TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Sofia Senkiv 213649IABB
Anastassija Gontšarova 213643IABB

Koodi kvaliteedi hindamine tehisintellekti ja teiste kvaliteedimõõdikute abil

Bakalaureusetöö

Juhendaja: Ants Torim
Doktorikraad

Tallinn 2024

Autorideklaratsioon

Kinnitame, et oleme koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autorid: Anastassija Gontšarova, Sofia Senkiv

20.05.2024

Annotatsioon

Bakalaureusetöö eesmärgiks on hinnata ChatGPT ja mõõdikute hinnangute kvaliteeti koodile ning viia läbi analüüs, millisel määral need hinnangud langevad omavahel kokku. Samuti hinnata, kuivõrd on ChatGPT võimeline koodi kvaliteeti parendama ja halvendama ning kas inimene saab antud ülesandega paremini hakkama. Eksisteerib erinevaid arvamusi tehisintellekti vastuste õigsuse ja usaldusväärsuse kohta, mida antud töö uurib.

Lõputöö tulemusena olid genereeritud nii ChatGPT kui ka autorite poolt parendatud ja halvendatud kvaliteediga koodide versioonid, hinnatud versioonide kvaliteedid klassikaliste mõõdikute ning tehisintellekti abil, mille järel olid saadud tulemused omavahel võrreldud ja analüüsitud. Analüüsi käigus selgus, milline vaadeldavatest lähenemisviisidest koodi kvaliteedi hindamiseks osutus kõige objektiivsemaks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 58 leheküljel, neli peatükki, 27 joonist, 12 tabelit.

Abstract

Code quality evaluation using artificial intelligence and other quality metrics

The aim of the thesis is to assess the quality of the ChatGPT and software metrics code evaluations and to analyse the level of consistency between these evaluations. It also assesses the extent to which ChatGPT is able to improve and degrade the quality of code, and whether a human is able to perform the given task better. Over the last several years, the use of AI, and ChatGPT in particular, has become increasingly common for assessing the quality of code. Hence, there are various opinions on the accuracy and reliability of AI responses, which this work investigates.

To achieve the objectives, authors used a set of classical quality metrics, the AHP tool, the ChatGPT chatbot, clean code rules and their own knowledge. During the working process, different code samples were selected for generating improved and degraded versions by both ChatGPT and the authors. These code versions were then evaluated using quality metrics and ChatGPT, where the results obtained were used to assess consistency.

As a result of the thesis, after the versions of the code with modified quality were generated and the quality of the versions was evaluated, the obtained results were compared to each other and analysed. The analysis revealed which of the approaches considered for assessing code quality proved to be the most objective.

The thesis is in Estonian language and contains 58 pages of text, 4 chapters, 27 figures, 12 tables.

Lühendite ja mõistete sõnastik

AHP	*Analytic hierarchy process*
AHP-OS	*AHP Online System*
AVG	*Average*
BPMSG	*Business Performance Management Singapore*
CC	*Cyclomatic complexity*, tsüklomaatiline keerukus
CNN	*Convolutional Neural Networks*
CR	*Consistency Ratio*, kooskõlalisuse suhtarv
CR_{max}	Maksimaalne kooskõlalisuse suhtarv
GPT	*Generative Pre-Trained Transformer*
GRASP	*General Responsibility Assignment Software Patterns*
LCOM	*Lack of Cohesion in Methods*
MI	*Maintainability Index*, hooldatavuse indeks
MS	*Method similarity*, kahe meetodi vaheline sarnasus
OOP	*Object-oriented programming*
RNN	*Recurrent Neural Network*
Self-attention	Mehhanism masinõppes sõltuvuste ja seoste jäädvustamiseks sisendjärjetes
SLOC	*Source Lines Of Code*

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Kaasaegses programmeerimises on koodi kvaliteedi hindamine eduka ja tõhusa tarkvaraarenduse jaoks väga oluline. On olemas mitmesuguseid koodikvaliteedi mõõdikuid, mis võivad anda väärtuslikku teavet koodi kvaliteedi kohta. Näiteks on olemas Chidamberi ja Kemereri meetrikakomplekt, mis pakub meetrikaid koodi vastavuse hindamiseks objektorienteeritud programmeerimise põhimõtetele (OOP (*Object-oriented programming*)) [1].

Vaatamata erinevate mõõdikute olemasolule on koodi kvaliteedi hindamine endiselt keeruline ja nõuab võimalike lähenemisviiside täiustamist. Teadlased ja arendajad on aja jooksul määratlenud mitmesuguseid koodilõhnatuvastajaid, et disainivigade diagnoosimise protsess oleks arendajatel palju sujuvam. Kuid olenemata sellest esineb ka nende tuvastajate puhul olulisi piiranguid, mis takistavad neid praktikas kasutamast, näiteks tekkivad raskused hea koodi tuvastamise lävede määramisel, kus kvaliteetse koodiosa eristus madala kvaliteediga osast võib mõningatel olukordadel olla üpriski keeruline, arendajate subjektiivne suhtumine taoliste tööriistade abil tuvastatud koodilõhnadesse ning vähene kooskõla erinevate tuvastajate vahel. Üheks viisiks taoliste piirangute vältimiseks pööratakse aina rohkem tähelepanu tehisintellekti kasutamisele [2].

Viimaste aastatega muutus tehisintellekti kasutamine kiirelt arenevas digitaalses maailmas üsna argiseks eriti programmeerimise maailmas ja sellesse süvenemise protsessis. ChatGPT keelemudel, mis oli välja töötatud tehisintellekti uuriva organisatsiooni OpenAI poolt, on saanud paljude algajate ja juba kogenud programmeerijate lahutamatuks ja oluliseks tööriistaks, kui tekib küsimus ühe või teise ülesande või programmeerimiskeele kohta. Üheks tähtsaks sammuks õppeprotsessi puhul on tagasiside saamine ja selle kaudu oma teadmiste täiendamine, seega pöörduvad paljud arendajad tehisintellektuaalse keelemudeli kui kõiketeadva teabeallika poole, mis tänu oma ulatuslikule andmebaasile suudab koodi täiustada või anda sellele hinnangu vastavalt inimese vajadustele. Kuna tehisintellektuaalne keelemudel on iseõppiv, siis mida rohkem

see kooditükke hindab, parandab või täiustab, seda korrektsemalt suudab ta oma tulevased vastused genereerida. Kerkib aga esile küsimus, kas keelemudeli olemasolevatest teadmistest piisab selleks, et seda võiks koodi analüüsi ja parandamise puhul usaldada [3]. Üheks kontrollimise võimaluseks on analüütiliste hierarhiate protsessi (edaspidi AHP (*Analytic hierarchy process*)) rakendamine kooskõlalisuse suhtarvu saamiseks [4].

1.1 Probleem

Kuna tänapäeva maailmas tehisintellekti võimekus järsult kasvab ning selle kasutamine integreerub aina rohkem igapäevaellu, hakkasid inimesed kasutama ChatGPT-d ka koodilõhnade tuvastamiseks, koodi kvaliteedi hindamiseks ja parendamiseks. Vaatamata sellele esineb erinevaid arvamusi, et tehisintellekti vastustesse tasub suhtuda teatud skeptilisusega ning kontrollida tulemusi oma teadmiste ja kogemuste põhjal või vastupidi, et ChatGPT on hea tööriist päringutele vastuse saamiseks, näiteks koodi genereerimise või kvaliteedi hindamise puhul [5], [6], [7]. Tähtis on aru saada, kuivõrd kooskõlalised on tehisintellekti poolt genereeritavad hinnangud koodinäidete kvaliteedile ning kas sama ülesannet lahendavate koodinäidete erinevad versioonid on mõõdikute terminites adekvaatsed. Kooskõlalisus on tagatud, kui A>B, B>C ning kehtib A>C, kus A, B ja C on vastavad objektide väärtused.

Näiteks ei ole IT-teaduskonna õppejõudude tiheda töökoormuse ja üliõpilaste suure arvu tõttu tudengitel võimalik saada alati kiiret, sisukat ja põhjalikku tagasisidet oma koodile sellises mahus, mida õpilane vajab oma programmeerimisoskuste parandamiseks. Samuti võib võtta võõra koodibaasi kvaliteedi hidamine palju aega, mille pärast pöördutakse vestlusroboti poole. Seetõttu on vaja välja selgitada, kas ChatGPT on usaldusväärne viis, et saada hinnangut kirjutatud koodile.

1.2 Eesmärk

Bakalaureusetöö eesmärgiks on hinnata ja analüüsida ChatGPT ja mõõdikute hinnangute kvaliteeti koodile ning uurida, kuivõrd need hinnangud langevad omavahel kokku. Lisaks hinnata, kuivõrd hästi on ChatGPT suuteline koodi parendama ja halvendama ning kas inimene saab selle ülesandega paremini hakkama.

Käesolev töö aitab aru saada, milline vaadeldavatest lähenemisviisidest on objektiivsem ja usaldusväärsem hinnangute andmisel, et parendada programmeerimisoskust ning mõista oma tugevusi ja nõrkusi koodi kirjutamisel.

Eesmärgi saavutamiseks on tööprotsess jagatud järgmisteks alametappideks:

- Koodinäidete andmestiku kogumine avatud lähtekoodist ja programmeerimise raamatust;
- Kogutud koodinäidete põhjal halvendatud ja parendatud koodi versioonide genereerimine;
- Koodi kvaliteedi hindamine ChatGPT ja klassikaliste mõõdikute abil;
- ChatGPT ja klassikaliste mõõdikute hinnangute kooskõlalisuse suhtarvude arvutamine vastavalt AHP-le ning nende võrdlus.

1.3 Töö struktuur

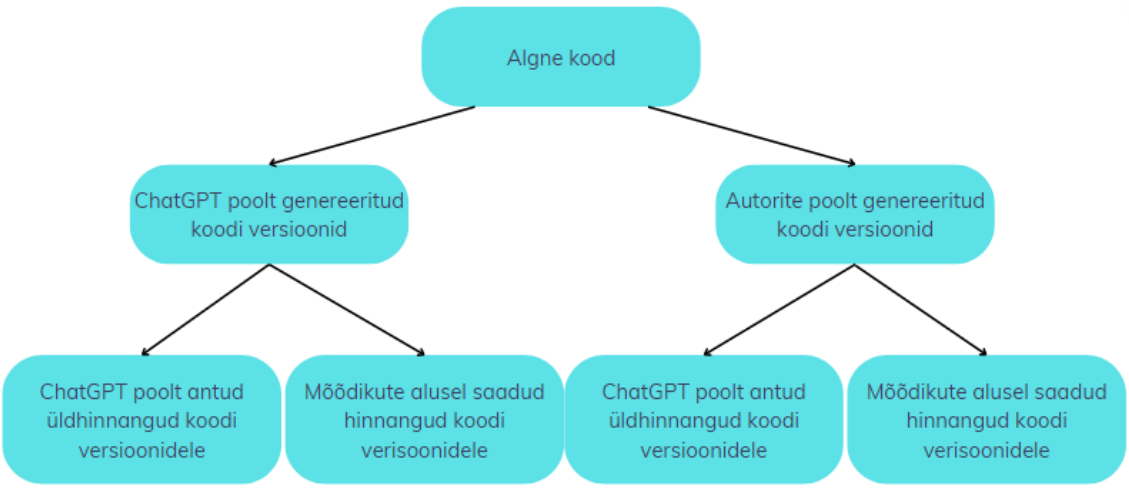
Käesolev töö koosneb neljast peamisest peatükist. Metoodika peatükis kirjeldatakse töö raames uuritud objekti, kasutatud tööriistu ning üldist tööprotsessi. Tulemuste peatükis annavad autorid ülevaate koodi genereerimise lähenemisviisidest, saadud mõõdikute prioritiseerimise, ChatGPT ja klassikaliste mõõdikute hinnangute ja kooskõlalisuse tulemustest ja võrdlevad neid. Analüüsi peatükis hindavad autorid saadud tulemusi, valitud metoodikat ning tööriistu, toovad välja tööriistadega seotud eeliseid ja puudusi, alternatiivseid lähenemisviise ning pakuvad võimalusi töö edasiarenduseks.

2 Metoodika

Käesolev peatükk kirjeldab bakalaureusetöö raames valminud tööd ning uuritud objekti, uurimiseks ja analüüsiks kasutatud tööriistu ning üldist tööprotsessi.

2.1 Objekti detailne kirjeldus

Kogu protsess koodi versioonide genereerimisest kuni kooskõlalisuse suhtarvu arvutamise ja analüüsini oli tehtud neljal viisil, et uurida, millisel neist on parim kooskõlalisus. Esiteks oli koodi versioonide genereerimiseks kasutatud kahte teed – ChatGPT poolt genereeritud parendatud ning halvendatud koodi versioonid ning autorite poolt parendatud ja halvendatud koodi versioonid. Seejärel oli mõlema koodide genereerimise viisi puhul arvutatud iga koodi versiooni jaoks nii mõõdikute komplekti väärtused, et luua neist paarisvõrdluse hinnanguid, kui ka palutud anda paarisvõrdluse üldhinnanguid ChatGPT poolt, et seejärel kasutada neid analüütiliste hierarhiate protsessis kooskõlalisuse suhtarvu saamiseks (Joonis 1).



Joonis 1. Koodi versioonide genereerimise ja hinnangute saamise viisid.

2.2 Tööriistade kirjeldus

Antud peatükis on toodud välja tööriistad, mida autorid kasutavad koodi kvaliteedi hindamiseks, kooskõlalisuse hindamiseks, saadud tulemuste analüüsiks ja erinevate koodi kvaliteedi hindamise lähenemisviiside võrdlemiseks.

2.2.1 Analüütiliste hierarhiate protsess (AHP)

Töö käigus kasutavad autorid BPMSG AHP-OS (*Business Performance Management Singapore AHP Online System*) tööriista eesmärgiga määratleda parimat koodi versiooni analüütiliste hierarhiate protsessi alusel kolmest alternatiivist: parendatud, esialgne ja halvendatud algoritm. Eesmärgiks on koostada kriteeriumide hierarhia, kus kriteeriumideks on mõõdikute komplekt, seejärel arvutada nende prioriteete ning saada paarisvõrdluste hinnanguid otsustamisalternatiivide kogumi kohta nende kriteeriumide alusel. Neid hinnanguid kasutatakse seejärel kooskõlalisuse indeksi arvutamiseks. Antud protsess viiakse läbi nii autorite poolt muudetud koodide kui ka ChatGPT genereeritud koodi versioonide puhul [8].

Iseenesest on AHP ehk analüütiliste hierarhiate protsessi meetod loodud professori Thomas L. Saaty poolt, mis on mõeldud ratsionaalseks mitmekriteeriumiliseks otsustamiseks. Antud meetod põhineb omaväärtuse probleemi lahendamisel, kus paarisvõrdluse tulemused on paigutatud maatriksisse. Maatriksi esimese parempoolse normaliseeritud omavektorist saab suhte skaala ning omaväärtus määrab omakorda kooskõlalisuse suhte [4].

Valitud AHP-OS tööriistas Thomas L. Saaty poolt pakutud kooskõlalisuse suhtarve valemi asemel

$$CR = \frac{\lambda - n}{(n-1)RI_n}$$

oli kasutatud Alonso ja Lamata poolt pakutud kooskõlalisuse suhtarve valemit [9]:

$$CR = \frac{\lambda - n}{2{,}7699 \cdot n - 4{,}3513 - n}$$

, kus

- RI_n– juhuslik kooskõlalisuse indeks,
- λ – domineeriv omaväärtus,
- n – kriteeriumide arv.

AHP paneb otsustamisprobleemi hierarhia struktureerimise käigus kohe probleemi läbi mõtlema, arvestades võimalikke otsustuskriteeriume ning nägema loogilisi vastuolusid paarisvõrdluse käigus [8].

2.2.2 ChatGPT

Kogutud koodinäidete andmestiku põhjal genereerivad autorid parendatud ja halvendatud koodi versioone ChatGPT abil. ChatGPT hinnanguid koodi versioonide kvaliteedile võrreldatakse klassikaliste mõõdikute hinnangutega ning otsustatakse, kumb hinnangutest on paremas kooskõlas inimgenereeritud ja ChatGPT poolt genereeritud koodi puhul.

ChatGPT on tehisintellektiga vestlusrobot, mis suudab anda detailse vastuse inimpäringutele, mis on esitatud loomulikus keeles. ChatGPT on näidanud võimsaid funktsioone erinevate keelte mõistmise ja genereerimise ülesannete puhul, ning suudab mäletada varasemat dialoogi kasutajaga [10].

ChatGPT põhineb algsel GPT (*Generative Pre-Trained Transformer*) mudelil, mis omakorda põhineb Transformer arhitektuuril [11]. Transformer on närvivõrgu arhitektuur järjestatud jadaandmete töötlemiseks. Nimetatud mudelit kasutatakse enamasti loomuliku keele töötlemiseks ja mõistmiseks [12]. Transformer ei kasuta järjestusega joondatud RNN-i (*Recurrent Neural Network*) või konvolutsiooni, vaid oma sisendi ja väljundi arvutamiseks tugineb täielikult *self-attention*'le [11].

Antud töös kasutasid autorid 3.5 ChatGPT versiooni [13].

