KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

GINTAUTAS MOTIEJŪNAS

TESTINIŲ DUOMENŲ GENERAVIMO ALGORITMŲ ANALIZĖ IR TYRIMAS

Magistro darbas

Darbo vadovas
dr. Š. Packevičius

Kaunas, 2014

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

GINTAUTAS MOTIEJŪNAS

TESTINIŲ DUOMENŲ GENERAVIMO ALGORITMŲ ANALIZĖ IR TYRIMAS

Magistro darbas

Darbo vadovas
dr. Š. Packevičius

_____ _____

Recenzentas
dr. A. Ušaniov

_____ _____

Studentas
G. Motiejūnas

2014-05-16 _____

Kaunas, 2014

AUTENTIŠKUMO PATVIRTINIMAS

Patvirtinu, kad įteikiamas baigiamasis magistro darbas „Testinių duomenų generavimo algoritmų analizė ir tyrimas“:

1. Autoriaus atliktas savarankiškai, jame nėra pateikta kitų autorių medžiagos kaip savos, nenurodant tikrojo šaltinio.
2. Nebuvo to paties autoriaus pristatytas ir gintas kitoje mokymo įstaigoje Lietuvoje ar užsienyje.
3. Nepateikia nuorodų į kitus darbus, jeigu jų medžiaga nėra naudota darbe.
4. Pateikia visą naudotos literatūros sąrašą.

Gintautas Motiejūnas	2014-05-16	
(studento vardas, pavardė)	(data)	(parašas)

SUMMARY

Currently in software development more and more attention is paid to its quality. One of the most frequently used methods for quality assurance and improvement of software is testing. Since it is very time and money consuming process (might take over 50% of the total project time), everyone tries to automate it. One of the biggest problems of test automation is test data generation. To solve it, there is created a lot of different data generation methods, which are being constantly improved. Not all methods are equally suitable for generating test data for various applications. Because of this it was decided to investigate the suitability of several data generation methods for various specifics programs.

For this work there was chosen and examined 4 dynamic methods: Random, Hill Climbing, Simulated Annealing and Genetic. It was also decided to compare the performance of the methods with different code coverage criteria (lines of code, instructions and branches). For this research was designed and developed Eclipse IDE plugin that allows generation of test data for selected Java class methods using data generation methods being researched. This plugin is also able to compare them with each other. In the conclusions presented experimental study results argues that the appropriate data generation method can save up to 43% of generation time. Some material of the study and its findings has been presented during the present conference “IVUS 2014”.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLĖLIŲ SĄRAŠAS

1. ĮVADAS

1.1. Dokumento paskirtis

Šis darbas yra autoriaus magistratūros studijų baigiamasis darbas, kuris reprezentuoja, jo įgytas žinias bei gebėjimus šių studijų metu. Darbas priklauso Kauno technologijos universiteto magistratūros studijų programų sistemų inžinerijos programai.

1.2. Santrauka

Šiuo metu kuriant programinę įrangą vis didesnis dėmesys yra skiriamas jos kokybei. Vienas iš dažniausiai naudojamų kokybės užtikrinimo ir kėlimo būdų yra programinės įrangos testavimas. Kadangi tai labai imlus laikui bei pinigams procesas (gali užimti virš 50% viso projekto laiko), jį stengiamasi automatizuoti. Viena iš didžiausių testavimo automatizavimo problemų yra testinių duomenų generavimas. Jai spręsti yra sukurta daug įvairių duomenų generavimo metodų, kurie yra pastoviai tobulinami. Ne visi metodai yra vienodai tinkamai generuoti testinius duomenis įvairioms programoms. Todėl buvo nuspręsta ištirti kelių generavimo metodų tinkamumą įvairios specifikos programoms.

Šio darbo metu tokiam tyrimui buvo pasirinkti ir išanalizuoti 4 dinaminiai metodai: atsitiktinis generatorius, kopimo į kalną bei modeliuojamuoju atkaitinimu pagrįsti metodai ir genetinis generatorius. Taip pat buvo nuspręsta palyginti metodų veikimą su skirtingais kodo padengimo kriterijais (kodo eilučių, instrukcijų bei šakų). Tyrimui atlikti buvo suprojektuotas ir sukurtas Eclipse programavimo aplinkos plėtinys, kuris leidžia pasirinktiems Java klasių metodams generuoti testinius duomenis tiriamais metodais bei geba juos palyginti tarpusavyje.

Darbo išvadose pateikiami atlikto eksperimentinio tyrimo rezultatai teigia, jog tinkamas duomenų generavimo metodas gali sutaupyti net iki 43% generavimo laiko. Dalis atlikto tyrimo ir jo metu gautų išvadų buvo pristatyti mokslinėje konferencijoje „IVUS 2014“.

1.3. Darbo tikslas ir uždaviniai

Šio darbo tikslas yra išanalizuoti skirtingais metodais paremtus testinių duomenų generatorius ir palyginti jų tinkamumą įvairios specifikos programoms testuoti. Tikslui pasiekti buvo iškelti tokie pagrindiniai uždaviniai:

1. Susipažinti su testinių duomenų generavimo metodais.
2. Pasirinkti kelis populiariausius metodus ir juos realizuoti.
3. Sukurti programinę įrangą leidžiančią tarpusavyje palyginti realizuotų generavimo metodų veikimo efektyvumą.
4. Palyginti pasirinktų metodų tinkamumą įvairios specifikos programoms.
5. Palyginti metodų veikimą su skirtingais kodo padengimo kriterijais.

1.4. Darbo struktūra

Šis darbas susideda iš įvado, trijų pagrindinių darbo dalių, darbo išvadų, literatūros sąrašo, terminų ir santrumpų žodyno bei priedų:

- Įvade pateikiama trumpa darbo santrauka, darbo tikslas bei jo struktūra.
- Probleminės srities analizės skyriuje yra apžvelgiamas automatinis testinių duomenų generavimas bei analizuojami tyrimui pasirinkti generavimo metodai. Taip pat apžvelgiami panašų funkcionalumą, kaip darbo metu kuriamas įrankis, teikiantys sprendimai.
- Testinių duomenų generatoriaus projekto dalyje yra aprašomas sukurtas duomenų generavimo metodų palyginimo įrankis. Pateikiami įrankiui keliami reikalavimai, esminiai jo architektūros aspektai bei atliktos testavimo procedūros, kuriomis siekiama užtikrinti įrankio kokybę.

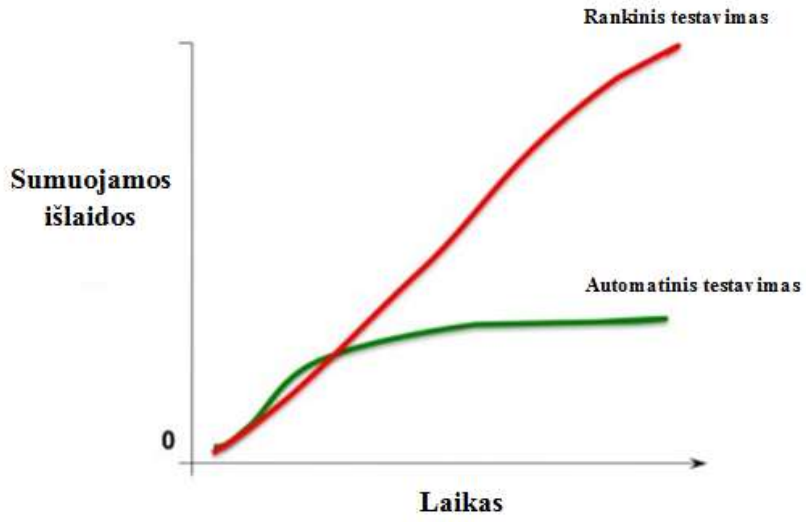
- Eksperimentinio tyrimo dalyje aprašoma atlikto eksperimento eiga bei pateikiami jo metu gauti rezultatai.
- Apibendrinimo ir išvadų skyriuje apibendrinami atlikti darbai ir pateikiamos suformuluotos pagrindinės viso darbo išvados.
- Literatūros sąraše pateikiami visi darbe paminėti bei cituoti informacijos šaltiniai.
- Terminų ir santrumpų žodyne pateikiamos pagrindinės darbe naudotos sąvokos.
- Prieduose pateikiami mokslinėje konferencijoje pristatyti straipsniai, kuriuose panaudota su šiuo darbu susijusi medžiaga.

2. PROBLEMINĖS SRITIES ANALIZĖ

2.1. Programinės įrangos testavimas

Šiuo metu kuriant programinę įrangą vis didesnis dėmesys yra skiriamas jos kokybei. Nebėra žiūrima vien tik į programos funkcionalumą (kuo jis didesnis, kuo programa atlieka daugiau funkcijų – tuo ji yra geresnė), bet vertinama ir jos kokybė (kaip gerai programa atlieka savo funkcijas, kaip efektyviai yra panaudojami turimi resursai, kokia saugi ji yra ir t.t.).

Vienas iš labiausiai naudojamų kokybės užtikrinimo ir kėlimo būdų yra programinės įrangos testavimas. Tai procesas arba jų serija, kurie yra sukurti įsitikinti, jog parašytas programos kodas atlieka būtent tai, kas buvo numatyta jį rašant ir neatlieka to, kas nebuvo numatyta [1]. Testavimas yra labai imlus laikui bei pinigams procesas ir kuriant programinę įrangą gali užimti daugiau negu 50% viso numatyto laiko [2]. Dėl šios priežasties testavimą siekiama įvairias būdais automatizuoti. Nors testavimo procedūra ją automatizuojant gali užtrukti žymiai ilgiau, nei vykdant ją paprastai, automatizuotas testavimas atsiperka jį pakartotinai panaudojant daug kartų (1 pav.) [3]. Vykdant rankinį testavimą testuotojams reikalingos pastovios išlaidos visą testavimo atlikimo trukmę. Taigi bendra testavimo savikaina auga tiesiškai nuo testavimo laiko. Grafike tai parodo kreivė „Rankinis testavimas“. Tuo tarpu automatizuojant testavimo procesą iš pradžių tam reikalingos didesnės lėšos, tačiau baigus kurti automatinius testus, jų palaikymas tampa žymiai pigesnis ir bendra savikainos testavimui kreivė pasidaro žymiai lėkštesnė – išlaidos žymiai sumažėja. Grafike tai atspindi kreivė „Automatinis testavimas“.



1 pav. Išlaidos testavimui

Automatinis testavimas lyginant su rankiniu pranašesnis yra tuoj, jog jis žymiai lengviau leidžia pridėti naujus testavimo atvejus bei žymiai paprasčiau yra įvykdyti testą iš naujo. Automatinio testavimo rezultatai yra labiau vaizdesni ir suprantami paprastam vartotojui, negu rankinio. Bet to naudojant automatinį testavimą tikimybė, jog testuojant bus padaryta klaida sumažėja iki nulinės (testas gali būti klaidingas tik tuo atveju, jei klaida įvelta kuriant patį testą) [4].

Automatinis testavimas susiduria ir su daug problemų: orakulo problemą [5][6], automatinis testinių duomenų generavimas [7][8], automatinis testinių objektų generavimas [9] [10]. Viena opiausių iš jų yra automatinis testinių duomenų generavimas, kuriam yra sukurta daug įvairių metodų ir jie visą laiką yra tobulinami. Metodus testinių duomenų generavimui galima suskirstyti į dvi klases: statinius metodus ir dinaminius metodus. Statiniai metodai remiasi simboliniu kodo vykdymu, vykdomo kodo aprėpiamos srities mažinimu bei kitomis

metodologijomis [11][12]. Jie testinius duomenis generuoja nevykdydami pilno programos kodo, todėl susiduria su problemomis, kai testuojamų programų apimtys didėja ir didėja jų sudėtingumas.

Dinaminiai metodai testinius duomenis generuoja iteracijomis. Kiekvienos iteracijos metu yra vykdomas testuojamos programos kodas su iteracijos metu sugeneruotais duomenimis. Įvykdžius kodą yra įvertinami sugeneruoti testai ir pagal tai vykdoma nauja iteracija. Tai tęsiam, kol pasiekiamas minimalus tenkintinas testų įvertis arba kitas generavimo metodo nutraukimo kriterijus. Testų įvertis skaičiuojamas įvairiais būdais. Du populiariausi iš jų yra: įvertinimas pagal aptiktų klaidų kiekį (tam naudojamas mutacinis testavimas [13][14]) ir įvertinimas pagal kodo padengimą. Šio darbo metu nagrinėjamas pastarasis įvertinimo būdas, kadangi norint jį panaudoti pilnai užtenka tik sugeneruotų testinių duomenų. Tuo tarpu įvertinimas pagal aptiktų klaidų kiekį reikalauja pilnų testinių rinkinių, kurie be testinių duomenų apima ir laukiamus testų rezultatus (testo orakulas). Kodo padengimo įvertis taip pat gali būti skaičiuojamas įvairias būdais. Plačiau apie jį kalbama skyriuje 2.3 „Padengimo kriterijus“.

Dinaminiai duomenų generavimo metodai apima atsitiktinius [15], paremtus vietine paieška [16][17][18], į tikslą orientuotus [16][19], grandinėlės principu veikiančius [16] bei evoliucinius generatorius [16][20]. Pastarieji testinius duomenis generuoja vykdydami testuojamų programų kodą ir remiantis testų rezultatais optimizuoja jų generavimą. Jie išvengia daugumos problemų, su kuriomis susiduria statiniai metodai. Dėl pačios testinių duomenų generavimo problemos sudėtingumo ne visi generavimo metodai yra vienodai tinkamai testuojamoms programos. Jų tinkamumas priklauso nuo programų specifikos.

2.2. Duomenų generavimo metodai

Plačiai naudojami dinaminiai duomenų generavimo metodai apima atsitiktinį generatorių, Hill Climbing metodą, Simulated Annealing metodą, genetinius algoritmus, tabu paiešką, skruzdžių kolonijos optimizaciją, dirbtinio intelekto sistemas, sklaidančiąją paiešką, bei evoliucines strategijas. Atliktame darbe buvo nuspręsta nagrinėti keturis iš jų: atsitiktinis generatorius (RA), Hill Climbing (HC), Simulated Annealing (SA) bei standartinis genetinis algoritmas (GA). Algoritmų aprašymuose, parametrai, nuo kurių priklauso jų veikimas yra išryškinti kursyvu.

2.2.1. Atsitiktinis generatorius

RA yra paprasčiausias testinių duomenų generavimo algoritmas. Jis atsitiktinai generuoja duomenis ir sustoja, kai yra pasiekiamas problemos sprendinys arba peržengiami algoritmo vykdymo rėžiai. Toks generatorius dažnai naudojamas kaip atskaitos taškas įvertinti kitų, sudėtingesnių generatorių efektyvumą.

RA veikimą galima aprašyti 5 žingsniais.

1. Atsitiktiniu būdu sugeneruojamas pradinis sprendinys.
2. Atsitiktiniu būdu sugeneruojamas antrasis sprendinys.
3. Jei antrasis sprendinys yra geresnis už pradinį sprendinį, antrąjį sprendinį padarome pradiniu.
4. Grįžtama į antrą žingsnį, kol pasiekiamas maksimalus iteracijų kiekis arba problemos sprendimas jau rastas (nagrinėjamu atveju pasiektas minimalus tenkintinas padengimas).
5. Pradinis sprendinys yra geriausias rastas sprendinys.

2.2.2. Hill Climbing

HC yra optimizacijos euristika, besiremianti vietine paieška. HC veikimą galima aprašyti 7 žingsniais.

1. Atsitiktiniu būdu sugeneruojamas pradinis sprendinys.
2. Sugeneruojamas gretimas sprendinys.
3. Jei gretimas sprendinys yra geresnis, gretimą sprendinį padarome pradiniu.

4. Grįžtama į antrą žingsnį, kol išnagrinėjami visi gretimi sprendiniai arba problemos sprendimas jau rastas (nagrinėjamu atveju pasiektas minimalus tenkintinas padengimas).
5. Jei išnagrinėti visi gretimi sprendiniai, tačiau minimalus tenkintinas sprendinys dar nerastas, pasirenkamas naujas pradinis sprendinys.
6. Grįžtama į antrą žingsnį, kol pasiekiamas maksimalus iteracijų kiekis arba problemos sprendimas jau rastas (nagrinėjamu atveju pasiektas minimalus tenkintinas padengimas.)
7. Pradinis sprendinys yra geriausias rastas sprendinys.

Iš algoritmo žingsnių matome, jog jo veikimas priklauso nuo strategijos, pagal kurią yra pasirenkamas gretimas sprendinys bei strategijos, pagal kurią pasirenkamas naujas pradinis sprendinys. Paprasčiausia gretimo sprendinio pasirinkimo strategija yra iš jau esamo sprendinio atsitiktinai pasirinkti vieną testuojamo metodo parametro reikšmę ir prie jos dvejetainės reikšmės pridėti arba atimti vieną bitą. Tai atliekant taip pat yra atsižvelgiama į parametro galimų reikšmių rėžius.

Tuo tarpu naujo pradinio sprendinio pasirinkimui dažniausiai yra naudojama atsitiktinė strategija, kai sprendinys pasirenkamas iš visų galimų variantų kaip pirmajame algoritmo žingsnyje.

2.2.3. Simualated Annealing

SA yra globalios optimizacijos euristika, kuri remiasi vietine paieška. SA metu yra vykdoma paieška, kurios pirmasis sprendinys yra pasirenkamas atsitiktinai. Visi po to einantys paieškos sprendiniai yra pasirenkami iš sprendinių gretimų prieš tai nagrinėtam sprendiniu. Tam yra naudojama tokia pati strategija kaip ir HC atveju. Pasirinktų sprendinių tinkamumas yra apsprendžiamas pagal tikimybę kuri priklauso nuo parametro vadinamo temperatūra (1).

$$P(c_0, c, t) = exp(-(c - c_0)/t); \quad (1)$$

čia c_0 – pradinio sprendinio padengimas, c – gretimo sprendinio padengimas, t - temperatūra

Pradinė temperatūros reikšmė yra didelė ir beveik visi paieškos žingsniai yra tinkami. Vykdant algoritmą temperatūros reikšmė yra pastoviai mažinama dauginant ją iš temperatūros mažinimo koeficiento. Žingsniai duodantys geresnį sprendimą yra visą laiką tinkami. Tuo tarpu žingsnių duodančių sprendimą, kuris yra prastesnis arba lygus prieš tai esančiam, tinkamumas yra apsprendžiamas pagal tikimybę. Kuo mažesnė temperatūra, tuo mažesnė tikimybė, kad blogesnis sprendinys bus tinkamas. Ši algoritmo savybė leidžia išvengti lokalaus optimumo. Kai SA temperatūros reikšmė yra lygi nuliui, paieškos efektyvumas pasidaro lygus HC algoritmui.

