Programavimo mokymasis: lyginamoji kalbos ir aplinkos analizė

Valentina Dagienė
Matematikos ir informatikos instituto vyriausioji mokslo darbuotoja, skyriaus vadovė, profesorė, daktarė
Institute of Mathematics and Informatics, Chief Research Scientist, Head of the Department, Prof., PhD
Akademijos g. 4, LT-08663 Vilnius
Tel. (+370 5) 2 72 97 36, faks. (+370 5) 2 72 92 09
El. paštas: *dagiene@ktl.mii.lt*

Jūratė Urbonienė
Matematikos ir informatikos instituto doktorantė
Institute of Mathematics and Informatics, doctoral student
Akademijos g. 4, LT-08663 Vilnius
El. paštas: *jurate99@gmail.com*

Straipsnyje nagrinėjami programavimo mokymo ypatumai, apžvelgiamos svarbiausios programavimo mokymo ir mokymosi tyrimo tendencijos pasaulyje, išskiriamos problemos ir jų sprendimo būdai. Remiantis mokslinės literatūros analize ir daugiamete Jaunųjų programuotojų mokyklos patirtimi, nagrinėjamos programavimo mokymosi sunkumo priežastys: programavimo srities specifiškumas, mokymosi būdų ir metodų parinkimas, mokinių gebėjimai ir nuostatos, psichologinis motyvuotumas. Daugiausia dėmesio skiriama programavimo kalbų ir aplinkos, tinkamos mokytis programavimo, kriterijams aptarti. Remiamasi fundamentaliais šios srities mokslininkų darbais, jie sisteminami, išskiriamos ir apibendrinamos esminės idėjos. Gilinamasi į programavimo mokymuisi tinkamų kalbų sintaksės ir semantikos ypatumus: patirtis rodo, kad pirmosios kalbos sintaksė daro didelį poveikį tolesniam mokymuisi, formuoja pažangią mąstyseną.

Pagrindiniai žodžiai: *programavimo mokymasis, programavimo kalbos, programavimo aplinka, sintaksė, programavimo kalbų kriterijai, Bloomo taksonomija, SOLO taksonomija.*

Įvadas

Programavimo mokymas, ypač mokymasis, – aktuali problema. Programavimo pradedama mokytis įvairaus amžiaus: nuo pradinių klasių iki aukštosios mokyklos ir dar vėliau. Lietuvos bendrojo lavinimo mokyklose galimybė susipažinti su pirmaisiais programavimo žingsniais įgyvendinta informacinių technologijų programoje: jau penktoje klasėje mokytojai turi supažindinti mokinius su programų kūrimu, pavyzdžiui, pasitelkdami *Logo* arba *Scratch* aplinką. Vėliau, devintoje ar dešimtoje klasėje, numatyta galimybė pasirinkti programavimo modulį (čia skiriama dėmesio algoritmavimui).

Lietuvoje nuo 1986 m. iki pat pertvarkos 1999 m. privalomas informatikos kursas buvo dėstomas 10–12 klasėse, didžiau-

sia dalis buvo skirta algoritmavimui, dėl kompiuterių stygiaus dažnai buvo mokoma tik teoriniu lygiu. Taip darė ir kitos šalys, ypač – Rytų Europos. Algoritmuojant siekiama išmokyti suprasti formalių žymenų reikšmę, gebėti jais išreikšti veiksmus. Šio kurso dalis glaudžiai susijusi su matematika.

Algoritmuojant svarbu ne tik veiksmų, bet ir duomenų, aprašų, pasirinktos kalbos formalizavimas, produkto užbaigimas ir pateikimas. Kreipiama dėmesio į formalios kalbos konstrukcijų supratimą, uždavinio algoritmo atlikimą kompiuteriu. UNESCO parengtose pavyzdinėse informatikos (informacinių technologijų) mokymo programose (Anderson, 2002) algoritmų siūloma mokyti pasirinktiniu sustiprintu lygiu. Kitos institucijos, pavyzdžiui, IFIP (*International Federation for Processing Information*) darbo grupė WG 3.1 kreipia dėmesį ugdyti algoritminį mąstymą (Taylor, 1993; Van Weert, 1993).

Lietuva, įgijusi ilgametę algoritmavimo mokymo patirtį, turi gajas sąsajas su taikomosios matematikos uždaviniais, nemažą pedagogų ir mokslininkų įdirbį šioje srityje. Sukurtus algoritmus reikia vykdyti kompiuteriu. Todėl aktualus programavimo kalbos pasirinkimas.

Nuo pat informatikos atsiradimo Lietuvos mokyklose algoritmų buvo siūloma mokyti Paskalio kalba (Dagienė, 2003): tai skatino mokyklose naudojami informatikos, vėliau – informacinių technologijų vadovėliai.

Reformuotos Lietuvos mokyklos informacinių technologijų bendrojoje programoje mokant algoritmų nesiūloma jokių konkrečių priemonių. Tačiau dauguma vadovėlių remiasi minimaliomis Paskalio kalbos konstrukcijomis. Naudojami sveikieji ir realieji skaičiai, loginis tipas, priskyrimo ir sąlyginis sakiniai, ciklai (Lietuvos ..., 2002; Pradinio ..., 2008; Vidurinio ..., 2011).

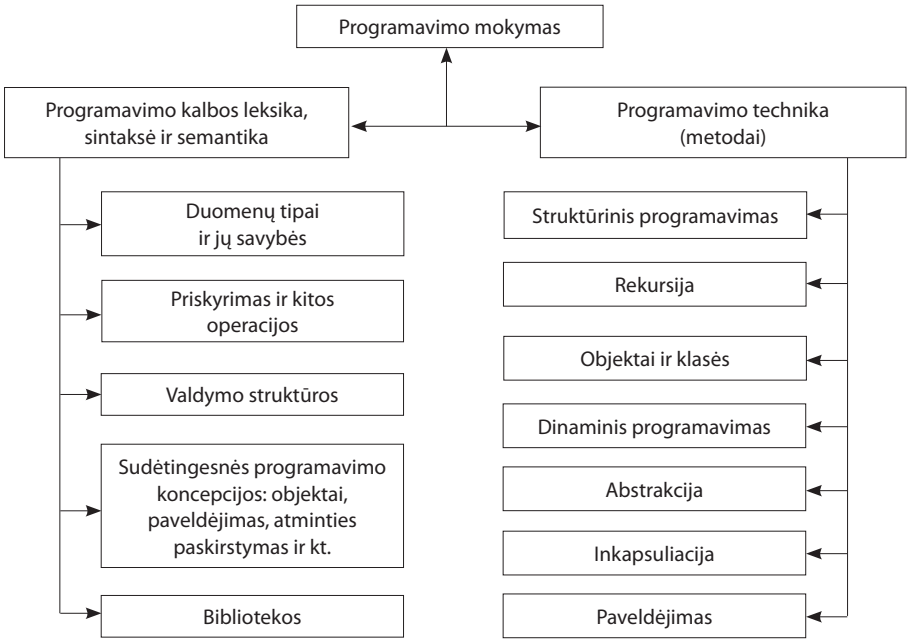
Straipsnio **tikslas** – apžvelgus programavimo mokymosi pradžios problemas, atlikti pirmųjų programavimo kalbų ir aplinkos analizę. Straipsnyje naudojami mokslinių dokumentų apžvalgos ir sintezės tyrimo metodai, remiamasi lyginamosios analizės metodologiniais principais, taip pat daugiamete darbo su mokiniais mokant programavimo patirtimi ir įžvalga.

Nors straipsnyje analizuojama apskritai pirmosios programavimo kalbos problematika mokantis nepriklausomai nuo besimokančiųjų amžiaus, tačiau daugiau dėmesio skiriama vyresniosioms bendrojo lavinimo mokyklos klasėms. Lietuvos mokyklose pasirinktinis programavimo modulis siūlomas pradedant nuo devintos ar dešimtos klasės (15–16 metų). Panašaus amžiaus mokiniai programavimo pradeda mokytis ir Jaunųjų programuotojų mokykloje.

Programavimo mokymosi ypatumai

Išmanyti programavimo techniką, metodus būtina kiekvienam programuotojui (Fischer, 2004). Programuotojas turėtų žinoti ir gebėti įvertinti alternatyvas, kai kalbama apie programinę įrangą. Programavimo mokymą galima skirstyti į dvi pagrindines kategorijas: 1) programavimo kalbos sintaksės, semantikos išmokimas, 2) gebėjimas taikyti programavimo techniką ir metodus (1 pav.).

