VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
KOMPIUTERIJOS KATEDRA

Baigiamasis magistro darbas

Edukacinės platformos modeliavimas

Atliko:
Gintarė Stalygaitė parašas

Vadovas:
dr. Agnė Brilingaitė

Vilnius
2018

Turinys

Santrauka **4**

Summary **5**

Įvadas **6**

1. Susijusių darbų analizė **8**
1.1. *Bloomo* taksonomijos 8
1.2. Šiuolaikinės edukacinės platformos 8
1.2.1. *Moodle* mokymosi erdvė 9
1.2.2. *Bebras* internetinė svetainė 9
1.2.3. *Ville* edukacinė platforma 10
1.2.4. Kitos edukacinės sistemos 10
1.3. Edukacinių sistemų architektūros lygmenys 11
1.4. Užduočių vertinimo metodai 11
1.5. Adaptyvus mokymasis 12

2. Pavyzdinių duomenų analizė **13**
2.1. Sintaksė 13
2.2. Kintamieji 14
2.3. Metodai 14
2.4. Masyvai 15
2.5. Klasės ir konstruktoriai 16
2.6. Išimčių tvarkymas 17
2.7. Analizės rezultatai 18

3. Duomenų modelis **19**
3.1. Pagrindiniai apibrėžimai 19
3.2. Užduočių aibės sukūrimas edukacinėje platformoje 24
3.2.1. Programinio kodo fragmentų parinkimas užduotims 24
3.2.2. Užduočių įvertinimai 25

4. Algoritmai **27**
4.1. Kodo iškraipymas neteisingos eilutės radimui 27
4.2. Pasirinkimų kiekio adaptavimas pagal lygmenį 29
4.3. Atviro tipo klausimo atsakymo vertinimas 30
4.4. Klausimų be kodo fragmentų generavimas 31
4.5. Užduočių generavimas nenaudojant taisyklių 33
4.6. Automatinis lygių keitimas 35

5. Edukacinės platformos prototipo įgyvendinimas **37**
5.1. Edukacinės platformos struktūra 37
5.2. API modelis 38
5.3. Duomenų bazės struktūra 39
5.4. Duomenų bazės užpildymas duomenimis 39
5.5. Skaitliukų parinkimai užduotims be kodo fragmentų 41

Išvados ir rekomendacijos **43**

Ateities tyrimų planas **44**

Literatūros šaltiniai **45**

Santrauka

Visuomenėje didėjant informacinių technologijų paklausai, vis dažniau reikalaujama turėti naudojimosi kompiuteriu įgūdžių. Tiems, kurie siekia įgyti programavimo kompetencijų, šiame darbe sukurta edukacinė platforma leidžia pagal programinio kodo fragmentus, temas, taksonomijas, algoritmų rūšis automatiškai pagal besimokančiojo dabartinį žinių lygį skirstyti užduotis. Modeliuojant edukacinę platformą, siūlomi kelių rūšių užduočių generavimo algoritmai. Informacinių technologijų žinių lygiui įvertinti šiame darbe naudojami taksonomijų lygiai. Dažniausiai pasitaikančioms programavimo klaidoms identifikuoti išanalizuota aibė studentų darbų ir apibrėžtos pagrindinės užduočių temos, klasifikatoriai. Taip pat pristatomas algoritmas, kuris automatiškai prisitaiko prie naudotojo žinių lygio.

Summary

Educational platform modeling

With increasing demand for information technologies, the need to be proficient in computer skills has become common requirement in contemporary society. For those who need to acquire computer-programming competencies, an educational platform was created in this work. The platform allows to generate individually assigned tasks according to the learner's current level of knowledge depending on code snippets, topics, taxonomies and types of algorithm. In the process of educational platform development, several types of task generation algorithms were proposed. A few Taxonomy levels were used in the paper to evaluate the level of knowledge of information technologies. To identify programming errors that occurs most frequently, a set of students' works has been analyzed. A few subjects (classifiers) for the main assignments have been defined. In addition, an algorithm that automatically adapts to the current level of user's knowledge has been introduced.

Įvadas

Temos aktualumas. Informacinių technologijų kompetencijos tampa bendrinėmis kompetencijomis įvairių sričių specialistams. Bet kokioje srityje jau yra reikalingi naudojimosi informacinių technologijų įrenginiais, tokiais kaip telefonas, planšetiniai ar nešiojamieji kompiuteriai, įgūdžiai. Informacinių technologijų srities naudojimo kilimas vyksta ne tik dėl patogumo, bet ir automatizavimo, kas palengvina žmonių gyvenimą ir darbą. Informacinėms technologijoms grįsto mokymosi sistemos bei metodai yra labai įvairūs, todėl svarbu pasirinkti tinkamus ir inovatyvius įrankius. Edukacinės platformos turi būti nuolat tobulėjančios kartu su technologijomis pasaulyje, palaikyti skirtingus mokymosi stilius ir profilius, gerbti besimokančiųjų specifinius suvaržymus, pateikti mokymosi medžiagą pavidalu „bet kur ir bet kuriuo metu“ [18]. Tam remiamasi Lodzinskio ir Pawlowskio pristatomu kokybės kūrimo procesu [10], kuris užtikrina sistemos skaidrumą, adaptyvumą, ištęsiamumą ir suderinamumą.

Darbo tematika – edukacinės platformos tiems, kas turi įgyti naudojimosi kompiuteriu žinių arba programavimo kompetencijas. Tokios edukacinės platformos turėtų pagerinti ir optimizuoti mokymosi procesą, padaryti, kad švietimas taptų gyvenimo dalimi, o ne privalomu dalyku norint įgyti specialybę ar reikiamus įgūdžius. Nuolatiniam savarankiškam įgūdžių tobulinimui reikėtų matyti savo žinių lygį per kompetencijas, kurios dažniausiai apibrėžiamos naudojant taksonomijas. Taip pat svarbu, kad kompetencijos atitiktų reikiamus informacinių technologijų konceptus bei temas. Tad, kokia turi būti pagalbinė mokymosi priemonė, kuri taupytų laiką, automatizuotai kurtų užduotis pagal kompetencijas, temas ir prisitaikytų prie naudotojų įgūdžių? [13] Kad būtų galima atsakyti į **probleminį klausimą**, šiame darbe akcentuojamas automatinis klausimų generavimas, kuris prisitaikytų prie besimokančiųjų žinių lygio pagal mokymosi temas.

Tikslas – sukurti elektroninės edukacinės platformos modelį ir algoritmus, kurie automatiškai generuotų užduotis norintiems įgyti programavimo žinių, pateiktų vertinimą ir nuolatos adaptyviai prisiderintų prie dabartinio besimokančiojo žinių lygio.

Tokiam tikslui pasiekti apibrėžti **uždaviniai**:

1. Išanalizuoti dabar naudojamas edukacines platformas, jų architektūrinius aspektus, vertinimo metodikas, adaptyvaus mokymosi galimybes;
2. Nustatyti taksonomijas ir jų lygius, kurie būtų naudojami kaip sudėtingumo lygmenys edukacinėje platformoje;
3. Išanalizuoti aktualiausias ir dažniausiai daromas studentų klaidas bei pagal tai apibrėžti taisykles duomenų transformacijai, temas, tipus;
4. Sukurti algoritmus, kurie automatiškai generuos skirtingus klausimus pagal rūšis, tipus, taisykles, temas, taip pat, aprašyti algoritmą, kuris automatiškai priderintų užduočių sudėtingumą prie besimokančiojo žinių lygio;
5. Sukurti duomenų modelį, kuris saugo klausimų informaciją, aprašyti sistemos panaudojimo atvejus, funkcinę specifikaciją, sumodeliuoti edukacinės platformos architektūrą;
6. Sukurti prototipą.

Literatūros analizėje aptariamos taksonomijos ir jų lygmenys edukacijos procese. Taip pat pristatomos tokios edukacinės sistemos kaip *Moodle*, *Bebras* bei *Ville*. Apžvelgiami sistemų architektūros kūrimo aspektai, vertinimo modeliai ir kaip sistemos adaptyviai prisitaiko prie besimokančiojo žinių lygio. Išanalizuotos studentų programėlės padėjo suformuoti temas edukacinės platformos

užduočių generavimui. Apibrėžus duomenų modelius, naudotas funkcijas, kintamuosius, sukurta klausimų generavimo ir vertinimo principai. Pritaikytos metodikos klausimų generavimui be kodo fragmentų arba be taisyklių, atviro tipo klausimų vertinimo algoritmas, kodo eilučių iškraipymo principas, eilučių tvarkos sumaišymo metodas ir adaptyvaus prisitaikymo prie besimokančiojo žinių lygio algoritmas. Sudėliota edukacinės platformos architektūra, kur pagrindiniai komponentai yra API ir duomenų bazė. Aptariama edukacinės platformos struktūra ir sąryšiai tarp komponentų, pristatoma kokie sprendimai priimti kuriant edukacinės platformos prototipą. Sprendimai susiję su duomenų bazės modelio užpildymu, klausimų generavimo algoritmų bei vertinimų patikslinimu.

Edukacinės platformos prototipo veikimas, lyginant su mokslo tiriamojo darbo projekto versija, praplėstas pridedant papildomą klausimų ir tipų identifikatorių *klasifikatorius*. Automatizuotas klausimų generavimas tokioms klausimų rūšims: *Neteisinga eilutė, Sumaišytos eilutės, Pasirinkimas su daug teisingų, Atviro tipo, Pasirinkimai su vienu teisingu*. Pridėta galimybė jiems sukurti atitinkamas taisykles. Taip pat prototipe galima išbandyti užduotis, kurios yra sugeneruojamos pagal šias klausimo rūšis ir pasirinkti pagal tam tikrą aktualią temą. Edukacinė platforma patikrina užduotis ir automatiškai nusprendžia kokio sudėtingumo užduotis toliau pateikti naudotojui.

Raktiniai žodžiai: Edukacija, Taksonomijos, Edukacinė platforma, Informacinės technologijos, Mokymosi įrankiai, Programavimas, Praktikavimasis, Programinis kodas, Automatizuotas testų generavimas, API architektūra, Mokymosi aplinka, Duomenų transformavimas, Adaptyvus mokymasis.

Šiame darbe 1 skyriuje atlikta susijusios literatūros analizė. Skyriuje 2 išanalizuoti ir aprašyti pavyzdiniai studentų darbai. Skyriuje 3 kalbama apie edukacinės platformos pagrindinius duomenų modulius. Skyrius 4 pateikia pagrindinius algoritmus naudojamus edukacinėje platformoje. Kitame, 5 skyriuje, pristatoma modeliuojamos edukacinės platformos architektūra bei jos pagrindiniai komponentai, prototipo įgyvendinimo pasirinkimai.

Šis darbas remiasi tęsiamu mokslo tiriamiojo darbo projektu, pradėtu ankstesniame semestre. Lyginant su mokslo tiriamojo darbo projektu, papildyta šaltinių analizė, praplečiant taksonomijų aprašymus, *Moodle, Ville* sistemų detalizacija. Pridėtas edukacinės platformos *Bebras* aprašymas, sistemų architektūrų apžvalga, vertinimo metodikų analizė bei adaptyvaus prisitaikymo prie naudotojo žinių lygio analizė. Pridėta studentų pavyzdinių užduočių analizė (žr. 2 skyrius), padarytos išvados apie jas ir suformuluotos temos edukacinės platformos užduotims. Praplėstas 3 skyrius, pridedant daugiau apibrėžimų, funkcijų, operatorių, bei užduočių generavimo bei vertinimo principų aprašymai. Skyriuje 4 pridėti nauji algoritmai pasirinkimų kiekio adaptavimui pagal lygmenis, atviro tipo klausimų atsakymo vertinimui, klausimams be kodo fragmentų ar be taisyklių generavimui bei rekomendacijos kaip juos vertinti. Taip pat pridėtas automatinio prisitaikymo prie naudotojo žinių lygio algoritmas. Lyginant su mokslo tiriamojo darbo projektu, atnaujintas duomenų bazės modelis, paaiškinta sluoksniuota sistemos architektūra ir kam naudojami apibrėžtieji sluoksniai, pridėta įgyvendinimo dalis (žr. 5 skyrių).

1. Susijusių darbų analizė

1.1. *Bloomo* taksonomijos

Taksonomijos yra tam tikra mokymosi ir ugdymo tikslų klasifikacija, taikoma klasifikacijos hierarchijai sukurti. Taksonomijos padeda kelti tokius siekinius mokiniams ir sudarinėti jiems klausimus taip, kad būtų galima įvertinti daugelį pažinimo veiklos sričių. Informacinių technologijų koncepte egzistuoja įvairių veiklos sričių: duomenų bazės, dizaino modeliai, objektinio programavimo konceptai ir pan. Yra įvairių taksonomijų, tačiau šiame darbe orientuojamasi į informacinių technologijų sričiai adaptuotas *Bloomo* taksonomijų kategorijas (žr. 1 lentelę), kurias pristato Bukauskas ir kiti [19].

1 lentelė. *Bloomo* Taksonomijos informacinių technologijų specialistams.

Lygis	*Kategorija*	*Pažintinis procesas*
1	Prisiminimas	Pažindinimas, priminimas, aprašymas, nurodymas
2	Supratimas	Interpretavimas, iliustravimas, klasifikavimas, lyginimas, paaiškinimas, perfrazavimas, apibendrinimas
3	Taikymas	Vykdyti, įgyvendinti, išspręsti, manipuliuoti
4	Analizė	Išskirti, atskirti, padalinti, organizuoti
5	Įvertinimas	Patikrinti, kritikuoti, įvertinti, palyginti, sugretinti
6	Kūrimas	Generuoti, planuoti, gaminti, kurti inovacijas, modeliuoti, organizuoti

Ginat ir Menashe [9] rekomenduoja mokomųjų programavimo užduočių lygius susieti su taksonomijomis, kurių sudėtingumas identifikuojamas iš užduoties savybių, tokių kaip programavimo komponentų pritaikymas, sąrašų panaudojimas, skaičiavimai, matricos skaitymas ir rašymas, ciklų iteracijos, kintamųjų naudojimas ir pan. Pagal šias sritis ir taksonomijos lygį galima kurti ir pateikti atitinkamo pobūdžio klausimus. Tai padėtų tinkamai įvertinti ir apibūdinti naudotojus.

1.2. Šiuolaikinės edukacinės platformos

ISO 9241 standartuose [12] apibrėžta, kad kokybiška sistema turi sugebėti užtikrinti žmogaus ir sistemos tarpusavio bendravimą: turi būti individualizuota, atitinkanti lūkesčius, lengvai ir lanksčiai valdoma, laikytis klaidų tolerancijos naudotojo atžvilgiu. Šie standartai nustato projektavimo principus vartotojo sąsajos dizainui, kad būtų tinkamai parinkta vaizdinė medžiaga, kad sistema būtų lengva ir greita naudotis. Kadangi kalbama apie edukacinę sistemą skirtą informacinių technologijų specialistams, svarbu, kad ši platforma suteiktų ir įgūdžių naudotis kompiuteriu, duotų ne tik teorinių bet ir praktinių panaudojimo žinių. Šiame darbe modeliuojamoji edukacinė sistema remiasi kokybės standartais ir orientuojasi į neapkrautą ir vartotojo sąsajos dizainą, pateikiamų užduočių turinį ir naudą.

Informatikos konceptai paremti sintaksės, raktažodžių bei gramatikos taisyklėmis, kurių privalu laikytis. Programuotojui sumaišius simbolių seką ar metodų pavadinimus gali sukelti prižiūrimos technikos veikimo problemas ar programos defektus. Kiekvienoje programavimo kalboje yra savotiškų niuansų, nenuoseklumo problemų, kurias reikia įsiminti ir tiesiog žinoti (pavyzdžiui, paprastos komponentų išraiškos – *services* ir *server*, *manager* ir *management* arba paprasčiausiai metodų pavadinimai – *getUserState*, *getUserStatus*) [7]. Šiame darbe apibrėžiami klausimų tipai ir

užduotys, kurie gali padėti lavintis ir geriau įsiminti raktinius žodžius, identifikuoti sintaksės bei gramatikos klaidas.

Šiame darbe aptariama keletas skirtingo pobūdžio edukacinių platformų: *Moodle*, *Ville*, *Bebras*. *Moodle* adaptuota mokymosi valdymo procesų kūrimui, *Ville* daugiau koncentruojasi į užduočių pateikimą, o *Bebras* orientuojasi į konkursų organizavimą ir kompetencijų kėlimą prieš juos. Be šių sistemų pristatytos dar kelios, *MyPeerReviewing* ir *Scratch*, kurios taip pat padeda mokytis kitokiomis metodikomis: kodo peržiūros ir dalinimosi principais.

1.2.1. *Moodle* mokymosi erdvė

Moodle – virtuali, atviro kodo, mokymosi aplinka [6], kurioje yra integruoti įvairaus pobūdžio įrankiai: veikloms organizuoti, medžiagai pateikti, valdyti naudotojų informacijai, įvertinti jų gebėjimus, mokymosi rezultatus, stebėti kokias užduotis atlieka, atlikti testus. *Moodle* palaiko automatizuotą mechanizmą, kuris leidžia dėstytojams kurti savikontrolės klausimus su keliais teisingais, neteisingais atsakymais, vaizdine, filmuota medžiaga, su pasirenkamais atsakymais, su parenkamomis poromis „prielaida-atsakymas“, taip/ne, laisvos formos atsakymo ir pan. Pagrindinės teigiamos *Moodle* sistemos savybės:

1. Tinka ir nuotoliniam mokymuisi, ir mokymuisi kompiuterių klasėje;
2. Elektroninio mokymosi kursus (dalykus) galima rūšiuoti pagal skirtingas kategorijas, vykdyti paiešką raktinių žodžių pagalba;
3. Galimybė nurodyti elektroninio mokymosi kurso (dalyko) aprašą ar modulio kortelę;
4. Galimybė stebėti vartotojų aktyvumą;
5. Integruotos saugumą užtikrinančios priemonės.

Daukilo ir Kasperiūnienės mokymo kursų kūrimo medžiagoje [6] pristatomi įskiepiai, kurie praplečia sistemos *Moodle* veikimą. Informatikos mokslų įskiepis naudojant nustatytą kodavimo stilių apdoroja programavimo išraiškas, leidžia naudoti duomenų bazių kodą. Modeliuojamoje edukacinėje platformoje nenumatyta papildomų įskiepių.