SA algoritmą galima aprašyti 8 žingsniais:

1. Atsitiktiniu būdu sugeneruojamas pradinis sprendinys.
2. Sugeneruojamas gretimas sprendinys.
3. Jei gretimas sprendinys yra geresnis, gretimą sprendinį padarome pradiniu.
4. Jei gretimas sprendinys nėra geresnis už pradinį, atsitiktinai sugeneruojamas skaičius $x \in [0, 1]$ ir lyginamas su sprendinio tinkamumo tikimybės (1) rezultatu. Jei $x < P(c0, c, t)$, tada pradinis sprendinys pakeičiamas gretimu, priešingu atveju pradinis sprendinys lieka nepakitęs.
5. Grįžtama į antrą žingsnį, kol pasiekiamas maksimalus vidinių iteracijų kiekis, išnagrinėti visi gretimi sprendiniai arba problemos sprendimas jau rastas (nagrinėjamu atveju pasiektas minimalus tenkintinas padengimas).
6. Temperatūra padauginama iš temperatūros mažinimo koeficiento.
7. Grįžtama į antrą žingsnį, kol pasiekiamas maksimalus iteracijų kiekis, išnagrinėti visi gretimi sprendiniai arba problemos sprendimas jau rastas (nagrinėjamu atveju pasiektas minimalus tenkintinas padengimas).
8. Pradinis sprendinys yra geriausias rastas sprendinys.

2.2.4. Standartinis genetinis algoritmas

Standartinis GA remiasi ląstelių kryžminimosi gyvuose organizmuose – miozės procesu. Gyvame organizme esančias chromosomas šiame algoritme atstoja dvejetainiai skaičiai (pavyzdžiui 101111001). Jie sudaro populiaciją. Populiacijos narių tinkamumą kryžminimui apibūdina tam tikra algoritmo funkcija. Vykstant algoritmui pagal išrinkimo funkciją, nusakančią, kuriuos populiacijos narius kryžminti, išrenkami du atskiri populiacijos individai, kurie po to yra kryžminami pagal kryžminimo funkciją ir vėl išskiriami. Galiausiai gauti du nauji individai mutuojami pagal mutavimo funkciją. Visas procesas yra kartojamas apibrėžtą kiekį kartų, kol yra pasiekiamas tam tikras pabaigimo kriterijus. Toliau detaliau aptariamos algoritme paminėtos funkcijos.

1) **Tinkamumo funkcija**. Tai funkcija skaičiuojanti populiacijos individo tinkamumo įvertį sprendžiamai problemai. Dažniausiai jos rezultatai priklauso nuo ankstesnės algoritmo iteracijos rezultatų. Darbe narinėjamų atvejų tinkamumo įvertis yra lygus paskutiniosios iteracijos metu pasiektai kodo padengimo kriterijaus reikšmei.

2) **Išrinkimo funkcija.** Tai funkcija išrenkanti porą populiacijos individų tolimesniam kryžminimui. Ši funkcija dažniausiai priklauso nuo tinkamumo funkcijos rezultatų. Individo tikimybė būti išrinktam kryžminimui yra tuo didesnė, kuo didesnis individo tinkamumo įvertis. Darbe nagrinėjamu atveju yra atsitiktinai išrenkami du geriausią tinkamumo įvertį turintys individai.

3) **Kryžminimo funkcija**. Ši funkcija apibūdina tai, kaip genai (bitai) yra keičiami išrinktų individų poroje. Tai ar kiekvienos iteracijos metu pats kryžminimas įvyks ar ne apsprendžia kryžminimo tikimybė išreiškiama skaičiumi nuo 0 iki 1. Prieš kryžminimą yra sugeneruojamas atsitiktinis skaičius nuo 0 iki 1. Jei šis skaičius yra mažesnis už kryžminimo tikimybę, tada kryžminimas vykdomas, jei ne – yra gražinama nepakitusi individų pora. Darbe nagrinėjamu atveju yra naudojama paprasta ir standartinė kryžminimo funkcija. Atsitiktiniu būdu yra pasirenkamas individo bitas. Tada individo dalis nuo pasirinktojo bito (įskaitant ir jį patį) yra sukeičiama su atitinkamą kito poros individo dalimi. Pavyzdžiui turime du individus 101111001 ir 110101010. Atsitiktinai pasirinktas bito numeris nuo 1 iki 9 yra 5. Tada individų pora po kryžminimo atrodys taip: 101101010 ir 110111001.

4) **Mutavimo funkcija.** Tai funkcija apibrėžia kaip individai turi mutuoti. Mutavimui dažniausiai naudojama paprasta funkcija – pasirinkto bito invertavimas. Mutuojant individą kiekvienam jo bitui yra sugeneruojamas atsitiktinis skaičius nuo 0 iki 1, kuris yra lyginamas su mutacijos tikimybe ir skaičiui esant mažesniam už šią tikimybę individo bitas yra invertuojamas.

Remiantis aprašytomis funkcijomis standartinio GA veikimą galima apibūdinti 9 žingsniais:

1. Sugeneruojama populiaciją susidedančią iš n individų (*n* – populiacijos dydis).
2. Apskaičiuojami tinkamumo įverčiai visiems populiacijos individams (pirmosios iteracijos metu visų individų tinkamumo įvertis lygus 1).
3. Pasinaudojant išrinkimo funkcija išsirenkami du individai iš populiacijos (tėviniai individai). Pastarieji ištrinami iš populiacijos.
4. Pasinaudojant kryžminimo funkcija bei atsižvelgiant į kryžminimo tikimybę kryžminami tėviniai individai. Sukryžminti individai pavadinami vaikiniais individais.
5. Pasinaudojant mutavimo funkcija bei atsižvelgiant į mutavimo tikimybę vykdoma mutacija kiekvieno vaikinio individo bito atžvilgiu.
6. Vaikiniai individai įtraukiami į naują populiaciją.
7. Grįžtama į antrą žingsnį, kol nauja populiacija įgyja n arba daugiau individu.
8. Jei n yra nelyginis skaičius, ištrinamas atsitiktinis individas. Sena populiacija pakeičiama nauja populiacija.
9. Grįžtama į antrą žingsnį, kol pasiekiamas maksimalus populiacijų skaičius arba problemos sprendimas jau rastas (nagrinėjamu atveju pasiektas minimalus tenkintinas padengimas).

2.3. Padengimo kriterijus

Testuojamo kodo padengimas nusako, kuri kodo dalis atliekant testus buvo vykdoma, o kuri testų nebuvo paliesta. Pagal kodo padengimą galima nustatyti, kurioms programos dalims dar reikia parašyti papildomus testus. Procentinis kodo padengimo įvertis yra viena iš dažniausiai naudojamų testų įvertinimo metrikų. Jis parodo koks geras yra testinių atvejų rinkinys ir kokią dalį programos jis testuoja. Daugumoje kokybės siekiančių programavimo įmonių siekiama užtikrinti bent 85% testuojamos programos padengimo [21]. Kodo padengimas geli būti skaičiuojamas pagal įvairias kodo charakteristikas:

- kodo eilučių padengimas
- instrukcijų padengimas
- šakų padengimas
- sąlygų padengimas
- išvestiniai padengimo įverčiai.

Kodo eilučių padengimas. Tai dažniausiai naudojamas ir paprasčiausiai suprantamas kodo padengimo kriterijus. Jis remiasi testavimo metu įvykdytomis kodo eilutėmis. Grafiškai su JaCoCo biblioteka [21] atvaizduotas kodo eilučių padengimas pateikiamas 2 paveikslėlyje. Jame žaliai pažymėtos testų padengtos eilutės, raudonai – nepadengtos, o geltonai eilutės, kurios buvo padengtos tik iš dalies. Skaičiuojant procentinį kodo eilučių padengimą, tokios eilutės skirtingų padengimo skaičiavimo įrankių būna interpretuojamos skirtingai. Vieni įrankiai jas priskiria prie padegtų eilučių, kiti prie nepadengtų.

```
289    public static String IntegerToRomanNumeral(int input) {
290        if (input < 1 || input > 3999)
291            return "Invalid Roman Number Value";
292        String s = "";
293        while (input >= 1000) {
294            s += "M";
295            input -= 1000;
296        }
297        while (input >= 900) {
298            s += "CM";
299            input -= 900;
300        }
301        while (input >= 500) {
302            s += "D";
303            input -= 500;
304        }
305        while (input >= 400) {
306            s += "CD";
307            input -= 400;
308        }
309        while (input >= 100) {
310            s += "C";
311            input -= 100;
312        }
```

2 pav. Grafinis kodo eilučių padengimo atvaizdavimas

Instrukcijų padengimas. Tai padengimo kriterijus suteikiantis objektyvesnę informaciją nei padengtos kodo eilutės. Jis remiasi įvykdytomis instrukcijomis, taip yra išvengiama dalinio eilučių padengimo.

Šakų padengimas. Šis padengimo kriterijus dar vadinamas sprendimų padengimu. Jis remiasi programos kodo šakomis (programos srauto atsišakojimai if, switch sakiniuose ir pan.), kurios buvo įvykdytos testavimo metu.

Sąlygų padengimas. Šis padengimo kriterijus yra labai panašus į šakų padengimo kriterijų. Pagrindinis skirtumas yra tas, kad kiekvienai sąlygai būtini bent du testinių duomenų rinkiniai, kurie tenkintų abi galimas sąlygos reikšmes.

Išvestiniai padengimo įverčiai. Siekiant geriau reprezentuoti programos kodo padengimą yra sukurti išvestiniai padengimo kriterijai tokie kaip daugiasąlyginis padengimas (angl. *multiple condition coverage*) [18] arba modifikuotas sąlygos ir sprendimo padengimas (angl. *modified condition decision coverage (MCDC)*) [18][23].

Kuriamame įrankyje dėl naudojamos kodo padengimo skaičiavimo bibliotekos JaCoCo apribojimų bus naudojami tik kodo eilučių, instrukcijų bei šakų padengimo įverčiai.

2.4. Panašūs įrankiai

Egzistuoja daug komercinių įrankių, kurie geba generuoti testinius duomenis. Tam jie dažniausiai naudoja savo modifikuotus generavimo algoritmus, kurių dėl komercinių priežasčių nenorima atskleisti.

Akademiniame pasaulyje taip pat yra vykdoma daug tyrimų šioje srityje. Generavimo metodai yra nuolatos tobulinami ir ieškoma naujų sprendimo būdų. Tam, kad įvertinti patobulinimų pranašumą dažniausiai naudojami palyginimo įrankiai būna sukurti pačios akademinės bendruomenės ir būna neprieinami plačiajai publikai.

Dėl šių priežasčių niekur nematyti projekto, kuris butų atviras ir leistų palyginti keletą skirtingų testinių duomenų generavimo algoritmų.

Kaip dažniausiai sutinkami testų generavimo įrankių pavyzdžius galima paminėti: CodePro [24], Squish Coco [25], jPET [26][27]. Jų palyginimas su darbo metu sukurtu įrankiu TDMG pateikimas 1 lentelėje.

CodePro. Tai yra Google tiekiamas nemokamas Eclipse IDE plėtinys. Be automatinių JUnit testų generavimo ir kodo padengimo tyrimo jis atlieka statinę kodo analizę bei pateikia išsamias ataskaitas. Įrankis turi lengvai suprantamą vartotojo sąsają, dirba su Java programavimo kalba.

Squish Coco. Tai yra komercinis produktas orientuotas į C/C++ programų testų generavimą. Programa taipogi atlieka išsamią kodo padengimo testais analizę, nustato, kurie testai yra pertekliniai, suranda optimalius testų panaudojimo eiliškumus. Programa turi savo nuosavą IDE.

jPET. Tai nemokamas Eclipse IDE plėtinys galintis generuoti vienetų testus ir juos atlikti su automatiškai parinktais testų duomenis. Įrankis turi labai lengvai suprantamą vartotojo sąsaja bei daug galimybių. Deja, jis nebėra atnaujinamas.

1 lentelė Įrankių palyginimas

Parametras	CodePro	Squish Coco	jPet	TDMG
Išplėčiamumas	Nėra	Nėra	Didelis (atviro kodo)	Didelis (atviro kodo)
Perkeliamumas	Windows/Linux /Mac	Windows/Linux /Mac	Windows/Linux /Mac	Windows/Linux /Mac
Panaudojamumas	Lengvas	Vidutinis	Lengvas	Lengvas
Patvarumas	Didelis	Vidutinis	Didelis	Vidutinis
Priklausomumas nuo IDE	Eclipse	Nuosava	Eclipse	Eclipse

Parametras	CodePro	Squish Coco	jPet	TDMG
Kaina	Nemokamas	Mokamas	Nemokamas	Nemokamas
Palaikomos kalbos	Java	C/C++	Java	Java

2.5. Išvados ir apibendrinimas

1. Išanalizuoti metodai taikomi automatiniam testinių duomenų generavimui bei pasirinkti keturi metodai: atsitiktinis, „Hill Climbing“, „Simulated Annealing“ bei genetinis, kurie detaliau aptarti ir bus realizuoti kuriamame įrankyje.
2. Aptarti padengimo kriterijai, kurie yra naudojami dinaminiuose duomenų generavimo algoritmuose. Iš jų pasirinkti trys: kodo eilučių, instrukcijų bei šakų padengimo kriterijai, kurie bus naudojami atliekant metodų tyrimą. Pasirinkimą sąlygojo kodo padengimą skaičiuojančios bibliotekos apribojimai.
3. Buvo atlikta panašių įrankių analizė, kurios metu nebuvo rasta komercinių įrankių leidžiančių palyginti duomenų generavimą keliais skirtingais metodais. Tuo tarpu akademiniai įrankiai skirti metodų palyginimui dažniausiai būna prieinami tik mažai auditorijai.

3. TESTINIŲ DUOMENŲ GENERATORIAUS PROJEKTAS

3.1. Atsakomybių pasiskirstymas projekte

Testinių duomenų generatorius buvo projektuotas ir realizuotas dviejų magistrantų: Gintauto Motiejūno (GM) ir Tito Kuckio (TK). GM buvo atsakingas už:

- Eclipse IDE plėtinio funkcionalumą;
- Atskirų duomenų generavimo modulių integraciją su plėtiniu Eclipse IDE plėtiniu;
- Atsitiktinio duomenų generavimo modulį;
- Duomenų generavimo modulį paremtą standartiniu genetiniu algoritmu.

TK atsakingas už:

- Eclipse IDE plėtinio grafinę sąsają.
- Duomenų generavimo modulį paremtą kopimo į kalną (HC) algoritmu.
- Duomenų generavimo modulį paremtą modeliuojamojo atkaitinimo (SA) algoritmu.
- Kitus duomenų generavimo modulius, kurie šiame darbe nėra nagrinėjami.

3.2. Sistemos paskirtis

Sistema yra skirta automatiniam testinių duomenų generavimui, kurį galima atlikti remiantis įvairiais generavimo algoritmais. Tokia programinė įranga leis ne tik sugeneruoti duomenis testams, bet ir padės nustatyti, kokie generavimo algoritmai kokiose situacijose yra efektyviausi testuojant realias Sistemas. Tai, kartu su pačių algoritmų analize, leis juos tobulinti arba apjungiant kelių metodų stipriąsias sritis, išvesti naujus – optimaliau veikiančius tam tikrose taikymo srityse.

Galima išskirti tokius pagrindinius sistemos vartotojų tikslus, kurių įgyvendinimas ir yra pagrindinis sistemos uždavinys:

- Galimybė automatiškai sugeneruoti testinius duomenis pasirinktai Sistemai.
- Tai atlikti kuo greičiau ir kuo kokybiškiau (išsirenkant tinkamiausią generavimo algoritmą pagal programos kodo specifiką).
- Galimybė palyginti tinkamiausio varianto charakteristikas su kitais generavimo algoritmais.
- Paprastas ir intuityvus naudojimasis sukurtu produktu.

3.3. Reikalavimai sistemai

3.3.1. Funkciniai reikalavimai

- Sistema turi teikti galimybę sugeneruoti testinius duomenis vartotojo pasirinktam metodui vienu iš galimų duomenų generavimo metodu.
- Sistema turi prašyti nurodyti atskirą metodą, kuriam bus generuojami duomenis.
- Sugeneravus duomenis Sistema taip pat turi pateikti generavimo proceso bei sugeneruotų duomenų svarbiausias charakteristikas: generavimo laikas, kodo padengimas, sugeneruotų testinių atvejų skaičius.
- Sistema turi leisti pakartotinai tam tikrą kiekį kartų pergeneruoti duomenis ir kaupti bendrą jų statistiką.
- Sistema turi leisti generuoti testinius duomenis naudojantis atsitiktiniu generavimo metodu.
- Sistema turi leisti generuoti testinius duomenis naudojantis genetiniu generavimo metodu.
- Sistema turi leisti generuoti testinius duomenis naudojantis Hill Climbing generavimo metodu.
- Sistema turi leisti generuoti testinius duomenis naudojantis Simulated Annealing generavimo metodu.

