TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKA TEADUSKOND
Arvutiteaduse instituut
Infotehnoloogia õppekava

Aarne Aramaa

Pythoni koodi muudatuste analüsaator

Bakalaureuse töö (6 EAP)

Juhendaja: Aivar Annamaa
Seminarijuhendaja: Margus Niitsoo

Tartu 2014

Pythoni koodi muudatuste analüsaator

Lühikokkuvõte:

Selles töös käsitletakse logimistarkvara kasutamist programmeerimise õpetamisel, logidest andmete kogumist ja struktureerimist ning koodi muudatuste analüüsimist ehituse seisukohalt.

Töö käigus täiendati programmeerimise õppimiseks mõeldud programmeerimiskeskkonna Thonny logimise funktsionaalsust ja kirjutati programm, mis Thonny logide põhjal teostab koodi muudatuste analüüsi ja genereerib analüüsi tulemuseks olnud andmetega HTML failid.

Võtmesõnad:

Logimine, koodi analüüs, struktuuri analüüs, programmeerimise algõpe

Python code change analyser

Abstract:

This thesis focuses on the use of logging software in teaching programming, collecting data from the logs and structuring them and analysing changes in code from the point of view of its structure.

As a result of this thesis the logging functionality of Thonny – an integrated developing environment for learning programming – was complemented. Additionally a program that analyses changes made in code by using the logs generated by Thonny was created. The results of the analysis are displayed in HTML files generated by the program.

Keywords:

Logging, code analysis, structure analysis, teaching beginner programming

Sisukord

Sissejuhatus

Programmeerimise algõppe käigus teevad õpilased tööd järgides neile ette antud juhendeid ja kujundavad selle käigus välja oma tööstiili, sõltuvalt tagasisidest, mida nad õppimise käigus saavad. Kuna juhendajatel puudub ülevaade õpilaste tööprotsessist, ei ole kergesti tuvastatavad raskustes olevad õpilased. Lisaks põhineb õppejõu tagasiside üksnes õpilaste valmis töödel, ja seega on tegu piiratud tagasisidega.

Antud bakalaureusetöö ülesanne on seotud programmeerimise õppimiseks mõeldud Pythoni programmeerimiskeskkonnaga Thonny. Thonny, lisaks tavapärastele toimingutele, mida toetab näiteks IDLE, suudab muuhulgas logida kõik kasutaja poolt programmiteksti kirjutamisel tehtud toimingud, nagu näiteks teksti sisestamine, kustutamine ja mujalt kopeeritud teksti kleepimine. Käesoleva töö ülesandeks on nii Thonny logimisfunktsiooni täiendamine, kui ka lisa realiseerimine, mis püüab lahendada probleemi seoses puuduva ülevaatega õpilaste tööprotsessidest. Nimetatud lisa teostab logides salvestatud koodi muudatustega seotud andmete analüüsi, koodi ehituse analüüsi ning struktureerib need andmed kergesti vaadeldavale kujule.

Töö sisu on jagatud neljaks peatükiks. Esimeses peatükis kirjeldatakse probleemi tausta. Teises peatükis on kirjeldatud võimalikke lahendusi koos olemasolevate analüüsivate tarkvaralahendustega. Kolmandas peatükis kirjeldatakse teostatud lahendust ja selle realiseerimiseks teostatud funktsionaalsusi. Neljandas peatükis kirjeldatakse mõningaid edasisi võimalusi, mida on võimalik käesoleva töö tulemuste põhjal realiseerida edasiseks programmeerimise õpetamise efektiivsemaks muutmiseks.

1 Probleemi püstitus

Programmeerimise algõppe käigus saab õpilane pidevalt oma kirjutatud programmi kohta tagasisidet. Tagasiside võib seisneda juhendaja arvamuses, edukas või mitteedukas kompileerumises või programmi rohkem või vähem ootuspärases käitumises. Pidev tagasiside võib motiveerida õpilast oma õpet jätkama ja annab teadmise, et tema käitumine on õige või vale. Akadeemilises keskkonnas, kus juhendaja instrueerib mitmeid õpilasi samaaegselt, võib jääda tema antud tagasiside piiratuks, kuna õpilaste suure arvu tõttu ei ole mõeldav, et juhendaja jälgib iga õpilase tegevusi töö ajal.

Sellisel juhul piirdub juhendaja antud tagasiside hinnangu ja kommentaaridega õpilase valmis tööle, mitte tema töökäigule ja seega ei saa õpilane olla kindel, kas tema lähenemine antud ülesandele on õige ega kas see on üks efektiivseimatest.

Jälgides Dreyfus'i oskuste omandamise mudelit [1], mille järgi õpilase oskustase jaguneb: algaja, edasijõudnud algaja, kompetentne, vilunud, ekspert, võib öelda, et programmeerimise algõppe eesmärgiks on aidata õpilast algaja programmeerija tasemelt kompetentse programmeerija tasemele. Algaja töömeetodeid iseloomustab tugev juhendi järgimine, kuni ta hakkab märkama seoseid enda käitumise ja tingitud tulemuste vahel. See tähendab, et mida täpsemat tagasisidet õpilane saab, seda kiiremini võib ta hakata märkama korrelatsioone oma tegevuste ja nende tagajärgede vahel ja seda kiirem on tema areng.

Õpilastele on kindlasti kasuks ka tagasiside nende tööstiilile. Selline täpsem tagasiside võib tagada selgema arusaama käsitletavast teemast. Et juhendaja või programmeerimiskeskkond ise saaksid tagasisidet anda mitmetele õpilastele nende protsessi kohta, on vajalik ülevaade nende töökäigust. Kuna ei ole mõeldav, et juhendaja jälgib mitme õpilase tööprotsessi samaaegselt, tuleb leida moodus, kuidas õpilase töö tegemisega seonduvad andmed struktureerida kujule, mis sisaldaks piisavalt informatsiooni, millest hiljem järeldusi teha.

Õpilaste töökäigu kohta saaks struktuurseid andmeid koguda logimistarkvaraga, mis salvestab õpilaste tehtud muudatusi programmikoodis töö ajal.

Õpilaste logitud töökäigu põhjal saab teostada statistilist analüüsi, mille tulemuseks oleks programmikoodi arengu seisukohalt tähtsad andmed. Struktureerides analüüsi käigus kogutud

informatsiooni lihtsalt uuritavale kujule on võimalik teha kiirelt mitmete õpilaste töökäikude kohta olulisi järeldusi – eelkõige nende enesekindluse ja oskustaseme kohta programmeerimise ajal – kuna andmetest on näha, kui õpilane teeb tööd katse-eksitus meetodil. Selline võimalus saada kiirelt ülevaadet õpilaste töökäigust tõhustaks oluliselt juhendaja tööd, andes talle teada võimalikest kitsaskohtadest ehk teemadest, millega õpilastel raskusi esineb.

Programmikoodi staatilise analüüsi teostamine annab koodi struktuuri kohta olulist informatsiooni. Sama programmikoodi staatiline analüüsimine erinevatel ajahetkedel võimaldab aga võrrelda kogutud andmeid ning seeläbi koguda tähtsat informatsiooni tehtud muudatuste kohta läbi aja. Näiteks saab selle meetodi abil anda programmeerimist õpetavale juhendajale ülevaate selle kohta, millise konstruktsiooni peale õpilane enim aega kulutanud on.

2 Olemasolevate lahenduste kirjeldus

Selles peatükis on probleemi tausta täpsemalt ja osadena kirjeldatud ning on välja toodud mõningaid võimalikke lahendusi koos nende puudustega.

2.1 Tööprotsessi logimine

2.1.1 Vajadus

Programmikoodi muudatuste analüüsimise aluseks on koodi kirjutamise ajal tehtud tegevuste logimine.

Salvestatud andmeid analüüsides teostatakse sisuliselt koodi muudatuste analüüs, kuna logides olevad märked vastavad tehtud tegevustele, mille tulemuseks on muudatused programmikoodis.