1.2.2. *Bebras* internetinė svetainė

Bebras – tarptautinė iniciatyva, kurios tikslas yra skatinti informatiką (kompiuterių mokslų studijas, ar kompiuteriją) ir informatikinį mąstymą. Tai yra mokymosi tikslais paremta svetainė, kurioje vyksta skaičiavimų ir kompiuterių mokslų konkursai, yra galimybė ir praktikuotis, mokytis pavyzdžių nagrinėjimo ir tarpusavio varžybų principais. Dagienė, Sentance ir Stupurienė [17] pristato penkias informacinių technologijų mokymosi sritis, kurios susijusios su pagrindinėmis šios sistemos turinio kategorijomis:

1. Algoritmai ir programavimas, įskaitant loginius argumentus;
2. Duomenys, duomenų struktūros ir reprezentacijos;
3. Kompiuterių procesai ir aparatūra (apima viską, kas susiję su kompiuterių darbais – planavimas, lygiagretus apdorojimas);
4. Ryšiai ir tinklų kūrimas (įskaitant kriptografiją, debesis);

5. Sąveika (žmogaus ir kompiuterio sąveika).

Šios sistemos užduotys suteikia skaičiavimo mąstymo įgūdžių ugdymą įtraukiant abstrakciją, algoritmais paremtą mąstymą, vertinimą ir apibendrinimą. Mokytojai, kurdami užduotis, stengiasi, kad viena užduotis atitiktų vienai informacinių technologijų mąstymo sričiai. Šiai sričiai pabrėžti naudojamas raktinių žodžių principas. Tai padeda ne tik atsifiltruoti užduotis pagal temas, bet ir suprasti, kokioms informacinių technologijų kategorijoms priskirtos užduotys. Modeliuojamoje edukacinėje platformoje kurdami klausimus mokytojai taip pat gali apibrėžti klausimų raktinius žodžius, klasifikatorius, apibrėžiančius klausimo turinį bei dalykinę sritį.

1.2.3. *Ville* edukacinė platforma

Ville – internetinis puslapis [14], pasižymintis paprastumu, naudojamas kaip edukacinė programa. Ji orientuota į informacinių technologijų specialistus, suteikia galimybę mokytis vizualiniais metodais. *Ville* sistema pateikia kelias skirtingas programavimo kalbas (*Java*, *C++* ir *pseudokodo*). *Ville* savybės:

1. Galimybė peržiūrėti programavimo pavyzdžius skirtingomis programavimo kalbomis ir atrasti skirtumus, panašumus tarp jų;

2. Leidžia vykdyti pavyzdžius keliose skirtingose programavimo kalbose vienu metu. Tai leidžia lygiagrečiai stebėti kaip vykdosi procesai keliose skirtingose kalbose.

3. Mokytojai gali sukurti daugkartinio pasirinkimo klausimus ir nustatyti juos tam tikriems naudotojams;

4. Kodo kintamųjų reikšmė yra paaiškinta eilutėse. Vykdant kodą eilutės taip pat paaiškinamos, nuolatos pateikiama informacija apie vykdymąsi, įgyvendintas veikimo pažingsniui rėžimas;

5. Mokytojai gali pridėti programavimo pavyzdžių kartu su grafiniais pavyzdžiais, publikuoti juos paskaitose arba visame internete;

6. Vizualizacijos gali būti kodo linijų tipo, testo klausimų tipo, grafiniai ar kaip kodo prototipai.

Kaila, Kurvinen ir Lokkila savo knygoje „Ville teacher‘s handbook“ [8] pristato šį įrankį programavimo mokymuisi mažų užduočių principu. Klausimo tipai, kurie padeda informacinių technologijų specialistams, yra metodų įvesties ar išvesties nustatymas, taip pat, kodo sumaišymo, kur naudotojas turi nustatyti teisingą programinio kodo eilučių seką, konvertavimo tarp skaičių sistemų užduotys. *Ville*, taip pat kaip *Bebras* ir *Moodle*, naudoja raktinių žodžių sistemą užduočių klasifikavimui bei filtravimui. Tipai pateikiami sudėtingumo tvarka nuo sunkiausio iki paprasčiausio. Pirmasis lygmuo reikalauja savarankiško mąstymo ir žinių apie programų vykdymą, o tuo tarpu, trečiasis lygmuo geriausias pradedantiesiems. Šiame darbe modeliuojamoje edukacinėje platformoje lygmenys taip pat naudojami, tačiau pristatomi atvirkštine tvarka ir panaudojant *Bloomo* taksonomijos kategorijas. Pirmasis lygmuo yra paprasčiausias, atitinkamai, kuo aukštesnis lygmuo, tuo sudėtingesnės užduotys.

1.2.4. Kitos edukacinės sistemos

Dar viena vizuali aplinka leidžianti mokytis programavimo yra *Scratch* [15]. Ji turi galingą mechanizmą, leidžiantį vartotojams tyrinėti ir mokytis vieniems iš kitų bei dalintis atviruoju kodu,

kurti koncepcines ir vizualines pamokas bei mokslo laboratorijos užduotis. Be kodo dalinimosi ir skaitymo pratimų sistema leidžia mokytis programavimo koncepcijų ir naudoja animacijas, padedančias suprasti sudėtingas programavimo temas. Ji naudoja *drag-and-drop* funkcionalumą, vengdama sintaksės kliūčių, kas irgi naudojama šiame darbe modeliuojamoje edukacinėje platformoje. Taip pat įgyvendina programavimo logiką, susijusią su *if-else, for, repeat, random* veiksmais, *random, and-or* išraiškomis.

MyPeerReview yra tarpusavio vertinimo ir peržiūros sistema [11], kurios tikslas pagerinti programavimo kokybę ir išlaikyti vienodus programavimo standartus. Šio įrankio pagalba galima atlikinėti užduotis, vertinti vieniems kitus, kas padeda pamatyti, kokioje srityje reikėtų patobulėti. Pagrindinė idėja yra mokymasis peržiūrint kitų atliktas užduotis, nes tai yra mokymosi forma, kai naudojami įsiminimo, analizės principai. Tačiau modeliuojamoje edukacinėje platformoje vertinimas ir žinių lygio nustatymas yra atliekamas automatiškai.

1.3. Edukacinių sistemų architektūros lygmenys

Kaip Stefan ir kiti [5] pabrėžia, modeliuojamoje edukacinėje sistemoje įžvelgiamos ir išlaikomos dvi esminės loginės pusės: verslo logika ir atvaizdavimas klientui. Serverio pusėje esantys verslo logikos komponentai atsakingi už duomenų saugojimą, traukimą, interpretaciją. Kliento pusėje yra daugiausiai orientuojamasi į vizualizavimą besimokančiųjų grupėms.

Medrek ir Tatarczank [18] platformas skirsto į tris esminius sluoksnius. Duomenų sluoksnis – duomenų bazės komponento apibrėžimas. Antrasis sluoksnis yra transformacijos, skirtas apdoroti einamuosius duomenis, informacijos ištraukimui iš duomenų bazės ir pateikimui į ją (konvertavimas, valymas, traukimas, suskaidymas, įrašymas). Trečiasis, analitinis sluoksnis, susideda iš duomenų gavybos, užklausų ir ataskaitų teikimo įrankių naudotojams. Šio sluoksnio pagrindinė funkcija yra skirtingai parodyti įgytą ir iš anksto apdorotą informaciją įvairiais formatais skirtingiems naudotojams.

Modeliuojamoji sistema remiasi dviem loginėmis pusėmis: verslo ir atvaizdavimo. Atvaizdavimo naudotojui langas pasižymi minimalistiniu dizainu, o verslo logikoje apibrėžiami dar keli sluoksniai: duomenų bazės ir transformacijos.

1.4. Užduočių vertinimo metodai

Kiekvieno besimokančiojo supratimo lygis tam tikrose koncepcijose yra skirtingas, todėl labai svarbu išsiaiškinti žinių lygį, kad būtų galima parinkti tinkamo tipo ir sudėtingumo užduotis [3]. Užduočių vertinimo modeliai skirstomi pagal užduočių tipus ir turi metrikas, kurios padeda įvertinti kokio lygio žinias turi naudotojai. Tokių vertinimo modelių, skirstymų ir metrikų yra įvairių.

Ginat ir Menashe [9] pristato tam tikrą užduočių klasifikavimą į nestruktūrines ir daugiafunkcines užduotis. Nestruktūrinės užduotys, kurių sprendimams reikalinga tiesiogiai taikyti bendrąjį modelį, pavyzdžiui, paprastą skaičiavimą. Tokios užduotys vertinamos skiriant mažiausią arba didžiausią skaičių taškų. Užduotis daugiafunkcinė, kai reikia tiesiogiai taikyti du (ar daugiau) bendrus modelius, ji apima daugiau nei vieną elementą. Tokia užduotis yra, pavyzdžiui, pakeisti įvesties sąrašą. Ji susijusi su paprastu sąsajos būdu bendriesiems sąrašo matmenims ir spausdinimu atvirkštinių masyvų. Šiame darbe pritaikomas nestruktūrinių užduočių vertinimo modelis, kai nusakoma kiek taškų galima gauti mažiausiai ir daugiausiai atitinkamai pagal tai, kokio sudėtingumo lygmens užduotys.

Vieną iš vertinimo metrikų, *Levenšteino atstumą*, pristato Day [1]. Šis atstumo matavimo vienetas skirtas palyginti nesutapimus tarp dviejų tekstinių simbolių aibių. Vienos aibės elementus tikrinant su kitos aibės elementais netinkami elementai pakeičiami teisingais. Tokiu būdu gaunama konstanta, įvertis, kiek veiksmų reikia atlikti, kad aibės taptų vienodos. Ši vertinimo metrika naudojama modeliuojamoje edukacinėje platformoje.

1.5. Adaptyvus mokymasis

Adaptyvus mokymasis yra kompiuterinė ir (arba) internetinė švietimo sistema, kuri keičia medžiagos pristatymą, atsižvelgiant į naudotojų veiklos rezultatus [4]. Taip automatizuotai sukuriama optimali individuali mokymosi zona, kur nebūtų per daug nuobodu ar per sunku. Oppl, Reisinger, Eckmaier ir Helm [4] pristato dviejų dalių naudotojų modelių tipus: psichologinis ir žinių. Psichologiniame atsižvelgiama į psichologines ir socialines savybes mokymo proceso metu. Šiame darbe gilinamasi į žinių modelį, kuris leidžia valdyti konkretaus naudotojo mokymosi konceptus priklausomai nuo jo specifinių žinių.

Naudotojo vertinimui gali būti naudojamas CAT (angl. *Computerized Adaptive Testing*) [4] bandymo procesas, kuris dinamiškai parinktų atitinkamą lygmenį kontekste, pasirenkant bandymo elementus. Tuomet parinktoje elementų grupėje galima nustatyti įgūdžių lygį. Elementai gali būti kelių rūšių: vienareikšmiai (*dichotominiai*) arba daugiareikšmiai (*polytomous*), kur vienareikšmiai vertinami kaip *teisingas/neteisingas*, o daugiafunkciniai turi keletą variacijų. Šiame darbe modeliuojamoje edukacinėje platformoje taip pat einamasis elemento sudėtingumas priklauso nuo anksčiau atsakytų elementų ir naudojami tiek vienareikšmiai, tiek daugiareikšmiai elementų tipai. Taip pat šiame darbe orientuojamasi į naudotojo žinių lygio nustatymą remiantis diagnostika: įvertinant atliktas naudotojo užduotis nustatomas jo žinių lygis ir toliau pateikiamos užduotys pagal tą lygmenį.

Millan, Loboda ir Perez-de-la-Cruz [2] pristato *Bayesian* trijų etapų tinklo kūrimo procesą:

1. Apibrėžimas. Šiame etape nusprendžiama kurios temos ar įgūdžiai priklauso nuo kitų temų ar įgūdžių. Taip sudaroma tinklo struktūra, kur nustatomi esminiai mokymosi mazgai ir ryšiai/lankai tarp jų;

2. Inicializuoti žinių vertę, įvertinti naudotojų mokymąsi. Tai gali būti atlikta įvairiais būdais: ekspertų spėliojimai, kitų naudotojų normatyvai ar konkretaus naudotojo ankstesni rezultatai;

3. Tikimybių atnaujinimas. Atnaujinamos *Bayesian* tinklo tikimybės naudojant praeities ir dabarties informaciją.

Šiame darbe modeliuojamoje edukacinėje platformoje naudojamas šis trijų etapų *Bayesian* tinklo modeliavimas automatiniam klausimų parinkimui. Antrajame žingsnyje koncentruojamasi į konkretaus naudotojo ankstesnius rezultatus.

Kaip siūlo Millan, Loboda ir Perez-de-la-Cruz [2], *Bayesian* tinklo vartotojų modelio mazgai, kintamieji gali būti mokymosi procesai: žinių vienetai, kompetencijos ar mokymosi įvykiai. Kiekvienam iš jų galima priskirti vertę, o lankai reiškia santykius tarp jų. Pristatomi dviejų tipų santykiai. Pirmasis yra santykis tarp kompetencijos ir žinių. Antrasis – santykis tarp žinių vieneto ir įrodinėjimo mazgo. Šiame darbe modeliuojamoje edukacinėje platformoje naudojamas tik pirmasis santykio tipas, kuris nurodo santykį tarp žinių tipo (kategorijos) ir kompetencijos (taksonomijos).

2. Pavyzdinių duomenų analizė

Šiame darbe užduočių įvairovė ir vertinimo metodikos projektuojamos remiantis pavyzdinių duomenų analize. Informacinių technologijų studijų programos studentams mokslo metų eigoje pateikiami kontroliniai, kurie padeda nustatyti jų žinių lygį. Suformuojamos vienodos užduotys ir yra vertinamas jų atlikimas: sintaksė, veikimas, metodai, algoritmai. Buvo atlikta Vilniaus universiteto Matematikos ir informatikos fakulteto Informacinių technologijų studijų programos pirmo kurso vienos grupės studentų kontrolinių darbų analizė. Viso peržiūrėtos 26-os skirtingų tos pačio grupės studentų programėlės, kur 14 programų skirta programavimo pagrindams tikrinti, o likusios 12 objektinio programavimo (paveldėjimo) žinioms patikrinti. Visos programėlės parašytos su *Java* programavimo kalba. Atlikta išsami šių programų analizė.

2.1. Sintaksė

Java, kaip ir kitos programavimo kalbos, remiasi gramatikos, sintaksės taisyklėmis, konstruktais. Raktiniai žodžiai, konstantos, operandai turi nustatytas taisykles, kaip turi būti apibrėžti. Net ir menkiausias papildomas ar praleistas simbolis gali sustabdyti visos sistemos veikimą. Atlikus studentų darbų peržiūrą identifikuota, kad dažnai praleidžiami tam tikri simboliai.

1 pavyzdys.

```
1  Unit[i] = Integer.parseInt( scaleToGrams[i][3];
2  if (scaleToGrams[i][0]) == name;
3  for(int i=0;I<input.length;i++)
4  int b=new args.length;
5  scaleToGrams.lengtht();
6  length.scaleToGrams;
7  if (scaleToGrams[i][2] != Null)
8  public static void main(Strings[] args)
```

Kaip matome 1 pavyzdyje, eilutėse 1 ir 2, nepridėtas simbolis ')' programinės eilutės pabaigoje. Pasitaiko, kad apibrėžiami kintamieji ne pagal nustatytus *Java* programavimo kalbos reikalavimus. Eilutėje 3 pirmą kartą *int* tipo kintamasis apibrėžtas mažąja *i*, vėliau naudojamas iš didžiosios *I* ir vėl iš mažosios. Eilutėse 4, 5 ir 6 neteisingai naudojamos standartinės programavimo kalbos bibliotekos ir jų metodai, šiuo atveju tai yra *length* išraiška. Eilutėje 7 neteisingai panaudojamas *null* raktažodis, nes *Java* programavimo kalboje nustatyta jį rašyti iš mažosios *n* raidės: *null*. Pateiktame 1 pavyzdyje, eilutėje 8 matome, kad visiškai neteisinga duomenų tipo išraiška *Strings*, nes *Java* programavimo kalboje šis duomenų tipo raktažodis naudojamas vienaskaitos formoje: *String*.

Objektinio programavimo paradigmos kalbose naudojami prieinamumo modifikatoriai, kurie nusako iš kur metodas gali būti pasiekiamas. Pagrindiniai modifikatoriai *Java* programavimo kalboje yra *public, private, protected*. Studentų darbuose pasitaiko, kad modifikatoriai apibrėžiami netinkamai.

2 pavyzdys.

```
1  Public static void main (String [] args)
2  Public void fillProducts (String [] produktai)
3  Public static int convertToGrams ( String name, int unit)
4  Public double GetWeight ()
```

Kaip matome 2 pavyzdyje modifikatoriai *public* parašyti iš didžiosios raidės, tačiau *Java* programavimo kalboje reikalaujama juos rašyti iš mažosios raidės.

2.2. Kintamieji

Java programavimo kalboje bet kokie kintamieji, kad jie būtų galimi naudoti, turi būti deklaruojami programiniame kode pagal norimą duomenų tipą. Primityviems kintamiesiems apibrėžimas nebūtina, tačiau objektai ir masyvai turi būti inicializuojami patikslinant dydį ar duomenų tipą.

3 pavyzdys.

```
1  public class Recipe
2  {
3    public double getWeight(){
4      weight = convertToGrams();
5      return weight;
6    }
7  }
```

Pavyzdyje 3 matome, kai visiškai neinicializuotai reikšmei *weight* (žr. 4 eilutė) bandomas priskirti metodo *convertToGrams()* rezultatas ir bandomas grąžinti kaip rezultatas metodui *getWeight()* (žr. 5 eilutė). Studentų darbuose pasitaiko atvejų, kai bando naudoti neinicializuotus kintamuosius (*String[][] products;* žr. 4 pavyzdyje, 4 eilutė) arba jie apibrėžiami netinkamai.

4 pavyzdys.

```
1  private String [][] products;
2    ...
3  public void fillProducts(String [] arr){
4    String [][] products;
5    ...
6  }
```

Java programavimo kalba nesuteikia galimybės dubliuoti kintamųjų naudojamų globaliai ir esančių metodo viduje. Kaip matosi 4 pavyzdyje, studentai bando apibrėžti kintamąjį *String[][]* tipo *products* du kartus: pirmą kartą globaliai (žr. 1 eilutė) ir antrą kartą viduje metodo *fillProducts* (žr. 4 eilutė).