- Sistema turi leisti pasirinkti kelis duomenų generavimo metodus, su kuriais vieno vykdymo metu bus sugeneruojami duomenys.
- Sistema turi generuoti kelių generavimo metodų palyginimo ataskaitą, sudarytą iš pavienių metodų generavimo charakteristikų ataskaitų. Joje taip pat turi būti pateikta bendra statistika ir nurodomi efektyviausi ir mažiausiai efektyvūs metodai.
- Sistema turi leisti peržiūrėti sugeneruotas ataskaitas standartiniame Eclipse lange tekstiniu formatu.
- Sistema turi leisti išsaugoti ataskaitas HTML formate.
- Sistema turi leisti išsaugoti ataskaitas paprastu tekstiniu formatu.
- Sistema turi leisti paprastai tvarkyti (pridėti naują arba pašalinti esamą) duomenų generavimo metodų modulius.
- Sistema turi leisti keisti atskirų generavimo metodų parametrus (jei jie tokiu turi), tokius kaip maksimalus iteracijų kiekis ir pan.
- Sistema turi kaupti bendrą naudojimo statistiką nuo pat programos naudojimo pradžios. Joje išskiriami įvairių charakteristikų atžvilgiu geriausi bei blogiausi generavimo metodai.
- Sistema turi leisti keisti formuojamų ataskaitų turinį – leisti pasirinkti kokios charakteristikos bus atvaizduojamos ataskaitose, o kokios ne.
- Sistema turi turėti pasirinkimą, generuojant testinius duomenis jų neišsaugoti faile, o baigus generavimą ir surinkus statistika juos ištrinti.

3.3.2. Nefunkciniai reikalavimai

- Intuityvi sąsaja.
- Iš Eclipse aplinkos neišsiskirianti sąsaja.
- Nesudėtingas meniu.
- Sistema įsisavinama be specialaus apmokymo.
- Sistemos universalumas kalbos atžvilgiu.
- Galimybė nutraukti testų generavimą.
- Sistema turi turėti pagalbą vartotojui.
- Sistema turi netrukdyti kitoms sistemoms, efektyviai naudoti resursus.
- Sistema turi atlikti užduotis kaip galima per greitesnį laiką, negali sukelti įtarimą vartotojui kad užduotis nevykdoma.
- Sistema turi leisti ją papildyti naujais testavimo komponentais.
- Paprastas produkto įdiegimas.
- Naujų Eclipse aplinkos versijų palaikymas.
- Sistema turi garantuoti, kad ja naudojantis net ir įvykus klaidai nebus sunaikinti vartotojo duomenys.
- Sistema turi užtikrinti, kad įdiegti nauji testavimo algoritmai negali padaryti jokios žalos nei pačiai sistemai, nei vartotojo duomenims.
- Bendros sistemos naudojimo statistikos išvalymo galimybė
- Programoje naudojami tekstai turi būti korektiški taisyklingi bei netrivialūs.
- Programa platinama kaip nemokama pagal GNU GPL licencija [28].

3.4. Architektūra

Šiame skyrelyje sistemos architektūra pateikiama keliais vaizdais, vaizduojančiais sistemą skirtingais aspektais: panaudojimo atvejų vaizdu, statiniu vaizdu, dinaminiu arba procesų vaizdu, ir išdėstymo vaizdu. Šie vaizdai yra pateikiami kaip Rational Rose modeliai ir juose naudojama unifikuota modeliavimo kalba (UML). Sistemos atvaizdavimui šiais vaizdais yra naudojamasi standartinėmis UML diagramomis:

- Panaudos atvejų vaizdas:
  - Panaudos atvejų diagrama.
- Statinis vaizdas:
  - Sistemos išskaidymo į paketus diagrama;
  - Klasių diagrama.
- Dinaminis (procesų) vaizdas:
  - Veiklos diagrama.
- Išdėstymo vaizdas:
  - Išdėstymo diagrama.

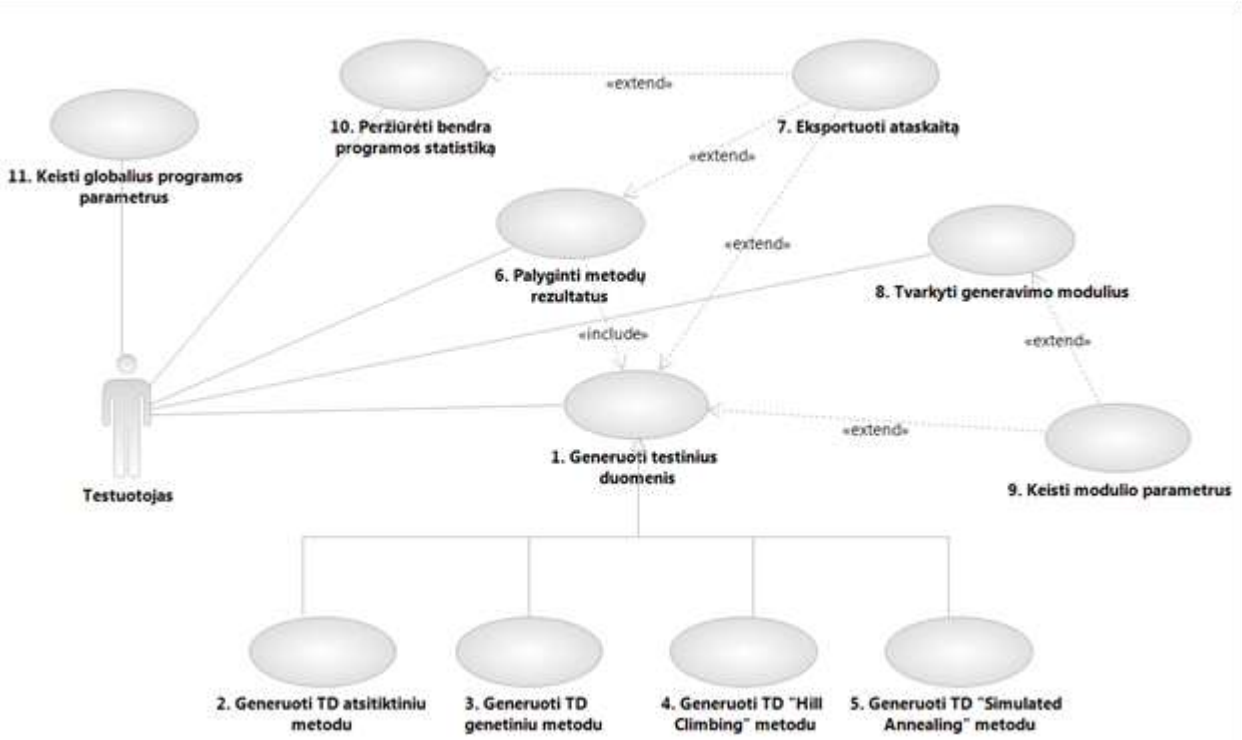
3.4.1. Architektūros tikslai ir apribojimai

Architektūrinius sprendimus įtakojantys reikalavimai ir apribojimai:

- Sistema turi būti kuriama kaip Eclipse IDE plėtinys.
- Papildinys neturi konfliktuoti su Eclipse IDE.
- Papildinio sąsaja turi pilnai integruotis į Eclipse IDE vartotojo sąsają ir iš jos neišsiskirti.
- Turi būti galimybė tiek generuoti duomenis vienu pasirinktu algoritmu, tiek ir keliais, tarpusavyje palyginant jų charakteristikas.
- Sugeneruotas ataskaitas turi būti galima išsaugoti įvairiais formatais.
- Sistemą turi būti įmanoma papildyti naujo duomenų generavimo algoritmo moduliu, pasinaudojant vien plėtinio vartotojo sąsaja.
- Sistema turi būti kuriama taip, kad jos funkcionalumą būtu galima lengvai išplėsti.

3.4.2. Panaudojimo atvejų vaizdas

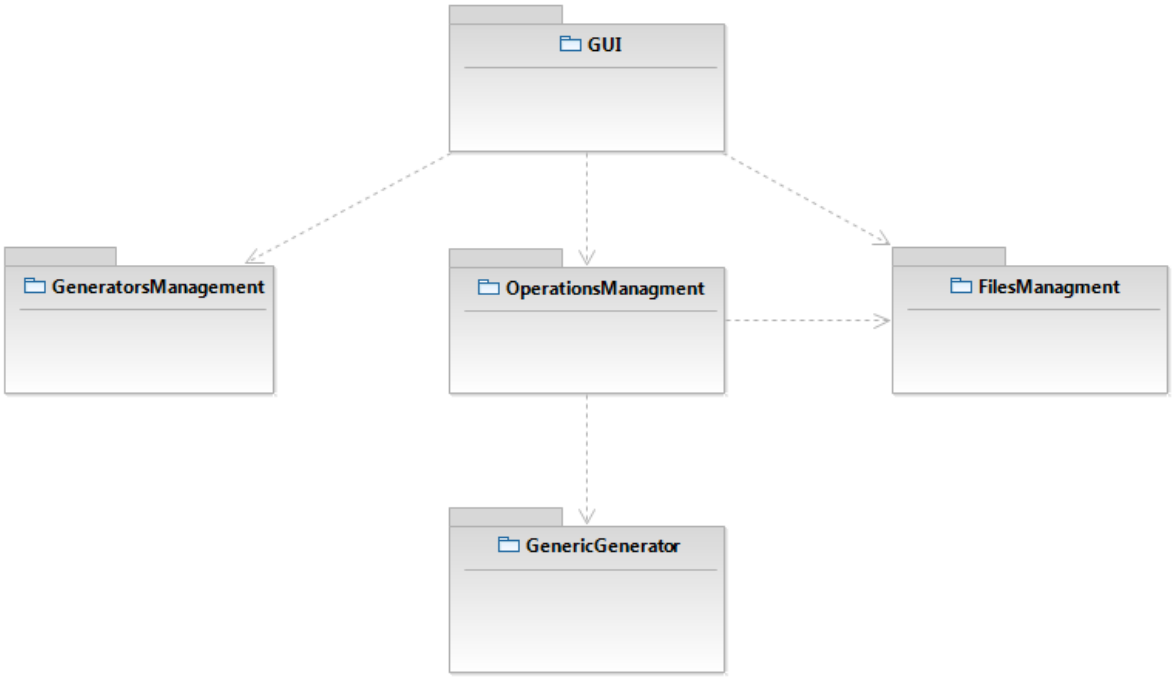
Sistemos panaudojimo atvejai pateikiami 3 paveikslėlyje.



3 pav. Panaudos atvejų diagrama

3.4.3. Sistemos statinis vaizdas

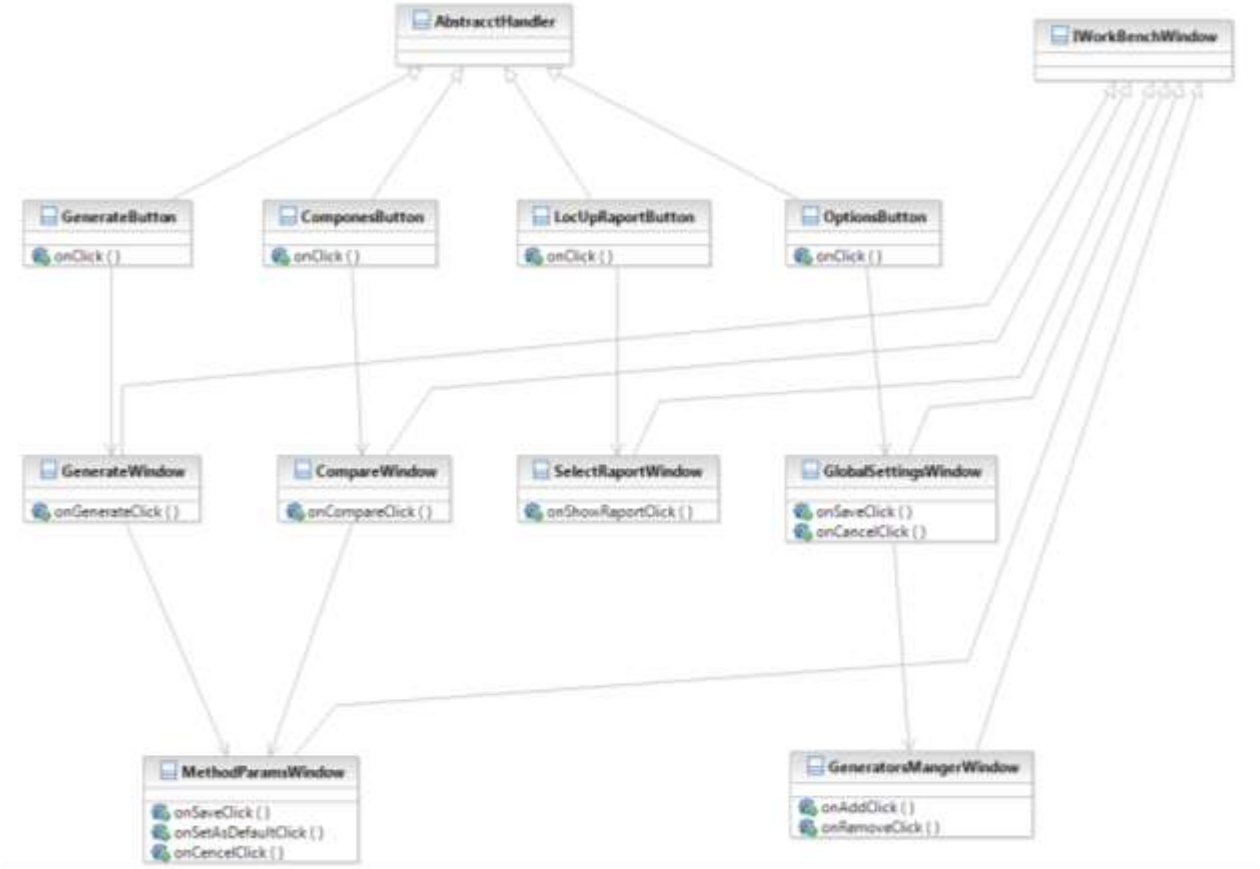
Sistemos loginę struktūrą galima suskirstyti į penkis pagrindinius paketus, kurie grafiškai pavaizduoti 4 paveikslėlyje.



4 pav. Sistemos suskaidymas į paketus

Paketas „GUI“

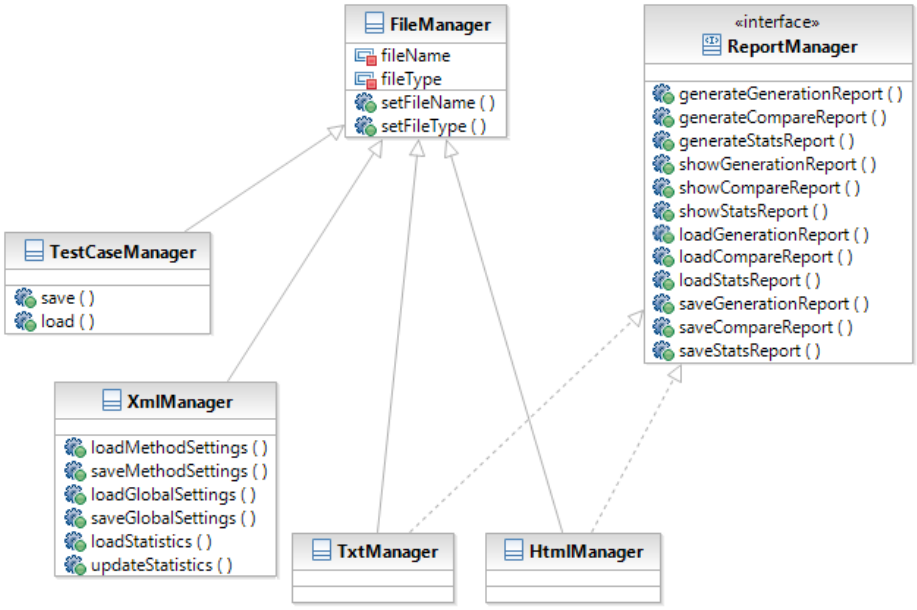
Šis paketas apima klases, kurios aprašo papildinio vartotojo sąsają. Pats papildinys neturi savo individualios grafinės vartotojo sąsajos. Bendravimui su vartotoju yra naudojama Eclipse IDE vartotojo sąsaja (valdymo mygtukai įterpiami į Eclipse IDE meniu, rezultatai atvaizduojami standartiniuose IDE languose ir t.t.). Tam pasiekti kuriamo papildinio sąsajos elementai paveldi reikiamas Eclipse IDE klases (AbstractHandler, IWorkBenhWindow). „GUI“ paketą sudarančios klasės pavaizduotos 5 paveikslėlyje.



5 pav. Paketo „GUI“ klasių diagrama

Paketas „FilesManagement“

Šis paketas apima klases, kurios dirba su failais. Tai apima sugeneruotų testinių duomenų išsaugojimą failuose bei jų užkrovimą iš jų, įvairių programos nustatymų saugojimą failuose XML formato failuose bei ataskaitų generavimą įvairiais failų formatais (kol kas tik TXT ir HTML). Įvertinant tai, jog ateityje ataskaitas gali tekti saugoti ir kitokių formatų failuose, yra suprojektuota ReportManager sąsajos klasė (angl. *interface*), kuri pateikia visą su ataskaitomis susijusį funkcionalumą. Norint pridėti naują ataskaitų formatą, pakanka sukurti klasę, kuri įgyvendiną minėtąją sąsajos klasę. „FilesManagement“ paketą sudarančios klasės yra pateikiamos 6 paveikslėlyje.