Programavimo kalbos sintaksė ir semantika pradedančiajam reikalinga, kad galėtų pradėti spręsti uždavinius. Svarbu ne tik išmokti sintaksės konstrukcijas, bet



1 pav. ***Svarbiausi programavimo pradmenų kurso konceptai***

ir gebėti jas taikyti. Pagrindinės programavimo kalbų konstrukcijos: duomenų tipai ir jų savybės, priskyrimo ir kitos operacijos, valdymo struktūros, objektai, paveldėjimas, atminties valdymas, bibliotekos ir kt. Šiuolaikinės programavimo kalbos, ypač taikomos mokytis, dažniausiai integruotos su kompiliatoriumi, turi savitą aplinką. Todėl šiuolaikinės programavimo kalbos paprastai suprantamos kartu su jų aplinka.

Programavimo technika (metodai) apima gebėjimą naudoti pagrindines koncepcijas ir techniką: pabrėžiama inkapsuliacija, pranešimų perdavimas, rekursija, paveldėjimas, lygiagretus programavimas, programos projekto šablonų „Modelis, peržiūra ir valdymas“ struktūra, algoritmo sudarymo modelis „Skaldyk ir valdyk“, dinaminis programavimas. Mokantis programavimo technikos, kalba yra tik priemonė bendroms programavimo sąvokoms išreikšti ir taikyti. Programavimo technikos koncepcijos turi būti įvedamos taip, kad nepriklausytų nuo kalbos. Kai pradedama mokytis programavimo technikos, reikia pasirinkti kalbą, kuri nebūtų per daug egzotinė ar abstrakti. Jei mokyti vartojama abstrakti kalba, kyla pavojus, kad besimokantysis gali vėliau nesugebėti įveikti realių situacijų.

Kartais būna klaidinga programavimo mokymosi samprata – laikoma, kad tai mokymas užrašyti uždavinio sprendimą programos tekstu naudojantis programavimo kalbos konstrukcijomis. Programos rašymas – tik vienas iš daugelio programavimo įgūdžių.

Itin pabrėžtinas gebėjimas skaityti ir suprasti programų tekstus – juk programuotojas nemažai laiko praleidžia nagrinėdamas kitų parašytas programas (Mannila, 2007). Galima manyti, kad mokantis rašyti programas savaime išmokstama jas skaityti, stebėti programos vykdymą. Ta-

čiau tyrimai rodo visai ką kitą, pavyzdžiui, nustatyta, kad yra labai maža gebėjimo parašyti programą ir gebėjimo ją skaityti koreliacija (Winslow, 1996). D. Spinelis savo straipsnyje analizuoja priežastis, kodėl sunku skaityti programą: „rašydami programos tekstą, galime rasti daug įvairių būdų, tinkančių mūsų programos specifikacijai įgyvendinti, laipsniškai mažinant alternatyvas, mažėja ir mūsų pasirinkimas [...]. Paskutinis mūsų įvedamas simbolis dažniausiai yra vienintelis, kuris leidžia gauti teisingą programą. O skaitant programą, kiekvienas skirtingas kelias, kuriuo mes interpretuojame sakinius ar struktūras, atveria daug naujų interpretacijų likusiam programos tekstui, ir tik vienintelis kelias yra teisinga interpretacija. Programą rašome eidami link pasirinkimų medžio šaknų, o skaitydami programą – atvirkščiai – einame link šakų“ (Spinellis, 2003). Taigi mokantis programavimo reikia nepamiršti ugdymosi skaityti ir suprasti kitų parašytas programas.

SOLO ir Bloomo taksonomijų taikymas mokant programavimo

Keletas mokslininkų panaudojo SOLO (*Structure of the Observed Learning Outcomes*) taksonomiją siekdami aprašyti pradedančių programuotojų pasiekimų lygį. Programavimo mokymosi kelią galima išreikšti penkiais SOLO lygmenimis (Lister, Simon ir kt., 2006). SOLO taksonomija apima du kompleksinius tarpsnius: „paviršinį“, arba kiekybinį, kurį sudaro ikistruktūris, vienastruktūris ir daugiastruktūris lygmenys, ir „gilųjį“, arba kokybinį, kurį sudaro sąryšinis ir išplėtotas abstraktusis lygmenys. Remiantis straipsniu (Lister, Simon ir kt., 2006) trumpai aptariami šie lygmenys.

1. **Ikistruktūris**. Besimokantys nesupranta studijuotos medžiagos. Spręsdamas užduotį jis negali rasti ir pateikti prasmingo sprendimo (atsakymo), naudoja nereikšmingą informaciją, t. y. dar nelabai skiria esminius programavimo konceptus, klaidingai supranta dalį programavimo konstrukcijų, pavyzdžiui, painioja masyvo indeksą su pačiu masyvo elementu.

2. **Vienastruktūris**. Pradedantysis susikoncentruoja į vieną aspektą ar studijuotos medžiagos elementą, jis parodo teisingai suprantąs kai kuriuos, bet ne visus, uždavinio aspektus. Vis dėlto, kai pradedantysis įgyja dalinį supratimą, jis tampa raštingas toje srityje. Pavyzdžiui, gali teisingai pritaikyti dalį programavimo konstrukcijų kai kuriems uždaviniams spręsti.

3. **Daugiastruktūris**. Besimokantysis jau susitelkia į keletą svarbių aspektų, tačiau jie nėra tarpusavyje siejami. Suprantamos sąvokos pateikiamos neorganizuotai ir nenuosekliai. Čia pradedantysis rodo suprantąs visas uždavinio dalis, tačiau ne visai supranta šių dalių ryšį.

4. **Sąryšinis**. Besimokantysis sujungia kelias esmines detales ir jas integruoja į visumą, detales susieja su turimais gauti rezultatais, gerai supranta mokymosi medžiagą, sujungia uždavinio dalis į vientisą struktūrą ir panaudoja šią struktūrą uždaviniui spręsti, pavyzdžiui, išanalizavęs visą programos tekstą, jis gali įvardyti atskirus programos struktūrinius elementus, nusakyti šių elementų ryšį ir priklausomybę programoje.

5. **Išplėtotas abstraktusis**. Besimokantysis išmoktą medžiagą pateikia kaip bendrą struktūrą, kartu remiasi ir papildoma medžiaga. Rodomi aukščiausio lygmens pažintiniai gebėjimai, konkrečios medžiagos abstrahavimas ir teorijų formavimas,

besimokančiojo uždavinio sprendimo būdas išeina iš spręstino uždavinio rėmų, uždavinys susiejamas su platesniu kontekstu, pavyzdžiui, besimokantysis gali įvardyti ir komentuoti programos ribojimus ar išskirtines sąlygas.

SOLO taksonomija apibūdina mokymosi lygmenis, kaip jais kylama nuo žemiausiojo iki aukščiausiojo. Pažinimo srities Bloomo taksonomija leidžia pajausti būdų, kuriuos taiko pradedantieji ir patyrusieji spręsdami uždavinį, skirtumus. Bloomo klasikinė taksonomija apibrėžia šešis mokymosi lygmenis, kuriuos reikia laipsniškai pereiti (Bloom, 1956). Nemažai mokslininkų nagrinėjo Bloomo taksonomijos pritaikymą kompiuterių mokslui. Vieni analizavo taksonomijos taikymą mokantis programavimo (Lister, Leaney, 2003), kiti tyrė, ar ši taksonomija apskritai gali būti taikoma kompiuterių moksle (Johnson, Fuller, 2007; Fuller, Johnson ir kt., 2007), dar kiti daugiausia dėmesio skyrė taksonomijai taikyti vertinant žinias (Thompson, Luxton-Reilly ir kt., 2008; Starr, Manaris, Stalvey, 2008). Kiekvieno lygmens mokymosi veikla parodyta 2 paveiksle. Aktyvūs veiksmažodžiai nurodo intelektualią veiklą tam tikru lygmeniu.

Pradedant nuo pirmo ir antro lygmenų – žinių ir supratimo – mokomasi pagrindinių kompiuterio programų konceptų (sąvokų, principų), susipažįstama su programavimo kalbos leksika, sintakse, semantika, įskaitant ir aplinkos supratimą. Žinių lygmeniu besimokantysis turi įvardyti programos dalis, kas tai per programa, supratimo lygmeniu – savais žodžiais apibūdinti programos konceptus.

Su sunkumais susiduriama paprastai vėliau, kai pereinama į kitą lygmenį – taikymų. Šiame lygmenyje pradedantieji turi pritaikyti įgytas žinias kurdami programas įvairių pažįstamų situacijų kontekste.

Ketvirtas lygmuo – analizės – gana sudėtingas besimokantiesiems. Šiuo lygmeniu besimokantieji turi perprasti programų derinimo mechanizmą, pavyzdžiui, dinamiškai (programos vykdant) analizuoti klaidas ir bandyti priimti sprendimus mažiau žinomomis situacijomis.