2.3. Metodai

Kitas programavimo aspektas yra korektiškas metodų parametrų įvesties ir išvesties apibrėžimas. Tipinės problemos: netinkamas metodo grąžinamų parametrų apibrėžimas, bandomi grąžinti ne tokio tipo duomenys, kokie buvo apibrėžti arba nepridedama metodui parametrų grąžinimo eilutė, netinkamai kviečiami metodai programiniame kode.

Studentų darbuose įsivelia ne tik rašybos klaidų apibrėžiant duomenų tipus, bet dažnai netinkamai apibrėžiami ir metodų grąžinamo parametro duomenų tipai (žr. 5 pavyzdys). Metodai turėtų grąžinti masyvo tipo kintamuosius (šiais atvejais masyvo ženklai *[][]* nurodyti neteisingoje vietoje.)

5 pavyzdys.

```
1  public class String products[][](String ingredient,  double amount,
       String measurementUnits)
2  public String fillProducts[]
```

Java programavimo kalboje metodai gali grąžinti kokio nors duomenų tipo reikšmes (pavyzdžiui *String*, *int*), objektus ar kitokias struktūras. Tokiu atveju visos metodo dalys turi turėti nurodytą *return* reikšmę, kuri grąžinama kaip išvestis metodui. Taip pat, yra galimybė sukurti metodą, kuris negrąžintų jokios reikšmės, tuomet jis apibrėžtas reikšme *void*. Studentų darbuose aptikta klaidų, kai metodo anotacijoje kaip išvesties rezultatas aprašomas vienas duomenų tipas, tačiau prie metodo *return* reikšmės bandoma grąžinti visai kitą.

6 pavyzdys.

```
1  public static int convertToGrams ( String name, int unit){
2    ...
3  return scaleToGrams [j][1];
4  }
```

Pavyzdyje 6, eilutėje 1 matome, kad metodo grąžinimo rezultatas yra *int* duomenų tipo, tačiau kaip pateikta eilutėje 3, prie *return* reikšmės, bandomas grąžinti dvimačio masyvo *scaleToGrams* elementas, tačiau šis masyvas yra *string* duomenų tipo.

Atrasta atvejų, kai net ir gerai apibrėžtiems metodams su tinkamai apibrėžtu pavadinimu, duomenų tipu ir modifikatoriumi pamirštama pridėti *return* reikšmę, kuri nusako, koks parametras nurodytame metode bus grąžintas pagal notacijoje nurodytą duomenų tipą. Yra ir tokių atvejų, kai *void* tipo metodams studentai bando pridėti *return* sakinį.

Kadangi *Java* yra objektinio programavimo kalba, ji suteikia galimybę iškviesti atvirus metodus iš bet kurios kodo vietos. Šioje programavimo kalboje kviečiant atskirus metodus reikia pateikti konkretų įvesties duomenų skaičių pagal reikalaujamus duomenų tipus.

7 pavyzdys.

```
1  public static int convertToGrams (String name, int unit)
2  convertToGrams(products[i][0], products[i][2]);
```

Studentų darbuose pasitaiko fragmentų, kur netinkamai kviečiamas metodas *convertToGrams* (žr. 7 pavyzdys). Eilutėje 1 matome, kad šis metodas tikisi gauti *String* ir *int* tipo kintamuosius, tačiau kviečiamas metodas su dviem *int* tipo kintamaisiais, nes *products* dvimatis masyvas yra *int* duomenų tipo (žr. 2 eilutę).

Vienas iš kritinių metodų panaudojimo ir iškvietimo būdų yra, kai metodą bandoma panaudoti kaip konstantą ar kintamąjį. Tokiu atveju bandoma kviesti metodą, nepaduodant parametrų ir nepridedant skliaustelių (kaip pavyzdžiui: *System.out.println(parseString(getWeight));*) ir tuomet visos programavimo kalbos supranta, kad *getWeight* yra kintamasis o ne metodas.

2.4. Masyvai

Daugumoje programavimo kalbų yra naudojami masyvai. Tai yra dažnai naudojamas komponentas programavime ir skirtingose programavimo kalbose masyvai aprašomi skirtingai. Kritinės klaidos, susijusios su masyvais yra inicializavimas, dydžio parinkimas bei reikšmių priskyrimas.

8 pavyzdys. String[][] products = new String[50];

Kaip matome 8 pavyzdyje, dvimačiui masyvui išskiriama vienmačio *String* duomenų tipo masyvo pozicijos. Programavimo kalbos tokių apibrėžimų nesupranta ir neinterpretuoja.

9 pavyzdys.

```
1  void fillProducts(String[] arr){
2    int[][] arr2;
3    int i =0;
4    for(int j=0; j<=(arr.length-1)/3; j++){
5      arr2[j][k]=arr[i];
6      arr2[j][k]=arr[i+1];
7      arr2[j][k]=arr[i+2];
8    }
9  return arr2;
10 }
```

Taip pat, 9 pavyzdyje atvaizduojama klaida, kur iteruojama per du masyvus *arr* ir *arr2* ir bandoma priskirti skirtingų duomenų tipų reikšmes. Kaip matome eilutėje 1, *arr* masyvas yra *String* duomenų tipo, o eilutėje 2 matome, kad *arr2* dvimačio masyvo elementai yra *int* duomenų tipo. Dvimačiui masyvui *arr2* bandomi priskirti *String* tipo kintamieji iš *arr* masyvo. Tokio automatinio duomenų konvertavimo *Java* programavimo kalba nepalaiko.

2.5. Klasės ir konstruktoriai

Objektinio programavimo kalbose naudojamos klasės objektams apibrėžti. Objektams sukurti naudojami konstruktoriai, kurie pasižymi savybėmis, kad jų pavadinimas turi sutapti su klasės pavadinimu ir konstruktoriai negrąžina jokio duomenų tipo rezultato. Studentų darbuose pasitaiko klaidų, kai netinkamai klasėms aprašomi konstruktoriai.

10 pavyzdys.

```
1  public class Event
2  {
3    void Event( LinkedList<Environment> env, int x){
4      this.env = env;
5    }
6  }
```

Pateiktame 10 pavyzdyje matyti, kaip nekorektiškai naudojamas konstruktoriaus apibrėžimas. Eilutėje 1 matome, kad klasės pavadinimas yra *Event* ir ji turi konstruktorių su tokiu pačiu pavadinimu *Event* (žr. 3 eilutė), tačiau šiam konstruktoriui bandoma aprašyti *void* išvesties rezultato tipą.

Tai ne vienintelė aptikta klaida susijusi su klasės konceptais. *Java*, kaip objektinio programavimo paradigmos kalba, leidžia apibrėžti *abstract* tipo klases ir metodus. Taip pat, turi galimybę aprašyti išplečiančiąsias klases, kurios privalo įgyvendinti tėvinėje klasėje esančius metodus. Studentų darbuose pasitaiko atvejų, kai naudojama abstrakti klasė (žr. 11 pavyzdį) su apibrėžtais metodais, tačiau ją išplečianti klasė neįgyvendina visų privalomų metodų.

11 pavyzdys.

```
1  abstract class Conference()
2  {
3    abstract double countPrice(float Number)
4    {
5          ...
6    }
7    abstract int getNOP()
8    {
9          ...
10   }
11 }
12 public class Event extends Conference
13 {
14   double countPrice(float Number)
15   {
16         ...
17   }
18 }
```

Kaip matome 11 pavyzdyje, klasė *Event* išplečia abstrakčią klasę *Conference* (žr. 12 eilutę), tačiau įgyvendina tik *countPrice(float Number)* metodą. Abstrakčioje klasėje *Conference* dar yra vienas metodas *getNOP()*, kurio vaikinė klasė neįgyvendina.

Java programavimo kalboje kodo vykdymas visada prasideda ties *main* metodu, kuris turi būti vienareikšmiškai apibrėžtas tam tikra nekintančia išraiška (žr. 12 pavyzdyje).

12 pavyzdys. public static void main(String[] args)

Kompiliuojant studentų darbus pasitaiko klaida *„Main method not found“*, kas reiškia, kad trūksta pagrindinio programos metodo. Tai parodo, kad studentas nelabai supranta kokia eilės tvarka turi veikti programos ir nuo kurios vietos prasideda veikimas.

2.6. Išimčių tvarkymas

Java programavimo kalboje yra įgyvendinta išimčių tvarkymas. Tai reiškia, jei koks nors įvykis sutrikdo įprastą instrukcijų srautą, jį galima sugauti ir sutvarkyti išimčių tvarkymu. Jos apibrėžiamos privalomais raktažodžiais *try*, *catch*, taip pat, gali būti ir *finally*.

13 pavyzdys.

```
1  try(){
2    fullPrice = fullPrice + price;
3  }
4  catch (IOExeption ){
5  }
```

Kaip pabrėžta 13 pavyzdyje, studentai neužtvirtina žinių apie *try-catch* blokų panaudojimą ir savybes. *Java* programavimo kalboje *try* išimčių tvarkymo blokas aprašomas be jokių parametrų, tačiau kaip matome 13 pavyzdyje, eilutėje 1, studentai bandydavo aprašyti *try* bloką kaip metodą, kuris tikisi parametrų. Taip pat pagal *Java* programavimo kalbos reikalavimus, *catch* blokas turi turėti *exception* kintamojo parametrą bei implementaciją bloko viduje. Kaip matome 13 pavyzdyje, eilutėje 4, įvestas ir kviečiamas *IOExeption* parametras, tačiau nenurodytas joks kintamasis jam, bei *catch* bloke nėra jokių veiksmų.

2.7. Analizės rezultatai

Atlikus pavyzdinių programų analizę galima teigti, kad apibrėžtos sintaksės klaidos yra smulkios, nereikalaujančios žinių, joms užtenka įsiminimo bei atidumo. Tokio pobūdžio užduotis galima priskirti pirmajai *Bloomo* taksonomijų kategorijai *Prisiminimas*.

Sintaksės kategorijos klausimai bus susiję su teisingu raktažodžių naudojimu, gramatikos taisyklėmis programavimo kode, konstantų ir operandų naudojimu, neteisingų eilučių identifikavimu, išraiškomis naudoti standartines programavimo kalbų bibliotekas.

Kintamųjų ir masyvų naudojimas bei inicializavimas, metodų apibrėžimas yra vieni iš esminių programų sudedamųjų dalių, konceptų programuojant. Šių kategorijų klaidas galima priskirti antrajai *Bloomo* taksonomijų kategorijai *Supratimas*.

Kintamųjų ir masyvų sritims tobulinti klausimai formuojami neteisingų eilučių identifikavimu kodo fragmente, akcentuojamas masyvų dydžių ir duomenų apibrėžimas bei duomenų priskyrimas. Tam panaudojama naudotojo įvestis, kad būtų galima patikrinti ar žino kaip naudojami masyvų konceptai programavimo kontekste. Suformuluojami klausimai su teisingų/neteisingų atsakymų pasirinkimais apie metodus, kurie atspindi žinias apie metodų principus, naudojimą. Taip pat, kaip mokomoji priemonė, metodo išvesties rezultatų įvedimas. Metodo struktūros ir eiliškumo supratimui gerinti naudojamas lavinimo būdas yra sumaišytų kodo eilučių sudėliojimas teisinga tvarka.

Peržiūrėjus objektinio programavimo konceptams skirtas patikrinti užduotis yra matyti, kad studentai jau yra pažengę ir nebekartoja smulkių klaidų. Šios užduotys daugiau orientuotos patikrinti kaip studentai moka naudoti objektinio programavimo principus. Klasės, konstruktoriai, išimčių tvarkymas jau yra aukšto lygmens konceptai, kuriems panaudoti neužtenka išvengti sintaksės klaidų, reikia žinoti ne tik algoritmus, bet ir sugebėti pritaikyti žinias tam tikroje vietoje. Šio tipo klaidas priskirtume *Bloomo* taksonomijų kategorijai *Taikymas*.

Klasių, konstruktorių, išimčių tvarkymo kategorijų žinioms lavinti naudojama sumaišytų eilučių surikiavimo teisinga tvarka metodika. Taip pat neteisingų eilučių identifikavimas ar atviro pobūdžio klausimai apie tam tikrus specifinius programavimo kalbų reikalavimus.

3. Duomenų modelis

3.1. Pagrindiniai apibrėžimai

Edukacinėje platformoje naudojamas žinių ir gebėjimų lygmuo, kuris nusakomas remiantis pasirinkta taksonomija.

1 Apibrėžimas. Taksonomija. Taksonomija t priklauso taksonomijų aibei T. Ugdymo tikslų klasifikacijos pavyzdžiai gali būti *Bloomo* (žr.1.1 skyrių), *SOLO* [19] taksonomijos, kuriose yra apibrėžti lygmenys.

2 Apibrėžimas. Taksonomijos lygmuo. Taksonomijos lygmuo $tl =< t, v >$, $tl \in TL$, kur $t \in T$ yra taksonomija, o $v \in V^*$ yra taksonomijos lygmuo, kur $V^* \in V$, o $V = \{v_0, v_1, ..., v_n\}$ seka apibrėžtų lygmenų pagal taksonomiją t.

14 pavyzdys. Taksonomijos lygmenų pavyzdys.

$tl_1 =< Bloomo, Prisiminimas >$
$tl_2 =< Bloomo, Supratimas >$
$tl_3 =< Bloomo, Taikymas >$

Bloomo taksonomijai padengti galėtų būti naudojami trys taksonomijos lygmenys V^* = *{"Prisiminimas", "Supratimas", "Taikymas"}*, kaip pateikta 14 pavyzdyje. Atitinkamai pagal tai, koks edukacinis lygmuo aktualus, pasirenkamas taksonomijos lygmuo.

3 Apibrėžimas. Eilutė. Eilutė l yra simbolių seka $l =< l_1, l_2, ..., l_n >$. Kiekviena eilutė priklauso eilučių aibei L.

1 išeities kodas. $Java$ programavimo kalba parašytas kodo fragmentas, kuris ekrane atspausdina visas sąrašo arr reikšmes į naujas eilutes.

```
1  public static void main(String[] args) {
2  {
3    int len = arr.length;
4    for(int i = 0 ; i < len ; i++) {
5      System.out.println(arr[i]);
6    }
7  }
```

15 pavyzdys. 1 išeities kodo eilučių seka:
$l_1 =$"$public\ static\ void\ main(String[]\ args)$",
$l_2 =$"{"
$l_3 =$"$int\ len\ =\ arr.length;$
$l_4 =$"$for(int\ i\ =\ 0;\ i < len;\ i++)${"
$l_5 =$"$System.out.println(arr[i]);$"
$l_6 =$"}"
$l_7 =$"}"

Kaip matome 15 pavyzdyje, pagal 1 išeities kodą sugeneruojama eilučių seka L.

4 Apibrėžimas. Klasifikatorius. *Klasifikatorius* yra tam tikra tema iš temų aibės *KLASĖ*.

16 pavyzdys. *KLASĖ = {"Sintaksė", "Metodai", "Masyvai", "Klasės ir konstruktoriai", "Išimčių tvarkymas"}*

Norėdami padengti *KLASĖ* aibę, jai galime priskirti temas kaip pateikta 16 pavyzdyje.

Edukacinės platformos modelis remiasi programinio kodo fragmentais, kurie bus laikomi eilučių sekomis, o eilutės, atitinkamai, simbolių sekomis.

5 Apibrėžimas. Kodo fragmentas. Kodo fragmentas $kf =< L, klasifikatorius >$, kur $L =< l_1, l_2, ..., l_n >$, o *klasifikatorius* – generuojamo klausimo tema.

6 Apibrėžimas. Kodo fragmentų aibė. Kodo fragmentų aibė KF yra sudaryta iš kodo fragmentų kf, $KF =< kf_1, kf_2, ..., kf_n >$.

7 Apibrėžimas. Klausimo rūšis. Klausimo rūšis $c \in C$ yra generavimo algoritmo apibrėžimas ir specifika. Klausimo rūšis c nurodo kokio pobūdžio klausimas, kaip jis vertinamas, kaip pritaikomi taksonomijos lygmenys tl klausimui.

17 pavyzdys. *c = Neteisinga eilutė*

Pavyzdyje 17 pateikiama viena galima klausimo rūšis c.

8 Apibrėžimas. Klausimo rūšių aibė. Klausimo rūšių aibė $C = \{c_0, c_1, ..., c_n\}$ yra klausimo rūšies c elementų aibė, kuri apibrėžia, kokios rūšies klausimų algoritmai bus generuojami modeliuojamoje edukacinėje platformoje.

18 pavyzdys. *C = {"Neteisinga eilutė", "Sumaišytos eilutės", "Pasirinkimas, su daug teisingų", "Atviro tipo", "Pasirinkimai, su vienu teisingu"}*

Pavyzdyje 18 pateikiama pavyzdinė klausimo rūšių aibė C.

9 Apibrėžimas. Klausimo tipas. Klausimo tipas $kt =< tl, c >\in KT$, kur taksonomijos lygmuo $tl =< t, v >\in TL$, o $c \in C$ – generuojamo klausimo rūšis.

19 pavyzdys. Klausimo tipų pavyzdžiai.

```
1   kt_1 =<< Bloomo, Prisiminimas >, Neteisinga eilutė >
2   kt_2 =<< Bloomo, Supratimas >, Atviro tipo >
```

Kaip matome 19 pavyzdyje, 1 eilutėje, klausimo tipas kt_1 gali būti taksonomijos *t = Bloomo* ir lygmens *v = Prisiminimas*. Būtent šis klausimo tipas yra rūšies *c = Neteisinga eilutė*. Tačiau 2 eilutėje pateikiamas klausimo tipas kt_2, kur taksonomija *t = Bloomo* ir lygmuo *v = Supratimas* o klausimo rūšis *c = Atviro tipo*.

Klausimų generavimo algoritmui reikalingos taisyklės ir interpretacijos, pagal kurias galima sugeneruoti aibę užduočių.

10 Apibrėžimas. Taisyklė. Taisyklė $r =$ *<identifikatorius, reikšmė >* $\in R$, kur *identifikatorius* sudarytas iš reikšmių *true, false*, o *reikšmė* nusako kokios eilutės naudojamos taisyklėms.

20 pavyzdys. $r_1 =< false, length() : length >$

Pavyzdyje 20 pateikiama kaip būtų galima aprašyti taisyklę r_1. Šiuo atveju taisyklės *identifikatorius = false*, o *reikšmė = length():length*.

