KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Mantvydas Dziakavičius

MOBILIOSIOS PROGRAMOS MIGRAVIMAS IŠ VIENOS APLINKOS Į KITĄ

Baigiamasis magistro darbas

Vadovas
lekt. dr. Š. Packevičius

KAUNAS, 2017

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

MOBILIOSIOS PROGRAMOS MIGRAVIMAS IŠ VIENOS APLINKOS Į KITĄ

Baigiamasis magistro darbas
Programų sistemų inžinerija (621E16001)

Vadovas
(parašas) doc. dr. Šarūnas Packevičius
(data)

Recenzentas
(parašas) Prof. dr. Vacius Jusas
(data)

Projektą atliko
(parašas) Mantvydas Dziakavičius
(data)

KAUNAS, 2017

KAUNO TECHNOLOGIJOS UNIVERSITETAS

Informatikos fakultetas

(Fakultetas)

Mantvydas Dziakavičius

(Studento vardas, pavardė)

Programų sistemų inžinerija (621E16001)

(Studijų programos pavadinimas, kodas)

Baigiamojo projekto „Mobiliosios programos migravimas iš vienos aplinkos į kitą“

AKADEMINIO SĄŽININGUMO DEKLARACIJA

20 17 m. gegužės 19 d.

Kaunas

Patvirtinu, kad mano, **Mantvydo Dziakavičiaus**, baigiamasis projektas tema „Mobiliosios programos migravimas iš vienos aplinkos į kitą“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

(vardą ir pavardę įrašyti ranka) *(parašas)*

Turinys

Lentelių sąrašas

Paveikslėlių sąrašas

Dziakavičius, M. Mobiliosios programos migravimas iš vienos aplinkos į kitą. Programų sistemų inžinerijos magistro baigiamasis projektas / vadovas lekt. dr. Š. Packevičius; Kauno technologijos universitetas, Informatikos fakultetas.

Kaunas, 2017. 54 p.

SUMMARY

At a present time, one of the most developing area of technology is software for mobile phones, tablets and multimedia devices. Growing numbers of users and improved hardware makes more and more companies to expand their activity in this market. There is some specificity of platforms, therefore developer needs a more knowledge or experts of mobile application developing where require a more resources, training, costs and it takes a time. One of the possible solutions to the problem, to make the tools which allow design and create mobile applications independent by platform keep the logic in design and development or testing phase.

In this research paper representing the tools developed to carry out transformations from Android source code to iOS source code. To prove the importance of transformations performed research of transformation tools with certain comparison of metrics between the programs of separated implementation.

1. ĮŽANGA

1.1. Dokumento paskirtis

Šio dokumento paskirtis yra pateikti visą informaciją susijusią su kurtos sistemos - mobiliosios programos migravimas iš vienos aplinkos į kitą prototipo realizacija, atliktais tyrimais ir jų rezultatais. Dokumentas sudarytas iš skyrių, kuriuose pateikiama atlikta mokslinių tyrimų, algoritmų ir įrankių analizė. Projektinėje dalyje pateikiama informacija apie sistemos realizacija: naudotus įrankius, realizuotus algoritmus. Eksperimentinėje dalyje aprašomas vykdytas tyrimas, sistemos plėtojimo perspektyvos ir kitos įžvalgos.

1.2. Darbo tikslai

Pagrindinis darbo tikslas yra sukurti sistemą, kuri pasiūlytų patogų ir laiką taupantį būdą konvertuoti sistemų išeities kodą skirtą Android sistemai į iOS aplinkai tinkantį kodą. Reikalinga sukurti priemones, arba pritaikyti esamas technologijas, kurios esamą mobilios programos kodą (arba modelį) konvertuotų į kitos mobilios sistemos programą. Priemonių veikimas turi pasižymėti:

- modelio sudarymu,
- logikos aprašymu modelio elementuose,
- transformacijų taisyklių užrašymas,
- modelio transformavimu į kodą,
- esamo kodo transformavimu į modelį.

1.3. Santrauka

Šiuo metu labiausiai plėtojama technologijų grupė yra programų sistemos mobiliesiems ir multimedijos įrenginiams bei planšetiniams kompiuteriams. Nuolat augantys šių įrenginių vartotojų skaičiai ir techninės įrangos gerinimas skatina vis daugiau naujų kompanijų plėtimasi šios veiklos rinkoje. Tačiau, šiose platformose programuotojams reikia daug žinių ar mobiliųjų programų kūrėjų ekspertų pagalbos, kurie reikalauja daugiau resursų, patirties, užima daugiau laiko ir kainuoja brangiau. Vienas iš galimų šios problemos sprendimo būdų yra įrankiai, kurie leidžia kurti ir projektuoti mobiliasias programas nepriklausomai nuo to, kokioje platformoje ji bus naudojama, kuriami tam, kad programavimo ir testavimo fazėse būtų palaikomas vienodas loginis išdėstymas.

Šiame darbe yra aprašomi įrankiai, kurie yra sukurti padėti transformuoti „Android“ programinį kodą į „iOS“. Kad būtų įrodyta programos perkėlimo svarba, buvo

atliktas perkėlimo įrankių tyrimas, lyginant tam tikrus įrankius, atsižvelgiant į skirtingus programos veiksmų atlikimus/vykdymus, šie įrankiai perkėlė programas naudojantis universaliais įrankiais.

2. ANALITINĖ DALIS

Šioje dalyje apžvelgsime sukurtas technologijas, kurių pagrindas yra programinės įrangos transformavimas iš vienos platformos į kitą. Esminis faktas yra tas, kad mobiliosios programos negali būti skaitomos nešiojamuoju ar staliniu kompiuteriu, nes joms reikalingos operacinės sistemos turi būti optimizuotos lietimui jautriu ekranu. Ieškomi metodai turėtų būti pritaikyti ribotų išteklių, grafinės sąsajos skirtos lietimui jautriems ekranams programų kūrimui ir transformavimui.

2.1. Kodo konvertavimas

Norint, kad programa veiktų dideliu našumu, ypač apdorojant vaizdą, reikia optimizuoti programinį kodą, pritaikant jį atitinkamai aparatinei įrangai. Kodo konvertavimas, tai išeities kodo, kuris yra pritaikytas vienai įrangai, perrašymas, kad būtų skirtas kitai įrangai.

Iš vienos aplinkos kodo konvertavimo į kitą kompiliatoriai arba interpretatoriai yra tokie kompiliatoriai, kurie paima išeities kodą parašytą tam tikra programavimo kalba ir sukuria ekvivalentinį išeities kodą parašytą kita programavimo kalba. Tai tokie kompiliatoriai kurie konvertavimą atlieka išlikdami to paties lygio programavimo kalbos tipui, kai tuo tarpu įprastiniai kompiliatoriai dažniausiai kodą konvertuoja iš aukštesniojo lygio į žemesnįjį. Pavyzdžiui kompiliatorius gali atlikti konvertavimą iš Pascal kalbos į C kalbą [1].

Kita transformavimo iš vienos aplinkos į kitą reikšmė yra perkelti užsilikusį kodą iš senesnių versijų į kitas API ar programavimo kalbas, kurios panaikina atbulinį palaikymą (backward compatibility). Jos atliks automatinį kodo suskaldymą, kuris yra naudingas skaidant programas, priklausančias kitam vykdytojui (pavyzdžiui, konvertuojant programas iš Python 2 į Python 3, ar perkeliant programas iš senojo API į naująjį API), ar kai programa yra per didelė, ir būna nepraktiška suskaldyti ją rankiniu būdu.

Kompiliatorius išverstą kodą gali palaikyti kiek įmanoma nepakitusį ir artimą originaliajam kodui, kad būtų palengvintas originalaus kodo klaidų taisymas, arba gali pakeisti originalaus kodo struktūrą taip stipriai, kad išverstas kodas gali būti visiškai nepanašus į originalą. Taip pat yra ir kitų klaidų taisymo programų įrankių, kurie sulygina išimtą kodo dalį su originalia kodo dalimi. [2]

2.2. Vyraujančios tendencijos rinkoje

2.2.1. JUniversal

JUniversal yra kompiliatorius, verčiantis Java kodą į C# programos kalbą, tuo pačiu metu išsaugant kodo formatą ir Javadoc dokumentaciją. Šis įrankis šiuo metu yra tobulinamas, kad verstų Java į tikslinę C++. „Micfosoft Open Technologies“ grupė siūlo šį nemokamą įrankį naudoti „Android“ programų konvertavimui, kad jis galėtų veikti „Windows Phone“ mobiliuosiuose telefonuose, o vėliau ir „iOS“.[3]

„JUniversal“ taikosi į verslo logikos kodą, o ne į mobiliųjų programų vartotojų sąsajas. Vartotojo sąsaja turi būti parašyta gimtoją kalba, atskirai nuo visų kitų platformų.

Tačiau yra vienas svarbus apribojimas: ne visos Java bibliotekos gali būti naudojamos. Pasak Breto Johnsono, „Microsoft“ programuotojo, „Java į C# kodo vertėjas/kompiliatorius verčia tik kelias pagrindines Java klases, tokias kaip String ir StringBuilder. Likusioms dažnai naudojamoms JDK dalims lieka naudoti įprastas JSimple bibliotekas“. JSimple yra biblioteka, kurioje yra:

- Pagrindiniai programavimo kalbos konstruktoriai: AutoCloseable, Comparable, Iterable, Math;
- Failų įvedimas: Directory, File, Path, Reader, StringReader, Writer, ir kt.
- JSON
- Prisijungimo bibliotekos: Logger, LoggerFactory ir kt.
- Internetinės užklausos: HttpRequest, Socket, Url ir kt.
- OAuth
- Vienetiniai testai: UnitTest, UnitTestBase
- Įrankiai: Array, Collection, DateTime, HashMap, Iterator, List ir kiti.

JSimple yra ištraukta iš Pache Harmony ir yra šiek tiek patobulinta, taip pat sumažinta rečiau naudojamų funkcijų. Pagrindinis tikslas yra įsitikinti, kad kodas tinkamai veiks Java programavimo kalboje, kaip ir kitose tikslinėse kalbose. JSimple remia ir kai kurias naujas Java funkcijas, tokias kaip Lambdos

2.2.2. J2ObjC

J2ObjC yra atviro kodo komandinės eilutės „Google“ įrankis, kuris verčia Java išeities kodą į Objektinę C kalbą „iOS“ („iPhone/iPad“) platformą. Šis įrankis leidžia Java išeities kodui būti „iOS“ programos dalimi, kadangi nėra reikalingas joks generuoto failo keitimas ar

taisymas. Tikslas yra parašyti ne programos grafinės sąsajos kodą (tokį, kaip programos logika ir duomenų modelis) Java kalboje, kuris vėliau yra naudojamas internetinių, Android ar iOS programų.[4]

J2ObjC remia didelę dalį Java programavimo kalbos ir vykdymo savybių. Į tai įeina išimtys, vidinės ir anoniminės klasės, bendrieji tipai, temos ir refleksijos. Junit testo vertimas ir vykdymas taip pat yra remiami.

J2ObjC neteikia jokio laisvos platformos UI įrankio, nėra jokių planų to daryti ir ateityje. Kūrėjų nuomone iOS UI kodas turi būti rašomas Objektyvioje C, Objektyvioje C++ ar Swift programavimo kalbomis, naudojant Apple iOS SDK („Android“ grafinė sąsaja turėtų būti rašomi naudojantis „Android“ API, internetinės programos grafinės sąsajos parašytos naudojant GWT ir t. t.).

2.3. Problemos iškylančios kuriant kodo konverterį

Transformuojant išeities kodą iš vienos programavimo kalbos į kitą, naudojami bendri abstraktūs konvertavimo modeliai. Kad transformacija vyktų sėkmingai turi būti aprašytos transformavimo taisyklės, kurias naudojant ir vyktų transformavimas. Tai reiškia, kad jei būtų susiduriama su išeities kodu, kuris neturi aprašytu transformavimo taisyklių, tuomet konvertavimas neįvyktu, jei sistema susideda iš daugybės tokių komponentų, tuomet išeities kodo transformavimo procentas lieka labai žemas. [6]

Taip pat transformuotas kodas gali nutolti nuo realios užduoties. Jeigu konvertuotas kodas tampa per didelis arba nebeefektyvus ir lėtas, tuomet prireikia programuotojo įsikišimo ištaisyti sudarytas kompiliatoriaus klaidas, ko pasėkoje išnaudojama daugiau resursų ir prarandama šio kuriamo įrankio prasmė.

Naudojant kodo konvertavimo ir modeliavimo algoritmus, kodo skaitomumas mažėja. Norint išlaikyti kuo aukštesnį skaitomą, reikia pasirūpinti, kad transformavimo taisyklės būtų tinkamai paruoštos ir naudojamos.

2.4. Techniniai aspektai

Šiame skyriuje pateikiama informacija apie techninius kodo konverterio kūrimo aspektus. Remiantis kitų mokslinių darbų medžiaga surinkta ir pateikiama apžvalga apie algoritmų ir duomenų, reikalingų kodo generatoriaus prototipo kūrimui, surinkimą bei apdorojimą.

2.4.1. Kodo transformavimo taisyklės

Kodo konvertavimo iš vienos programavimo kalbos į kitą veikimas įgyvendinamas pasitelkiant kodo transformavimo taisyklėmis. Šios taisyklės aprašo susietumą tarp dviejų programavimo kalbų ekvivalenčių kintamųjų, klasių ir metodų tipų. Šių tipų užrašymas ir net kartais veikimas skirtingose kalbose skiriasi. Tačiau šie skirtumai nėra dinaminiai, tai reiškia, kad galima surinkti biblioteką tokių taisyklių ir naudoti atitinkamas, kuomet jų prireikią. [7]

1 lentelė Kintamųjų tipų skirtumai tarp kalbų

Java	**Objektinė C**
boolean	BOOL
byte	char
char	unichar
double	double
float	float
int	int

2.4.2. Išeities kodo paruošimas

Kuriant kodo konvertavimo sistemas vienas iš pirminių darbų yra tinkamai paruošti pradinį išeities kodą. Pradinį kodą reikia perrašyti ir pritaikyti nespecializuotai aplinkos architektūrai. Pateikiami pagrindiniai išeities kodo apdorojimo metodai [8]:

1. Naudoti modeliais pagrįsta architektūros modelį. Šiame modelyje išeities kodas yra suskirstomas į atskirus modelius, taip palengvinamas išeities kodo konvertavimas, kuomet galima transformuoti programavimo kodą iš vienos kalbos į kitą dalimis.
2. Kodo kintamieji yra paruošiami konvertavimui.
3. Kodo funkcijos yra paruošiamos konvertavimui.
4. Surenkamos ir paruošiamos kodo transformavimo taisyklės, pagal išeities kodo turimas klases, funkcijas, bei kintamuosius.

