VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ BAKALAURO STUDIJŲ PROGRAMA

Kodo skirstymo į paketus šablonų tyrimas

Analysis of code packaging patterns

Bakalauro baigiamasis darbas

Atliko: Martyna Ubartaitė

Darbo vadovas: Gediminas Rimša

Darbo recenzentas: doc. dr. Vardauskas Pavardauskas

Vilnius – 2024

Santrauka

Glaustai aprašomas darbo turinys: pristatoma nagrinėta problema ir padarytos išvados. Santraukos apimtis ne didesnė nei 0,5 puslapio. Santraukų gale nurodomi darbo raktiniai žodžiai. Automatiškai naudojamos lietuviškos kabutės: „tekstas“.

Raktiniai žodžiai: raktinis žodis 1, raktinis žodis 2, raktinis žodis 3, raktinis žodis 4, raktinis žodis 5

Summary

Santrauka anglų kalba. Santraukos apimtis ne didesnė nei 0,5 puslapio. Automatiškai naudojamos angliškos kabutės: “tekstas”.

Keywords: keyword 1, keyword 2, keyword 3, keyword 4, keyword 5

Turinys

Įvadas

Įvade nurodomas darbo tikslas ir uždaviniai, kuriais bus įgyvendinamas tikslas, aprašomas temos aktualumas, apibrėžiamas tiriamasis objektas akcentuojant neapibrėžtumą, kuris bus išspręstas darbe, aptariamos teorinės darbo prielaidos bei metodika, apibūdinami su tema susiję literatūros ar kitokie šaltiniai, temos analizės tvarka, darbo atlikimo aplinkybės, pateikiama žinių apie naudojamus instrumentus (programas ir kt., jei darbe yra eksperimentinė dalis). Darbo įvadas neturi būti dėstymo santrauka. Įvado apimtis 2–4 puslapiai.

Įvadas

0.1. Problema ir jos aktualumas

Teisingai įgyvendintas kompiuterinės sistemos dizainas yra vienas iš kritinių sėkmingo verslo elementų. Tam, jog verslas išlaikytų stabilų augimą yra būtina sukurti sistemą, kuri sumažintų atotrūkį tarp organizacijos tikslų ir jų įgyvendinimo galimybių. Mąstant apie programinio kodo dizainą, kodo paketų kūrimas, klasių priskyrimas jiems ir paketų hierarchijos sudarymas paprastai nėra pagrindinis prioritetas, tačiau tai parodo praleistą galimybę padaryti sistemos dizainą labiau patikimu[Sho19], suprantamu [Eli10] ir lengviau palaikomu. Modernios sistemos yra didžiulės, programinis kodas yra padalintas į daugybę failų, kurie išskaidyti per skirtingo gylio direktorijas, todėl apgalvotai išskirstytas programinis kodas daro daug didesnę įtaką kodo kokybei, nei gali atrodyti iš pirmo žvilgsnio. Sistemos paketų studijavimas ir analizė norint įvertinti programinės įrangos kokybę tampa vis svarbesne tema, dėl pastoviai augančio failų ir paketų skaičiaus[Eli10].

Norint išsiaiškinti, kaip efektyviausiai gali būti skaidomas programinis kodas, tam jog jo struktūra darytų teigiama įtaką sistemos kokybei, reikalinga atlikti skirstymo į paketus šablonų analizę - išsiaiškinti galimus šablonus ir būdus, kaip skirstyti programinį kodą į paketus, turėti aiškius šablonų apibrėžimus su jų privalumai bei trūkumais.

0.2. Darbo tikslas

Šio darbo tikslas - identifikuoti šablonus kodo skirstymui į paketus. Remiantis moksliniais straipsniais apie sistemos kokybę bei palaikomumą aprašyti kriterijus, kurie būtų naudojami šablonus įvertinti, nustatant jų įtaką sistemos palaikomumui.