2.1.2 Tööpõhimõte

Piisavalt täpseks töökäigu logimiseks on vajalik ligipääs kindlatele andmetele, mis on seotud iga tehtud muudatusega. Näiteks logides vaid nupuvajutustele vastavat sisestatud teksti, saab salvestatud informatsiooni abil küll mingi ülevaate tegevustest, kuid jääb puudu informatsioon näiteks sisestuskursori asukohast nende ajal. Et tagada ligipääs kõikidele muudatuste seisukohalt tähtsatele andmetele, on oluline, et logimisfunktsionaalsus oleks osa programmeerimiskeskkonnast.

Logidesse on võimalik salvestada kõiki madala taseme tegevusi, mis on seotud programmeerimiskeskkonnaga. Muudatuste analüüsi seisukohalt pole aga tähtis salvestada tegevusi, mille tagajärjel ei teostata ühtegi muutust koodis, seega tuleb kas logida kõrgema taseme tegevusi, või valida, milliseid madala taseme tegevusi logida. Üheks kõige olulisemaks tegevuseks, mida tuleks logida on kindlasti teksti või koodi sisestus ja muutmine.

Teksti sisestust tuvastatakse klahvivajutuste jälgimise teel. Kirjutades koodi tuvastatakse üksikutele sümbolitele vastavate klahvide vajutused, ning salvestatakse see tegevus, kui vastava sümboli sisestus sisestuskursori hetkepositsioonile. Kõiki muutusi põhjustavaid tegevusi koos neid kirjeldava informatsiooniga salvestades saab tagada piisava ülevaate tehtud muudatustest.

2.1.3 Olemasolevad tööriistad

Arenduskeskkonna Eclipse jaoks loodud pistikprogramm Fluorite [2] on näide olemasolevast tarkvarast, mis logib programmeerimise käigus tehtud madala taseme tegevusi ja salvestab need XML failidena. Logidest saab välja lugeda erinevaid statistilisi andmeid, mida saab kasutada mõningate järelduste tegemiseks programmeerija tööstiili kohta. Fluorite jaguneb eraldi logijaks ja logide analüsaatoriks, enda loodud analüüsija kasutamine on julgustatud. Fluorite logidest saab näiteks uurida programmeerijate harjumusi seoses tagurdusmeetoditega [3], ehk milliseid meetodeid kasutatakse programmikoodi varasemate versioonide juurde naasmiseks. Fluorite üks peamiseid omadusi on, et selle logisid saab kasutada ekraanisalvestuse tarkvara asenduse alusena, ehk logide põhjal on võimalik kogu tööprotsessi samm haaval esitada.

Algaja programmeerija tööst ülevaate saamiseks ei ole Fluorite kõige parem tööriist. Kuigi Eclipse on programmeerimiskeskond "kõige ja mitte millegi kindla jaoks" [4] ja vastavate pistikprogrammidega on võimalik keskkonnas Eclipse keeles Python programmeerida, pole see algaja programmeerija vajaduste järgi modelleeritud ning on viimase jaoks tihti liiga keeruline.

2.2 Koodi muudatuste statistiline analüüs

2.2.1 Vajadus

Tehtud muudatustest ülevaate saamiseks on tarvis nendega seotud andmed struktureerida kujule, mis lubab neid kiirelt läbi analüüsida.

Võimalus saada kiirelt iga õpilase töö kohta ülevaadet tõhustab juhendaja tööd ja läbi selle kiirendab õpilase arengut programmeerimise seisukohalt.

2.2.2 Tööpõhimõte

Koodi enda statistiliseks analüüsiks tuleb seda läbida, ning selle käigus selle kohta andmeid koguda. Mõned lihtsaimad toimingud oleksid koodi ridade lugemine programmikoodi pikkuse mõõtmiseks ja kommentaaride pikkuse mõõtmine koodi ja kommentaaride pikkuste suhte teada saamiseks.

Et saada statistilisi andmeid programmikoodi muudatuste, mitte ainult koodi enda kohta, saab teostada koodi statistilist analüüsi erinevatel ajahetkedel ja nende tulemusi võrrelda. Samuti saab teostada analüüsi logitud muudatuste peal, kuna logid sisaldavad muudatuste kohta oluliselt rohkem informatsiooni, kui koodi seisud erinevatel ajahetkedel ise.

Tehtud töö logisid protseduuriliselt läbides saab koguda informatsiooni programmikoodi pikkuse muutumise, lisatud teksti pikkuse ja lisamiste põhjustajate (kleepimine, kirjutamine või laadimine) kohta. Lisaks tehtud muudatustele võib logidest välja lugeda tähtsaid andmeid programmeerija käitumise kohta, nagu programmi käivitamised, akna fookuse muutmised ja programmi salvestamised.

2.2.3 Olemasolevad tööriistad

Kirjutatud koodi kohta statistiliste andmete kogumiseks saab kasutada analüsaatoreid, nagu näiteks PyMetrics [5] ja SourceMonitor [6]. PyMetrics tagastab koodi kohta informatsiooni, nagu selle pikkus ridades ja McCabe'i tsüklomaatiline keerukus, mille väärtus iseloomustab programmi läbimiseks võimalike teede arvu [7]. PyMetrics on aga mõeldud programmeerija endapoolseks kasutamiseks ja tagastab informatsiooni ainult ühe kindla ajahetke kohta.

Vabavaraline SourceMonitor tagastab samuti statistilist informatsiooni koodi ülesehituse ja keerukuse kohta ning lisaks salvestab ta mõõdetud andmeid perioodiliselt, et saada ülevaade koodi arengust läbi aja. SourceMonitor töötab programmikoodidega, mis on kirjutatud keeltes, nagu C, C++, C#, Java või Delphi. SourceMonitor on mõeldud samuti programmeerija endapoolseks kasutamiseks, mis võib põhjustada probleeme algajate puhul, sest see raskendaks nende tööd, segades programmeerimisele keskendumist.

2.3 Koodi struktuuri analüüs

2.3.1 Vajadus

Et koguda informatsiooni programmikoodi struktuuri kohta ja läbi selle saada täpsemat ülevaadet programmi ehitusest, tuleb seda uurida kui struktuurset objekti, mitte kui teksti. Koodis tehtud muudatuste uurimiseks selle ülesehituse seisukohalt, tuleb võrrelda koodi struktuure erinevatel ajahetkedel.

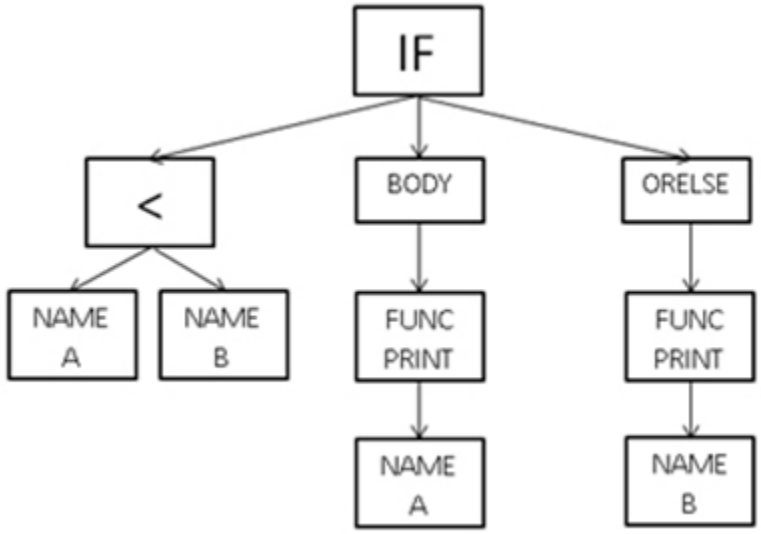
Koodi struktuuri muudatuste analüüsi käigus kogutud andmed annavad täpsema ülevaate sellest, mille kallal on programmeerimise käigus tööd tehtud.

2.3.2 Abstraktne süntaksipuu

Üheks programmikoodi struktureeritud kujuks on abstraktne süntaksipuu, mis moodustatakse, kui kood tehakse programmi jooksutavale arvutile arusaadavaks.

