sõnastada. On ka selline variant, et kursuse korraldajad võiksid võimalusel näiteks mõned teemaplokid ette anda, mis lihtsustaksid ja aitaksid õpilastel seda õiget teemat valida, sest esmakordselt sellist ülesannet teha pole tõesti lihtne.

Lõpetuseks paluti õpilastel hinnata, kui hästi nende arvates projekt sobib antud kursusesse. Joonisel 16 on näidatud õpilaste hinnangud projekti sobivusele sellesse kursusesse.



Joonis 16. Kursusel osalejate hinnangud projekti sobivusele antud kursusele.

Enamus õpilasi (68%) olid seda meelt, et projekt sobib kenasti ja on väga väärtuslik. Kuid oli ka palju arvamusi, et projekt võiks olla vabatahtlik ülesanne (27%) või selle võiks üldse kursuselt välja jätta (5%). Need arvamused panevad mõtlema, sest projekt on eeskätt loodud selleks, et õpilased saaksid oma teadmisi kinnistada, üle korrata ja asju juurde õppida. Kui osad leiavad, et projekti pole vaja, siis tekib küsimus, et kas nad andsid alla, neil polnud vajaminevaid oskusi, neil ei olnud teooria nii selge või nad oleks soovinud lihtsalt suuremat toetust juhendajatelt. Kursuse mentorid võiksid lõputöö autori arvates pakkuda rohkem omapoolset abi välja, et õpilased oleksid motiveeritud projekti tegema ja sinna oma aega panustama. Nagu eelnevalt ka mainitud, võiks anda ette mõningad näidisteemasid või kuidagi suunata, et õpilased saaksid sealt mugavalt ise edasi minna. Kui kursusel osalejad plaanivad informaatika õpingutega jätkata, on neile igatpidi oluline projekte koostada, sest peamiselt käibki IT alal töö iseloodud projektide kallal.

Kokkuvõte

Tehnoloogia arengust ja nii-öelda nutistumisest tingituna on veebikursused muutunud aastatega aina populaarsemaks. Teine valdkond, mis iga aastaga veel populaarsust juurde kogub, on informaatika ja sealjuures programmeerimine. See on viinud programmeerimise algkursuste arvu suurenemiseni ja ka nende täiustumisele.

Käesolevas bakalaureusetöös anti ülevaade Tartu Ülikooli informaatika didaktika töörühma korraldatud programmeerimiskursusest "Tehnoloogia tarbijast loojaks". Töö eesmärgiks oli välja selgitada mainitud kursuse vältel tehtavate projektide tugevad ja nõrgad küljed ning anda omapoolseid soovitusi kursuse läbiviijatele. Andmete kogumiseks vaadati läbi 197 projekti, millest kirjutati erinevad tähelepanekud andmetabelisse. Lisaks koguti andmeid ka korraldajate koostatud ankeetküsitlusega. Eesmärgist lähtuvalt esitati kolm uurimisküsimust.

Esimene uurimisküsimus oli, millised olid õpilaste struktuurilised eelistused projekti koodi kirjutamisel. Selgus, et koodi eelistati kirjutada eestikeelsete muutujate ja funktsioonide nimedega. Õppurid kasutasid rohkem *for*-tsüklit kui *while*-lauset. Oma koodi kommenteeriti piisavalt, kuid mitte ülemäära palju. Kasutati heal määral kursusel tutvustatud ja ühtlasi kursuse kavas mitteleiduvaid mooduleid. Graafilistest liidestest eelistati kasutada *easygui*'d üle *tkinter*'i. Ridu oli programmides keskmisel 109 ja funktsioone 3. Projekte koostati enim mängudena, kuid populaarsed teemad olid ka päriselul põhinevad, näiteks majanduse ja matemaatika valdkonnaga seotud ülesanded. Programmid olid suurel määral võõrale vaatajale arusaadavalt tehtud, kuid siiski mõningate probleemidega, millele järgmine uurimisküsimus vastuse toob.

Teine uurimisküsimus otsis vastust, millised olid õpilaste projektide peamised nõrgad kohad. Esmalt selgus, et veerand õpilastest jätsid ühe või enama etteantud tingimuse täitmata. Tuli ka välja, et õppurid tegid algkursusel oma programmides veel päris palju vigu. Esines kordusi ja üleliigsuseid, mida ei osatud likvideerida. Kõik muutujate ja funktsioonide nimed polnud täiesti asjakohased. Varasemalt läbi lahendatud ülesannetest ja õpikus toodud näidisülesannetest saadi olulisemal määral inspiratsiooni ehk veel ei osatud ise nullist projekti koostada.