0.3. Keliami uždaviniai

- Išskirti gerai įgyvendinto kodo požymius
- Aprašyti skirstymo į paketus šablonus, remiantis pavyzdžiais teorinėje medžiagoje bei egzistuojančiose atviro kodo sistemose
- Įvertinti kiek realių sistemų struktūra nutolusi nuo teorinių šablonų apibrėžimų
- Pasiūlyti kriterijus, įvertinančius kodo suskirstymo šablono įtaką sistemos kokybei, remiantis rastai gerai įgyvenditos sistemos požymiais
- Pasirinkti kelias sistemas ir pertvarkyti jų failų struktūrą pagal aprašytus šablonus, įvertinant kiek sudėtinga pasiekti kiekvieno šablono strukūrą
- Naudojant pertvarkytas sistemas, įvertinti kiekvieną kodo skirstymo šabloną pagal pasiūlytus kriterijus
- Pateikti rekomendacijas, kokius šablonus kodo skirstymui tinkamiausia naudoti

todo: apubidinti šaltinius todo: aprašyti kas yra šablonai todo: pamineti praktine dalį

0.4. Numatomas darbo atlikimo procesas

- Remiantis teorine medžiaga aprašomi gerai įgyvendinto kodo požymiai, užtikrinantys sistemos stabilumą ir palaikomumą.
- Aprašomi kriterijai, kuriuos naudojant galima įvertinti kodo suskirstymo įtaką sistemos kodo kokybei, pavyzdžiui:
  - Komponentų skaičius, priklausomas nuo pasirinkto komponento[Mah03]
  - Tiesioginės ir netiesioginės priklausomybės (matomumas)[Ala07]
- Galimų kodo skirstymo šablonų išskyrimas pasitelkiant teorinę informaciją [Sho11] ir egzistuojančių sistemų architektūrą.
- Atviro kodo projektų pasirinkimas. Pasirenkami skirtingo tipo projektai, užtikrinant objektyvesnę šablonų analizę skirtingose srityse. Galimi tipai:
  - Taikomoji programinė įranga, teikianti paslaugas įrangos naudotojams. Pavyzdžiui, internetinė programėlė priminimams ir darbams užsirašyti
  - Techninė programinė įranga, naudojama taikomosios programinės įrangos duomenų saugojimui, siuntimui, paieškai. Pavyzdžiui, duomenų bazės, pranešimų eilės, talpyklos (angl. cache)
  - Programinės įrangos įrankiai, skirti naudoti kitose sistemose supaprastinant programinį kodą, naudojant jau įgyvendintas funkcijas. Pavyzdžiui, Java programavimo kalbos Spring karkasas internetinių programėlių kūrimui
- Pasirinktų projektų paketų sturktūros pertvarkymas pagal pasirinktus skirstymo šablonus
- Pertvarkytų projektų įvertinimas, naudojant išskirtus kriterijus, nustatant, kokią įtaką skirtingi skirstymo šablonai turi sistemos kokybei

0.5. Laukiami rezultatai

- Įdentifikuoti kodo skirstymo šablonai, remiantis teorine informacija ir praktikoje sutinkamais pavyzdžiais
- Sukurti kriterijai, įvertinantys sistemos paketų struktūros indėlį sistemos kokybei
- Pasirinkti projektai pertvarkyti pagal kodo skirstymo šablonus
- Įvertinus šablonus sukurtais kriterijais, pateiktos rekomendacijos kodo skirstymo šablonų naudojimui

0.6. todo:

aprašyti sekančius chapterius.

1. Kompiuterinės sistemos vertinimas

1.1. Teisingai įgyvendinta kompiuterinė sistema

Norint išsiaiškinti, kokią įtaką sistemos kokybei daro skirtingos paketų skirstymo metodologijos ir kaip ojektyviai tą įtaką pamatuoti, pirmiausia reikėtų apsibrėžti kokiais požymiais pasižymi teisingai įgyvendinta kompiuterinė sistema. Martin Kleppmann savo knygoje *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems* išskyria šiuos pagrindinius kriterijus:

- Patikimumas, reiškiantis, kad net ir klaidų (įrangos, programinių ar žmogiškųjų) atveju, sistema veikia stabiliai ir patikimai, paslepiant tam tikras klaidas nuo vartotojo[Kle17].
- Prižiūrimumas, reiškiantis jog skirtingų abstrakcijų pagalba sumažintas sistemos kompleksiškumas. Dėl to nesunku keisti esamą sistemos funkcionalumą bei pritaikyti naujiems verslo naudojimo atvejams. Tai supaprastina darbą inžinierių ir operacijų komandoms dirbančioms su šia sistema, taip pat leidžia prie sistemos prisidėti naujiems žmonėms, o ne tik jos ekspertams. Tai ypač aktualu atviro kodo sistemoms[Kle17].
- Plečiamumas, reiškiantis jog sistema turi strategijas, kaip išlaikyti gerą našumą užklausų srautui didėjant ir sistemai augant, tai atliekant su pagrįstais kompiuteriniais resursais ir priežiūros kaina[Kle17].