Abstraktne süntaksipuu on programmi lähtekoodi esitus, mis sisaldab struktureeritud informatsiooni lähtekoodi ja tema struktuuri kohta, mida saab kasutada tema õigsuse hindamiseks ja masinkoodi genereerimiseks.

```
if( a < b ) :
    print( a )
else:
    print( b )
```



Joonis 1: Näide programmilõigust ja talle vastava lihtsustatud abstraktse süntaksi puu kujutusest

Abstraktses süntaksipuus sisalduv informatsioon on kasulik programmianalüüsi seisukohalt, kuna koodi lõikude kohta on olemas kontekstuaalsed andmed, ehk on eristatavad näiteks muutuja deklaratsioonid ja funktsiooni definitsioonid. Nagu **Jooniselt 1** on näha, sisaldab abstraktne süntaksipuu ka informatsiooni selle kohta, millised elemendid on omavahel seotud.

Tavapäraselt moodustatakse programmikoodist abstraktne süntaksipuu näiteks kompileerimisprotsessi käigus. Kompileerimisprotsess jaguneb analüüsi ja sünteesi faasiks. Analüüsi faas jaguneb kolmeks alamfaasiks, milleks on: leksiline analüüs, süntaksi analüüs ja semantiline analüüs. Kuna abstraktne süntaksipuu moodustatakse analüüsi faasi käigus, pole sünteesi faas programmeerija töö uurimise seisukohalt nii tähtis.

Abstraktse süntaksipuu moodustamine toimub süntaksi analüüsi käigus. Abstraktne süntaksipuu moodustatakse programmi poolt, mida kutsutakse parseriks. Parseri sisendiks on leksilise analüüsi käigus moodustatud programmikoodi tähtsate elementide, ehk lekseemide jada. Üheks näiteks sellisest lekseemide jadast oleks lihtsale aritmeetilisele avaldisele *Tulemus* = *a* + *b* vastav jada, mille elemendid on *Tulemus*, =, *a*, + ja *b*.

2.3.3 Struktuuri analüüs muudatuste kirjeldamiseks

On selge, et koodi struktuuri analüüs üksikul ajahetkel - üksiku süntaksipuu analüüsimine - ei anna palju informatsiooni tehtud töö ega koodi arengu kohta ning seega pole see väga kasulik õpetamisolukorras, kuna valmis programmi kohta saab juhendaja hinnanguid anda lihtsalt programmikoodi vaadates. Tehtud töö kohta andmete saamiseks tuleb analüüsida programmikoodis tehtud muudatusi.

Programmikoodis tehtud muutusi saab kirjeldada mitmel moel. Muutuste tuvastamiseks võrreldakse kahte erinevat versiooni programmikoodist – varasem ja hilisem. Muutuste tuvastamiseks tuleb sobitada erinevate koodi versioonide vastavad osad, et neid võrrelda [8].

Kui eeldada, et programmi elementide nimed on pidevad, ehk näiteks muutujanimed ja funktsioonide nimed ei muutu läbi koodi arengu, saab programmikoodi lihtsalt selle elementide paaridena võrrelda ja leida nende vahelisi erinevusi, aga kuna ei ole võimalik tagada, et programmi elementide nimed on püsivad läbi koodi arengu, ei ole selline lähenemine väga edukas.

Kui programmi analüüsida tekstina ja eeldada, et ainsad funktsioonid millega teksti muudetakse, on teksti juurde kirjutamine, kustutamine ja välja vahetamine, on võimalik sobitada kõige pikemad tekstide alamosad, mis on muutmata ja seejärel leida kõige vähemate muutmistega moodus jõuda esimese teksti seisust teiseni. See vähimate muutmiste arvuga moodus loetakse teostatud muudatusteks. Sarnasel põhimõttel töötab Levenshteini kauguse leidmine kahe sõne

vahel [9] ehk kahe sõne vahelise tehtud muudatuste arvu leidmine. Programmi analüüs teksti tasemel paraku ei anna aga informatsiooni koodi struktuuri kohta.

Kui programmikoodi versioone vaadeldakse kui abstraktseid süntaksipuid, saab Levenshteini kauguse leidmise põhimõttega leida ka erinevusi puude vahel. Teksti lisamise, kustutamise ja välja vahetamise asemel tuleb muutmise funktsioonideks võtta puu tipu loomist, kustutamist ja välja vahetamist [10]. Väiksemad muutused programmikoodis loetakse selle meetodiga terve puu tipu asenduseks ja seega võivad jääda avastamata tehtud muudatused.

Kuna programmikoodi erinevatele versioonidele vastavate abstraktsete süntaksipuude tipud võivad olla muutunud, mitte tervenisti välja vahetatud, ja kuna nad ei ole nummerdatud tuleb nende paari panemisel leida kas identsed või võimalikult sarnased tipud. On selge et vähesel määral erinevad tipud, mis asuvad süntaktiliselt samal positsioonil, ehk näiteks identsete alampuudega tipud, on suure tõenäosusega üksteisele vastavad [11]. Leides kõik võimalikud paarid kahe süntaksipuu vahel, saab neid võrrelda ja tagastada andmed muutuste kohta. Kõik erinevused paarilisteks loetud tippude vahel on muutused, ning varasemat koodi seisu tähistavast puust puuduvad tipud loetakse lisamisteks ning hilisemat koodi seisu tähistavast puust puuduvad tipud loetakse kustutamisteks.

2.3.4 Olemasolevad tööriistad

On olemas mitmeid analüsaatoreid, mis teostavad Pythoni koodi analüüsi selle struktuuri seisukohalt.

Pythoni programmikoodi analüsaator Pylint [12] tagastab informatsiooni koodi standarditele vastavuse ja võimalike tehtud vigade kohta ning lubab koostada koodile vastavaid UML diagramme. Analüsaatorit Pylint saab ka kasutada koostöös rakendustega nagu Jenkins [13] et tagada pidev integratsioon ja seeläbi koguda andmeid, milles väljenduks koodi areng läbi aja. Pylint kasutamine jääb pigem programmeerija enda hooleks, ning seega võib see raskendada algaja programmeerija tööd. Pideva integratsiooni tulemusena kogutud informatsioon annab ülevaate peamiselt käivitamiste edukusest, mitte tehtud muudatustest, ja on mõeldud suurematel tarkvaraprojektidel rakendamiseks, seega kogutud andmed ei ole piisavad programmeerimise algõppe tõhustamisel.

Pythoni programmikoodi analüsaator PyChecker [14] tagastab, sarnaselt analüsaatoriga Pylint, informatsiooni võimalike koodis tehtud stiili- ja programmivigade kohta. PyChecker jääb samuti programmeerija enda kasutada ning ei anna ülevaadet koodis tehtud muudatustest.

Keeles Java kirjutatud projektides tehtud muudatuste analüüsimiseks on olemas JDiff [15] nimeline analüsaator, mis võrdleb kahte projekti versiooni ning genereerib HTML faile dokumentatsiooniga kõikide erinevuste, ehk liidetud, eemaldatud ja muudetud elementide kohta. JDiff kasutamine annab detailse ülevaate kahe erineva Java projekti väljaande erinevuste kohta, kuid see pole mõeldud üksikutes programmides töö käigus tehtud muudatuste kirjeldamiseks. Kahjuks see ei tööta keeles Python kirjutatud programmidel.

3 Implementatsioon ja tööpõhimõtted

Selles peatükis on kirjeldatud käsoleva bakalaureusetöö implementatsiooni osas teostatud lahendust, mis lahendab probleemi seoses puuduva ülevaatega õpilaste tööst.

3.1 Logimine

Programmeerimise õppimiseks mõeldud Python'i integreeritud programmeerimiskeskond Thonny [16] toetab õppimisprotsessi funktsionaalsustega nagu: programmi lausete käivitamine sammhaaval, visuaalne muutujate väärtustamine ning eraldi aknad funktsioonide väljakutsete täitmiseks. Üheks Thonny funktsionaalsuseks on kasutaja töö käigus tehtud madala taseme tegevuste logimine. Mõned näited tegevustest, mida logitakse on:

- Teksti sisestamine;
- Teksti kustutamine;
- Programmi jooksutamine;
- Programmikoodi salvestamine-laadimine;
- Akna fookuse muutmine.