11 Apibrėžimas. Klausimas. Klausimas $q =< R^*, kt, kf >\in Q$ yra pagrindinis edukacinės platformos komponentas, kuris nusako kaip atrodys sugeneruotos užduotys. Kiekvienas klausimas turi unikalų kodo fragmentą $kf =< L, klasifikatorius >$, identifikuojamas pagal unikalų tipą $kt =< tl, c >$ ir jo variacijos nusakomos pagal taisyklių poaibį $R^* \in R$.

21 pavyzdys. $q_1 =< \varnothing, << Bloomo, Prisiminimas >, Pasirinkimai,\ su\ vienu\ teisingu >$ $, < \varnothing, \textit{Sintaksė} >>$

Kaip matome 21 pavyzdyje, klausimo q_1 taisyklių poaibis R^* ir kodo fragmento *kf* eilučių seka yra tuščios. Šio klausimo taksonomija *t = Bloomo*, o lygmuo *v = Supratimas*. Klausimo rūšis *c – Pasirinkimai su vienu teisingu*, o kodo fragmento *klasifikatorius = Sintaksė*.

12 Apibrėžimas. Užduotis. Užduotis $u =< kf, q >\in U$ yra viena iš sugeneruotų užduočių aibės elementų ir yra sudaryta iš modifikuoto kodo fragmento kf bei klausimo q.

Klausimų generavimo algoritmams naudojami tam tikri apibrėžti kintamieji ir operatoriai.

13 Apibrėžimas. Kintamasis *atsakymas*. Kintamasis *atsakymas* yra eilutės tipo kintamasis, kuris gaunamas vartotojui atsakinėjant į klausimo rūšies *c = Atviro tipo* užduotis. *Atsakymas* yra naudotojo įvestas tekstas.

14 Apibrėžimas. Kintamasis *m*. Kintamasis *m* yra sveiko skaičiaus tipo reikšmė, skirta atstumo matavimo tarp dviejų tekstinių simbolių aibių rezultatui aprašyti.

22 pavyzdys. $m = LevenshteinDistance(r, atsakymas)$

Pavyzdyje 22 pateikiama kaip būtų galima naudoti kintamąjį *m*.

15 Apibrėžimas. Kintamasis *skaitiklis*. Kintamasis *skaitiklis* yra sveiko skaičiaus tipo reikšmė, skirta nustatyti iš kiek elementų susidarys aibės rinkiniai.

16 Apibrėžimas. Operatorius <. Operatorius $< TL \times TL \rightarrow \{true, false, undefined\}$ įvertina ar vienas taksonomijos lygmuo yra žemesnis už kitą. Operatorius grąžina *undefined*, jeigu lygmenys priklauso skirtingoms taksonomijoms.

23 pavyzdys. Taksonomijų palyginimo pavyzdys.

```
1   tl_1 =< Bloomo, Prisiminimas > tl_2 =< Bloomo, Taikymas >, tl_1 < tl_2 → true
2   tl_3 =< Bloomo, Supratimas > tl_4 =< SOLO, Supratimas > tl_3 < tl_4 → undefined
```

Kaip matome 23 pavyzdyje, 1 eilutėje, lyginant *Bloomo* taksonomijos lygmenis v_1 = *Prisiminimas* ir $v_2 = Taikymas$, gaunamas rezultatas *true*, kas reiškia, kad lygmuo tl_1 yra žemesnis nei tl_2. Tačiau 2 eilutėje lyginant taksonomijų lygmenis gaunamas rezultatas yra *undefined*, nes taksonomijos lygmuo tl_3 priklauso *Bloomo* taksonomijai, o tl_4 *SOLO*.

17 Apibrėžimas. Operatorius ∪. Operatorius aibių sąjunga ∪ yra kelių aibių sujungimas, bet leidžia į aibę pridėti ir vieną elementą.

24 pavyzdys. $U^* \cup u$

Pavyzdyje 24 pateikta kaip užduočių aibei U^* pridedamas užduoties elementas u.

18 Apibrėžimas. Operatorius $\leftarrow$. Operatorius $\leftarrow$ naudojamas algoritmų pavyzdžiuose ir reiškia *priskyrimą* arba *pridėjimą*.

25 pavyzdys. Operatoriaus $\leftarrow$ naudojimo pavyzdys.

```
1  U* ← ∅
2  U* ← U* ∪ u
```

Kaip matome 25 pavyzdyje, 1 eilutėje, aibė U^* priskiriama tuščiai aibei $\varnothing$ naudojant priskyrimo operatorių $\leftarrow$. Eilutėje 2 aibei U^* pridedamas elementas u.

19 Apibrėžimas. Operatorius $\varnothing$. Operatorius $\varnothing$ naudojamas kintamuosius apibrėžti kaip tuščią aibę.

Pateiktame 25 pavyzdyje, 1 eilutėje, užduočių aibė U^* priskiriama tuščiai aibei $\varnothing$.

20 Apibrėžimas. Operatorius |x|. Operatorius |x| grąžina sveiko skaičiaus reikšmę, nurodydamas kiek elementų yra elementų aibėje *x*, sekoje arba kiek simbolių yra eilutėje *l*.

26 pavyzdys. *A = {1, 2, 3};*

Pavyzdyje 26 pateikta aibė iš trijų elementų, tai jos ilgis $|A| = 3$.

21 Apibrėžimas. Kintamasis *rezultatas*. Sveiko skaičiaus kintamasis *rezultatas* nusako, kiek taškų procentaliai skiriama naudotojui už užduočių aibės U^* atliktas užduotis.

27 pavyzdys. $U^* = \{\{String, int, double\}, \{String, double, char\}, \{String, int, char\}\}$

Turime tokią užduočių aibę U^*, kaip pateikta 27 pavyzdyje. Šios užduočių aibės U^* rūšis *v = Pasirinkimai su vienu teisingu*, taksonomija *t = Bloomo* su taksonomijos lygmeniu *v = Supratimas*. Tokiu atveju šioje užduočių aibėje yra trys pasirinkimo varianta iš kurių du neteisingi variantai ir visose užduotyse po vieną teisingą pasirinkimą *String*. Jei naudotojas visus teisingus atsakymus pasirenka, galima sakyti, kad *rezultatas = 100%*. Atitinkamai, jei tik du teisingi *rezultatas = 66,6%*, jei teisingas vienas *rezultatas = 33,3%*. Jei nepasirinko nei vieno atsakymo teisingai, tuomet *rezultatas = 0%*.

22 Apibrėžimas. Operatorius $\sim$. Operatorius $\sim$ palygina du vienodo tipo elementus *x* ir *y* ir grąžina rezultatą *true*, jei jie turi sutapimų ir *false*, jei atitikimų nerasta.

28 pavyzdys. Operatorius $\sim$ pavyzdys.

```
1  "atviras" ~ "ras" → true
2  "atviras" ~ "mėnuo" → false
```

Eilutėje 1, 28 pavyzdyje matome kaip palyginamos eilutės *atviras* ir *ras* ir gaunamas rezultatas *true*, o 2 eilutėje objektai neturi jokių atitikmenų ir gaunamas neigiamas rezultatas *false*.

23 Apibrėžimas. Operatorius : . Operatorius : naudojamas klausimo rūšiai *c = Neteisinga eilutė* ir taisyklėse *r* naudojama reikšmėje *r.reikšmė*. Taisyklės *r.reikšmė* simbolių eilutėje randame operatorių : . Kadangi, klausimo rūšiai *c = Neteisinga eilutė* naudojame raktažodį *"kokį raktažodį pakeisti"* ir *"į ką pakeisti"*, tai taisyklės *r.reikšmė* dalis iki operatoriaus : yra raktažodis *"kokį raktažodį pakeisti"* o likusi eilutės *r.reikšmė* po operatoriaus : yra *"į ką pakeisti"*.

29 pavyzdys. $r_1 =< false, length : length() >$

29 pavyzdyje pateikta taisyklė r_1 ir kaip matome, jos reikšmė *r.reikšmė = length:length()* išskirta su operatoriumi : . Šiuo atveju *length* yra pervardinimo raktažodis, o *length()* nauja reikšmė.

24 Apibrėžimas. Operatorius ·. Operatorius · nusako kokį objekto vidinį vaikinį konkretų elementą naudojame.

30 pavyzdys. Operatorius · pavyzdys.

```
1  kt = < <Bloomo, Prisiminimas>, Neteisinga eilutė>
2  kt.tl.v = Prisiminimas
```

Turime klausimo tipo elementą kaip 30 pavyzdyje, 1 eilutėje, kur $kt =< tl, c >$, o $tl =< t, v >$. Tokiu atveju, jei norime naudoti konkretų vaikinį elementą, taksonomijos lygmenį v, pasinaudoję · operatoriumi, galime pasiekti (žr. 2 eilutę).

Klausimų generavimui palengvinti panaudojamos tam tikros matematinės arba aibių ir sekų funkcijos.

25 Apibrėžimas. Funkcija *replace(l, r.reikšmė)*. Šios funkcijos įvestis yra taisyklės *r.reikšmė* ir viena eilutė *l*. Ši funkcija, pagal taisyklės *r.reikšmė* esantį operatorių : , atrenka pervardinimo raktažodį eilutėje *l* ir pakeičia jį pervardinimo reikšme.

31 pavyzdys. Funkcijos *replace(l, r.reikšmė)* pavyzdys.

```
1        r_1 =< false, length : length() >
2        l = "int arrayLength = arr.length;"
3        replace(l, r.reikšmė) → "int arrayLength = arr.length();"
```

Pavyzdyje 31 matome, kaip panaudojama *replace* funkcija. Eilutėje 1 apibrėžiame taisyklę r_1, o eilutėje 2 eilutę *l*. Su tokiais įvesties parametrais atlikus *replace* funkciją, eilutėje *l* pakeisime reikšmę *length* į *length()*. Rezultatą matome eilutėje 3.

26 Apibrėžimas. Funkcija *GetPermutations(list, skaitiklis)*. Šios funkcijos įvestis yra aibė *list* ir pasirinktas skaičius *skaitiklis*. Pagal šiuos įvesties parametrus iš aibės *list* surenkami visi galimi poaibiai, kurie kiekvienas turi po *skaitiklis* elementų.

32 pavyzdys. *GetPermutations({1, 2, 3}, 2)*

Pateiktame 32 pavyzdyje parodoma kaip būtų galima panaudoti *GetPermutations* funkciją ir jai perduoti parametrus. Pagal šiuos parametrus ši funkcija grąžintų rezultatą naują aibę $G = \{(1,1)(1,2)(1,3)(2,3)\}$.

27 Apibrėžimas. Funkcija *LevenshteinDistance(r, atsakymas)*. Šios funkcijos įvestis yra taisyklė *r* ir naudotojo įvesta eilutė *atsakymas*. Pagal šiuos įvesties parametrus *LevenshteinDistance* funkcija apskaičiuoja atstumą tarp dviejų tekstinių simbolių aibių *r.reikšmė* ir *atsakymas*.

33 pavyzdys. Funkcijos *LevenshteinDistance* pavyzdys.

```
1  r =< true, "labas" >
2  atsakymas = "labas"
3  m = LevenshteinDistance(r, atsakymas) → 0
4  atsakymas = "lapas"
5  m = LevenshteinDistance(r, atsakymas) → 1
```

Pateiktame 33 pavyzdyje, 1 eilutėje apibrėžiama taisyklės reikšmė *r.reikšmė = labas* ir 2 eilutėje kintamasis *atsakymas = labas*. 3 eilutėje su šiais parametrais panaudojama *LevenshteinDistance* funkcija ir šiuo atveju, kintamajam *m* priskiriamas atstumas lygus nuliui. Tačiau 4 eilutėje priskyrus kitą reikšmę kintamajam $atsakymas = lapas$ ir vėl pasinaudojus *LevenshteinDistance* funkcija, atstumo kintamajam *m* priskiriama reikšmė yra vienas, nes tarp simbolių aibių *labas* ir *lapas* yra vienas neatitinkantis simbolis (trečioje pozicijoje).

28 Apibrėžimas. ***Dekarto sandauga*.** Dekarto sandauga (angl. *Cartesian product*) [16] iš kelių skirtingų aibių elementų sudaro visų galimų porų aibę.

34 pavyzdys. *A = {1, 2}; B = {3, 4}*
A × B = {1, 2} × {3, 4} = {(1, 3), (1, 4), (2, 3), (2, 4)}

Kaip matome 34 pavyzdyje, sukuriama $A \times B$ aibių sandauga ir rezultate gaunasi porų aibė $A \times B = \{(1,3),(1,4),(2,3),(2,4)\}$.

29 Apibrėžimas. Funkcija *Min(list)*. Tai yra funkcija, kurios įvestis yra sveikų skaičių sąrašas *list* ir ši funkcija grąžina patį mažiausią skaičių iš aibės *list*.

35 pavyzdys. *Min({1, 2, 3})*

Pavyzdyje 35 pateikta kaip galima panaudoti funkciją *Min*. Šiuo atveju ši funkcija grąžintų skaičių vienas, nes tai yra mažiausias skaičius iš gautos aibės.

3.2. Užduočių aibės sukūrimas edukacinėje platformoje

Užduotys modeliuojamoje edukacinėje platformoje generuojamos pagal tam tikrą specifiką.

1 algoritmas. Užduočių generavimo algoritmo pavyzdys.

Įvestis: $q =< R^*, kt, kf >, kt =< tl, c >, tl =< t, v >, r =< identifikatorius,$ ***reikšmė*** $>\in R, kf =< L, klasifikatorius >$
Išvestis: U^*

1: **for each** *r in* R^* **do**
2: $u = q \leftarrow r$
3: $U^* \leftarrow U^* \cup u$
4: **end for**
5: **return** U^*

Algoritme 1 matome, kaip sugeneruojama konkreti užduočių aibė U^*. Algoritmas gauna konkretų klausimą q, taisyklių aibę $R^* \in R$, klausimo tipą kt ir kodo fragmento kf, visų pirma iteruojama per visą jo turimą taisyklių poaibį R^* (žr. 1 eilutę). Atitinkamai pagal kiekvieną iš taisyklių r modifikuojamas klausimas ir 2 eilutėje jis priskiriamas naujai užduočiai u. Kiekviena nauja užduotis u pridedama prie bendros užduočių aibės U^* konkrečiam klausimui q ir vėl iteruojama per taisyklių aibę R^*.

3.2.1. Programinio kodo fragmentų parinkimas užduotims

Modeliuojamoje edukacinėje platformoje kai kurių klausimų generavimas remiasi programinės kalbos kodo fragmentais. Perdarant kodo fragmentus galima taikyti įvairias taisykles, todėl parinkti fragmentai turi turėti įvairių savybių ir aspektų. Kuo daugiau jame įvairių programavimo aspektų, tuo daugiau taisyklių jam galima pritaikyti.

2 išeities kodas. *Java* programavimo kalba parašytas metodas, kuris išsaugo duomenis į duomenų bazę.

```
1  public void AddQuestion(QuestionDto question)
2  {
3    if (question.Name.length < 3)
4    {
5      throw new ValidationException("Name must be at least 3 characters
           .");
6    }
7    IEducationDal dal = new EducationContext();
8    int questionId = dal.AddQuestion(new Question()
9    {
10     TypeId = question.Type.TypeId,
11     RuleId = 1,
12     QuestionTitle = question.Name
13   });
14   dal.AddLines(_mapper.LinesFromDtoToDB(question.Lines, questionId));
15 }
```

Kodo fragmento pavyzdys pateikiamas 2 išeities kode. Šiame išeities kode yra įvairių programavimo aspektų, tokių kaip kintamieji, ciklai. Pavyzdžiui, pasižiūrėję į 1 išeities kodo eilutę, matome kaip yra apibrėžiamas metodas, kokie naudojami prieinamumo raktažodžiai, kaip ir kokie duomenų tipai perduodami metodui. Tuo tarpu 3 eilutėje naudojamas loginis tikrinimas *if*. Eilutėse 7 ir 8 matosi kaip kuriami ir aprašomi objektai, jų inicializavimas. Taip pat, 8 eilutėje naudojamas vienas iš duomenų tipų *int*, o eilutėse 10, 11 ir 12 vyksta reikšmių priskyrimas kintamiesiems. Eilutėse 8 ir 14 atvaizduojamas teisingas metodų kvietimas su parametrais.

36 pavyzdys. *kt = < <Bloomo, Prisiminimas>, Neteisinga eilutė>*

Pavyzdyje 36 apibrėžtas klausimo tipas *kt*, kur taksonomijos lygmuo *tl = <Bloomo, Prisiminimas>* yra pats žemiausias. Klausimo rūšis *c = Neteisinga eilutė* nusako, kad taisyklės turi būti sudarytos tokios, kad tam tikras eilutes iš klausimo eilučių sekos L pakeistų į neteisingas išraiškas. $R^* = \{< false, length : length() >, < false, length : Length >, < false, int : Integer, int : integer >\}$.

Šis taisyklių sąrašas sudarytas atsižvelgiant kokias programavimo rašybos klaidas daro studentai. Taip pat parinkta taip, kad būtų atitikmenų pavyzdiniame kodo fragmente (žr. 2 išeities kodas). Taisyklių sąrašo dydis nėra svarbu, tik išlieka tokia savybė, kad kuo daugiau taisyklių aprašyta, tuo didesnę skirtingų užduočių aibę galima sudaryti.

3.2.2. Užduočių įvertinimai

Tinkamas kompetencijų lygių išskyrimas gali padėti nustatyti užduočių filtravimo kontekstą. Vertinime būtina atsižvelgti kokios taksonomijos lygmens užduotys yra vertinamos ir pritaikyti atitinkamai proporcingą užduoties sudėtingumui vertinimo algoritmą.

Modeliuojamoje edukacinėje sistemoje taikoma taškų sistema. Už pirmojo taksonomijos lygmens *Prisiminimas* teisingai atsakytus klausimus skiriama po vieną tašką, už antrojo *Supratimas* lygmens klausimų teisingus atsakymus skiriama du taškai ir atitinkamai trečiojo lygmens *Taikymas* klausimai vertinami trim balais. Už klaidas taškai neatimami.