2.2.3 Mõõdikud

Koodi kvaliteedi hindamiseks kasutatakse erinevaid tarkvaramõõdikuid, mis aitavad mõista koodi omadusi ning tuvastada koodi lõhnad. Uuringute kohaselt kasutatakse kõige sagedamini keerukuse ja sidumise mõõdikuid, millele järgnevad kohesiooni ja suuruse mõõdikud [14].

Koodi kvaliteedi hindamiseks ning ChatGPT hinnangutega võrdlemiseks kasutavad autorid järgmiseid mõõdikuid:

- ABC mõõdik – mõõdik, mida kasutatakse tarkvara suuruse arvutamiseks. Erinevaid probleeme lahendavate koodide ABC mõõdiku väärtusi ei saa omavahel võrrelda. Võrreldes aga sama probleemi lahendavaid koode kehtib reegel, et mida väiksem on ABC mõõdiku väärtus, seda parem. ABC vektoril on kolm komponenti:
    - *Assignment* – andmete ülekandmine muutujasse;
    - *Branch* – programmi hargnemine väljapoole kohaldamisala;
    - *Condition* – loogiline/boolean test.

  Iga komponent tähistab põhiliste operatsioonide arvu – andmete salvestamine toimub muutujate määramise kujul (A-täht ABC-s), hargnemine toimub funktsiooni kutsete kujul (B-täht ABC-s) ning tingimused on sätestatud *if, else* ja sarnaste konstruktsioonidega (C-täht ABC-s). ABC vektor kirjutatakse täisarvude järjustatud kolmikuna ning ABC suuruse võib leida, kasutades vektori skalaarse suuruse valemit [15]:

$$|ABC| = \sqrt{A^2 + B^2 + C^2}$$

- Tsüklomaatiline keerukus (*CC (Cyclomatic complexity)*) – mõõdik, mida kasutatakse programmi loogilise keerukuse arvutamiseks. Seda mõõdikut võib kasutada ka programmi funktsioonide, moodulite või klasside keerukuse hindamiseks. Tsüklomaatilise keerukuse arvutatakse valemiga:

  $v(G) = e - n + 2p$

  , kus
    - *e* on servade arv kontrollvoogude graafis,
    - *n* on sõlmede arv kontrollvoogude graafis,
    - *p* on ühendatud komponentide arv [16].

- Meetodite kohesiooni puudumise (LCOM4) mõõdik – kohesiooni mõõdik, mis on oluliseks teguriks objektorienteeritud tarkvara projektide puhul ning näitab, kui palju ühendatud komponente ehk seotud meetodite ja klassitasemel muutujate kogumeid on ühes klassis.

  LCOM4 väärtuste arvutamisel kasutatakse suunamata graafi, kus sõlmedeks on meetodid ja serv tähistab nendevahelist ühendust. Lisaks arvestab LCOM4 mitte ainult meetodi ja atribuudi vahelise suhtega, vaid ka meetod-meetod koostoimega ning juhul kui meetodid kutsuvad üksteist, tõmmatakse nende vahele serv.

  Ühendamata komponentide arv annab antud mõõdiku väärtuse. LCOM4 väärtus 1 näitab, et klass on sidus ning kõik klassi atribuudid on meetoditega seotud. Väärtus 0 tähendab, et klass on meetoditest tühi, mida peetakse samuti halvaks, ning väärtus 2 või rohkem viitab sellele, et klassi on tarvis jagada mitmeks väiksemaks klassiks [17].

  Autorid valisid LCOM4 viie LCOM (*Lack of Cohesion in Methods*) mõõdiku seast, kuna see on ainuke mõõdik, mis arvestab väärtuste arvutamisel meetodite väljakutsumisega klassi sees. Antud tingimust on tähtis arvesse võtta, kuna just meetodite väljakutsumised klassi sees on hea indikaator, et kõik meetodid toimivad sama eesmärgi nimel [18].

- Hooldatavuse indeks (MI (*Maintainability Index*)) – mõõdik, mis näitab, kuivõrd kergesti toetav ja muudetav on lähtekood. Hooldatavuse indeksi arvutamiseks autorid kasutasid Pythoni tööriista Radon. Radon kasutab järgmist valemit MI arvutamiseks:

$$MI = max\left[0{,}100\,\frac{171 - 5.2\ln V - 0.23G - 16.2\ln L + 50\sin(\sqrt{2.4C})}{171}\right]$$

  , kus

  - V on Halsteadi maht,
  - G on kogu tsüklomaatiline keerukus,
  - L on lähtekoodi ridade arv (SLOC (*Source Lines Of Code*)),

- C on kommentaari ridade protsent (teisendatud radiaanideks) [19].

- Halstead mõõdikud – mõõdikud, mis kuuluvad Halstead´i tarkvarateooria alla tarkvaraprogrammide keerukuse hindamiseks operandide ja operaatorite arvu põhjal.

  - Programmi sõnavara – ainulaadsete operaatorite ja operandide esinemise arv;

$$n = n_1 + n_2$$

  - Programmi pikkus – operaatorite ja operandide esinemiste koguarv;

$$N = N_1 + N_2$$

  - Programmi arvutatud pikkus – seos programmi pikkuse ja sõnavara vahel;

$$N = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

  - Programmi maht (*V*) – näitab, kui palju ruumi bittides on tarvis programmi salvestamiseks;

$$V = N \log_2 n$$

  - Programmi raskustase (D) – näitab suhet programmi unikaalsete operaatorite ja operaatorite koguarvu vahel;

$$D = \left(\frac{n_1}{2}\right) \times \left(\frac{N_2}{n_2}\right)$$

  - Vaimne pingutus (*E*) – näitab nõutavat vaimset pingutust ja aega programmist arusaamiseks või selle loomiseks ning on leitav programmi raskustaseme ja mahu taseme korrutisega;

$$E = D \cdot V$$

  - Aeg (*T*) – näitab hinnangulist aega programmi implementatsiooniks;

$$T = \frac{E}{S}$$

- Vead (*B*) – näitab eeldatavat vigade esinemise arvu programmis;

$$B = \frac{V}{3000}$$

, kus

n_1 = erinevate operaatorite arv programmis,

n_2 = erinevate operandide arv programmis,

N_1 = operaatorite summaarne esinemiste arv programmis,

N_2 = operandide summaarne esinemiste arv programmis,

S = 18 – Stround number, konstant, mis sõltub kindlast programmeerimiskeelest [18], [20], [21], [22].

2.3 Tööprotsessi kirjeldus

Antud peatükk käsitleb bakalaureusetöö raames valitud koodi kvaliteedi hindamise metoodikat, loodud teisendusreeglit koodi versioonide paarilise võrdluse jaoks ning hinnangute kooskõlalisuse hindamise metoodikat.

2.3.1 Koodi genereerimine

Tööprotsessi alguses kogusid autorid koodinäidete andmestiku avatud lähtekoodist [23], [24] ning John Zelle „Python Programming“ raamatu põhjal kirjutatud lähtekoodist [25]. Kokku tuli 14 koodinäidet. Kasutades kogutud koodinäiteid, saatsid autorid ChatGPT vestlusrobotile päringu, et ta prooviks tagastada ühe parendatud ja ühe halvendatud versiooni koodist. Selle jaoks olid eelnevalt kogutud kokku puhta koodi reeglid Robert Martini „Clean Code: A Handbook of Agile Software Craftsmanship“ raamatust [26], mis on toodud välja analüüsi peatükis. Neid puhta koodi reegleid edastasid autorid tehisintellektile, et ta genereeriks igale algsele koodinäitele kaks lisaversiooni. Aeg-ajalt andis ChatGPT mittetöötava koodi versiooni, sellisel juhul päringu tegemine kordus seni, kuni oli genereeritud töötav versioon.

Lisaks ChatGPT abil genereeritud koodinäidete versioonidele, parendasid ja halvendasid autorid ise kuut algset koodinäidet vastavalt parimatele programmeerimise tavadele, puhta koodi reeglitele ning oma teadmistele. Kuna kohesiooni mõõdik osutus kõige olulisemaks ning selle kaal oli suurim mõõdikute komplektist, valisid autorid neid esialgseid koode, kus vastava mõõdiku väärtus erines ühest. Kõik esialgsed koodid, tehisintellekti kui ka autorite poolt parendatud ja halvendatud koodide versioonid on kättesaadavad avatud lähtekoodis GitHub'is [27]. Kõik kood oli kirjutatud Python programmeerimiskeeles.

2.3.2 Koodi kvaliteedi hindamise metoodika

Paremaks teadusuuringuks oli koodi kvaliteedi hindamine viidud läbi kahel viisil. Esimene viis põhineb koodi hindamisel mõõdikute väärtuste alusel. Selleks kasutasid autorid iga koodi versiooni jaoks *radon* [28], *python_abc* [29] ja *LCOM* [30] mooduleid, et saada vastavalt tsüklomaatiline keerukus, hooldatavuse indeks, Halstead (kus kõik kuuluvad radon moodulisse), ABC ja kohesiooni mõõdikute väärtuseid, mis seejärel olid kantud iga koodi tabelite sisse. Kokku hindasid autorid mõõdikute alusel 54 koodi versiooni, millest 14 oli algne kood, 28 koodi versiooni oli ChatGPT poolt genereeritud ning 12 koodi versiooni olid autorite poolt genereeritud koodid. Kokku oli saadud 20 tabelit.

Näidet sellest, kuidas olid struktureeritud tabelid mõõdikute väärtustega iga koodi jaoks koos kolme koodi versiooniga, saab vaadata lisas 2.

Teine viis koodi kvaliteedi hindamiseks oli üldhinnangute saamine ChatGPT poolt paarisvõrdluse põhjal. Selleks saatsid autorid ChatGPT vestlusrobotile päringu, kus palusid anda iga koodi puhul koodi versioonide kaupa paarilisi hinnanguid skaalas 1–9 vastavalt AHP protsessile. Hinnanguid oli palutud anda vastavalt efektiivsuse, optimaalsuse, jõudluse ning üldise algoritmi loetavuse põhjal. Väärtus 1 paarilises võrdluses tähendab, et paaris olevad koodi versioonid on kriteeriumide põhjal võrdsed, ning 2–9 näitab, mitu korda on üks koodi versioon teisest parem ja prioriteetsem. Lisaks oli palutud lisada ka põhjendus, miks üks koodi versioon on teisest parem või halvem.

Tabelid koos ChatGPT poolt antud üldiste hinnangutega ja põhjendustega on struktureeritud järgmiselt:

Tabel 1. Näidistabel ChatGPT poolt antud paarisvõrdluste hinnangute ja põhjendustega.

Koodi versioon	**Paarisvõrdluse hinnang**	**Koodi versioon**	**Paarisvõrdluse hinnang**	**Põhjendus**
INITIAL	1	*GOOD*	4	<Põhjendus>
INITIAL	1	*BAD*	3	<Põhjendus>
GOOD	1	*BAD*	2	<Põhjendus>

Näide ChatGPT põhjendusest esialgse ja parendatud versioonidele: “Hea algoritmi versioon pakub olulisi täiustusi võrreldes esialgsega. Hea versioon on oluliselt loetavam ja tõhusam. Hea versioon on märkimisväärselt parem kui esialgne. See on oluliselt loetavam, tänu selgemale ja kompaktsemale koodile ning pakub mitmeid täiustusi, sealhulgas efektiivsemat andmete manipuleerimist.“

Kokku on autorid saanud 20 tabelit, millest 6 on inimgenereeritud koodiga ning 14 on ChatGPT poolt genereeritud koodiga.

Mõlemad viisid hinnangute saamiseks koodi kvaliteedile olid viidud läbi nii ChatGPT poolt genereeritud koodi versioonide kui ka autorite poolt parendatud ja halvendatud koodi versioonide puhul.

2.3.3 Hinnangud paarisvõrdluse põhjal

Järgmise AHP etapi jaoks oli tarvis arvutada hinnanguid koodi versioonide paarisvõrdluse põhjal skaalas 1–9. ChatGPT poolt antud paarisvõrdluse üldhinnangud olid juba saadud, seega jäi neid saada ka mõõdikute väärtuste alusel. Selleks, et saada kooskõla suhtarvu, tuleb vastavalt AHP protsessile teha paarilisi hindamisi iga mõõdiku kohta. Kuna ühe koodi erinevate versioonide puhul võivad konkreetse mõõdiku väärtused olla väiksema või suurema erinevusega, aga paarilisi hinnanguid soovitakse skaalas 1–9, on autorid loonud järgmise teisendusreegli, mis on vajalik selleks, et mõõdikute väärtuseid teisendada hinnanguteks:

$$Teisendusreegel = \left(\frac{\max V\ \text{äär}tus}{\min V\ \text{äär}tus}\right) \cdot C$$

, kus

- *maxVäärtus* – on koodi versioonide paarisvõrdluse suurim väärtus;

- *minVäärtus* – on koodi versioonide paarisvõrdluse väikseim väärtus;
- *C* – valitud koefitsient.

Tsüklomaatilise keerukuse, hooldatavuse indeksi, ABC ja Halstead mõõdikute puhul on koefitsiendi väärtuseks 1,3, kohesiooni puhul aga 3. Kuna mõõdikute väärtused või koefitsient võisid olla kümnendmurru kujul, olid teisendusreegli põhjal saadud väärtused ümardatud täisarvuni, et saavutada täpsemaid paremuse hinnanguid vastavalt skaalale 1–9. Kõik teisendusreegli abil sooritatud arvutused ja lõplikud mõõdikute hinnangud on avalikult kättesaadavad *Google Sheets* töölehtedel [31].

2.3.4 Mõõdikute prioritiseerimine

Selleks, et saada AHP tööriistas koodi versioonide kooskõlalisuse suhtarvu, on vajalik määrata viie mõõdiku prioriteetsust. Kuna autorid ei leidnud teadusartikleid, kus oleks käsitletud vajalikku mõõdikute komplekti ning analüüsitud mõõdikute tähtsuse osakaalu koodis, oli tarvis pöörduda ChatGPT poole ning lasta tal anda prioriteetide hinnanguid iga mõõdiku paarile skaalas 1–9 .

Hinnangud ChatGPT poolt igale mõõdikutepaarile on struktureeritud järgmiselt:

Tabel 2. Näidistabel paarisvõrdluse hinnangutega ChatGPT poolt.

Mõõdik 1	**Hinnang**	**Mõõdik 2**	**Hinnang**	**Põhjendus**
Cyclomatic complexity	1	*Cohesion*	5	<Põhjendus>

Näide ChatGPT põhjendusest tsüklomaatilise keerukuse ja kohesiooni mõõdikute võrdlusele: “Üldiselt on cohesion olulisem, kuna see aitab tagada algoritmi loogilist struktuuri ja seostatust, mis on oluline mitmete algoritmide, sealhulgas graafi- ja järjendipõhiste algoritmide puhul. *Cyclomatic Complexity*, kuigi oluline, ei ole alati nii kriitiline kui loogiline sidusus.”

Kokku said autorid 10 mõõdikutepaari võrdluseks.

2.3.5 Kooskõlalisuse hindamine

Viimase sammuna tuli arvutada AHP tööriistas ChatGPT ja klassikaliste mõõdikute hinnangute kooskõlalisust. Selle jaoks oli vaja esiteks luua mõõdikute hierarhiat, kasutades ChatGPT poolt genereeritud mõõdikute prioritiseerimist. Hierarhia loomiseks sisestasid autorid mõõdikute prioriteedid paari kaupa (Joonis 2, Joonis 3).

	A - wrt AHP priorities - or B?		Equal	How much more?
1	○ Cyclomatic Complexity	◉ Maintainability index	○ 1	○ 2 ◉ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9
2	◉ Cyclomatic Complexity	○ ABC	○ 1	◉ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9
3	○ Cyclomatic Complexity	◉ Cohesion	○ 1	○ 2 ○ 3 ○ 4 ◉ 5 ○ 6 ○ 7 ○ 8 ○ 9
4	○ Cyclomatic Complexity	◉ Halstead	○ 1	○ 2 ◉ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9
5	◉ Maintainability index	○ ABC	○ 1	◉ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9
6	○ Maintainability index	◉ Cohesion	○ 1	○ 2 ○ 3 ◉ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9
7	◉ Maintainability index	○ Halstead	○ 1	○ 2 ○ 3 ◉ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9
8	○ ABC	◉ Cohesion	○ 1	○ 2 ○ 3 ○ 4 ○ 5 ◉ 6 ○ 7 ○ 8 ○ 9
9	○ ABC	◉ Halstead	○ 1	○ 2 ◉ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9
10	◉ Cohesion	○ Halstead	○ 1	○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ◉ 7 ○ 8 ○ 9

CR = 11.3% Adjust highlighted judgments to improve consistency

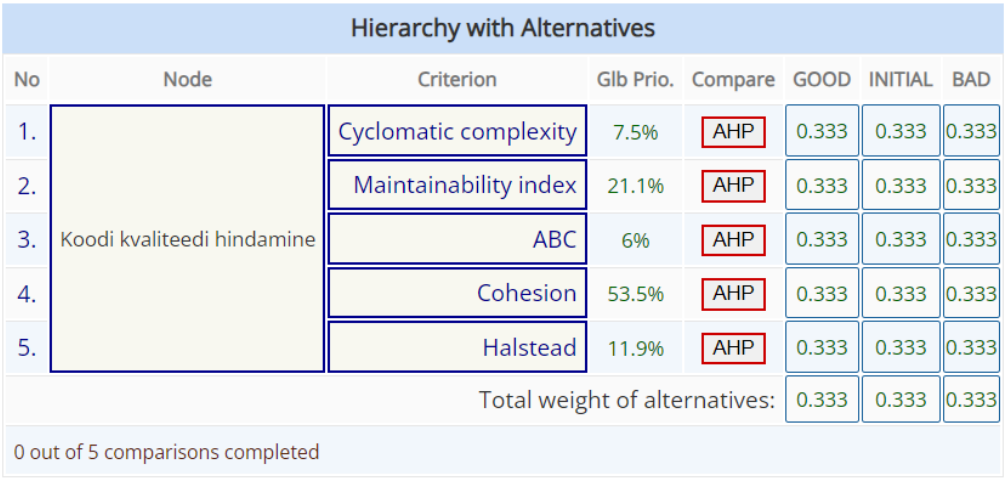
Joonis 2. Mõõdikute paariline võrdlus hierarhia loomiseks.