Tegevuste logimine toimub sündmuste tuvastamise tagajärjel. Sündmusteks loetakse näiteks nupuvajutusi hetkel, kui Thonny aken on fookuses. Nupuvajutustele vastavad tegevused, mis on omakorda sündmused - näiteks *control* ja v klahvi vajutuse kombinatsioon tuvastatakse kui teksti kleepimise sündmus ja kõikide sümbol klahvide vajutused tuvastatakse kui teksti sisestuse sündmused. Et hoida logifailid mõistliku pikkusega, tuli teostada sarnaste sündmuste koondamine. Võimalikud sündmused salvestatakse koos, suuremate sündmustena - näiteks järjestikused teksti sisestamise sündmused, mis on esile kutsutud sümbol klahvi vajutustega, logitakse ühe juhtumina.

```
NewFile(editor_id='45345424') at 2014-04-20T19:16:33.431980
TextInsert(editor_id='45345424', position='1.0', source='None', tags='None', text='a') at 2014-04-20T19:17:02.889665
TextInsert(editor_id='45345424', position='1.1', source='None', tags='None', text=' ') at 2014-04-20T19:17:03.096676
TextInsert(editor_id='45345424', position='1.2', source='None', tags='None', text='=') at 2014-04-20T19:17:03.317689
TextInsert(editor_id='45345424', position='1.3', source='None', tags='None', text=' ') at 2014-04-20T19:17:03.459697
TextInsert(editor_id='45345424', position='1.4', source='None', tags='None', text='1') at 2014-04-20T19:17:03.761714
TextInsert(editor_id='45345424', position='1.5', source='None', tags='None', text='0') at 2014-04-20T19:17:03.962726
TextInsert(editor_id='45345424', position='1.6', source='None', tags='None', text='\n') at 2014-04-20T19:17:04.204740
TextInsert(editor_id='45345424', position='2.0', source='None', tags='None', text='b') at 2014-04-20T19:17:04.906780
TextInsert(editor_id='45345424', position='2.1', source='None', tags='None', text=' ') at 2014-04-20T19:17:05.169795
TextInsert(editor_id='45345424', position='2.2', source='None', tags='None', text='=') at 2014-04-20T19:17:05.432810
TextInsert(editor_id='45345424', position='2.3', source='None', tags='None', text=' ') at 2014-04-20T19:17:05.535816
TextInsert(editor_id='45345424', position='2.4', source='None', tags='None', text='2') at 2014-04-20T19:17:05.977841
TextInsert(editor_id='45345424', position='2.5', source='None', tags='None', text='0') at 2014-04-20T19:17:06.139850
TextInsert(editor_id='45345424', position='2.6', source='None', tags='None', text='\n') at 2014-04-20T19:17:06.621878
TextInsert(editor_id='45345424', position='3.0', source='None', tags='None', text='\n') at 2014-04-20T19:17:07.043902
TextInsert(editor_id='45345424', position='4.0', source='None', tags='None', text='i') at 2014-04-20T19:17:07.345919
TextInsert(editor_id='45345424', position='4.1', source='None', tags='None', text='f') at 2014-04-20T19:17:07.467926
TextInsert(editor_id='45345424', position='4.2', source='None', tags='None', text='(') at 2014-04-20T19:17:07.750943
TextInsert(editor_id='45345424', position='4.3', source='None', tags='None', text='a') at 2014-04-20T19:17:08.635993
TextInsert(editor_id='45345424', position='4.4', source='None', tags='None', text=' ') at 2014-04-20T19:17:08.859006
TextInsert(editor_id='45345424', position='4.5', source='None', tags='None', text='>') at 2014-04-20T19:17:09.222027
TextInsert(editor_id='45345424', position='4.6', source='None', tags='None', text=' ') at 2014-04-20T19:17:10.185082
TextInsert(editor_id='45345424', position='4.7', source='None', tags='None', text='b') at 2014-04-20T19:17:10.468098
TextInsert(editor_id='45345424', position='4.8', source='None', tags='None', text=')') at 2014-04-20T19:17:10.753114
TextInsert(editor_id='45345424', position='4.9', source='None', tags='None', text=':') at 2014-04-20T19:17:12.516215
TextInsert(editor_id='45345424', position='4.10', source='None', tags='None', text='\n    ') at 2014-04-20T19:17:12.879236
```

Joonis 2: Väljavõte Thonny salvestatud logi näidisest ilma koondamiseta

Joonisel 2 on näha väljavõtet ühest Thonny poolt genereeritud logifailist. Kuna klahvivajutuste tagajärjel sisestatud tekst on logitud sümbolite kaupa on logi fail väga pikk.

Thonny logides on sündmused salvestatud kujul sündmuse klass, sündmuse parameetrid ja ajamärge. Sündmuse klassiks on klassi nimi, mis iseloomustab salvestatud sündmuse liiki, nagu näiteks teksti sisestamine, kustutamine või faili salvestamine, ajamärkeks on toimumisaeg kujul aasta, kuu, päev, kellaaeg. Sündmuse parameetrid erinevad klassiti, sündmuste koondamise seisukohalt on tähtsad teksti sisestamine ja kustutamine. Teksti sisestuse andmed on kujul *editor_id, position, source, tags* ja *text*.

- *editor_id* - identifikaator vastava redaktori tähistamiseks.
- *position* - punktiga eraldatud rea ja veeru number, kus sisestus toimus.
- *source* - muudatuse põhjustanud sündmus, näiteks klahvi vajutus, kleepimine või faili laadimine.
- *tags* - lisamärked väljundiaknasse ilmunud teksti jaoks.
- *text* - sisestatud tekst.

Teksti kustutamise andmed on kujul *editor_id, from_position, source, to_position*.

- *editor_id* – identifikaator vastava redaktori tähistamiseks.
- *from_position* – punktiga eraldatud rea ja veeru number, kustutatud teksti lõigu alguse tähistamiseks.

- *source* – muudatuse põhjustanud sündmus.
- *to_position* – punktiga eraldatud rea ja veeru number, kustutatud teksti lõigu lõpu tähistamiseks

Sündmuste koondamine töötab enne teksti sisestuse logimist kontrollides, kas viimane salvestatud sündmus sündmuste järjendis on samuti teksti sisestus – kui viimane salvestatud tegevus on teksti sisestus, mis toimus samal real, ühe sümboli võrra vasakul ja selle toimumisaeg ei erine hetkeajast üle kahe sekundi, lisatakse viimasele salvestatud tegevusele juurde käesolevale sündmusele vastav tekst ja asendatakse toimumisaeg hetkeajaga, lisaks jäetakse meelde sündmuse asukoht. Kui viimane sündmus järjendis on midagi muud kui teksti sisestus, on teksti sisestus, mis toimus mujal kui ühe koha võrra vasakul või toimus rohkem, kui kaks sekundit varem, salvestatakse käesolev teksti sisestus järjendisse eraldi sündmusena.

Kustutamiste koondamine toimub analoogselt. Kui viimane salvestatud tegevus on kustutamine ja tema lõpu asukoht on käesoleva kustutamise alguse asukohast ühe sümboli võrra paremal ning ajaline vahe nende sündmuste vahel on alla kahe sekundi, muudetakse viimase kustutamise lõpu asukoht ja salvestatud sündmuse toimumise aeg. *Delete* klahvi vajutamisega kustutamised koondatakse juhul, kui viimane sündmus on kustutamine, mis toimus samal asukohal mitte rohkem, kui kaks sekundit tagasi.