Yra daug skirtingų elementų, sudarančių sistemą, kuri tenkintų aukščiau paminėtus kriterijus, pavyzdžiui, pasirinktos technologijos, aukšto lygio architektūra, dokumentacija, sistemos testavimo procesai, jų kiekis ir pan. Vienas iš svarbių elementų, prisidedančių prie gerai įgyvendintos sistemos dizaino yra programinio kodo dizainas, jo skaitomumas, patikimumas. Konvencijos, kaip vadinti kodo paketus, kokias klasės jiems priskirti ir kokios paketų hierarchijos laikytis sudaro svarbią programinio kodo dizaino dalį, Todėl programinės įrangos kurimo metu, laikas skirtas rasti sistemai tinkamą paketų skirstymo šabloną ir to šablono laikymasis atsiperka, padarant programinį kodą geriau suprantamu, taip prisidėdant prie bendro sistemos dizaino patikimumo ir lengvesnio palaikomumo.

Straipsnyje *Investigating The Effect of Software Packaging on Modular Structure Stability*, jo autoriai akcentuoja, kad gerai įgyvendintos, objektiškai orientuotos sistemos turėtų vystytis be didelių pakeitimu jų architektūroje. To siekiama todėl, nes architektūriniai pakeitimai paveikią didelę sistemos dalį ir todėl jų įgyvendinimo ir priežiūros kaštai yra ženkliai didesni[Sho19]. Paketų struktūra, kuri užtikrina atsieta (angl. *decoupled*) komunikavima tarp paketų, enkapsuliuoja paketų vidinius elementus, neleidžiant pakeitimais išplisti už paketų ribų yra pagrindas tvirtai sistemos architektūrai, gebančiai efektyviai plėstis, ženkliai nesikeičiant ir sutaupant programos priežiūros kaštus.

1.2. Kodo skirstymo paketais metodų vertinimas

Ankstesniame skyriuje buvo nagrinėjama gerai įgyvendintos paketų struktūros įtaka geram kompiuterinės sistemos dizainui, tačiau lieka neatsakytas klausimas - kaip įvertinti metodą kodui į

paketus grupuoti, kaip objektyviai užtikrinti jog šis sprendimas yra butent toks kokio reikia ir kokia būtent jo įtaka kompiuterinei sistemai? Tvarkingas, aiškiai suprantamas kodas yra subjektyvi tema, priklausanti nuo komandos, naudojamos programavimo kalbos ar programinių įrankių bei programinės sistemos dalykinės srities. Kodo grupavimo į paketus metodai, taip, kaip ir bendros tvarkingo kodo praktikos, gali būti labai subjektyvūs ir patogus tik metodą formavusiam asmeniui. Tam, kad pagrįstai įvertinti skirtingus kodo skirstymo šablonus, pasiekiant kuo objektyvesnį, plačiau priimtiną rezultatą, reikėtų aprašyti kriterijus, ko tikimasi iš paketų struktūros.

Robert C. Martin savo knygoje *Agile Software Development, Principles, Patterns, and Practices* aprašo principus padedančius teisingai grupuoti klases į paketus. Rodiklis, kiek kiekvienas paketas sistemoje laikosi nurodytų principų, tam tikrame paketų skirstymo šablone, gali būti kriterijus, įvertinti to šablono kokybei ir įtakai bendram sistemos dizainui.

todo: prideti DSM analize, jei truksta

1.2.0.1. Bendro sąryšio principas

Klasės pakete turėtų būti susietos kartu, kad turėtų tą pačią priežastį pasikeisti. Pakeitimas kuris paveikia paketą, paveikia visas to paketo klasėms ir jokiems kitiems paketams.