Decision Hierarchy

Level 0	Level 1	Glb Prio.
Koodi kvaliteedi hindamine AHP	Cyclomatic complexity 0.075	7.5%
	Maintainability index 0.211	21.1%
	ABC 0.060	6.0%
	Cohesion 0.535	53.5%
	Halstead 0.119	11.9%
		1.0

Joonis 3. Mõõdikute hierarhia AHP tööriistas.

Hierarhiale defineerisid autorid alternatiive „*GOOD*“, „*BAD*“ ja „*INITIAL*“, mis vastavalt tähendavad parendatud, halvendatud ja esialgse koodi versiooni (Joonis 4).



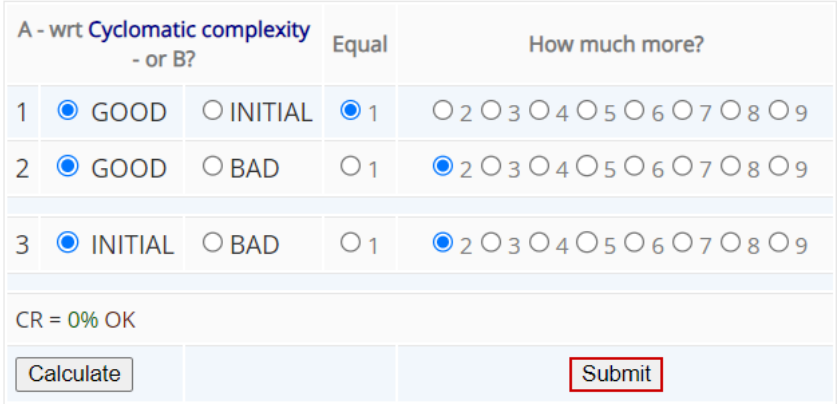
Hierarchy with Alternatives

No	Node	Criterion	Glb Prio.	Compare	GOOD	INITIAL	BAD
1.	Koodi kvaliteedi hindamine	Cyclomatic complexity	7.5%	AHP	0.333	0.333	0.333
2.		Maintainability index	21.1%	AHP	0.333	0.333	0.333
3.		ABC	6%	AHP	0.333	0.333	0.333
4.		Cohesion	53.5%	AHP	0.333	0.333	0.333
5.		Halstead	11.9%	AHP	0.333	0.333	0.333
				Total weight of alternatives:	0.333	0.333	0.333

0 out of 5 comparisons completed

Joonis 4. Mõõdikute hierarhia koos alternatiividega.

Pärast alternatiivide defineerimist automaatselt olid loodud prioriteedid iga kriteeriumi iga alternatiivi jaoks. Neid prioriteete tuli muuta, vajutades nuppu „AHP“ ja sisestades eelnevalt arvutatud või ChatGPT poolt genereeritud hinnanguid versioonidele paari kaupa (Joonis 5).



A - wrt Cyclomatic complexity - or B?			Equal	How much more?
1	◉ GOOD	○ INITIAL	◉ 1	○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9
2	◉ GOOD	○ BAD	○ 1	◉ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9
3	◉ INITIAL	○ BAD	○ 1	◉ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9

CR = 0% OK

Calculate Submit

Joonis 5. Alternatiivide paarilised võrdlused ja kooskõla suhtarvu arvutamine.

Seejärel, vajutades nuppu „*Calculate*“, arvutati hinnangute kooskõla suhtarvu ning “*Submit*” nuppu vajutamisel salvestati uued prioriteedid kriteeriumi alternatiividele (Joonis 6).

Hierarchy with Alternatives

No	Node	Criterion	Glb Prio.	Compare	GOOD	INITIAL	BAD
1.	Koodi kvaliteedi hindamine	Cyclomatic complexity	7.5%	AHP	0.400	0.400	0.200
2.		Maintainability index	21.1%	AHP	0.333	0.333	0.333
3.		ABC	6%	AHP	0.333	0.333	0.333
4.		Cohesion	53.5%	AHP	0.333	0.333	0.333
5.		Halstead	11.9%	AHP	0.333	0.333	0.333
				Total weight of alternatives:	0.338	0.338	0.323

1 out of 5 comparisons completed

Joonis 6. Uued prioriteedid *Cyclomatic Complexity* kriteeriumi alternatiivide jaoks.

Uued prioriteedid olid arvutatud iga kriteeriumi jaoks, et saaks hinnata kriteeriumi hinnangute kooskõlalisust ning lõpuks näha, milline versioon kolmest alternatiivist on parim (Joonis 7). Kõik AHP tööriista abil saadud tulemused on avalikult kättesaadavad *Google Sheets* töölehtedel [31].

Cat		Priority	Rank
1	GOOD	60.2%	1
2	INITIAL	25.6%	2
3	BAD	14.3%	3

Joonis 7. Alternatiivide prioriteedid.

3 Tulemused

Käesolevas peatükis annavad autorid ülevaate koodi genereerimise lähenemisviisidest, saadud mõõdikute prioritiseerimise, ChatGPT ja klassikaliste mõõdikute hinnangute ja kooskõlalisuse tulemustest ja võrdlevad neid.

3.1 Koodi genereerimine

Peale ChatGPT poolt genereeritud koodide versioonide kooskõlalisuse hindamist soovisid autorid hinnata ka inimgenereeritud koodide versioonide kooskõlalisust, mis nõudis algse koodi läbitöötamist, analüüsimist ning lõpuks parendatud ja halvendatud versioonide loomist.

Koodi kvaliteedi parendamiseks kasutasid autorid järgmiseid lähenemisviise:

- Meetodide jaotamine alammeetoditeks, et igal meetodil oleks oma kindel eesmärk ja vastutus;

```
def playGame(self):
    # Play the game to completion
    y = self.playerA.getProb()
    while not self.isOverGame():
        x = random()
        if x < y:
            self.playerA.incScore()
        else:
            self.playerB.incScore()
```

Joonis 8. Esialgse versiooni INIMGEN_zelleEx2 näide.

```
def play_game(self):
    # Play the game to completion
    y = self.player_a.get_probability()
    while not self.is_over_game():
        self.play_point()

def play_point(self):
    # Simulate a single point in the game
    if self.current_server.wins_serve():
        self.current_server.inc_score()
    else:
        self.change_server()
```

Joonis 9. Parendatud versiooni INIMGEN_zelleEx2 näide.

- Muutujate ja meetodite nimede arusaadavamaks muutmine;

```
def screen1(self):
    userid = None
    pin = None
    while not self.verify(userid, pin):
        userid, pin = self.interface.getUserInput()
    self.user = self.getUser(userid)

def screen2(self):
    terminate = self.interface.close()
    while not terminate:
        checking, savings = self.user.checkBalances()
        tran_type = self.interface.chooseTransaction()
        self.__updateAccounts()
        if tran_type[0] == "C":
              self.interface.displayBalance(checking,
              savings)
        elif tran_type[0] == "W":
              value, account_type =
              self.interface.withdrawCash()
              self.user.exchangeCash(value,
              account_type)
        elif tran_type[0] == "T":
              inaccount, outaccount, value =
              self.interface.getTransferInfo()
              self.user.transferMoney(inaccount,
              outaccount, value)
        elif tran_type[0] == "S":
              recipient, type_sender, type_recipient,
              value = self.interface.getSendInfo()
              recipient = self.getUser(recipient)
              self.sendMoney(self.user, recipient,
              value, type_sender, type_recipient)
        elif tran_type[0] == "Q":
              self.user.exchangeCash(-100, "Checking")
        elif tran_type[0] == "E":
             self.__updateAccounts()
             terminate = True
```

Joonis 10. Näide esialgse INIMGEN_zelleAtmManager versiooni meetodi nimedest.

```
def getATMUser(self):
    while True:
        userid, pin = getUserInput()
        if self.verifyUser(userid, pin):
            self.user = self.getUser(userid)
            break
            print("Credentials are invalid.
            Please try again.")

def chooseTransaction(self):
    terminate = self.interface.close()
    while not terminate:
         checking, savings = self.user.checkBalances()
        transaction_type =
        self.interface.chooseTransaction()
         self.__updateAccounts()
         if transaction_type[0] == "C":
            displayBalance(checking, savings)
         elif transaction_type[0] == "W":
            value, account_type = withdrawCash()
            self.user.exchangeCash(
            value, account_type)
         elif transaction_type[0] == "T":
            in_account, value = getTransferInfo()
            self.user.transferMoney(in_account, value)
         elif transaction_type[0] == "S":
           recipient, type_sender, type_recipient,
           value = getSendInfo()
            recipient = self.getUser(recipient)
           sendMoney(self.user, recipient,
           value, type_sender, type_recipient)
        elif transaction_type[0] == "Q":
            self.user.exchangeCash(-100, "Checking")
        elif transaction_type[0] == "E":
            self.__updateAccounts()
            terminate = True
```

Joonis 11. Näide parendatud INIMGEN_ zelleAtmManager versiooni nimedest, kus meetodi eesmärk on arusaadavam.

- Kommentaaride lisamine;

```
class WordTruer:
    def __init__(self, dictionary, string):
        self.dictionary = dictionary
        self.string = string
        self.anagrams = self.permute(self.string)
        self.twords = []
        self.dwords = []
        self.read_dict()

        self.check_anagrams()
        self.summary()
```

Joonis 12. Esialgse versiooni INIMGEN_zelleChap13Ex11 näide.

```
class WordTruer:
    def __init__(self, dictionary, string):
        # Initialization of the WordTruer class object
        self.dictionary = dictionary  # Path to
        # the dictionary file
        self.string = string  # String to find in dictionary
        # Generate anagrams from the input string
        self.anagrams = {}
        self.found_words = []  # List to store words
        # found in the dictionary
        self.dictionary_words = []  # List to store
        # words from the dictionary file
```

Joonis 13. Parendatud versiooni INIMGEN_ zelleChap13Ex11 näide.

- Klassi atribuute mitte kasutavate meetodite välja tõstmine klassist;

```
class TextInterface:
    def __init__(self):
        self.active = True
        print("Welcome to the ATM")

    def getUserInput(self):
        # get UserID and Pin
        a = input("Username >> ")
        b = input("Pin >> ")
        return a, b

    def displayBalance(self, checking, savings):
         # display savings and checking
        print("Checking: {0}    Savings: {1}".format(
        checking,  savings))

    def withdrawCash(self):
       print("This function will withdraw
       or deposit cash.")
       value = eval(
       input("How much should we deposit/withdraw?
       Enter negative value to withdraw. >> "))
       account = input("Checking or Savings? >> ")
       print("Your transaction is complete.\n")
       return value, account
```

Joonis 14. Esialgse INIMGEN_zelleAtmManager versiooni näide, kus *TextInterface* klassis asuvad meetodid, mis ei kasuta klassi atribuute.

```
import json

# valid account types written to avoid duplication
valid_account_types = {"Checking", "Savings"}


# get UserID and Pin
def getUserInput():
    username = input("Username >> ")
    pin = input("Pin >> ")
    return username, pin


# display savings and checking
def displayBalance(checking, savings):
    print(f"Checking: {checking}    Savings: {savings}")


def withdrawCash():
    """Withdraw or deposit cash."""
    value = get_valid_float(
    "How much should we deposit/withdraw?
    Enter negative value to withdraw: ")
    account = get_account_type("Checking or Savings? ")
    print("Your transaction is complete.\n")
    return value, account
```

Joonis 15. Parendatud INIMGEN_zelleAtmManager versiooni näide, kus meetoditest on tehtud funktsioonid.

- *Try-catch* plokkide lisamine;

```
@staticmethod
def read_float(prefix, default = None):
      """ Read a (casted) float from console, with a prefix """
      return ConsoleReader.__cast(
                  ConsoleReader.read_string(prefix),
                  float) or default
```

Joonis 16. Esialgse versiooni INIMGEN_zelleEx2 meetod konsoolist lugemiseks.

```
@staticmethod
def read_float(prompt):
    """ Read a float from console with a given prompt """
    while True:
        try:
            return float(ConsoleReader.read_string(prompt))
        except ValueError:
            print("Please enter a valid float value.")
```

Joonis 17. Parendatud versioon INIMGEN_zelleEx2 meetodist, kus kasutatakse try-catch plokk.

Koodi kvaliteedi halvendamiseks olid kasutatud aga järgmised lähenemisviisid:

- Meetodite kombineerimine *Single Responsibility Principle* [32] rikkumiseks;

```
class SkunkApp:
    def __init__(self, interface):
        self.dice = Dice()
        self.interface = interface
        self.players = []
        self.makeplayers()

    def run(self):
        while not self.gameOver():
            self.interface.newRound()
            self.refreshPlayers()
            print("Round: {}".format(self.interface.getRound() +
            1))
            print("\n")
            self.playRound()
        self.interface.gameSummary(self.players)

    def playRound(self):
        books = self.interface.getRound()
        while not self.roundOver():
            for player in self.players:
                if player.active():
                    values = self.doRoll(player)
                    if values != [0, 0]:
                        x, y = values[0], values[1]
                        if x == 1 and y == 1:
                            player.resetRoundScores(books)
                        elif x ==1 or y == 1:
                            player.reset_rScore()
                        else:
                            player.inc_Score(x + y, books)
                            self.interface.currentscores(
                            values, self.players)
        for player in self.players:
            player.tallyRound(books)

    def refreshPlayers(self):
        for player in self.players:
            player.reactivate()
            player.rScoreOnly()

class TextInterface:
    def __init__(self):
        self.round = -1
    def gameSummary(self, players):
        self.finalScores(players)
```

Joonis 18. Esialgse INIMGEN_zelleEx5 versiooni näide.

```
class SkunkApp:
    def __init__(self, interface):
        self.dice = Dice()
        self.interface = interface
        self.players = []
        self.make_players()

    def runGame(self):
        while not self.GameOver():
            self.interface.newRound()
            for player in self.players:
                player.deactivate()
                player.roundScoreOnly()
            print(f"Round: {self.interface.getRound() + 1}")
            print("\n")

            books = self.interface.getRound()
            while not self.Round_Over():
                for player in self.players:
                    if not player.isActive() == False:
                        values = self.DoRoll(player)
                        if values != [0, 0]:
                            x = values[0]
                            y = values[1]
                            player.countScoreOrReset(
                            x, y, books)
                            for player in self.players:
                                 name = player.getName()
                                 roundScore =
                                 player.get_rScore()
                                 total_score =
                                 player.getTotalScore()
                                 print(f"Player: {name}
                                 Score: {roundScore}
                                 Total: {total_score}")
                            print(f"You rolled {values}")
                            print("\n")

            for player in self.players:
                player.tallyRound(books)

        print("The game is over")
        self.interface.finalScores(self.players)
```

Joonis 19. Halvendatud INIMGEN_zelleEx5 näide, kus *Single Responsibility Principle* on rikutud.

- Klassile sobivate meetodite lisamine, mida ei kasutata;

```
def winsServe(self):
    # Returns a Boolean that is true
    # with probability self.prob
    return random() <= self.prob
```

Joonis 20. Esialgse versiooni INIMGEN_zelleObjrball näide, kus antud meetod on kasutusel.

```
def winsServe(self):
    if random() < self.getProb():
        wins = True
    else:
        wins = False
    return wins
```

Joonis 21. Halvendatud versiooni INIMGEN_zelleObjrball näide, kus meetod eksisteerib, aga pole kasutusel.

- Lihtsate probleemide jaoks keeruliste lahenduste kasutamine, näiteks keerukamate *if-else* plokkide või *for*-tsüklite kasutamine;

```
def isOver(self):
    # Returns game is finished (i.e. one of the
    # players has won).
    a, b = self.getScores()
    return a == 15 or b == 15 or \
        (a == 7 and b == 0) or (b == 7 and a == 0)
```

Joonis 22. Esialgse versiooni INIMGEN_zelleObjrball näide.

```
def isOver(self):
    a, b = self.getScores()
    if a == 15 or b == 15:
        return True
    if a == 7 and b == 0:
        return True
    if b == 7 and a == 0:
        return True
    return False
```

Joonis 23. Halvendatud versiooni INIMGEN_zelleObjrball näide.