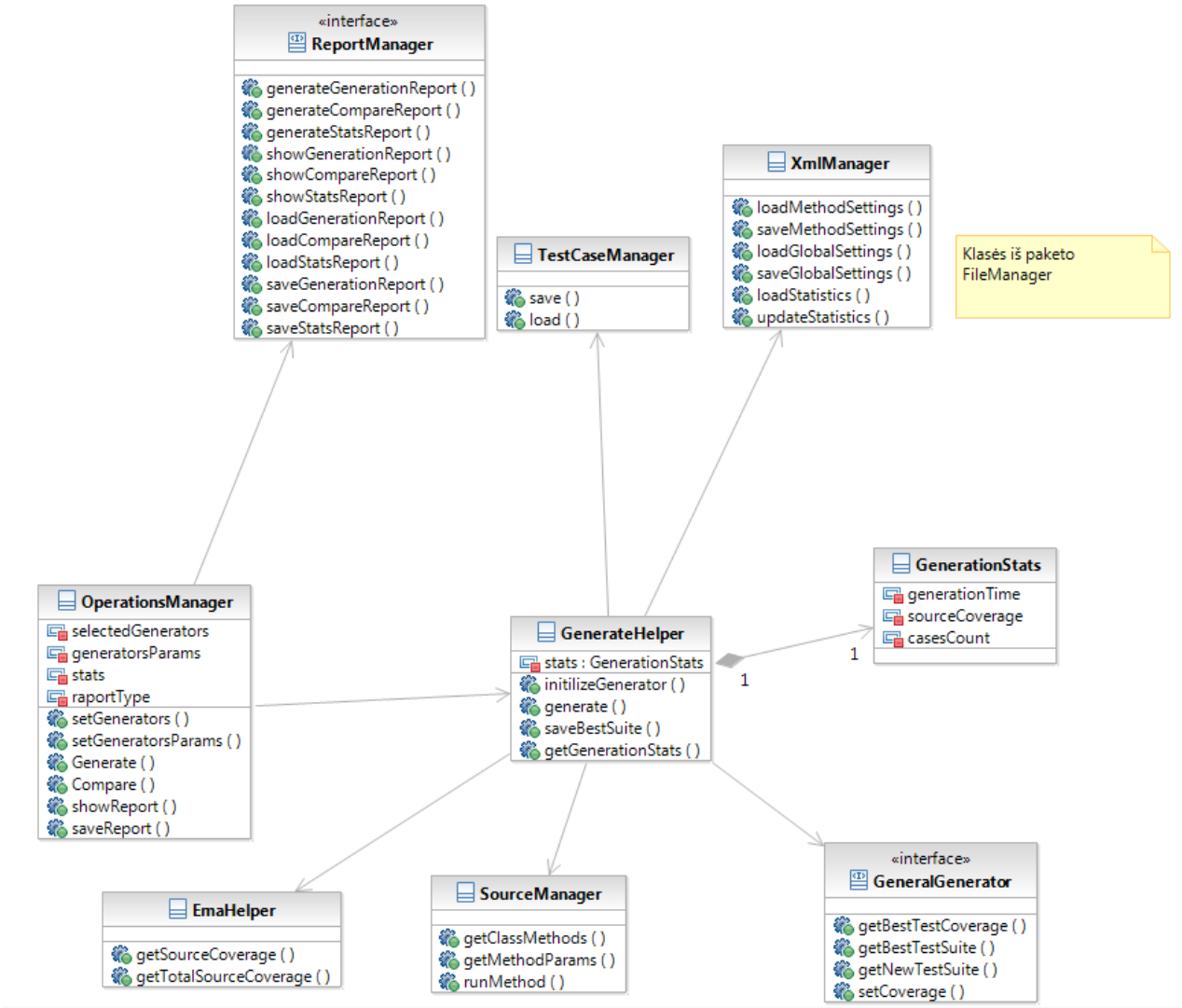


6 pav. Paketo „FilesManagement“ klasių diagrama

Paketas „OperationsManagement“

Šiame pakete yra pateikiamos klasės, nusakančios pagrindinį programos funkcionalumą, susijusį su testinių duomenų generavimu. Vartotojo veiksmai per „GUI“ paketą gali iškviesti tam tikras klasės „OperationsManager“ funkcijas, kurios naudodamos tiek vidines paketo klases, tiek ir klases iš paketų „FileManagement“ ir „GeneralGenerator“ įgyvendina norimą funkcionalumą. Siekiant aiškiau parodyti su kokiomis klasėmis „OperationsManager“ klasė sąveikauja, paketo klasių diagramoje (7) šalia paketo „OperationsManagment“ klasių, pateikiamos ir kelios susijusios paketo „FilesManagement“ klasės.

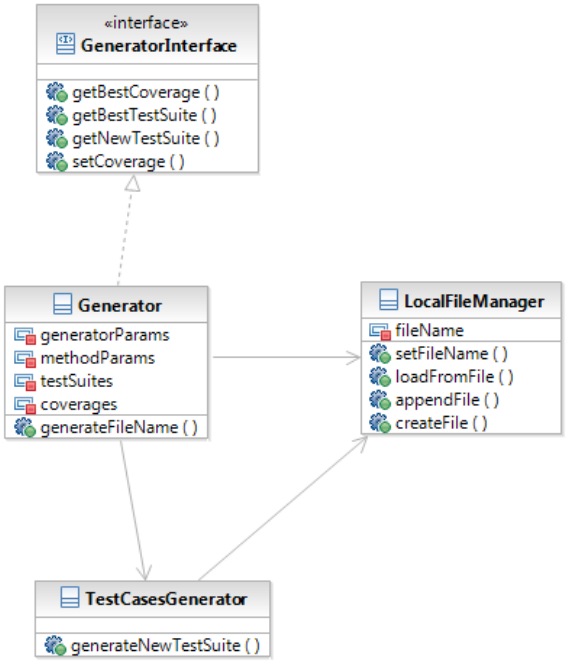
Tam, kad būtų įmanoma lengvai pridėti naują duomenų generavimo algoritmą, pakete yra suprojektuota sąsajos klasė GeneralGenerator, kuria privalo įgyvendinti kiekvienas duomenų generatorius.



7 pav. Paketo „OperationsManagement“ klasių diagrama

Paketas „GenericGenerator“

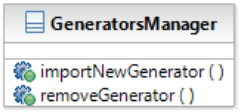
Tai nėra realus paketas, o tik jo atitikmuo. Kiekvienas testinių duomenų generavimo algoritmas bus realizuojamas kaip atskiras modulis, sudarytas iš keleto klasių, kuriose aprašyta generavimo logika. Toks modulis įgyvendins „OperationsManagement“ paketo „GenericGenerator“ sąsajos klasę ir jį bus galima paprastai pridėti arba atimti neperkompiliuojant visos programos. Toks išorinis modulis savo struktūra labai primena atskirą programos paketą, todėl jis ir yra nagrinėjamas kaip paketo atitikmuo. Visi moduliai savo struktūra turėtų būti panašus: turėti klasę, kuri paveldį minėtąją sąsajos klasę, klasę, kuri dirba su laikinais tik modulio naudojamais failais, bei esminį funkcionalumą atliekančią klasę, kurioje yra aprašytas testinių duomenų generavimo algoritmas. Tokio paketo struktūra pateikiama 8 paveikslėlyje.



8 pav. Paketo „GenericGenerator“ klasių diagrama

Paketas „GeneratorsManagement“

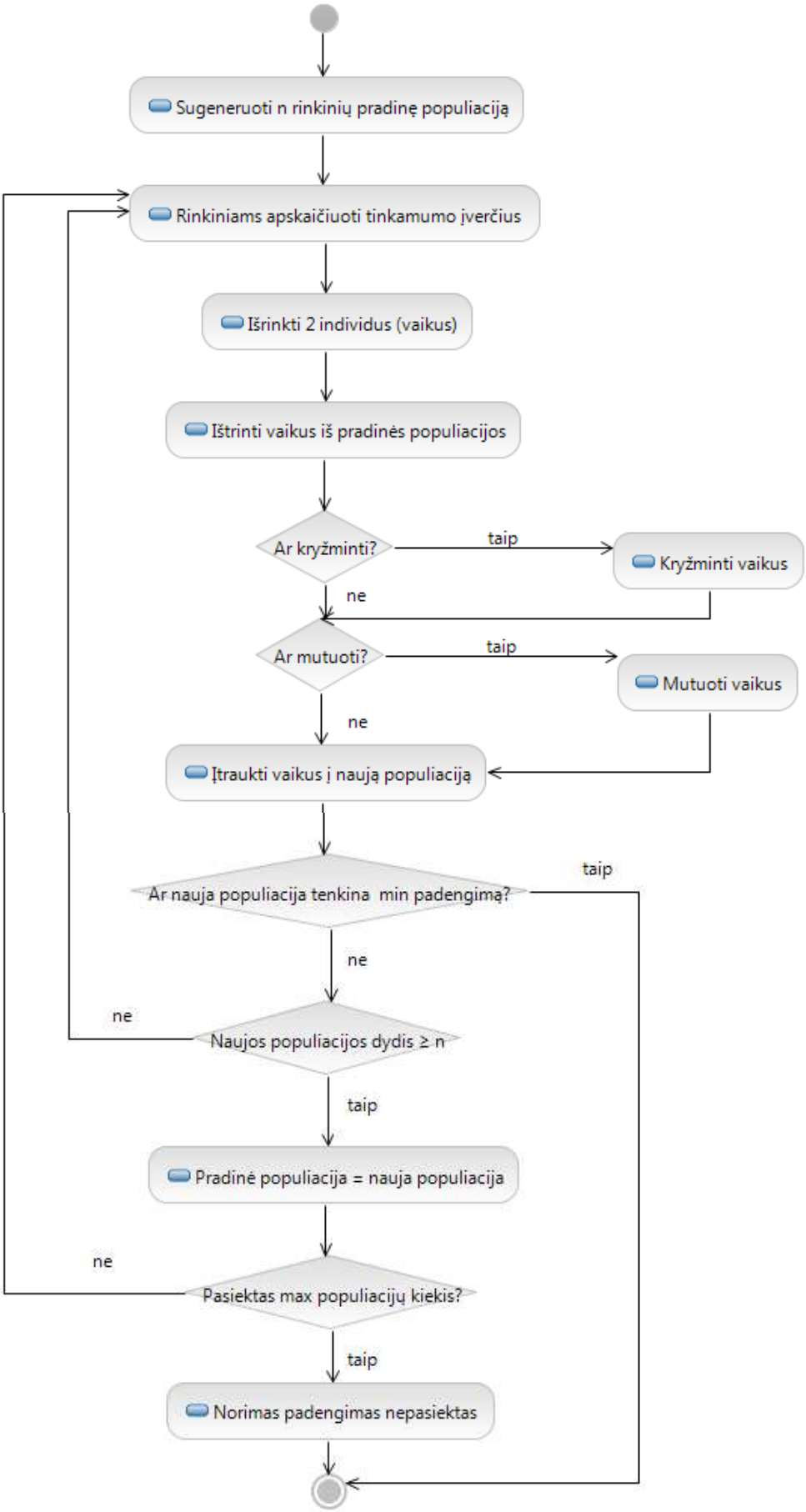
Šiame pakete yra pateikiama klasė valdanti anksčiau aptartus duomenų generatorių modulius. Jos teikiamas funkcionalumas leidžia paprastai pridėti naujus arba pašalinti esamus generatorius. Paketo klasių diagrama pateikiama 9 paveikslėlyje.



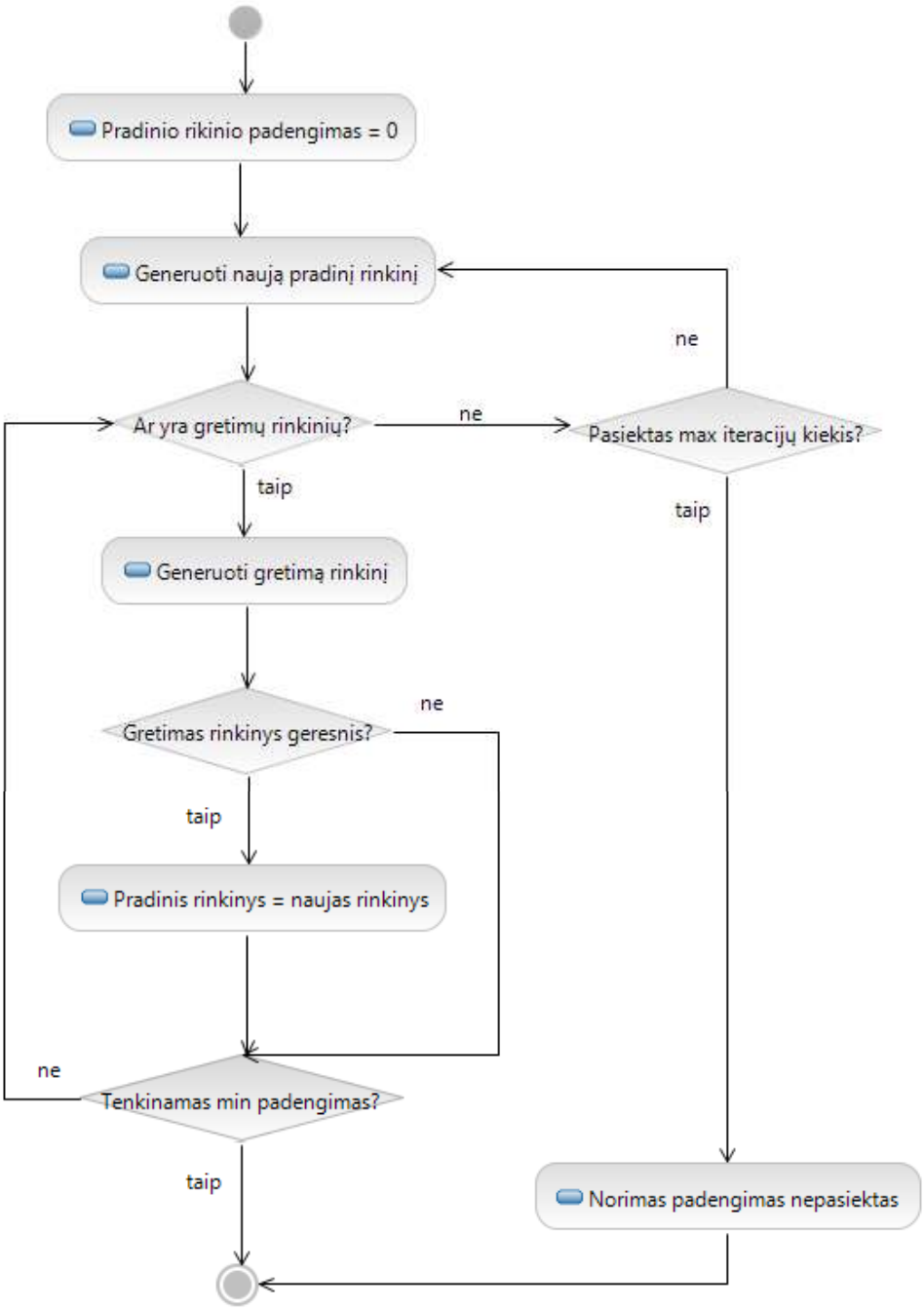
9 pav. Paketo „GeneratorsManagement“ klasių diagrama

3.4.4. Sistemos dinaminis vaizdas

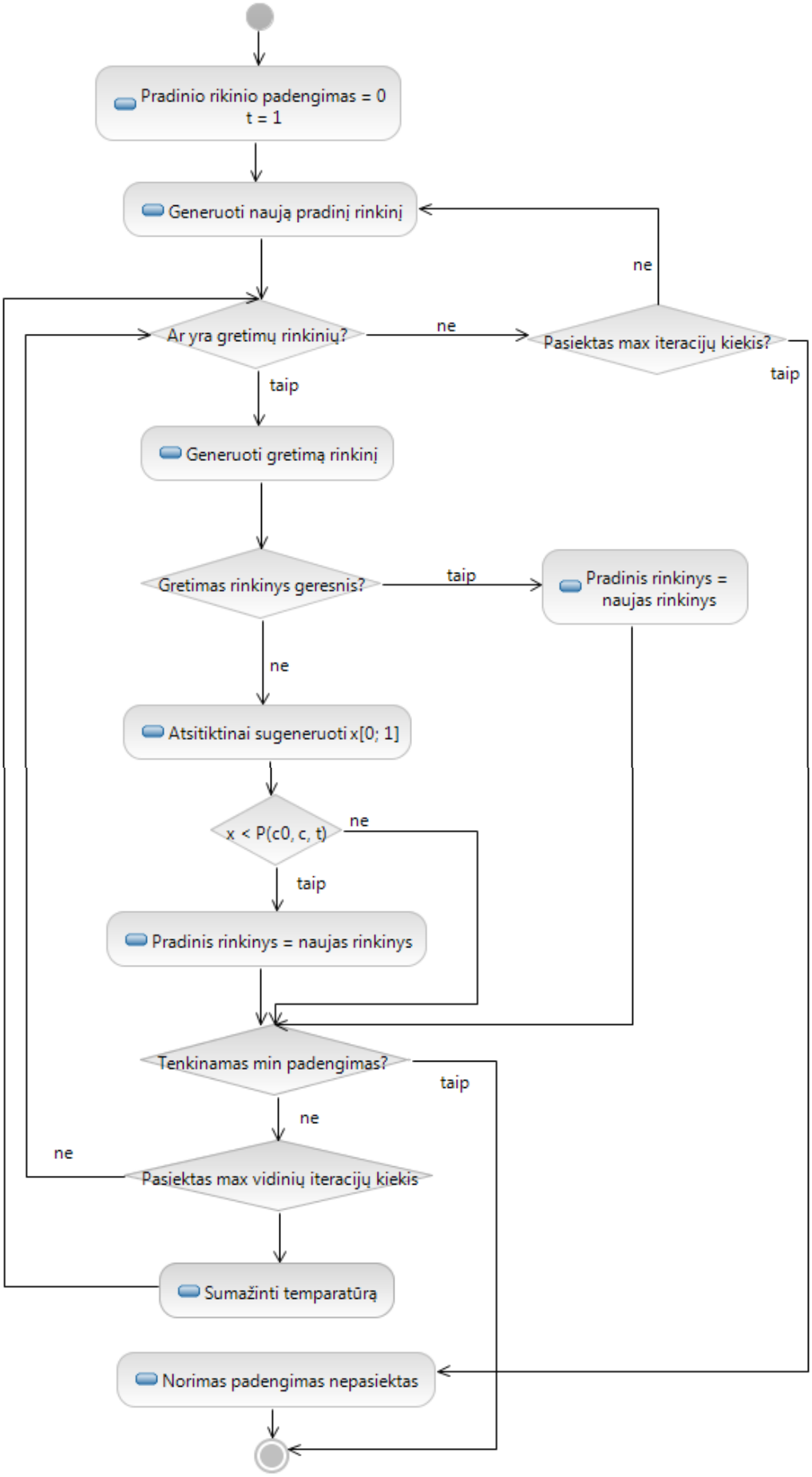
Šiame skyrelyje pateikiamos duomenų generavimo metodų veiklos diagramos: genetinio generatoriaus (10 pav.), Hill Climbing metodo (11 pav.), Simualted Annealing metodo (12 pav.), atsitiktinio generatoriaus (13 pav.).