2.5. Algoritmų analizė

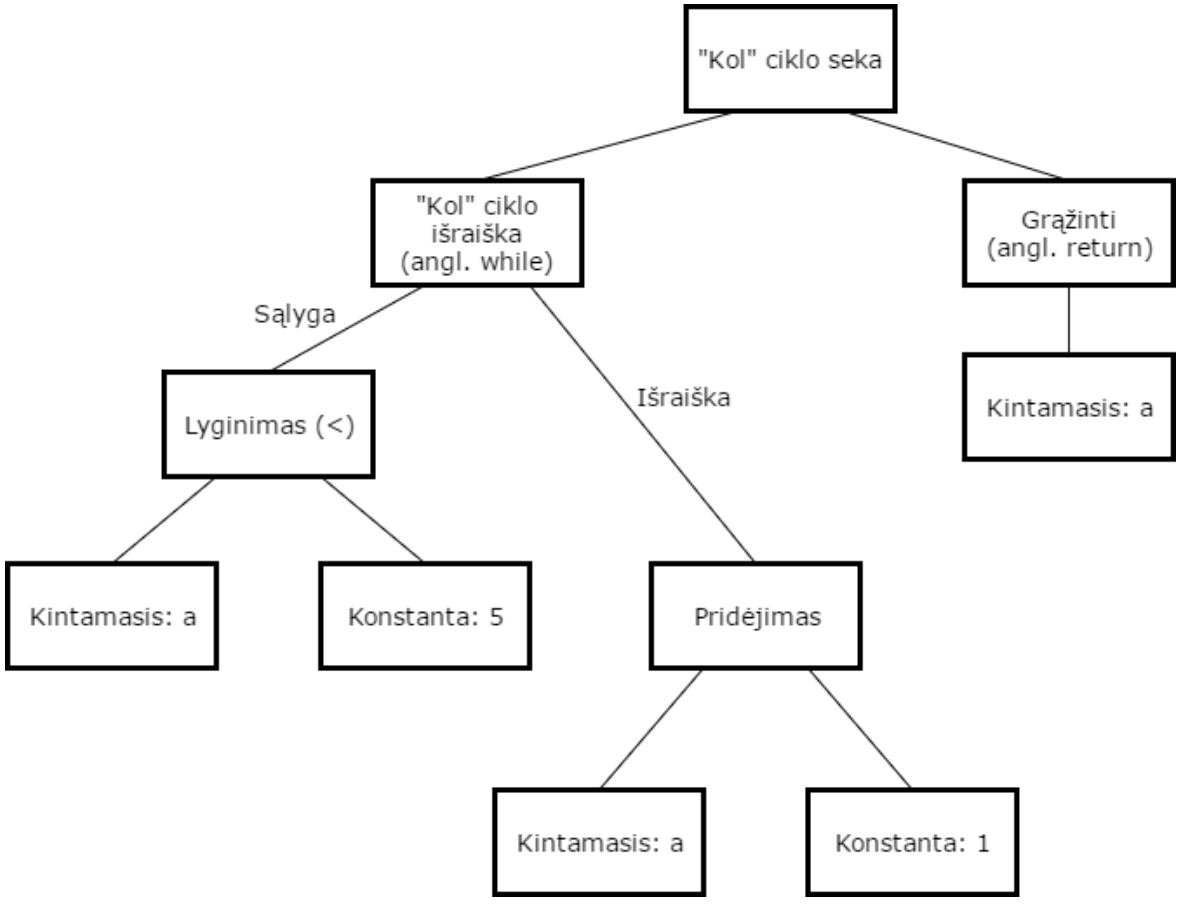
Svarbiausios mobiliųjų programų konvertavimo dalys iš „Android“ aplinkos į „iOS“ būtų šios:

- komponentas, atsakingas už tipų konvertavimą į ekvivalenčius.
- komponentas, kuris atsakingas už klasių konvertavimą ir normalizavimą.
- komponentas, kuris pasirūpina, kad būtų išlaikomas sąryšis tarp klasių

Tolesniuose skyriuose apžvelgiami algoritmai ir įrankiai, kurie padeda ir galbūt jau yra naudojami kodo konverteriuose.

2.5.1. Abstrakčios sintaksės medis (AST)

Abstrakčios sintaksės medis (angl. Abstract Syntax Tree - AST) yra medžio tipo grafas atspindintis išeities kodo abstrakčios sintaksės struktūrą. Kiekvienas mazginis taškas atspindi teiginį programavimo kalbos išeities kode. Sintaksė yra abstrakti todėl, kad neatvaizduoja kiekvienos detalės tikrosios sintaksės, pavyzdžiui lenktinių skliaustų grupavimas nėra atvaizduojamas grafe. Jei-sąlyga-tada išraiškos gali būti būti atvaizduojamas kaip vienas mazginis taškas su trimis šakomis. Tai atskiria abstrakčios sintaksės medį nuo konkrečios sintaksės medžio, kuris yra naudojamas išeities kodo nagrinėjimui, vertimui ir kompiliavimui į žemesnio lygio programavimo kalbą [9].



1 pav. Pavyzdinė AST grafo struktūra

AST dažnai naudojamas programinio kodo analizavimui ir transformavimui. Pagrindinės AST medžio taisyklės:

- Kintamųjų tipai ir vieta turi likti nepakitę.
- Vykdomoji tvarka turi būti aiškiai nurodyta ir tiksliai apibrėžta.
- Kairieji ir dešinieji dvejetainių operacijų komponentai turi būti teisingai nustatyti ir atvaizduoti.
- Identifikatoriai ir jų priskirtos reikšmės turi būti saugomos priskyrimo ataskaitų.

2.5.2. Modeliais pagrįsta architektūra (MDA)

Modeliais pagrįsta architektūra (angl. Model-Driven Architecture) – modeliavimo procese sukurtų modelių transformavimas į programos kodą nepriklausomai nuo technologijų architektūros. Taikant modelių transformacija ir modeliavimą į skirtingas architektūras šis modelis skatina efektyvų programinės įrangos kūrimo procesą ir palaiko geriausias reinžinerijos praktikas. MDA leidžia pakartotinai naudoti esamus išteklius naujuose galimybėse, kuriuose įgyvendinamos verslo funkcionavimas laike, net ir kaip tikslius infrastruktūros vystymas. [10]

Pagrindinis MDA tikslas yra atskirti dizainą nuo architektūros. Koncepcijos ir technologijos, skirtos suvokti dizainą ir koncepcijos ir technologijos, naudojamos suvokti architektūrą pasikeitė bėgant laikui, atsiejant juos nuo vienas kito leidžia sistemos kūrėjams pasirinkti iš geriausių ir labiausiai tinkamų stiprybių iš abiejų sričių. Dizainas adresuoja funkcinius (angl. Use Case) reikalavimus, o architektūra suteikia infrastruktūrą, per kurią nefunkciniai reikalavimai, pavyzdžiui, mastelis, patikimumas ir efektyvumas yra realizuojami. MDA numato, kad nepriklausomas platformos modelis (angl. platform independent model - PIM), kuris atstovauja koncepcinį dizainą įvykdant funkcinius reikalavimus, išgyvens pokyčius kurie atsiranda realizuojant technologijų ir programinės įrangos architektūras.

2.5.3. SSA algoritmas

Kompiliatorių dizaine, statinė vienos užduoties forma (SSA angl. Static single assignment form) yra tarpinio atstovavimo (IR angl. Intermediate representation) savybė,

kurioje reikalaujama, kad kiekvienas kintamasis yra priskirtas tik vieną kartą, ir kiekvienas kintamojo rodiklis apibrėžiamas dar prieš naudojimą.

2 lentelė Įprastinis kintamųjų naudojimas

1: a = x + y 2: a = a + 3 3: b = x + y

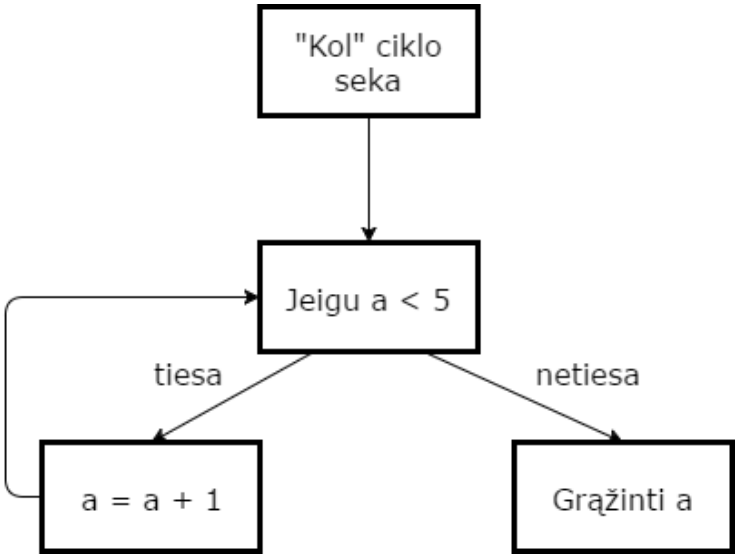
3 lentelė Kintamųjų naudojimas pagal SSA modulį

1: $a_1 = x_0 + y_0$ 2: $a_2 = a_1 + 3$ 3: $b_1 = x_0 + y_0$

Egzistuojantys kintamieji išeities kode yra padalinami į versijas, naujus kintamuosius paprastai turinčius originalius pavadinimus su pridėtiniu indeksu, todėl, kad kiekvienas apibrėžimas gauna savo versiją [11].

2.5.4. Abstrakčios semantikos grafas (ASG)

Abstrakčios semantikos grafas (ASG angl. Abstract semantic graph) yra abstrakčios sintaksės formą, kurioje programavimo kalbos išraiškos yra atvaizduojamos grafe, kurio viršūnės atvaizduoja išraiškų sąlygos.



2 pav. Pavyzdinė ASG grafo struktūra

ASG turi aukštesnio abstraktumo lygį nei abstrakčios sintaksės medis (AST). ASG yra sudėtingesni ir glaustesni nei AST, nes jie turi bendrai naudojamų mazginių taškų. Semantiniai grafikai dažnai naudojami kaip tarpinis grafas prieš AST, kuomet sudėtingesni mazginiai taškai nėra suskaidyti į paprastesnius. AST yra medžio tipo grafikai, todėl jie negali atvaizduoti bendrai naudojamas išraiškas [12].

2.5.5. Pertvarkymas

Pertvarkymas (angl. refactoring) yra esamo kompiuterio kodo restruktūrizavimas nekeičiant programos veiklos proceso. Pertvarkymas pagerina ne funkcinius programinės įrangos atributus. Kodo pertvarkymo privalumai yra patobulintas kodo aiškumas ir sumažintas sudėtingumas, tai gali pagerinti išeities kodo palaikymą ir sukurti plačiau išraiškingą vidaus architektūros ar objekto modelį, skirtą gerinti plečiamumą. Paprastai pertvarkymas susideda iš standartizuotų bazinių mažesnių pertvarkymų (angl. micro-refactoring), kurių kiekvienas yra (dažniausiai) nedidelis programinio kodo pakeitimas, kad arba išsaugo programinės įrangos elgesį, arba bent jau ne keistu atitikimų funkcinių reikalavimų. [13] Daugelis kodavimo aplinkų teikia automatizuotą pagalbinį kodo pertvarkymą atliekant pagrindinius mechaninius pertvarkymo aspektus. Jei atliekamas teisingai, pertvarkymas, gali išspręsti paslėptas arba neatrastas sistemos klaidas ar pažeidžiamumus supaprastinant logiką ir pašalinant nereikalingus sudėtingumo lygius. Jei atliekama neteisingai, sistema gali nebeatitikti reikalavimų, kai nebeatlieka reikiamų funkcijų arba įdiegti naujų klaidų.

Nuolat gerinant kodo dizainą, mes lengviname savo darbą. Nedidelis pertvarkymas ir didelis dėmesio skyrimas priveda prie naujų funkcijų. Atliekant nuolatinį pertvarkymą, pastebima, kad vis lengviau yra išplėsti ir prižiūrėti programinį kodą [14].

Štai keletas pertvarkymo tipų: Kai kurie šie pertvarkymai gali tikti tik tam tikros programavimo kalboms. Pilnesnis sąrašas šių tipų gali būti rastas Fowlerio pertvarkymo knygoje [15] arba internetinėje svetainėje [16].

- Tipai kurie suteikia kodui didesnį abstraktumą:
    - Pakeisti kintamųjų pasiekiamumą tik per gavimo ir nustatymo (angl. getter and setter) metodus.
    - Tipų apibendrinimas – Apibendrinami tipai daugkartinio kodo naudojimui.
    - Sąlygų pakeitimas polimorfizmais

- Kodo išskaidymas į mažesnes dalis
    - Kodo išskaidymas į daugkartinio naudojimo semantinius vienetus, kurie aiškiai ir paprastai apibrėžia sąsajas.
    - Klasės kodo dalies perkėlimas į naują klasę.
    - Metodo skaldymas iš vieno metodo į kelis mažesnius. Skaidant metodus į kelis mažesnius yra didinamas kodo įskaitomumas (angl. code readability). Tai taip pat galioja ir funkcijoms.
- Metodai skirti vardų ir vietos gerinimui
    - Metodo ar kintamojo perkėlimas iš vienos klasės į kitą
    - Metodo ar kintamojo pervadinimas – tinkamai pakeičiant metodo ar kintamojo vardą, galima lengviau nustatyti, kam jis naudojamas
    - Metodo ar kintamojo perkėlimas į tėvinę klasę
    - Metodo ar kintamojo perkėlimas į vaikinę klasę

2.5.6. Algoritmų palyginimas

Atlikus patiktų algoritmų analizę galima išskirti pagrindinius algoritmus, kurie galimi naudoti kodo konvertavimui: „AST“ ir „ASG“. Algoritmų palyginimas:

- Didėjant kodo eilučių kiekiui „Abstrakčios sintaksės medžio“ efektyvumas mažėja ir veikimo laikas didėja.
- Didėjant duomenų kiekiui tikslumui išsaugoti tinkamesnis yra abstrakčios semantikos grafikai.
- „ASG“ algoritmo veikimo laikas taip pat didėja didėjant duomenų imčiai, tačiau lyginant su „AST“ algoritmu, efektyvumas išlieka aukštesnis.
- Abstrakčios sintaksės medžiai negali atvaizduoti bendras programavimo išraiškas, dėl grafo struktūros. Šis medžio paprastumas kainuoja efektyvumo sąnaudas, kadangi reikalingas atskiras besikartojančių sąlygų atvaizdavimas atskirais mazginiais taškais. Dėl šios priežasties pirma kuriamas tarpinis ASG grafas, kad palengvinti vėlesnį AST grafo generavimą.

Kiti nagrinėti algoritmai gali būti taip pat naudojami išeities kodo paruošimui prieš pritaikant kodo transformavimo taisykles.

2.6. Įrankių analizė

Apžvelgiami įrankiai, skirti padėti kurti konverterius, kuriuose naudojami ir realizuoti aukščiau paminėti algoritmai. Įrankiai išrinkti pagal didžiausia tinkamumą projektui.

2.6.1. Įrankis „ROSE“

„ROSE” yra kompiliavimo karkasas, skirtas padėti sukurti išeities kodo analizavimui ir transformavimui iš vienos kalbos į kitą skirtus įrankius. „ROSE“ buvo sukurtas „Lawrence Livermore” nacionalinėje laboratorijoje[5]. Šis įrankis siekia suteikti vartotojams kompiliavimo metodų bibliotekas. Jis taip pat suteikia vartotojui prieigą prie išeities kodo analizės įrankių, leidžiančius vartotojui kurti savo kompiliatorius, analizatorius, vertėjus, bei pirminio apdorojimo procesus. „ROSE” suteikia turiningą rinkinį įrankių skirtų padėti vartotojams kurti jų pačių įrankiams analizei, konvertavimui ir specializuotoms transformacijoms [5].

„ROSE” veikimo principas susideda iš išeities kodo skaitymo ir Abstrakčios sintaksės medžio (AST) generavimo. AST suformuoją grafą atspindintį išeities kodą ir yra laikomas atmintyje, taip pasirūpinant greitu pasiekiamumu. „ROSE“ parūpina įrankius skirtus modifikuoti AST. Tuomet iš AST galimas kodo generavimas.

Kūrėjų svarbiausi papildomi įrankiai:

- OpenMP vertimas,
- Masyvų optimizavimas,
- Ciklų analizavimas,
- Kodo sudėtingumo analizatorius,
- Įvesties ir išeities duomenų analizavimas,
- Kodo padengimo analizavimo įrankiai.