Kõiki eespool loetletud reegleid ei kasutanud autorid iga kord uue koodi versiooni loomisel, vaid lähtuvalt olukorrast ning esialgse koodi eripäradest valisid nad kas mõned parendamise ja halvendamise reeglid või kõik olemasolevad reeglid.

3.2 Mõõdikute prioritiseerimine ChatGPT poolt

Selleks, et koodi versioonide kvaliteeti hinnata oli oluline seada mõõdikute prioriteetsuse järjekorda, et määrata olulisemaid kriteeriume koodi kvaliteedi puhul. ChatGPT mõõdikute paarisvõrdluse hinnangutest on näha, et tähtsaimaks mõõdikuks pidas vestlusrobot kohesiooni mõõdiku 53,5% prioriteediga teiste mõõdikute suhtes, kus antud osakaal moodustab üle poole kõigist mõõdikute prioriteetide osast, omades suurt tähtsust koodi kvaliteedi puhul. Teisest kuni neljanda kohani asuvad mõõdikud hooldatavuse indeks, Halstead ja tsüklomaatiline keerukus tähtsuse osakaaluga vastavalt 21,2%, 11,9% ning 7,5%. Viimast kohta saavutab aga ABC koodi pikkuse mõõdik prioriteediga 6% teiste mõõdikute suhtes.

Uurides mõõdikute paarisvõrdluse hinnanguid selgub, et kohesioon oli 4 kuni 7 korda olulisem kui ülejäänud neli mõõdikut AHP 1–9 skaalas. Hooldatavuse indeksi hinnangud olid võrreldes eelmise mõõdikuga vähem tähtsamad, kus MI oli olulisem kolmest mõõdikust 2 kuni 4 korda. Halstead-i mõõdik omas 3 korda rohkemat tähtsust kui tsüklomaatiline keerukus ja ABC mõõdikud. Tsüklomaatiline keerukus oli vaid 2 korda tähtsam kui ABC ning ABC ei olnud ühegi eelneva mõõdiku suhtes prioriteetsem.

Tabel 3. Mõõdikute prioritiseerimise hinnangud ChatGPT poolt.

Mõõdik 1	**Paarisvõrdluse hinnang**	**Mõõdik 2**	**Paarisvõrdluse hinnang**
Cyclomatic complexity	1	*Maintainability index*	3
Cyclomatic complexity	2	*ABC*	1
Cyclomatic complexity	1	*Cohesion*	5
Cyclomatic complexity	1	*Halstead*	3
Maintainability index	2	*ABC*	1
Maintainability index	1	*Cohesion*	4

Mõõdik 1	Paarisvõrdluse hinnang	Mõõdik 2	Paarisvõrdluse hinnang
Maintainability index	4	*Halstead*	1
ABC	1	*Cohesion*	6
ABC	1	*Halstead*	3
Cohesion	7	*Halstead*	1

Tabel 4. Mõõdikute protsentuaalsed prioriteedid.

Mõõdik	Prioriteet	Koht tähtsuse järgi
Cyclomatic complexity	7,5%	4
Maintainability index	21,2%	2
ABC	6%	5
Cohesion	53,5%	1
Halstead	11,9%	3

3.3 ChatGPT hinnangud ja kooskõlalisus

Andes ChatGPT-l hinnata nii enda poolt genereeritud parendatud ja halvendatud koodi versioone kui ka inimgenereeritud koodi versioone vastavalt jõudluse, efektiivsuse ning muudele üldiste koodi loetavuse reeglite järgi ja arvutades saadud hinnangute kooskõlalisuse suhtarvu AHP tööriistas, said autorid järgmised tulemused.

3.3.1 ChatGPT poolt genereeritud kood

ChatGPT poolt muudetud koodide versioone puhul selgitasid autorid välja, et ainult viis erineva koodi versioonide komplekti ChatGPT hinnangut neljateistkümnest olid omavahel kooskõlas, teiste komplektide hinnangute puhul näitas aga AHP tööriist suurt vastuolulisust, kus kõige väiksem vastuolulisus kooskõlalisuse suhtarvus moodustas 11,20% ning kõige suurem oli 40%. Keskmine kooskõlalisuse suhtarv on aga 14,69%, mis viitab hinnangute kooskõlalisuse puudumisele enamiku koodide puhul.

Kuna diplomitöö üheks alameesmärgiks oli uurida, kas paludes ChatGPT’d parendada koodi, osutub see koodi versioon tõesti paremaks, selgus, et ainult 9 parendatud koodi versiooni olid ChatGPT hinnangul koodi reeglite puhul tõesti paremad, teistel juhtudel

saavutas parendatud koodi versioon kas teise või kolmanda koha. Üldiselt olid parendatud koodi versioonid keskmiselt 58,52% rohkem prioriteedis teiste koodi versioonide seast. Teist kohta koodi versioonide paremuse järgi sai halvendatud koodi versioon, omades keskmiselt 22,39% prioriteeti koodi versioonide seast ning asetsedes 9 korda teisel kohal paremuse järgi neljateistkümnest. Kõige vähem prioriteetsemaks ehk halvemaks koodi versiooniks ChatGPT hinnangul osutus esialgne koodi versioon 19,08% keskmise prioriteediga teiste koodi versioonide seast, olles 14 koodi puhul 8 korda viimasel kohal prioriteetjärjestuses.

Tabel 5. Koodide versioonide prioriteetjärjestus ja kooskõlalisuse suhtarv ChatGPT hinnangute põhjal ChatGPT genereeritud koodile.

Näitaja **Koodi nimetus**	***GOOD Priority***	***GOOD Rank***	***INITIAL Priority***	***INITIAL Rank***	***BAD Priority***	***BAD Rank***	**CR (*Consistency Ratio*)**
Dikstra	54,00%	1	16,30%	3	29,70%	2	1,00%
Doubly_linked_list	19,60%	3	49,30%	1	31,10%	2	5,60%
Graph	19,60%	3	49,3%	1	31,10%	2	5,60%
Job_scheduling	13,50%	3	58,40%	1	28,1%	2	14,10%
Kosaraju	81,40%	1	11,40%	2	7,20%	3	5,60%
Matrix_class	35,90%	2	12,40%	3	51,70%	1	11,20%
Priority_queue	37,90%	2	11,30%	3	50,80%	1	17,00%
Singly_linked_list	77,00%	1	6,30%	3	16,70%	2	18,40%
Zelle_ex_3	80,80%	1	6,20%	3	13,00%	2	14,10%
Zelle_ex_5	79,70%	1	5,20%	3	15,10%	2	30,80%
Zelle_atm_manager	80,80%	1	6,20%	3	13,00%	2	14,10%
Zelle_ex_2	81,40%	1	11,40%	2	7,20%	3	5,60%

Näitaja **Koodi nimetus**	***GOOD Priority***	***GOOD Rank***	***INITIAL Priority***	***INITIAL Rank***	***BAD Priority***	***BAD Rank***	**CR (*Consistency Ratio*)**
Zelle_Objrball	77,50%	1	17,80%	2	4,70%	3	40,00%
Zelle_chap13_ex11	80,20%	1	5,60%	3	14,10%	2	22,60%
AVG (*Average*)	58,52%	1,571	19,08%	2,357	22,39%	2,071	14,69%

3.3.2 Inimgenereeritud kood

Inimgenereeritud koodi puhul näitavad tulemused, et kolme koodi versioonide komplekti puhul on ChatGPT poolt antud hinnangud kooskõlas ning ülejäänud kolm osutusid aga mittekooskõlaliseks, kus kõige väiksem kooskõlalisuse suhtarvu väärtus moodustas 32,7% ja kõige suurem 58,5%. Arvutades kooskõlalisuse suhtarvu aritmeetilist keskmist inimgenereeritud koodide puhul said autorid väärtuse 21,68%, mis omakorda tähendab, et üldiselt olid ChatGPT hinnangud inimgenereeritud koodidega vastuolus.

ChatGPT hinnangute järgi osutus, et kõik autorite poolt parendatud koodi versioonid olid parema kvaliteediga, kui esialgne ja halvendatud versioonid. Keskmiselt omasid parendatud koodide versioonid 74,5% prioriteeti. Teist kohta jagavad esialgne ja halvendatud koodide versioonid, kuna mõlemad on kolme koodi puhul teisel kohal ja kolme koodi puhul kolmandal kohal. Keskmiselt omab esialgne versioon 11,88% prioriteeti ning halvendatud versioon 13,62%, mis näitab, et need on peaaegu samal tasemel ning ChatGPT hinnangu järgi ei õnnestunud autoritel esialgse koodi kvaliteeti eriti halvendada.

Tabel 6. Koodide versioonide prioriteetjärjestus ja kooskõlalisuse suhtarv ChatGPT hinnangute põhjal inimgenereeritud koodile.

Näitaja **Koodi nimetus**	***GOOD Priority***	***GOOD Rank***	***INITIAL Priority***	***INITIAL Rank***	***BAD Priority***	***BAD Rank***	**CR**
INIMGEN_zelleEx3	73,1%	1	8,1%	3	18,8%	2	6,8%
INIMGEN_zelleEx5	73,1%	1	8,1%	3	18,8%	2	6,8%
INIMGEN_zelleEx2	77,9%	1	18%	2	4,2%	3	58,5%
INIMGEN_zelleAtmManager	70,8%	1	6,2%	3	23%	2	25,0%
INIMGEN_zelleObjrball	75,7%	1	18,8%	2	5,4%	3	32,7%
INIMGEN_zelleChap13Ex11	76,4%	1	12,1%	2	11,5%	3	0,3%
AVG	74,5%	1	11,88%	2,5	13,62%	2,5	21,68%

3.4 Klassikaliste mõõdikute hinnangud ja kooskõlalisus

Kasutades klassikalisi mõõdikuid koodi kvaliteedi hindamiseks nii ChatGPT poolt genereeritud parendatud ja halvendatud koodi versioonide kui ka inimgenereeritud koodi versioonide puhul, arvutasid autorid mõõdikute hinnanguid teisendusreegli abil ja hinnangute kooskõlalisuse suhtarvu AHP tööriistas ning said järgmised tulemused.

3.4.1 ChatGPT poolt genereeritud kood

ChatGPT poolt genereeritud koodide versioonide puhul tulemused näitavad, et kõikide koodide mõõdikute hinnangud on kooskõlas ning ainult ühe koodi hinnangud osutusid

mittekooskõlaliseks, kus kooskõlalisuse suhtarvu maksimaalne väärtus moodustas 15,9%. Arvutades kooskõlalisuse suhtarvu aritmeetilist keskmist said autorcid väärtuse 3,34%, mis omakorda tähendab, et üldiselt olid mõõdikute hinnangud ChatGPT poolt genereeritud koodide puhul kooskõlas.

Mõõdikute hinnangute järgi selgus, et ainult kuus ChatGPT poolt parendatud koodi versiooni olid parima kvaliteediga, teistel juhtudel sai parendatud koodi versioon seitse korda teise koha ja üks kord kolmanda koha. Keskmiselt omas parendatud koodi versioon 36,30% prioriteeti ning asub esimesel kohal. Teise koha koodi versioonide paremuse järgi saavutas halvendatud koodi versioon, omades keskmiselt 33% prioriteeti. Halvendatud koodi versioon asetses kolm korda esimesel kohal paremuse järgi, viis korda teisel kohal ning kuus korda kolmandal kohal. Kõige halvemaks koodi versiooniks mõõdikute hinnangute järgi osutus esialgne koodi versioon, mille puhul keskmine prioriteet teiste koodi versioonide seast oli 30,54%. Esialgne koodi versioon oli neljateistkümne koodi puhul kuus korda esimesel kohal, kolm korda teisel kohal ning viis korda viimasel kohal.

Tabel 7. Koodide versioonide prioriteetjärjestus ja kooskõlalisuse suhtarv mõõdikute hinnangute põhjal ChatGPT genereeritud koodile.

Näitaja **Koodi nimetus**	***GOOD Priority***	***GOOD Rank***	***INITIAL Priority***	***INITIAL Rank***	***BAD Priority***	***BAD Rank***	**CR_{max}**
Dijkstra	33,1%	2	37,9%	1	29%	3	5,6%
Doubly_linked_list	32,8%	2	38%	1	29,2%	3	0%
Graph	36,7%	1	32,7%	2	30,5%	3	5,6%
Job_scheduling	32,1%	2	35,9%	1	32,1%	2	0%
Kosaraju	37,6%	1	34,7%	2	27,7%	3	5,6%
Matrix_class	42,4%	1	28%	3	29,6%	2	5,6%
Priority_queue	30,8%	3	37,4%	1	31,9%	2	5,6%
Singly_linked_list	37,5%	1	25%	3	37,5%	1	0%

Näitaja **Koodi nimetus**	***GOOD Priority***	***GOOD Rank***	***INITIAL Priority***	***INITIAL Rank***	***BAD Priority***	***BAD Rank***	**CR_{max}**
Zelle_ex_3	35,8%	2	19,7%	3	44,5%	1	0%
Zelle_ex_5	43,8%	1	15,6%	3	40,6%	2	1,9%
Zelle_ex_2	45,5%	1	19,6%	3	34,9%	2	15,9%
Zelle_atm_manager	35,3%	2	35,8%	1	28,9%	3	1%
Zelle_Objrball	33,5%	2	35,9%	1	30,6%	3	0%
Zelle_chap13_ex11	31,3%	2	31,3%	2	37,5%	1	0%
AVG	36,30%	1,64	30,54%	1,93	33%	2,21	3,34%

Lisas 3 on välja toodud tabel ChatGPT poolt genereeritud koodide iga mõõdiku prioriteedi ja kooskõlalisuse suhtarvu väärtuste kohta.

3.4.2 Inimgenereeritud kood

Inimgenereeritud koodi puhul tuli välja, et kõik kuue koodi versioonide komplekti hinnangud on omavahel kooskõlas, kus kooskõlalisuse suhtarvu keskmine moodustab 3,73%, kõige suurem kooskõlalisuse suhtarv koodide seast on 5,6% ning kõige väiksem 0%. Lisaks sellele on hinnangute koondtabelist näha, et parendatud koodi versioon oli keskmiselt 54,60% prioriteedis teiste koodi versioonide seast, olles viis korda esimesel kohal parema kvaliteedi järgi ja üks kord teisel kohal. Järgmisel kohal prioriteedi järgi seisab esialgne koodide versioon, kus selle keskmine moodustab 23,53% prioriteeti teiste versioonide hulgast ning iga koodi puhul saavutas esialgne versioon teise koha koodi kvaliteedi järgi, välja arvatud *INIMGEN_zelleEx2*, kus ta seisab kolmandal kohal. Viimasel kohal seisab aga halvendatud koodide versioon, omades keskmiselt 21,88% prioriteeti ning olles viis korda kolmandal kohal koodi versioonide seast koodi kvaliteedi järgi ning ainult üks kord esikohal.

Tabel 8. Koodide versioonide prioriteetjärjestus ja kooskõlalisuse suhtarv mõõdikute hinnangute põhjal inimgenereeritud koodile.

Näitaja **Koodi nimetus**	***GOOD Priority***	***GOOD Rank***	***INITIAL Priority***	***INITIAL Rank***	***BAD Priority***	***BAD Rank***	**CRmax**
INIMGEN_zelleEx3	59,90%	1	25,60%	2	14,50%	3	5,60%
INIMGEN_zelleEx5	61,90%	1	22,40%	2	15,80%	3	5,60%
INIMGEN_zelleEx2	26,50%	2	23,6%	3	49,90%	1	0,00%
INIMGEN_zelleAtmManager	56,80%	1	26,20%	2	16,9%	3	0,00%
INIMGEN_zelleObjrball	55,50%	1	25,20%	2	19,30%	3	5,60%
INIMGEN_zelleChap13Ex11	67,00%	1	18,20%	2	14,90%	3	5,60%
AVG	54,60%	1,167	23,53%	2,167	21,88%	2,667	3,73%

Lisas 4 on välja toodud tabel inimgenereeritud koodide iga mõõdiku prioriteedi ja kooskõlalisuse suhtarvu väärtuste kohta.

4 Analüüs

Käesolevas peatükis annavad autorid hinnangu saadud tulemustele ChatGPT poolt genereeritud koodi ning inimgenereeritud koodi puhul, sealhulgas valitud mõõdikutele koos eeliste ja puudustega, AHP tööriistale ning põhjendavad puhta koodi reeglite valiku. Lisaks käsitletakse analüüsis alternatiivseid lähenemisviise, pakutakse võimalikku töö edasiarendust ning lõpuks avaldatakse autorite arvamus teostatud töö protsessi kohta.

4.1 Hinnang tulemustele

Antud alapeatükk kajastab autorite hinnanguid nii ChatGPT kui ka inimgenereeritud koodide tulemustele.