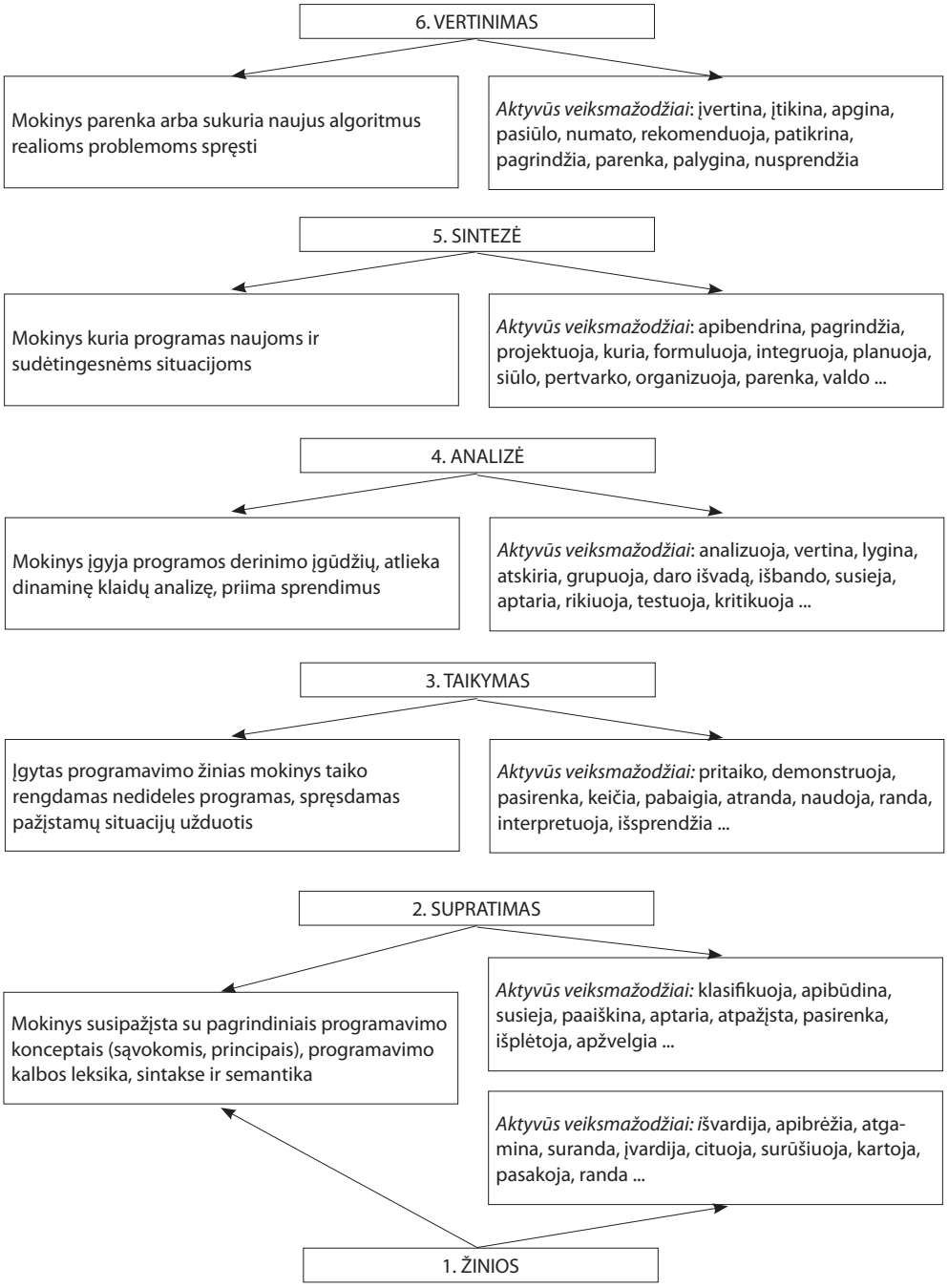
Penktas lygmuo – sintezės – gerokai viršija vidutinius besimokančiojo programuoti gebėjimus. Programuotojas turi sukurti sprendimus naujoms ir sudėtingoms situacijoms.

Šeštą lygmenį – vertinimą – įvaldo pažengusieji programuotojai, paprastai įvardijami ekspertais. Šiame lygmenyje kuriami nauji algoritmai, taikomos alternatyvios platformos, algoritmai, metodai ir pan., atsižvelgiant į jų tinkamumą esamam uždaviniui spręsti. Pateikus dvi tą pačią problemą sprendžiančias problemas, ekspertas gali įvardyti, kuri iš šių programų yra geresnė.

Kodėl sunku išmokti programuoti

Programavimas – sudėtingas intelektinis procesas, tad norint juo sudominti mokinius ar studentus, reikia pateikti jį paprastai, aiškiai, patraukliai. Tačiau kad ir kaip patraukliai būtų pateikiama, vien žinių ar gerų pavyzdžių neužtenka: būtina pačiam įsitraukti į procesą, lavinti įgūdžius, logiškai mąstyti. Daugybė pažinimo teorijų šalininkų bandė atsakyti į klausimą, kodėl tiek daug besimokančiųjų nesugeba išmokti programavimo? Štai keletas pastabų:

- sunku suprasti programų tikslus ir jų ryšį su kompiuteriu;
- sunku suprasti specifinės programavimo kalbos sintaksę ir semantiką (Robins, Rountree, Rountree, 2003);



2 pav. ***Bloomo taksonomijos taikymas mokantis programavimo***

- įgyjamas neteisingas programavimo konstrukcijų suvokimas;
- nesugebama spręsti problemų (Jenkins, 2002; Gomes, Mendes, 2007);
- nesugebama skaityti ir suprasti programų tekstus (Lister, Adams ir kt., 2004; Mannila, 2007).

Giliau tiriant programavimo mokymo problemas, galima išskirti penkis komponentus, lemiančius programavimo mokymo sunkumus: 1) mokymo metodai, 2) mokymo metodų taikymas, 3) besimokančiųjų gebėjimai ir nuostatos, 4) programavimo prigimtis, 5) psichologiniai motyvai.

Mokymo metodai. Programavimo mokymas vis dar nėra personalizuotas, mokytojo naudojami metodai nedera su besimokančiųjų mokymosi stiliais, dažnai dinaminių konceptų mokoma naudojant statinę medžiagą, mokytojas labiau susitelkęs į programavimo kalbos ir jos sintaksės mokymą, o ne į uždavinių sprendimą taikant programavimo kalbą ir aplinką.

Mokymosi metodų naudojimas. Besimokantieji naudoja netinkamus mokymosi metodus ar metodologiją, pavyzdžiui, daugelis jų mano, kad gali išmokti programuoti vien skaitydami programas, tačiau pagrindinė jų veikla turėtų būti skirta spręsti uždavinius, kita pastebima tendencija – besimokantieji nepakankamai daug dirba savarankiškai, kad įgytų programavimo kompetenciją.

Besimokančiųjų gebėjimai ir nuostatos. Besimokantieji turi būti įgiję ar norėti įgyti įvairių su programų kūrimu susijusių gebėjimų: problemų supratimo, turimų žinių susiejimo su problema, uždavinio ir jo sprendimo apmąstymo (refleksijos), atkaklumo sprendžiant uždavinius, mokėti taikyti pagrindines matematines ir logines žinias, konkrečias programavimo žinias (Gomes, Carmo ir kt., 2006). Pažymima, kad besimokantiesiems sunku gauti ne patį rezultatą, t. y. parašyti programą, bet pereiti patį kūrimo procesą. Daugelis pradedančiųjų netinkamai naudoja žinias rašydami pažingsninę specifikaciją natūralia kalba, t. y. neteisingai transformuoja natūralios kalbos semantiką į programavimo kalbą (Bennedsen, 2008).

Programavimo prigimtis. Programuojant reikia aukšto abstrakcijos lygio, programavimo kalbų sintaksė yra gana sudėtinga.

Psichologiniai motyvai. Besimokantieji neturi motyvacijos, paprastai jie pradeda mokytis programavimo sudėtingu jų gyvenimo periodu, pavyzdžiui, paauglystėje (Jenkins, 2002).

Norint, kad programavimo mokymasis taptų patrauklus ir paprastas, reikia mokytojo ir besimokančiojo pastangų, nuostatų, palankaus nusiteikimo. Taip pat svarbu pasirinkti tinkamas programavimo priemones: kalbą, aplinką.