37 pavyzdys. Turime sugeneruotą užduočių aibę U^*, kur vienas iš klausimų q_n yra klausimo tipo *kt = < <Bloomo, Supratimas>, Neteisinga eilutė>*, kur taksonomijos *t = Bloomo* lygmuo *v =*

Supratimas yra laikomas kaip antrasis sudėtingumo lygmuo. Klausimo rūšiai *c = Neteisinga eilutė* teisingai identifikavus neteisingas eilutes naudotojas gaus viso du taškus, jei identifikuojama tik viena neteisinga eilutė, skiriamas vienas taškas iš galimų dviejų ir jei visos pasirinktos eilutės ne tos, kurios modifikuotos – naudotojui taškai neskiriami.

4. Algoritmai

4.1. Kodo iškraipymas neteisingos eilutės radimui

Užduočių generavimas yra priklausomas nuo klausimo $q =< R^*, kt, kf >$, jo tipo $kt =< tl, c >$, individualaus kodo fragmento $kf =< L, klasifikatorius >$ ir taisyklių sąrašo $R^* \in R$, sugeneruojamo pagal klausimo tipą kt.

Panagrinėkime pavyzdį, kaip sugeneruojama užduočių aibė tokiam klausimui *q = <R^*, kt, kf>*, kur klausimo q kodo fragmentas *kf = <L, klasifikatorius>*, kur eilučių aibė $L =< l_1, l_2, ..., l_n >$ yra analogiška kaip 2 išeities kodas, tad $L =<$*"public void AddQuestion(QuestionDto question)", "{", ... , "}"*$>$. Tuo tarpu, šiam klausimui q nustatytas klausimo tipas $kt =< tl, c >$, kur klausimo rūšis *c = Neteisinga eilutė*, o taksonomijos lygmuo $tl =< t, v >$.

---

2 algoritmas. Algoritmo pavyzdys klausimui, kurio rūšis *„Neteisinga eilutė“*.

Įvestis: $q =< R^*, kt, kf >\in Q, R^* \in R, kf =< L, klasifikatorius >, kt =<< t, v >$ *, Neteisinga eilutė>*

Išvestis: $U^* \in U$

1: $U^* \leftarrow \varnothing$
2: **for each** *r in* R^* **do**
3: $u =< kf, q >$
4: $u.kf \leftarrow q.kf$
5: $u.q \leftarrow q$
6: **for each** l *in u.kf.L* **do**
7: **if** $l \sim$ *r.reikšmė* **then**
8: $l = replace(l,$ *r.reikšmė*$)$
9: $U^* \leftarrow U^* \cup u$
10: break;
11: **end if**
12: **end for**
13: **end for**
14: **return** U^*

Algoritmas 2 demonstruoja kaip yra generuojama užduočių aibė $U^* = \{u_0, u_1, ..., u_n\}$, kur $u =< kf, q >$ pagal klausimo q tipą $kt =<< t, v >,$ *Neteisinga eilutė* $>$. Visų pirma, 2 algoritmo 1 eilutėje, sukuriama nauja tuščia užduočių aibė U^*. Tuomet, 2 algoritmas pradeda iteruoti per visas klausimo taisykles $r_n \in R^*$ (žr. 2 eilutę), kurios atrinktos pagal *identifikatorių = false* ir klausimo tipą kt. Algoritmo 3 eilutėje apsibrėžiame naują kintamąjį $u =< kf, q >$, kur toliau šios užduoties kodo fragmentui $u.kf$ priskiriamas einamojo klausimo kodo fragmentas kf ir nustatomas užduoties klausimas $u.q$. Pateiktoje 6 eilutėje pradedame iteruoti per užduoties kodo fragmento eilučių seką $u.kf.L$ ir toliau su kiekviena kodo fragmento eilute $l \in u.kf.L$ bus atliekami veiksmai. Toliau 7 eilutėje matome, kad bandoma patikrinti, ar užduoties kodo fragmento eilutė l turi atitikmenų su taisyklės reikšme *r.reikšmė*. Jei atitikmenų rasta, atliekama *replace* funkcija (žr. 8 eilutę) ir 9 eilutėje į užduočių aibę U^* pridedame naujai sugeneruotą užduotį u. Nustojama iteruoti per kodo fragmento eilutes $u.kf.L$ (10 eilutė) ir tęsiame iteracijas per taisyklių poaibį R^*.

Naudotojas šiam klausimui turi apibrėžęs klausimų aibę $R^* = \{r_0, r_1, ..., r_n\}$, kur *r= <identifikatorius, reikšmė>* kaip 38 pavyzdyje.

38 pavyzdys. Taisyklių aibės R^* pavyzdys.

```
1  r1 =< false, length() : length >
2  r2 =< false, length() : Length() >
```

Kaip matome 38 pavyzdyje, taisyklės turi *identifikatorių = false*, kas reiškia, kad generuojant užduotis abi šios taisyklės klausimo rūšiai *c = Neteisinga eilutė* bus naudojamos pakeičiant teisingas kodo fragmento dalis į neteisingas. Kaip matome 38 pavyzdyje, 1 eilutėje, tai *String* duomenų tipo ilgio raktažodį *length()* pakeisime į *length*. Eilutėje 2 nurodytoje taisyklėje galime pamatyti, kad raktažodį *length()* pakeisime į *Length()*.

39 pavyzdys. Sugeneruota užduotis u_1.

```
1  public void AddQuestion(QuestionDto question)
2  {
3    if (question.Name.length < 3)
4    {
5      new ValidationException("Name must be at least 3 characters.");
6    }
7    IEducationDal dal = new EducationContext();
8    int questionId = dal.AddQuestion(new Question()
9      {
10       TypeId = question.Type.TypeId,
11       RuleId = 1,
12       QuestionTitle = question.Name
13     });
14   dal.AddLines(_mapper.LinesFromDtoToDB(question.Lines, questionId));
15 }
```

40 pavyzdys. Sugeneruota užduotis u_2.

```
1  public void AddQuestion(QuestionDto question)
2  {
3    if (question.Name.Length() < 3)
4    {
5      new ValidationException("Name must be at least 3 characters.");
6    }
7      IEducationDal dal = new EducationContext();
8      int questionId = dal.AddQuestion(new Question()
9    {
10     TypeId = question.Type.TypeId,
11     RuleId = 1,
12     QuestionTitle = question.Name
13   });
14   dal.AddLines(_mapper.LinesFromDtoToDB(question.Lines, questionId));
15 }
```

Kaip matome 39 ir 40 pavyzdžiuose, sugeneruotos dvi užduotys u_1 ir u_2. Jos gautos klausimo pirminiame kodo fragmente pakeitus raktinius žodžius pagal raktažodžius iš taisyklių aibės (žr. 38 pavyzdyje). Pavyzdžiuose 39 ir 40 pateikiami beveik identiški kodo fragmentai, tačiau kiekviename iš jų yra po vieną neteisingą eilutę (39 pavyzdyje 3 eilutė ir 40 pavyzdyje 3 eilutė).

Klausimo rūšies *c* = *Neteisinga eilutė* užduotys galėtų būti vertinamos pagal tai, kiek užšifruotų neteisingų eilučių naudotojas rado. Jei rado visas, vartotojas gauna visus galimus taškus, atitinkamai mažinant taškų skaičių, kiek eilučių neaptiko. Didinant taksonomijos lygmenis į užduoties kodo fragmentus vertėtų įdėti daugiau neteisingų kodo eilučių.

4.2. Pasirinkimų kiekio adaptavimas pagal lygmenį

Dar vienas iš algoritmų yra vartotojo pasirinkimų adaptavimas pagal lygmenis. Toks algoritmas adaptyviai prisitaikytų prie taksonomijos lygmens v ir pagal tai nuspręstų kokią pasirinkimų aibę sudaryti. Pagrindinė algoritmo idėja, kad vartotojui pateiktų aibę pasirinkimo elementų, kur dalis yra teisingi, kita dalis neteisingi ir naudotojas turėtų identifikuoti tinkamus pasirinkimus. Tokia aibė generuojama *Dekarto sandaugos* pagalba.

Kiekviena klausimui $q =< R^*, kt, null >$, kur klausimo kodo fragmentas $kf = null$ ir klausimo tipas *kt* = *<tl, Pasirinkimas su daug teisingų>*, kur rūšis $c =$ *Pasirinkimas su daug teisingų*, pagal taisyklių poaibį $R^* \in R$, sugeneruojami visi įmanomi pasirinkimų rinkiniai. Tokių užduočių generavimo principą galima pamatyti 3 algoritme.

3 algoritmas. Algoritmo pavyzdys klausimo rūšiai *„Pasirinkimai su daug teisingų“*.

Įvestis: $q =< R^*, kt, kf >\in Q, R^* \in R, kf =< \varnothing, klasifikatorius >, kt =<< t, v >$, *Pasirinkimai su daug teisingų* $>$

Išvestis: $U^* \in U$

1: $U^* \leftarrow \varnothing$
2: $G = GetPermutations(R_1^*, skaitiklis_1)$
3: $S = GetPermutations(R_2^*, skaitiklis_2)$
4: **for each** *g in G* **do**
5: **for each** *s in S* **do**
6: $u =< kf, q >$
7: $u.q \leftarrow q$
8: $u.kf \leftarrow g$
9: $u.kf \leftarrow s$
10: **end for**
11: $U^* \leftarrow U^* \cup u$
12: **end for**
13: **return** U^*

Algoritme 3 matome, kad pagrindiniai įvesties parametrai yra klausimas $q =< R^*, kt, kf >$, kur klausimo q kodo fragmentas $kf =< L, klasifikatorius >$, kur eilučių aibė $L = \varnothing$ tuščia. Tuo tarpu, šiam klausimui q nustatytas klausimo tipas $kt =< tl, c >$, kur klausimo rūšys c = *Pasirinkimai su daug teisingų*, o taksonomijos lygmuo $tl =< t, v >$. Taip pat pridedamas taisyklių poaibis $R^* \in R$. Visų pirma, kaip matome 1 eilutėje, sukuriama tuščia užduočių aibė U^*. Toliau, 2 ir 3 eilutėse kviečiama *GetPermutations(list, skaitiklis)* funkcija ir jai pateikiami parametrai $list = R_1^*$ ir $skaitiklis_1$, kitu atveju $list = R_2^*$ ir $skaitiklis_2$. Pirmasis taisyklių poaibis $R_1^* \in R^*$, o atitinkamai taisyklių poaibis R^* priklauso pradiniam klausimui q ir R_1^* simbolizuoja visas taisykles r_n, kurios nusako, kokios klausimo q pasirinkimo galimybės yra teisingos. Ši savybė apibrėžiama taisyklėse *identifikatorius* = *true*. R_2^* taisyklių poaibis nusako visus galimus variantus, kurie priklauso einamajam klausimui q, kaip neteisingi pasirinkimo variantai. Taisyk-

lėse tai atspindi *identifikatorius = false*. Algoritmo $skaitiklis_1$ nusako kiek teisingų pasirinkimo taisyklių r paimti generuojant užduotį iš teisingų klausimo pasirinkimo taisyklių aibės R_1^*. Atitinkamai, metodo 3 įvesties skaitliukas $skaitiklis_2$ nurodo, kiek reikia paimti neteisingų taisyklių r iš taisyklių poaibio R_2^*. Taip sugeneruojamos dvi aibės G ir S, kurios sudarytos iš poaibių taisyklių, kur kiekviename elemente yra po $skaitiklis_1$ ir $skaitiklis_2$ taisyklių. Toliau 4 ir 5 eilutėse pradedama iteruoti per abi gautas naujas aibes G ir S ir priskiriamos sukurtos naujos užduoties kodo fragmento eilutės $u.kf$, *Dekarto sandaugos* principu užpildomos po vieną elementą g ir s iš aibių G ir S (žr. 8 ir 9 eilutes). Baigus iteruoti per aibę S užduotis u priskiriama aibei U^* ir toliau tęsiasi iteracija per aibę G.

41 pavyzdys. Klausimo pavadinimas: Kokie *Java* programavimo kalboje naudojami duomenų tipai?

```
1  q =< {< true, integer >, < true, string >, < false, for >, < false, public >}, << Bloomo,
       Prisiminimas >, Pasirinkimai, su daug teisingų >, ∅ >
2  skaitiklis_1 = 1
3  skaitiklis_2 = 2
4  R_1^* = < true, integer >, < true, string >
5  R_2^* = < false, for >, < false, public >
6  U^* = {(integer, for, public), (string, for, public)}
```

Pavyzdyje 41 pateikiamas trumpas aprašymas, kaip yra automatiškai sugeneruojama užduotis. Klausimas q yra rūšies c = *Pasirinkimai su daug teisingų*, taksonomijos lygmuo $tl =< Bloomo, Prisiminimas >$, o kodo fragmentas $kf =< \varnothing$, *Sintaksė* $>$, kur *klasifikatorius = Sintaksė*, o eilučių aibė $L = \varnothing$ tuščia. Pateikiama taisyklių aibė $R* =$ *{<true, integer>, <true, string>, <false, for>, <false, public>}*. R^* suskirstomas į du poaibius R_1^* ir R_2^* pagal *identifikatorių* (žr. 4 ir 5 eilutes). Apsibrėžiame kintamuosius $skaitiklis_1 = 1$ ir $skaitiklis_2 = 2$ (žr. 41 pavyzdžio 2 ir 3 eilutes). Pasinaudodami šiais parametrais ir 3 algoritmu, 6 eilutėje sugeneruota užduočių aibė U^* naudotojams.

Šios rūšies užduotys gali būti vertinamos pagal tai, kiek teisingų/neteisingų teiginių naudotojas atrado teiginių aibėje. Jei identifikavo visas teisingas eilutes, tuomet skiriama maksimali taškų suma, jei ne, ji proporcingai mažinama atsižvelgiant iš kokio dydžio aibės elementų naudotojas turėjo atrasti teisingus.

4.3. Atviro tipo klausimo atsakymo vertinimas

Modeliuojamoji edukacinė sistema pasižymi *Atviro tipo* užduočių generavimo algoritmu. Pagrindinis šių klausimų išskirtinumas yra tinkamai įvertinti naudotojo įvestą atsakymą.

Algoritme 4 vaizduojama, kaip įvertinama *atsakymas* klausimo *Atviro tipo*. Matome, kad šiam aloritmui įvesties parametrai yra klausimas $q =< R^*, kt, kf >$ ir naudotojo įvestas tekstas *atsakymas*. Šiam algoritmui kodo fragmento aibė $L = \varnothing$ yra tuščia. Esminis dalykas yra taisyklių poaibis R^*. Šiam vertinimo algoritmui taisyklės $r \in R^*$ turi savybę *identifikatorius = true* ir taisyklių *r.reikšmė* suprantama kaip teisingas atsakymas į *Atviro tipo* klausimus. Kadangi naudojame ne vieną taisyklę, o jų poaibį, tai reiškia, kad gali būti ne vienas tinkamas atsakymas. Kaip matome 1 eilutėje, iteruojama per visą taisyklių poaibį R^* ir kiekvienos taisyklės reikšmė *r.reikšmė*, pasinaudojant *Levenšteino atstumo* skaičiavimo algoritmu, lyginama su vartotojo įvestu $atsakymu$. Vartotojo įvestas atsakymas yra lyginamas su kiekvienos taisyklės *r.reikšmė* reikšme iš taisyklių poaibio R^* (žr. 2 eilutę). Kaip matome 3 eilutėje, šis atstumas priskiriamas į atstumų aibę *M* ir

4 algoritmas. Atsakymo įvertinimo algoritmas klausimo rūšiai *„Atviro tipo“*.

Įvestis: $q =< R^*, kt, kf >\in Q, R^* \in R, kf =< \varnothing, klasifikatorius >, kt =<< t, v >, Atviro\ tipo >, atsakymas$

Išvestis: m

1: **for each** *r in* R^* **do**
2: $m \leftarrow LevenshteinDistance(r, atsakymas)$
3: $M^* \leftarrow M \cup m$
4: **end for**
5: **return** $Min(m)$

algoritmas grąžina mažiausią atstumą *m* iš aibės *M* (žr. 5 eilutę). Tai reiškia, kad bus grąžintas mažiausias atstumas tarp naudotojo įvesto atsakymo *atsakymas* ir vienos iš taisyklių *r.reikšmės*.

3 išeities kodas. *Java* programavimo kalba parašytas kodo fragmentas.

```
1  public class Program
2  {
3    public static void main(String[] args) {
4    {
5      System.out.println("Press any key to exit.");
6    }
7  }
```

42 pavyzdys. Klausimo pavadinimas: Kokį tekstą išspausdins kodo fragmentas?
$atsakymas = Press\ any\ key\ to\ exit$
$R^* = \{< true, Press\ any\ key\ to\ exit >, < true, Press\ any\ key\ to\ exit. >\}$
$M = \{1, 0\}$

Pavyzdyje 42 pateikiama kaip veikia 4 algoritmas. Matome, kad klausimo pavadinimas *Kokį tekstą išspausdins kodo fragmentas?* ir pagal pateiktą išeities kodą 3 atsakymas turėtų būti *Press any key to exit.*, tačiau kurdamas klausimą vartotojas įvedė taisykles r_1 = <*true, "Press any key to exit"*> ir r_2 = <*true, "Press any key to exit."*>. Kiekviena ši taisyklės *r.reikšmė* palyginama su vartotojo įvestu *atsakymu* ir *Levenšteino atstumo* atstumų aibėje M išsaugomi atstumai tarp šių eilučių. Rezultate gauta aibė *M* sudaryta iš dviejų sveikų skaičių vienas ir du. Tai reiškia, kad naudotojo įvestas atsakymas *atsakymas* = *Press any key to exit* su taisyklės r_1 reikšme turėjo vieną neatitikimą, o su taisyklės r_2 reikšme sutapo identiškai.

Tokių klausimų vertinimo algoritmas susideda iš to, koks mažiausias neatitikimo atstumas. Jei *Levenšteino atstumo* atstumas aibėje M yra bent kartą lygus nuliui, tai reiškia, kad vartotojo įvestas atsakymas sutapo su bent vienos taisyklės $r \in R^*$ reikšme. Tokiu atveju galima skirti tris taškus. Jei mažiausias atstumas m aibėje M yra vienas – vartotojas gauna tik du taškus, nes jo pateiktas atsakymas turėjo vieną neatitikimą su kažkuriuo galimu atsakymo variantu. Atitinkamai, jei mažiausias atstumas lygus du, tai neatitikimai rasti net du ir skiriamas bent vienas taškas. Jei neatitikimai viršija du, tuomet vartotojas jau nebegauna taškų.