4.1.1 ChatGPT poolt genereeritud kood

Analüüsides ChatGPT ja mõõdikute hinnanguid ChatGPT poolt genereeritud koodile tulid autorid järeldusele, et üldiselt osutusid ChatGPT hinnangud enamikul juhtudel mittekooskõlalisteks ja ainult mõnel juhul olid need kooskõlas, mida näitab keskmine kooskõlalisuse suhtarv 14,69%. Mõõdikute hinnangud tehisintellekti poolt genereeritud koodile olid aga alati kooskõlalised, välja arvatud ühe koodi puhul, ning keskmine kooskõlalisuse suhtarv moodustas ainult 3,34%. See tõestab, et klassikalised mõõdikud on usaldusväärsem viis paarikaupa hinnangute saamiseks kui ChatGPT vestlusrobot, ning ChatGPT hinnangutesse tuleks suhtuda suurema skeptilisusega.

Tabel 9. Koodide versioonide prioriteetjärjestuse ja kooskõlalisuse suhtarvu keskmiste tabel ChatGPT poolt genereeritud koodile.

Keskmine näitaja **Hinnangu tüüp**	***GOOD Priority* (AVG)**	***GOOD Rank* (AVG)**	***INITIAL Priority* (AVG)**	***INITIAL Rank* (AVG)**	***BAD Priority* (AVG)**	***BAD Rank* (AVG)**	**CR (AVG)**
ChatGPT hinnangud	58,52%	1,571	19,08%	2,357	22,39%	2,071	14,69%
Mõõdikute hinnangud	36,30%	1,64	30,54%	1,93	33%	2,21	3,34%

ChatGPT hinnangute peale vaadates oli sageli näha, et konkreetne hinnang on mittekooskõlaline ja ebausaldusväärne, kuna vaadeldud versioon ei olegi nii palju halvem või parem kui teine. Ning mõnikord juhtus ka nii, et ChatGPT andis parema versiooni oma arvamusel, kuid selle koodi versiooni peale vaadates on näha, et tegelikult see on halvem kui esialgne versioon või samaväärne.

INITIAL	1	*GOOD*	9
INITIAL	2	*BAD*	1
GOOD	9	*BAD*	1

Joonis 24. ChatGPT hinnangud *Kosaraju* koodi versioonidele, mis olid genereeritud tehisintellekti poolt.

Näiteks ChatGPT paarilise võrdluse hinnang tehisintellekti poolt genereeritud *Kosaraju* koodi „*INITIAL*/*GOOD*“ paari jaoks oli 1/9, mis tähendab, et ChatGPT hinnangul on parendatud versioon 9 korda parem, kui esialgne versioon. Aga koodi struktuuri ja kvaliteedi analüüsides järeldub, et „*GOOD*“ versioon on „*BAD*“ versioonist parem, kuid mitte 9 korda.

```
class Kosaraju:

    def dfs(self, i, V, adj, visited, stk):
        visited[i] = 1

        for x in adj[i]:
            if visited[x] == -1:
                self.dfs(x, V, adj, visited, stk)

        stk.append(i)

    def kosaraju(self, V, adj):

        stk, visited = [], [-1]*(V+1)

        for i in range(V):
            if visited[i] == -1:
                self.dfs(i, V, adj, visited, stk)

        stk.reverse()
        res = stk.copy()

        ans, visited1 = 0, [-1]*(V+1)

        adj1 = [[] for x in range(V)]

        for i in range(len(adj)):
            for x in adj[i]:
                adj1[x].append(i)

        for i in range(len(res)):
            if visited1[res[i]] == -1:
                ans += 1
                self.dfs(res[i], V, adj1, visited1, stk)

        return ans
```

Joonis 25. *Kosaraju* koodi *INITIAL* versioon.

```
class Kosaraju:
    def init(self, vertices):
        self.vertices = vertices
        self.adjacency_list = [[] for _ in range(vertices)]
        self.adjacency_list_transpose =
        [[] for _ in range(vertices)]

    def add_edge(self, src, dest):
        self.adjacency_list[src].append(dest)

    def dfs(self, vertex, visited, stack):
        visited[vertex] = True
        for neighbor in self.adjacency_list[vertex]:
            if not visited[neighbor]:
                self.dfs(neighbor, visited, stack)
        stack.append(vertex)

    def transpose_graph(self):
        for src in range(self.vertices):
            for dest in self.adjacency_list[src]:
                self.adjacency_list_transpose[dest].append(src)

    def dfs_scc(self, vertex, visited):
        visited[vertex] = True
        for neighbor in self.adjacency_list_transpose[vertex]:
            if not visited[neighbor]:
                self.dfs_scc(neighbor, visited)

    def kosaraju(self):
        stack = []
        visited = [False] * self.vertices

        for vertex in range(self.vertices):
            if not visited[vertex]:
                self.dfs(vertex, visited, stack)

        self.transpose_graph()
        visited = [False] * self.vertices
        num_scc = 0
        while stack:
            vertex = stack.pop()
            if not visited[vertex]:
                self.dfs_scc(vertex, visited)
                num_scc += 1
        print("Number of Strongly Connected
        Components:", num_scc)
```

Joonis 26. *Kosaraju* koodi *GOOD* versioon.

Rääkides ChatGPT poolt genereeritud koodidest näitas pingerida, et halvendatud koodi versioonid ei osutunud tegelikkuses ChatGPT ja mõõdikute hinnangutel halvimateks – mõlemal juhul oli „*BAD*“ versioon prioriteedi järgi keskmiselt teisel kohal ja omandas vastavalt 22,39% ja 33% prioriteeti. See tähendab, et ChatGPT sai halvendatud koodi versiooni genereerimise ülesandega halvasti hakkama. Kuid hea koodi versiooni genereerimise ülesannega sai ChatGPT paremini hakkama, kuna nii ChatGPT kui ka mõõdikute hinnangul asus „*GOOD*“ versioon prioriteedi järgi keskmiselt esimesel kohal ning omandas vastavalt 58,52% ja 36,30% prioriteeti. Tabelist aga on näha, et vahe hea versiooni ja algse versiooni vahel ei olnud nii suur, mis tähendab, et ChatGPT ei suutnud koodi palju parendada.

4.1.2 Inimgenereeritud kood

Uurides saadud protsentuaalseid prioriteete inimgenereeritud koodile võib järeldada, et võrreldes ChatGPT poolt muudetud koodidega olid autorite poolt muudetud verisoonid parema osakaaluga. Autorid suutsid koodi kvaliteeti hästi parendada ning mõlemate hinnangute saamise viiside puhul ületas saadud prioriteet 50%. Lisaks õnnestus mõõdikute arvates järjestada ka õige „*GOOD*“, „*INITIAL*“, „*BAD*“ koodi versioonide hierarhia, kuigi halvendatud versiooni keskmine polnud esialgsest eriti kehvem. Siiski ei tulnud ChatGPT hinnangute põhjal koodi versioonide hierarhia korrektne välja ja halvendatud versioon osutus veidi parema kvaliteediga olevat kui esialgne versioon. Prioriteetsuse poolest jagavad aga mõlemad versioonid nii teist kui ka kolmandat kohta.

Tabel 10. Koodide versioonide prioriteetjärjestuse ja kooskõlalisuse suhtarvu keskmiste tabel inimgenereeritud koodile.

Keskmine näitaja / **Hinnangu tüüp**	***GOOD Priority* (AVG)**	***GOOD Rank* (AVG)**	***INITIAL Priority* (AVG)**	***INITIAL Rank* (AVG)**	***BAD Priority* (AVG)**	***BAD Rank* (AVG)**	**CR (AVG)**
ChatGPT hinnangud	74,5%	1	11,88%	2,5	13,62%	2,5	21,68%
Mõõdikute hinnangud	54,60%	1,167	23,53%	2,167	21,88%	2,667	3,73%

Lisaks sellele saavutasid autorid mõõdikute hinnangute kohaselt kooskõlalisuse kõigi koodi versioonide puhul, samas kui ChatGPT hinnangute tulemusena olid 3 koodikomplekti kuuest suures vastuolus. Näiteks hindas vestlusrobot *INIMGEN_zelleEx2* koodi versioone järgmiselt:

INITIAL	1	*GOOD*	9
INITIAL	9	*BAD*	1
GOOD	9	*BAD*	1

Joonis 27. ChatGPT hinnangud *INIMGEN_zelleEx2* koodi versioonidele, mis olid genereeritud autorite poolt.

Nagu on näha, andis ChatGPT versioonide paarisvõrdluses maksimaalseid hinnanguid AHP skaalas 1 kuni 9, kus viimane hinnang näitab ühe koodi ekstreemset paremust teise suhtes. Selle tulemusena oli parendatud versioon esikohal 77,9% prioriteediga, seejärel esialgne versioon 18%-ga ning halvendatud versioon 4,2%-ga. Kooskõlaisuse suhtarv oli selle komplekti puhul aga kõikides tabelites olevatest koodide suhtarvudest maksimaalne – 58,5%, mis näitab äärmislikku mittekooskõlalisust, teiste sõnadega pole antud paarisvõrdluse hinnangud loogilised. Samas kui mõõdikute alusel osutus prioriteetseimaks versiooniks just halvendatud versioon 49,90%-ga, teise koha saavutas parendatud versioon 26,5% ja viimase koha esiglane versioon 23,6%-ga, kus versioonid olid vastupidi täielikus kooskõlas – 0%. *INIMGEN_zelleEx2* koodi versioone uurides pooldavad autorid just mõõdikute hinnanguid [27].

Vaatamata sellele, et mõõdikute väärtuste põhjal olid kõik koodi komplektide hinnangud kooskõlas ning keskmine CR moodustab 3,73%, selgus, et ChatGPT hinnangute puhul on sama suhtarv kõigi vaadeldud keskmiste suhtarvude seast kõige suurem – 21,68%. Taoline suhtarvude erinevus viitab ühe hinnangute saamise viisi ekslikkusele ning vaadeldes koodide versioone, näib, et mõõdikute hinnangud on täpsemad.

Antud analüüsist saab järeldada, et autorid said koodi parandamise ja halvendamise ülesandega paremini hakkama versioonide järjestamise kontekstis kui ChatGPT ning et mõõdikud annavad koodi kvaliteedi taseme määratlemiseks usaldusväärsemaid ja korrektsemaid hinnanguid, kui seda teeb ChatGPT, mille vastused osutusid taas pigem juhuslikeks ning vähem usutavateks.

4.2 Praktilise teostuse põhjendus

Käesolevas alapeatükis põhjendatakse mõõdikute komplekti, AHP tööriista ning puhta koodi reeglite valikut.

4.2.1 Miks just sellised mõõdikud?

Üheks viisiks koodi kvaliteeti hinnata valisid autorid mõõdikuid, kuna erinevalt ChatGPT üldistest hinnangutest põhinevad need kindlatel valemitel, mis võtavad arvesse koodi erinevaid mõõdetavaid ja loendatavaid aspekte, nagu pikkus, kommentaaride arv, operandide ja operaatorite kasutuse arv, meetodite omavaheline sidusus klassis või muu. Lõpliku mõõdikute komplekti kuuluvad mitmed objektorienteeritud disaini-, klassipõhised- ning tarkvaramõõdikud. Eesmärgiks oli valida mitmekesist kogumit, mis võimalikult laialt kajastaks kasuliku informatsiooni iga koodi kohta parima koodialternatiivi valikuks. Vaatamata sellele, et on välja töötatud tohutu hulk mõõdikuid koodi kvaliteedi hindamiseks, valisid autorid välja nende seast sagedamini kasutatavad ja olulisemad, mis esinesid kõige tihemini programmeerimise raamatutes ja teadusartiklites [15], [18], [33], [34]. Kuigi oli antud töös kasutusel viis mõõdikut: kohesioon, Halstead, tsüklomaatiline keerukus, hooldatavuse indeks ja ABC, pidi algne kogum olema suurem. Üheks tähtsaks mõõdikuks, mis ei sattunud mõõdikute komplekti, on sidestus. Sidestus näitab, kuivõrd palju on üks moodul seotud teiste moodulite, väliskeskkonna ja globaalsete andmetega [33]. Lisaks sellele jäi välja ka MS (*Method similarity*) mõõdik, mis näitab, kuidas on kaks meetodit seotud ühiste muutujate kasutamisega klassis ning kuivõrd need meetodid on teineteisega sarnased [35]. Põhjuseks nendest mõõdikutest loobumiseks oli, et peale mõõdikute sagedast kasutamist lõhnatuvastajatena oli autorite jaoks oluline, et antud mõõdiku jaoks eksisteeriks Python moodul, mis võimaldaks koheselt saada mõõdikute tulemusi iga koodi jaoks. Kahjuks ei suutnud autorid leida hästi töötavaid sidestuse ja MS mooduleid, seega alternatiivina sidestuse mõõdikule oli valitud ABC mõõdik, mida esialgselt polnud mõõdikute hulgas, kuid mis ikkagi annab kasulikku infot koodi pikkuse kohta ning lisaks oli valitud ka LCOM4 mõõdik, mis on mõeldud klassi meetodite sidususe määramiseks.

Analüüsides olemasoleva komplekti peale selle kasutamist saab väita, et mõnedel mõõdikute moodilutel kerkisid esile teatud puudused. Nii näiteks arvestab hooldatavuse indeksi mõõdik kommentaaride arvu hulka, mille tõttu võib mõõdiku väärtus drastiliselt

langeda, kui väärtuse arvutamisel jääb faili sisse mõni välja kommenteeritud funktsioon või koodi versioon. Lisaks ei arvesta ükski mõõdik Larmani GRASP (*General Responsibility Assignment Software Patterns*) Eksperdi mustriga. Eksperdi mustri kohaselt tuleb anda vastutust klassile, kus on selle täitmiseks vajalikud atribuudid [36]. Peab arvestama ka sellega, et ühes klassis ei tohi kutsuda teise klassi atribuute välja, vaid selleks peab olema vastav atribuuti tagastav meetod. Seega kui parandatud versioonides oli Eksperdi muster rikutud, polnud see avaldanud mõõdikute väärtustele negatiivset mõju ning ühe koodi puhul suurendasid atribuutide tagastamise meetodid kohesiooni mõõdiku väärtust 1-st 4-ni, kus viimane tähendab, et klassi on tarvis jagada mitmeks klassiks, kuigi see pole mõistlik, kuna tegu oli *User* klassiga, kus atribuutideks oli 5 kasutaja tunnust.

4.2.2 Miks just see AHP tööriist?

AHP tundma õppimise alguses leidsid autorid rakenduse, mis täielikult sobis kooskõlalisuse hindamiseks. Nimetatud rakendusel oli intuitiivne kasutajaliides, võimalus alternatiivide lisamiseks ning paariliste võrdluste läbiviimiseks ja erinevad graafikud tulemuste näitamiseks, kuid rakendus osutus tasuliseks [37]. Seega jätkates sobiva rakenduse või veebilehe otsimist, leiti veebileht, mis pakkus tasuta demo versiooni analüütiliste hierarhiate protsessi läbiviimiseks. Antud veebileht osutus mittesobivaks, kuna tulemus ei näidanud, milline alternatiiv oli üldiselt kõige parem ja prioriteetsem, vaid näitas ainult prioriteete ning pingerida konkreetselt iga mõõdiku kohta [38]. Autorid leidsid ka muid rakendusi või veebilehti kooskõlalisuse arvutamiseks, kuid ka seal kas puudus vajalik funktsionaalsus või ei olnud need intuitiivsed ega töötanud üldse.

Tänu pikale otsingule õnnestus autoritel leida sobiv veebileht, mis pakkus kõike vajalikku kooskõlalisuse hindamiseks ja informatiivsete tulemuste saamiseks – kriteeriumite hierarhia loomist, paariliste võrdluste läbiviimise võimalust, alternatiivide lisamist, kooskõlalisuse arvutamist ning prioriteete ja pingerea näitamist nii iga kriteeriumi jaoks kui ka üldiselt [39]. Veebilehe täiendavaks eeliseks oli see, et hierarhia puudumisel oli võimalik ka ühe kriteeriumi põhjal alternatiivide paarilisi võrdlusi läbi viia ning saada kätte arvutatud prioriteete ja pingerida. Kuna kõik vajalik funktsionaalsus oli veebilehel olemas, siis puuduseid autorid ei märganud.

4.2.3 Millised reeglid olid kasutatud ChatGPTle andmiseks ning miks just need?

ChatGPT abil koodi genereerimiseks kasutasid autorid kõige levinumaid puhta koodi reegleid Robert Martini „Clean Code: A Handbook of Agile Software Craftsmanship“ raamatust. Samu reegleid kasutasid autorid ka oma koodi genereerimiseks. Täpsemalt olid konspekteeritud reeglid teisest, kolmandast ning kümnendast peatükist, kus on käsitletud nimede, funktsioonide ja klasside reeglid [26]. Nimetatud reeglid olid valitud seetõttu, et neid peetakse üldtuntud reegliteks ja need sobivad autorite poolt püstitatud ülesande täitmiseks.

Lisades 5, 6, 7 on välja toodud tabelid valitud puhta koodi reeglite kohta klasside, meetodite ja nimede kontekstis.