2.6.2. Įrankis „LLVM“

Įrankis LLVM - Žemo Lygio Virtuali Mašina (angl. „Low Level Virtual Machine“) yra modelinių kompiliatorių įrankių rinkinys. Nors iš pavadinimo galima manyti, kad įrankis skirtas tradicinėms virtualioms mašinoms, tačiau taip nėra, nors ir yra integruotų bibliotekų skirtų joms kurti. „LLVM“ prasidėjo 2000 metais, kaip tyrimo projektas Ilinojaus universitete, su tikslu sukurti SSA tipo pagrįstu kompiliavimo strategija, kuri padėtu su programavimo kalbų kompiliavimu statiniu ir dinaminiu būdų [17].

„LLVM“ privalumai:

- „LLVM“ naudoja paprastą žemo lygio programavimo kalbą su griežtai nustatytomis semantikomis.
- Palaikomos kalbos: C, C++ ir JAVA.
- Palaikomi kompiliavimo modeliai: sąsajos proceso (angl. link-time), diegimo proceso (angl. install-time), vykdymo proceso (angl. run-time).
- Įrankis turi pilną palaikymą šiukšlių surinkėjui (angl. garbage collector).
- „LLVM“ kodo generatorius yra lengvai konfigūruojamas.
- Įrankis turi plačią ir aiškią dokumentaciją.
- „LLVM“ yra aktyviai tobulinamas ir nuolat plečiamas ir tvarkomas.

3. SIŪLOMAS SPRENDIMAS

Pagal atliktą analizę siūlomi tokie sistemos realizacijos sprendimai:

- Pradiniam išeities kodo paruošimui naudojama ASG grafų technologija.
- Tikslesnei kodo analizei, pasitelkiama AST grafų technologija.
- Pagal sudarytus grafus išeities kodas yra transformuojamas pagal nustatytas transformavimo taisykles.

Transformavimo taisyklių saugojimui ir naudojimui pasitelkiamas „ROSE“ įrankis. Šis įrankis pasirinktas, nes jis yra plačiai naudojamas kituose tokio pobūdžio tyrimuose. Taip su šiuo įrankiu pateikiama informatyvi naudojimosi instrukcija ir juo yra paprastą naudotis. Tam, kad kodo konvertavimas vyktu greičiau ir tiksliau siūlomi tokie kodo tvarkymo metodai:

- Pradinis išeities kodas suskirstomas į modulius.
- Moduliai kurių sudėtingumo koeficientas yra didelis, skirstomi į mažesnius modulius.
- Moduliai, kurių neįmanoma suskirstyti į mažesnius yra netransformuojami ir ši informacija pateikiama vartotojui.

4. PROJEKTINĖ DALIS

4.1. Sistemos pagrindinis funkcionalumas ir veikimo principas

Pagrindinės sistemos funkcijos yra konvertuoti mobiliųjų programų išeities kodus iš „Android“ aplinkos į „iOS“ kiek įmanoma geriau išlaikant įskaitomumą.

Sukurtas kodo konvertavimo prototipas pagrįstas tokiu veikimu: pirminio išeities kodui yra atliekas pertvarkymas (angl. *refactoring*), tuomet yra konvertuojamas iš vienos programavimo kalbos į kitą ir išsaugomi naujai sukurti failai. Sistemoje informacijos apdorojimo ir atvaizdavimo funkcionalumas vykdomas lygiagrečiai.

4.2. Reikalavimų analizė

Sistemos kūrimo pradžioje buvo atlikta reikalavimų analizė, kurioje buvo išskirtos pagrindinės sistemos savybės:

- Gebėti atlikti programos kodo pertvarkymą, kad paruošti kodą konvertavimui.
- Konvertuoti kodą iš Java kalbos į objektinę C.

Konvertavimo metu gali pasitaikyti tokių kodo dalių, kurių nėra įmanoma konvertuoti, dėl išeities kodo dydžio ar sudėtingumo arba naudojamų bibliotekų realizavimo. Sistema turėtu aptikti tokias vietas ir atsisakius jas konvertuoti, nurodyti jas vartotojui. Užtiktos klaidos atliekant kodo konvertavimą, turėtu būti nurodytos vartotojui, pagal išeities kodą (pavyzdžiui: turi būti nurodytas kodo eilutės numeris).

Kodas gautas po konvertavimo turėtu būti kuo lengviau įskaitomas vartotojų, jei būtų norima koreguoti ar pertvarkyti išeities kodą.

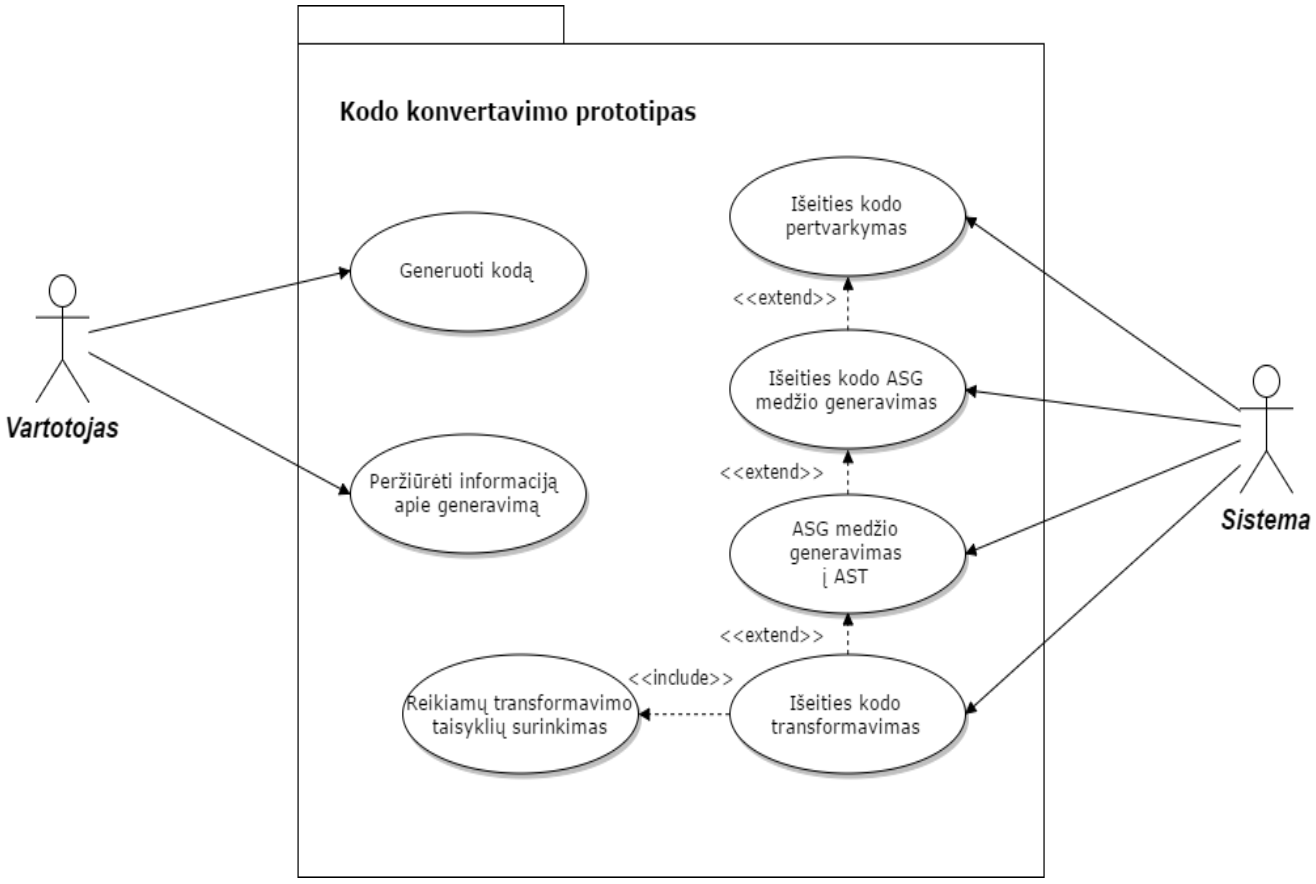
4.2.1. Nefunkciniai reikalavimai

Sistemos nefunkciniai reikalavimai :

- Konvertavimas turi būti greitas.
- Konvertuotas išeities kodas turi užimti kiek įmanomą mažiau vietos.
- Konvertuotas išeities kodas turi būti efektyvus.
- Komandinės eilutės sąsajos palaikymas.
- Išeities kodo komentarai, turėtu išlikti galutiniame kode.

4.3. Panaudos atvejų diagrama

Atsižvelgiant į prieš tai išskirtus reikalavimus buvo paruošta panaudos atvejų diagrama (3 pav.). Išskirti trys pagrindiniai aktoriai: vartotojas ir sistema.



3 pav. Panaudos atvejų diagrama

Daugiausiai panaudos atvejų priskirta sistemai, kadangi pagrindinis kodo transformavimo procesas turi vykti automatiškai, be žmogaus įsikišimo. Sistemos vartotojas gali peržiūrėti sistemos veikimo rezultatus – sugeneruotą kodą. Taip pat peržiūrėti visą informaciją apie generavimą įskaitant ir klaidas, bei jų priežastis.

4.3.1. Panaudos atvejų sąrašas

4 lentelė Panaudos atvejo „Konvertuoti kodą“ detali specifikacija

PA „Konvertuoti kodą“		
Aktorius		Vartotojas
Prieš sąlyga		Aktorius yra įsijungęs programą
Sužadinimo sąlyga		Aktorius nori konvertuoti kodą
Susiję Panaudojimo atvejai	**Išplečia PA**	
	Apima PA	
	Specializuoja PA	
Pagrindinis įvykių srautas		**Sistemos reakcija ir sprendimai**
Aktorius įkelia savo kodą		Sistema patikrina failą(-us) ir konvertuoja kodą
Po sąlyga		Konvertuotas kodas
Alternatyvūs scenarijai		
1. Vartotojas įkelią ne kodą		Sistema pateikia klaidos pranešimą
2. Sistema neatpažįsta kodo		Sistema pateikia klaidos pranešimą

5 lentelė Panaudos atvejo „Peržiūrėti informacija apie konvertavimą“ detali specifikacija

PA „Peržiūrėti informacija apie konvertavimą“		
Aktorius		Vartotojas
Prieš sąlyga		Aktorius yra įkėlęs failus
Sužadinimo sąlyga		Sistema konvertuoja kodą
Susiję Panaudojimo atvejai	**Išplečia PA**	
	Apima PA	
	Specializuoja PA	
Pagrindinis įvykių srautas		**Sistemos reakcija ir sprendimai**
Sistema konvertuoja kodą		Sistema konvertuoja kodą ir teikia informaciją
Po sąlyga		Pateikta informacija

6 lentelė Panaudos atvejo „Išeities kodo pertvarkymas“ detali specifikacija

PA „Išeities kodo pertvarkymas“		
Aktorius		Sistema
Prieš sąlyga		Aktorius yra įkėlęs failus
Sužadinimo sąlyga		Aktorius nori konvertuoti kodą
Susiję Panaudojimo atvejai	**Išplečia PA**	
	Apima PA	
	Specializuoja PA	
Pagrindinis įvykių srautas		**Sistemos reakcija ir sprendimai**
Sistema pertvarko įkeltą išeities kodą		Sistema pertvarko įkeltą išeities kodą
Po sąlyga		Pertvarkytas išeities kodas
Alternatyvūs scenarijai		
1. Sistema negali pertvarkyti kodo		Sistema pateikia klaidos pranešimą
2. Sistema neturi ko pertvarkyti		Sistema grąžina netvarkytą kodą

7 lentelė Panaudos atvejo „ASG medžio generavimas“ detali specifikacija

PA „ASG medžio generavimas“		
Aktorius		Sistema
Prieš sąlyga		Sistema yra pertvarkiusi kodą
Sužadinimo sąlyga		Sistema nori generuoti ASG medį
Susiję Panaudojimo atvejai	**Išplečia PA**	
	Apima PA	
	Specializuoja PA	
Pagrindinis įvykių srautas		**Sistemos reakcija ir sprendimai**
Sistema generuoja ASG medžio grafą		Sistema generuoja ASG medžio grafą
Po sąlyga		Sugeneruotas ASG medžio grafas
Alternatyvūs scenarijai		
1. Sistema negali sugeneruoti medžio		Sistema pateikia klaidos pranešimą

8 lentelė Panaudos atvejo „AST medžio generavimas“ detali specifikacija

PA „AST medžio generavimas“		
Aktorius		Sistema
Prieš sąlyga		Sugeneruotas ASG medis
Sužadinimo sąlyga		Sistema nori generuoti AST medį
Susiję Panaudojimo atvejai	**Išplečia PA**	ASG medžio generavimas
	Apima PA	
	Specializuoja PA	
Pagrindinis įvykių srautas		**Sistemos reakcija ir sprendimai**
Sistema generuoja AST medžio grafą		Sistema generuoja AST medžio grafą
Po sąlyga		Sugeneruotas AST medžio grafas
Alternatyvūs scenarijai		
1. Sistema negali sugeneruoti medžio		Sistema pateikia klaidos pranešimą

9 lentelė Panaudos atvejo „Išeities kodo transformavimas“ detali specifikacija

PA „Išeities kodo transformavimas“		
Aktorius		Sistema
Prieš sąlyga		Sugeneruotas AST medis
Sužadinimo sąlyga		Sistema nori transformuoti kodą
Susiję Panaudojimo atvejai	**Išplečia PA**	AST medžio generavimas
	Apima PA	Transformavimo taisyklių surinkimas
	Specializuoja PA	
Pagrindinis įvykių srautas		**Sistemos reakcija ir sprendimai**
Sistema naudoja transformavimo taisykles		Sistema naudoja transformavimo taisykles
Sistema transformuoja kodą		Sistema transformuoja kodą
Po sąlyga		Pateikiamas transformuotas kodas
Alternatyvūs scenarijai		
1. Sistema negali transformuoti kodo		Sistema pateikia klaidos pranešimą
2. Sistema negali panaudoti transformavimo taisyklių		Sistema grąžina netvarkytą kodą

10 lentelė Panaudos atvejo „Transformavimo taisyklių surinkimas“ detali specifikacija

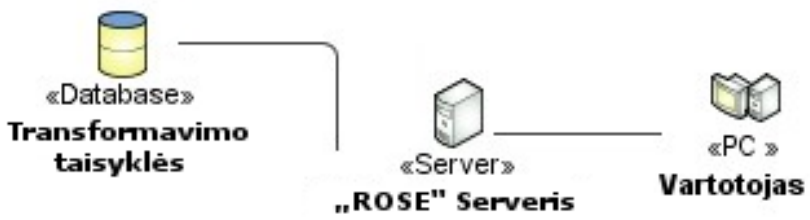
PA „Transformavimo taisyklių surinkimas“		
Aktorius		Sistema
Prieš sąlyga		Sistema nori transformuoti kodą
Sužadinimo sąlyga		Sistema pradeda transformuoti kodą
Susiję Panaudojimo atvejai	**Išplečia PA**	
	Apima PA	
	Specializuoja PA	
Pagrindinis įvykių srautas		**Sistemos reakcija ir sprendimai**
Sistema ištraukia iš duomenų bazės transformavimo taisykles		Sistema ištraukia iš duomenų bazės transformavimo taisykles
Po sąlyga		Pateikiamos transformavimo taisyklės sistemai
Alternatyvūs scenarijai		
1. Nėra sąryšio su duomenų baze		Sistema pateikia klaidos pranešimą
2. Grąžinamos klaidingos taisyklės		Sistema pateikia klaidos pranešimą

4.4. Sistemos prototipo projektavimas

Sukurtos sistemos specifikacija pateikiama šiomis diagramomis: sistemos išdėstymo, sistemos statinio ir duomenų vaizdo. Tai pat aprašomi pagrindiniai kodo apdorojimo algoritmai.

