Rīgas 64. vidusskola

Programmēšanas paradigmu efektivitātes pētīšana

Zinātniski pētnieciskais darbs programmēšanā

Darba autors:
Rīgas 64. vidusskolas 12. klases skolnieks
Iaroslav Viazmitin

Darba vadītājs:
Rīgas 64. vidusskolas programmēšanas skolotājs
Edvards Bukovskis

Rīga 2024

Anotācija

Zinātniski pētniecisko darbu “Programmēšanas paradigmu efektivitātes pētīšana” izstrādāja Rīgas 64. vidusskolas 12.DIT klases skolnieks Iaroslav Viazmitin.

Zinātniski pētnieciskais darbs tika veikts ar mērķi izpētīt, kāda ir noderīgāka programmēšanas paradigmām programmēšanas valodai, kas strādātu visefektīvāk, t.i., visātrāk. Zinātniski pētnieciskajā darbā tiks veidotas kompleksās programmas dažādās programmēšanas paradigmās un tiks izvirzīti secinājumi, vai programmēšanas paradigmām ir efektivitātes statuss, kuru var ņemt vērā, izstrādājot programmētājiem programmas.

Atslēgas vārdi: efektivitāte, programmēšana, programmēšanas valoda, programmēšanas paradigma.

Abstract

Scientific research work “Researching the effectiveness of programming paradigms” developed by Iaroslav Viazmitin, a student of the 12.DIT class of Riga Secondary School No. 64.

Scientific research work was carried out with the aim of investigating which is more useful programming paradigms for the programming language that would work most efficiently, i.e. the fastest. In the scientific research work, complex programs will be created in various ways in programming paradigms and conclusions will be drawn whether programming paradigms have efficiency status that can be taken into account when developing programs for programmers.

Key words: efficiency, programming, programming language, programming paradigm.

Saturs

Ievads

Mūsdienās programmatūru kods ne vienmēr ir efektīvs. Kopā ar programmēšanas vienkāršošanu, pateicoties *zero-coding*[*] un dažādiem papildus rīkiem, kuri ne vienmēr ir efektīvi, ja programmētājs tos lieto tikai dažreiz vai tur, kur būtu ieteicams lietot efektīvākus rīkus, programmatūrām var ciest izpildīšanas efektivitāte. Rezultātā lietotājs var saskarties ar to, ka programma ērtākai lietošanai var prasīt apjomīgākus datora resursus (jaudīgāku procesoru vai videokarti, ātrāku vai vairāk operatīvās atmiņas), kaut gan, ja to uzprogrammēt citā veidā, var sasniegt labākus koda efektivitātes rezultātus. Viens no variantiem, kā varētu šo situāciju uzlabot ir veidot programmu tādā veidā, lai tā izpildītos labāk.

Pētījuma mērķis

Mērķis ir dot atbildi uz jautājumu "Kāda ir efektīvāka programmēšanas paradigma *Python* programmēšanas valodai?" un "Kādu programmēšanas paradigmu būtu ieteicams lietot?", ņemot vērā visus iegūtus datus pētījuma gaitā.

Pētījuma uzdevumi

1. Apkopot informāciju izplatītākām programmēšanas paradigmām;
2. Izstrādāt sarežģītu programmu, kuru varētu uzprogrammēt vairākās programmēšanas paradigmās;
3. Vairākas reizes palaist programmatūras un aprēķināt vidējo izpildes laiku;
4. Apkopot datus par katru programmēšanas paradigmu un izdarīt secinājumus.

Pētījumā izmantotās metodes

- Publisko informācijas avotu analīze;
- Blokshēmu izveide;
- Koda izstrāde.

Hipotēze

Ņemot vērā to, ka *Python* ir objektorientētā valoda, programmatūra šajā paradigmā izpildīsies ātrāk un starpība būs liela.

---

* *Zero-coding* (angļu val. "Nulle-kods") – programmēšanas veids, kurā nav nepieciešamības rakstīt kodu programmai. Programmēšana notiek vizuāli, parasti ar secīgu bloku savienošanu.

Teorija

Programmēšanas paradigmas

Programmēšanas paradigmas ir veids, kā strukturēti rakstīt kodu, kuru pielieto, lai kods būtu saprotamāks pašam programmētājam vai citiem programmētājiem. Programmēšanas paradigmas lielākoties tiek lietotas, ja ir garš kods, kuru ir jāsakārto. Kodā var būt daudz darbību un mainīgo, tādēļ nesagatavots cilvēks var apjukt kodā, ja tas nebūs sakārtots, lai programmētājiem būtu vienkāršāk to uztvert. Programmēšanas paradigmu lietošana arī var palīdzēt koda dokumentācijas sastādīšanā.