Kolmas uurimisküsimus uuris, millise hinnangu andsid kursusel osalejad projekti tegemise etapile. Enamus õpilasi jäi projektiga väga rahule ja leidsid, et see sobib kursusesse hästi. Keskmine projektile kulutatud aeg oli ligilähedane tingimustes välja toodud arvuga. Töö tegemise

etapid olid õppuritel sarnased. Suurem osa tundis, et neil ei jäänud ühestki oskusest või teadmisest puudu. Need, kellele aga jäi, tundsid enim puudust elementaarsetest oskustest, et teha mahukamaid ja puhtamaid programme.

Koostatud statistika põhjal pandi kirja soovitusi materjalide parendamise ja abipakkuvama juhendamise osas. Tulemused näitasid, et õpilased tegid lihtsaid struktuurilisi eksimusi, mida võiks aidata ennetada peatükk materjalides, mis keskendub puhta ja hea ülesehitusega koodi kirjutamisele. Pakuti välja ka idee, et õpilased võiksid saada võimaluse enne mentorite hindamist ise kaasõppurite programmidest vigu otsida. Veel soovitati mentorite poolt pakkuda rohkem tuge, et iga väiksemagi küsimuse korral leiaks kohest abi ja teooria teadmises ei jääks auke.

Kursuse korraldajad saavad antud töö põhjal kursust täiendada ja paremaks muuta. Ennekõike saavad nad rohkem tähelepanu pöörata hädas olevate õpilaste aitamisele ja õpilastes programmeerimise vastu veel suurema huvi tekitamisele. Tööd saab kasutada täiendavate uurimuste tegemisel, keskendudes rohkem sellele, et kas kursuselt saadud kogemus motiveerib õpilasi programmeerimisega jätkama.

Viidatud kirjandus

Annamaa, A. (s.a). *Programmeerimise õpik*. Tartu Ülikooli informaatika eriala õpik. Vaadatud 02.05.2021 https://progeopik.cs.ut.ee/01_sissejuhatus.html

Černochová, M., Selcuk, H., & Černý O. (2020). Factors Influencing Lower Secondary School Pupils' Success in Programming Projects in Scratch, *Informatics in Schools. Engaging Learners in Computational Thinking. ISSEP 2020*. Tallinn, 119-129. doi: 10.1007/978-3-030-63212-0_10

Cliburn, D. C., Miller, S. M., & Bowring, E. (2010). Student preferences between open-ended and structured game assignments in CS1, *2010 IEEE Frontiers in Education Conference (FIE)*. Washington DC, F2H-1-F2H-5, doi: 10.1109/FIE.2010.5673668

Fitzgerald, S., Lewandowski, G., McCauley, R., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers, *Computer Science Education, 18(2),* 93-116. doi: 10.1080/08993400802114508

Ghobadi, S., Campbell, J., & Clegg, S. (2017). Pair programming teams and high-quality knowledge sharing: a comparative study of cooperative reward structures, *Information Systems Frontiers, 19(2),* 397-409. doi: 10.1007/s10796-015-9603-0

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

Hristova, M., Misra, A., Rutter, M., & Mercuri, R. (2003). Identifying and correcting Java programming errors for introductory computer science students, *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '03). Association for Computing Machinery*. New York, USA, 153-156. doi: 10.1145/611892.611956

Informaatika didaktika töörühm. (s.a). Tartu Ülikooli arvutiteaduste instituudi kodulehekülg. Vaadatud 02.05.2021 https://didaktika.cs.ut.ee/

Jazayeri, M. (2015). Combining Mastery Learning with Project-Based Learning in a First Programming Course: An Experience Report, *International Conference on Software Engineering*. Florence, Italy, 315-318. doi: 10.1109/ICSE.2015.163

Juurak, R. (2018). Millal saab informaatikast kohustuslik õppeaine?, *Õpetajate Leht 16.märtsil.* Vaadatud 02.05.2021 https://opleht.ee/2018/03/millal-saab-informaatikast-kohustuslik-oppeaine/