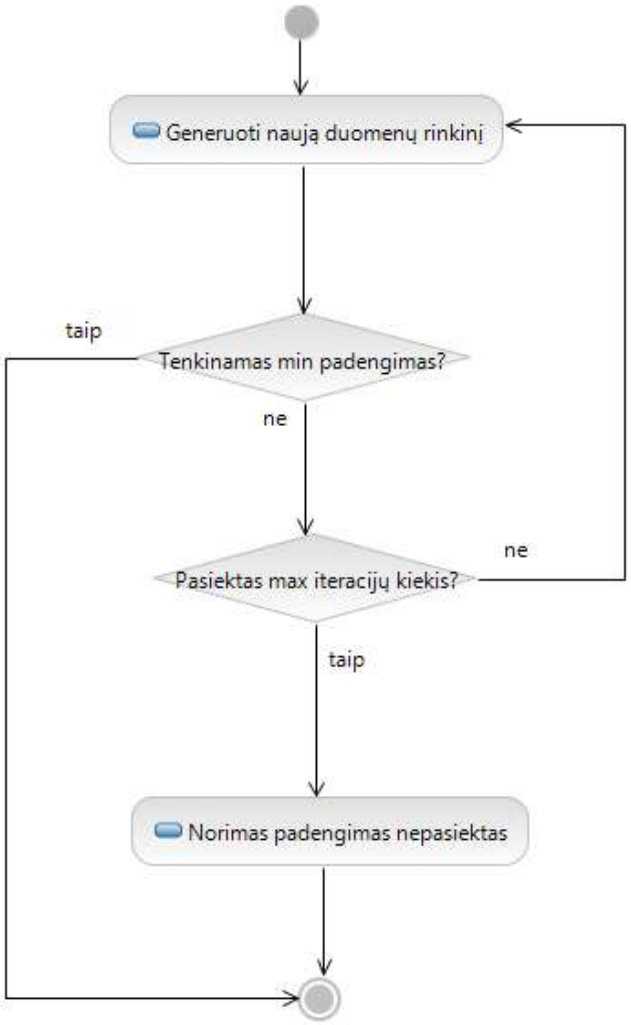
10 pav. Genetinio generatoriaus veiklos diagrama



11 pav. Hill Climbing generatoriaus veiklos diagrama



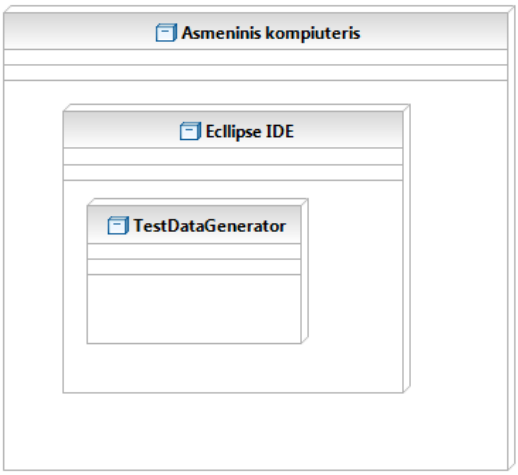
12 pav. Simulated Anneling generatoriaus veiklos diagrama



13 pav. Atsitiktinio generatoriaus veiklos diagrama

3.4.5. Išdėstymo vaizdas

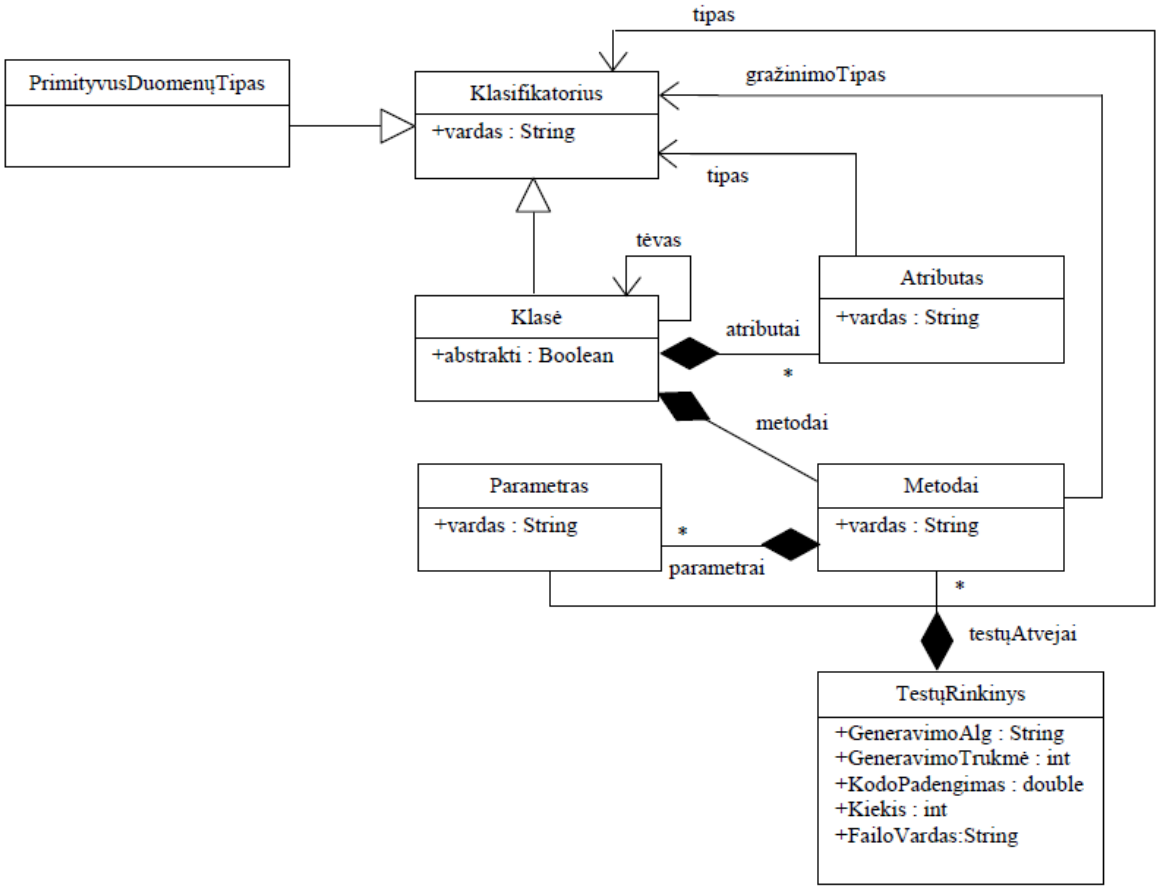
Testų generavimo sistema TDMG veikia asmeniniame kompiuteryje, kuriame jau yra įdiegta Eclipse IDE 4.2.2 (Juno) arba naujesnė jos versija. Plėtinio bendravimas su išore (vartoju) vyksta per Eclipse IDE sąsają. Sistemos išdėstymo vaizdas grafiškai pateikiamas 14 paveikslėlyje.



14 pav. Sistemos išdėstymo vaizdas

3.4.6. Duomenų vaizdas

Pagrindiniai duomenys, kuriais manipuliuojama sistemoje, yra testuojamos Java klasės metodai, šių metodų parametrų tipai bei reikšmės. Visą tai į duomenų struktūra apjungia UML metamodelis. Pastarasis jau yra realizuotas Eclipse IDE, todėl juo pasinaudojama ir aprašomame plėtinyje. Jo struktūra yra pateikiama 15 paveikslėlyje.



15 pav. Duomenų modelio schema

3.4.7. Kokybė

- Aprašyta sistemos architektūra, kurioje yra pasinaudojama sąsajos klasėmis (angl. *interface*), leidžia vėliau gan paprastai praplėsti jos funkcionalumą (pridėti naujų duomenų generavimo modulių, papildyti sąrašą formatų, kuriais gali būti generuojamos ataskaitos).
- Sistemos nustatymų saugojimas XML formato failuose, leidžia juos nesunkiai keisti arba reikalui esant išsaugoti visą nustatymų rinkinį ir jį užkrauti kitame kompiuteryje veikiančioje programoje.
- Sistemos bendravimui su vartotoju naudojamos paveldėtos Eclipse IDE klasės. Tai padidina kuriamo papildinio integraciją į Eclipse vartotojo sąsają ir naudojimosi juo paprastumą.

3.5. Testavimas

Šiame skyrelyje trumpai pateikiama bendra testavimo specifikacija, testavimo procedūros bei atlikto testavimo rezultatai. Tam kad kuriama sistema būtų geriau ištestuota ir pasiekta aukštesnė jos kokybė, buvo atliekami įvairaus tipo testavimai:

- Vienetų testavimas
- Integracijos testavimas
- Vartotojo sąsajos testavimas
- Priėmimo testavimas

3.5.1. Vienetų testavimas

Šio testavimo metu buvo tikrinimas atskirų sistemos vienetų (klasių arba metodų) veikimas. Vienetai buvo testuojami paduodant jiems įėjimo duomenis ir stebint jų išėjimus bei lyginant juos su laukiamais rezultatais. Testavimui buvo naudojamas tiek juodos dėžės (funkcinis) tiek ir baltos dėžės (struktūrinis) testavimas. Pirmuoju atveju įėjimo duomenys ir laukiami rezultatai suformuojami nesigilinant į vieneto realizaciją, o tik naudojantis sistemos specifikacija. Tuo tarpu baltos dėžės testuose minėti duomenų rinkiniai sudaromi atsižvelgiant į vieneto realizaciją ir parenkami taip, kad būtų įvykdyta kuo daugiau testuojamo vieneto kodo eilučių.

Remiantis reikalavimų specifikacija bei sistemos architektūra sistemai testuoti buvo sudaryti 46 vienetų testai, kurie pasinaudojant jUnit testavimo karkasu buvo automatizuoti. Keičiantis reikalavimams šie testavai taip buvo tobulinami sistemos kūrimo eigoje. Sudarytų vienetų testų, skirtų testuoti klasės „SourceManager“ metodą „runMethod“, aprašo pavyzdys pateikiamas 2 lentelėje.

2 lentelė Vienetų testai „runMethod“ metodui

Nr	Įėjimas	Laukiamas rezultatas	Klaida
1.5.1.3.1	method: test1 params: 4, „line“ test1 metodo parametrai: int, String.	Įvykdomas metodas „test1“ (atspausdinama „4, line“)	-
1.5.1.3.2	method: test1 params: „text“, „line“ test1 metodo parametrai: int, String.	Metodas „test1“ nevykdomas	Neteisingi metodo parametrai.
1.5.1.3.3	method: test1 params: „text“ test1 metodo parametrai: int, String.	Įvykdomas metodas „test1“ (atspausdinama „text“)	-
1.5.1.3.4	method: test1 params: 4, „line“, „text“ test1 metodo parametrai: int, String.	Įvykdomas metodas „test1“ (atspausdinama „4, line“)	-

Nr	Įėjimas	Laukiamas rezultatas	Klaida
1.5.1.3.5	method: test2 test2 metodas neturi parametrų	Įvykdomas metodas „test2“	-
1.5.1.3.6	method: testNo testNo metodas neegzistuoja	Gražinama null reikšmė.	Pasirinktas metodas nerastas.

Sistemos kūrimo metu vienetų testai rado daug defektų, kurie buvo ištaisyti ir galutinėje sistemos versijoje visi 46 testai buvo sėkmingi.

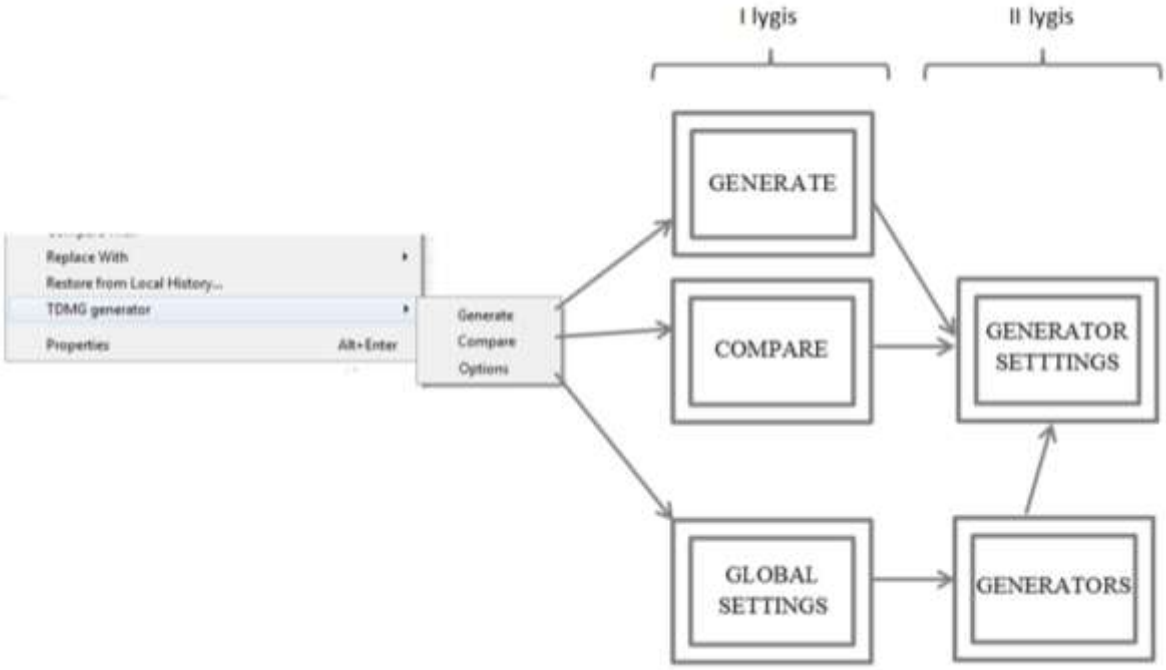
3.5.2. Integracinis testavimas

Šio testavimo metu buvo apjungiami keli vienetų testavimą sėkmingai praėję vienetai ir testuojama jų tarpusavio sąveika. Testavimui buvo naudojamas „Bottom-up“ metodas, kai pirmiausiai testuojami žemiausio lygio vienetai atliekantys operacijas su duomenimis, o vėliau jie sisteminami į modulius, kol galiausiai prieinama prie galinių vartotojo sąsajos vienetų ir atliekamas vartotojo sąsajos testavimas.

Integraciniam testavimui buvo sukurta 18 vienetų testų, kurias buvo manipuliuoja taip pat kaip ir su anksčiau aptartais vienetų testais. Galutinėje sistemos versijoje visi 18 testų buvo sėkmingi.

3.5.3. Vartotojo sąsajos testavimas

Vartotojo sąsajos testavimas atliekamas kaip paskutinis integracinio testavimo lygmuo, kai per vartotojo sąsają yra testuojamas visas sistemos funkcionalumas. Kadangi sistemos funkcionalumas nėra labai didelis ir prie vartotojo sąsajos jungiama jau vienetų testais dalinai ištestuota sistema, sąsajai testuoti nebuvo naudojami automatiniai įrankiai. Testavimas buvo atliktas rankiniu būdu spaudant sąsajos mygtukus ir tikrinant kaip ji atitinka darbo pradžioje apsibrėžtą sąsajos viziją. Tam kad testuotojui būtų lengviau atlikti savo darbą, kiekvienas sąsajos langas (16) buvo testuojamas atskirai pradedant nuo didžiausio lygio.



16 pav. Vartotojo sąsajos sandara

Kiekvienam langui buvo sukurti specialūs sąsajos testavimo scenarijai, kuriuose aprašyta į ką reikia atkreipti dėmesį spaudant sąsajos mygtukus. Iš viso buvo sukurti 28 testavimo scenarijai. Tokių scenarijų aprašo pavyzdys langui „Compare“ pateikiamas 3 lentelėje.

3 lentelė Testavimo scenarijai langui „Compare“

Nr	Atliekami veiksmai	Laukiamas rezultatas
2.2.1	Paspaudžiamas mygtukas „Settings“, esantis prie generatoriaus „Random“.	Atidaromas langas „Generator Settings“ su „Random“ generatoriaus individualiais parametrais.
2.2.2	Paspaudžiamas mygtukas „Settings“, esantis prie generatoriaus „TestGenerator“.	Atidaromas langas „Generator Settings“ su „TestGenerator“ generatoriaus individualiais parametrais.
2.2.3	Pažymimas „Random“ ir „TestGenerator“ generatoriai; Spaudžiamas mygtukas „Compare“.	Atliekamas duomenų generavimas pirmiausia „Random“ po to „TestGenerator“ generatoriais ir parodoma palyginimo ataskaita.
2.2.4	Pažymimas „Random“ generatorius; Spaudžiamas mygtukas „Compare“.	Atliekamas duomenų generavimas „Random“ generatoriumi ir parodoma generavimo ataskaita.
2.2.5	Pažymimas „Random“ ir „TestGenerator“ generatoriai; „Random“ generatoriui nurodomas 10 generacijų kiekis; „TestGenerator“ nurodomas 5 generacijų kiekis; Spaudžiamas mygtukas „Compare“.	Pirmiausia 10 kartų atliekamas „Random“ duomenų generavimas, po to 5 kartus „TestGenerator“ ir parodoma palyginimo ataskaita.
2.2.6	Pažymimas „Random“ ir „TestGenerator“ generatoriai; „Random“ generatoriui nurodomas 10 generacijų kiekis; „TestGenerator“ nurodomas 0 generacijų kiekis; Spaudžiamas mygtukas „Compare“.	Rodomas klaidos pranešimas apie blogai įvestą generacijų kiekį.
2.2.7	Pažymimas „Random“ generatorius; „Random“ generatoriui nurodomas 10 generacijų kiekis; „TestGenerator“ nurodomas 0 generacijų kiekis; Spaudžiamas mygtukas „Compare“.	10 kartų atliekamas duomenų generavimas „Random“ generatoriumi ir parodoma generavimo ataskaita.
2.2.8	Spaudžiamas mygtukas „Cancel“.	Uždaromas langas „Compare“.