Järjestikuste sündmuste koondamine hoiab logid mõistliku pikkusega. Salvestades sündmused eraldi, kui nende vahel on toimunud teist liiki sündmus või nende vahel on möödunud üle kahe sekundi, tagab aga logides oleva informatsiooni täpsuse.

```
NewFile(editor_id='502111192') at 2014-04-20T19:25:16.831916
TextInsert(editor_id='502111192', position='1.0', source='KeyPressEvent', tags='None', text='a = 10') at 2014-04-20T19:25:21.401178
TextInsert(editor_id='502111192', position='1.7', source='KeyPressEvent', tags='None', text='\n') at 2014-04-20T19:25:21.603189
TextInsert(editor_id='502111192', position='2.0', source='KeyPressEvent', tags='None', text='b = 20') at 2014-04-20T19:25:23.420293
TextInsert(editor_id='502111192', position='2.7', source='KeyPressEvent', tags='None', text='\n') at 2014-04-20T19:25:24.002327
TextInsert(editor_id='502111192', position='3.1', source='KeyPressEvent', tags='None', text='\n') at 2014-04-20T19:25:24.364347
TextInsert(editor_id='502111192', position='4.0', source='KeyPressEvent', tags='None', text='if(a > b):') at 2014-04-20T19:25:27.999555
TextInsert(editor_id='502111192', position='4.11', source='KeyPressEvent', tags='None', text='\n    ') at 2014-04-20T19:25:28.383577
TextInsert(editor_id='502111192', position='5.4', source='KeyPressEvent', tags='None', text='print("A on suurem")') at 2014-04-20T19:25:33.668879
TextInsert(editor_id='502111192', position='5.25', source='KeyPressEvent', tags='None', text='\n    ') at 2014-04-20T19:25:34.812945
TextDelete(editor_id='502111192', from_position='6.0', source='KeyPressEvent', to_position='6.4') at 2014-04-20T19:25:35.857005
TextInsert(editor_id='502111192', position='6.0', source='KeyPressEvent', tags='None', text='elif(A') at 2014-04-20T19:25:37.743112
TextDelete(editor_id='502111192', from_position='6.5', source='KeyPressEvent', to_position='6.6') at 2014-04-20T19:25:38.888178
TextInsert(editor_id='502111192', position='6.5', source='KeyPressEvent', tags='None', text='a>') at 2014-04-20T19:25:39.836232
TextDelete(editor_id='502111192', from_position='6.6', source='KeyPressEvent', to_position='6.7') at 2014-04-20T19:25:40.620277
TextInsert(editor_id='502111192', position='6.6', source='KeyPressEvent', tags='None', text='<') at 2014-04-20T19:25:40.784286
TextInsert(editor_id='502111192', position='6.7', source='KeyPressEvent', tags='None', text='b):') at 2014-04-20T19:25:44.578503
TextInsert(editor_id='502111192', position='6.11', source='KeyPressEvent', tags='None', text='\n    ') at 2014-04-20T19:25:45.082532
TextInsert(editor_id='502111192', position='7.4', source='KeyPressEvent', tags='None', text='print("A on väiksem")') at 2014-04-20T19:25:49.749799
TextInsert(editor_id='502111192', position='7.26', source='KeyPressEvent', tags='None', text='\n    ') at 2014-04-20T19:25:50.735856
TextDelete(editor_id='502111192', from_position='8.0', source='KeyPressEvent', to_position='8.4') at 2014-04-20T19:25:51.521901
TextInsert(editor_id='502111192', position='8.0', source='KeyPressEvent', tags='None', text='else:') at 2014-04-20T19:25:53.590019
TextInsert(editor_id='502111192', position='8.6', source='KeyPressEvent', tags='None', text='\n    ') at 2014-04-20T19:25:53.896036
TextInsert(editor_id='502111192', position='9.4', source='KeyPressEvent', tags='None', text='print(A') at 2014-04-20T19:25:55.680138
TextDelete(editor_id='502111192', from_position='9.10', source='KeyPressEvent', to_position='9.11') at 2014-04-20T19:25:56.307174
TextInsert(editor_id='502111192', position='9.10', source='KeyPressEvent', tags='None', text='"A on võrdne")') at 2014-04-20T19:25:59.243342
Command(cmd_id='save_file', source='shortcut') at 2014-04-20T19:25:59.972384
```

Joonis 3: Väljavõte Thonny salvestatud logi näidisest koondamisega

Joonisel 3 on näha väljavõtet ühest Thonny poolt genereeritud logifailist, millel on teksti sisestuse ja kustutamise sündmused koondatud.

Kuigi logisid vaadates saab juba mingit informatsiooni tehtud töö käigu kohta, pole see ülevaate saamiseks iseenesest parim moodus. Kui logitud on tööd üle pika aja, ja tulemusena on logi väga pikk, on sellelt tehtud töö uurimine ebamugav.

Thonny logidest saab võrdlemisi lihtsalt välja lugeda andmeid osa lisatud ja kustutatud koodi kohta, töö alguse ja lõpu aega, ning milliste failidega tööd tehakse, aga see on vaid väike osa nendes logides olevast informatsioonist, mis ei anna piisavalt detailset ülevaadet õpilase tööst. **Joonisel 3** kujutatud logil on näha logitud tööd faili loomisest selle salvestamiseni. Toodud näitelt on võimalik välja lugeda, et tegu on lühikese kahe muutuja väärtusi võrdleva programmiga, aga pikkade logide lugemine on tülikas ja aeganõudev tegevus. Logides sisalduvat informatsiooni on parem uurida statistiliste andmetena, mida on võimalik kätte saada protseduuriliselt logisid läbides.

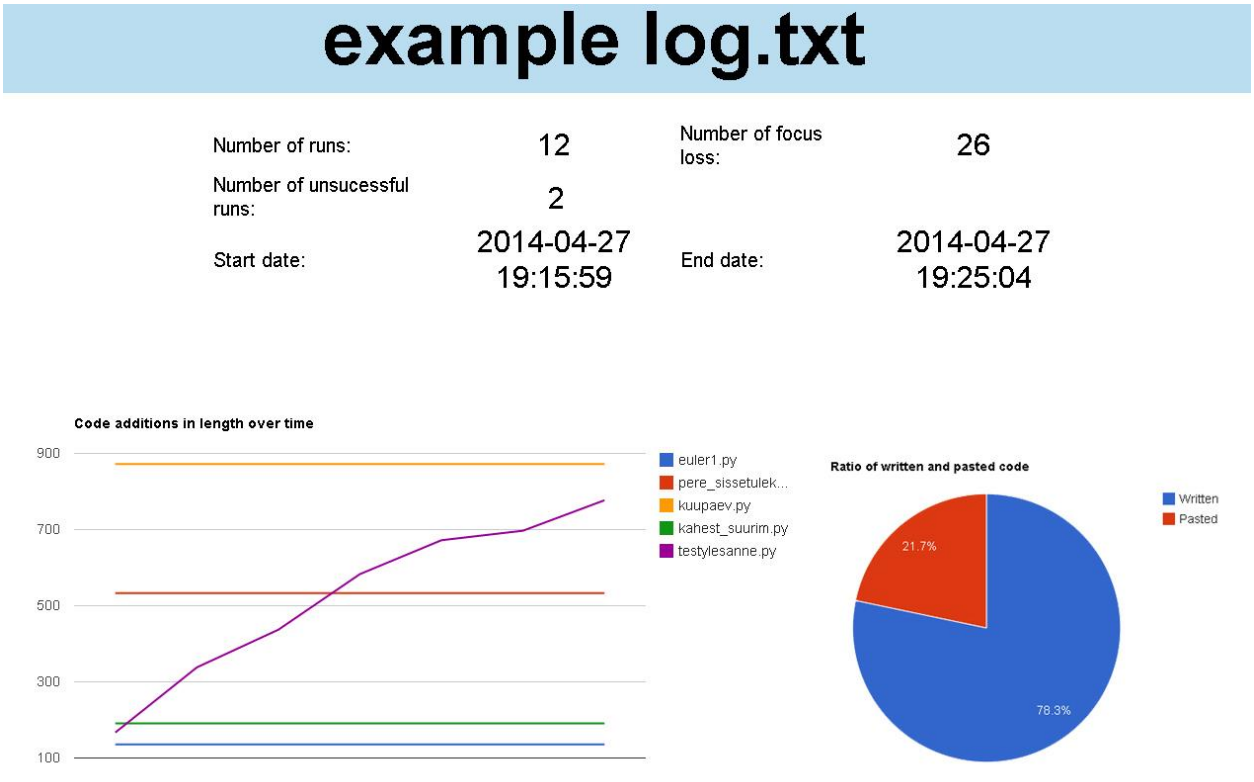
3.2 Koodi muudatuste statistiline analüüs

Kasutades Thonny genereeritud logisid saab programmikoodi ja selle arengu kohta rohkem informatsiooni, kui lihtsalt programmikoodi ennast uurides. Lisaks, analüüsides logisid, ei pea õpilane töö ajal muretsema oma kirjutatud koodi peal mingi analüsaatori kasutamise pärast. Õpilase tööks oleks lihtsalt oma ülesande lahendamine Thonny's programmeerides ja oma tulemuseks oleva programmi koos logiga juhendajale esitamine.