Kopumā ir sastopamas sekojošas programmēšanas paradigmas:

- imperatīvā,
- deklaratīvā,
- objektorientētā.

Imperatīvā programmēšanas paradigma

Imperatīvā programmēšanas paradigma apraksta soļus, kurus ir jāizdara, lai iegūt gala rezultātu vai veiksmīgi izpildīt darbību. Šādā programmēšanā paradigmā tiek veidots fokuss uz to, lai aprakstīt darbības.

Turpmāk tiks attēlots kods attiecīgajā virsraksta paradigmā, kas no skaitļiem masīvā aprēķinās to pāru skaitļu kvadrātu summu.

```
1 const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
2 let sum = 0;
3 for (let i = 0; i < numbers.length; i++) {
4   if (numbers[i] % 2 === 0) {
5     sum += numbers[i] * numbers[i];
6   }
7 }
8 console.log(sum);
```

1.att. Imperatīvās programmēšanas koda piemērs JavaScript valodā.

Deklaratīvā programmēšanas paradigma

Deklaratīvā programmēšanas paradigma apraksta pašu rezultātu, tā iegūšanu. Kods var sastāvēt no tām pašām darbībām, no kurām sastāvētu tāds pats kods imperatīvajā programmēšanas paradigmā, bet tas tiek pierakstīts citā formātā.

```javascript
const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
const evenNumbers = numbers.filter(num => num % 2 === 0);
const sum = evenNumbers.reduce((acc, curr) => acc + curr * curr, 0);
console.log(sum);
```

2.att. Deklaratīvās programmēšanas koda piemērs JavaScript valodā.

Objektorientētā programmēšanas paradigma

Objektorientētā programmēšanas paradigma organizē programmatūru ar objektiem, kuros ietilpst vērtības un darbības metožu veidā. Šāda programmēšana noder, piemēram, spēļu izstrādē, ja spēlē ir varonis un tā datus un darbības (piemēram, veselības līmenis, pārvietošanās u.c.) ir jāapraksta, kas ir visērtāk darāms ar objektiem.

```javascript
class NumberProcessor {
  constructor(numbers) {
    this.numbers = numbers;
  }

  getEvenNumbers() {
    return this.numbers.filter(num => num % 2 === 0);
  }

  getSumOfSquares() {
    return this.getEvenNumbers().reduce((acc, curr) => acc + curr * curr, 0);
  }
}

const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
const processor = new NumberProcessor(numbers);
console.log(processor.getSumOfSquares());
```

4.att. Objektorientētās programmēšanas koda piemērs JavaScript valodā.

Efektivitātes jautājums

Programmēšanā pastāv programmatūru efektivitātes problēma - tās var patērēt pārāk daudz resursu (piemēram, operatīvā atmiņa, enerģija u.c.), tādēļ tās tiek optimizētas, lai tiktu sasniegts tas pats rezultāts ar mazāku resursu patēriņu[4]. Šī problēma ir saistīta ar citu problēmu, ka efektivitātes labad kods var palikt grūtāks programmētāju saprašanai un tā loģika kļūst grūtāk uztverama. Tādēļ programmētāji lieto koda pieraksta veidus jeb programmēšanas paradigmas, lai kods gan būtu strukturēts un saprotams cilvēkam, gan saglabātu efektivitāti. Tomēr, pastāv vairākas programmēšanas paradigmas, kādēļ tas veido pretrunu starp efektivitāti un koda saprotamību - skaidrāka cilvēkam paradigma var raisīt mazefektīvāku programmatūru.

Ne visas programmēšanas valodas atbalsta visas programmēšanas paradigmas. Piemēram, programmēšanas valoda *C#* tiek uzskatīta par vairākparadigmu (angļ. val. *multi-paradigm*)

programmēšanas valodu[1], bet programmēšanas valoda *SQL*[8] par tikai deklaratīvo. Tomēr, dažādas programmēšanas valodas var būt orientētas uz noteiktām programmēšanas paradigmām. Piemēram, programmēšanas valodas *Python*[9] un *Java*[10] tiek uzskatītas par objektorientētām programmēšanas valodām – tās ir labāk pielāgotas objektorientētajai programmēšanas paradigmai.