Kaip vienos atsakomybės principas (angl. *Single responsibility principle*) sako, kad klasė turėtų neturėti skirtingų priežasčių keistis, šis principas teigia, kad paketas taip pat neturėtų turėti skirtingų priežasčių pasikeisti. Principas ragina suburti visas klases, kurios gali keistis dėl tų pačių priežasčių, į vieną vietą. Jei dvi klasės yra taip stipriai susietos, kad jos visada keičiasi kartu, tada jos turėtų būti toje pačioje pakuotėje. Kai reikia išleisti pakeitimus, geriau, kad visi pakeitimai būtų viename pakete. Tai sumažina darbo krūvį, susijusį su pakeitimu išleidimu, pakartotiniu patvirtinimu ir programinės įrangos perskirstymu, be reikalo nevaliduojant ir leidžiant kitų, nesusijusių modulių[Mar02].

1.2.1. Aciklinių priklausomybių principas

Paketo priklausomybės diagramoje neturi būti žiedinių ciklų.

Priklausomybių ciklai sukuria neatidėliotinų problemų. Žiedinės priklausomybės gali sukelti domino efektą, kai nedidelis lokalus vieno modulio pokytis išplinta į kitus modulius, dėl to norit patestuoti viena nedidėli modulį, reikia iš naujo sukompiliuoti didžiulę sistemos dalį. Taip pat žiedinės priklausomybės lemia programos ir kompiliavimo klaidas, kadangi pasidaro labai sunku sudaryti tvarką, kaip kompiliuoti paketus. Iš tiesų, gali nebūti teisingos tvarkos, nes žiedinės priklausomybės gali sukelti begalinę rekursiją, kuri sukelia nemalonių problemų tokioms kalboms kaip Java, kurios skaito savo deklaracijas iš sukompiliuotų dvejetainių failų[Mar02].

1.2.2. Stabilių priklausomybių principas

Paketų priklausomybės turetų laikytis stabilumo krypties Sistema negali būti visiškai statiška. Norint jai plėstis būtinas tam tikras nepastovumas. Kaikurie paketai yra sukurti taip, kad būtų nepastovus, iš jų tikimasi pokyčių. Nuo paketo, kuris yra manoma nepastovus, neturėtų priklausyti sunkiai pakeičiami paketai, nes priešingu atveju nepastovų paketą taip pat bus sunku pakeisti. Tai

yra programinės įrangos niuansas, kad modulis, sukurtas taip, kad jį būtų lengva pakeisti, tampa sunkiai keičiamu, kažkam kitam pridėjus priklausomybė nuo jo[Mar02].

1.2.3. Stabilių abstrakcijų principas

Paketas turi būti tiek abstraktus, kiek stabilus Šis principas nustato ryšį tarp stabilumo ir abstraktumo. Stabilus paketai turetų būti abstraktus, todėl ir lengvai praplečiami. Tai pat šis principas teigia, kad nestabilus pakuotė turi būti konkreti, nes jos nestabilumas leidžia lengvai pakeisti jos turinio kodą. Taigi, jei paketas yra stabilus, jį taip pat turėtų sudaryti abstrakčios klasės, užtikrinant jo išplėčiamumą. Stabilus paketai, kuria yra lengvai išplečiami yra lankstūs pakeitimams, nedarant didelės įtakos sitemos dizainui[Mar02].

1.2.3.1. Abstrakcijos matavimas

Paketo abstrakciją galima pamatuoti santykių tarp abstrakčių klasių (arba sąsajų (angl. *interface*)) pakete ir bendro klasių skaičiaus:

$$Abstrakcija = \frac{N_{abstrakcios}}{N_{visos}} \tag{1}$$

Abstrakcijos reikšmė gali būti tarp 0 ir 1. Nulis reiškia kad paketas neturi jokių abstrakčių klasiu, o vienetas nurodo, kad pakete yra tik abstrakčios klasės.

Beveik visi autoriaus aprašyti principai turi aiškiai apibrėžtas metrikas, kuriomis galima pamatuoti, kaip stipriai paketas laikosi šių principų. Būtent šios metrikos bus naudojamos įvertinti paketus, analizuojamuose šablonuose, kad suprasti šablonų kokybę.