Kuna logid sisaldavad informatsiooni sündmuste toimumisaja, sündmuse toimumise vaheakna, sündmusi välja kutsuva tegevuse, sündmuse tulemuse ja vastava faili nime kohta, saab neid andmeid kindlatel viisidel kombineerides arvutada koodi arengu kohta ülevaadet andvateks statistilisteks andmeteks.

Käesoleva bakalaureusetöö implementatsiooni üheks osaks on programm, mis genereerib logifailide alusel HTML failid, mis sisaldavad struktureeritud andmeid õpilaste tööprotsesside kohta. Genereeritud HTML failides on õpilase töö kohta otseselt logides olevast informatsioonist välja loetud andmed nagu:

- Logifaili nimi;
- Programmi jooksutamiste arv;
- Akna fookuse kaotuste arv;
- Ebaõnnestunud jooksutamiste arv;
- Töö alguse ja lõpu ajahetk;
- Joondiagramm programmikoodide pikkuse kasvu kohta läbi aja;
- Sektordiagramm kirjutatud ja kleebitud koodi suhtega.



Joonis 4: kompaktne näide genereeritud HTML failist

Joonisel 4 on näha genereeritavates HTML failides sisalduvaid andmeid, mis on saadud otseselt logidest, ja nende esitamise kuju.

Nende andmete põhjal saab juba õpilase tööprotsessi kohta mõningaid järeldusi teha. Näiteks kui HTML failides olevatest andmetest on näha, et õpilane proovib oma programmi pidevalt jooksutada ja suur osa nendest üritustest on ebaõnnestunud, on see märk sellest, et õpilane pole kindel oma teadmistes, ja ta on katse-eksitusmeetodil üritanud oma programmi tööle saada. Kui suur osa õpilase koodist on kleebitud, võib see olla märk sellest, et õpilane toetub liialt kellegi teise kirjutatud koodile. Vastupidiselt, kui andmetest on näha, et õpilane jooksutab oma koodi harva, kirjutab seda pidevalt juurde ja sama õpilane on esitanud õige lahenduse oma ülesandele, on see märk pädevast õpilasest. Õpilaste tuvastamisel, kellel esineb ülesannete lahendamisega raskusi, saab neile näiteks pakkuda juhendamist, või teemakohaseid abimaterjale.

Logidest igalt realt andmete lugemiseks ja nende struktuurseks kuvamiseks tuli luua funktsioon, mis sõne kujul ette antud logifaili reast loeks sündmuse informatsiooni sõnastikku, et kõik andmed selle kohta oleksid lihtsalt kättesaadavad ja võrreldavad. Esimesena saadakse kätte ajamärge, mis on alati samas asukohas sissekande lõpu suhtes, ja sündmuse klassi nimi, mis on sissekande alguse ja esimese sulu sümboli vahel. Ülejäänud andmed jäävad sündmuse klassinime ja ajamärke vahele, kus need on komadega eraldatud.

Logide analüüsimiseks tuli luua funktsioon, mis etteantud failist, ülevalpool kirjeldatud funktsiooni kasutades tagastab järjendina programmikoodi pikkuse kasvu igal minutil ja täisarvudena kleebitud teksti summaarse pikkuse, kirjutatud teksti summaarse pikkuse, redaktorite fookuse kaotuste arvu, käivitamiste arvu, vigaste käivitamiste arvu ning alguse ja lõpu ajad. Nimetatud funktsiooni töö käigus seatakse vastavusse redaktorite identifikaator ning selles redaktoris käsitletud faili nimi.

HTML failide genereerimiseks tuli luua funktsioon, mis võtab sisendiks ülevalpool mainitud logidest välja loetud andmed ja kirjutab need koos HTML koodiga iga logifaili kohta eraldi faili. Mõned arvulised andmed nagu programmi käivitamiste arv, redaktorite fookuse kaotamiste arv, ebaõnnestunud käivitamiste arv ja kuupäevad seatakse koos kirjeldava sildiga tabelisse. Koodi pikkus läbi aja, süntaksi puu tippude arv käivitamiste aegadel ja kirjutatud-kleebitud koodi pikkuste suhe esitatakse graafidena.

Graafide loomiseks on kasutatud Google Charts'i [17], kuna selle kasutamine on lihtne ja ei eelda lisatarkvara lisamist faile genereerivasse arvutisse. Joon- ja sektordiagrammide koostamiseks esitatakse andmed mitmetasemeliste järjenditena, nagu neid andmeid andmete kogumise funktsiooni poolt ka tagastatakse.

Lihtsaks ligipääsuks genereeritud HTML failidele genereeritakse lisaks HTML fail, mis sisaldab viiteid andmefailidele.

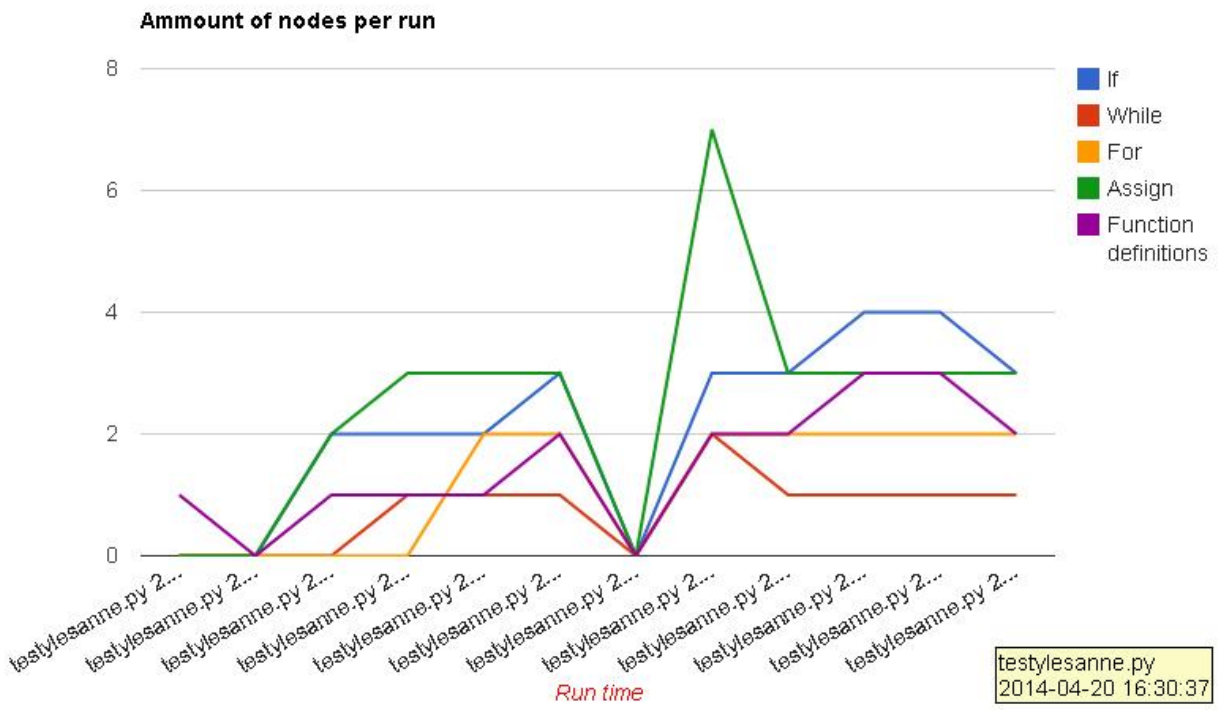
Kuna uute logifailide lisandumisel ja nendele HTML failide genereerimisel, tuleb arvestada juba olemasolevate HTML failidega. Et vältida uute failide töötlemisel korratud tööd, jäetakse vastavatest logifailidest andmete korjamine ja vastavate HTML failide genereerimine tegemata, juhul kui nende logifailide nimega HTML failid juba olemas on.