Mokomosios programavimo kalbos ir aplinkos kriterijai

Programavimo mokymas yra neatsiejamas nuo programavimo kalbos ir jos aplinkos. Mokymo sėkmingumas daug priklauso nuo to, kokia programavimo kalba (be abejo, drauge su aplinka – ir toliau visur turima omenyje, kad mokomoji programavimo kalba neatsiejama nuo aplinkos) pasirenkama pirmiesiems žingsniams. Norint pasirinkti programavimo kalbą mokymui, pirmiausia reikia nustatyti kriterijus. Vieni iš plačiausiai žinomų mokomosios kalbos kriterijų suformuluoti prieš dešimtmetį ir dar nepaseno (Lawlis, 1997):

1. Mokomosios programavimo kalbos sintaksė turi nepriklausyti nuo techni-

nių priemonių ar nuo konkrečios operacinės sistemos.
2. Mokomosios programavimo kalbos sintaksė turi būti standartizuota, kompiliatoriai turi derėti su šiais standartais.
3. Programavimo kalbos aplinka turi palaikyti programinės įrangos kūrimo technologijas, riboti ar net uždrausti prastus metodus, skatinti programų priežiūros darbus.
4. Programavimo kalba drauge su aplinka turi veiksmingai palaikyti tam tikrų interesų taikymo sritį ar sritis.
5. Programavimo aplinka turi užtikrinti tam tikrą sistemos patikimumo ir saugumo lygį.
6. Mokomosios programavimo kalbos kompiliatorių vykdymas turi derėti su esama technologijų būsena.
7. Programavimo kalba drauge su aplinka turi turėti programinės įrangos kūrimo ir priežiūros priemones.

Laikytis nurodytų kriterijų – gana sudėtinga, nes skirtingomis situacijomis gali tekti naudoti skirtingas kalbas. Tačiau nesilaikant šių kriterijų kyla grėsmė programavimo mokymui atitrūkti nuo realių uždavinių, tapti teorine sritimi, prarasti ryšį su technologijų plėtra ir vykstančiais pokyčiais. Keletas svarbesnių pavojų:

1. Jeigu kalba ir aplinka priklausoma nuo konkrečios platformos, atsiranda mobilumo (perkeliamumo) problemų, ribojamos techninės ir programinės įrangos galimybės diegiant originalią sistemą arba būsimus naujinius.
2. Jeigu kompiliatorių vykdymas neatitinka standartinės kalbos sintaksės, tai vėlgi atsiranda mobilumo (perkeliamumo) sunkumų, apsunkinama atnaujinimų galimybė.
3. Jeigu programinei įrangai kurti naudojami netinkami metodai ir juos leidžiama naudoti programos kūrimo ir derinimo metu, tai dėl blogai parengtų programų testavimas ir priežiūra tampa itin problemiški, eikvojama daug laiko ir išteklių.
4. Menkas susietumas su taikymo sritimi – papildoma kliūtis programos teksto aiškumui ir skaitomumui.
5. Jeigu sistema kelia patikimumo problemų, ji ne tik atliks mažiau, nei tikėtasi, bet ir taps kur kas brangesnė per visą savo gyvavimo laikotarpį. Iškilus grėsmei sistemos saugumui, programos gyvavimas gali staigiai nutrūkti.
6. Sunku naudotis kompiliatoriais, kurie neatspindi naujų technologijos galimybių. Kai aplinka technologiškai pasenusi, sudėtinga kurti šiuolaikines programas. Senstelėjęs kompiliatorius gali trukdyti net suvokti programavimo konstrukcijas.
7. Atitinkamų automatizuotų kūrimo priemonių trūkumas komplikuoja kūrėjo produktyvumą ir sistemos kokybę.

Mokomosios programavimo kalbos (kartu su aplinka) kriterijus nagrinėjo daugelis mokslininkų, atlikta tyrimų, paskelbta nemažai mokslinių publikacijų (Jordan, 2006; Kölling, Koch, Rosenberg, 1995; Milbrandt, 1993; Parker, Chao ir kt., 2006; Mannila, de Raadt, 2006). Suomijoje buvo atliktas ketverius metus trukęs kokybinis tyrimas, kurio metu analizuotos studentų parašytos programos *Java* ir *Python* kalbomis (Grandell, Peltomäki, Salakoski, 2005). Mokslininkų numatyti mokomosios programavimo kalbos kriterijai yra gana skirtingi, tik nedaugelis jų visiškai sutampa (1 lentelė).

1 lentelė. ***Įvairių mokslininkų siūlomi tinkamos programavimo kalbos mokymo kriterijai***

Kriterijus	Kölling, Koch, Rosenberg (1995)	Milb-randt (1993)	Parker, Chao, Ottaway, Chang (2006)	Mannila, De Raadt (2006)
1. Naudojama ne tik mokyti			+	+
2. Tinka mokyti įvairiais lygiais			+	+
3. Tvarkingos, paprastos konstrukcijos	+			
4. Prasmingi baziniai žodžiai, leidžia vartoti ilgus prasmingus vardus		+		
5. Palaiko pagrindines objektinio programavimo koncepcijas	+		+	
6. Programavimo konstrukcijos leidžia išvengti programų klaidų	+			
7. Reikalauja aprašyti tipus ir kintamuosius	+	+		
8. Prasmingai sudarytos programavimo konstrukcijos	+			
9. Nepriklauso nuo operacinės sistemos			+	
10. Palengvina programų vykdymą	+			
11. Paprasta, aiški, nuosekli sintaksė	+	+	+	
12. Nedaug programavimo konstrukcijų, jos nedubliuojamos	+	+		
13. Leidžia atlikti sudėtingus skaičiavimus		+		
14. Leidžia programas skaidyti į gabalus			+	+
15. Siūlo vientisą programų kūrimo sistemą			+	+
16. Patikima, saugi, efektyvi			+	+
17. Lengvai išplečiama			+	+
18. Lengva pereiti prie kitų kalbų	+			
19. Numato pagalbą kuriant programą	+			+
20. Skatina rašyti iš karto teisingas programas				+
21. Greitas grįžtamasis ryšys		+		
22. Turi patogią ir paprastą kūrimo aplinką	+		+	
23. Integruota programų kūrimo aplinka				+
24. Programos kuriamos ne tik rašant tekstą				+
25. Sudaro galimybę tobulinti programas				+
26. Skatina programinės įrangos mokymąsi				+
27. Lengvai išmokstama	+	+		
28. Paprastas įvedimas ir išvedimas		+		
29. Turi patogius programų derinimo įrankius		+	+	
30. Gali tikti įvairiems įtaisams valdyti				+
31. Turi priemonių programuoti tinkluose			+	
32. Yra nemaža naudotojų bendruomenė				+
33. Patinka mokytojams			+	
34. Integruota su aplinka, visiems prieinama			+	
35. Integruota su aplinka, yra atvira, kiekvienas gali prisidėti prie jos plėtojimo			+	
36. Parengta geros mokymosi medžiagos			+	+
37. Nereikia finansinių išlaidų			+	

L. Mannila, M. de Raadt ir K. R. Parker buvo svarbu, kad programavimo kalba būtų tinkama mokymui įvairiais lygiais, tačiau pageidautina, kad kalba būtų naudojama ne vien mokymo tikslais (1, 2[1]). Įsivyravus objektinio programavimo paradigmai, M. Kölling, B. Koch ir J. Rosenberg pirmajai mokomajai programavimo kalbai kelia objektų įgyvendinimo kriterijų, jie nurodo, kad kalba turi palaikyti pagrindines objektinio programavimo koncepcijas (5), nes objektinės konstrukcijos yra ne papildomos kitų galimų struktūrų dalys, bet pagrindinės programuojant naudojamos abstrakcijos.

Nemažai dėmesio skiriama programavimo kalbų sintaksei – mokantis labai svarbus sintaksės aiškumas, paprastumas. (3, 4, 5, 6, 7, 8, 10, 11, 12). Nuoseklumas ir suprantamumas sudaro sąlygas sintaksės universalumui – ji bus naudojama panašioms semantinėms konstrukcijoms. Kalboje neturėtų būti konstrukcijų, kurios susijusios su naudojamos įrangos techninėmis ir programinėmis priemonėmis arba neturi semantinės vertės (11, 12).

Dažnai minimas programavimo kalbos efektyvumas ir lankstumas. G. Milbrandt svarbu, kad kalba leistų atlikti sudėtingus skaičiavimus (13), K. R. Parker, L. Mannila, M. de Raadt pabrėžia programos skaidomumą gabalais (14), vieningos sistemos užtikrinimą (15), kalbos patikimumą ir saugumą (16), išplečiamumo galimybę (17). M. Kölling ir kt. pažymi galimybę pareiti prie kitų plačiai naudojamų programavimo kalbų (18).