Tomēr pielāgotība nenozīmē, ka programma šajā paradigmā izpildīsies daudz ātrāk, nekā citās. Tas nozīmē, ka šī programmēšanas valoda atbalsta galvenās vai tajā skaitā papildus noteiktas paradigmas principus un idejas. Ar šo mērķi tiks pētīta programmēšanas paradigmu efektivitāte, lai turpmāk izstrādāt efektīvākas programmatūras. Pateicoties tam, ka programmēšanas valodas ir daudzveidīgas, katrai varbūt savas noteiktas priekšrocības dažādās lietošanas jomās. Piemēram, robotizētais Marsa pētnieks izmanto *C* programmēšanas valodu[3], bet viedās mājas ierīcēm lieto *Python* un *Java*[4]. Tieši tādēļ, ka ne visas valodas der katrai jomai, šajā pētījumā tiek salīdzināta tikai paradigmu efektivitāte, nevis programmēšanas valodu. Pētījumā ir jāņem vērā arī tas, ka programmēšanas valoda *Python*, kā ir minēts, ir objektorientētā programmēšanas valoda, tādēļ objektorientētajai programmatūrai var būt priekšrocības virs citām programmēšanas valodām.

Praktiskā daļa

Kā var izmērīt efektivitāti?

Tā kā programmēšana ir cieši saistīta ar skaitļošanu, izstrādātās programmas veiks vienādas matemātiskas operācijas. Fiksētā datora specifikācija, uz kura tiks izpildītas programmas, kā arī fiksētās darbības, nozīmē to, ka tie neietekmēs pētījuma rezultātus.

Pašu efektivitāti var mērīt laika mērvienībā. Pastāv arī citi veidi, kā noteikt efektivitāti, piemēram, gala kompilētās programmatūras darbību skaits, strāvas patēriņš, kā arī visu šo lielumu kombinācijas, bet pētījumā tiks mērīts programmas izpildes laiks.

Fiksētais	**Neatkarīgais**	**Atkarīgais**
Izpildāmās darbības	Programmēšanas paradigma	Programmatūras izpildes laiks
Datora specifikācija		
Ārējo datora procesu ietekme		
Programmēšanas valoda		

Kā uzzināt koda izpildes laiku?