3.3 Koodi struktuuri muudatuste analüüs

Thonny genereeritud logides on olemas informatsioon sisestatud ja kustutatud teksti kohta koos toimumise asukohaga muu teksti suhtes. Kasutades neid andmeid, ehk logisid samm-sammult läbides saab Thonny's kirjutatud programmi koodi seisu taastada peaegu igal ajahetkel.

Võimalus taastada programmi koodi seisu igal ajahetkel, ehk luua hetktõmmised, võimaldab saada täielikku ülevaadet tehtud tööst seda läbi mängides. Nii oleks näha kõik tehtud koodi lisamised, kustutamised ja millises kontekstis need tehti. Kuigi kasulik funktsionaalsus, ei pruugi programmeerimist õpetaval juhendajal olla põhjust iga õpilase töökäiku otseselt vaadata, seega on tähtsam enne tagada võimalus saada üldisemat ülevaadet õpilaste töö kohta. Iga õpilase töökäigu otsese vaatamise asemel annavad programmi struktuursest arengust ülevaate kiiremini struktureeritud andmed selle kohta.

Kui on juurdepääs programmikoodi seisule igal ajahetkel, on ka läbi selle tagatud juurdepääs sellele koodile vastavale abstraktsele süntaksipuule. Kuna programmi loomisprotsessi käigus on selle kood tihti olekus, kus see ei vasta programmeerimiskeele reeglitele, ei saa parser sellest genereerida abstraktset süntaksipuud. Selle asemel, et püüda pidevalt koodi parsida – mis oleks ka väga resursinõudlik, arvestades et juhendajal on mitmeid õpilasi ja seega mitmeid logisid, mida töödelda – on mõistlik proovida koodi parsida hetkedel, kui programm on suurema tõenäosusega keele reeglitele vastavas olekus, nagu näiteks programmi jooksutamise ajahetkedel.



Joonis 5: Joondiagramm, tähistamaks programmi käivitamishetkedel koodile vastava abstraktse süntaksipuu tippude arvu.

Käesoleva bakalaureusetöö implementatsiooni osas valminud programmi genereeritavates HTML failides on lisaks varem mainitud andmetele ka joondiagramm nagu **Joonisel 5**, millel on kujutatud igal programmi jooksutamise ajahetkel selle koodile vastava süntaksipuu tippude arv, välja arvatud juhud, kui käivitamisel ei olnud kood vastav süntaksi reeglitele. Diagrammil kujutatakse programmikoodis olevaid *if, while* ja *for* lausete, muutuja väärtuste omistamiste ja funktsiooni definitsioonide arvud. Diagramm annab kiirelt ülevaate õpilase käitumisest programmi käivitamiste vahel. Kui diagrammilt on näha mitmeid käivitamisi, millel ei kuvata tippude arvu, mitut järjestikust käivitamist, mille vahel pole tippude arvu oluliselt muudetud või süntaksipuu tippude arvu vähenemist käivitamiste vahel, on see potentsiaalne märk sellest, et õpilase teadmised pole väga kindlad ja ta püüab programmi käivitamiste abil veenduda oma koodi õigsuses. Vastupidisel juhul, kui programmi käivitamiste vahel on pikem ajaline vahe ja tippude arv kasvab pidevalt, on see märk sellest, et õpilane töötab kindlalt soovitud tulemuse suunas, aeg-ajalt veendudes kirjutatu õigsuses. Diagrammilt on näha ka iga käivitamise kohta, milline programm käivitati ja selle käivitamise ajahetk. Joonise all paremas nurgas on näha kasti, mida kuvatakse, kui kursor hõljub diagrammi x-teljel oleva märke kohal.

3.3.1 Hetktõmmised

Programmi struktuuri muutuste analüüsiks oli tarvis luua funktsioon, mis koostaks hetktõmmised, mis vastavad programmi seisule etteantud ajahetkel. Hetktõmmiste loomiseks läbitakse logifail rida haaval, võrreldes igal real olevat ajamärget ja soovitud hetktõmmise ajahetke, et lõpetada logifaili läbimine, jõudes soovitud kaugusele.

Teksti sisestuse märke korral, kui tegu on uuritava redaktoriga seotud sissekandega, loetakse sündmuse positsioon ja sisestatud tekst ning lisatakse koodi ridasid tähistavasse järjendisse vastavale kohale.

Teksti kustutamise märke korral, kui tegu on uuritava redaktoriga seotud sissekandega, loetakse sündmuse alguse ja lõpu positsioon, ning kustutatakse koodi ridu tähistavast järjendist vastavatel positsioonidel tekst ära.

Jõudes reani, mis tähistab sündmust, mis toimus hiljem, kui uuritav ajahetk, lõpetatakse logifaili läbimine ja tagastatakse hetktõmmis.

3.3.2 Parsimine ja tippude lugemine

Hetktõmmisele vastava abstraktse süntaksipuu saamiseks parsitakse seda Pythoni sisseehitatud mooduli *ast* [18] parseriga, mis tagastab *ast* mooduli objekti. Mooduliga *ast* kaasnevad ka meetodid, millega genereeritud abstraktse süntaksipuu tippe mugavamalt läbida saab.

Süntaksipuu tippude arv leitakse läbides puud rekursiivselt, liites iga tipu juures vastavat liiki tippude arvu tähistavale muutujale üks juurde.

4 Edasised võimalused

Mõningad käesolevas töös teostatud funktsionaalsused on aluseks erinevatele võimalustele, mida saab realiseerida, edendamaks programmeerimise algõpet veelgi, andes juhendajatele võimaluse saada täpsemat ülevaadet õpilaste töökäikudest.

4.1 Hetktõmmiste vaatleja

Valmis oleva hetktõmmise looja põhimõttel saaks realiseerida hetketõmmiste vaatleja – programmi, mille abil saab juhendaja otseselt näha õpilase töö käiku seda esitades sarnaselt video ettemängimisele. Selline vaatleja aitaks juhendajal teada saada, kui logidest loetud andmetest on näha, et mingil õpilasel võib olla raskusi ülesande lahendamisega, millises kohas täpselt raskused esinesid.

Hetktõmmiste vaatleja abil saaks ka taastada programmi seisu mingil ajahetkel, et kasutada seda tagurdusmeetodina.

4.2 Täpsemad struktuurimuutused

Käesoleva bakalaureusetöö implementatsiooni osa tulemuses kuvatakse informatsiooni programmikoodi struktuuris tehtud muudatuste kohta minimaalsel viisil, pidades silmas vajadust saada kiiret ülevaadet mitme õpilase tööprotsessist, mis võib olla raskendatud, juhul kui korraga on nähtaval liiga palju informatsiooni iga õpilase kohta. Sarnaselt ülevalpool kirjeldatud hetktõmmiste vaatlejaga võib abiks olla raskusi valmistanud kohtade tuvastamisega täpsem informatsioon tehtud muutuste kohta programmi struktuuris. Näiteks kui analüüsi tulemusest on näha, et on tehtud palju järjestikuseid muudatusi sama süntaksipuu tipule vastavas koodijupis on see selge viide raskusi valmistavale kohale. Et tuvastada, millises süntaktilises kohas tehtud muudatused toimusid, tuleb programmikoodi parsida, ning rakendada süntaksipuu tippude paaritamist, et puude võrdlemine jagada tippude võrdlemiseks.