Minėta, kad programavimo kalba dažniausiai neatsiejama nuo aplinkos, jai skiriama daug dėmesio, daugelis tyrėjų pabrėžia aplinkos patogumą, kad besimokantieji galėtų susikaupti mokydamiesi programavimo koncepcijų, o ne perprasti pačią aplinką (24, 25, 26, 27, 28). Aplinka turi skatinti rašyti teisingas programas (20), teikti greitą grįžtamąjį ryšį (21) ir pagalbą rašant programas ir jas testuojant (19).

Žmogiškieji veiksniai taip pat svarbūs mokantis programavimo – pabrėžiamos programavimo kalbos naudotojų ir mokytojų bendruomenės (32, 33, 37, 38).

L. Mannila ir M. de Raadt (2006) atliko išsamius tyrimus ir suformulavo 17 kriterijų rinkinį, kuris buvo pasiūlytas žymiausiems keturių vadinamųjų mokymo programavimo kalbų kūrėjams Seymour Papert (LOGO kūrėjas), Niklaus Wirth (*Pascal* kūrėjas), Guido van Rossum (*Python* kūrėjas) ir Bertrand Meyer (*Eiffel* kūrėjas) įvertinti jų sukurtąsias kalbas. Pagal šį rinkinį buvo palygintos ir kitos mokymui skirtos programavimo kalbos (2 lentelė).

Visi 17 kriterijų suskirstyti į keturias grupes, apimančias tam tikras sritis: mokymąsi, kūrimą ir aplinką, pagalbą ir prieinamumą bei taikymą. Mokymosi atžvilgiu iš analizuojamų kalbų tinkamiausia yra *Eiffel* kalba. Ji tenkina visus keturis kriterijus (1, 2, 3 ir 4[2]), t. y. tinkama mokymui, gali būti naudojama su įvairiais fiziniais įtaisais, siūlo bendrą sistemą, skatina programinės įrangos mokymąsi kuriant programas. Beveik visos likusios programavimo kalbos netenkina arba tinkamumo mokymui (1), arba skatinimo mokytis programinės įrangos kriterijų (4). *Eiffel* programavimo kalba taip pat turi draugiškiausią aplinką ir yra patogi programoms rašyti. Ši kalba yra vienintelė, kuri turi vientisą programų kūrimo aplinką (8). Be to, *Eiffel* programavimo kalba yra vienintelė kalba, skatinanti

[1] Čia ir toliau skaitmenys skliausteliuose nurodo kriterijų numerius pagal 1 lentelę

[2] Čia ir toliau skaitmenys skliausteliuose nurodo kriterijų numerius pagal 2 lentelę

2 lentelė. ***Programavimo kalbų palyginimas pagal 17 kriterijų (Mannila, de Raadt, 2006)***

Kriterijai \ Programavimo kalba	C	C++	Eiffel	Java	JavaScript	Logo	Pascal	Python	VisualBasic
Mokymasis									
1. Tinka mokyti įvairiais lygiais	-	-	+	-	-	+	+	+	-
2. Gali tikti įvairiems įtaisams valdyti	-	-	+	+	+	+	-	+	+
3. Siūlo vientisą programų kūrimo sistemą	+	+	+	+	+	-	+	+	+
4. Skatina programinės įrangos mokymąsi	-	-	+	±	-	+	-	-	-
Aplinka									
5. Integruota ir interaktyvi	-	-	-	-	-	+	-	+	-
6. Skatina rašyti iš karto teisingas programas	-	±	+	±	-	-	-	±	-
7. Leidžia programas skaidyti į gabalus	+	+	+	+	+	+	+	+	+
8. Siūlo vientisą programų kūrimo sistemą	-	-	+	±	-	-	-	-	-
Pagalba ir prieinamumas									
9. Yra nemaža naudotojų bendruomenė	+	+	+	+	+	-	-	+	+
10. Integruota su aplinka, yra atvira, kiekvienas gali prisidėti prie jos plėtojimo	-	-	-	-	-	-	-	+	-
11. Yra nuolat palaikoma įvairiose aplinkose	+	+	+	+	+	+	+	+	-
12. Integruota su aplinka, visiems prieinama	+	+	+	+	+	+	+	+	-
13. Yra parengta geros mokymosi medžiagos	-	+	+	+	-	+	+	+	+
Tinkamumas programuotojams specialistams									
14. Naudojama ne tik mokymui	+	+	+	+	+	-	-	+	+
15. Lengvai išplečiama	+	+	+	+	-	-	-	+	+
16. Patikima, saugi, efektyvi	+	+	+	+	+	-	+	+	+
17. Programos kuriamos ne tik rašant tekstą	-	+	+	+	+	+	-	+	+
Iš viso:	**8**	**11**	**15**	**14**	**9**	**9**	**7**	**15**	**9**

rašyti teisingas programas (6). Visos nagrinėjamos kalbos leidžia skaidyti rašomą programą ir derinti bei vykdyti atskiras programos dalis (7) – tai mokymuisi svarbus kalbos kriterijus.

Kalbant apie programavimo kalbos draugiškumą ir prieinamumą, aiškiai išsiskiria *Python* kalba. Ji tenkina visus penkis šios dalies kriterijus. Iš lyginamų kalbų, turinčių integruotą aplinką, tik *Python* aplinka yra atviroji (10).

VisualBasic įvardijama nedraugiškiausia programavimo kalba ir aplinka. Nors ši kalba turi nemažą naudotojų bendruomenę (9), nors parengta neblogos mokymosi medžiagos (13), tačiau šios kalbos konstrukcijas sunku suvokti, aplinka nėra patogi ir teikianti pagalbą.

Iš 2 lentelės galima matyti, kad daugiausiai kriterijų tenkina *C++*, *Eiffel*, *Java* ir *Python* programavimo kalbos. Jos ir šiandien turėtų labiausiai tikti mokymuisi. Deja, nėra taip paprasta. Pavyzdžiui, *Eiffel* programavimo kalba nėra populiari daugelyje taikomųjų sričių, be to, ją intensyviau naudoja tik viena kita šalis. Kita vertus,

Pascal programavimo kalba, nors ir senstelėjusi, nors ir tenkina nedaug kriterijų, vis dar grakšti kai kuriomis semantinėmis konstrukcijomis ir populiari įvairiose šalyse (Lietuvoje, Rusijoje, Bulgarijoje ir kt.) mokant programavimo, ypač algoritmavimo.

Programavimo kalbos konstrukcijų įtaka mokant programavimo

Mokant programuoti daugiausia dėmesio kreipiama į programavimo įgūdžių lavinimą, o ne į programavimo kalbos ar jos sintaksės išmokimą. Tačiau vis dėlto programavimo kalbos konstrukcijų sintaksės ir semantikos mokymasis yra neatsiejamas nuo programavimo kalbos. Siekiant algoritmą užrašyti kompiuteriui suprantama kalba, reikalingos sintaksės ir semantikos žinios.

Reikia atsisakyti dalies nereikalingų ir specifines funkcijas atliekančių kalbos konstrukcijų nagrinėjimo. Antraip mokiniai gali būti apkrauti papildoma informacija, kuri sprendžiant tipinius programavimo uždavinius nereikalinga. Kuo kalbos sintaksė ar nagrinėjama jos dalis bus paprastesnė ir universalesnė, tuo mokymasis bus sistemingesnis.

Kita vertus, paprasta ir nedidelė kalbos sintaksė ne visada yra paranki ir gera. Kažkada mokyklose naudota *Basic* programavimo kalba turėjo itin mažą sintaksę. Ši kalba tiko užrašyti nedidelėms, nesudėtingoms programoms, tačiau sudarant sudėtingesnes programas atsirasdavo daug problemų, algoritmų užrašai tapo painūs, sunkiai skaitomi.

Vargu ar dažnas besimokantysis suvokia, kada jis mokosi programuoti, o kada studijuoja kalbą. Šiuo klausimu turėtų rūpintis mokytojas (dėstytojas), lavindamas besimokančiųjų gebėjimą skirti šias dvi mokymosi kryptis.

Mokantis algoritmų, būtina juos vizualizuoti. Galima pasinaudoti struktūrinėmis schemomis, tačiau jos iš esmės tinka tik pagrindinėms valdymo struktūroms suprasti. Sudėtingiems algoritmams reikia automatinių priemonių.

Nemažai programavimo objektų galima susieti su matematiniais objektais – duomenų struktūras su algebrinėmis sistemomis, valdymo struktūras su taisyklėmis ir teoremomis, nežinomuosius su kintamaisiais ir pan. Taip galima siekti analogijos su matematiniais objektais. Sakoma, kad žmogus suvokia ir atpažįsta matematinį objektą tik tada, kai moka jį išreikšti bent dviejų skirtingų raiškos sistemų simbolių rinkiniais (Duval, 2006). Gretinant matematinius ir programavimo objektus, galima teigti, kad reikalingos dvi raiškos sistemos (kalbos).