4.3 Alternatiivsed lähenemisviisid

Otsides alternatiivseid töid teemal, kus uuritakse, kuivõrd ChatGPT hinnangud erinevate koodi versioonidele on kooskõlas, ei suutnud autorid sarnast tööd leida. Vaatama sellele, kuna antud töös käsitleti ka ChatGPT võimet genereerida parendatud ja halvendatud koodi versioone vastavalt kasutaja päringule, leidsid autorid teadusartikli, kus uuriti 728 ChatGPT poolt viiel programmeerimiskeelel genereeritud koodi kvaliteeti, kus koodi hindamine oli keskendatud kolmele aspektile, milleks osutusid keerukus, korrektsus ja turvalisus. Analüüsi käigus selgus, et ChatGPT võime teha töötava funktsionaalsuse saavutamise nimel parandusi vigases koodis oli suhteliselt nõrk. Lisaks sellele erines kasutatud keelte koodiosade kognitiivse ja tsüklomaatilise keerukuse tasemete jaotus. Samas saab ChatGPT uuringu kohaselt paremini hakkama koodi genereerimisega probleemidele enne 2021. aastat kui probleemidele pärast 2021. aastat ning mitmekordse vigaste koodiosade parandamise protsessi käigus üle 89% haavatavustest said eemaldatud [40].

Kuna selles töös võrdlesid autorid ChatGPT ja mõõdikute käest saadud hinnanguid, oli leitud artikkel, kus viidi läbi uuring ChatGPT võimekusega järjestada viie mudeli genereeritud vastuste kogumeid, mis sisaldasid erinevaid kasutusjuhtumeid. Seejärel võrreldi saadud tulemusi mudelite vastuste järjestusega inimese poolt ning leiti, et ChatGPT eelistuste järjestus oli teatud ulatuses kooskõlas inimese eelistustega [41].

4.4 Edasiarendus

Kuna töö käigus koosnes mõõdikute komplekt viiest mõõdikust, oleks üheks edasiarenduse võimaluseks suurendada mõõdikute kogumit, sest hetkel polnud arvesse võetud mõnda tähtsat mõõdikut, nagu näiteks sidestus [33]. Olemasolevate mõõdikute puhul omas kõige suuremat prioriteeti kohesiooni mõõdik osakaaluga 53,5%, seega oleks kasulik uurida, kuivõrd prioriteetne oleks sidestus teiste mõõdikute suhtes ning kuidas muudaks selle mõõtmine koodi versioonide hinnangute kooskõlalisust. Mida rohkem on vaatluses mõõdikuid, mis hõlmavad koodi erinevaid aspekte, seda täpsemaid hinnanguid koodi kvaliteedile on võimalik tulemusena saada.

Lisaks, kuna Transformeri mudelil põhinev ChatGPT sai koodi versioonide hindamise ülesandega kehvasti hakkama, andes mittekooskõlalisi hinnanguid, võiks alternatiivselt kaaluda edasiarenduseks tehisintellektuaalse masinõppe mudeli õpetamist koodi kvaliteeti hindama eesmärgiga vaadata, kuivõrd hästi ja kiiresti suudaks ta koodilõhnasid tuvastada ning kas selle hinnangud oleksid rohkem kooskõlas. Masinõpe mudel võiks põhineda konvolutsioonilistel (CNN (*Convolutional Neural Networks*)) [42] või rekurrentsetel närvivõrkudel (RNN) [43]. CNN oleks soovitatav valik, kuna see koosneb konvolutsioonilistest, koondavatest ja täielikult ühendatud kihtidest ning on kasutusel tekstide, objektide ja koodide tuvastamisel ning lausete klassifitseerimisel [42], [44]. Eelkõige oleks tarvis olemasolevate koodi versioonide põhjal viia läbi andmete annotatsiooni, kus oleksid märgistatud nii halva kui ka hea kvaliteediga koodiosad. Peale seda, kui kood on masinale loetavaks muudetud, õpiks ta algsete koodide põhjal koodilõhnasid ära tundma. Lõpuks oleks viidud läbi kontroll, kus õpetatud masin hindaks testandmete põhjal koodi kvaliteeti eri koodireeglite alusel.

Samuti polnud koodi versioonide parendamisel ja halvendamisel arvestatud Larmani GRASP Ekspert mustriga ning sellele sarnase Demetri *Information hiding* seadusega. Nii *Information hiding* kui ka Ekspert mustri kohaselt tuleks vältida ühel meetodil otse teise objekti alamosa väljakutsumist, vaid selleks peab kasutama vahepealseid meetodeid, mis atribuute tagastavad [45]. Seega oleks esiteks vajalik parendatud versiooni puhul sellest kinni pidada ja halvendatud koodi versiooni puhul antud seadust rikkuda ning leida vastav mõõdik, mis võtaks seda printsiipi arvesse.

Kuna antud töös kasutasid autorid ChatGPT-ga suhtlemiseks selle 3.5 versiooni, võib edasiarendusena kaaluda ka uuema ja võimekama ChatGPT 4.0 mudeli kasutamist [46] või Meta alternatiivset keelemudelit Llama [47].

4.5 Hinnang töö teostamise protsessile ja projekti logid

Antud töösse panustasid mõlemad autorid võrdselt. Töö protsess kestis 4 kuud, kus ülesanded ühise eesmärgi täitmiseks olid jagatud võrdselt mõlema meeskonnaliikme vahel. Mõlemad liikmed tegelesid teadusartiklite otsimisega, avalikust lähtekoodist mitmesuguste koodide kogumisega, puhta koodi reeglite kogumisega ja erinevate tarkvara mõõdikute uurimise ning nende jaoks moodulite otsimisega. Kogutud kood oli jaotud võrdselt kahe liikme vahel, mille põhjal mõlemad liikmed tegelesid koodi versioonide genereerimisega ChatGPT vestlusroboti abil, koodi versioonide parendamise ja halvendamisega, mõõdikute väärtuste arvutamisega koodi versioonidele ning nende väärtuste hinnanguteks teisendusega *Google Sheets´*is [31], ChatGPT üldiste hinnangute saamisega koodi versioonidele, mõõdikute ja ChatGPT hinnangute ülekandmisega AHP tööriista, kooskõlalisuse tulemuste saamisega ning dokumentatsiooni kirjutamisega.

Üldine koostöö toimus diplomitöö kirjutamise vältel suurepäraselt ja sujuvalt, kus liikmed hoidsid igapäevaselt ühendust, arutades töö järgmisi samme, jaotades omavahel ülesandeid ning visualiseerides, kuidas peab praktilise osa protsess algusest lõpuni välja nägema. Juhendajaga pidasid autorid iganädalaseid konsultatsioone Teams´i vahendusel, kus rääkisid kindla nädala täidetud ülesannetest, jagasid oma muresid ning edukohti. Lisaks tekkis palju küsimusi konsultatsioonivälisel ajal, kus juhendaja oli alati valmis selgust konteksti sisse tooma. Kuna AHP protsessiga tutvusid autorid esimest korda, siis ainuke kitsaskoht tekkis bakalaureusetöö kirjutamise alguses, kus töö spetsiifilisuse tõttu kulus palju aega sellele, et kõik meeskonnaliikmed jõuaksid ühisele arusaamisele ja nägemusele praktilise osa kohta protsessi kohaselt, kuna esmalt kujutas iga liige ühise eemärgini jõudmise protsessi omamoodi. Antud takistuse kõrvaldamiseks kulus mitu nädalat arutamist, teadusartiklitesse süvenemist ning juhendajaga detailide täpsustamist. Võib ka öelda, et mida rohkem liikmed praktilise osaga tegelesid, seda täpsemaks ja arusaadavamaks muutus eesmärk.

Tähelepanekutest võib tuua välja ainult seda, et mõnikord polnud liikmed mõningaid aspekte või detaile arvesse võtnud või polnud need eelnevalt läbi räägitud, mille tõttu pidi parandusi tegema.

4.5.1 Sofia Senkiv'i ajalogide kokkuvõte

Sofia Senkiv'i tegevused nädala kaupa on kirjeldatud järgmises tabelis (Tabel 13).

Tabel 11. Sofia Senkiv'i ajalogide kokkuvõte.

Nädal	**Tegevused**
29.01–04.02	• Esialgse bakalaureusetöö teema välja mõtlemine
05.02–11.02	• Teema kinnitamine ja esialgne sõnastamine • Eesmärkide sõnastamine • Hea koodi tavade uurimine ja kogumine • Tehisintellekti õpetamiseks mudelite info kogumine
12.02–18.02	• Lõputöö dokumentatsioon (sissejuhatus) • Ülesandepüstitus
19.02–25.02	• GraphCode2Vec uurimine • Transformer, CNN uurimine ja konspekteerimine • Lõputöö dokumentatsioon (eesmärgid)
26.02–03.03	• Tööriistade kirjeldamine dokumentatsioonis • Esialgsete mõõdikute kogumine • Puhta koodi reeglite kogumine • PyTorch ja CUDA seadistamine • IPython installeerimine
04.03–10.03	• Diplomitöö teema muutmine • Sissejuhatuse, eesmärkide ja probleemi osa muutmine dokumentatsioonis • Koodi komplekti kogumine • ChatGPT abil koodide kvaliteedi halvendamine ja parendamine
11.03–17.03	• Mõõdikute uurimine, mis lõpuks ei läinud mõõdikute komplekti sisse (sidestus, klassi suurus)

	• Lõplike tarkvara mõõdikute uurimine ja kinnitamine (tsüklomaatiline keerukus, hooldatavuse indeks, ABC) • Mõõdikute moodulite otsimine • Mõõdikute väljakirjutamine dokumentatsioonis
18.03–24.03	• ChatGPT poolt genereeritud koodi versioonide kontrollimine, mõõdikute väärtuste saamine 12-ne koodi versioonide jaoks, nende sisestamine *Google Sheets* töölehte • Digikogust sarnaste tööde olemasolu kontrollimine • Analüütiliste hiearhiate protsessiga tutvumine, teadusartiklite lugemine
25.03–31.03	• Uue ABC mõõdiku jaoks mooduli otsimine, selle kohta lugemine • ABC mõõdiku dokumenteerimine • Uue ABC ja LCOM4 mõõdiku väärtuste arvutamine olemasoleva 12-ne koodi versioonide jaoks
01.04–07.04	• AHP tööriista valimine ja sellega tutvumine dokumentatsiooni järgi • Täiendava koodi kogumi otsimine uurimishulga laiendamiseks • Uue koodi kogumi halvendatud ja parendatud versioonide genereerimine ChatGPT poolt (kolme koodi jaoks) • Teisendusreegli välja mõtlemine • ChatGPT poolt mõõdikute prioritiseerimise genereerimine pairwise comparison alusel
08.04–14.04	• ChatGPT poolt genereeritud täiendava koodi komplekti versioonide kontrollimine, mõõdikute väärtuste saamine üheksa koodi versiooni jaoks, nende sisestamine *Google Sheets* töölehte • Olemasolevate koodide mõõdikute väärtuste teisendamine hinnanguteks vastavalt teisendusreeglile • ChatGPT hinnangute genereerimine koodi paaride kaupa • AHP tööriistas koodi hinnangute CR arvutamine
15.04–21.04	• Kolme koodi kvaliteedi parendamine ja halvendamine käsitsi • Inimgenereeritud 9 koodi versioonide jaoks mõõdikute väärtuste ja nende põhjal hinnangute arvutamine, versioonide paarisvõrdluse hinnangute saamine ChatGPT poolt • Hinnangute kandmine AHP tööriista sisse ning hinnangute CR´ide arvutamine • ChatGPT poolt valesti genereetitud ühe koodi versiooni parandamine, uue koodi vestlusrobotilt saamine, uute mõõdikute

	väärtuste ja ChatGPT poolt hinnangute saamine, uute CR´ide arvutamine • Seitsme koodi versioonide puhul MI mõõdiku ümber arvutamine, seejärel ka CR´i väärtuste ümber arvutamine
22.04–28.04	• Metoodika osa kirjutamine
29.04–05.05	• Tulemuste osa kirjutamine dokumentatsioonis
06.05–12.05	• Analüüsi osa kirjutamine • Kokkuvõte osa kirjutamine
13.05–19.05	• Dokumentatsiooni nõuetekohane vormistamine

4.5.2 Anastassija Gontšarova ajalogide kokkuvõte

Anastassija Gontšarova tegevused nädala kaupa on kirjeldatud järgmises tabelis (Tabel 14).

Tabel 12. Anastassija Gontšarova ajalogide kokkuvõte.

Nädal	**Tegevused**
29.01–04.02	• Esialgse bakalaureusetöö teema välja mõtlemine
05.02–11.02	• Teema kinnitamine ja esialgne sõnastamine • Eesmärkide sõnastamine • Tehisintellekti õpetamise sammude uurimine ja kogumine (andmete annotatsioon) • Tehisintellekti õpetamiseks mudelitega tutvumine
12.02–18.02	• Lõputöö dokumentatsioon • Ülesandepüstitus • (Esialgne) Probleemi ja eesmärkide püstitamine
19.02–25.02	• Sissejuhatuse teise osa kirjutamine • GraphCode2Vec teadusartiklite lugemine • Transformer, CNN teadusartiklite lugemine, konspekteerimine
26.02–03.03	• Tööriistade kirjeldus dokumentatsioonis

	• Esialgsete mõõdikute kogumine • Puhta koodi reeglite kogumine • PyTorch ja CUDA seadistamine
04.03–10.03	• Diplomitöö teema muutmine • Sissejuhatuse, eesmärkide ja probleemi osa muutmine dokumentatsioonis • Koodi komplekti kogumine • ChatGPT abil koodide kvaliteedi halvendamine ja parendamine
11.03–17.03	• Mõõdikute uurimine, mis lõpuks ei läinud mõõdikute komplekti sisse (MS, The new metric for Class Cohesion) [35] • Lõplike tarkvara mõõdikute uurimine ja kinnitamine (halstead, cohesion) • Mõõdikute moodulite otsimine • Mõõdikute väljakirjutamine dokumentatsioonis
18.03–24.03	• ChatGPT poolt genereeritud koodi versioonide kontrollimine, mõõdikute väärtuste saamine 12-ne koodi versioonide jaoks, nende sisestamine *Google Sheets* töölehte • Digikogust sarnaste tööde olemasolu kontrollimine • Analüütiliste hiearhiate protsessiga tutvumine, teadusartiklite lugemine
25.03–31.03	• Uue LCOM4 mõõdiku Python mooduli otsimine, selle kohta lugemine • LCOM4 mõõdiku dokumenteerimine • Uue ABC ja LCOM4 mõõdikute väärtuste arvutamine olemasolevate 12-ne koodi versioonide jaoks
01.04–07.04	• AHP tööriistaga tutvumine dokumentatsiooni järgi • Täiendava koodi kogumi otsimine uurimishulga laiendamiseks • Uue koodi kogumi halvendatud ja parendatud versioonide genereerimine ChatGPT poolt (kolme koodi jaoks) • Teisendusreegli välja mõtlemine • ChatGPT poolt mõõdikute prioritiseerimise genereerimine pairwise comparison alusel
08.04–14.04	• ChatGPT poolt genereeritud täiendava koodi komplekti versioonide kontrollimine, mõõdikute väärtuste saamine üheksa koodi versiooni jaoks, nende sisestamine *Google Sheets* töölehte

	• Olemasolevate koodide mõõdikute väärtuste teisendamine hinnanguteks vastavalt teisendusreeglile • ChatGPT hinnangute genereerimine koodi paaride kaupa • AHP tööriistas koodi hinnangute CR arvutamine
15.04–21.04	• Kolme koodi kvaliteedi parendamine ja halvendamine käsitsi • Inimgenereeritud 9 koodi versioonide jaoks mõõdikute väärtuste ja nende põhjal hinnangute arvutamine, versioonide paarisvõrdluse hinnangute saamine ChatGPT poolt • Hinnangute kandmine AHP tööriista sisse ning hinnangute CR´ide arvutamine • ChatGPT poolt valesti genereetitud kahe koodi versioonide parandamine, uue koodi vestlusrobotilt saamine, uute mõõdikute väärtuste ja ChatGPT poolt hinnangute saamine, uute CR´ide arvutamine • Seitsme koodi versioonide puhul MI mõõdiku ümber arvutamine, seejärel ka CR´i väärtuste ümber arvutamine
22.04–28.04	• Metoodika osa kirjutamine
29.04–05.05	• Tulemuste osa kirjutamine
06.05–12.05	• Analüüsi osa kirjutamine • Kokkuvõte osa kirjutamine • Enda parendatud koodis vea parandamine (mõõdikute arvutamine, kooskõlalisuse suhtarvude saamine AHP tööriista abil)
13.05–19.05	• Dokumentatsiooni nõuetekohane vormistamine

Kokkuvõte

Viimase mitme aasta jooksul on muutunud tehisintellekti ja eelkõige ChatGPT kasutamine koodi kvaliteedi hindamiseks eriti sagedaseks, kuid esineb erinevaid arvamusi tema vastuste korrektsuse ja õigsuse suhtes [5], [6], [7].

Käesoleva bakalaureusetöö eesmärgiks oli hinnata, kuivõrd on ChatGPT genereeritavad vastused usaldusväärsed ja kooskõlalised, täpsemalt hinnata tema võimet parandada ja halvendada koodi, ning teha kindlaks, kas ChatGPT hinnangud on klassikaliste mõõdikutega vastavuses. Lisaks uurida, kas inimene saab koodi muutmisega paremini hakkama.