Paketų kokybės metrikos:

- *Klasių skaičius* - klasių skaičiaus metrika paketui nurodo, skaičių klasių (konkrečių ir abstrakčių) pakete. Ši metrika matuoja paketo dydį.
- *Aferentinės jungtys (angl. Afferent Couplings)* - aferentinių jungčių metrika nurodo skaičių kitų paketų, kurie priklauso nuo klasių esančių pasirinktame pakete. Ši metrika matuoja ateinančias priklausomybes.
- *Eferentinės jungtys (angl. Efferent Couplings)* - eferentinių jungčių metrika nurodo skaičių kitų paketų, nuo kuriu priklauso klasės pasirintame pakete. Ši metrika matuoja išeinančias priklausomybės.
- *Nestabilumas* - nestabilumo metrika nurodo santyki tarp eferentinių jungčių ir visų jungčių (Aferentinės + Eferentinės) pakete. Ši metrika matuoja paketo atsparumui pokyčiams, apie kurį buvo akcentuojama stabilių priklausomybių principė. Reikšmės rėžiai - nuo nulio iki vieno, kur vienas nurodo visiškai stabilų paketą, o vienetas visiškai nestabilų.
- *Atstumas* - Atstumo metrika apibrėžiamas kaip statmenas pakuotės atstumas nuo idealizuotos linijos (A + I = 1), kur A yra abstrakčių klasių procentas nuo bendro paketo klasių skaičiaus. Ši metrika yra paketo abstraktumo ir stabilumo pusiausvyros rodiklis. Paketas tiesiai pagrindinėje sekoje yra optimaliai subalansuotas, atsižvelgiant į jos abstraktumą ir stabilumą. Šios metrikos diapazonas yra nuo nulio iki vieneto, o nulis nurodo paketą,

kuris sutampa su pagrindine seka, o vienas – paketą, kuris yra kuo toliau nuo pagrindinės sekos. todo: pataisyti

- *Žiedinės priklausomybės* - žiedinių priklausomybių metrika skaičiuoja atvejus, kur pasirinkto paketo išeinančios priklausomybes taip pat yra paketo ateinančios priklausomybes (tiesiogiai arba netiesiogiai). Ši metrika aciklinių priklausomybių rodiklis, minėtas aciklinių priklausomybių principe.

todo: argumentuoti metriku pagristuma

2. Galimi kodo skirstymo į paketus šablonai

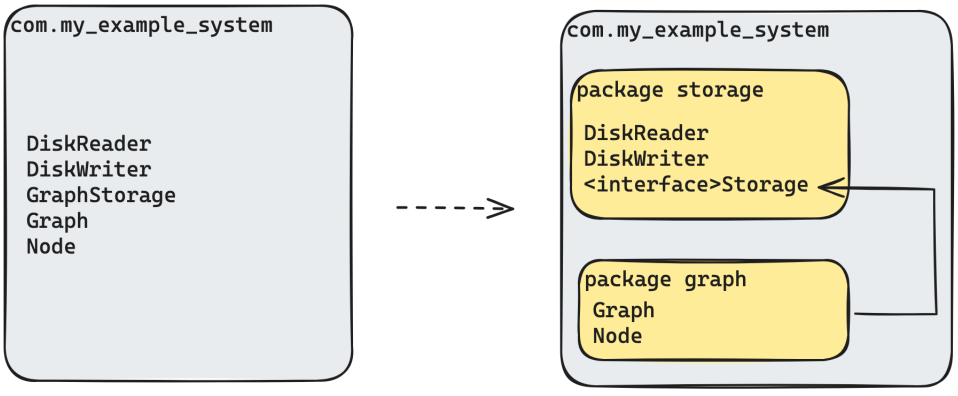
Diskusijose, kaip reikėtų skirstyti programini kodą paprastai akcentuojami du šablonai - pagal *techninį sluoksnį*, kur kiekvienam funkcionalumui arba kompiuterinės sistemos sluoksniui yra sukuriamas paketas, grupuojant skirtingas dalykinės srities esybes, arba pagal *dalykinės srities esybes*, kur vienos esybės kodas, dalykinės srities esybės funkcionalumas, skirtingose programiniuose sluoksniuose yra patalpintas viename pakete. Tačiau šie du šablonai yra gan platūs ir galėtų būti išskaidyti į daugiau smulkesnių ir tiksliau aprašytų šablonų. Taip pat, minėtuose šablonuose, būdai kaip ir kodėl skaidyti programinį kodą parinkti akecentuojant tai, kaip programinį kodą supranta žmones, dirbantys prie to kodo. Nuspresti, kaip žmones supranta programinį kodą yra gan sudėtingas ir subjektyvus procesas, todėl aprašant šablonus, kodo skirstymui geriau akcentuoti, kaip sugrupuoti paketai bendrauja tarpusavyje ir skirstyti juos pagal klasių naudojimo atvejus ir priklausomybes. Taip kodo grupavimo metodai yra labiau artimi Martino aprašytiems principams.