4.4.1. Išdėstymo vaizdas

Reikalinga techninė įranga, kurioje sistema bus išdėstyta ir veiks, pateikiama žemiau esančioje diagramoje.



4 pav. Sistemos išdėstymo vaizdas

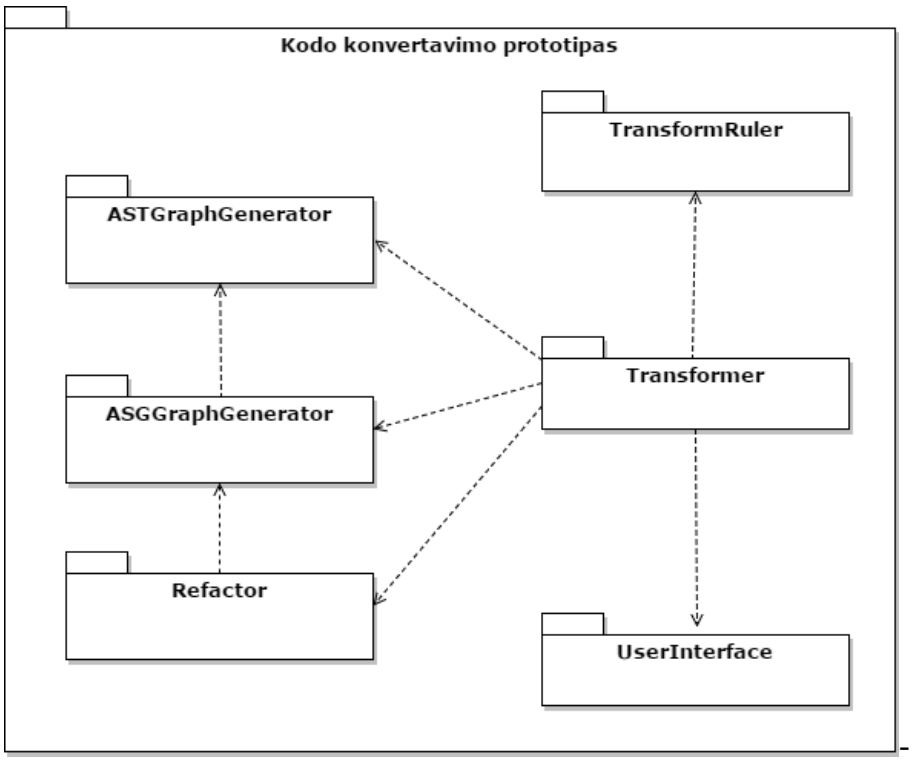
Svarbūs faktai apie sukurtą sistemą:

- Sistemos prototipas gali vėliau būti praplėstas į galutinę išplėstą sistemą.
- Dalis sistemos funkcionalumo buvo realizuojama naudojantis jau sukurtais komponentais.

- Pagrindinis sistemos veikimas vyksta vartotojo kompiuteryje.
- “ROSE” serveris naudojamas reikiamoms transformavimo taisyklėms gauti.

4.4.2. Sistemos statinis vaizdas

Sistema suskaidyta į septynis paketus aukščiausiame lygyje, kuriuose realizuotas skirtingas funkcionalumas, bei visi paketai glaudžiai susiję vienas su kitų (5 pav.).



5 pav. Sistemos statinis vaizdas

Sistemą suskaldyta į šiuos paketus:

- „ASTGraphGenerator“ paketas atsakingas už AST tipo grafo sukūrimą.
- „ASGGraphGenerator“ paketas atsakingas už ASG tipo grafo sukūrimą.
- „UserInterface“ paketas atsakingas už informacijos pateikimą vartotojui.
- „Refactor“ pakete realizuotos su kodo pertvarkymo susijusios funkcijos. Jos atsakingos, kad kodas būtų paruoštas prieš modelių generavimą.
- „Transformer“ pakete realizuotos visos funkcijos reikalingos kodo transformavimui iš vienos kalbos į kitą.
- „TransformRuler“ paketas atsakingas už transformavimo taisyklių gavimą ir naudojimą.

4.4.3. Bendras sistemos veikimo aprašymas

Sistemoje saugomos transformacijos taisyklių gavimas ir atnaujinimas vyksta automatiškai, kaskart prieš vykdant pagrindinį transformavimą. Informacijos apie generavimą atvaizdavimas ekrane ir pats generavimas vyksta lygiagrečiai. Sistema inicijuoja generavimo procedūrą, kuri susideda iš šių fazių:

1. Inicijuojamas kodo pertvarkymas. Kodas pertvarkomas, kad palengvinti ASG ir AST diagramų generavimui, bei išeities kodo transformavimui iš vienos aplinkos į kitą.
2. Inicijuojamas ASG grafiko generavimas.
3. Inicijuojamas AST grafiko generavimas.
4. Inicijuojamas kodo transformavimas, pasitelkiant kodo transformavimo taisykles.

Vartotojas mato visą transformavimo informaciją. Kadangi šios informacijos generavimas ir pateikimas gali vykti lygiagrečiai - funkcionalumas, atsakingas už informacijos atvaizdavimą, yra netrikdomas.

4.4.4. Programos kodo pertvarkymo realizavimas

Programos kodo pertvarkymas (*angl. refactoring*) – tai procesas, kurio metu programinės įrangos išeities kodas keičiamas taip, kad nepasikeistu sistemos veikimas, bet vidinė kodo struktūra pagerėtų. Programos kodo pertvarkymas yra procesas kuris prastą programos kodą perverčia į tinkamesnį kodą. Pertvarkomas kodas nebūtinai turi būti prastas, pertvarkymas gali būti atliekamas programuotojo arba kuriamos sistemos darbo palengvinimui.

Pagrindinės kodo pertvarkymo savybės:

- Pertvarkymo įrankiai gali būti sudėtingi, nes nėra lengva suvaldyti chaosą tačiau jie yra reikalingi ir svarbūs programavimui.
- Pertvarkymas vykdomas pagal tam tikras logikos taisykles. Pertvarkyti kodą galima skirtingais keliais būdai, todėl skirtingi programuotojai gali pasirengti skirtingus pertvarkymo variantus. Kai tai daro sistema, o ne žmogus, kodas gali tapti sunkiai skaitomas, todėl pravartu turėti tinkamas pertvarkymo taisykles.
- Tinkami pertvarkymo algoritmai gali programavimo kodą padaryti lengviau suprantamą didesnei žmonių grupei.
- Pertvarkymo kaštai gali būti maži arba dideli.

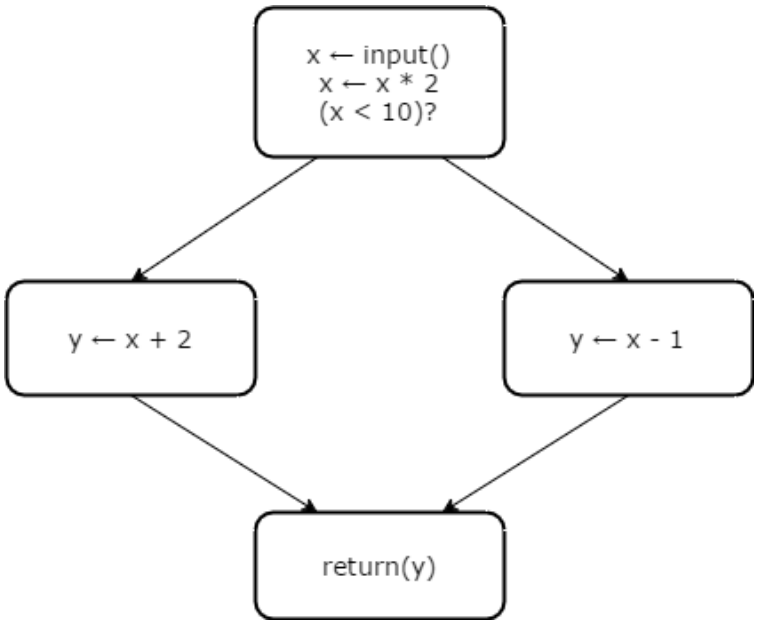
Programos kodo pertvarkymo pavyzdys, kuomet norima pakeisti metodo pavadinimą:

- Patikrinama ar keičiamas metodas nėra naudojamas tėvo ar vaikų klasėse. Jei naudojamas, tuomet sekantys žingsniai atliekami kiekvienoje tokioje klasėje.
- Sukuriamas naujas metodas su norimu nauju pavadinimu ir įdedamas senojo metodo turinys. Jei reikia, atlikite pakeitimus.
- Pakeičiama senojo metodo realizacija taip, kad šis iškviestų naująjį metodą
- Ieškomos visos nuorodos į senąjį metodą ir pakeičiamos nuorodomis į naująjį metodą.
- Jeigu įmanomą, ištrinamas senasis metodas, jeigu neįmanoma, tuomet pažymima, kad jis yra pasenęs (*angl. deprecated*).
- Kompiliuojama ir išbandoma.

4.4.5. SSA algoritmo realizavimas

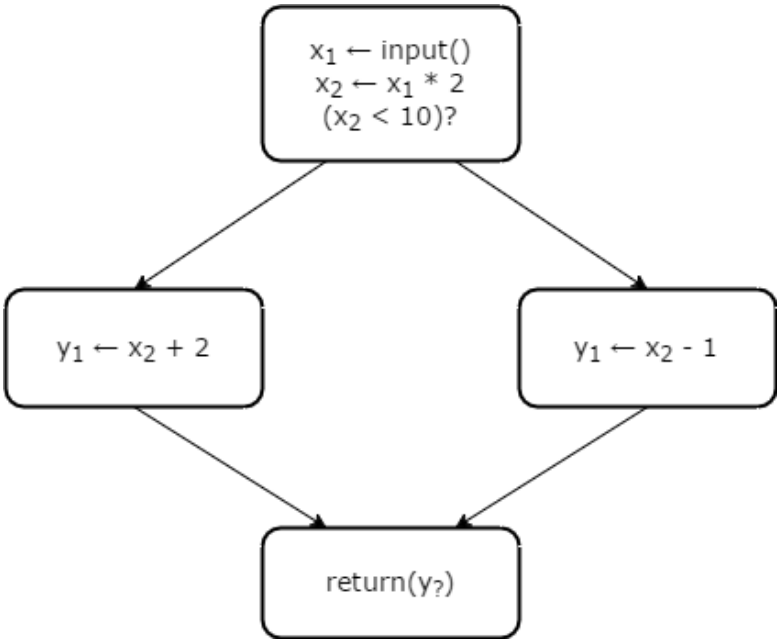
Pavyzdinis SSA algoritmo naudojimas:

(1) Nuskaitomas pavyzdinis kodas, kuriame kaip nurodyta nuskaitoma y reikšmė, ši reikšmė padauginama iš 2 ir jei gautas skaičius yra didesnis už 10, tuomet iš šios reikšmės atimamas vienetas, jei gautas skaičius mažesnis tuomet pridedamas dvejetas ir gautas rezultatas yra grąžinamas.



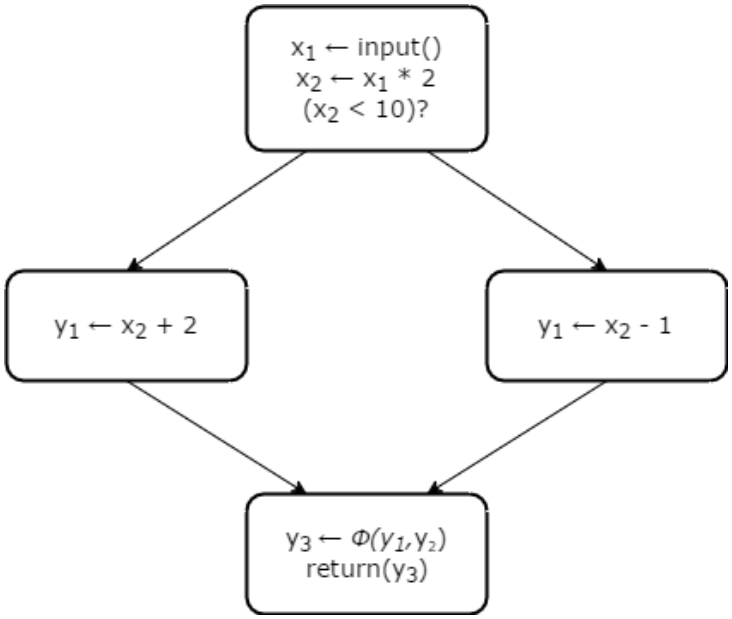
6 pav. Kodo atvaizdavimas prieš SSA algoritmo pritaikymą

(2) Pakeitus kintamuosius jų versijomis programos veikimo principas išlieką toks pat, tačiau dabar kiekvienas kintamasis yra unikalus.



7 pav. Kodo atvaizdavimas nebaigus SSA algoritmo pritaikymą

(3) Tačiau, dabar nėra aišku kurį „y“ kintamąjį gražinti. Tam tikslui yra pridedama speciali konstanta Φ (fi), kuri išves naują kintamąjį y_3, kuris turės y_1 arba y_2 reikšmę, priklausomai pagal kurį kelią buvo atvykta.



8 pav. Kodo atvaizdavimas baigus SSA algoritmo pritaikymą

4.4.6. AST grafo realizacija

11 lentelė Kintamųjų naudojimas pagal SSA modulį

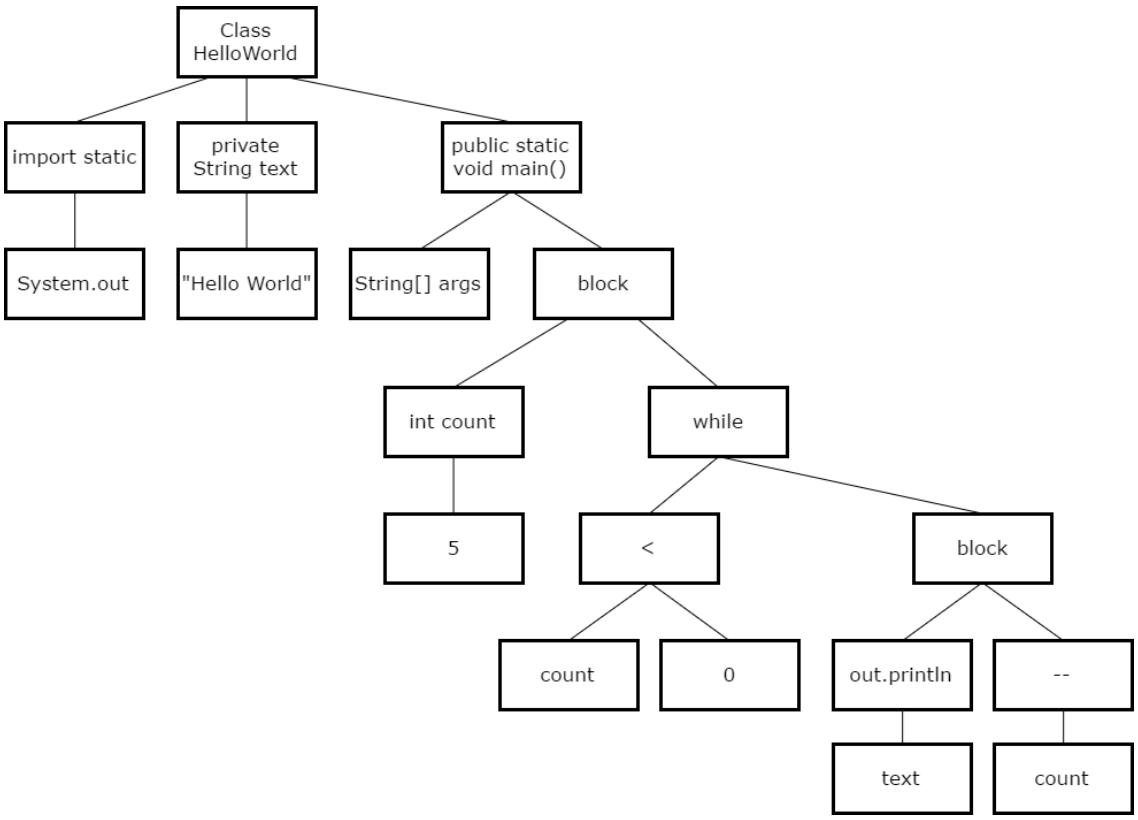
```
import static java.lang.System.out;

public class HelloWorld {

    private static String text = "Hello world!";

    public static void main(String[] args) {
        int count = 5;
        while (count > 0) {
            out.println(text);
            count--;
        }
    }
}
```

Abstrakčios sintaksės medis yra kodo reprezentacija, kuri yra dažniausiai naudojama kompiliatoriuose. AST yra išeities kodo atvaizdavimas taip, kad jį būtų įmanoma iš naujo sugeneruoti iš AST grafo į lygiavertį kodą originaliam kodui. Vieninteliai dalykai kurie nėra modeliuojami AST grafe yra tarpai, tuščios eilutės ir komentarai.