Vykdant testavimo scenarijus buvo rasta daug neatitikimų tarp realizuotos grafinės vartotojo sąsajos ir sistemos reikalavimų. Dauguma jų buvo smulkus ir neįtakojantys tiesioginio sistemos funkcionalumo. Nepaisant neatitikimų didžio jie visi buvo ištaisyti ir galutinėje sistemos versijoje visi 28 testavimo scenarijai buvo įvykdyti sėkmingai.

3.5.4. Priėmimo testavimas

Priėmimo testavimas buvo vykdomas sėkmingai užbaigus integracijos testavimą. Šio testavimo metu buvo tikrinama kaip gerai sistema atitinka specifikacija ir užsakovo poreikius. Sistema buvo testuojama juodosios dėžės metodu tikrinant kiekvieną panaudos atvejį. Jei

testavimo metu buvo aptiktas neatitikimas tarp užsakovo poreikių ir sistemos funkcionalumo, buvo tikrinama ar poreikis yra specifikacijoje. Radus specifikacijoje užsakovo poreikį, kurio sistema netenkina, buvo registruojama klaida, kurią reikėjo ištaisyti. Tuo tarpu esant užsakovo poreikiui, kurio specifikacijoje nėra, buvo registruojamas norimas naujas programos pakeitimas, kuris bus įgyvendinamas naujausioje programos versijoje.

Po pirminio priėmimo testavimo buvo aptiki keli smulkus specifikacijoje dokumentuoti vartotojo poreikiai, kurių sistema netenkino. Galutinėje sistemos versijoje, ji buvo patobulinta, kad tenkintu visus specifikuotus užsakovo poreikius.

Testuojant sistemą kartu su užsakovu taip pat buvo nustatyti keli užsakovo poreikiai, kurie nebuvo įtraukti į pradinę sistemos specifikaciją. Nespecifikuoti užsakovo poreikiai, kurie reikalavo minimalių sistemos pakeitimų buvo patenkinti galutinėje sistemos versijoje. Tuo tarpu poreikiai reikalaujantys didesnių modifikacijų bus įgyvendinti realizuojant sistemos atnaujinimą.

3.6. Išvados ir apibendrinimas

1. Specifikuotas, suprojektuotas bei realizuotas automatinis testinių duomenų generavimo įrankis.
2. Kartu su įrankiu realizuoti keturi duomenų generavimo metodai: atsitiktinis generatorius, kopimo į kalną bei modeliuojamuoju atkaitinimu pagrįsti metodai ir genetinis generatorius.
3. Įrankis suprojektuotas ir realizuotas taip, kad ateityje būtų paprasta jį praplėsti naujais duomenų generavimo metodais.
4. Sukurta programinė įranga buvo ištestuota pagal sudarytą testavimo planą. Testavimo metu buvo rasta daug defektų, į kuriuos buvo reaguota ir galutinėje sistemos versijoje jie visi pašalinti.
5. Realizuota programinė įranga atitiko apibrėžtus reikalavimus ir sėkmingai atiduota naudoti užsakovui.
6. Su užsakovu aptarti galimi tolimesni įrankio plėtimo ir tobulinimo būdai.

4. EKSPERIMENTINIS TYRIMAS

4.1. Tyrimo aplinka

Tyrimas atliktas nešiojamame kompiuteryje turinčiame:

- Intel Core 2 Duo dviejų branduolių 2,40 GHz procesorių;
- 2Gb operatyviosios atminties;
- 32 b Windows 7 operacinę sistemą.

Tyrimo metu naudota programinė įranga:

- Eclipse IDE 4.2.2 (Juno);
- TDMG v1.2 (šio darbo metu sukurtas Eclipse programavimo aplinkos plėtinys).

4.2. Testuojamos programos

Tyrimui atlikti buvo pasirinktos 6 programos dažnai naudojamos kaip testiniai pavyzdžiai (angl. *benchmarks*). Pagal savo specifiką, jos buvo suskirstytos į 3 grupes: matematinių skaičiavimų, masyvų apdorojimo bei žaidimų logikos. Visų jų įėjimo duomenys yra elementarieji Java kalbos duomenų tipai arba jų masyvai.

Toliau yra aprašomos pasirinktosios programos ir pateikiamas jų charakteristikų palyginimas (6 lent.).

Sinuso skaičiavimas. Tai paprasta programėlė skirta apskaičiuoti sinuso reikšmei. Jos įėjimo duomenis yra vienas *double* tipo kintamasis.

Didžiausio bendrojo daliklio (DBD) skaičiavimas. Tai programėlė skirta surasti didžiausią dviejų skaičių bendrąjį daliklį. Jos įėjimo duomenys yra du *integer* tipo kintamieji.

Įrašo išrinkimas iš sąrašo. Tai programėlė skirta iš duomenų sąrašo išrinkti įrašus pagal tam tikrus kriterijus. Jos įėjimo duomenys yra *integer* tipo kintamųjų masyvas (duomenų sąrašas) bei penki *integer* tipo kintamieji (išrinkimo kriterijai). Duomenų sąrašo dydis gali svyruoti nuo 0 iki maksimalios generatoriaus naudojamos *integer* reikšmės (žiūrėti 7 lent.).

Sąrašo rikiavimas burbulu. Tai programėlė rikiuojanti duomenų sąrašą didėjimo tvarka tam pasinaudodama burbulo metodu. Jos įėjimo duomenys yra *integer* tipo kintamųjų masyvas. Duomenų sąrašo dydis gali svyruoti nuo 0 iki maksimalios generatoriaus naudojamos *integer* reikšmės (žiūrėti 7 lent.).

Tetrio žaidimo logikos klasė. Tai tetrio logiką realizuojanti klasė. Tam kad būtų lengviau ją apdoroti, jį buvo šiek tiek modifikuota ir jai sukurtas vienas įėjimo metodas, per kuri klasė bendrauja su išore. Šiam metodui yra paduodama esama žaidimo būsena bei bandomas atlikti žaidėjo veiksmas. Metodo gražinama reikšmė nusako žaidimo būseną po žaidėjo atlikto veiksmo.

Įėjimo metodo parametrai šiai klasei pateikiami 4 lentelėje.

4 lentelė Tetrio žaidimo logikos klasės įėjimo metodo parametrai

Parametras	Tipas	Masyvo dydis	Rėžiai	Paskirtis
windowState	boolean masyvas	200		Masyvas nusakantis, kurie tetrio lango elementai yra užimti, o kurie laisvi
partX	integer	-	-1 - 10	Naujos detalės apatinio kairiojo kampo X koordinatė
partY	integer	-	-1 - 20	Naujos detalės apatinio kairiojo kampo

Parametras	Tipas	Masyvo dydis	Rėžiai	Paskirtis
				Y koordinatė
partWidth	integer	-	-1 - 4	Naujos detalės plotis
partHeight	integer	-	-1 - 4	Naujos detalės aukštis
partState	boolean masyvas	16		Masyvas nusakantis, kurie naujos detalės elementai yra užimti, o kurie laisvi
partRotatePoint	integer	-	-1 - 16	Naujos detalės taškas, kurio atžvilgiu yra vykdoma dalies sukimo operacija
action	integer	-	-1 - 4	Žaidėjo norimas atlikti veiksmas

Gyvatėlės žaidimo logikos klasė. Tai gyvatėlės logiką realizuojanti klasė. Tam kad būtų lengviau ją apdoroti, jį buvo šiek tiek modifikuota ir jai sukurtas vienas įėjimo metodas, per kuri klasė bendrauja su išore. Šiam metodui yra paduodama esama žaidimo būsena bei bandomas atlikti žaidėjo veiksmas. Metodo gražinama reikšmė nusako žaidimo būseną po žaidėjo atlikto veiksmo.

Įėjimo metodų parametrai šiai klasei pateikiami 5 lentelėje.

5 lentelė Gyvatėlės žaidimo logikos klasės įėjimo metodo parametrai

Parametras	Tipas	Masyvo dydis	Rėžiai	Paskirtis
startX	integer	-	-1 - 20	Gyvatėlės pradžios X koordinatė
startY	integer	-	-1 - 20	Gyvatėlės pradžios Y koordinatė
snakeLengths	integer masyvas	0 - 400	-1 - 20	Gyvatėlės dalių ilgiai
snakeTurns	Integer	0 - 400	-1 - 4	Gyvatėlės posūkiai
candyX	integer	-	-1 - 20	Saldainio X koordinatė
candyY	integer	-	-1 - 20	Saldainio Y koordinatė
action	integer	-	-1 - 4	Žaidėjo norimas atlikti veiksmas

6 lentelė Testuojamos programos

Algoritmo pavadinimas	Tipas	Kodo eilučių kiekis	Instrukcijų kiekis	Šakų kiekis	Įėjimo duomenys
Sinuso skaičiavimas	Matematinis	26	121	16	*double* tipo kintamasis
Didžiausio bendrojo daliklio (DBD) skaičiavimas	Matematinis	11	32	10	du *integer* tipo kintamieji
Įrašo išrinkimas iš sąrašo	Masyvų apdorojimo	23	62	14	*integer* tipo masyvas bei penki *integer* tipo kintamieji (išrinkimo kriterijai)
Sąrašo rikiavimas burbulu	Masyvų apdorojimo	6	49	6	*integer* tipo masyvas

Algoritmo pavadinimas	Tipas	Kodo eilučių kiekis	Instrukcijų kiekis	Šakų kiekis	Įėjimo duomenys
Tetrio žaidimo logikos klasė	Žaidimas	86	194	30	aštuoni įvairių elementariųjų tipų kintamieji apibūdinantys esamą žaidimo būseną
Gyvatėlės žaidimo logikos klasė	Žaidimas	70	170	23	septyni įvairių elementariųjų tipų kintamieji apibūdinantys esamą žaidimo būseną

4.3. Tyrimo eiga

Prieš atliekant tyrimą remiantis testuojamų programų sudėtingumu bei kitais panašiais darbais [7][18] buvo nustatytos testinių duomenų generavimo metodų parametrų reikšmės, kurios buvo naudojamos viso tyrimo metu. Jos pateikiamos 7 lentelėje. Šalia parametrų reikšmių taip pat yra nurodyta, kokie kintamųjų reikšmių rėžiai buvo naudojami generuojant duomenis. Šie rėžiai buvo naudojami tik tais atvejais, kai nebuvo nurodyti kiti testuojamų programų parametrų rėžiai aptarti 4.2 skyrelyje.

7 lentelė Generavimo metodų parametrų reikšmės

Parametras	Reikšmė	Generavimo metodas			
		RA	***HC***	***SA***	***GA***
minimalus tenkinamas padengimas	90%	X	X	X	X
maksimalus iteracijų kiekis	1000	X	X	X	
maksimalus vidinių iteracijų kiekis	50			X	
temperatūros mažinimo koeficientas	0,9			X	
populiacijos dydis	10				X
kryžminimo tikimybė	0,9				X
mutavimo tikimybė	0,01				X
maksimalus populiacijų skaičius	1000				X
short, *int*, *long* rėžiai (jei nenurodyta kitaip)	-10 - 1000	X	X	X	X
float, *double* režiai (jei nenurodyta kitaip)	-10,0 - 1000,0	X	X	X	X
string rėžiai	[a-z], [A-Z], [0-9]	X	X	X	X

Tyrimo metu kiekvienai iš pasirinktų testuojamų programų visais generavimo metodais buvo generuojami testiniai duomenys ir stebimas pasiektas kodo padengimas, testinių duomenų rinkinių kiekis reikalingas jam pasiekti bei sugaištas jiems generuoti laikas. Generavimai buvo atlikti naudojantis visais trimis TDMG palaikomais padengimo kriterijais.

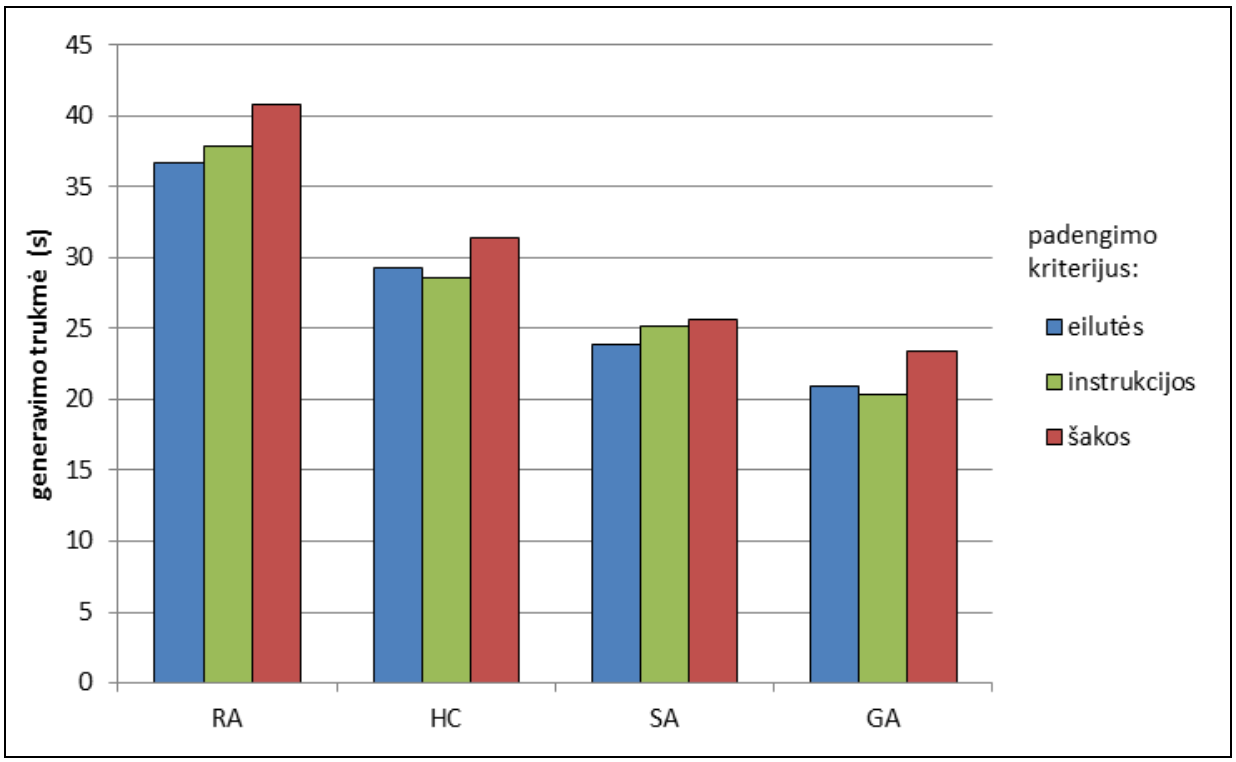
Kadangi testiniai algoritmai yra gana paprasti, duomenų generavimas jiems dažnai užtrunka labai trumpą laiką (iki 1s.). Siekiant didesnio tyrimo duomenų patikimumo, vienos generacijos metu testiniai duomenys buvo generuojami 100 kartų ir fiksuojamas suminis laikas bei vidutinis sugeneruotų duomenų rinkinių kiekis. Taip pat kiekviena generacija buvo pakartota 5 kartus ir rezultatuose pateikiamas visų generacijų aritmetinis vidurkis.

Generuojant duomenis pasitaikydavo atvejų, kai minimalus tenkintinas kodo padengimas (90%) nebūdavo pasiekiamas. Tokiu atveju, kaip generavimo iteracijos trukmė būdavo fiksuojamas laikas, sugaištas bandant pasiekti minimalų padengimą. Tuo tarpu iteracijos metu sugeneruotų duomenų rinkinių kiekiui būdavo priskiriamas didžiaisiais kitų generacijos iteracijų metu pasiektas rinkinių kiekis, su kuriuo minimalus padengimo kriterijus buvo tenkinamas. Kadangi daugumos generavimo iteracijų metų minimalaus padengimo kriterijus būdavo pasiektas (apie 95% visų iteracijų), tai toks iteracijų, netenkinančių minimalaus padengimo kriterijaus, modifikavimas nesudaro didelės įtakos tyrimo objektyvumui.