4.4. Klausimų be kodo fragmentų generavimas

Kai kurie klausimų algoritmai nenaudoja kodo fragmentų. Klausimo rūšies *c = Pasirinkimai su vienu teisingu* užduočių generavimui nenaudojami kodo fragmentai, nes šio algoritmo pagrin-

dinė idėja yra pagal tam tikrą taksonomijos lygį sugeneruoti aibę pasirinkimų, kurioje tik vienas iš atsakymų būtų teisingas. Svarbu, kad šios aibės būtų sudarytos iš daugiau nei dviejų elementų. Kuriant tokios rūšies klausimus, svarbu yra apibrėžti tinkamai taisykles $r \in R^*$, kurių *identifikatorius* būtų *true* arba *false*. Dekarto sandaugos pagalba sudaromi rinkiniai, kur viena reikšmė yra teisinga ir keletas neteisingų. Naudotojas turi identifikuoti tinkamą vieną reikšmę. Neteisingų reikšmių skaičius priklauso nuo taksonomijos lygmens tl.

5 algoritmas. Algoritmas klausimo rūšiai *„Pasirinkimai su vienu teisingu“*.

Įvestis: $q =< R^*, kt, kf >\in Q, R^* \in R, kf =< \varnothing, klasifikatorius >, kt =<< t, v >, Pasirinkimai\ su\ vienu\ teisingu >$

Išvestis: $U^* \in U$

1: $U^* \leftarrow \varnothing$
2: $S = GetPermutations(R_1^*, skaitiklis)$
3: **for each** *s in S* **do**
4: **for each** *r in* R_2^* **do**
5: $u =< kf, q >$
6: $u.q \leftarrow q$
7: $u.kf \leftarrow r$
8: $u.kf \leftarrow s$
9: **end for**
10: $U^* \leftarrow U^* \cup u$
11: **end for**
12: **return** U^*

Algoritme 5 matome, kad pagrindiniai įvesties parametrai yra klausimas $q =< R^*, kt, kf >$, kur klausimo q kodo fragmentas $kf =< L, klasifikatorius >$, eilučių aibė $L = \varnothing$ tuščia. Tuo tarpu, šiam klausimui q nustatytas klausimo tipas $kt =< tl, c >$, kur klausimo rūšys *c = Pasirinkimai su vienu teisingu*, o taksonomijos lygmuo $tl =< t, v >$. Taip pat pridedamas taisyklių poaibis $R^* \in R$. Visų pirma, kaip matome 1 eilutėje, sukuriama tuščia užduočių aibė U^*. Toliau, 2 eilutėje kviečiama *GetPermutations(list, skaitiklis)* funkcija ir jai pateikiami parametrai $list = R_1^*$ ir $skaitiklis$. Taisyklių poaibis $R_1^* \in R^*$ priklauso pradiniam klausimui q ir R_1^* simbolizuoja visas taisykles r_n, kurios nusako, kokios klausimo q pasirinkimo galimybės yra neteisingos. Šis požymis taisyklėse pasižymi *identifikatorius = false*. Algoritmo $skaitiklis$ parametras nusako kiek neteisingų pasirinkimo taisyklių r paimti generuojant užduotį iš neteisingų klausimo pasirinkimo taisyklių poaibio R_1^*. Taip sugeneruojama aibė S, kuri sudaryta iš poaibių taisyklių, kur kiekviename elemente yra po *skaitiklis* taisyklių. Taisyklių poaibis $R_2^* \in R^*$ yra sudarytas iš taisyklių, kurios yra teisingi pasirinkimo variantai. Taisyklėse tai atsispindi *identifikatorius = true*. Kaip matome 3 ir 4 eilutėse, iteruojama per poaibį R_2^* ir aibę S ir *Dekarto sandaugos* principu sukuriami visi galimi rinkiniai taisyklių. Eilutėje 7 užduočiai u priskiriamas vienas teisingas pasirinkimas iš taisyklių poaibio R_2^*, o 8 eilutėje prie užduoties pridedamas rinkinys neteisingų taisyklių. Baigus iteruoti per aibę R_2^* užduotis u priskiriama aibei U^* (žr. 10 eilutę), toliau vėl iteracija per aibę S.

43 pavyzdys. Klausimo pavadinimas: Kokie duomenų tipų pavadinimai naudojami *Java* programavimo kalboje?

$q =< \{< true, integer >, < true, string >, < false, for >, < false, public >\}, <<Bloomo, Prisiminimas >, Pasirinkimai\ su\ vienu\ teisingu >, \varnothing >$

3 $skaitiklis = 1$
4 $R_1^* = < true, integer >, < true, string >$
5 $R_2^* = < false, for >, < false, public >$
6 $U^* = \{(integer, for), (integer, public), (string, for), (string, public)\}$

Pavyzdyje 43 pateikiamas trumpas aprašymas, kaip yra automatiškai sugeneruojama užduotis. Klausimas q yra rūšies c = *Pasirinkimai su vienu teisingu*, taksonomijos lygmuo $tl =< Bloomo, Prisiminimas >$, kur taksonomija *t = Bloomo*, taksonomijos lygmuo *v = Prisiminimas*, o kodo fragmentas *kf = < ∅, Sintaksė>*, kur *klasifikatorius = Sintaksė*, o eilučių aibė $L = \varnothing$ tuščia. Pateikiama taisyklių poaibis $R* =$ *{<true, integer>, <true, string>, <false, for>, <false, public>}*. Taisyklių poaibis R^* išskiriamas į du poaibius R_1^* ir R_2^* (žr. 4 ir 5 eilutes). Apsibrėžiamas kintamasis $skaitiklis = 1$ (žr. 41 pavyzdžio 2 eilutę). Pasinaudodami šiais parametrais ir 5 algoritmu, 6 eilutėje sugeneruota užduočių aibė U^* naudotojams.

Jei atlikinėjant tokias užduotis, kaip pavaizduota 43 pavyzdyje naudotojas teisingai identifikuoja, kuri reikšmė teisinga, jam gali būti skiriamas vienas taškas. Jei klausimo taksonomijos lygmuo $v = Supratimas$, tuomet pasirinkimų aibei padidėjus, sunkiau identifikuoti teisingą, tuomet teisingai pasirinkus skiriama du taškai. Atitinkamai, jei taksonomijos lygmuo $v = Taikymas$, už teisingą atsakymą skiriami trys taškai.

4.5. Užduočių generavimas nenaudojant taisyklių

Klausimai gali būti generuojami modifikuojant pradinio kodo fragmento eilučių tvarką ir nenaudojant jokių apibrėžtų taisyklių. Toks algoritmas pristatomas klausimų rūšiai *c = Sumaišytos eilutės*. Šis klausimas sugeneruojamas įvesto kodo fragmento eilutes sumaišius atsitiktine tvarka. Pagrindinė užduotis naudotojui nusakyti kokia yra tinkama eilučių seka. Ryškiausia šio klausimo rūšies vieta yra vertinimo modelis, nes taškai turi būti skiriami ne už tinkamai atsektą eilutės poziciją, bet už tinkamai sudėliotą eilučių seką ar bent jau jos dalį.

6 algoritmas. Algoritmo pavyzdys klausimo rūšiai *„Sumaišytos eilutės“*.

Įvestis: *q = < ∅, kt, kf> ∈ Q, kf = <L, klasifikatorius>, kt = <<t, v>, Sumaišytos eilutės>*
Išvestis: $U^* \leftarrow U$
1: $U^* \in \varnothing$
2: $U^* = GetPermutations(kf.L, |kf.L|)$
3: **return** U^*

Algoritme 6 pateikiama kaip sugeneruojamos užduotys nenaudojant taisyklių ir sumaišant kodo fragmento eilučių tvarką. Kaip matome, šio algoritmo taisyklių įvesties parametras yra tuščia aibė $R^* = \varnothing$. Klausimo tipas $kt =<< t, v >,$ *Sumaišytos eilutės*>, kur klausimo rūšis *c = Sumaišytos eilutės*. Eilutėje 2 matome, kad naudojama *GetPermutations* funkcija, kuriai perduodama klausimo kodo fragmento eilučių aibė *L* ir funkcijos rezultatai iš karto priskiriami užduočių aibei U^*. Funkcija *GetPermutations* grąžina visas galimas kombinacijas sumaišytų eilučių, kiekvienoje kombinacijoje yra $|kf.L|$ elementų.

Kombinacijų skaičius yra lygus užduočių aibės dydžiui $|U^*|$. Šį dydį galima apibrėžti paprastu faktorialu. Jei kodo fragmento $kf =< L, klasifikatorius >$, kur $L =<$ "a","b","c" $>$ eilučių aibės L dydis $|U^*| = 3$, tai gautos užduočių aibės dydis yra faktorialas $3! = 3 * 2 * 1 = 6$. Tokių klausimų sudėtingumą gali nulemti patys naudotojai įvedinėdami klausimo q kodo fragmentus kf,

nes kuo didesnis kodo fragmento kf eilučių skaičius $|L|$, tuo vartotojui sunkiau surikiuoti eilutes teisinga tvarka.

4 išeities kodas. $Java$ programavimo kalba parašytas metodas

```
1    public static void WriteLine(
2      String value
3    )
```

44 pavyzdys. Klausimo pavadinimas: Kokia yra teisinga eilučių tvarka?
$L = \{"public\ static\ void\ WriteLine(","String\ value",)"\}"$
$|L| = 3;$
$|U^*| = 6;$

Pavyzdys 44 demonstruoja, kaip veikia 6 algoritmas. Kodo fragmento *kf* = *<L, klasifikatorius>* eilučių seka kaip 4 išeities kodas. Tuomet *L = {"public static void WriteLine(", "String value", ")"}*, apibrėžkime, kad *klasifikatorius* = *Sintaksė*, klausimo tipas $kt =< tl, c >$, kur taksonomijos lygmuo $tl =< Bloomo, Supratimas >$ ir klausimo rūšis *c* = *Sumaišytos eilutės*. Kodo fragmento eilučių sekos ilgis $|L| = 3$, tad 6 algoritmas sugeneruos $3! = 6$ užduotis su sumaišytomis kodo fragmento eilutėmis.

Pavyzdyje 45 matome, kokios užduotys u sugeneruotos užduočių aibėje U^*.

45 pavyzdys. Užduočių aibė U^*.

```
1        public static void WriteLine(
2        String value
3        )
```

```
1        public static void WriteLine(
2        )
3        String value
```

```
1        String value
2        public static void WriteLine(
3        )
```

```
1        String value
2        )
3        public static void WriteLine(
```

```
1        )
2        String value
3        public static void WriteLine(
```

```
1        )
2        public static void WriteLine(
3        String value
```

Tokių klausimų vertinimo algoritmas susideda iš to, koks mažiausias neatitikimo indeksas. Jei *Levenšteino atstumo* indeksas užduoties eilučių aibėje u yra lygus nuliui, lyginant su pradiniu kodo fragmentu, tai reiškia, kad vartotojo sugeneruota kodo eilučių seka atitiko pradinį kodo fragmentą, todėl skiriami trys taškai. Jei *Levenšteino atstumas* yra du, tai reiškia, kad tik dvi eilutės neteisingose vietose, todėl vartotojas gauna tik du taškus, nes jei viena eilutė neteisingoje vietoje, vadinasi, jos vietoje esamos eilutės reikšmė, todėl skaičiuojama kaip viena klaida. Atitinkamai, jei *Levenšteino atstumas* yra trys arba keturi, tuomet dvi eilutės neteisingose vietose, todėl skiriamas tik vienas taškas. Kitais atvejais, kai *Levenšteino atstumas* > 4, taškai neskiriami.

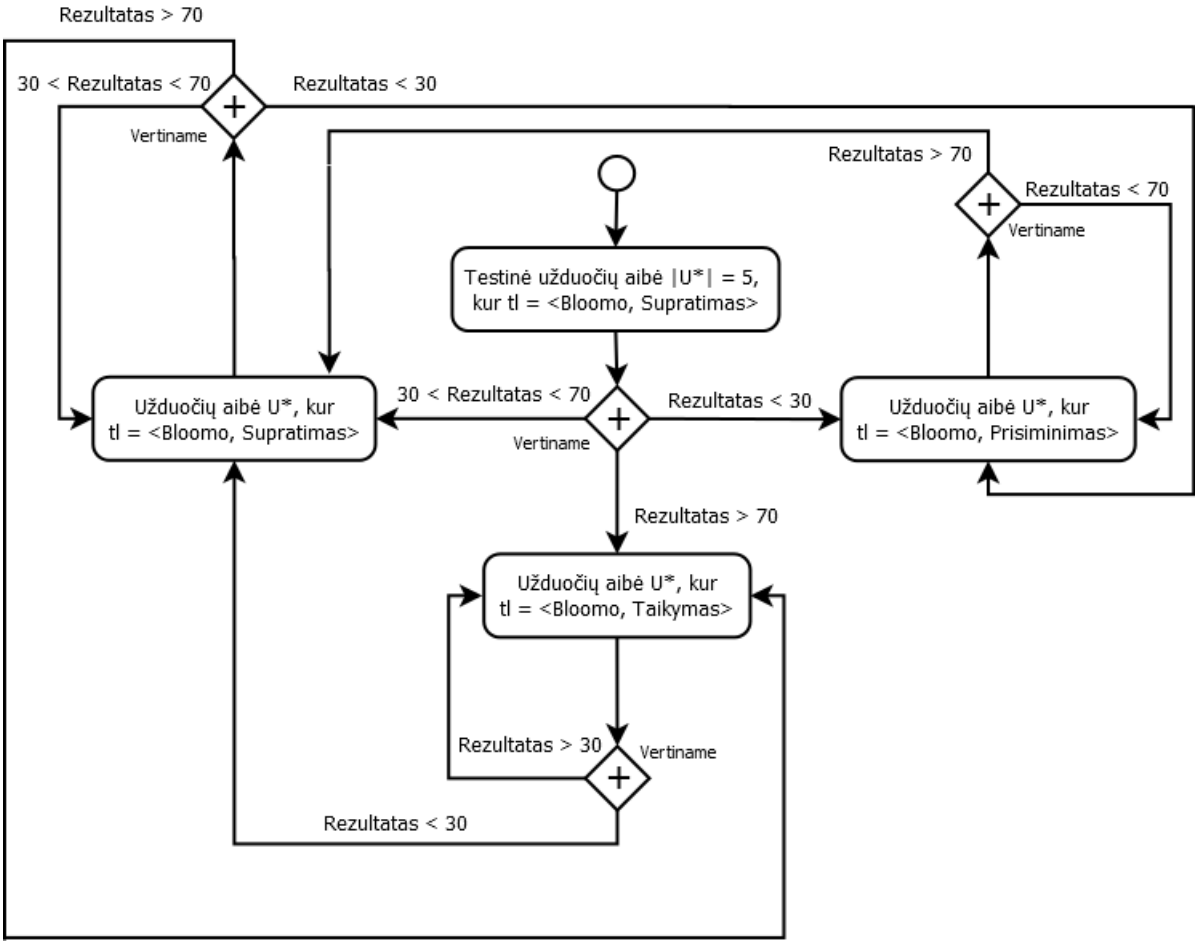
4.6. Automatinis lygių keitimas

Kaip rekomenduojama CAT [4] dokumentacijoje, visų klausimų aibės elementai suskirstyti į grupes pagal klasifikatorius ir temas *KLASĖ = {"Sintaksė", "Metodai", "Masyvai", "Klasės ir konstruktoriai", "Išimčių tvarkymas"}*, klausimų rūšis *C = {"Neteisinga eilutė", "Sumaišytos eilutės", "Pasirinkimas su daug teisingų", "Atviro tipo", "Pasirinkimai su vienu teisingu"}* ir sudėtingumą, taksonomijos lygmenis V^* *= {"Prisiminimas", "Supratimas", "Taikymas"}*. Kiekvieną kartą klausimų elementai paimami iš tam tikros kategorijos skirtingų tipų, pagal tai, koks naudotojo žinių lygio įvertinimas einamuoju metu.

Kaip rekomenduoja Millan, Loboda ir Perez-de-la-Cruz [2], pirmą kartą prisijungęs prie sistemos naudotojas gauna bandomųjų elementų aibę, pagal kuriuos nusprendžiamas preliminarus naudotojo modelio žinių lygis. Modeliuojamoje edukacinėje platformoje pirmą kartą prisijungus prie sistemos ir pasirinkus vieną iš kategorijų c pateikiama vidutinio sudėtingumo lygmens (*tl = <Bloomo, Supratimas>*) klausimų aibė. Ji sudaryta iš elementų, surinktų iš visų galimų klausimų rūšių c, atsitiktinai parinkus vieną užduotį $u \in U^*$. Modeliuojamos edukacinės platformos klausimo rūšių skaičius $|C| = 5$, tai reiškia, kad vartotojui pirmą kartą prisijungus prie sistemos ir pasirinkus atitinkamą *klasifikatorių* $\in$ *KLASĖ*, jis gauna penkias skirtingų rūšių užduotis.

Atsakius į eksperimentinius klausimus įvertinamas naudotojo žinių lygis. Jei galime daryti prielaidą, kad naudotojas supranta einamajame taksonomijos lygmenyje pateiktą mokymosi medžiagą, padidiname sudėtingumo lygmenį viena taksonomija aukščiau, tačiau visą laiką tame pačiame $klasifikatoriuje \in$ *KLASĖ*. Jei aukštesnio taksonomijos lygmens jau nėra, pateikiamos užduotys iš aukščiausio taksonomijos lygmens, tačiau keičiant klausimų variacijas iš atitinkamos klausimų kategorijos užduočių aibės U^*. Jei naudotojas neturi tam tikros taksonomijos lygio žinių, tai galime daryti prielaidą, kad jis neišmoko žemesnio lygmens taksonomijos užduočių, todėl naudotojui toliau pateikiamos užduotys iš žemesnio taksonomijos lygmens. Jei žemesnio taksonomijos lygmens jau nėra, pateikiamos užduotys iš žemiausios taksonomijos lygmens, tačiau keičiant klausimų variacijas iš atitinkamos klausimų kategorijos užduočių aibės U^*.