9 pav. AST grafo generavimo realizacijos pavyzdys pagal pateiktą kodą

AST yra glaudžiai susijęs su analizės medžiu. Vienintelis skirtumas tarp AST ir analizės-medžio yra tai, kad AST paprastai pašalina nereikalingas konstrukcijas, pavyzdžiui, nenaudingus kintamuosius skliausteliuose. Nėra aiškiai apibrėžtų taisyklių kuriuos nenaudingus konstruktorius pašalinti ir gali skirtis nuo vieno įgyvendinimo ir kito.

Pagrindiniai AST mazginių taškų principai:

- Yra baigtinis mazginių taškų skaičius. Skirtingi mazginiai taškai atitinka skirtingas konstrukcijas programavimo kalboje (Deklaracija, kviečiantis metodas, visi pagrindiniai operatoriai ir kt.)
- Mazginiai taškai paprastai įgyvendinami klasių su tinkama hierarchija. Pavyzdžiui, mazginiai taškai skirti konstantai ir paprastam kintamajam paprastai pratęsti nuo kai kurios abstrakčios mazgo modeliavimo išraiškos.
- Kiekvienas mazginis taškas turi apibrėžtas savybes. Pavyzdžiui "metodo deklaracijos" mazgas turi metodo pavadinimą, modifikatorius ir grąžinimo tipą.
- Kiekvienas mazginis taškas turi iš anksto nustatytus galimus vaikinius mazgus. Pavyzdžiui "metodo deklaracijos" mazginis taškas turi du vaikinius mazgus: vienas skirtas parametrų sąrašui, ir vienas skirtas įgyvendinimui.

4.4.7. Kodo transformavimo taisyklių naudojimas

Transformavimo taisyklių saugojimui ir naudojimui pasitelkiamas „ROSE“ įrankis. Transformuojant prireikus transformavimo taisyklės siunčiama užklausa į „ROSE“ duomenų bazę. Iš ten gavus atsaką šį taisyklė laikomi sistemoje visą sistemos veikimo laiką. Taip pasirūpinama, kad sistema visada naudotu naujausias transformavimo taisykles.

4.4.8. Kodo konvertavimo realizacija

„Rewriter“ modulis modifikuoją pirminius Java AST modelius, kad būtų palengvintas konvertavimas į objektinę C kalbą.

Objektinė C kalba neturi abstraktaus metodo atitikmens, todėl šis algoritmas pasirūpina, kad tokie metodai, būtų pakeisti atitinkamais jų įgyvendinimais. Java taip pat leidžia implementuoti sąsajas nedeklaruojant visus, tos klasės metodus, jei jie įgyvendinti tėvinėje klasėje. Pavyzdys būtų „java.util.Map“ klasės metodai „equals()“ ir „hashCode“ yra įgyvendinti klasėje „java.lang.Object“. Objektinė C reikalauja, kad visi naudojami klasės metodai būtų deklaruojami joje, todėl „Rewriter“ pasirūpina, kad tai būtų atliktą. Objektinė C kalba taip pat neleidžia kintamiesiems ar klasėms turėti tokius pat pavadinimus, kuomet

Java kalboje, tai leidžiama. Jei „Rewriter“ randa tokius metodus ar kintamuosius, juos pervadina.

4.4.9. Informacijos apie konvertavimą atvaizdavimo realizacija

Informacijos atvaizdavimui pasirinkta naudoti komandinės eilutės sąsają kadangi konvertavimo informacijos kiekis nėra didelis ir susideda tik iš teksto. Sukurtame sistemos prototipe atvaizduojama tokia informacija:

- Baigtų konvertuoti ir dar laukiamu konvertuoti failų skaičius.
- Einamo metu konvertuojamo failo pavadinimas.
- Sėkmės arba nesėkmės įspėjimas.
- Klaidų sąrašas ir priežastys.

5. EKSPERIMENTINĖ DALIS

5.1. Eksperimentinio tyrimo tikslas

Eksperimentu siekiama ištirti kaip kinta generavimo rezultatai taikant skirtingas programavimo funkcijas. Matuojamos skirtingų sudėtingumo ir dydžio lygio metrikų klasės. Analizuojama ar esat dideliam programavimo sudėtingumui, sistema susitvarko su išeities kodo transformavimu, rezultatai lyginami naudojant skirtingas tiriamąsias metrikas.

Eksperimentams atlikti naudojamos metrikos:

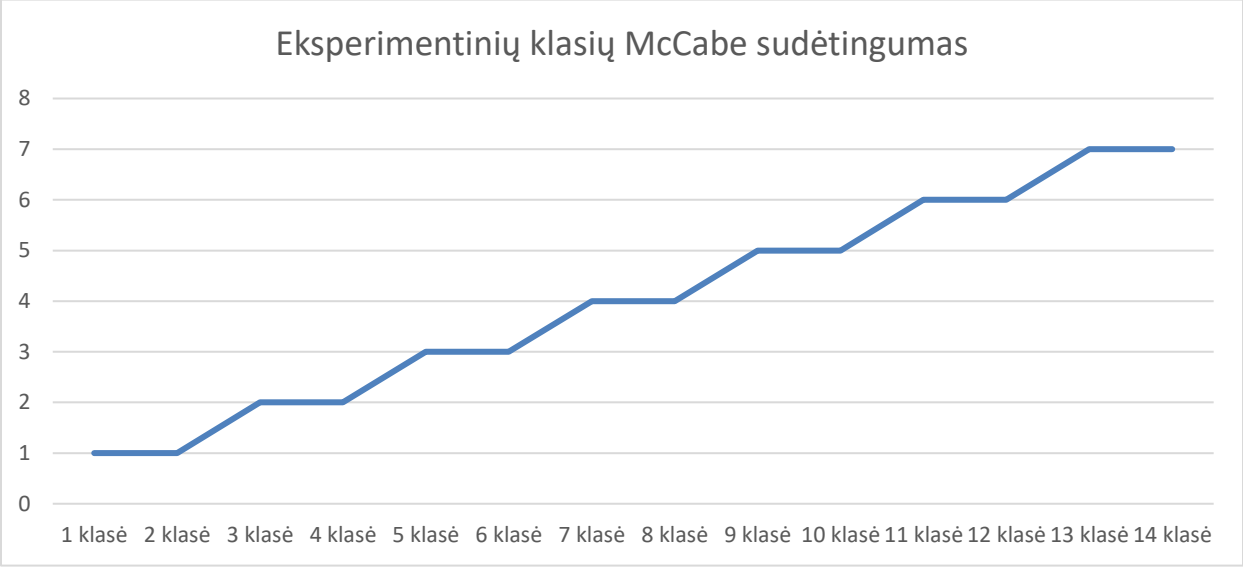
- Programos dydis kodo eilutėmis (LOC).
- McCabe Ciklo matinis sudėtingumas (angl. *Cyclomatic Complexity,* CC) – nepriklausomų kelių programos kodo grafe skaičius.
- Metodų skaičius klasėje (angl. *weighted methods per class,* WMC).

Pagal šiuos kriterijus atrenkami pradiniai duomenys. Viso surinkta 14, skirtingo sudėtingumo, atvirojo kodo programų (angl. open source).

5.2. Eksperimentinio tyrimo aprašymas

5.2.1. Eksperimentinių klasių McCabe sudėtingumas

Tiriama, kaip vyksta kodo transformavimas, pateikiant skirtingo sudėtingumo išeities kodus. Išskiriamos trys sudėtingumo metrikos: programos kodo eilučių skaičiaus, metodų skaičius klasėje ir McCabe ciklo matinio sudėtingumo.

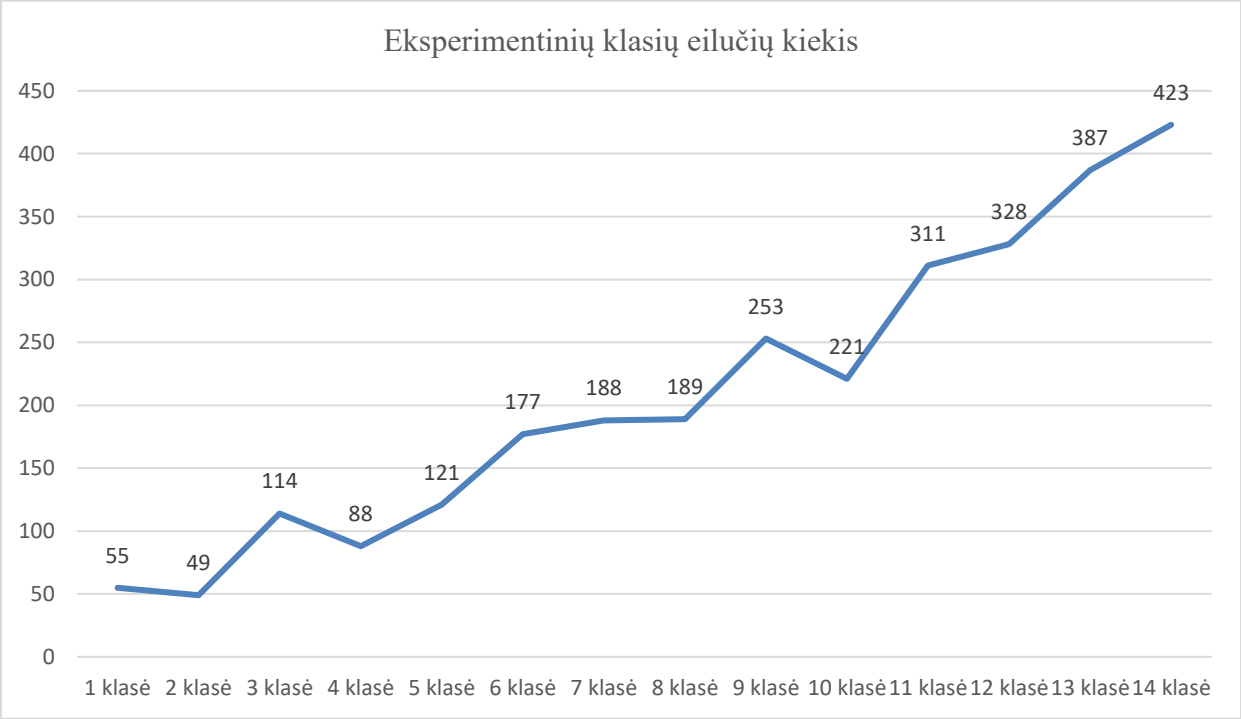


10 pav. Eksperimentinių klasių McCabe sudėtingumas

Paveikslėlyje 11, pateikiami visų surinktų programų sudėtingumai. Buvo parinktos po 2 atvirojo kodo programos skirtingoms McCabe sudėtingumo metrikoms rėžiuose nuo 1 iki 7.

Ciklo matinio sudėtingumo variantas yra esminis sudėtingumas (*Essential complexity*, ev(G)) – tai ciklo matinis sudėtingumas skaičiuojamas programoje pakeitus visus struktūrizuotus sakinius paprastu sakiniu. Jei ev(G) > 4, programos prižiūrimumas yra blogas [18].

5.2.2. Eksperimentinių klasių eilučių kiekis



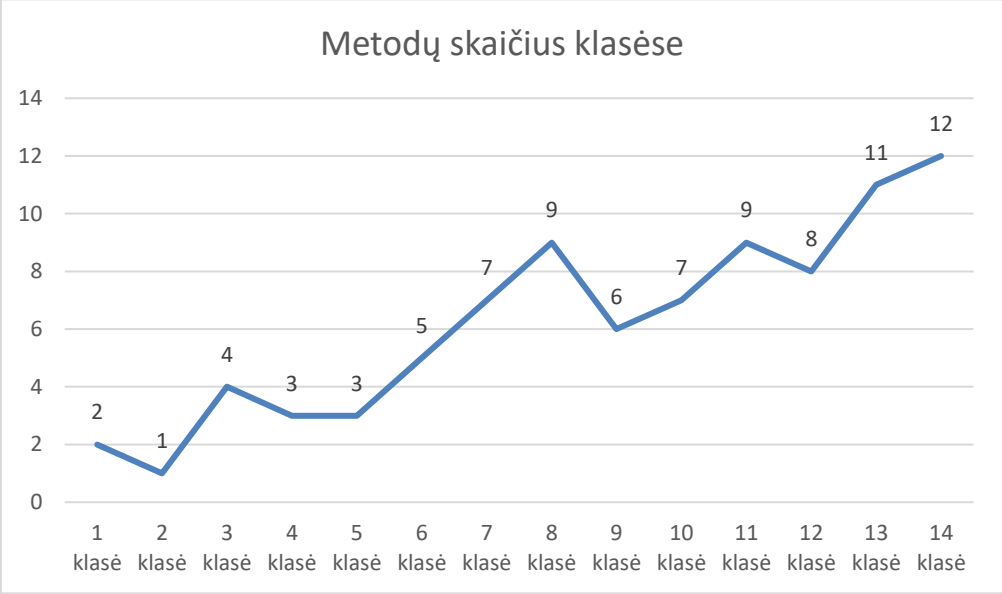
11 pav. Eksperimentinių klasių eilučių kiekis

Programos dydžio kodo eilutėmis metrika buvo pasirinkta, todėl, nes norėta išsiaiškinti ar sistema tinkamai susidoroja, su dideliomis klasėmis. Programos modulio (klasės, komponento) sudėtingumas yra žemas, jei jo dydis yra mažesnis nei 100 LOC. Taip pat bus tikrinama kaip pasikeitė kodo eilučių skaičius po transformacijos.

5.2.3. Eksperimentinių klasių metodų skaičius

Taip pat eksperimento metu tikrinama kaip keičiasi metodų skaičius klasėje (angl. *weighted methods per class*, WMC). Kuo WMC didesnis, tuo klasėje gali būti daugiau klaidų, tuo daugiau reikia pastangų klasės sukūrimui ir priežiūrai. Klasės su aukšta WMC

reikšme turi būti išskaidytos į kelias mažesnes klases. Rekomenduojama, kad WMC neviršytų 20, o sistemoje būtų ne daugiau kaip 10% klasių, kurių WMC reikšmė yra didesnė nei 24 [19].