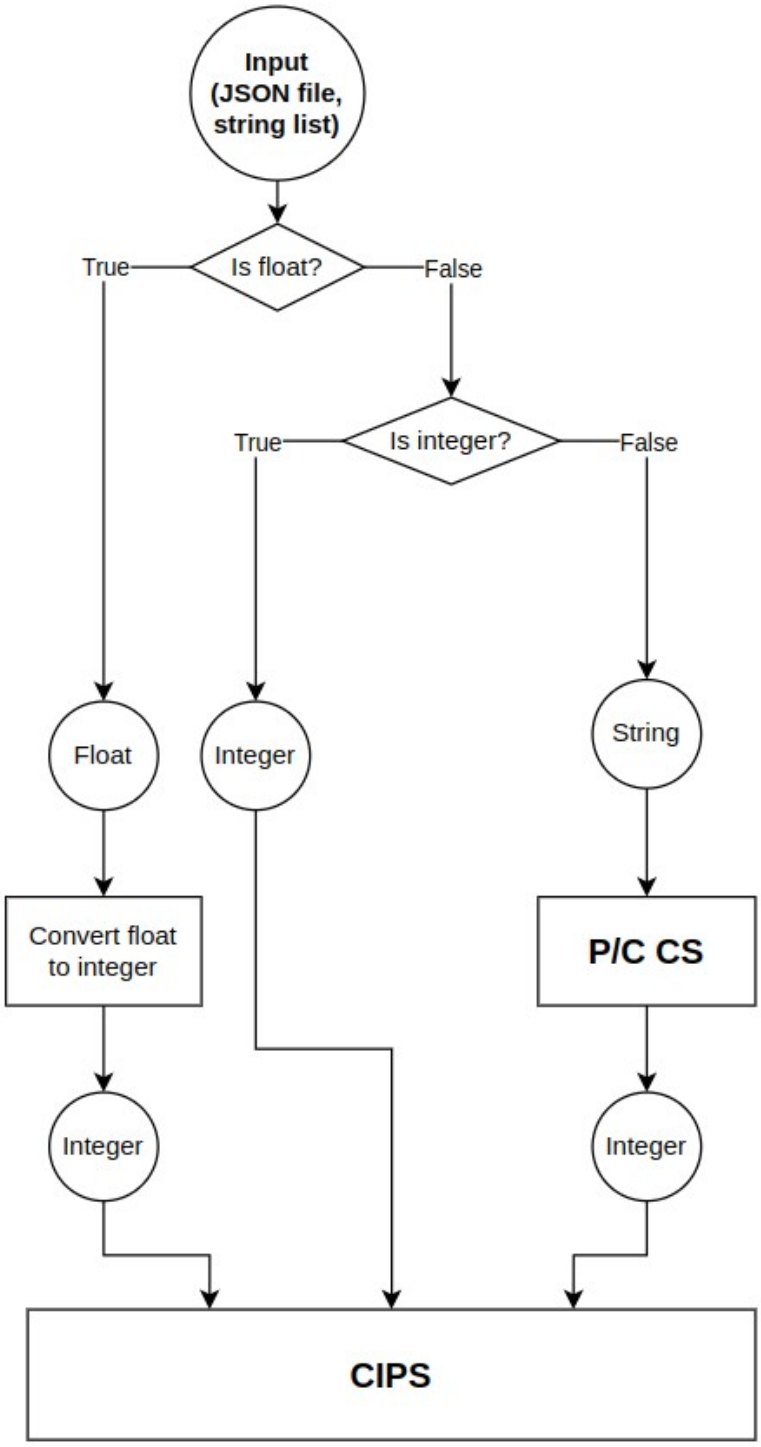
Lai uzzinātu laiku, kas ir nepieciešams koda izpildei, var rīkoties dažādi. Piemēram, ***Code::Blocks*** integrētajā izstrādes vidē (angļu val. *integrated development environment*, turpmāk – *IDE*) ir iebūvēta laika skaitīšanas funkcija, kurš bija nepieciešams programmas izpildei. Tajā arī tiek ietverts laiks, piemēram, kas tika zaudēts citu procesu darbības vai ietekmes uz šo programmu dēļ. Ja IDE nav iebūvētas funkcijas, kas mēra programmas izpildes laiku, tad var pašā kodā iebūvēt funkciju, kas to mērīs.

```
1   import timeit
2
3   # код, время выполнения которого нужно измерить
4   code_to_test = """
5   a = range(1000000)
6   b = []
7   for i in a:
8       b.append(i*2)
9   """
10
11  # вычисление времени выполнения кода
12  elapsed_time = timeit.timeit(code_to_test, number=100)/100
13  print('Elapsed time: ', elapsed_time)
```

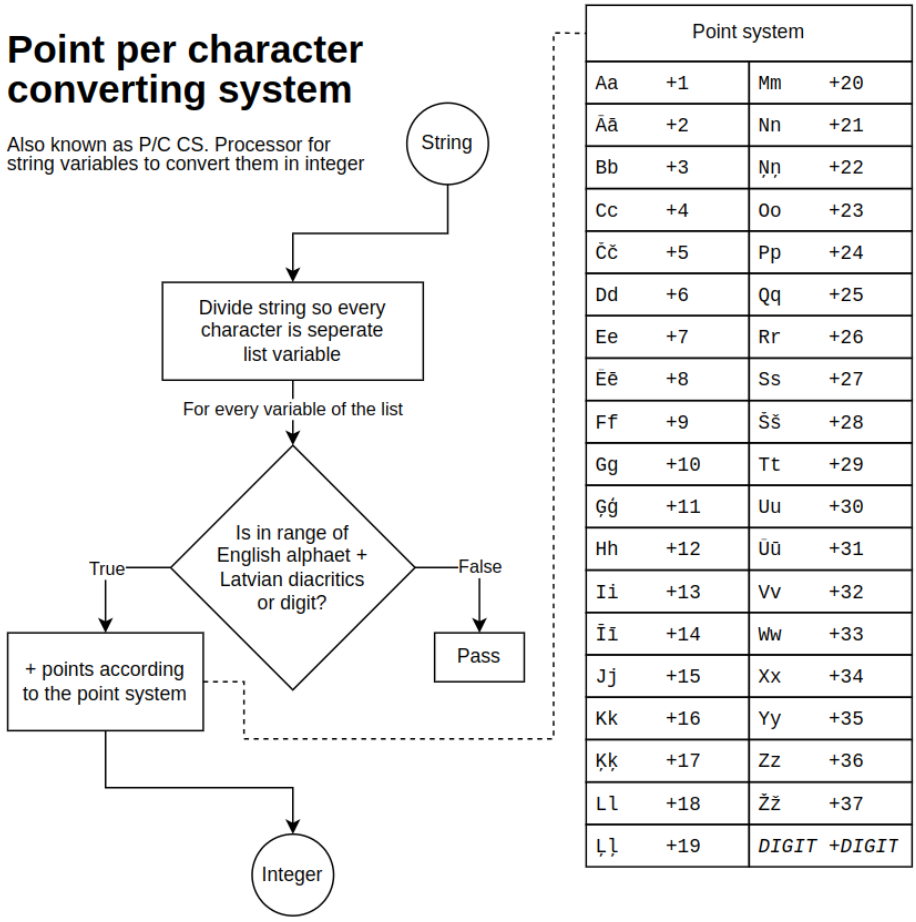
5.att. Piemērs Python programmēšanas valodā, ja IDE nav iebūvētas funkcijas laika uzskaitei.[1] *"elapsed_time" glabā mainīgā "code_to_test" saturētā koda izpildes laika vidējo aritmētisko no 100 izpildēm.*