Eesmärkide saavutamiseks kasutasid autorid klassikaliste mõõdikute komplekti, AHP tööriista, ChatGPT vestlusrobotit, puhta koodi reeegleid ja oma teadmisi. Töö käigus oli valitud 14 koodinäidet halvendatud ja parendatud versioonide genereerimiseks ChatGPT poolt ning 6 koodinäidet autorite poolt versioonide genereerimiseks. Need koodide versioonid olid hinnatud mõõdikute ning ChatGPT abil ja saadud paarisvõrdluse hinnangud olid kasutatud kooskõlalisuse suhtarvude arvutamiseks.

Tulemusena prioritiseeris ChatGPT vestlusrobot klassikaliste mõõdikute komplektist kohesiooni mõõdikut kõige rohkem ja ABC mõõdikut kõige vähem. Ainult viie ChatGPT poolt genereeritud koodi puhul neljateistkümnest olid ChatGPT hinnangud kooskõlalised, inimgenereeritud koodide puhul olid aga ainult poole koodi kogumi ChatGPT hinnangud kooskõlalised. Klassikaliste mõõdikute hinnangud ChatGPT poolt genereeritud koodile olid peaaegu alati kooskõlalised, välja arvatud ühe koodi puhul, samas kui kõik hinnangud inimgenereeritud koodile olid kooskõlas.

Analüüsi käigus selgus, et suurimat keskmist mittekooskõlalisust näitasid ChatGPT hinnangud inimgenereeritud koodile, moodustades 21,68%. Väiksema keskmise kooskõlalisuse suhtarvuga olid mõõdikute hinnangud ChatGPT poolt genereeritud koodile ning mõõdikute hinnangud inimgenereeritlarmaud koodile, mis moodustasid vastavalt 3,34% ja 3,73%. Kõige õigema koodi versioonide järjestuse sai inimgenereeritud kood mõõdikute hinnangul ning kõige halvem koodi järjestus oli

ChatGPT hinnangud enda genereeritud koodile, kus keskmine esialgse koodi proriteet oli väiksem kui keskmine halvendatud versiooni prioriteet. Analüüsist järeldub, et ChatGPT tuleb toime koodi halvendamise ja hindamisega kehvemini, kui seda teeb inimene klassikaliseid mõõdikuid kasutades.

Kirjanduse loetelu

[1] L. Frunzio, B. Lin, M. Lanza ja G. Bavota, „RETICULA: Real-time code quality assessment,“ *IEEE 25th International Conference on Software Analysis*, Campobasso, Italy, 2018.

[2] M. I. Azeem, F. Palomba, L. Shi ja Q. Wang, „Machine learning techniques for code smell detection: A systematic literature review and meta-analysis,“ *Elsevier,* kd. 108, pp. 115-138, April 2019.

[3] J. Krstić ja M. Vukojičić, „ChatGPT in programming education: ChatGPT as a programming assistant,“ *InspirED Teachers` Voice,* 2023.

[4] T. L. Saaty, „Fundamentals of the Analytic Hierarchy Process,“ *The Analytic Hierarchy Process in Natural Resource and Environmental Decision Making,* kd. 3, Springer, Dordrecht, 2001.

[5] Vedraj, „ChatGPT: Top 10 Use Cases In Software Development,“ [Võrgumaterjal]. Available: https://www.valuecoders.com/blog/blockchain-ml/chatgpt-use-cases-in-software-development/. [Kasutatud 16 May 2024].

[6] I. Amaro, A. Della Greca, R. Francese, G. Tortora ja C. Tucci, „AI Unreliable Answers: A Case Study on ChatGPT,“ 2023.

[7] Botpress Community, „How Accurate Is ChatGPT In Providing Information Or Answers?,“ 1 May 2023. [Võrgumaterjal]. Available: https://botpress.com/blog/how-accurate-is-chatgpt-in-providing-information-or-answers. [Kasutatud 16 May 2024].

[8] K. D. Goepel, „BPMSG’s AHP Online System,“ 28 April 2022. [Võrgumaterjal]. Available: https://bpmsg.com/ahp/docs/BPMSG-AHP-OS.pdf. [Kasutatud 16 March 2024].

[9] K. D. Goepel, „Implementation of an Online Software Tool for the Analytic Hierarchy Process (AHP-OS),“ *International Journal of the Analytic Hierarchy Process,* kd. 10, 6 December 2018.

[10] T. Wu et al., „A Brief Overview of ChatGPT: The History, Status Quo and Potential Future Development,“ *IEEE/CAA Journal of Automatica Sinica,* kd. 10, May 2023.

[11] A. Vaswani et al., „Attention Is All You Need,“ Long Beach, CA, USA, 2017.

[12] A. Svyatkovskiy, S. K. Deng, S. Fu ja N. Sundaresan, „IntelliCode Compose: Code Generation using Transformer,“ *Proceedings of the 28th ACM Joint European Software Engineering Conference and,* New York, NY, USA, 2020.

[13] „ChatGPT,“ OpenAI, 2023. [Võrgumaterjal]. Available: https://chat.openai.com. [Kasutatud 4 March 2024].

[14] M. Agnihotri ja A. Chug, „A Systematic Literature Survey of Software Metrics, Code Smells and Refactoring Techniques,“ *Journal of Information Processing Systems,* kd. 16, pp. 915-934, 2020.

[15] J. Fitzpatrick, Applying the ABC Metric To C, C++, and Java, 1997.

[16] T. J. McCabe, „A Complexity Measure,“ *IEEE Transactions on Software Engineering,* pp. 308-320, December 1976.

[17] E. N. H. Kirğil ja T. E. Ayyildiz, „Analysis of Lack of Cohesion in Methods (LCOM): A Case Study,“ *2021 2nd International Informatics and Software Engineering Conference*, Ankara, Turkey, 2021.

[18] T. Sharma ja D. Spinellis, „Do We Need Improved Code Quality Metrics?,“ 2020.

[19] M. Lacchia, „Introduction to Code Metrics,“ [Võrgumaterjal]. Available: https://radon.readthedocs.io/en/latest/intro.html#maintainability-index. [Kasutatud 10 March 2024].

[20] B. A. Kitchenham, „Measures of programming complexity,“ *ICL TECHNICAL JOURNAL,* kd. 2, nr 3, 1981.

[21] T. Hariprasad, K. Seenu, G. Vidhyagaran ja C. Thirumalai, „Software Complexity Analysis Using Halstead Metrics,“ *2017 International Conference on Trends in Electronics and Informatics*, Tirunelveli, India, 2017.

[22] M. Lacchia, „Introduction to Code Metrics,“ [Võrgumaterjal]. Available: https://radon.readthedocs.io/en/latest/intro.html#halstead-metrics. [Kasutatud 10 March 2024].

[23] C. Clauss et al., „GitHub,“ [Võrgumaterjal]. Available: https://github.com/TheAlgorithms/Python. [Kasutatud 4 March 2024].

[24] Keon, „GitHub,“ [Võrgumaterjal]. Available: https://github.com/keon/algorithms/tree/master/algorithms. [Kasutatud 4 March 2024].

[25] D. Young, „GitHub,“ [Võrgumaterjal]. Available: https://github.com/drycode/zelle-python/tree/master. [Kasutatud 1 April 2024].

[26] R. C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, Pearson, 2008, p. 464.

[27] S. Senkiv ja A. Gontšarova, „GitHub,“ 2024. [Võrgumaterjal]. Available: https://github.com/AnastassijaG/bachelor_thesis_code_versions.git.

[28] M. Lacchia, „Welcome to Radon’s documentation!,“ 2012. [Võrgumaterjal]. Available: https://radon.readthedocs.io/en/latest/index.html. [Kasutatud 10 March 2024].

[29] E. Noble ja I. V. Kausel, „GitHub,“ [Võrgumaterjal]. Available: https://github.com/eoinnoble/python-abc. [Kasutatud 25 March 2024].

[30] Potfur, „Pypi,“ [Võrgumaterjal]. Available: https://pypi.org/project/lcom/. [Kasutatud 25 March 2024].

[31] A. Gontšarova ja S. Senkiv, „Bakalaureusetöö arvutused ja tulemused (AHP),“ 2024. [Võrgumaterjal]. Available: https://docs.google.com/spreadsheets/d/1OT1py3aX3j6DK-Bf-i6ZK1s8O1oiQ5IP1lrzH2w2ejM/edit?usp=sharing.

[32] R. C. Martin, Agile Software Development, Principles, Patterns, and Practices, Pearson Education Limited, 2014, pp. 95-98.

[33] R. S. Pressman, SOFTWARE ENGINEERING: A PRACTITIONER’S APPROACH, New York, NY: McGraw-Hill, 20010, pp. 613-642.

[34] T. Heričko ja B. Šumak, „Exploring Maintainability Index Variants for Software Maintainability Measurement in Object-Oriented Systems,“ *Applied Sciences,* kd. 13, nr 5, pp. 1-36, 2023.

[35] C. Bonja ja E. Kidanmariam, „Metrics for class cohesion and similarity between methods,“ *Proceedings of the 44th annual Southeast regional conference*, New York, NY, USA, 2006.

[36] R. M. Noorullah, „Grasp and GOF Patterns in Solving Design Problems,“ *International Journal of Engineering and Technology,* kd. 1, nr 3, pp. 196-205, December 2011.

[37] „AHP Software,“ [Võrgumaterjal]. Available: https://onlineoutput.com/ahp-software/. [Kasutatud 1 April 2024].

[38] „Analytic Hierarchy Process Software,“ [Võrgumaterjal]. Available: https://www.spicelogic.com/Products/ahp-software-30. [Kasutatud 1 April 2024].

[39] K. D. Goepel, „AHP Online System - AHP-OS,“ 2018. [Võrgumaterjal]. Available: https://bpmsg.com/ahp/. [Kasutatud 1 April 2024].

[40] Z. Liu, Y. Tang, X. Luo, Y. Zhou ja L. F. Zhang, „No Need to Lift a Finger Anymore? Assessing the Quality of Code Generation by ChatGPT,“ *IEEE Transactions on Software Engineering,* pp. 1-35, 23 April 2024.

[41] Y. Ji et al., „Exploring ChatGPT's Ability to Rank Content: A Preliminary Study on Consistency with Human Preferences,“ 2023.

[42] K. O'Shea ja R. Nash, „An Introduction to Convolutional Neural Networks,“ 2015.

[43] S. A. Zargar, „Introduction to Sequence Learning Models: RNN, LSTM, GRU,“ 2021.

[44] A. Dhillon ja G. K. Verma, „Convolutional neural network: a review of models, methodologies and applications to object detection.,“ *Springer Link,* p. 85–112, 20 December 2019.

[45] K. Lieberherr, I. Holland ja A. Riel, „Object-oriented programming: an objective sense of style,“ *ACM SIGPLAN Notices,* p. 323–334, 1988.

[46] „GPT-4 is OpenAI’s most advanced system, producing safer and more useful responses,“ OpenAI, 2024. [Võrgumaterjal]. Available: https://openai.com/index/gpt-4/. [Kasutatud 18 May 2024].

[47] „Discover the possibilities with Meta Llama,“ Meta, 2023. [Võrgumaterjal]. Available: https://llama.meta.com/. [Kasutatud 18.05 May 2024].

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Meie, Sofia Senkiv ja Anastassija Gontšarova

1. Anname Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Koodi kvaliteedi hindamine tehisintellekti ja teiste kvaliteedimõõdikute abil" , mille juhendaja on Ants Torim
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Oleme teadlikud, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autoritele.
3. Kinnitame, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

20.05.2024

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Moodulite abil arvutatud mõõdikute väärtuste näidistabel koodi erinevate versioonide jaoks

Koodi versioon **Mõõdik**	***GOOD***	***INITIAL***	***BAD***
Cyclomatic complexity	A (1.8461538461)	A(1.75)	A (2.3125)
Maintainability index	A (70.67)	A (48.92)	A (35.35)
ABC	<12, 25, 3> (27.9)	<16, 37, 3> (40.4)	<41,41,10> (58.8)
Cohesion	*Attendee*: 1 *ConferenceManager*: 1 *Average*: 1	*Attendee*: 1 *ConferenceManager*: 2 *Average*: 1.5	*Attendee*: 1 *ConferenceManager*: 2 *Average*: 1.5
Halstead • *vocabulary* • *length* • *calculated length* • *volume* • *difficulty* • *effort* • *time* • *bugs*	• 4 • 9 • 4,754887 • 18 • 1 • 18 • 1 • 0,006	• 5 • 6 • 8 • 13,93156 • 0,5 • 6,965784 • 0,38698 • 0,00464	• 21 • 45 • 79,81353 • 197,6542 • 2,5 • 494,1357 • 27,45198 • 0,065884

Lisa 3 – ChatGPT poolt genereeritud koodi mõõdikute prioriteedid ja kooskõlalisus

Näitaja **Koodi nimetus**	**Mõõdik**	***GOOD Priority***	***GOOD Rank***	***INITIAL Priority***	***INITIAL Rank***	***BAD Priority***	***BAD Rank***	**CR**
Dijkstra	*Cyclomatic Complexity*	40%	1	40%	1	20%	3	0%
	Maintainability index	32,7%	2	41,3%	1	26%	3	5,6%
	ABC	40%	1	40%	1	20%	3	0%
	Cohesion	33,3%	1	33,3%	1	33,3%	1	0%
	Halstead	25%	2	50%	1	25%	2	0%
Doubly_linked_list	*Cyclomatic Complexity*	33,3%	1	33,3%	1	33,3%	1	0%
	Maintainability index	25%	2	50%	1	25%	2	0%
	ABC	40%	1	40%	1	20%	3	0%
	Cohesion	33,3%	1	33,3%	1	33,3%	1	0%
	Halstead	40%	1	40%	1	20%	3	0%
Graph	*Cyclomatic Complexity*	41,3%	1	32,7%	2	26%	3	5,6%
	Maintainability index	33,3%	1	33,3%	1	33,3%	1	0%
	ABC	33,3%	1	33,3%	1	33,3%	1	0%
	Cohesion	33,3%	1	33,3%	1	33,3%	1	0%
	Halstead	57,1%	1	28,6%	2	14,3%	3	0%
Job_scheduling	*Cyclomatic Complexity*	40%	1	20%	3	40%	1	0%
	Maintainability index	25%	2	50%	1	25%	2	0%
	ABC	33,3%	1	33,3%	1	33,3%	1	0%

Näitaja **Koodi nimetus**	**Mõõdik**	***GOOD Priority***	***GOOD Rank***	***INITIAL Priority***	***INITIAL Rank***	***BAD Priority***	***BAD Rank***	**CR**
	Cohesion	33,3%	1	33,3%	1	33,3%	1	0%
	Halstead	33,3%	1	33,3%	1	33,3%	1	0%
Kosaraju	*Cyclomatic Complexity*	54%	1	29,7%	2	16,3%	3	1%
	Maintainability index	32,7%	2	41,3%	1	26%	3	5,6%
	ABC	40%	1	40%	1	20%	3	0%
	Cohesion	33,3%	1	33,3%	1	33,3%	1	0%
	Halstead	54%	1	29,7%	2	16,3%	3	1%
Matrix_class	*Cyclomatic Complexity*	50%	1	25%	2	25%	2	0%
	Maintainability index	49,3%	1	19,6%	3	31,1%	2	5,6%
	ABC	54%	1	29,7%	2	16,3%	3	1%
	Cohesion	33,3%	1	33,3%	1	33,3%	1	0%
	Halstead	60%	1	20%	2	20%	2	0%
Priority_queue	*Cyclomatic Complexity*	32,7%	2	41,3%	1	26%	3	5,6%
	Maintainability index	33,3%	1	33,3%	1	33,3%	1	0%
	ABC	25%	2	50%	1	25%	2	0%
	Cohesion	33,3%	1	33,3%	1	33,3%	1	0%
	Halstead	16,3%	3	54%	1	29,7%	2	1%
Singly_linked_list	*Cyclomatic Complexity*	40%	1	20%	3	40%	1	0%
	Maintainability index	40%	1	20%	3	40%	1	0%
	ABC	42,9%	1	14,3%	3	42,9%	1	0%
	Cohesion	33,3%	1	33,3%	1	33,3%	1	0%
	Halstead	47,4%	1	5,3%	3	47,4%	1	0%
Zelle_ex_3	*Cyclomatic Complexity*	40%	1	40%	1	20%	3	0%