12 pav. Metodų kiekis klasėse

Kadangi eilučių kiekis klasėse nėra didelis, todėl ir metodų skaičius yra palyginus mažas. Ši metrika naudojama tam, kad būtų galima patikrinti kaip skiriasi metodų skaičius klasėse po transformacijos.

5.3. Eksperimentinio tyrimo eiga

Eksperimentu siekiama ištirti kaip kinta generavimo rezultatai taikant skirtingas programavimo funkcijas. Matuojami kodo eilučių skaičiai po transformacijos, gautas klaidų skaičius. Analizuojama ar transformuotas kodas kompiliuojasi ar reikalingi papildomi taisymai. Rezultatai lyginami naudojant skirtingas atvirojo kodo programas.

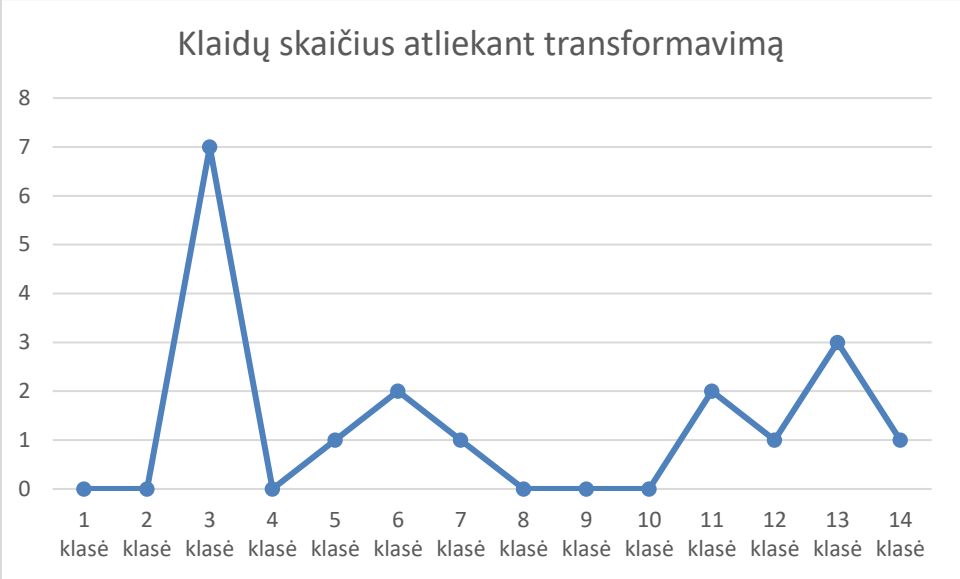
Pagrindinis eksperimento tikslas yra išsiaiškinti, silpnąsias sistemos vietas ir ar pasirinkti transformacijos metodai yra teigiami.

5.4. Eksperimentinio tyrimo rezultatai

5.4.1. Transformacijos klaidų skaičius

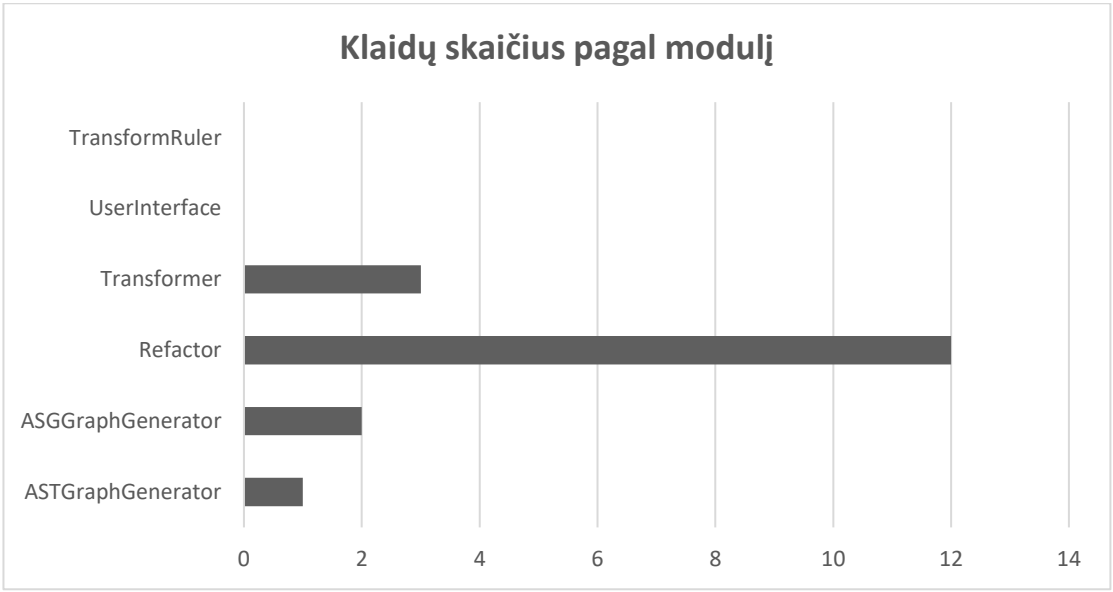
Transformuojant su keliomis klasėmis iškilo problemų ir sistema metė klaidas. Prireikė pradinio išeities kodo pertvarkymo, kad sistema priimtu ir transformuotu jį sėkmingai.

Didžiausios problemos kilo su 3 klase, dėl naudojamos pasenusios (*angl. deprecated*) „FloatMath“ bibliotekos. Reikėjo pakeisti šią biblioteką į „Math“, kad sistema, galėtu sėkmingai transformuoti kodą.



13 pav. Klaidų skaičius atliekant transformavimą

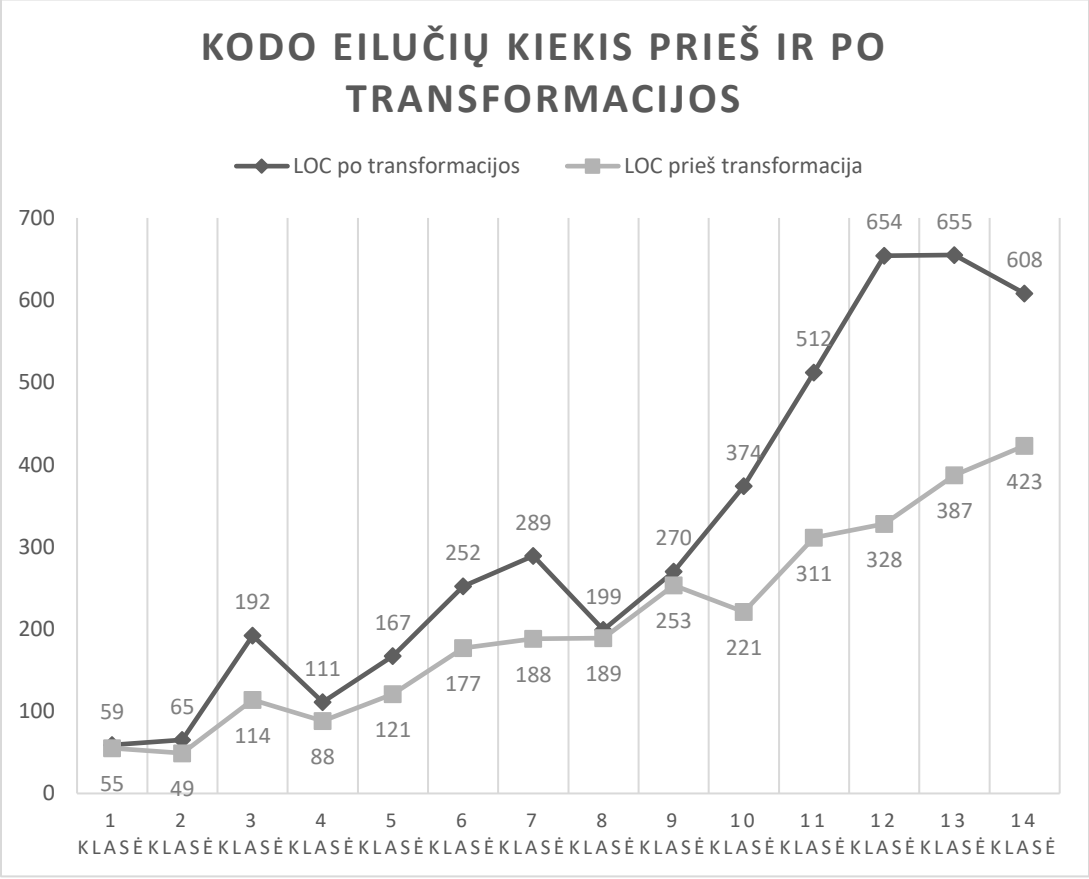
Daugiausia klaidų grįžo iš kodo pertvarkymo („Refactor“) modulio. Galima teigti, kad tai silpniausia sistemos dalis. Norint sumažinti šių klaidų skaičių, reikia peržiūrėti ir pertvarkyti šį modulį. „TransformRuler“ ir „UserInterface“ klaidų negrąžino, o „ASTGraphGenerator“, „ASGGraphGenerator“ ir „Transformer“ moduliai gražino priimtiną klaidų skaičių.



14 pav. Klaidų skaičius pagal modulį

5.4.2. Kodo eilučių skaičiai po transformacijos

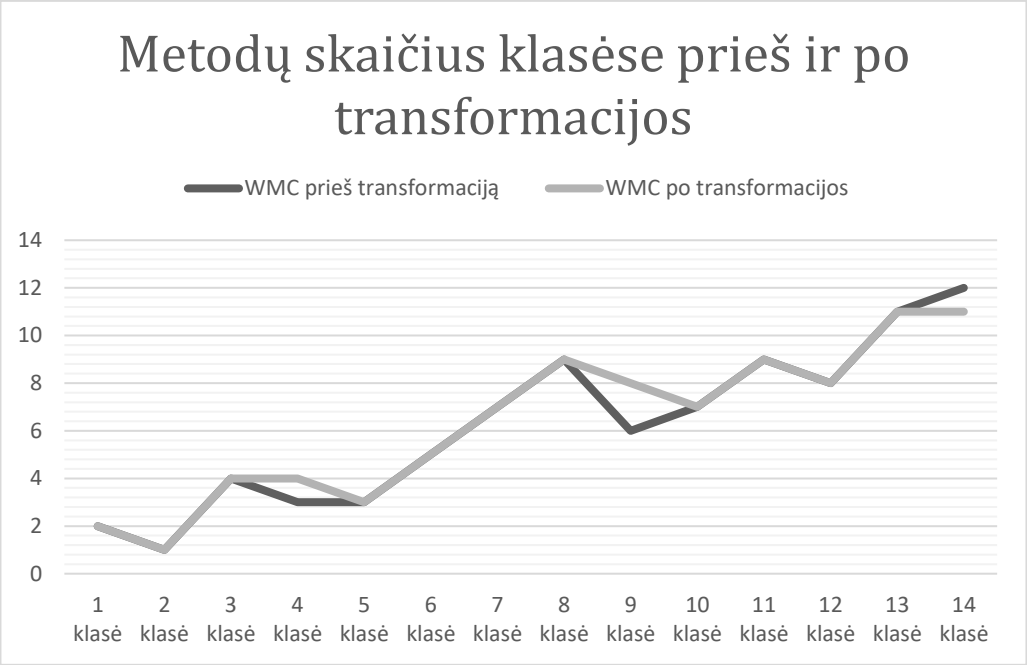
Atlikus transformaciją visų klasių kodo eilučių kiekis išaugo, pirmos, antros, ketvirtos, aštuntos ir devintos klasės kodo eilučių skaičius išaugo nežymiai, taigi galime teigti, kad programos transformacija yra vertinama teigiamai. Prasčiausiai vertinamos yra trečia, vienuolikta, dvylikta ir trylikta klasės, kurių kodo eilučių skaičius beveik padvigubėjo. Taip pat iš grafiko (13 paveikslėlis) galime matyti, kad sudėtingesnių programų kodo eilučių skaičius smarkiai išaugą. Tai gali būti sudėtingų AST medžio grafų. Sprendimas būtų tobulinti kodo pertvarkymo (*angl. refactor*) modulį, kad didelės klasės būtų išskaidomos į mažesnes. Tačiau šiuo skaidymo didėja jungumas tarp klasių (*angl. coupling between object classes,* CBO) – kuo didesnis jungumas, tuo sudėtingesnė klasės priežiūra. Jei CBO > 14, klasę reikia labai gerai ištestuoti [20].



15 pav. Kodo eilučių kiekis prieš ir po transformacijos

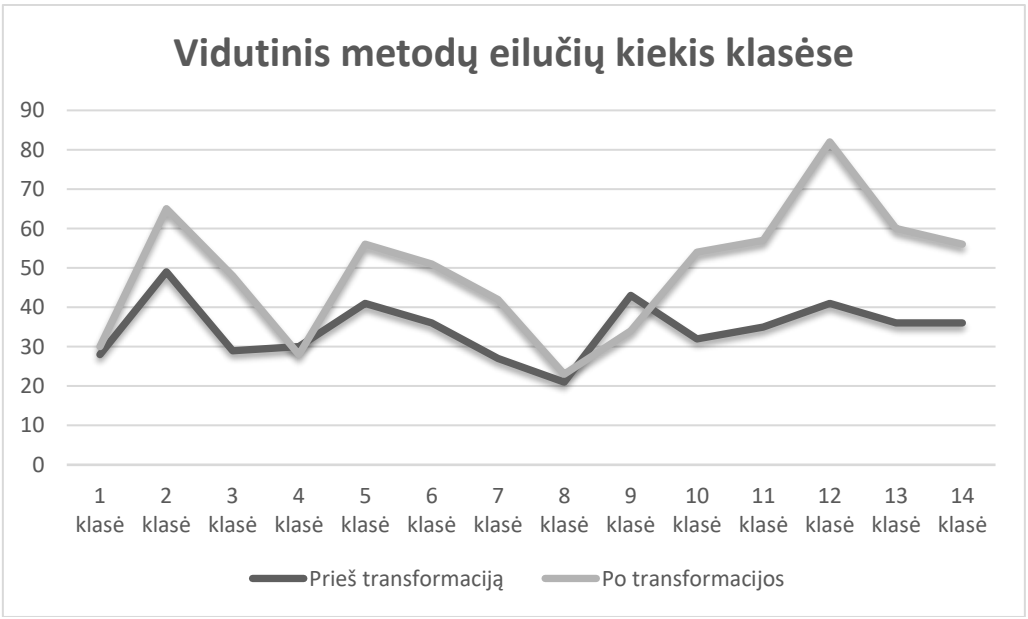
5.4.3. Metodų skaičius klasėse po transformacijos

Atlikus transformaciją daugumos klasių metodų kiekis liko nepakitęs. Ketvirtos ir devintos klasės metodų skaičius padidėjo, „Refactor“ modulis išskaidė kelis metodus, o keturioliktos klasės metodų skaičius sumažėjo.