Programmatūras izstrāde

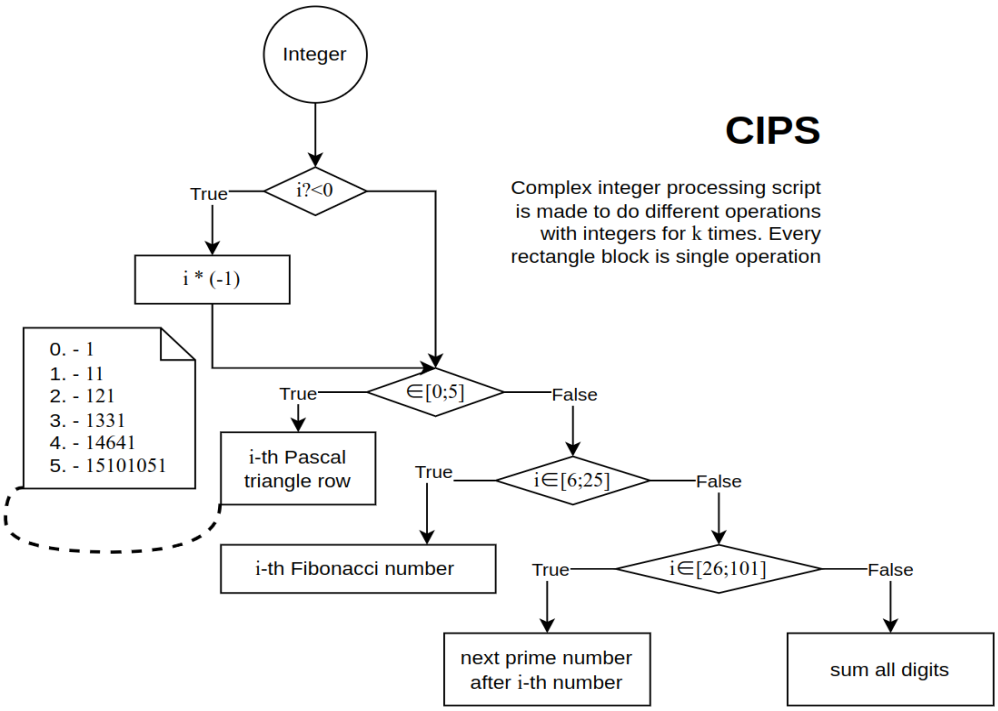
Uzdevums ir izveidot programmu, kas varētu apstrādāt vairāku tipu datus un izveidot pietiekamu slodzi datoram, lai tā būtu līdzīga sarežģītāku programmu darbam, kurās dažādu programmēšanas paradigmu lietošanas efekts būtu vairāk uztverams.



6. attēls. Shēma[l] (angļu valodā), kas attēlo koda darbības.



7. attēls. Shēma (angļu valodā), kas attēlo "P/C CS" bloka darbību.



8. attēls. Shēma (angļu valodā), kas attēlo "CIPS" bloka darbību. "K" koeficients ir 100000.

Galā tika izstrādātas programmas[II], kuru darbības gaita ir attēlots attēlos 6., 7., un 8. attēlos.

Visas programmatūras tiks palaistas 25 reizes uz operētājsistēmas *Linux* un laiks tiks mērīts ar rīku "time", ņemot vērā vērtību "real". Datoru (2 procesora kodoli, 8GB brīvpiekļuves atmiņas, 32GB lasāmatmiņas) nodrošina *GitHub Codespace* – tiek lietots īstais dators, bet attālinātajā veidā.