Pradedantiesiems pirmoji raiškos sistema galėtų būti nesudėtingi simboliai, struktūrinės schemos, o vėliau tas pats objektas pasinaudojus programavimo kalba perkeliamas į kompiuterį. Tada nuo vienos programavimo kalbos nesunku pereiti prie kitos. Toks mokymas būtų labiau skirtas programoms kurti, o naujos konstrukcijos nagrinėjamos tada, kai žinomų nepakaktų objektams realizuoti. Taip teorinės žinios būtų taikomos praktikoje, o ne liktų nepanaudotos ir galiausiai pamirštos.

Mokomoji programavimo kalba turi būti lengvai perprantama, aiškios struktūros, universali ir galinti atlikti sudėtingus skaičiavimus (Milbrandt, 1993). Itin svarbi programavimo kalbos aplinka: greitas grįžtamasis ryšys, patogios derinimo ir programos tikrinimo priemonės.

Programavimo kalbos sintaksės svarba mokymui

Programavimo srityje, kaip ir kiekvienoje veikloje, niekas, ypač pradedantieji, nėra apsaugoti nuo klaidų darymo. Analizuojant klaidas ieškoma priežasčių, įgyjama patirties, formuojasi kompetencija. Ne iš karto pavyksta parašyti veikiančias programas. Jei iš klaidų mokomasi, jos daro postūmį mokantis, todėl mokiniams turi būti sudarytos galimybės stebėti ir analizuoti savo klaidas. Iš dalies šios galimybės priklauso nuo programavimo kalbos aplinkos. Mokiniai turėtų mokėti algoritmo teisingumą vertinti ne tik pagal galutinius rezultatus, bet ir pagal tarpinius. Norint patikrinti algoritmą, aptikti ir taisyti klaidas, tarpiniai algoritmo rezultatai dažnai būna vertingesni už galutinius. Todėl kalba, kuria mokomasi programuoti, turėtų turėti paprastas įvedimo ir išvedimo konstrukcijas, kad tarpinius algoritmo darbo rezultatus būtų nesunku pavaizduoti kompiuterio ekrane.

Jei mokinys, norėdamas pamatyti tarpinius rezultatus, turės atlikti sudėtingus veiksmus, pavyzdžiui, skaičiui nurodyti tikslią vietą, žymimą specialiais simboliais, formatą ar kreiptis į papildomas funkcijas, kurios skaičių paverstų simboliu, tai dalis brangaus laiko, skirto uždaviniui spręsti, būtų iššvaistyta rezultatams parodyti ekrane. Jei sprendimo laikas ribotas, pirmiausiai nukentėtų vertinimas. Situacijai kartojantis besimokantieji imtų formuotis neigiamą požiūrį į programavimą. Pastebima, kad esant sudėtingesnei sintaksei besimokantysis nejučiomis ima programuoti bandymų ir klaidų būdu, kuris tikrai nėra tinkamas būdas mokytis.

Programavimo kalbų sintaksės paprastumą ir tinkamumą pradedantiesiems galima matyti iš pavyzdžio, vaizduojančio pranešimą ekrane.

C++

```
#include <iostream.h>

\\main: išveda paprastą pranešimą
void main ()
{
count << "Sveikas, pasauli!" <<
endl;
}
```

Java

```
Class Sv {
Public static void main {String arrgs[]) {
System.out.println("Sveikas, pasauli!");
}
}
% javac Sv.java
% java Sv
```

Pascal

```
writeln('Sveikas, pasauli!');
```

Python

```
Print "Sveikas, pasauli!"
```

C++, Pascal, Python ir *Java* – populiariausios programavimo kalbos, naudojamos ir mokyti, ir taikomiesiems darbams atlikti. Iš pateikto pavyzdžio matoma, kad norint *Java* ir *C++* kalba pavaizduoti paprastą pranešimą, reikia parašyti gana nemažai programos teksto ekrane. Daug teksto – didesnė tikimybė įvelti sintaksės ar loginę klaidą. Be to, trumpą programą visada lengviau suprasti nei ilgą. *Pascal* ir *Python* kalba užrašyti algoritmai trumpi ir aiškūs. Nesudėtinga sintaksė tei-

3 lentelė. ***Programavimo kalbų populiarumas (pagal Tiobe svetainę www.tiobe.com)***

Vieta *2010 m. liepos mėn.*	**Vieta** *2009 m. liepos mėn.*	**Vietos pokytis**	**Programavimo kalba**	**Reitingas** *2010 m. liepos mėn.*	**Pokytis nuo** *2009 m. liepos mėn.*
1	1	=	Java	18.673%	-1.78%
2	2	=	C	18.480%	+1.16%
3	3	=	C++	10.469%	+0.05%
4	4	=	PHP	8.566%	-0.70%
5	6	⇧	C#	5.730%	+1.19%
6	5	⇩	(Visual) Basic	5.516%	-2.27%
7	7	=	Python	4.217%	-0.22%
8	8	=	Perl	3.099%	-1.10%
9	21	⇧⇧⇧⇧⇧⇧⇧⇧⇧	Objective-C	2.498%	+1.99%
10	9	⇩	JavaScript	2.432%	-1.08%
11	11	=	Delphi	2.323%	+0.33%
12	10	⇩⇩	Ruby	1.982%	-0.59%
13	12	⇩	PL/SQL	0.772%	-0.12%
14	13	⇩	SAS	0.701%	-0.09%
15	15	=	Pascal	0.639%	-0.07%
16	17	⇧	Lisp/Scheme/Clojure	0.622%	+0.01%
17	20	⇧⇧⇧	MATLAB	0.581%	+0.07%
18	16	⇩⇩	ABAP	0.548%	-0.15%
19	19	=	Lua	0.535%	+0.00%
20	28	⇧⇧⇧⇧⇧⇧⇧⇧	PowerShell	0.493%	+0.17%

kia daugiau galimybių algoritmams studijuoti.

Programavimo kalbų reitingavimo svetainės *Tiobe.com* (www.tiobe.com) duomenimis, 2010 metų liepą populiariausia programavimo kalba buvo *Java*. Kitų kalbų vieta ir kaita per metus parodyta 3 lentelėje. Šioje svetainėje renkama populiariausia metų kalba: 2009 m. laimėtoja buvo *Go*, 2008 m. – *C*, 2007 m. – *Python*, 2006 m. – *Ruby*, 2005 m. – *Java*, 2004 m. – *PHP*, 2003 m. – *C++*, 2000 m. – vėl *C*. Svetainės duomenimis, 1985 m. tik trys programavimo kalbos, dabar esančios populiariausių kalbų dešimtuke, buvo tarp populiariausių. Tai – *C* (ji buvo pirmoji), *C++* (buvo 10) ir *Visual Basic* (buvo 4). Kokios programavimo kalbos buvo populiarios 2005 m., 1995 m. ir 1985 m., galima matyti 4 lentelėje.

Kalbant apie *Python* programavimo mokymo kalbą, reikia pasakyti, kad ji turi nemažų pranašumų. Programuojant šia kalba sugaištama mažiau laiko, yra mažiau sąvokų, dalis programos teksto komponuojama iš atskirų dalių. Tai didina produktyvumą ir teikia galimybę daugiau laiko skirti svarbesniems dalykams.

Kitas svarbus aspektas mokantis programuoti – algoritmo teksto struktūra. Kai nesuvokiama programos struktūra, nematoma logikos tarp dalių, sunku parašyti pro-

4 lentelė. ***Programavimo kalbų populiarumas prieš 5, 15 metų ir 25 metus (www.tiobe.com)***

Programavimo kalba	Užimama vieta			
	2010 m.	*2005 m.*	*1995 m.*	*1985 m.*
Java	1	2	–	–
C	2	1	1	1
C++	3	3	2	10
PHP	4	4	-	-
C#	5	7	-	-
(Visual) Basic	6	6	3	4
Python	7	8	24	-
Perl	8	5	7	-
Objective-C	9	42	-	-
JavaScript	10	9	-	-
Lisp/Scheme/Clojure	16	15	9	2
Ada	28	17	6	3

gramą. *Python* kalba rašomas programos tekstas lygiuojamas automatiškai. Suomijoje atlikto tyrimo metu (tai longitudinis kokybinis tyrimas, trukęs ketverius metus, kurio metu buvo analizuojamos programos, parašytos per programavimo egzaminą 2002–2003 m. *Java* ir 2004–2005 m. *Python*. 30 *Java* ir 30 *Python*; studentams dėstė tie patys dėstytojai ir tą patį programavimo kursą) buvo nustatyta, kad studentai, programas rašę *Python* kalba, įvėlė tik dvi sintaksės klaidas, o *Java* kalba – devyniolika klaidų (Grandell, Peltomäki, Salakoski, 2005; Mannila, Peltomäki, Salakoski, 2006). Tikrinant programas nustatyta, kad dažniausiai pasitaikiusios priežastys, dėl kurių nesikompiliavo *Java* kalba parašytos programos, buvo praleisti skliaustai, kabliataškiai, klaidingai parašyti baziniai žodžiai. Loginių (semantinių) klaidų skaičius taip pat vertas dėmesio: *Python* kalba programavę padarė 17 loginių klaidų, *Java* – 40 tokių klaidų.