Kadangi modeliuojamoje edukacinėje platformoje užduočių įvertinimai nėra vienareikšmiški, tai turime daryti prielaidas, kada naudotojas supranta einamajame taksonomijos lygmenyje pateiktas užduotis. Veiklos diagramoje 1 vaizduojamas pavyzdinis procesas, kaip vyksta prisitaikymas prie naudotojo žinių lygio. Visų pirma, atliekamos užduotys iš testinės užduočių aibės $|U^*| = 5$, kur taksonomijos lygmuo *tl = <Bloomo, Supratimas>*. Kitas žingsnis – vertinimas. Patikriname, kiek teisingų atsakymų vartotojas parinko, įvedė. Jei *rezultatas > 70*, darome prielaidą, kad naudotojas išmokęs užduotis šiame taksonomijos lygmenyje *v = Supratimas* ir toliau jam pateikiamos užduotys iš aukštesnio taksonomijos lygmens *tl = <Bloomo, Taikymas>*. Jei *rezultatas < 30*, darome prielaidą, kad naudotojas neišmokęs žemesnio taksonomijos lygmens *v = Prisiminimas*, tad toliau jam pateikiamos užduotys iš taksonomijos lygmens *tl = <Bloomo, Prisiminimas>*. Kitokiu



1 pav. Edukacinės platformos automatinio adaptavimosi veiklos diagrama.

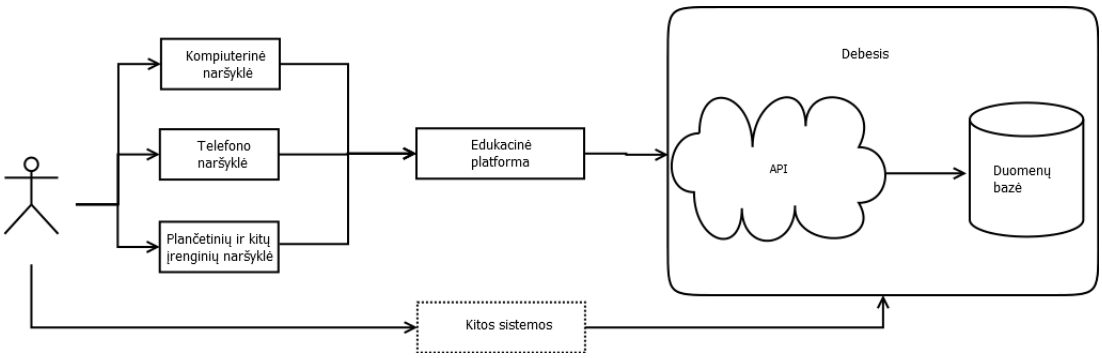
atveju, jei *rezultatas > 30* ir *rezultatas < 70*, naudotojo mokymosi procesas lieka tame pačiame taksonomijos lygmenyje *v = Supratimas*. Toliau, kaip matome 1 sekos diagramoje, pateikus taksonomijos lygmens *tl = <Bloomo, Supratimas>* užduotis, vėl vyksta vertinimas. Kur procesas kartojasi, jei *rezultatas > 70* sekantis taksonomijos lygmuo *v = Taikymas*, jei *rezultatas < 30* tada sekantis taksonomijos lygmuo *v = Prisiminimas*, kitu atveju, naudotojo žinių tobulinimo lygis lieka tas pats. Kaip demonstruoja 1 diagrama, matome, kad jei mokomasis taksonomijos lygmuo *v = Prisiminimas* ir atlikus vertinimą *rezultatas > 70*, tuomet taksonomijos lygmuo perkeliamas į *tl = <Bloomo, Supratimas>*, kitu atveju darome prielaidą, kad naudotojas neišmokęs einamosios temos, tačiau, kadangi žemesnio lygmens jau nėra, toliau užduočių aibė U^* pateikiama su užduotimis su taksonomijos lygmeniu *v = Prisiminimas*. Sekos diagramoje 1 galime pastebėti, kad jei naudotojų žinių lygis yra taksonomijos lygmens *tl = <Bloomo, Taikymas>*, tai po vertinimo, jam pateikiamos žemesnio lygmens *tl = <Bloomo, Supratimas>* tik tokiu atveju, jei *rezultatas < 30*, kitais atvejais daroma prielaida, kad naudotojui nėra per daug sunkus sudėtingumo lygmuo *v = Taikymas* ir užduočių aibė U^* generuojama su šio lygmens užduotimis. Kadangi aukštesnio lygmens apibrėžime neturime, tai taksonomijos lygmuo *tl = <Bloomo, Taikymas>*, laikomas kaip sudėtingiausias.

5. Edukacinės platformos prototipo įgyvendinimas

Modeliuojamoje edukacinėje platformoje naudojama taksonomija $t = Bloomo$ su apibrėžtų lygmenų poaibiu V^* = *{"Prisiminimas", "Supratimas", "Taikymas"}*. Šie taksonomijos lygmenys naudojami kaip edukacinės platformos užduočių sudėtingumo lygmenys, kur žemiausias taksonomijos lygmuo $v_1 = Prisiminimas$ yra prilyginama lengviausioms užduotims, atitinkamai, aukščiausias taksonomijos lygmuo $v_3 = Taikymas$ sudėtingiausioms užduotims. Taip pat apibrėžiama klausimų rūšių aibė *C = {"Neteisinga eilutė", "Sumaišytos eilutės", "Pasirinkimas, su daug teisingų", "Atviro tipo", "Pasirinkimai, su vienu teisingu"}*. Pagal atliktą pavyzdinių programėlių analizę apibrėžta temų *klasifikatorių* aibė *KLASĖ = {"Sintaksė", "Metodai", "Masyvai", "Klasės ir konstruktoriai", "Išimčių tvarkymas"}*.

Platforma susideda iš kelių pagrindinių konstrukcinių elementų (žr. 2 paveiksliukas):

1. Programavimo sąsaja veiksmų automatizavimui bei integravimui (API);
2. Duomenų bazė;
3. Edukacinė platforma.



2 pav. Edukacinės platformos architektūros modelis.

Eskiziniame edukacinės platformos architektūros modelyje, pavaizduotame 2 paveikslėlyje, matome, kad API ir duomenų bazė yra patalpinta viename serveryje, į kurį kreipiasi edukacinė platforma. Pati edukacinė platforma gali būti naudojama per kompiuterinę naršyklę, telefono, planšetės ar kitų įrenginių naršykles. Tokiam sistemos adaptyvumui užtikrinti naudojamas prisitaikantis naudotojo dizainas. Kaip demonstruoja 2 paveiksliukas, į serverį, kuriame yra duomenų bazė ir edukacinės platformos API, gali kreiptis ir kitos sistemos, kaip pavyzdžiui, mobilioji programėlė. Tokia sistemos architektūra edukaciniai platformai suteikia lankstumo ir plečiamumo.

5.1. Edukacinės platformos struktūra

Platformai igyvendinti buvo naudota C#, MVC, ASP.NET, HTML, CSS, Javascript technologijos. Įrenginiai, per kuriuos bus naudojama edukacinė platforma, turi turėti prieigą prie interneto. Naršyklė ir serveris duomenis siunčia ir gauna per https protokolą.

Sistemos naudotojai turi galimybę valdyti, kurti klausimus. Taip pat svarbu, kad galėtų į juos atsakinėti kiti naudotojai, bei pamatyti kokie atsakymai yra teisingi. Tam įgyvendinti apibrėžti tam tikri panaudojimo atvejai:

1. Kaip naudotojas, aš noriu prisijungęs prie sistemos pasirinkti man aktualią temą, pagal kurią norėsiu gauti klausimus;
2. Kaip naudotojas, aš noriu gauti klausimus man aktualia tema ir matyti ar teisingus atsakymo variantus pasirinkau;
3. Kaip naudotojas, aš turiu galėti sukurti naują klausimų sritį ir nustatyti jai galimus sudėtingumo lygius, rūšis;
4. Kaip naudotojas, aš galiu sukurti specifinei sričiai tam tikrus klausimus ir nustatyti visus įmanomus atsakymų variantus.
5. Kaip naudotojas, aš noriu gauti klausimus pagal mano pasirinktą temą, nuolatos individualiai pritaikant sudėtingumo lygmenį;
6. Kaip naudotojas, aš noriu matyti visų taisyklių sąrašą bei klausimus ir kokios užduotys pagal juos sugeneruotos.

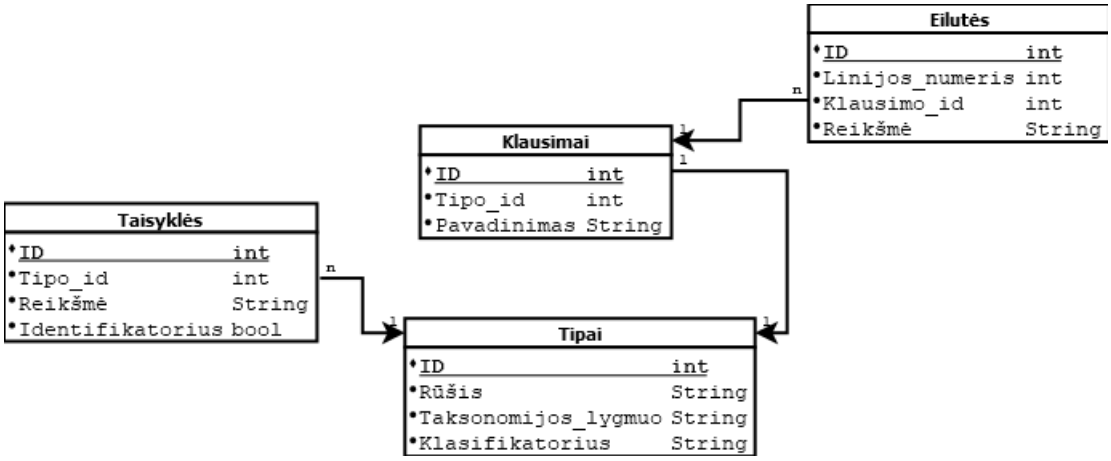
5.2. API modelis

Aplikacijų programavimo sąsajos (*angl. Application Programming Interface, API*) pagrindinė funkcija yra darbas su duomenimis. Ši sąsaja rūpinasi duomenų ištraukimu/įrašimu/pakeitimu duomenų bazėje, rezultatų skaičiavimu, užduočių generavimu pagal algoritmus. API su duomenų baze bendrauja per Entity Framework, o prisijungimas prie API iš edukacinių platformų vyksta per HTTP. Užtikrinti pilną sistemos veikimą, apibrėžti API funkciniai reikalavimai:

1. Klausimų atžvilgiu:
   (a) API turi turėti galimybę išsaugoti klausimą į duomenų bazę su visais jo parametrais;
   (b) API turi turėti galimybę išsaugoti ir apdoroti įvairių rūšių klausimo atsakymo variantus į duomenų bazę:
      i. Neteisinga eilutė;
      ii. Pasirinkimai, kur keli galimi atsakymai teisingi;
      iii. Atviro tipo;
      iv. Pasirinkimas, kur tik vienas atsakymas yra teisingas;
      v. Sumaišytos eilutės;
   (c) API turi galėti grąžinti sąrašą visų klausimų, susijusių su tam tikra aktualia tema;
   (d) API turi galėti grąžinti sąrašą visų klausimų, susijusių su tam tikra taksonomija;
   (e) API turi galėti grąžinti sąrašą visų klausimų su ta pačia rūšimi;
   (f) API turi turėti galimybę priimti ir patikrinti naudotojo įvestus atsakymus;
   (g) API turi įvertinti naudotojo žinių lygį ir pagal jį grąžinti klausimų sąrašą.

5.3. Duomenų bazės struktūra

Duomenų bazės modelis yra skirtas duomenų saugojimui. Jame saugojami pagrindiniai automatinio klausimų generavimo proceso komponentai: klausimų informacija, taisyklių, tipų ir eilučių informacija. Edukacinei platformai įgyvendinti buvo naudota MS SQL duomenų bazė. Į duomenų bazę API perduoda arba pateikia užklausas duomenų sinchronizacijai atlikti. Reliacinį duomenų bazės modelį demonstruoja 3 paveiksliukas.



3 pav. Modeliuojamos edukacinės platformos duomenų bazės reliacinis modelis.

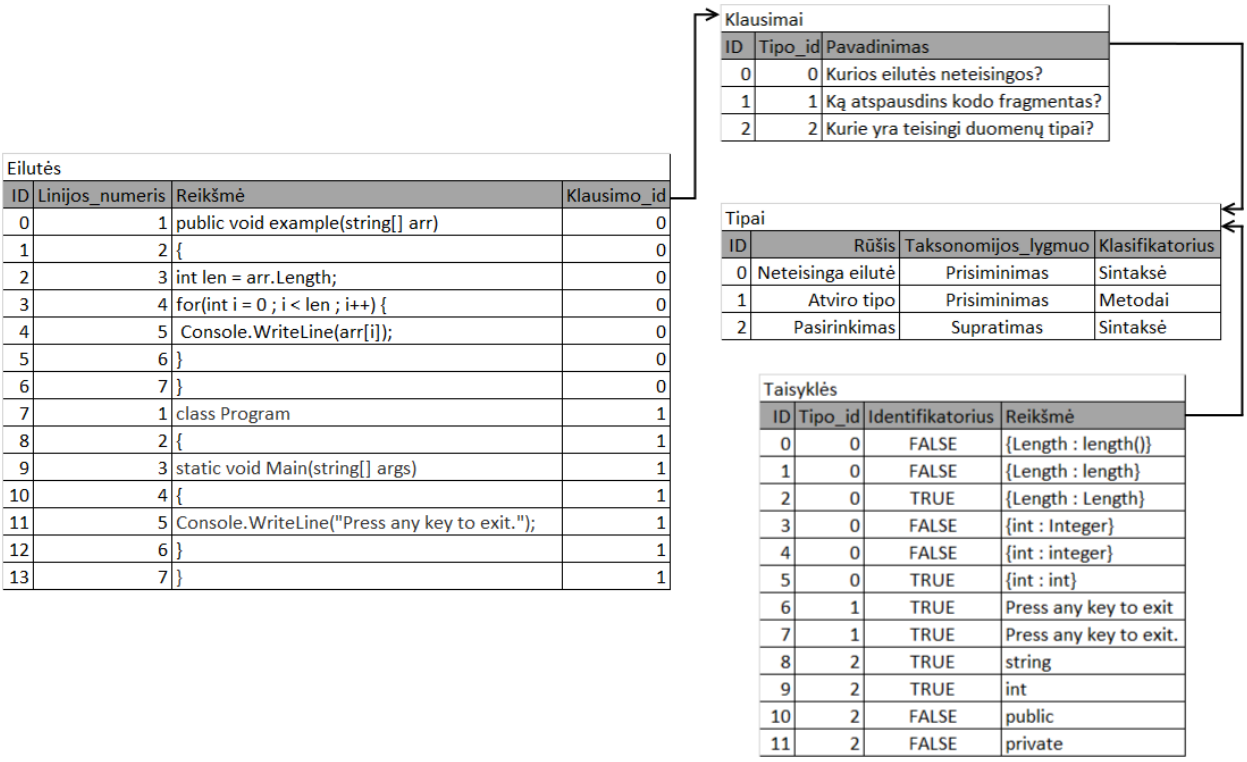
Kaip matome, lentelės *Klausimai* pirminis raktas yra sveiko skaičiaus tipo stulpelis *ID*. Ši lentelė, taip pat, laiko eilutės tipo informaciją apie klausimo pavadinimą (stulpelis *Pavadinimas*) ir išorinį raktą *Tipo_id* į lentelę *Tipai*. Lentelės *Tipai* pirminis raktas yra *ID*. Šioje lentelėje dar yra trys eilutės tipo stulpeliai *Rūšis, Taksonomijos_lygmuo* ir *Klasifikatorius*. Šios lentelės tarpusavyje turi *vienas su vienu* sąryšį, nes vienas klausimas apibrėžiamas vienu tipu. Tuo tarpu, lentelė *Eilutės* su lentele *Klausimai* turi sąryšį *vienas su daug*, nes vienas klausimas gali turėti daug eilučių. *Eilutės* lentelės pirminis raktas yra *ID* ir ji turi išorinį raktą *Klausimo_id* į lentelę *Klausimai*. Taip pat, šioje lentelėje yra eilutės tipo stulpelis *Reikšmė* ir sveikas skaičius *Linijos_numeris*. Kaip matome 3 paveiksliuke, lentelė *Tipai* turi sąryšį vienas su daug su lentele *Taisyklės*, nes vienam tipui apibrėžti gali būti daug taisyklių. Lentelėje *Taisyklės* pirminis raktas yra *ID* ir išorinis raktas *Tipo_id* į lentelę *Tipai*. Taip pat, šioje lentelėje yra eilutės tipo stulpelis *Reikšmė* ir loginio tipo stulpelis *Identifikatorius*.

5.4. Duomenų bazės užpildymas duomenimis

Kaip matėme 5.3 skyriuje pavaizduotas duomenų bazės modelis (žr. 3 pav.), kuris naudojamas edukacinei platformai modeliuoti. Pateiktame 4 paveiksliuke galima pamatyti kaip duomenų bazės modelis užpildomas pavyzdiniais duomenimis.

Viena iš pagrindinių duomenų bazės lentelių modeliuojamoje edukacinėje platformoje yra *Tipai*. Kaip matome 4 paveiksliuke, duomenų bazės lentelė *Tipai* saugo tokią informaciją: *Rūšis, Taksonomijos_lygmuo ir Klasifikatorius*. *Rūšis* stulpelio duomenys nusako kokia klausimo rūšys c^* = *{"Neteisinga eilutė", "Atviro tipo", "Pasirinkimas"}*. Lentelės *Tipai* stulpelis *Taksonomijos_lygmuo* saugo $Bloomo$ taksonomijos lygmenų pavadinimai V^*=*{"Prisiminimas", "Supratimas", "Taikymas"}*. *Klasifikatorius* duomenys nusako kokiai temai priklauso lentelės *Tipai* įrašas.

Dar viena duomenų bazės lentelė yra *Taisyklės*. Šioje lentelėje kiekviena eilutė reiškia atskirą taisyklę klausimo generavimui. Lentelės stulpelis *Tipo_id* yra nuoroda į įrašą esantį *Tipai* lentelė-



Klausimai

ID	Tipo_id	Pavadinimas
0	0	Kurios eilutės neteisingos?
1	1	Ką atspausdins kodo fragmentas?
2	2	Kurie yra teisingi duomenų tipai?