Objektorientētā paradigma	**Imperatīvā paradigma**	**Deklaratīvā paradigma**
8,996	6,449	9,877
9,987	5,949	9,707
8,614	6,028	9,709
8,631	6,042	9,566
8,530	6,017	9,909
8,764	6,018	9,611
8,372	5,951	9,663
8,759	6,101	9,515
9,427	6,031	10,310
8,657	5,959	10,037
8,489	5,876	9,860
8,662	5,876	9,762
8,620	6,054	9,729
8,648	5,909	10,079
8,597	5,975	9,842
8.513	5,981	9,931
8,587	5,890	10,383
8,491	5,936	9,945
8,907	5,969	9,883
8,858	5,973	9,696
8,718	5,835	9,712
8,703	5,967	9,885
8,515	5,911	9,983
8,554	6,080	9,974
8,600	6,112	9,864
Vidējais aritmētiskais: ***8,39***	*Vidējais aritmētiskais:* ***6***	*Vidējais aritmētiskais:* ***9,86***

Iegūtie rezultāti (sekundes) par izpildes laiku pēc katras programmas palaišanas.

Secinājumi

1. Imperatīvā programmēšanas paradigma ir efektīvāk par citām paradigmām *Python* programmēšanas valodai.
2. Apgalvojums, ka programmēšanas valoda *Python* ir objektorientētā, nav ekvivalents tam, ka programmatūra efektīvāk veiks savu darbību, ja tās kods tiks veidots saskaņā ar objektorientētās programmēšanas paradigmas vadlīnijām.
3. Imperatīvā programmēšanas paradigma, pateicoties savai vienkāršībai un efektivitātei, ir ieteicama lietošanai tajās jomās un programmēšanas valodās, kur tas ir iespējams, jo laika starpība paliek liela.

Izmantotā literatūra

1. Aleksejs K. (2023.) *Измерение времени выполнения кода в Python.* Iegūts no https://sky.pro/media/izmerenie-vremeni-vypolneniya-koda-v-python/
2. Daniel A. (2020.) *C# Development: The Complete Guide to Getting Started.* Iegūts no https://www.trio.dev/blog/csharp-development-guide
3. Kārlis Č. (n.d.) *Programmēšanas valodas.* Iegūts no http://www.ltn.lv/~karlisc/StudKursi/ProgValodas/Materiali/PV-01-Ievads.ppt
4. Klaus, H., Alex G., Margaret S., Howard B. (2014.). *Monitoring the Execution of Space Craft Flight Software*. Iegūts no https://vdocuments.mx/monitoring-the-execution-of-space-craft-flight-software.html?page=6
5. Harvie C. (2022.). *What Programming Language is Used for Smart Homes?* Iegūts no https://aspiredvision.com/technology/software/what-programming-language-is-used-for-smart-homes
6. Zachary G. (2016.) *The Six Commandments of Good Code: Write Code That Stands the Test of Time.* Iegūts no https://www.toptal.com/software/six-commandments-of-good-code
7. *Programmēšanas paradigma* (2023.). Iegūts no https://lv.wikipedia.org/wiki/Programmēšanas_paradigma
8. *Императивное программирование* (2023.). Iegūts no https://ru.wikipedia.org/w/index.php?title=Императивное_программирование&stable=1
9. *Парадигмы программирования: какие бывают и на что влияют.* (2023.) Iegūts no https://gb.ru/blog/paradigmy-programmirovaniya/
10. *Python Classes and Objects* (n.d.). Iegūts no https://www.w3schools.com/python/python_classes.asp
11. *Java (programmēšanas valoda)* (2023.). Iegūts no https://lv.wikipedia.org/wiki/Java_(programmēšanas_valoda)

Pielikumi

I. Shēma (angļu valodā), kas attēlo izstrādāta koda darbību. Pieejams: https://viewer.diagrams.net/index.html?tags=%7B%7D&highlight=0000ff&edit=_blank&layers=1&nav=1&title=Datu%20apstr%C4%81des%20sist%C4%93ma.drawio#Uhttps%3A%2F%2Fdrive.google.com%2Fuc%3Fid%3D1nRWWTx4Ncx6RTPQqwjCCxhpp7Gxmvy0E%26export%3Ddownload#%7B%22pageId.

II. Visi trīs izstrādātie kodi, kuri tika testēti. Pieejams: https://github.com/sn0wgit/ZPD.