Testavimo ir koregavimo priemonės ir grįžtamasis ryšys

Mokantis programuoti, būtina suprasti loginius atskirų programos dalių ryšius. Tai padaryti sunkiau, jeigu ryšiai sudėtingi, daugiareikšmiai. Sunkumų sukelia ir didelis išimčių skaičius. Gerai, kai daugelis funkcijų ir procedūrų yra universalios, tinkamos ir taikomos įvairiu kontekstu.

Mokantis rašyti programas, svarbi programavimo aplinka: turi būti priemonės kintamųjų reikšmėms stebėti, algoritmams pažingsniui vykdyti, stabdyti. Šitaip daug lengviau stebėti programos darbo rezultatus, nustatyti klaidas ir ieškoti jų priežasčių.

Mokantis programuoti patariama prieš rašant programą numatyti, kokių prireiks kintamųjų, kokios procedūros ar funkcijos bus naudojamos, tačiau realybė dažnai būna kitokia. Pradėjus rašyti programą, prireikia vis naujų kintamųjų, procedūrų, funkcijų, valdymo ir duomenų struktūrų. Čia gelbsti dinaminis kintamųjų aprašy-

mas. Pagrindinis tokių kintamųjų aprašymo trūkumas galėtų būti pavojus tą patį kintamąjį panaudoti skirtingiems duomenims, tačiau tyrimo metu nerasta nė vienos dinaminį kintamųjų apibrėžimą palaikančios *Python* parašytos programos, kurioje būtų pastebėta tokia klaida. Studentai, rašę programas *Java*, visus kintamuosius stengėsi aprašyti programos pradžioje (Grandell, Peltomäki, Salakoski, 2005). Programuojant *Python* kalba kintamųjų aprašyti nereikia. 20 proc. *Java* programų rasta kintamųjų naudojimo klaidų: 13 proc. sudaro kintamųjų aprašų klaidos (parinktas netinkamas duomenų tipas), 7 proc. – neaprašytų kintamųjų naudojimas. Įdomu, kad *Python* kalba parašytose programose nebuvo nė vienos kintamųjų naudojimo klaidos.

Greitą reagavimą į programos veiksmus garantuoja interpretuojamos programavimo kalbos. Kaip rodo tyrimai (Grandell, Peltomäki, Salakoski, 2005), kompiliuojama kalba programavę studentai padarė daug daugiau sintaksės klaidų. Gali pasirodyti keista, nes kompiliatorius praneša programuotojui apie sintaksės klaidas. Tačiau dažnai iš kompiliatoriaus pranešimų sunku nustatyti klaidos šaltinį, įspėjimai ir pranešimai apie klaidas parodomi tik sukompiliavus programą. Stengiantis suprasti ir ištaisyti visas sintaksės klaidas, švaistomas laikas.

Kompiliatoriaus pranešimai kartais gali klaidinti, o ne padėti. Pradedančiajam „iššifruoti“ kompiliatoriaus pranešimus yra daug sunkiau nei turinčiajam patirties. Dažnai klaida pasirodo esanti visai kitoje vietoje, nei nurodoma kompiliatoriaus pranešime. Taip gali atsitikti dėl to, kad viena klaida sukelia kitas, apie kurias ir praneša kompiliatorius. Galima įsivaizduoti, kaip jaustųsi besimokantysis, jei, palikęs menką sintaksės klaidą ir sukompiliavęs programą, pamatytų pranešimą su daugybe klaidų. Jis gali prarasti motyvaciją. To išvengiama naudojant interpretatorių. Tačiau būna ir taip, kad neaptiktos sintaksės klaidos patenka į semantines, kurios aptinkamos tik tikrinant programą testais. Šiuo atveju situacija dar keblesnė – programa veikia, duoda klaidingus rezultatus, o klaidą surasti pradedančiajam gali būti labai sudėtinga. Vis dėlto minėtame tyrime tokių atvejų neužfiksuota.

Dėl greito reagavimo į programuotojo veiksmus, nesudėtingos sintaksės ir didelio priemonių rinkinio *Python* programavimo kalba galima rašyti nesudėtingas, tačiau įdomius veiksmus atliekančias programas (Back, Mannila ir kt., 2007).

Išvados

Mokytis programuoti nelengva. Dar sunkiau išmokti. Vienas iš svarbiausių programavimo mokymosi tikslų – kurti programinę įrangą, taikomąsias programas, to reikia nepamiršti formuojant mokomuosius programavimo modulius.

Programavimo mokymas apima programavimo kalbos leksikos, sintaksės, semantikos, programavimo metodų ir metodikos mokymą, taip pat gerą programavimo aplinkos įvaldymą. Pažymėtina, kad pastaruoju metu programavimo kalbos sąvoka dažnai vartojama bendrąja reikšme – drauge su kompiliatoriumi, aplinka, visomis programai sukurti reikalingomis priemonėmis.

Kadangi programuoti mokoma visame pasaulyje, vos ne visuose technikos universitetuose, tad šis klausimas tampa visuotinai problemiškas. Mokiniai vis daugiau naudoja kompiuterius. Norint, kad mokiniai ugdytųsi gilesnę technikos sampratą, kad galėtų modifikuoti programas pagal poreikius, reikia suteikti jiems

bent minimalių programavimo įgūdžių. Mokslininkai nuolatos ieško geriausių, sparčiausių, efektyviausių programavimo mokymo būdų. Suformuluotas rinkinys kriterijų, kuriuos turėtų tenkinti mokomoji programavimo kalba kartu su aplinka. Dalis kriterijų sugrupuota, apibendrinta.

Remiantis tirtais pavyzdžiais ir mokslininiais straipsniais, galima teigti, kad pradedantiesiems netinka sudėtingos sintaksės, sudėtingų konstrukcijų programavimo kalba. Pramonėje vyraujančios programavimo kalbos, pavyzdžiui, *Java*, *C++*, labiau tinka pažengusių programuotojų įgūdžiams gilinti. Svarbu atsižvelgti į programavimo kalbos aplinkos draugiškumą, patogumą.

Remiantis daugelio šalių mokslininkų tyrimais, ypač Suomijos universitetuose atliktu ilgalaikiu kokybiniu tyrimu, galima teigti, kad pradedantiesiems vienas iš geresnių pasirinkimų yra *Python* programavimo kalba, pasižyminti paprasta ir aiškia sintakse, struktūrizuotu dizainu, plačiu taikymu, turinti draugišką aplinką.

Mokant programavimo mokinius ar studentus būtina remtis bendraisiais pedagogikos ir psichologijos dėsniais, atsižvelgti į specifinius reikalavimus, numatyti problemas. Todėl mokant programuoti vertinga taikyti Bloomo, SOLO ir kitas pažinimo mokslų taksonomijas.

LITERATŪRA

ANDERSON, Johnathan; VAN WEERT, Tom (2002). Information and Communication Technology in Education. A Curriculum for Schools and Programme of Teacher Development. Unesco, 2002.

BACK, Ralph-Johan; MANNILA, Linda; PELTOMÄKI, Mia; SALAKOSKI, Tapio (2007). Improving Mathematics and Programming Education – The IMPED Initiative. *Seventh Baltic Sea Conference on Computing Education Research Proceses* (Koli Calling 2007), Koli National Park, Finland. CRPIT, Vol. 88, Lister, R. and Simon, Eds. ACS, p. 167–170.

BENNEDSEN, Jens (2008). Teaching and Learning Introductory Programming – A Model-Based Approach. *Dissertation for Dr. Philos degree in the Faculty of mathematics and Natural Sciences,* University of Oslo, Norway, 2008.

BLOOM, Benjamin S., et al. Taxonomy of Educational Objectives*:* Handbook I: Cognitive Domain, Longmans, Green and Company, 1956, ISBN: 0-582-28010-9.

DAGIENĖ, Valentina (2003). Informacinių technologijų taikymo švietime konceptualusis pagrindimas. *Informacijos mokslai*, 2003, t. 25, p. 127–133.

DUVAL, Raymond (2006). A Cognitive Analysis of Problems of Comprehension in a Learning of Mathematics. *Educational Studies in Mathematics.* February 2006, Vol. 61, No. 1–2.