4.3 Mitmete õpilaste ühisstatistika

Vajaduse ilmnemisel on võimalik realiseerida serveripoolne rakendus, mis teostaks õpilaste poolt saadetud logifailidest, andmete lugemisega, HTML failide genereerimist. Peale eraldi andmete kuvamisele arvutataks ning näidataks andmeid näiteks praktikumirühmade kaupa, andes nii

ülevaate lahendamiseks antud ülesannete raskusastmest. Selline ühisstatistika võiks anda ülevaate sellest kas ülesannete raskusastet tuleks edaspidi langetada või on õpilased valmis suurematekс väljakutseteks.

4.4 Automaatne stiilivigade tuvastamine

Kui realiseerida täpsemate struktuurimuutuste tuvastamine ja nende salvestamine on võimalik koguda andmeid õpilaste tööstiilide kohta. Kuna erineva edukuse tasemega õpilastele on omane kasutada erinevaid võtteid saab kategoriseerida nimetatud võtteid kui positiivsed või negatiivsed tavad.

Õpilaste tegevusi jälgides saaks rakendada masinõppemeetodeid nagu anomaaliatuvastus. Anomaaliatuvastus on andmekaeve protsess, mille käigus tehakse kindlaks andmete mustrid (objektid või sündmused), mis ei sobitu oodatava mustriga [19]. Õpilaste logidest andmete kogumisel oleks võimalik tuvastada tegevused, mis ei klapi positiivseks loetud tavadega.

Automatiseerides erinevate käitumiste ja tegevuste salvestamist ja hindamist oleks võimalik ilma juhendaja sekkumiseta teavitada õpilasi võimalikest halbadest tavadest, mida ei tohiks harjutada.

Kokkuvõte

Bakalaureusetöö raames täiendati programmeerimiskeskkonna Thonny logimisfunktsiooni ja loodi programm, mis logifailidest saadud andmetest loeb välja logitud töökäigu kohta tähtsat informatsiooni ja selle abil teostab struktuurianalüüsi muutuste kirjeldamiseks ja struktureerib nii analüüsi käigus saadud andmed, kui ka logidest välja loetud andmed kergesti uuritavale kujule, mis annab ülevaate tehtud töö kohta.

Töös toodi välja milliseid andmeid saab erinevate automatiseeritud analüüsimeetoditega programmide kohta koguda ning selgitati lugejale programmi analüüsi tagamaid, andes nii lugejale elementaarse arusaama automatiseeritud programmianalüüsi kohta seoses muudatustega programmikoodis.

Töö käigus täiendati Thonny logimise funktsionaalsust ja valmis programm, mis nende logide põhjal teostab koodi muudatuste analüüsi ja genereerib saadud andmeid kujutavad HTML failid. Lisaks kirjeldati lühidalt otsusi, mis mõjutasid implementatsiooni ja millised ülesanded tuli implementeerimiseks lahendada.

Andmed, mida programm on võimeline koguma, annavad piisava ülevaate õpilaste tööprotsessist, et märgata raskustes olevaid õpilasi. Lisaks toodi välja mõned edasised võimalused, mida antud töö tulemuste põhjal teostada saab.

Viited

[1] Dreyfus, Stuart E.; Dreyfus, Hubert L., „A Five-Stage Model of the Mental Activities Involved in Directed Skill Acquisition,“ univ. California, Berkeley, Rep. ORC 80-2, Feb. 1980

[2] Fluorite koduleht, ülevaade – viimati külastatud 07.05.2014
https://www.cs.cmu.edu/~fluorite/index.html

[3] Yoon, YoungSeok, and Brad A. Myers. "An exploratory study of backtracking strategies used by developers." Cooperative and Human Aspects of Software Engineering (CHASE), 2012 5th International Workshop on. IEEE, 2012.
https://www.cs.cmu.edu/~yyoon1/papers/chase12-yyoon-Backtracking.pdf

[4] Eclipse koduleht, Eclipse ülevaade – viimati külastatud 07.05.2014
http://www.eclipse.org/eclipse/eclipse-charter.php

[5] PyMetrics SourceForge leht, kirjeldus – viimati külastatud 07.05.2014
http://sourceforge.net/projects/pymetrics/

[6] SourceMonitor koduleht – viimati külastatud 07.05.2014
http://www.campwoodsw.com/sourcemonitor.html

[7] McCabe, Thomas J. "A complexity measure." Software Engineering, IEEE Transactions on 4 (1976): 308-320.

[8] Kim, Miryung, and David Notkin. "Program element matching for multi-version program analyses." Proceedings of the 2006 international workshop on Mining software repositories. ACM, 2006.

[9] Wikipedia „Levenshtein distance „ – viimati külastatud 01.05.2014
https://en.wikipedia.org/wiki/Levenshtein_distance

[10] Zhang, Kaizhong, and Dennis Shasha. "Simple fast algorithms for the editing distance between trees and related problems." SIAM journal on computing 18.6 (1989): 1245-1262.

[11] Chawathe, Sudarshan S., et al. "Change detection in hierarchically structured information." ACM SIGMOD Record. Vol. 25. No. 2. ACM, 1996.

[12] Pylint koduleht – viimati külastatud 01.05.2014
http://www.pylint.org/

[13] Jenkins koduleht, Jenkinsi tutvustus – viimati külastatud 01.05.2014
https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins

[14] PyChecker SourceForge leht, kirjeldus – viimati külastatud 01.05.2014
http://sourceforge.net/projects/pychecker/

[15] JDiff SourceForge leht, kirjeldus – viimati külastatud 01.05.2014
http://sourceforge.net/projects/javadiff/

[16] Thonny koodi repositoorium – viimati külastatud 07.05.2014
https://bitbucket.org/aivarannamaa/thonny

[17] Google Charts appspot leht – viimati külastatud 01.05.2014
https://google-developers.appspot.com/chart/

[18] AST mooduli dokumentatsioon – viimati külastatud 10.05.2014
https://docs.python.org/3/library/ast.html

[19] Chandola, Varun, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey." ACM Computing Surveys (CSUR) 41.3 (2009): 15.

Python code change analyser

Bachelor's Thesis (6 ECP)

Aarne Aramaa

Summary

With the current academic setup of teaching beginner programmers, the feedback which they receive is strongly based on only the result of their final work and their instructors have no overview of their work process and thus can not easily identify students who need more instructions. There is an abundance of information that could give a thorough overview of a students work, it is just a matter of collecting and saving said data. By this information, we mean data, that can be collected about the students work, for example by applying the use of logging software that would save data associated with changes made in their code by students.

This thesis concentrates on applying the use of logging software in students work and structuring the information collected by the mentioned logging software. As a result of this thesis, the logging functionality of Python IDE Thonny was complemented. Additionally a program that structures the data collected from logfiles was created. The program displays collected data in HTML files that it generates.

The reader is given some background information about different methods of analysing code and how they can be used to describe changes made in code.

Lisad

I. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina **Aarne Aramaa** (sünnikuupäev:08.06.1992)

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

Pythoni koodi muudatuste analüsaator,

mille juhendaja on Aivar Annamaa,

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **13.05.2014**

II. Koodi repositoorium

Käesoleva bakalaureusetöö tulemusena valminud Thonny lisa repositoorium asub https://github.com/AarneaA/PythonAnalyser/tree/Log_stats

Repositoorium sisaldab *log_stats* kaustas HTML faile genereerivat programmi ja nendega kaasaskäivat *style.css* faili.

Examples kaustas on mõned näited Thonny genereeritud logidest ja neile vastavatest HTML failidest.

III. Juhend

Paigaldamine

Eeldusel, et on paigaldatud Thonny, tuleb asukohast https://github.com/AarneaA/PythonAnalyser/tree/Log_stats *Download ZIP* nupule vajutades alla laadida repositooriumi sisu. Allalaetud failist tuleb paigaldada kaust *log_stats* Thonny kausta.

Kasutamine

Eeldusel, et kaustas *user_logs* on olemas logifaile, luuakse *generateHTML.py* käivitamisel *log_stats* kausta iga logifaili kohta HTML fail. Logifailidele pääseb ligi *index.html* faili abil.