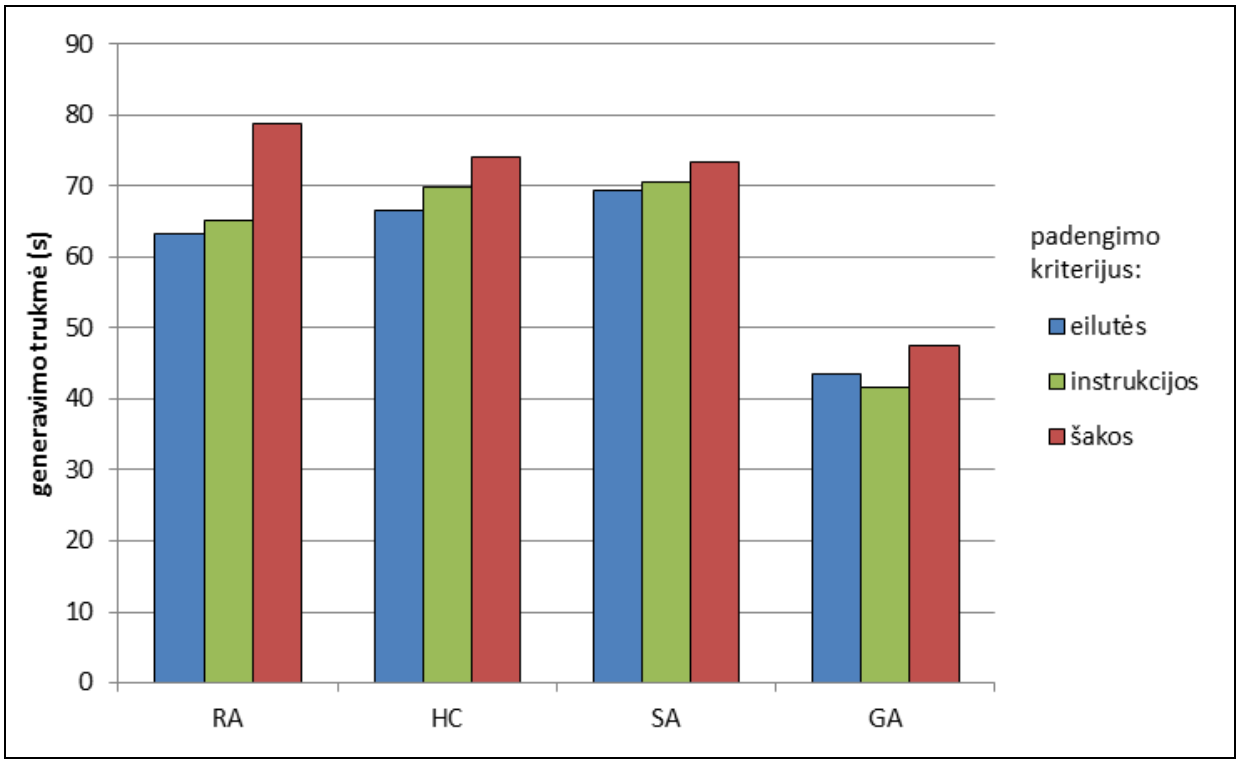
4.4. Tyrimo rezultatai

4.4.1. Testinių duomenų generavimo trukmės palyginimas

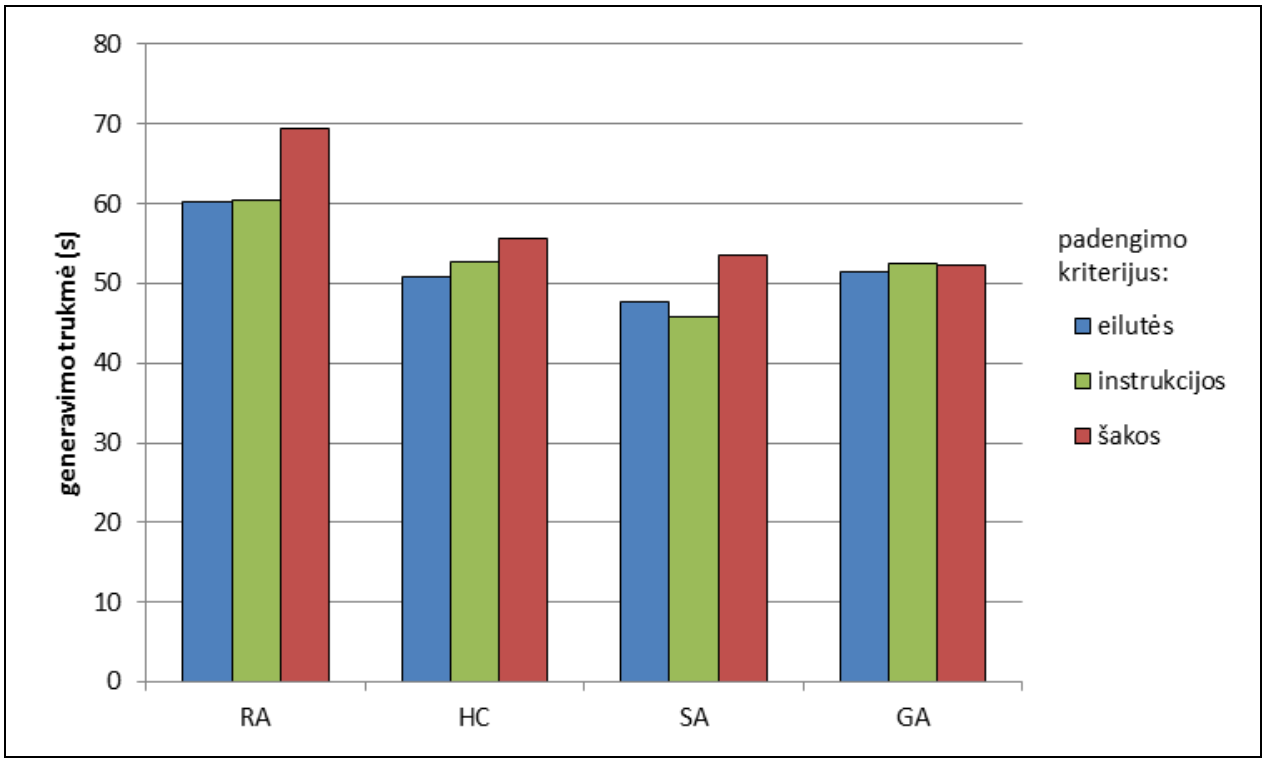
Generavimo metodų veikimo trukmės naudojant skirtingus padengimo kriterijus yra pateikiamos 17-19 paveikslėliuose. (17 pav. – sinuso skaičiavimo algoritmui, 18 pav. – įrašo išrinkimo algoritmui, 19 pav. – tetrio žaidimo logikos klasei). Iš šių rezultatų matyti, jog beveik visais atvejais testinių duomenų generavimas remiantis šakų padengimu užtrunka ilgiau negu remiantis instrukcijų arba kodo eilučių padengimais.



17 pav. Testinių duomenų generavimo sinuso skaičiavimo programai trukmė



18 pav. Testinių duomenų generavimo sinuso įrašo išrinkimo programai trukmė



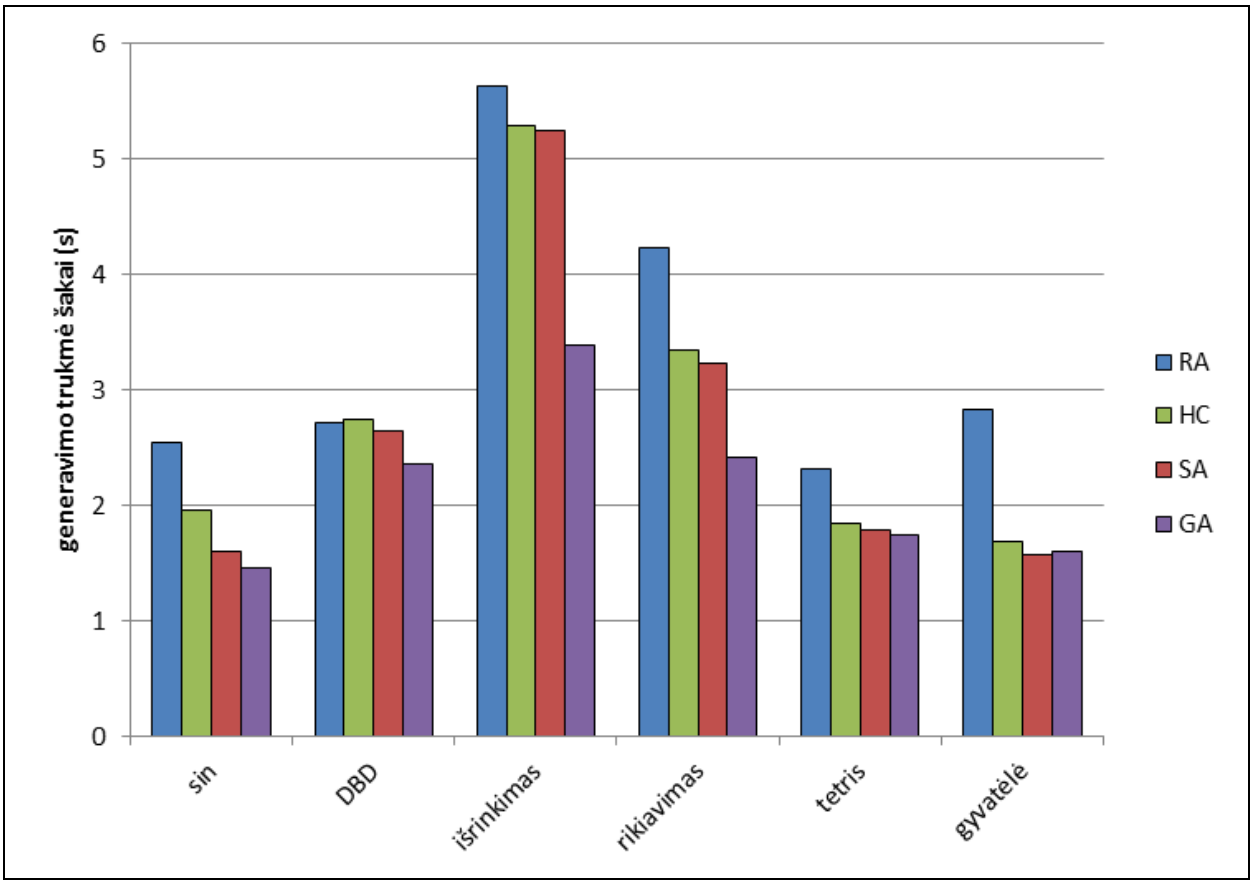
19 pav. Testinių duomenų generavimo tetrio logikos klasei trukmė

Testinių duomenų generavimo trukmės palyginimas priklausomai nuo testuojamos programos pateikimas 8 lentelėje ir 20 paveikslėlyje. Jose pateikiamos trukmės yra gautos remiantis šakų padengimu, kuris iš visų TDMG palaikomų padengimo kriterijų teikia objektyviausią įvertį. Tam, kad eliminuoti tyrimo netikslumą dėl nevienodo šakų kiekio

skirtingose testuojamose programose, rezultatuose yra pateikiamos santykinės generavimo trukmės vienai programos šakai.

8 lentelė Testinių duomenų generavimo trukmė programos kodo šakai

Duomenų generavimo metodas	sin	DBD	išrinkimas	rikiavimas	tetris	gyvatėlė
RA	2,550	2,720	5,628	4,233	2,316	2,834
HC	1,962	2,750	5,285	3,350	1,853	1,686
SA	1,600	2,650	5,242	3,233	1,786	1,569
GA	1,462	2,360	3,392	2,416	1,743	1,608



20 pav. Testinių duomenų generavimo trukmė skirtingoms programoms

Iš šio palyginimo matyti, jog RA trukmės skirtumas, lyginat su kitais generavimo metodais, labai svyruoja (gyvatėlės logikos klasės atveju net iki 68%) ir neturi jokio dėsningumo. RA veikimas dažniausiai užtrunka ilgiausiai iš visų arba panašiai kaip HC bei SA, tačiau jis niekada nebūna greičiausias.

SA trukmė lyginant su HC visų generacijų metu yra mažesnė ir dauguma atvejų skiriasi nedaug (iki 7%). Didesnis skirtumas tik sinuso skaičiavimo algoritmo atveju (18%).

GA trukmė beveik visų generacijų metu yra trumpiausia. Matematinių algoritmų bei žaidimų logikos klasių atveju GA nežymiai skiriasi nuo HC ir SA. Tačiau dirbant su masyvų apdorojimo algoritmais GA užtrunka net iki 35% trumpiau (įrašo išrinkimo algoritmas).

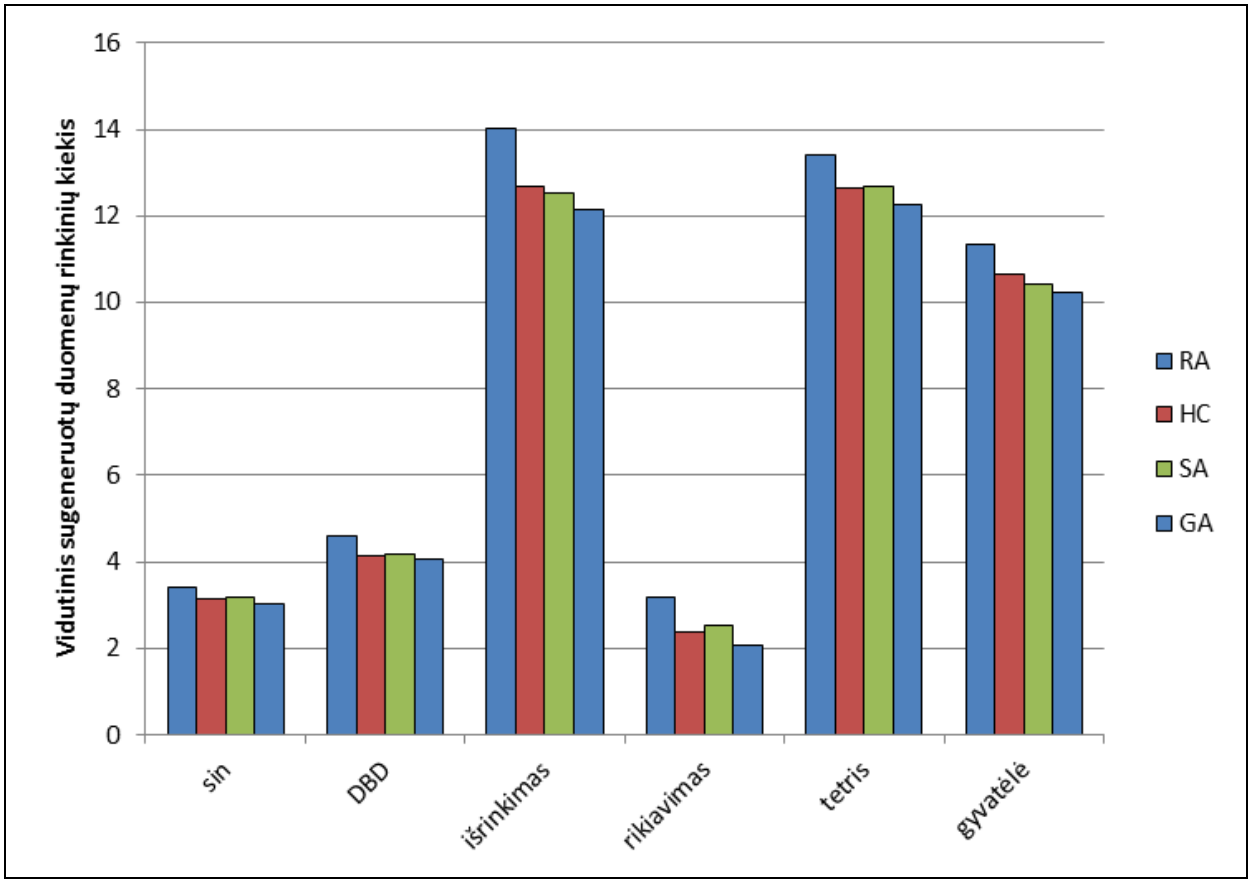
4.4.2. Sugeneruotų testinių duomenų rinkinių kiekio palyginimas

Palyginus generacijų metu gautą sugeneruotų testinių duomenų vidutinį kiekį naudojant skirtingus padengimo kriterijus, nebuvo įžvelgta jokio dėsningumo. Todėl buvo padaryta išvada, kad generuojamų duomenų rinkinių kiekis nepriklauso nuo pasirinkto padengimo kriterijaus.

Sugeneruotų rinkinių kiekio palyginimas priklausomai nuo pasirinkto duomenų generavimo metodo visoms testuojamoms programoms pateikiamas 9 lentelėje ir 21 paveikslėlyje. Jame parodoma kiek vidutiniškai generacijos metu būdavo sugeneruojama testinių rinkinių.

9 lentelė Vidutinis sugeneruotų duoemenų rinkinių kiekis

Duomenų generavimo metodas	sin	DBD	išrinkimas	rikiavimas	tetris	gyvatėlė
RA	3,40	4,60	14,04	3,18	13,40	11,33
HC	3,15	4,14	12,70	2,39	12,65	10,67
SA	3,17	4,17	12,54	2,52	12,69	10,44
GA	3,04	4,07	12,15	2,06	12,25	10,23



21 pav. Vidutinis sugeneruotų duomenų rinkinių kiekis

Iš šio palyginimo matyti, kad generuojant testinius duomenis RA minimalus tenkintinas padengimas vidutiniškai yra pasiekiamas sugeneravus didesnį kiekį duomenų rinkinių nei kitais tirtais generavimo metodais. Tuo tarpu vidutinis sugeneruotų duomenų rinkinių kiekis naudojant HC, SA ir GA metodus yra labai panašus (skiriasi iki 4,5%).

Iš tirtų generavimo metodų, kaip ir generavimo trukmės atžvilgiu, geriausi rezultatai yra GA, kuris beveik visais atvejais pasiekia minimalų tenkinamą padengimą sugeneravęs mažaisiais duomenų rinkinių lyginant su kitais metodais.

4.5. Tyrimo išvados ir apibendrinimas

1. Tyrimui atlikti pasirinktos 6 testuojamos programos, kurios priklauso skirtingos specifikos grupėms (matematinės programos, masyvo apdorojimo programos bei žaidimų logikos klasės)
2. Pasirinktoms programoms visais duomenų generavimo metodais ir naudojantis skirtingais padengimo kriterijais buvo generuojami testiniai duomenis ir fiksuojamos generavimo charakteristikos, kurios vėliau buvo analizuojamos.
3. Išbandžius testinių duomenų generavimą remiantis įvairiais kodo padengimo kriterijais, nustatyta, kad duomenų generavimas remiantis šakų padengimo kriterijumi, kuris yra objektyviausias iš tyrime naudotų kodo padengimo kriterijų, užtrunka ilgiausiai.
4. Išbandžius įvairius testinių duomenų generavimo metodus išsiaiškinta, jog tinkamai pasirinkus duomenų generavimo metodą generavimo procesą galima atlikti net iki 43% greičiau (skirtumas tarp RA (4,23 s) ir GA (2,42 s) trukmių masyvo rikiavimo atveju).
5. Remiantis atliktų tyrimų rezultatais galima teigti, kad testuojant matematinius algoritmus bei žaidimų logikos klases HC, SA bei GA trukmės atžvilgiu yra labai panašūs. Tačiau testuojant masyvų apdorojimo algoritmus labiausiai tam tinkamas yra GA, kuris veikia net iki 35% (skirtumas tarp SA (5,24 s) ir GA (3,39 s) trukmių įrašo išrinkimo atveju) greičiau negu kiti generavimo metodai.
6. Lyginat tarpusavyje sugeneruotų testinių duomenų rinkinių kiekį naudojantis skirtingais generavimo metodais išryškėja tai, kad RA metodu dažniausiai yra sugeneruojamas didesnis rinkinių kiekis nei kitais generavimo metodais.
7. Vidutinis sugeneruotų duomenų rinkinių kiekis remiantis HC, SA ir GA metodais yra labai panašus (skiriasi iki 4,5%), tačiau beveik visais atvejais GA sugeneruoja mažiausiai duomenų rinkinių.