Ko, J. A., & Myers, A. B. (2005). A framework and methodology for studying the causes of software errors in programming systems, *Journal of Visual Languages and Computing, 16(1-2),* 41–84. doi: 10.1016/j.jvlc.2004.08.003

Konecki, M., Lovrenčić, S., & Kaniški, M. (2016). Using real projects as motivators in programming education, *39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO).* Opatija, 883-886. doi: 10.1109/MIPRO.2016.7522264

Kori, K., Beldman, P., Tõnisson, E., Luik, P., Suviste, R., Siiman, L., & Pedaste, M. (2019). *IT oskuste arendamine Eesti koolides*. Uuring. Vaadatud 02.05.2021 https://transferwise.com/documents/IT%20oskuste%20arendamine%20Eesti%20koolides.pdf

Luxton-Reilly, A., Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A., Ott, M. L., Paterson, J., Scott, A. M., Sheard, J., & Szabo, C. (2018). Introductory Programming: A Systematic Literature Review, *ITiCSE 2018 Companion: Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. *Association for Computing Machinery*. New York, USA, 55-106. doi: 10.1145/3293881.3295779

Monaghan, C., Bizumic, B., Reynolds, K., Smithson, M., Johns-Boast, L., & Van Rooy, C. (2014). Performance of student software development teams: the influence of personality and identifying as team members, *The European Journal of Engineering Education, 40(1),* 52-67. doi: 10.1080/03043797.2014.914156

Müller, M. M. (2007). Do programmer pairs make different mistakes than solo programmers, *Journal of Systems and Software, 80(9),* 1460-1471. doi: 10.1016/j.jss.2006.10.032

Puniste, S. (2015). *Eesti gümnaasiumides õpetatavad programmeerimiskursused*. Tartu Ülikooli arvutiteaduse instituudi bakalaureusetöö. Vaadatud 02.05.2021 https://dspace.ut.ee/bitstream/handle/10062/56092/thesis.pdf?sequence=1&isAllowed=y

Sancho-Thomas, P., Fuentes-Fernández, R., & Fernández-Manjón, B. (2009). Learning teamwork skills in university programming courses, *Computers & Education, 53(2),* 517-531. doi: 10.1016/j.compedu.2009.03.010

Topalli, D., & Cagiltay, N. E. (2018). Improving programming skills in engineering education through problem-based game projects with Scratch, *Computers & Education, 120,* 64-74. doi: 10.1016/j.compedu.2018.01.011

Tõnisson, E., Palts, T., Säde, M., Tõnisson, K. jt. (2019a). *Programmeerimine.* Informaatika valikkursus gümnaasiumile õpik. Vaadatud 02.05.2021 https://web.htk.tlu.ee/digitaru/programmeerimine/

Tõnisson, E., Palts, T., Tõnisson, K., Säde, M. jt. (2019b). *Tarkvaraarendus.* Informaatika valikkursus gümnaasiumile õpik. Vaadatud 02.05.2021 https://web.htk.tlu.ee/digitaru/tarkvara/

Vallistu, J., & Danilov, T. (2018). *Tööturg 2035. Tööturu tulevikusuunad ja -stsenaariumid*. Uuring. Vaadatud 02.05.2021 https://www.riigikogu.ee/wpcms/wp-content/uploads/2018/08/tooturg_2035_tooturu_tulevikusuunad_ja_stsenaariumid_A4_veeb.pdf

Lisad

I. Projektide andmetabel

Järgnevalt lingilt leiab arvestatud lõpptulemusega lahenduste analüüsi andmetabeli:

https://docs.google.com/spreadsheets/d/1-TifXkwm96gzWqhBqeLoin7Eyt8D6CgHiZ7Y7mmOHYk/edit#gid=0

II. Projektides esinenud täitmisaegsed vead

Järgnevas tabelis on välja toodud täitmisaegsed vead, mis õppurite programmides ette tulid.

Järjekorra-number	**Vea nimetus**	**Esine-misarv**
1.	Fail on tühi, faili ei kirjutata midagi või kirjutatakse valesti	4
2.	Ristist sulgemisel veateated	3
3.	Sellise nimega muutuja pole defineeritud	2
4.	*Randint* funktsioon valesti defineeritud	2
5.	Negatiivse arvuga jätkab kuni saab	2
6.	Küsitud muutuja ei sobi	1
7.	Faili asukoht väga spetsiifiline	1
8.	Järjendi indeks on väljaspool valikut (ingl. *list index out of range*)	1
9.	*Proxy*'ga seotud veateated programmi sulgemisel	1
10.	Pole täpsustatud, et valimiseks tuleb number sisestada	1
11.	Viskab kõik aknad korraga lahti ja ei lase kinni panna	1
	Kokku:	**19**

III. Projektides esinenud loogikavead

Järgnevas tabelis on välja toodud loogikavead, mis esinesid kursusel osalejate programmides.