FISCHER, Paul (2004). Teaching Programming to Beginners. [interaktyvus]. [žiūrėta 2010 m. liepos 19 d.]. Prieiga per internetą: <http://www2.imm.dtu.dk/~tb/fischer.pdf>.

FULLER, Ursula; JOHNSON, Colin G.; AHONIEMI, Tuukka; CUKIERMAN, Diana; HERNÁN-LOSADA, Isidoro; JACKOVA, Jana; LAHTINEN, Essi; LEWIS, Tracy L.; THOMPSON, Donna McGee; RIEDESEL, Charles; THOMPSON, Errol (2007). Developing a Computer Science-specific Learning Taxonomy. *ACM SIGCSE Bulletin*, 39 (4), p. 152–170. ISSN 0097-8418.

GOMES, Anabela; CARMO, Lilian; BIGOTTE, Emilia; MENDES, António (2006). Mathematics and programming problem solving, in *Proc. of the 3rd E-Learning Conf. – Computer Science Education,* September 2006, Coimbra, Portugal.

GOMES, Anabela; MENDES, A. J. (2007). Learning to program – difficulties and solutions. *International Conference on Engineering Education – ICEE 2007*, *Coimbra, Portugal*

GRANDELL, Linda; PELTOMÄKI, Mia; SALAKOSKI, Tapio (2005). High School Programming – A Beyond-Syntax Analysis of Novice Programmers. Difficulties. *Koli Calling International Conference on Computing Education Research Proceses*. 2005, p. 17–24.

JENKINS, Tony (2002). On the Difficulty of Learning to Program. *Proceedings of 3rd annual conference of the LTSN-ICS,* Loughborough University, United Kingdom, August 2002, p. 53–58.

JOHNSON, Colin G.; FULLER, Ursula (2007). Is Bloom's Taxonomy Appropriate for Computer Science? *6th Baltic Sea Conferenceon Computing Education Research Koli Calling 2006 Proceedings*, Department of Information Technology Uppsala University, Sweden, February 2007, p. 120–122, ISSN 1404-3203.

JORDAN, Patrick (2006). A Very Quick Comparison of Popular Languages for Teaching Computer Programming. [interaktyvus]. [žiūrėta 2010 m. liepos 23 d.]. Prieiga per internetą: <http://www.ariel.com.au/a/teaching-programming.html>.

KÖLLING, Michael; KOCH, Bett; ROSENBERG, John (1995). Requirements For A First Year Object-Oriented Teaching Language**.** *Proceedings of 26th SIGCSE Technical Symposium on Computer Science Education,* 1995, Nashville, Tennessee, U.S.A., SIGCSE Bulletin 27, 1, March, p. 173–177.

LAWLIS, Patricia K. (1997). Guidelines for Choosing A Computer Language: Support For The Visionary Organization. 2nd Edition, 1997, [interaktyvus]. [žiūrėta 2010 m. liepos 17 d.]. Prieiga per internetą: < http://archive.adaic.com/docs/reports/lawlis/content.htm>.

LIETUVOS bendrojo lavinimo mokyklos bendrosios programos ir išsilavinimo standartai (2002). XI–XII klasėms / patvirtinta Lietuvos Respublikos švietimo ir mokslo ministro 2002 m. rugpjūčio 21 d. įsakymu Nr. 1465. [interaktyvus]. [žiūrėta 2010 m. liepos 19 d.]. Prieiga per internetą: <http://www.pedagogika.lt/puslapis/bps.htm>

LISTER, Raymond; ADAMS, Elizabeth S.; FITZGERALD, Sue; FONE, William; HAMER, John; LINDHOLM, Morten; MCCARTNEY, Robert; MOSTRÖM, Jan Erik; SANDERS, Kate; SEPPÄLÄ, Otto; SIMON, Beth; THOMAS, Lynda (2004). A Multi-National Study of Reading and Tracing Skills in Novice Programmers. *SIGCSE Bulletin,* December 2004, Volume 36, Issue 4, p. 119–150.

LISTER, Raymond; LEANEY, John (2003). First Year Programming: Let All the Flowers Bloom. *5th Australasian Computer Education Conference (ACE2003)*, Adelaide, 2003, Vol. 20.

LISTER, Raymond; SIMON, Beth; THOMPSON, Errol; WHALLEY, Jacqueline. L.; PRASAD Christine (2006). Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. *SIGCSE Bull.*, 2006, 38(3), p. 118–122.

MANNILA, Linda (2007). Novices' Progress in Introductory Programming Courses. *Informatics in Education*, 2007, Vol. 6, No. 1, p. 139–152.

MANNILA, Linda; de RAADT, Michael (2006). An Objective Comparison of Languages for Teaching Introductory Programming. *Proceedings*, Koli Calling 2006, p. 32–37.

MANNILA, Linda; PELTOMÄKI Mia; SALAKOSKI, Tapio (2006). What About a Simple Language? Analyzing the Difficulties in Learning to Program. *Computer Science Education*, September 2006, Vol. 16, No. 3, p. 211–227.

MILBRANDT, George (1993). Using Problem Solving to Teach a Programming Language in Computer Studies. *Journal of Computer Science Education,* 1993.

PARKER, Kevin R.; CHAO, Joseph T.; OTTAWAY, Thomas A.; CHANG, Jane (2006). A Formal Language Selection Process for Introductory Programming Courses. *Journal of Information Technology Education,* 2006, Vol. 5, p. 133–151.

PRADINIO ir pagrindinio ugdymo bendrosios programos (2008). [interaktyvus]. [žiūrėta 2010 m. liepos 19 d.]. Prieiga per internetą: <http://www.pedagogika.lt/index.php?-469374926>

ROBINS, Anthony; ROUNTREE, Janet; ROUNTREE, Nathan (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, 2003, Vol. 13, No. 2, p. 137–172.

SPINELLIS, Diomidis (2003). Reading, writing, and code. *ACM Queue*, 2003, Vol. 1, Issue 7, p. 84–89.

STARR, Christopher W.; MANARIS, Bill; STALVEY, RoxAnn H. (2008). Bloom's Taxonomy Revisited: Specifying Assessable Learning Objectives in Computer Science. *SIGCSE'08*, March 12–15, 2008, Portland, Oregon, USA, ACM 978-1-59593-947-0/08/0003.

TAYLOR, Harriet Graham; AIKEN, Robert McLean; VAN WEERT, Tom. J. (1993). Guidelines for Good Practice: Informatics Education in Secondary Schools, IFIP Working Group 3.1, Geneva, 1993.

THOMPSON, Errol; LUXTON-REILLY, Andrew; WHALLEY, Jacqueline L.; HU, Minjie; ROBBINS, Phil (2008). Bloom's Taxonomy for CS As-

sessment. *Tenth Australasian Computing Education Conference (ACE2008) in Research and Practice in Information Technology Proceedings*, Wollongong, Australia, January 2008, Vol. 78.

VAN WEERT, Tom J. (1993). Guidelines for Good Practice: Integration of Information Technology into Secondary Education, IFIP Working Group 3.1, Geneva, 1993.

VIDURINIO ugdymo bendrųjų programų projektai (2010). [interaktyvus]. [žiūrėta 2010 m. liepos 19 d.]. Prieiga per internetą: <http://www.pedagogika.lt/index.php?1136240962>

WINSLOW, Leon E. (1996). Programming pedagogy, a psychological overview. *ACM SIGCSE Bull.*, 1996, Vol. 28, Issue 3, p. 17–22.

LEARNING PROGRAMMING: COMPARATIVE ANALYSIS OF LANGUAGES AND ENVIRONMENTS

Valentina Dagienė, Jūratė Urbonienė

A b s t r a c t

Developing the abilities to master modern technologies and skills for solving problems is among the most important capabilities of an educated future citizen of any society. Problem solving based on the learning of programming is a very important part in understanding the information technologies.

The question which language (together with environment) should be used in introductory programming has been discussed for many years. Several studies on the benefits of a certain language or comparisons between two languages have been conducted, but there is still a lack of systematic overviews of teaching and learning programming.

The paper discusses the features of programming teaching, the most important research trends in programming education over the world, identifies the problems and their solutions. Based on literature review and multiyear experience in the Young Programmers' School, the paper deals with programming teaching difficulties, especially with the selection of programming languages, learning and teaching methods, developing students' skills and attitudes, psychological motivation. Investigations show that the first language syntax has a significant impact on the further learning and develops a certain mindset.

The paper discusses a list of criteria based on an analyzis of research works all over the world. The criteria are used to compare some programming languages used at introductory programming courses. It focuses on the programming language suitable to start learning programming. Based on fundamental research works in thes area, the related criteria are organized, the key ideas are identified and summarized.

The commonly accepted cognitive skills, Bloom's taxonomy as well as the SOLO taxonomy and their application in teaching programming are discussed.