16 pav. Metodų kiekis prieš ir po transformacijos

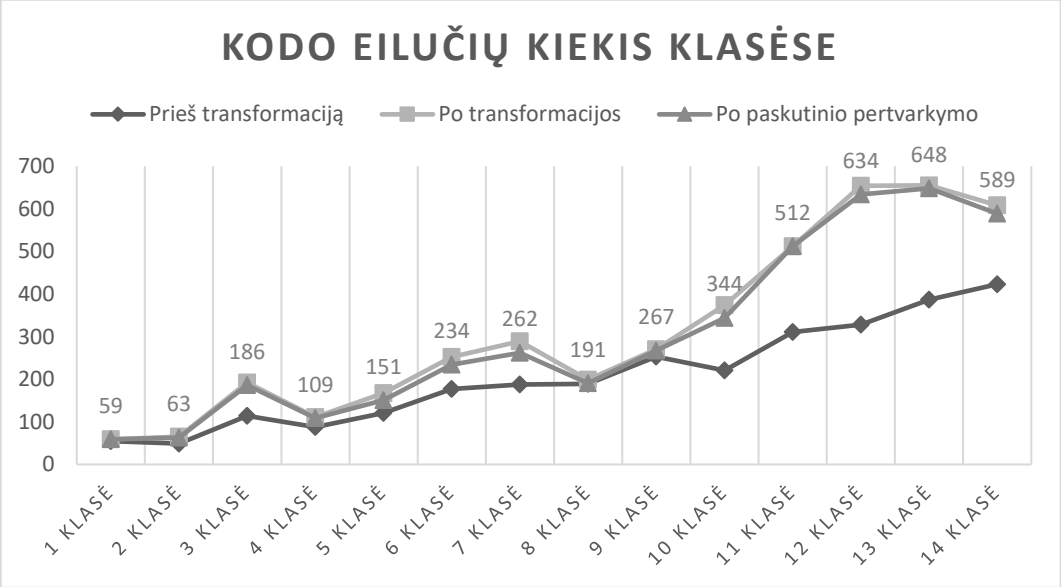
Taip pat palyginamas vidutinis metodų kodo eilučių kiekis. Iš diagramos (18 paveikslėlis) galime matyti, kad eilučių kiekis smarkiai išaugą.



17 pav. Vidutinis metodų eilučių kiekis klasėse

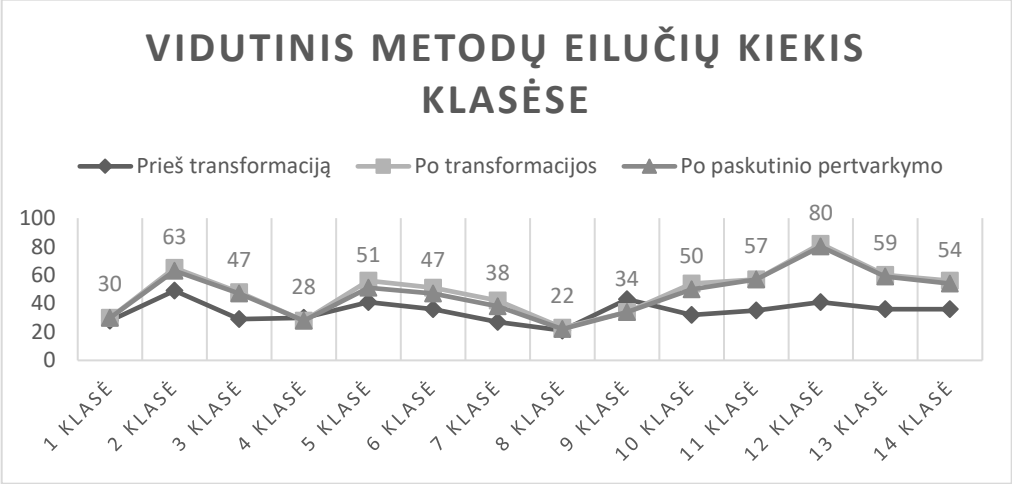
Tik keturiose klasėse šis vidurkis buvo nežymiai išaugęs arba net sumažėjęs. Viena iš sistemos plėtojimo kryptis galėtu būti „Refactor“ modulio pakartotinis naudojimas po transformacijos. Tai galėtu sumažinti vidutinį kodo eilučių kiekį metoduose ir taip pat pagerinti kodo įskaitomumą.

Atlikus galutinį pertvarkymą po transformacijos kodo eilučių kiekis visose klasėse sumažėjo arba liko toks pat, tačiau skirtumas nėra didelis.



18 pav. Kodo eilučių kiekis klasėse po paskutinio pertvarkymo

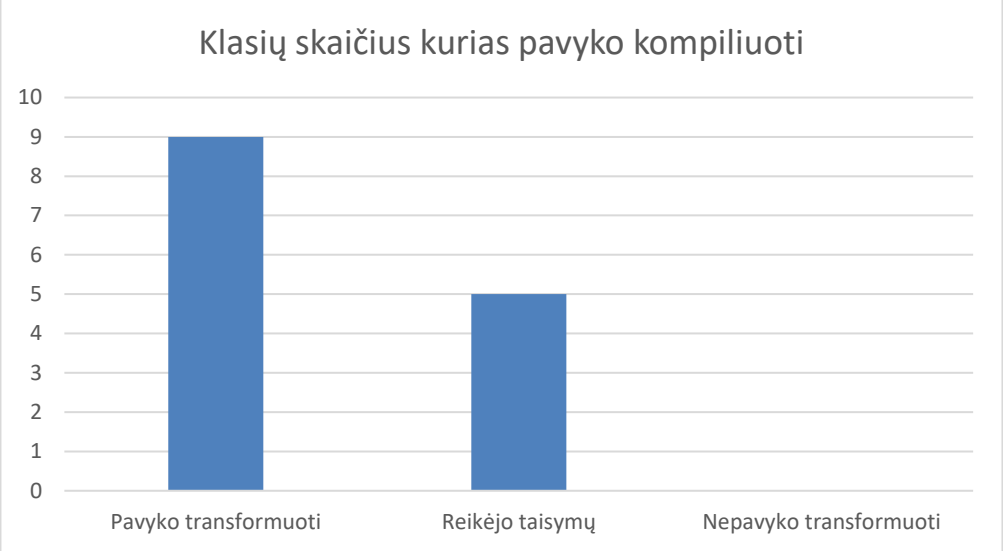
„Refactor“ modulis nėra pritaikytas mažinti kodo eilučių skaičių, bet labiau paruošti išeities kodą transformavimui, todėl jo naudojimosi po transformacijos daug naudos neatneša. Reikėtu rašyti naują modulį, kuris ir būtent būtų skirtas kodo eilučių skaičiui mažinti.



19 pav. Vidutinis metodų eilučių kiekis klasėse po paskutinio pertvarkymo

5.4.4. Transformuoto kodo kompiliavimas

Po transformacijos buvo vykdomas kompiliavimas, tikrinimui ar sėkmingai transformuota. 9 klasės buvo kompiliuotos be jokių problemų, tačiau penkioms klasėms prireikė nedidelių pataisymų, kad kompiliavimas įvyktų. Didžiausios klaidos buvo kintamųjų neatitikimai po transformacijos. Nebuvo nei vienos klasės, kurios nepavyktu pataisyti, kad įvyktų sėkmingas kompiliavimas.



20 pav. Klasių skaičius kurias pavyko kompiliuoti

5.5. Eksperimentinio tyrimo išvados

- Atlikus transformavimo eksperimentą nustatyta, kad daugiausia klaidų grįžo iš kodo pertvarkymo („Refactor“) modulio. Galima teigti, kad tai silpniausia sistemos dalis. Šią vietą planuojama tobulinti ateityje
- Eksperimento metu nebuvo nei vienos klasės, kurios nepavyktu pataisyti, kad įvyktų sėkmingas kompiliavimas po išeities kodo transformavimo. Galima teigti, kad sistema veikia teisingai.
- Atliekant eksperimentą nustatyta, kad išeities kodo eilučių skaičius gali padidėti iki dviejų kartų. Tačiau buvo ir pavyzdžių kuomet kodo eilučių skaičius tarp pradinės programos klasės ir transformuotos buvo nežymus.
- Eksperimento metu nustatyta, kad sudėtingesnių programų kodo eilučių skaičius smarkiai išaugą. Tai gali būti sudėtingų AST medžio grafų. Sprendimas būtų tobulinti kodo pertvarkymo (*angl. refactor*) modulį, kad didelės klasės būtų išskaidomos į mažesnes.
- Didžiausios problemos kilo su 3 klase, dėl naudojamos pasenusios (*angl. deprecated*) „FloatMath“ bibliotekos. Reikėjo pakeisti šią biblioteką į „Math“, kad sistema, galėtu sėkmingai transformuoti kodą.
- „Refactor“ modulis nėra pritaikytas mažinti kodo eilučių skaičių, bet labiau paruošti išeities kodą transformavimui, todėl jo naudojimosi po transformacijos daug naudos neatneša. Reikėtu rašyti naują modulį, kuris ir būtent būtų skirtas kodo eilučių skaičiui mažinti.
- Apibendrinant šio eksperimento rezultatus, norint išvengti transformacijos metu programos kodo eilučių kieko augimo ir išspręsti išeities kodo pertvarkymo problemas, šiuo atveju reikėtų, pagrinde atlikti papildomus tiriamuosius darbus, kurie padėtu rasti arba realizuoti papildomus įrankius arba karkasus, kurie pagerintu kodo pertvarkymą prieš transformacijas. Tokiu atveju programos kodas bus tinkamai paruoštas - universalus, lengvai prižiūrimas, ir nereikalaus papildomų žinių, rašant atskiroms platformoms programas, ir atliekant jų transformavimą, pildant jas iki galutinės realizacijos.

6. ĮŽVALGOS IR TOLESNĖS SISTEMOS PLĖTOJIMO GALIMYBĖS

Šiame skyriuje pateikiama informacija apie duomenis ir jų ryšius, kurie nepateko į pagrindinį eksperimentinį tyrimą, bei pateikiamos numatomos sistemos plėtojimo galimybes ir siūlymai. Sistemos prototipo realizacijos metu ir atliekant tyrimą buvo pastebėti gaunami papildomi duomenų sąryšiai, kurie gali būti naudingi tolesnei sistemos plėtrai. Taip pat šiame skyriuje pateikiami papildomi statistiniai duomenys.

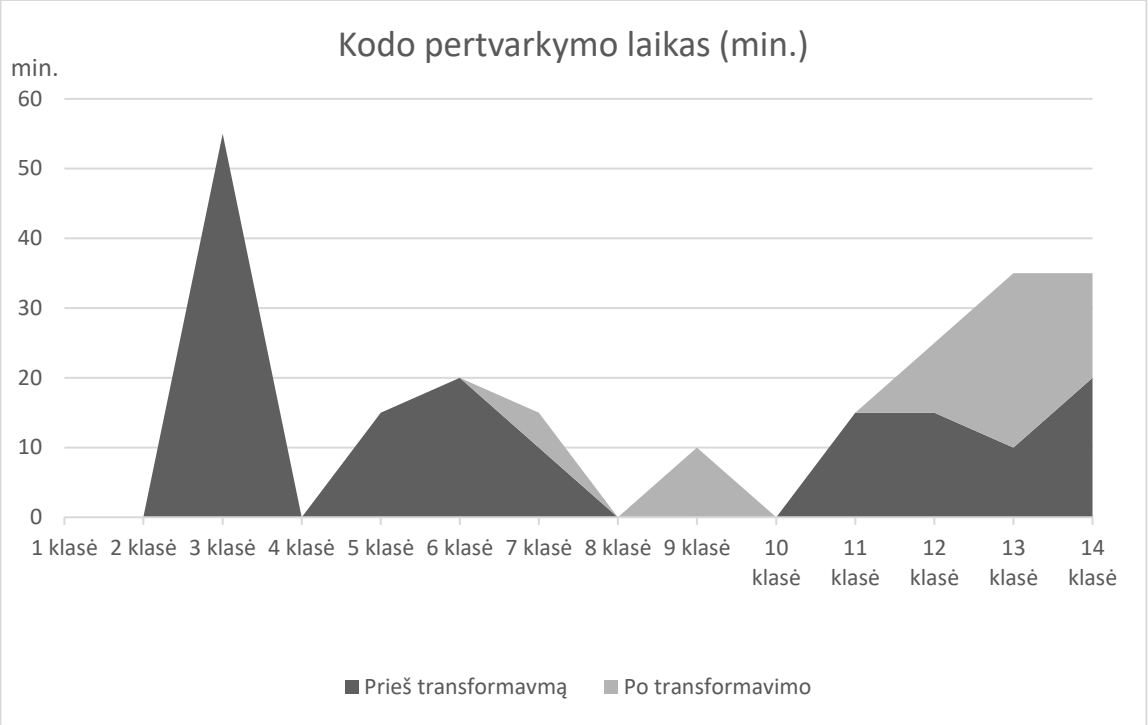
6.1. Išeities kodo transformavimo laikas

Eksperimento metu atlikti išeities kodo pertvarkymo laiko matavimai prieš ir po kodo transformavimo, bei pačio transformavimo. Visų klasių transformavimo laikai yra labai trumpi, todėl galima teigti, kad konvertuojant kodą iš vienos aplinkos į kitą yra sunaudojamą mažiau resursų, tačiau kai kurios klasės reikalavo kodo pertvarkymo po arba prieš transformavimą.

12 lentelė Kodo transformavimo laikai

Klasė	**Prieš transformavimą kodo pertvarkymo laikas**	**Transformavimo laikas**	**Po transformavimo kodo pertvarkymo laikas**	**Bendras laikas**
1 klasė	0 min.	1 319 ms	0 min.	< 1 min.
2 klasė	0 min.	1 879 ms	0 min.	< 1 min.
3 klasė	55 min.	4 101 ms	0 min.	55 min.
4 klasė	0 min.	2 547 ms	0 min.	< 1 min.
5 klasė	15 min.	3 789 ms	0 min.	15 min.
6 klasė	20 min.	4 919 ms	0 min.	20 min.
7 klasė	10 min.	6 321 ms	5 min.	15 min.
8 klasė	0 min.	4 012 ms	0 min.	< 1 min.
9 klasė	0 min.	6 419 ms	10 min.	10 min.
10 klasė	0 min.	7 091 ms	0 min.	< 1 min.
11 klasė	15 min.	8 979 ms	0 min.	15 min.
12 klasė	15 min.	12 171 ms	10 min.	25 min.
13 klasė	10 min.	12 278 ms	25 min.	35 min.
14 klasė	20 min.	13 893 ms	15 min.	35 min.

Bendram programinio kodo pertvarkymo laikui esant didesniam nei 30 min. skaitoma, kad transformavimas nėra efektyvus. Tačiau tokie laikai pagrinde pastebėti esant aukštesniam McCabe sudėtingumo dydžiui.



21 pav. Kodo pertvarkymo laikai

6.2. Sistemos plėtojimo perspektyvos

Šiuo metu sistemoje yra didesnis klaidų grįžimo skaičius, nei pageidaujama. Todėl viena iš prototipo tobulinimo krypčių būtų „Refactor“ modulio tobulinimas, iš kurio klaidų grįžimo skaičius ir yra didžiausias. Vien šio modulio tobulinimas leistu sistemai pateikti tikslesnius transformuotus išeities kodus ir juos sėkmingai transformuoti neįsikišant vartotojui.

Kita sistemos plėtros kryptis būtų gerinti transformavimo taisyklių naudojimą, kadangi po transformacijos teko kelias klases pertvarkyti, kad įvyktų sėkmingas kompiliavimas. Taip pat įsitikinome, kad sukurtos bendrinės AST ir ASG struktūros panaudojimas visiškai nedaro arba nedaro žymių pokyčių sistemos vykdymo laikui.

7. IŠVADOS

(1) Analizės metu buvo buvo nuspręsta, kurie algoritmai tinkami projektui. Pasirenkami buvo Pradiniam išeities kodo paruošimui naudojama ASG grafų technologija. Tikslesnei kodo analizei, pasitelkiama AST grafų technologija.