Šablonus kaip grupuoti kodą, akcentuojant klasių naudojimo atvejus ir priklausomybes nagrinėja Martin Sandin savo straipsnyje *Four Strategies for Organizing Code*. Šis straipsnis idomus tuo, kad autorius nesiplečia į du dažniausiai sutiknamus šablonus - grupuoti pagal techninį sluoksni arba dalykinės srities esybes, o aprašo keturis grupavimo būdus arba šablonus, kurie nors ir įkvepti minėtų dviejų budų, yra gan unikalus ir labiau techniškai apibrėžti.

2.1. Pagal komponentą

Organizavimas pagal komponentus sumažina sistemos sudėtingumą, pabrėždamas išorinę ir vidinę kodo vienetų darną. Išorinė darna reiškia, kad paketas turi minimalią sąsają (angl. *()*interface), kuri atskleidžia tik konceptus (metodus arba duomenų tipus), kurios yra glaudžiai susijusios su komponento teikiama paslauga. Vidinė darna reiškia, kad pakuotėje esantis kodas yra stipriai susijęs tarpusavyje ir susijęs su teikiama paslauga.

Kodas yra grupuojamus į mažus paketus, turinčius vieną, aiškiai apibrėžtą funkcionalumą ar tikslą, aprašant abstrakciją, kokie paketo elementai yra pasiekiami iš išorės ir kaip jie naudojami. Taip sukuriamas kodas, kuris yra lengviau suprantamas. Toks kodo grupavimo šablonas nėra lengvai plaikomas tačiau jo rezultatas - kodas kuris yra lengviau suprantamas, lengviau pagerinamas, lengviau testuojamas ir dėl aiškiai aprašytų sąsajų lengviau pernaudojamas.



1 pav. Sistemos sugrupuotos pagal komponentą pavizdys

2.2. Pagal techninį sluoksnį

2.3. Pagal tipą

3. Įrankiai šablonų analizei ir įvertinimui

4. Kodo skirstymo metodai realiose sistemose

4.1. Sistemų pasirinkimas

4.2. Sistemų analizės procesas

5. Sistemų pertvarkymas pagal šablonus

6. Medžiagos darbo tema dėstymo skyriai

Medžiagos darbo tema dėstymo skyriuose išsamiai pateikiamos nagrinėjamos temos detalės: pradiniai duomenys, jų analizės ir apdorojimo metodai, sprendimų įgyvendinimas, gautų rezultatų apibendrinimas.

Medžiaga turi būti dėstoma aiškiai, pateikiant argumentus. Tekste dėstomas trečiuoju asmeniu, t.y. rašoma ne „aš manau“, bet „autorius mano“, „autoriaus nuomone“. Reikėtų vengti informacijos nesuteikiančių frazių, pvz., „...kaip jau buvo minėta...“, „...kaip visiems žinoma...“ ir pan., vengti grožinės literatūros ar publicistinio stiliaus, gausių metaforų ar panašių meninės išraiškos priemonių.

Skyriai gali turėti poskyrius ir smulkesnes sudėtines dalis, kaip punktus ir papunkčius.

6.1. Poskyris

Citavimo pavyzdžiai: cituojamas vienas šaltinis [**PvzStraipsnLt**]; cituojami keli šaltiniai [**PvzStraipsnEn**; **PvzStraipsnLta**; **PvzKonfLt**; **PvzKonfEn**; **PvzKnygLt**; **PvzKnygEn**; **PvzElPubLt**; **PvzElPubEn**; **PvzBakLt**; **PvzMagistrLt**; **PvzPhdEn**].

Anglų kalbos terminų pateikimo pavyzdžiai: priklausomybių injekcija (angl. *dependency injection*, dažnai trumpinama kaip *DI*), saitų redaktorius (angl. *linker*).

Išnašų[1] pavyzdžiai[2].