Järjekorra-number	**Vea nimetus**	**Esine-misarv**
1.	Tsükli ebakorrektne kasutamine (tingimust ei vähendata või ei suurendata; tsüklist ei minda välja; *True/False* ei vahetata; kasutamise eesmärk pole õige)	12
2.	Funktsioonide vead (kaks ühe nimega funktsiooni; vale koodilõik funktsiooniks; vale kasutus-põhimõte; valesti väljakutsumine)	11
3.	Tingimuslause pole korrektne (kontroll ühe võrra nihkes; võrdlematute muutujate võrdlemine; sulgude vale asetus; lihtsalt valesti tehtud tingimus; tingimuslaused vales järjekor-ras; *if*, *elif* ja *if* järjest kasutamine vale loogika alusel)	10
4.	Arvud võivad negatiivsed olla (kuigi loogika alusel ei tohiks)	9
5.	Programmi ei saa kinni panna või see ei lõppe ära	7
6.	Akna suurust ei saa muuta ja kastid vales kohas (pole täielikult nähtavad või kirjutavad andmeid üle)	7
7.	Panustada/valida saab väljaspool valikut ehk puuduvad vajalikud tingimu-sed	6
8.	Järjendist valesti lugemine (vaid esimese elemendi kuvamine järjendist; järjendist info võtmine koha võrra nihkes; vale kasutuspõhimõte)	4
9.	Vale arvutuskäik	3
10.	Kuupäevaks võib panna ükskõik mis arvu, sõna või sümboli	3
11.	Skoor ei lähe vahepeal nulli või skoor kirjutatakse iga sekundi järel	2

12	Samanimelist asja juurde lisades ei lisa olemasolevale, vaid teeb uue välja	2
13.	Sulgude vale järjekord *round* funktsiooni kasutades	2
14.	Aken tuleb enne lahti kui oled jõudnud kõigile küsimustele vastata	2
15.	*Len* funktsiooniga vale sõnepikkuse võtmine	1
16.	Graafikul andmete üle kirjutamine või andmete puudumine	1
17.	Ülikoolis hinnatakse tähtedega, mitte numbritega	1
18.	Korduvate elementide mitmekordne väljastamine	1
19.	Hakkab suvaliselt kaarte võtma, kuigi failis on need otsas	1
20.	Joonistamismängus ei õnnestu midagi peale "ei sobi" saada	1
21.	Kui mängus oli viik, siis ei ütle midagi ja seetõttu ei saa aru, mis edasi tuleb teha	1
22.	*Turtle* ei reageeri ja joonistamine ei toimi korrektselt	1
23.	Valida saab varianti "*add more choices*" vales kontektis ja sellel pole väärtust	1
24.	"Ringil puudub ümbermõõt" on täiesti vale fakt	1
	Kokku:	**90**

IV. Projektides esinenud iluvead

Järgnevas tabelis on välja toodud iluvead, mis esinesid kursusel osalejate programmides.

Järjekorra-number	**Vea nimetus**	**Esine-misarv**
1.	Funktsioon ei asu programmis kõige ees, vaid koodilõikude vahel	6
2.	Ebaselge teksti näitamine (suurte kirjavigadega, ilma tekstita arvude näitamine, tulemuse puudumine)	4
3.	Üks funktsioon teise sees	3
4.	Osad asjad käsureal ja osad kasutajaliidesega - väga kirju ülesehitus	3
5.	Tühikute puudumine väljastamisel, teksti printimisel	3
6.	Ei näidata sõna, mida peab arvama või ei näidata hästi tähe asukohta arvamisel	2
7.	Nullide puhul tühikute väljastamine, mis pole kõige paremini loetavad	1
	Kokku:	**22**

V. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, **Carolin Kirotar**,

(autori nimi)

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

Programmeerimiskursuse „Tehnoloogia tarbijast loojaks“ projektide 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 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

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

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

Carolin Kirotar

05.05.2021