(2) Abstrakčios sintaksės medžiai negali atvaizduoti bendras programavimo išraiškas, dėl grafo struktūros. Šis medžio paprastumas kainuoja efektyvumo sąnaudas, kadangi reikalingas atskiras besikartojančių sąlygų atvaizdavimas atskirais mazginiais taškais. Dėl šios priežasties pirma kuriamas tarpinis ASG grafas, kad palengvinti vėlesnį AST grafo generavimą.

(3) Eksperimento metu nebuvo nei vienos klasės, kurios nepavyktu pataisyti, kad įvyktų sėkmingas kompiliavimas po išeities kodo transformavimo. Galima teigti, kad sistema veikia teisingai.

(4) Atliekant eksperimentą nustatyta, kad išeities kodo eilučių skaičius gali padidėti iki dviejų kartų. Tačiau buvo ir pavyzdžių kuomet kodo eilučių skaičius tarp pradinės programos klasės ir transformuotos buvo nežymus.

(5) Viena iš prototipo tobulinimo krypčių būtų „Refactor“ modulio tobulinimas, iš kurio klaidų grįžimo skaičius ir yra didžiausias. Vien šio modulio tobulinimas leistu sistemai pateikti tikslesnius transformuotus išeities kodus ir juos sėkmingai transformuoti neįsikišant vartotojui.

8. SANTRUMPŲ IR TERMINŲ ŽODYNAS

Išeities kodas	Programavimo kalbos tekstas (kodas) skirtas žmogui, suprasti kokius veiksmus vykdys kompiuteris, tiek pat skirtas kompiuteriui, transliuoti programos kodą į kompiuterio vykdomą (mašininį) kodą.
AST	Abstrakčios sintaksės medis (angl. Abstract Syntax Tree - AST).
ASG	Abstrakčios semantikos grafas (ASG angl. Abstract semantic graph).
SSA	Statinė vienos užduoties forma (SSA angl. Static single assignment form).
Kompiliavimas	Programos išeities tekstų (kodo) vertimas į skaitmeninį (mašininį) kodą.
MDA	Modeliais pagrįsta architektūra (angl. Model-Driven Architecture).
LOC	Kodo eilučių skaičius (angl. Lines of code).
UI	Naudotojo sąsaja (angl. *user interface, UI*) – visuma aparatinių ir programinių priemonių, sudarančių kompiuterio naudotojui patogias sąlygas valdyti operacinę sistemą ir taikomąsias programas..
API	Aplikacijų programavimo sąsaja (angl. Application Programming Interface, API) – tai sąsaja, kurią suteikia kompiuterinė sistema, biblioteka ar programa tam, kad programuotojas per kitą programą galėtu pasiekti jos funkcionalumą ar apsikeistu su ja duomenimis.
Kodo konvertavimas	Išeities kodo, kuris yra pritaikytas vienai įrangai, perrašymas, kad būtų skirtas kitai įrangai.
Mobiliosios programos	Taikomoji programinė įranga, skirta išmaniesiems telefonams, planšetiniams kompiuteriams ir kitiems mobiliesiems įrenginiams.

iOS	Kompanijos „Apple“ sukurta operacinė sistema, pagrinde projektuota „iPhone“ išmaniajam telefonui, o vėliau pritaikyta ir kitiems „Apple“ įrenginiams, įskaitant „iPod Touch“, „iPad“ ir „Apple TV“.
Android	Atviro kodo operacinė sistema, daugiausia naudojama išmaniuosiuose telefonuose, nors ją galima įdiegti ir kituose mobiliuosiuose įrenginiuose, kaip kad planšetiniame kompiuteryje.
Programos (sistemos) biblioteka	Sistemos arba programos elementas leidžiantis išplėsti funkcionalumą.
Platforma	Mobilaus įrenginio operacinė sistema, ar jos komponentai, fizinės detalės skirtos mobiliam įrenginiui funkcionuoti.
Specifikacija	Dokumentas aprašantis sistemos projektą ir jos atliekamas funkcijas.
Refactoring	Kodo pertvarkymas. Tai toks programinės įrangos pakeitimo procesas, kuris nepakeičia išorinio programos kodo elgesio, tačiau pagerina jo vidinę struktūrą

9. LITERATŪROS SĄRAŠAS

[1] "Types of compilers" [žiūrėta 2017-03-20]. Prieiga per internetą: http://www.compilers.net/paedia/compiler/index.htm

[2] "On source-to-source compilers" [žiūrėta 2017-03-25]. Prieiga per internetą: http://cyberleninka.ru/article/n/on-source-to-source-compilers.pdf

[3] "JUniversal: A Microsoft Tool for Porting Android Apps to Windows Phone and iOS" [žiūrėta 2017-04-03]. Prieiga per internetą: https://www.infoq.com/news/2015/02/juniversal

[4] "Java to iOS Objective-C translation tool and runtime" [žiūrėta 2017-04-03]. Prieiga per internetą: http://j2objc.org

[5] „ROSE“ [žiūrėta 2017-04-07]. Prieiga per internetą: http://rosecompiler.org/ROSE_HTML_Reference/index.html

[6] Athanasios Konstantinidis "Source-to-Source Compilation of Loop Programs for Manycore Processors" [žiūrėta 2017-04-12]. Prieiga per internetą: https://www.doc.ic.ac.uk/~phjk/PhDTheses-LocalCopies/ThanasisThesisFinalVersion.pdf

[7] Jurgen Jordanus Vinju "Analysis and Transformation of Source Code by Parsing and Rewriting" [žiūrėta 2017-03-25]. Prieiga per internetą: http://homepages.cwi.nl/~jurgenv/papers/thesis-2005.pdf

[8] Nicolas Juillerat "Models and Algorithms for Refactoring Statements" University of Fribourg 2009 [žiūrėta 2017-04-03]. Prieiga per internetą: https://diuf.unifr.ch/main/pai/sites/diuf.unifr.ch.main.pai/files/publications/2009_Juillerat_Thesis.pdf

[9] Rodrigo E. Caballero "A Graph-Based Algorithm for Automated Refactoring" [žiūrėta 2017-04-03]. Prieiga per internetą: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.1889&rep=rep1&type=pdf

[10] Richard Soley and the OMG Staff Strategy Group „Model Driven Architecture“ 2000 [žiūrėta 2017-04-11] Prieiga per internetą: http://www.geocities.ws/pravin_suman/Resources/00-11-05.pdf

[11] Kenneth Zadeck „The Development of Static Single Assignment Form“ 2008 [žiūrėta 2017-04-30] Prieiga per internetą: http://citi2.rice.edu/WS07/KennethZadeck.pdf

[12] Edward Duffy „The Design & Implementation of an Abstract Semantic Graph for Statement-Level Dynamic Analysis of C++ Applications“ [žiūrėta 2017-04-28] Prieiga per internetą: http://tigerprints.clemson.edu/cgi/viewcontent.cgi?article=1832&context=all_dissertations

[13] Kerievsky, Joshua „Refactoring to Patterns“ 2004 [žiūrėta 2017-04-07].

[14] Suryanarayana, Girish „Refactoring for Software Design Smells“ 2014 [žiūrėta 2017-04-08].

[15] Martin Fowler „Refactoring: Improving the design of existing code“ 1999. [žiūrėta 2017-05-05]

[16] Martin Fowler „Refactoring techniques“ [žiūrėta 2017-05-05] Prieiga per internetą: https://refactoring.com/catalog/index.html

[17] Chris Lattner, Vikram Adve „LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation“ 2004 [žiūrėta 2017-04-10] Prieiga per internetą: http://wwwi10.lrr.in.tum.de/~gerndt/home/Teaching/HPCSeminar/llvm_lifelong_program_analysispdf.pdf

[18] „McCabe's Cyclomatic Complexity“ [žiūrėta 2017-05-05] Prieiga per internetą: http://staff.unak.is/andy/staticanalysis0809/metrics/cyclomatic_complexity.html

[19] „Weighted Methods per Class (WMC)“ [žiūrėta 2017-05-05] Prieiga per internetą: http://staff.unak.is/andy/staticanalysis0809/metrics/wmc.html

[20] Alexander Serebrenik „Software metrics“ [žiūrėta 2017-05-07] Prieiga per internetą: http://www.win.tue.nl/~aserebre/2IS55/2012-2013/10.pdf

[21] Programų evoliucija. [Žiūrėta 2017-04-15] Prieiga per internetą: http://www.mif.vu.lt/~ragaisis/PSI_inf2012/SE-11-Evolution.pdf

[22] Programinės įrangos inžinerija. Įvadas. [Žiūrėta 2017-04-15] Prieiga per internetą: http://www.techmat.vgtu.lt/konspektai/PSI/psi_01_ivadas.pdf

10. PRIEDAI

6 pavyzdinės klasės išeities kodas:

```
package com.afollestad.aestheticsample;

import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.support.v7.widget.SwitchCompat;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.Spinner;

import com.afollestad.aesthetic.Aesthetic;
import com.afollestad.aesthetic.BottomNavBgMode;
import com.afollestad.aesthetic.BottomNavIconTextMode;

import butterknife.BindView;
import butterknife.ButterKnife;
import butterknife.OnClick;
import butterknife.Unbinder;

public class MainFragment extends Fragment {

  @BindView(R.id.switch_theme)
  SwitchCompat switchThemeView;

  @BindView(R.id.spinner)
  Spinner spinnerView;

  private Unbinder unbinder;

  @Nullable
  @Override
  public View onCreateView(
      LayoutInflater inflater, @Nullable ViewGroup container,
            @Nullable Bundle savedInstanceState) {
    return inflater.inflate(R.layout.fragment_main, container, false);
  }

  @Override
  public void onViewCreated(View view, @Nullable Bundle
savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);
    unbinder = ButterKnife.bind(this, view);

           Aesthetic.get()
          .activityTheme(R.style.AppTheme)
          .textColorPrimaryRes(R.color.text_color_primary)
          .textColorSecondaryRes(R.color.text_color_secondary)
          .colorPrimaryRes(R.color.md_white)
          .colorAccentRes(R.color.md_blue)
          .colorStatusBarAuto()
          .colorNavigationBarAuto()
          .textColorPrimary(Color.BLACK)
          .navigationViewMode(NavigationViewMode.SELECTED_ACCENT)
```

```
            .bottomNavigationBackgroundMode(BottomNavBgMode.PRIMARY)

.bottomNavigationIconTextMode(BottomNavIconTextMode.SELECTED_ACCENT)
            .apply();

    // Update the dark theme switch to the last saved isDark value.
    Aesthetic.get().isDark().take(1).subscribe(isDark ->
switchThemeView.setChecked(isDark));

    // Further view setup
    ArrayAdapter<String> spinnerAdapter =
        new ArrayAdapter<>(
            getContext(),
            R.layout.list_item_spinner,
            new String[] {
              "Spinner One",
              "Spinner Two",
              "Spinner Three",
              "Spinner Four",
              "Spinner Five",
              "Spinner Six"
            });

spinnerAdapter.setDropDownViewResource(R.layout.list_item_spinner_dropdow
n);
    spinnerView.setAdapter(spinnerAdapter);
  }

  @Override
  public void onDestroyView() {
    unbinder.unbind();
    super.onDestroyView();
  }

  @OnClick(R.id.switch_theme)
  public void onThemeChange(SwitchCompat switchCompat) {
    if (switchCompat.isChecked()) {
      Aesthetic.get()
          .activityTheme(R.style.AppThemeDark)
          .isDark(true)
          .textColorPrimaryRes(R.color.text_color_primary_dark)
          .textColorSecondaryRes(R.color.text_color_secondary_dark)
          .apply();
    } else {
      Aesthetic.get()
          .activityTheme(R.style.AppTheme)
          .isDark(false)
          .textColorPrimaryRes(R.color.text_color_primary)
          .textColorSecondaryRes(R.color.text_color_secondary)
          .apply();
    }
  }

  @OnClick({
    R.id.btn_black,
    R.id.btn_red,
    R.id.btn_purple,
    R.id.btn_blue,
    R.id.btn_green,
    R.id.btn_white
```

```
  })

  public void onClickButton(View view) {
    switch (view.getId()) {
      case R.id.btn_black:
        Aesthetic.get()
            .colorPrimaryRes(R.color.text_color_primary)
            .colorAccentRes(R.color.md_purple)
            .colorStatusBarAuto()
            .colorNavigationBarAuto()
            .bottomNavigationBackgroundMode(BottomNavBgMode.PRIMARY_DARK)

.bottomNavigationIconTextMode(BottomNavIconTextMode.BLACK_WHITE_AUTO)
            .apply();
        break;
      case R.id.btn_red:
        Aesthetic.get()
            .colorPrimaryRes(R.color.md_red)
            .colorAccentRes(R.color.md_amber)
            .colorStatusBarAuto()
            .colorNavigationBarAuto()
            .bottomNavigationBackgroundMode(BottomNavBgMode.PRIMARY_DARK)

.bottomNavigationIconTextMode(BottomNavIconTextMode.BLACK_WHITE_AUTO)
            .apply();
        break;
      case R.id.btn_purple:
        Aesthetic.get()
            .colorPrimaryRes(R.color.md_purple)
            .colorAccentRes(R.color.md_lime)
            .colorStatusBarAuto()
            .colorNavigationBarAuto()
            .bottomNavigationBackgroundMode(BottomNavBgMode.PRIMARY_DARK)

.bottomNavigationIconTextMode(BottomNavIconTextMode.BLACK_WHITE_AUTO)
            .apply();
        break;
      case R.id.btn_blue:
        Aesthetic.get()
            .colorPrimaryRes(R.color.md_blue)
            .colorAccentRes(R.color.md_pink)
            .colorStatusBarAuto()
            .colorNavigationBarAuto()
            .bottomNavigationBackgroundMode(BottomNavBgMode.PRIMARY_DARK)

.bottomNavigationIconTextMode(BottomNavIconTextMode.BLACK_WHITE_AUTO)
            .apply();
        break;
      case R.id.btn_green:
        Aesthetic.get()
            .colorPrimaryRes(R.color.md_green)
            .colorAccentRes(R.color.md_blue_grey)
            .colorStatusBarAuto()
            .colorNavigationBarAuto()
            .bottomNavigationBackgroundMode(BottomNavBgMode.PRIMARY_DARK)

.bottomNavigationIconTextMode(BottomNavIconTextMode.BLACK_WHITE_AUTO)
            .apply();
        break;
      case R.id.btn_white:
```

```
        Aesthetic.get()
            .colorPrimaryRes(R.color.md_white)
            .colorAccentRes(R.color.md_blue)
            .colorStatusBarAuto()
            .colorNavigationBarAuto()
            .bottomNavigationBackgroundMode(BottomNavBgMode.PRIMARY)

.bottomNavigationIconTextMode(BottomNavIconTextMode.SELECTED_ACCENT)
            .apply();
        break;
    }
  }
}
```

1 pavyzdinės klasės išeities kodas:

```
package com.controller;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import com.service.ContactService;

@WebServlet("/add-contact")
public class AddContact extends HttpServlet {
          private static final long serialVersionUID = 1L;

          protected void doPost(HttpServletRequest request,
HttpServletResponse response)
                                throws ServletException, IOException {
                      String name = request.getParameter("name");
                      String address = request.getParameter("address");
                      String phoneno = request.getParameter("phoneno");
                      ContactService contactService = new
ContactService();
                      contactService.addContact(name, address, phoneno);
                      HttpSession session = request.getSession();
                      session.setAttribute("contactList",
contactService.getAllContacts());
                      response.sendRedirect("index.jsp");

          }

          protected void doGet(HttpServletRequest request,
HttpServletResponse response)
                                throws ServletException, IOException {
                      ContactService contactService = new
ContactService();
                      HttpSession session = request.getSession();
                      session.setAttribute("contactList",
contactService.getAllContacts());
                      response.sendRedirect("index.jsp");
                                }
}
```