6.2. Faktorialo algoritmas

1 algoritmas parodo, kaip suskaičiuoti skaičiaus faktorialą.

1 algoritmas. Skaičiaus faktorialas

1: $N \leftarrow$ skaičius, kurio faktorialą skaičiuojame
2: $F \leftarrow 1$
3: **for** $i := 2$ to N **do**
4: $\quad F \leftarrow F \cdot i$
5: **end for**

6.2.1. Punktas

6.2.1.1. Papunktis

6.2.2. Punktas

[1] Pirma išnaša.

[2] Antra išnaša.

7. Skyrius

7.1. Poskyris

7.2. Poskyris

Rezultatai

Rezultatų skyriuje išdėstomi pagrindiniai darbo rezultatai: kažkas išanalizuota, kažkas sukurta, kažkas įdiegta. Tarpinių žingsnių išdavos skirtos užtikrinti galutinio rezultato kokybę neturi būti pateikiami šiame skyriuje. Kalbant informatikos termi- nais, šiame skyriuje pateikiama darbo išvestis, kuri gali būti įvestimi kituose panašios tematikos darbuose. Rezultatai pateikiami sunumeruotų (gali būti hierarchiniai) sąrašų pavidalu. Darbo rezultatai turi atitikti darbo tikslą.

Išvados

1. Išvadų skyriuje daromi nagrinėtų problemų sprendimo metodų palyginimai, siūlomos rekomendacijos, akcentuojamos naujovės.
2. Išvados pateikiamos sunumeruoto (gali būti hierarchinis) sąrašo pavidalu.
3. Darbo išvados turi atitikti darbo tikslą.

Šaltiniai

[Eli10] M. Elish. Exploring the Relationships between Design Metrics and Package Understandability: A Case Study. 2010 [žiūrėta 2024-03-04]. Prieiga per internetą: https://www.researchgate.net/publication/221219583_Exploring_the_Relationships_between_Design_Metrics_and_Package_Understandability_A_Case_Study.

[Kle17] M. Kleppmann. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media, 2017.

[Mar02] R. C. Martin. *Agile Software Development, Principles, Patterns, and Practices*. Pearson, 2002.

[Sho19] M. A. Shouki A. Ebad. Investigating The Effect of Software Packaging on Modular Structure Stability. 2019 [žiūrėta 2024-03-04]. Prieiga per internetą: https://d1wqtxts1xzle7.cloudfront.net/75578419/pdf-libre.pdf?1638471828=&response-content-disposition=inline%3B+filename%3DInvestigating_the_Effect_of_Software_Pac.pdf&Expires=1709553794&Signature=BWU2DSSXDNujfvTllUWYspcqNlbW1Fgg~doSlc6JoeK7XXJ5bGLPB1B1yBD0tnojJ0yNuWZzQP9fpTjd~yffOhlxnM4GyB2gNMuGZyXsDBXWQuD66kZpwWdluJ63GWjvs28T2ArRpaekqk9JAc-Icun18nyonYJ~W4pIViibjHA4k9yN5r1FZRSWFeUSHpRmflEpJ1opD2Nh889TquLxsA2DhPP3L5_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA.

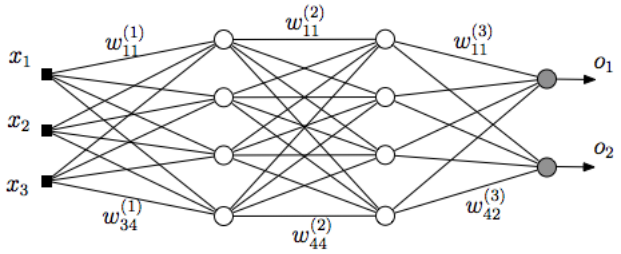
Santrumpos

Sąvokų apibrėžimai ir santrumpų sąrašas sudaromas tada, kai darbo tekste vartojami specialūs paaiškinimo reikalaujantys terminai ir rečiau sutinkamos santrumpos.

Priedai

Priedas nr. 1

Neuroninio tinklo struktūra



2 pav. Paveikslėlio pavyzdys

Priedas nr. 2

Eksperimentinio palyginimo rezultatai

1 lentelė. Lentelės pavyzdys

Algoritmas	$\bar{x}$	σ^2
Algoritmas A	1.6335	0.5584
Algoritmas B	1.7395	0.5647