Eilutės

ID	Linijos_numeris	Reikšmė	Klausimo_id
0	1	public void example(string[] arr)	0
1	2	{	0
2	3	int len = arr.Length;	0
3	4	for(int i = 0 ; i < len ; i++) {	0
4	5	Console.WriteLine(arr[i]);	0
5	6	}	0
6	7	}	0
7	1	class Program	1
8	2	{	1
9	3	static void Main(string[] args)	1
10	4	{	1
11	5	Console.WriteLine("Press any key to exit.");	1
12	6	}	1
13	7	}	1

Tipai

ID	Rūšis	Taksonomijos_lygmuo	Klasifikatorius
0	Neteisinga eilutė	Prisiminimas	Sintaksė
1	Atviro tipo	Prisiminimas	Metodai
2	Pasirinkimas	Supratimas	Sintaksė

Taisyklės

ID	Tipo_id	Identifikatorius	Reikšmė
0	0	FALSE	{Length : length()}
1	0	FALSE	{Length : length}
2	0	TRUE	{Length : Length}
3	0	FALSE	{int : Integer}
4	0	FALSE	{int : integer}
5	0	TRUE	{int : int}
6	1	TRUE	Press any key to exit
7	1	TRUE	Press any key to exit.
8	2	TRUE	string
9	2	TRUE	int
10	2	FALSE	public
11	2	FALSE	private

4 pav. Užpildytos duomenų bazės modelis.

je. Stulpelyje *Reikšmė* užpildomos konkrečių *taisyklių* reikšmės. Požymis *Identifikatorius* nusako taisyklių požymius. Jei *Identifikatorius* = *false*, tai suprantama, kaip apgaulinga taisyklė, o jei *Identifikatorius* = *true*, suprantama, kad tai teisinga taisyklė. Pavyzdžiui, jei klausimas yra *Atviro tipo*, tai visos jo taisyklės turi požymį *Identifikatorius* = *true*.

Klausimai talpina duomenis apie klausimų aibę, kuri saugo *Pavadinimą* ir *Tipo_id*. Lentelėje *Klausimai* matome, kad yra sukurti trys klausimai: *Kurios eilutės neteisingos? Ką atspausdins kodo fragmentas?* ir *Kurie yra teisingi duomenų tipai?* Kiekvienas iš klausimų turi savo unikalų *ID* ir nuorodą į lentelę *Tipai*, kurioje apibrėžiama, kokio tipo yra klausimas.

Lentelė *Eilutės* saugo informaciją apie kiekvieną kodo fragmento eilutę, taip pat, laiko išorinį raktą į lentelę *Klausimas*, kad būtų galima atsekti, kurio klausimo išvesties kodui priklauso eilutės. Kiekvienas įrašas *Eilutės* lentelėje turi *Linijos_numerį*, kuris nusako, kelinta pagal eilę ši eilutė yra kodo fragmente. Šis stulpelis užpildomas automatiškai išsaugant klausimus ir jų kodo fragmentus.

46 pavyzdys. Kaip 5.4 paveikslėlyje pavaizduotoje duomenų bazėje matome, *Klausimai* lentelėje įrašas *Kurios eilutės neteisingos?* turi nuorodą į lentelę *Tipai*, į konkretų tipą su $ID = 0$. Pagal lentelę *Tipai* matome, kad šis klausimas yra *Neteisinga eilutė* klausimo rūšies, atitinka pirmąjį $Bloomo$ taksonomijos lygmenį *Prisiminimas*. Kaip matome pagal *Klasifikatorių*, šis klausimas priklauso temai *Sintaksė*. *Taisyklės* lentelėje šiam tipui suteikiamos taisyklės, kur $ID = \{0, 1, 2, 3, 4, 5\}$. Šios taisyklės nurodo, kokius raktažodžius reikia pakeisti klausimo kodo fragmente. Lentelėje *Eilutės*, šiam klausimui priskiriamos linijos, kur $ID = \{0, 1, 2, 3, 4, 5, 6\}$ ir tai yra klausimo įvesties kodo fragmentas.

47 pavyzdys. *Klausimai* lentelėje įrašas *Ką atspausdins kodo fragmentas?* turi nuorodą i lentelę *Tipai*, į konkretų tipą su $ID = 1$. Pagal lentelę *Tipai* matome, kad šis klausimas yra *Atviro tipo* klausimo rūšies, atitinka pirmąjį $Bloomo$ taksonomijos lygmenį *Prisiminimas* ir pagal *klasifikatorių* priklauso temai *Metodai*. *Taisyklės* lentelėje šiam tipui suteikiamos taisyklės, kur $ID = \{6, 7\}$.

Tai yra atsakymų variantai, kuriuos modeliuojamoji edukacinė platforma palyginusi su vartotojo įvestais, užskaitytų kaip teisingus. Lentelėje *Eilutės*, šiam klausimui priskiriamos linijos, kur $ID = \{7, 8, 9, 10, 11, 12, 13\}$.

48 pavyzdys. Duomenų bazės lentelėje *Klausimai* yra dar vienas įrašas *Kurie yra teisingi duomenų tipai?*, kuris turi nuorodą į lentelę *Tipai*, į konkretų tipą su $ID = 2$. Pagal lentelę *Tipai* matome, kad šis klausimas yra *Pasirinkimas* klausimo tipo, atitinka jau antrąjį $Bloomo$ taksonomijos lygmenį (*Supratimas*). *Klasifikatorius* nusako, kad šis klausimas priklauso *Sintaksė* temai. *Taisyklės* lentelėje šiam tipui suteikiamos taisyklės, kur $ID = \{8, 9, 10, 11\}$. Lentelėje *Eilutės* šiam klausimui priskiriamų linijų nėra, nes galimi pasirinkti variantai vartotojui formuojami tik iš lentelės *Taisyklės*.

5.5. Skaitliukų parinkimai užduotims be kodo fragmentų

Skyriuose 4.4 ir 4.2 pristatyti algoritmai, kurie generuoja užduotis nenaudojant kodo fragmentų. Šiems algoritmams reikalingi ne tik taisyklių poaibiai R_1^* ir R_2^*, bet ir atitinkami skaitliukai $skaitiklis_1$ ir $skaitiklis_2$, kurie nurodo, kokio dydžio rinkinius sudarinėti iš taisyklių aibių. Šie skaitliukai modeliuojamoje edukacinėje platformoje parinkti ne atsitiktinai. Atitinkamai pagal tai, koks yra klausimo tipo kt taksonomijos lygmuo v, pagal tai ir parenkami šie skaitliukai. Lentelėje 2 pateikiama, kokie skaitliukai parinkti pagal taksonomijos lygmenis klausimo rūšiai *c = Pasirinkimas su daug teisingų*.

2 lentelė. $skaitiklis_1$ ir $skaitiklis_2$ skaitliukų parinkimas.

Taksonomijos lygmuo	$skaitiklis_1$	$skaitiklis_2$
Prisiminimas	1	1
Supratimas	1	2
Taikymas	2	3

Kaip matome 2 lentelėje, kuo sudėtingesnis taksonomijos lygmuo, tuo daugiau galimų teisingų ir neteisingų reikšmių gali būti pridėta į pasirinkimo tipo užduotis besimokantiems.

49 pavyzdys. Klausimo pavadinimas: Kokie duomenų tipų pavadinimai naudojami *Java* programavimo kalboje?

q = <{<true, integer>, <true, string>, <false, for>, <false, public>, <false, static>}, < <Bloomo, Taikymas>, Pasiriknimai, su daug teisingų>, ∅ >

$R_1^* = \{< true, integer >, < true, string >\}$

$R_2^* = \{< false, for >, < false, public >, < false, static >\}$

u = *{(integer, for, public, string, static)}* $\in U^*$

Kaip matome 49 pavyzdyje, tai klausimui q su klausimo rūšimi *v = Pasirinkimai su daug teisingų*, taksonomija *t = Bloomo* taksonomijos lygmeniu *v = Taikymas*, iš taisyklių poaibio R^* *= {<true, integer>, <true, string>, <false, for>, <false, public>, <false, static>}* sugeneruojama užduotis $u \in U^*$, kuri sudaryta iš penkių reikšmių, dvi teisingos ir trys neteisingos. Jei pažiūrėtume į 2 lentelę, matome, kad pirmajam taksonomijos lygmeniui *Prisiminimas* yra viena teisinga ir viena neteisinga eilutė. Antrajam taksonomijos lygmeniui *Supratimas* pateikiama, taip pat, tik viena teisinga reikšmė, tačiau 2 neteisingos, todėl yra sunkiau identifikuoti tą vieną teisingą.

3 lentelė. *skaitiklio* parinkimas.

Taksonomijos lygmuo	$skaitiklis$
Prisiminimas	2
Supratimas	3
Taikymas	4

Klausimo rūšiai c = *Pasirinkimai su vienu teisingu* skaitliukas *skaitiklis* taip pat parenkamas pagal tai, kokiam taksonomijos lygmeniui v priklauso klausimas (žr. 3 lentelę).

Kaip matome 3 lentelėje, $skaitiklis$ žemiausiam taksonomijos lygmeniui *v = Prisiminimas* pradedamas jau nuo dviejų, tai reiškia, kad šiam taksonomijos lygmeniui klausimui sudaryti bus naudojamos dvi neteisingos taisyklės iš taisyklių poaibio R_1^*. Atitinkamai, didėjant taksonomijos lygmeniui v, neteisingų taisyklių $skaitiklis$ didėja. Visiems taksonomijos lygmenims teisingų taisyklių skaitliukas išlieka toks pat, nes šie klausimai turi tik vieną teisingą atsakymą.

50 pavyzdys. Klausimo pavadinimas: Kokiu duomenų tipu apibrėžiamos eilutės *Java* programavimo kalboje?

q = <{<true, String>, <false, int>, <false, double>, <false, char>}, < <Bloomo, Prisiminimas>, Pasirinkimai, su vienu teisingu>, $\varnothing >\}$

$R_1^* = \{< true, String >\}$

$R_2^* = \{< false, int >, < false, double >, < false, char >\}$

$U^* = \{\{String, int, double\}, \{String, double, char\}, \{String, int, char\}\}$

Kaip matome 50 pavyzdyje, tai klausimui q su klausimo rūšimi *v = Pasirinkimai su vienu teisingu*, taksonomija *t = Bloomo* taksonomijos lygmeniu *v = Supratimas*, iš taisyklių poaibio R^* *= {<true, String>, <false, int>, <false, double>, <false, char>}* sugeneruojama užduočių aibė U^*, kuri sudaryta iš trijų rinkinių užduočių. Matome, kad visose užduotyse $u \in U^*$ yra vienas teisingas pasirinkimas *String* ir po du neteisingus. Atitinkamai, jei klausimo taksonomijos lygmuo būtų *v = Supratimas*, tai kiekvienoje užduotyje $u \in U^*$ būtų jau trys neteisingos reikšmės ir viena teisinga. Padidinus taksonomijos lygmenį iki *v = Taikymas*, praplėstume pasirinkimų aibę ir teisingų atsakymų liktų vis tiek vienas, neteisingų jau keturi.

Išvados ir rekomendacijos

Išnagrinėta literatūra ir apžvelgtos panašaus pobūdžio sistemos padėjo apibrėžti koks funkcionalumas yra aktualiausias mokymosi platformose, padėjo suformuluoti sistemos ir jos veikimo algoritmų pagrindinę mintį. Taksonomijų apibrėžimai padėjo nustatyti sudėtingumo lygmenis, adaptuotus informacinių technologijų srities atstovams, automatiškai generuojamiems klausimams. Išnagrinėti pavyzdiniai studentų darbai parodė, kokie esminiai programavimo aspektai svarbūs besimokant ir programuojant, tad iš to sudėliotas klausimų temų sąrašas edukacinėje platformoje.

Sumodeliuotas algoritmų modelis, kuris pristato kaip automatizuotai sugeneruoti aibę užduočių. Algoritmai atsižvelgia į taisykles, klausimų rūšis, sudėtingumo lygmenis, temas. Sumodeliuotas algoritmas, kuris automatiškai prisitaiko prie naudotojo žinių modelio.

Sudėliota edukacinės platformos architektūra, kuri pabrėžia, kad už pagrindinę logiką atsakingas API talpinamas atskirai nuo naudotojų sistemų, kas užtikrina lankstų perpanaudojimą per skirtingus įrenginius. Apgalvotas duomenų bazės modelis demonstruoja kaip saugoti duomenis susijusius su klausimais ir užduotimis, kad ateityje, plečiant klausimų tipų sąrašą, būtų patogu naudoti šią duomenų bazės struktūrą. Įgyvendinus API ir duomenų bazę pagal apibrėžtus modelius, modeliuotos edukacines platformos algoritmai atvaizduojami internetinėje platformoje.

Ateities tyrimų planas

Šiame darbe modeliuojamos edukacinės platformos ateities tyrimų kryptys yra dvi: modelio ir įgyvendinimo. Modelis gali būti plečiamas padidinant klausimų netrivialumą. Tai įmanoma padaryti įvairiais būdais:

1. Sudaryti klausimų klasifikatorių hierarchijas. Tokiu būdu mokantis kiekvienoje kategorijoje galima koncentruotis į tam tikrą šaką;

2. Įgyvendinti daugiau klausimų algoritmų, kad užduočių aibė būtų dar įvairesnė. Tokiu atveju naudotojas turės didesnes galimybes klausimų kūrimui;

3. Praplėsti automatinį lygių keitimą iki to, kad parinktų užduotis ne tik pagal naudotojo žinių lygį, bet ir pagal kitų naudotojų dažniausiai padaromas klaidas;

4. Taisyklių ir programinio kodo perpanaudojimas. Praplėsti algoritmus, kad tarpusavyje dalintųsi įvesties duomenimis, taisyklėmis, tokiu atveju, pagal vieną kodo fragmentą būtų galima sukurti didesnę užduočių aibę.

Žiūrint iš įgyvendinimo pusės, į sistemą pridėjus naudotojų registraciją, galima identifikuoti naudotojus pagal jų roles, kur atitinkamai skirtingoms rolėms suteikiamos skirtingos prieigos prie sistemos funkcionalumo. Vieni vartotojai gali turėti galimybę kurti klausimus, kiti tik atsakinėti, treti – matyti rezultatus.

Įgyvendinus autentifikavimą sistemoje galima saugoti visų naudotojų veiklos istoriją, kuri padėtų dar geriau parinkti užduotis naudotojams. Taip pat istorijos saugojimas gali padėti matyti naudotojų veiklos statistiką, kaip keičiasi jų rezultatai, koks jų aktyvumas.

Modeliuojamos edukacinės sistemos saugumui padidinti ir užtikrinti į sistemą galima įdiegti papildomus saugumo įrankius bei korektiško veikimo palaikymui integruoti klaidų įrašymo (angl. *logging*) mechanizmą.

Literatūros šaltiniai

[1] William H. E. Day. Properties of levenshtein metrics on sequencest. Bulletin of Mathematical Biology Vol. 46, No. 2, pp. 327-332, 1984.

[2] Eva Millan, Tomasz Loboda, Jose Luis Perez de-la Cruz. Bayesian networks for student model engineering. Computers Education, Vol. 55, No. 4, 1663-1683, 2010.

[3] Katrien Verbert, Nikos Manouselis, Xavier Ochoa, Martin Wolpers, Hendrik Drachsler, Ivana Bosnic, Erik Duval. Context-aware recommender systems for learning: A survey and future challenges. Institute of Electrical and Electronics Engineers transactions on learning technologies, Vol. 5, No. 4,, 2012.

[4] Stefan Oppl, Florian Reisinger, Alexander Eckmaier, Christoph Helm. A flexible online platform for computerized adaptive testing. International Journal of Educational Technology in Higher Education Vol. 14, No. 2, 2017.

[5] Paul Stefan, Popescu Marian, Cristian Mihaescu, Costel Ionascu. Intelligent tutor recommender system for on-line educational environments. In Proceedings of International Conference on Educational Data Mining, 2015.

[6] Sigitas Daukilas, Judita Kasperiūnienė. E. mokymosi kursų projektavimas ir realizavimas. Aleksandro Stulginskio universiteto metodinė medžiaga, 2011.

[7] Suntae Kim, Dongsun Kim. Automatic identifier inconsistency detection using code dictionary. Empirical Software Engineering, Vol. 21, No. 2, 565–604, 2016.

[8] Erkki Kaila, Einari Kurvinen, Erno Lokkila. Ville – teacher's handbook. Ville dokumentacija, 2014.
https://oppimisanalytiikka.fi/en/support/documentation/book-ville.

[9] David Ginat, Eti Menashe. Solo taxonomy for assessing novices' algorithmic design. Proceedings of the 46th ACM Technical Symposium on Computer Science Education, 452-457, 2015.

[10] Thomas Lodzinski, Jan M. Pawlowski. The quality mark e-learning: Developing process- and product-oriented quality. Handbook on Quality and Standardisation in E-Learning pp 109-124.

[11] Ville Hyyrynen, Harri Hämäläinen, Jouni Ikonen, Jari Porras. Mypeerreview: An online peer-reviewing system for programming courses. Proceedings of the 10th Koli Calling International Conference on Computing Education, 2010.

[12] Thomas Berger, Ulrike Rockmann. Quality of e-learning products. Handbook on Quality and Standardisation in E-Learning ISBN 978-3-540-32787-5, 2004.

[13] Robert Ruprecht. How to guarantee quality in education. Problems of education in the 21st century, Vol. 7, 2008.

[14] Teemu Rajala, Mikko-Jussi Laakso, Erkki Kaila, Tapio Salakoski. Ville – a language-independent program visualization tool. Practice in Information Technology, Vol. 88, 2014.
http://crpit.com/confpapers/CRPITV88Rajala.pdf.

[15] Linda Seiter. Using solo to classify the programming responses of primary grade students. Proceedings of the 46th ACM Technical Symposium on Computer Science Education, 540-545, 2015.

[16] Sunil Kumar Singh. Cartesian product. OpenStax-CNX, 2009. http://cnx.org/contents/nO8YYdjs@5/Cartesian-product.

[17] Valentina Dagienė, Sue Sentance, Gabrielė Stupurienė. Developing a two-dimensional categorization system for educational tasks in informatics. Informatica, Vol. 28, No. 1, 23–44, 2017.

[18] Marek Medrek, Anna Tatarczak. Business intelligence and data analysis in an adaptive web-based integrated learning environment. In Proceedings of International Technology, Education and Development Conference, 2017.

[19] Linas Bukauskas, Vitalijus Denisovas, Virgilijus Kuklierius, Genadijus Kulvietis, Jonas Kazimieras Matickas, Romanas Tumasonis. Guidelines of competence development in the study field of informatics. ISBN 978-609-462-001-0, 2014. http://www.ects.cr.vu.lt/Files/File/08_Informatics_guidelines.pdf.