Näitaja **Koodi nimetus**	**Mõõdik**	***GOOD Priority***	***GOOD Rank***	***INITIAL Priority***	***INITIAL Rank***	***BAD Priority***	***BAD Rank***	**CR**
	Maintainability index	25%	2	25%	2	50%	1	0%
	ABC	25%	2	25%	2	50%	1	0%
	Cohesion	45,5%	1	9,1%	3	45,5%	1	0%
	Halstead	14,3%	3	42,9%	1	42,9%	1	0%
Zelle_ex_5	*Cyclomatic Complexity*	33,3%	1	33,3%	1	33,3%	1	0%
	Maintainability index	40%	1	20%	3	40%	1	0%
	ABC	44,3%	1	16,9%	3	38,7%	2	1,9%
	Cohesion	44,4%	1	11,1%	3	44,4%	1	0%
	Halstead	54%	1	16,3%	3	29,7%	2	1%
Zelle_ex_2	*Cyclomatic Complexity*	20%	3	40%	1	40%	1	0%
	Maintainability index	20%	3	40%	1	40%	1	0%
	ABC	29,7%	2	16,3%	3	54%	1	1%
	Cohesion	66,5%	1	9%	3	24,5%	2	15,9%
	Halstead	20%	2	20%	2	60%	1	0%
Zelle_atm_manager	*Cyclomatic Complexity*	50%	1	25%	2	25%	2	0%
	Maintainability index	25%	2	50%	1	25%	2	0%
	ABC	33,3%	1	33,3%	1	33,3%	1	0%
	Cohesion	33,3%	1	33,3%	1	33,3%	1	0%
	Halstead	54%	1	29,7%	2	16,3%	3	1%
Zelle_Objrball	*Cyclomatic Complexity*	33,3%	1	33,3%	1	33,3%	1	0%
	Maintainability index	25%	2	50%	1	25%	2	0%
	ABC	33,3%	1	33,3%	1	33,3%	1	0%
	Cohesion	33,3%	1	33,3%	1	33,3%	1	0%
	Halstead	50%	1	25%	2	25%	2	0%

Näitaja **Koodi nimetus**	**Mõõdik**	***GOOD Priority***	***GOOD Rank***	***INITIAL Priority***	***INITIAL Rank***	***BAD Priority***	***BAD Rank***	**CR**
Zelle_chap13_ex11	*Cyclomatic Complexity*	33,3%	1	33,3%	1	33,3%	1	0%
	Maintainability index	33,3%	1	33,3%	1	33,3%	1	0%
	ABC	25%	2	25%	2	50%	1	0%
	Cohesion	33,3%	1	33,3%	1	33,3%	1	0%
	Halstead	20%	2	20%	2	60%	1	0%

Lisa 4 – Inimgenereeritud koodi mõõdikute prioriteedid ja kooskõlalisus

Näitaja **Koodi nimetus**	**Mõõdik**	***GOOD Priority***	***GOOD Rank***	***INITIAL Priority***	***INITIAL Rank***	***BAD Priority***	***BAD Rank***	**CR_{max}**
INIMGEN_zelleEx3	*Cyclomatic Complexity*	40%	1	40%	1	20%	3	0%
	Maintainability index	54%	1	29,7%	2	16,3%	3	1%
	ABC	49,3%	1	31,10%	2	19,60%	3	5,6%
	Cohesion	71,4%	1	14,3%	2	14,3%	2	0%
	Halstead	36,4%	2	57,8%	1	5,7%	3	5,6%
INIMGEN_zelleEx5	*Cyclomatic Complexity*	40%	1	40%	1	20%	3	0%
	Maintainability index	54%	1	29,7%	2	16,3%	3	1%
	ABC	50%	1	25%	2	25%	2	0%
	Cohesion	71,4%	1	14,3%	2	14,3%	2	0%
	Halstead	52,8%	1	33,3%	2	14,0%	3	5,6%
INIMGEN_zelleEx2	*Cyclomatic Complexity*	40%	1	40%	1	20%	3	0%
	Maintainability index	40%	1	40%	1	20%	3	0%
	ABC	25%	2	25%	2	50%	1	0%
	Cohesion	14,3%	2	14,3%	2	71,4%	1	0%
	Halstead	50%	1	25%	2	25%	2	0%
INIMGEN_zelleAtmManager	*Cyclomatic Complexity*	40%	1	40%	1	20%	3	0%
	Maintainability index	40%	1	40%	1	20%	3	0%

Näitaja **Koodi nimetus**	**Mõõdik**	***GOOD Priority***	***GOOD Rank***	***INITIAL Priority***	***INITIAL Rank***	***BAD Priority***	***BAD Rank***	**CR_{max}**
	ABC	40%	1	40%	1	20%	3	0%
	Cohesion	71,4%	1	14,3%	2	14,3%	2	0%
	Halstead	40%	1	40%	1	20%	3	0%
INIMGEN_zelleObjrball	*Cyclomatic Complexity*	41,3%	1	32,7%	2	26%	3	5,6%
	Maintainability index	40%	1	40%	1	20%	3	0%
	ABC	40%	1	40%	1	20%	3	0%
	Cohesion	66,7%	1	16,7%	2	16,7%	2	0%
	Halstead	50%	1	25%	2	25%	2	0%
INIMGEN_zelleChap13Ex11	*Cyclomatic Complexity*	49,3%	1	31,1%	2	19,6%	3	5,6%
	Maintainability index	49,3%	1	31,1%	2	19,6%	3	5,6%
	ABC	50%	1	25%	2	25%	2	0%
	Cohesion	75%	1	12,5%	2	12,5%	2	0%
	Halstead	81,8%	1	9,1%	2	9,1%	2	0%

Lisa 5 – Valitud puhta koodi reeglid klasside kontekstis

Reegli kategooria	**Reeglid/kommentaarid**
Klassi organisatsioon	• Klass peaks algama muutujate nimekirjaga. • Avalikud staatilised piirangud, kui need on olemas, peaksid olema esikohal. • Privaatsed staatilised muutujad, millele järgnevad eksemplarmuutujad. • Avalikud funktsioonid peaksid järgnema muutujate nimekirjale. • Avaliku funktsiooni poolt välja kutsutud privaatsed funktsioonid peaksid tulema kohe pärast avalikku funktsiooni ennast.
Klassid peaksid olema väikesed	• Klasside puhul mõõdetakse suurust vastutuste arvu põhjal • Klassi nimi peaks kirjeldama, milliseid ülesandeid see täidab. • Mida mitmetähenduslikum on klassi nimi, seda tõenäolisemalt on tal liiga palju vastutusi • *The Single Responsibility Principle*: klassidel peaks olema üks vastutus – üks põhjus muutmiseks
Kohesioon	• Klassidel peaks olema väike arv eksemplarmuutujaid. • Iga klassi meetod peaks käsitsema ühte või mitut neist muutujast. • Kui kohesioon on kõrge, tähendab see, et klassi meetodid ja muutujad sõltuvad üksteisest ja moodustavad loogilise terviku. • Muutujaid ja meetodeid tuleks eraldada kahte või enamasse klassi, et uued klassid oleksid ühtsemad.
Kohesiooni tulemuste säilitamine paljudes väikestes klassides	• Kui klassid kaotavad kohesiooni, jagatagu need osadeks • Suure funktsiooni jagamine paljudeks väiksemateks funktsioonideks annab meile sageli võimaluse jagada ka mitut väiksemat klassi. See annab programmile palju parema organiseerituse ja läbipaistvama struktuuri.

Reegli kategooria	**Reeglid/kommentaarid**
Muudatuste organiseerimine	• Puhtas süsteemis korraldatakse oma klass nii, et vähendada muudatuste riski • Ideaalses süsteemis lisatakse uusi lahendusi süsteemi laiendades, mitte muutes olemasolevat koodi.
Muudatustest isoleerumine	• Kasutusele saab võtta liidesed ja abstraktsed klassid, mis aitavad muudatuste mõju isoleerida. • Klassid peaksid sõltuma abstraktsioonidest, mitte konkreetsetest detailidest.

Lisa 6 – Valitud puhta koodi reeglid meetodite kontekstis

Reegli kategooria	**Reeglid/kommentaarid**
Üldised reeglid	• Funktsioonid peaksid olema väikesed • *If/else* ja *switch* laused peaksid olema väikesed • Ennast korrata ei tohi • Meetodid, mida ei kutsuta kunagi, tuleks kõrvale jätta.
Funktsioonid peaksid tegema ühte asja	• Kui funktsioon teeb ainult neid samme, mis on ühe taseme võrra madalamad funktsiooninimest, siis teeb funktsioon ühe asja. • Võimalus tunda, et funktsioon teeb rohkem kui ühte asja, on see, kui sellest saab eraldada teise funktsiooni, mille nimi ei ole lihtsalt selle implementatsiooni ümbersõnastamine.
Üks abstraktsioonitase funktsiooni kohta	• On vaja, et igale funktsioonile järgneksid järgmisel abstraktsioonitasemel olevad funktsioonid, nii et saaks programmi lugeda, langedes üks abstraktsioonitase korraga, kui loetakse funktsioonide nimekirja alla
Funktsiooni argumendid	• Funktsiooni ideaalne argumentide arv on null (*niladic*). Järgneb üks (*monadic*), millele järgneb kaks (*dyadic*). Võimaluse korral tuleks vältida kolme argumenti (*triadic*). • Kui tundub, et funktsioon vajab rohkem kui kahte või kolme argumenti, siis on tõenäoline, et mõned neist argumentidest tuleks koondada omaette klassi.
Käskude päringu eraldamine	• Funktsioonid peaksid kas midagi tegema või vastama, kuid mitte mõlemat.
Eelistage erandeid veakoodide tagastamisele	• Parem on välja tõmmata *try/catch* plokkide sisu eraldi funktsioonideks

Lisa 7 – Valitud puhta koodi reeglid nimede kontekstis

Reegli tüüp	**Reeglid/kommentaarid**
Üldised reeglid	• Kasutage kavatsust väljendavaid nimesid
Liikme eesliited	• Klassid ja funktsioonid peaksid olema piisavalt väikesed, et ei oleks vaja lisada liikmemuutujate eesliidet *m_*
Vältige vaimset kaardistamist	• Kindlasti võib tsükli loenduri nimi olla *i* või *j* või *k* (kuid mitte kunagi *l*), kui selle ulatus on väga väike ja muud nimed ei saa sellega konflikti sattuda.
Klassi nimed	• Klassidel ja objektidel peaks olema nimisõna või nimisõnafraas nimena nagu *Customer*, *WikiPage*, *Account* ja *AddressParser*. Vältige klassi nimes selliseid sõnu nagu *Manager*, *Professor*, *Data* või *Info*. Klassi nimi ei tohiks olla tegusõna.
Meetodi nimed	• Meetoditel peaks olema tegusõnad või tegusõnafraasid nimena, nagu *postPayment*, *deletePage* või *save*.
Vältige desinformatsiooni	• Ärge nimetage kontode rühmitust *accountList*'iks, kui see ei ole tegelikult *List*. Kui kontosid siseldav konteiner ei ole tegelikult *List*, võib see viia valede järeldusteni. Seega oleks parem *accountGroup* või *bunchOfAccounts* või lihtsalt *accounts*. • Kui te kodeerite hüpotenuusi ja „*hp*“ tundub hea lühendina, võib see olla desinformatiivne. • Ebajärjekindlate kirjaviiside kasutamine on desinformatsioon.
Tehke tähenduslikke eristusi	• Halb lähenemisviis: kuna te ei saa kasutada sama nime viitamaks kahele erinevale asjale samas ulatuses, võib tekkida soov muuta ühte nime suvalisel viisil.

Lisa 8 – Sofia Senkiv eneseanalüüs

Bakalaureusetööd teostasime kahekesi Anastassija Gontšarovaga, kus panustasime mõlemad töö etappidesse võrdselt. Meie mõlemad tegelesime järgmiste ülesannetega:

- Avalikust lähtekoodist erinevate koodide uurimine ja kogumine;
- Puhta koodi reeglite kogumine;
- Erinevate tarkvara mõõdikute uurimine, valimine ja nende jaoks Python moodulite leidmine (ABC, hooldatavuse indeks, tsüklomaatiline keerukus);
- Kogutud koodi põhjal parendatud ja halvendatud versioonide genereerimine ChatGPT abil (Dijkstra, Doubly_linked_list, Graph, Job_scheduling, Zelle_ex_2, Zelle_Objrball, Zelle_chap13_ex11);
- Esialgsete koodide iseseisev parendamine ja halvendamine (INIMGEN_zelleEx2, INIMGEN_zelleObjrball, INIMGEN_zelleChap13Ex11);
- ChatGPT ning minu poolt genereeritud versioonide mõõdikute väärtuste arvutamine, väljamõeldud teisendusreegli abil mõõdikute väärtuste teisendamine hinnanguteks;
- ChatGPT üldiste hinnangute saamine nii ChatGPT kui ka minu poolt loodud koodi versioonidele;
- Mõõdikute ja ChatGPT hinnangute ülekandmine AHP tööriista ning kooskõlalisuse tulemuste saamine;
- Dokumentatsioon.

Töö protsessi käigus minu jaoks oli kõige keerulisem see, et esialgseks teemaks valisime tehisintellektuaalse masina õpetamist koodi kvaliteedi hindama. Pärast mitut nädalat uurimist ja arutelu juhendajaga jõudsime järeldusele, et antud teema vajab rohkemat tausta ja kogemust, mille tõttu pidime teema vektorit muutma. Üheks väljakutseks oli veel

sobiva AHP tööriista leidmine, kuna paljud olid kas tasulised või analüütiliste hiararhiate protsessi läbiviimisel ei olnud seal piisavalt informatiivseid tulemusi kooskõlalisuse kohta. Mõõdikute jaoks Python moodulite otsimisel tekkis ka selline olukord, et mõned moodulid olid vananenud ning seega ei töötanud, mis tõttu kulus palju aega teiste töötavate moodulite otsimisele ning tööle panemisele.

Mina olen aga rahul enda panusega lõputöösse ning valitud meeskonnakaaslasega. Diplomitöö elluviimise protsess oli meie jaoks huvitav ning tänu sellele, et oleme Anastassijaga eelnevalt palju koostööd teinud, aitas see meid lõputöö tegemisel ning protsess läks sujuvalt.

Kokkuvõttes arvan, et lõputöö annab väärtust arendajatele ning inimestele, kes soovivad oma programmeerimisoskusi parendada, kasutades koodi kvaliteedi hindamiseks tehisintellekti abi. Antud töö aitab mõista ChatGPT võimet anda objektiivset hinnangut koodi kvaliteedile ning uurib, kas tasub ChatGPT vastuseid usaldada.

Lisa 9 – Anastassija Gontšarova eneseanalüüs

Diplomitööd tegime Sofia Senkiv'iga kahekesi koos, kus kõik lõputöö sammud olid teostatud ja jagatud meie mõlema poolt võrdselt. Nii mina kui ka Sofia oleme teinud järgmisi ülesandeid:

- Avalikust lähtekoodist erinevate koodide uurimine ja kogumine;
- Puhta koodi reeglite kogumine;
- Erinevate tarkvara mõõdikute uurimine, valimine ja nende jaoks Python moodulite leidmine (Halstead, LCOM4);
- Kogutud koodi põhjal parendatud ja halvendatud versioonide genereerimine ChatGPT abil (Kosaraju, Matrix_class, Priority_queue, Singly_linked_list, Zelle_ex_3, Zelle_ex_5, Zelle_atm_manager);
- Esialgsete koodide iseseisev parendamine ja halvendamine (INIMGEN_zelleEx3, INIMGEN_zelleEx5, INIMGEN_zelleAtmManager);
- ChatGPT ning minu poolt genereeritud versioonide mõõdikute väärtuste arvutamine, väljamõeldud teisendusreegli abil mõõdikute väärtuste teisendamine hinnanguteks;
- ChatGPT üldiste hinnangute saamine nii ChatGPT kui ka minu poolt loodud koodi versioonidele;
- Mõõdikute ja ChatGPT hinnangute ülekandmine AHP tööriista ning kooskõlalisuse tulemuste saamine;
- Dokumentatsioon.

Diplomitöö puhul oli kõige suuremaks väljakutseks see, et alguses pidi lõputöö teema olema veidi teine, kus ise õpetaksime tehisintellektuaalset masinat koodi kvaliteeti hindama. Antud teema tundus meile väga aktuaalne olevat, kuid lõpuks jõudsime

juhendajaga järeldusele, et taolise projekti elluviimiseks eeldab see põhjalikumat tausta ja kogemust. Teema vektori muutmine nõudis palju uurimist ja töö protsessi väljaselgitamist ühiste eesmärkide arusaamiseks. Lisaks tundus päris ajakulukas ka ChatGPT-ga suhtlemine ja tema käest nii koodi versioonide, mõõdikute prioritiseerimise kui ka koodile hinnangute saamine, kuna vestlusrobot ei andnud alati koheselt õigeid vastuseid vastavalt meie vajadustele, vaid võis genereerida kas mittetöötavat koodi, valesid mõõdikute priorititeete, mis ei vasta õigele 1–9 AHP skaalale vms. Samuti võttis aega ka töötavate Python moodulite leidmine mõõdikute arvutamise jaoks, kuna mõnede moodulite puhul esines raskusi nende tööle panemisega.

Üldiselt oli diplomitöö arendamise protsess minu jaoks väga huvitav. Nii mina kui minu meeskonnakaaslane panustamise sellesse tööse maksimaalselt ning lõpptulemusena olen enda panusega rahul. See polnud esimene kord, kus oleme Sofiaga ühes meeskonnaprojektis tegutsenud, seega temaga koostöö tegemine edenes tõrkedeta ja sujuvalt, seejuures alati leidsime kompromisse, motiveerisime teineteist ja püüdsime jõuda ühise selge arusaamisele erinevate aspektide puhul.

Lõppkokkuvõttes usun, et nii minu kui ka mu meeskonnaliikme tehtud töö ja analüüs annab märkimisväärset väärtust koodi kvaliteedile hinnangu saamise ja koodi muutmise lähenemisviiside osas ning arvan, et oleme püstitatud eesmärgiga edukalt hakkama saanud.