5. DARBO APIBENDRINIMAS IR IŠVADOS

5.1. Darbo apibendrinimas

1. Darbo metu buvo apžvelgti įvairūs testinių duomenų generavimo metodai, iš kurių pasirinkti ir detaliau išanalizuoti bei realizuoti 4 dinaminiai metodai: atsitiktinis generatorius, kopimo į kalną bei modeliuojamuoju atkaitinimu pagrįsti metodai ir genetinis generatorius.
2. Buvo sukurtas Eclipse programavimo aplinkos plėtinys skirtas generuoti duomenis realizuotais generavimo metodais bei gebantis palyginti jų veikimo charakteristikas tarpusavyje. Plėtinys buvo sukurtas taip, kad ateityje jį būtų paprasta praplėsti naujais generavimo metodais.
3. Pasinaudojant sukurtu įrankių buvo atliktas eksperimentinis tyrimas, kurio metu buvo stebimas realizuotų generavimo metodų tinkamumas skirtingos specifikos testuojamoms programoms.
4. Taip pat buvo palygintas metodų veikimas naudojant skirtingus kodo padengimo kriterijus (kodo eilučių, instrukcijų bei šakų).

5.2. Darbo išvados

1. Atliktos testinių duomenų generavimo įrankių analizės metu nebuvo rasta komercinių įrankių leidžiančių palyginti duomenų generavimą keliais skirtingais metodais. Tuo tarpu akademiniai įrankiai skirti metodų palyginimui dažniausiai būna prieinami tik siaurai auditorijai.
2. Išbandžius testinių duomenų generavimą remiantis įvairiais kodo padengimo kriterijais, nustatyta, kad duomenų generavimas remiantis šakų padengimo kriterijumi, kuris yra objektyviausias iš tyrime naudotų kodo padengimo kriterijų, užtrunka ilgiausiai.
3. Išbandžius įvairius testinių duomenų generavimo metodus išsiaiškinta, jog tinkamai pasirinkus duomenų generavimo metodą generavimo procesą galima atlikti net iki 43% greičiau (skirtumas tarp RA (4,23 s) ir GA (2,42 s) trukmių masyvo rikiavimo atveju).
4. Remiantis atliktų tyrimų rezultatais galima teigti, kad testuojant matematinius algoritmus bei žaidimų logikos klases HC, SA bei GA trukmės atžvilgiu yra labai panašūs. Tačiau testuojant masyvų apdorojimo algoritmus labiausiai tam tinkamas yra GA, kuris veikia net iki 35% (skirtumas tarp SA (5,24 s) ir GA (3,39 s) trukmių įrašo išrinkimo atveju) greičiau negu kiti generavimo metodai.
5. Lyginat tarpusavyje sugeneruotų testinių duomenų rinkinių kiekį naudojantis skirtingais generavimo metodais išryškėja tai, kad RA metodu dažniausiai yra sugeneruojamas didesnis rinkinių kiekis nei kitais generavimo metodais.
6. Vidutinis sugeneruotų duomenų rinkinių kiekis remiantis HC, SA ir GA metodais yra labai panašus (skiriasi iki 4,5%), tačiau beveik visais atvejais GA sugeneruoja mažiausiai duomenų rinkinių.

6. LITERATŪRA

[1] G. J. Myers, C. Sandler, T. Badgett, T. M. Thomas, The art of software testing, ISBN-978-0471469124, Wiley, 2004.

[2] B. Antonia, Software Testing research: achievements, challenges, dreams 2007 Future of software engineering: IEEE Computer Society, 2007.

[3] T. Garret, Implementing automated software testing continously track progress and sdjust sccordingly, methods & tools, practical knowledge for software developer, tester and project manager fall 2009 (Volume 17 – number 3), ISSN: 1661-402X, p. 15-30, 2009.

[4] Bhavin T., Automated Testing vs Manual Testing, [Prieiga internete] http://wiki.directi.com/x/AgAa [žiūrėta 2014.02.10].

[5] L. Baresi, M. Young, Test oracles, Technical report CIS-TR-01-02, 2001.

[6] M. Staats, G. Gay, M. P. E. Heimdahl, Automated oracle creation support, or: how I learned to stop worrying about fault propagation and love mutation testing, Proceedings of the 2012 International Conference on Software Engineering, Zurich, Switzerland, 2012.

[7] P. B. Nirpal & K. V. Kale, Comparison of software test data for automatic path coverage using genetic algorithm, International Journal of Computer Science & Engineering Technology (IJCSET), ISSN: 2229-3345, Vol. 1 No. 1, Sep 2012.

[8] S. Ali, L. C. Briand, H. Hemmati, R. K. Panesar-Walawege, A systematic review of the application and empirical investigation of evolutionary testing, Software Engineering: IEEE Transactions on, ISSN: 0098-5589, Vol. 36, Issue 6, p. 742-762, 2010.

[9] D. Saff, M. D. Ernst, Automatic mock object creation for test factoring, ACM SIGPLAN/SIGSOFT workshop on program analysis for software tools and engineering (PASTE'04), (Washington, DC, USA), p. 49-51, 2004.

[10] S. Freeman, T. Mackinnon, N. Pryce, J. Walnes, Mock roles, objects, Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications, Vancouver, BC, Canada, 2004.

[11] S. Zhang, D. Staff, Y. Bu, M. D. Ernst, Combined static and dynamic automated test generation, ISSTA ‘11, (Toronto, ON, Canada) p. 17-21, 2011.

[12] P. Godefroid, Test generation using symbolic execution, IARCS Annual conference on foundations of software technology and theoretical computer science (FSTTCS 2012), Vol. 18, p. 24-33, Dagstuhl, Germany, 2012.

[13] Y. Jia, M. Harman, An analysis and survey of the development of mutation testing, IEEE Transactions on Software Engineering, Vol. 37, no. 5, p. 649-678, 2011.

[14] D. Schuler, V. Dallmeier, A. Zeller, Efficient mutation testing by checking invariant violations, Proceedings of the eighteenth international symposium on Software testing and analysis, Chicago, IL, USA, 2009.

[15] Bj Rollison, Parametrized random test data generation, PNSQC, 2011.

[16] J. A. Edvardsson, Survey on automatic test data generation, Second Conference on Computer Science and Engineering in Linkoping, 1999.

[17] A. Arcuri, P. K. Lehre, Xin Yao, Theoretical runtime analyses of search algorithms on the test data generation for the triangle classification problem, Software testing verification and validation workshop, 2008. ICSTW ‘08. IEEE International conference., p. 161-169, 2008.

[18] K. Ghani, J. A. Clark, Automatic test data generation for multiple condition and MCDC coverage, Fourth International Conference on Software Engineering Advances (ICSEA ‘09), p. 152-157, 2009.

[19] A. Gotlieb, T. Denmat B. Botella, Goal-oriented test data generation for programs with pointer variables, Raport de recherche n°5528, Institut national de recherche en informatique et en automatique, ISSN: 0249-6399, 2005.

[20] E. K. Prebys, The genetic algorithm in computer science, MIT undergraduate journal of mathematics, p. 165-170, 2007.

[21] T. W. Williams, M. R. Mercer, J. P. Mucha, Code Coverage, What Does It Mean in Terms of Quality?, Reliability and Maintainability Symposium, Philadelphia, 2001.

[22] JaCoCo Java Code Coverage Library, Mountainminds GmbH & Co, prieiga internete http://www.eclemma.org/jacoco/, [žiūrėta 2014-03-02].

[23] K. J. Hayhurst, D. S. Veerhusen, J. J. Chilenski, L. K. Rierson, "A practical tutorial on modified condition/decision coverage", NASA, 2001.

[24] CodePro Analytix, prieiga internete https://developers.google.com/java-dev-tools/codepro/doc/ [žiūrėta 2012.11.30].

[25] Squish Coco Code Coverage, prieiga internete http://www.froglogic.com/squish/coco/?gclid=CIbP56-46LMCFUmN3godhCAAnA [žiūrėta 2012.11.30].

[26] jPET, prieiga internete http://costa.ls.fi.upm.es/pet/download [žiūrėta 2012.11.30].

[27] Albert E., Cabanas I., Flores-Montoya A., Gomez-Zamalloa M., Gutierrez S, jPET: an Automatic Test-Case Generator for Java, Complutense University of Madrid, Spain, 2011.

[28] GNU General Public License, prieiga internete http://www.gnu.org/copyleft/gpl.html [žiūrėta 2014.01.30].

7. TERMINŲ IR SANTRUMPŲ ŽODYNAS

Baltosios dėžės testavimas	–	testavimo būdas, kai testuojant nagrinėjamas programos kodas
IDE	–	integruota programinės įrangos kūrimo aplinka (angl. *Integrated Development Environment*)
GA	–	genetinio algoritmo pagrindu veikiantis duomenų generavimo metodas (angl. *Genetic Algorithm*)
GPL	–	GNU bendroji viešoji licencija (angl. *GNU General Public License*)
HC	–	kopimo į kalną optimizacijos algoritmo pagrindu veikiantis duomenų generavimo metodas (angl. *Hill Climbing*)
HTML	–	turinio aprašymo standartas (angl *Hyper text Markup Language*)
Juodosios dėžės testavimas	–	testavimo būdas, kai testuojant nenagrinėjamas programos kodas, o testams sudaryti naudojama vartotojo reikalavimai ir specifikacijos.
JVM	–	Javos virtuali mašina, sluoksnis tarp programos ir operacinės sistemos leidžiantis tą pačią programą vykdyti įvairiose operacinėse sistemose (angl. *Java Virtual Machine*)
OOP	–	objektiškai orientuotas programavimas
RA	–	atsitiktinio generavimo algoritmo pagrindu veikiantis duomenų generavimo metodas (angl. *Random Algorithm*)
RUP	–	Rational unifikuotas procesas (angl. *Rational Unified Process*)
SA	–	modeliuojamo atkaitinimo optimizacijos algoritmo pagrindu veikiantis duomenų generavimo metodas (angl. *Simulated Annealing*)
UC	–	panaudojimo atvejis (angl. *Use Case*)
UML	–	unifikuota modeliavimo kalba (angl. *Unified Modeling Language*)
TDMG	–	automatinis testinių duomenų multi-generatorius (angl. *Test Data Multi-Generator*)
TXT	–	paprasto tekstinio failo formatas
XML	–	duomenų aprašymo standartas (angl. *eXtensible Markup Language*)

8. PRIEDAI

Prieduose yra pateikiami šio darbo autoriaus magistro studijų metu su kolegomis parašyti straipsniai, kuriuose panaudojami šiame darbe analizuoti duomenų generavimo metodai bei atlikto tyrimo medžiaga:

I. Gintautas Motiejūnas, Titas Kuckis, Darius Liepinaitis. „Automatinio testinių duomenų generavimo metodų tyrimas“ (priedas A).
II. Darius Liepinaitis, Gintautas Motiejūnas, Titas Kuckis. „Klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant laikines charakteristikas, kūrimas ir tyrimas“ (priedas C).

Šie straipsniai buvo pateikti ir pristatyti XIX tarpuniversitetinėje magistrantų ir doktorantų konferencijoje „Informacinė visuomenė ir universitetinės studijos“ (IVUS 2014) bei išspausdintas knygoje „Informacinės technologijos. XIX tarpuniversitetinė magistrantų ir doktorantų konferencija „Informacinė visuomenė ir universitetinės studijos“ (IVUS 2014). Konferencijos pranešimų medžiaga“. ISSN 2029-4832. Konferencijos sekcijoje „IT testavimas ir sauga“ II straipsnis buvo pripažintas geriausiu.

Kartu su straipsniais yra pateikiamos pažymų liudijančių apie straipsnių pristatymą konferencijoje bei geriausio sekcijos straipsnio vardą kopijos (priedai B, D, E).

A. Straipsnis „Automatinio testinių duomenų generavimo metodų tyrimas

Automatinio testinių duomenų generavimo metodų tyrimas

Gintautas Motiejūnas[1], Titas Kuckis[2], Darius Liepinaitis[3]
Programų inžinerijos katedra. Informatikos fakultetas.
KTU
Kaunas, Lietuva
[1]gintautas.motiejunas@stud.ktu.lt; [2]titas.kuckis@stud.ktu.lt; [3]darius.liepinaitis@stud.ktu.lt

***Anotacija* – Šiame straipsnyje yra kalbama apie automatinio testinių duomenų generavimo metodus. Nagrinėjamas atsitiktinis generatorius bei keli optimizacijų algoritmais (kopimo į kalną, modeliuojamojo atkaitinimo bei genetiniu) paremti metodai. Tyrimo metu visi minėti metodai yra palyginami tarpusavyje generuojant duomenis įvairios specifikos testinėms programoms. Formuluojamos rekomendacijos testinius duomenis automatiniu būdu generuojantiems vartotojams.**

Reikšminiai žodžiai – testinių duomenų generavimas; genetinis duomenų generavimas; kopimo į kalną algoritmas, modeliuojamojo atkaitinimo algoritmas

I. ĮVADAS

Šiuo metu kuriant programinę įrangą vis didesnis dėmesys yra skiriamas jos kokybei. Nebėra žiūrima vien tik į programos funkcionalumą (kuo jis didesnis, kuo programa atlieka daugiau funkcijų – tuo ji yra geresnė), bet vertinama ir jos kokybė (kaip gerai programa atlieka savo funkcijas, kaip efektyviai yra panaudojami turimi resursai, kokia saugi ji yra ir t.t.).

Vienas iš dažniausiai naudojamų kokybės užtikrinimo ir kėlimo būdų yra programinės įrangos testavimas. Tai procesas arba jų serija, kurie yra sukurti įsitikinti, jog parašytas programos kodas atlieka būtent tai, kas buvo numatyta jį rašant ir neatlieka to, kas nebuvo numatyta [1]. Testavimas yra labai imlus laikui bei pinigams procesas. Kuriant programinę įrangą jis vidutiniškai užima daugiau negu 50% viso numatyto laiko [2]. Dėl šios priežasties testavimą siekiama įvairias būdais automatizuoti. Nors testavimo procedūra ją automatizuojant gali užtrukti žymiai ilgiau, nei vykdant ją paprastai, automatizuotas testavimas atsiperka jį pakartotinai panaudojant daug kartų [3]. Automatinis testavimas susiduria su daug problemų: orakulo problema [4], automatinis testinių duomenų generavimas [5, 6], automatinis testinių objektų generavimas [7].

Viena didžiausių iš jų yra automatinis testinių duomenų generavimas. Jai spręsti yra sukurta daug įvairių metodų, kurie visą laiką yra tobulinami. Metodus testinių duomenų generavimui galima suskirstyti į dvi klases: statinius metodus ir dinaminius metodus. Statiniai metodai remiasi simboliniu kodo vykdymu, vykdomo kodo aprėpiamos srities mažinimu bei kitomis metodologijomis [8]. Jie testinius duomenis generuoja nevykdydami pilno programos kodo, todėl susiduria su problemomis, kai testuojamų programų apimtys didėja ir didėja jų sudėtingumas.

Dinaminiai metodai apima atsitiktinius [9], paremtus vietine paieška [10, 11, 12], į tikslą orientuotus [10, 13], grandinėlės principu veikiančius [10] bei evoliucinius generatorius [10, 14]. Pastarieji testinius duomenis generuoja vykdydami testuojamų programų kodą ir remiantis testų rezultatais optimizuoja jų generavimą. Jie išvengia daugumos problemų, su kuriomis susiduria statiniai metodai. Dėl pačios testinių duomenų generavimo problemos sudėtingumo ne visi generavimo metodai yra vienodai tinkamai testuojamoms programos. Jų tinkamumas priklauso nuo programų specifikos.

Šio straipsnio tikslas yra išanalizuoti skirtingus generavimo metodus ir palyginti jų tinkamumą įvairios specifikos programos testuoti. Toliau yra pateikiama šio straipsnio struktūra. II skyriuje yra apžvelgiami jau atlikti panašūs tyrimai. III skyriuje yra aprašomi pasirinkti testinių duomenų generavimo metodai, kurie IV skyriuje aprašyta metodika yra palyginami. Atlikto tyrimo rezultatai yra pateikiami V skyriuje bei apibendrinami išvadose VI skyriuje. VII skyriuje aprašomi ateityje numatyti atlikti darbai.

II. PANAŠŪS TYRIMAI

Tiek sukurti nauji tiek ir patobulinti testinių duomenų generavimo metodai yra dažnai lyginami su atsitiktinių duomenų generatoriumi [15, 16], kuris yra naudojamas kaip atskaitos taškas ir parodo generatoriaus naudingumą. Palyginimuose atsitiktiniai generatoriai dažniausiai pasirodo prasčiausiai.

Yra atlikta ir nemažai tyrimų, kuriuose tarpusavyje lyginami įvairių klasių generavimo metodai [17, 18]. Daugumoje atliktų tyrimų buvo tiriamos modifikuotos, tam tikros specifikos programoms (matematinių skaičiavimų, lygiagrečių skaičiavimų, saugumo, internetinių ir t.t.) orientuotos metodų versijos. Tokių tyrimų rezultatai nėra vienareikšmiški, kurio nors metodo naudai. Tai lemia pastaruųjų specifika ir jų nevienodas tinkamumas testuojamoms programoms.

III. GENERAVIMO METODAI

Plačiai naudojami dinaminiai duomenų generavimo metodai apima atsitiktinį generatorių bei metodus, kurie generavimui naudoja įvairius optimizacijų algoritmus, tokius kaip: kopimo į kalną algoritmas, modeliuojamojo atkaitinimo algoritmas, tabu paieška, genėtinai algoritmai, skruzdžių kolonijos optimizaciją, dirbtinio intelekto sistemos,

sklaidančioji paieška. Mūsų atliktame tyrime yra nagrinėjami keturi iš jų: atsitiktinis generatorius (angl. *Random Algorithm*, RA), kopimo į kalną algoritmu paremtas metodas (angl. *Hill Climbing*, HC), modeliuojamojo atkaitinimo optimizacija paremtas metodas (angl. *Simulated Annealing*, SA) bei standartiniu genetiniu algoritmu besiremiantis metodas (angl. *Genetic Algorithm*, GA).

A. Atsitiktinis generatorius

Tai yra paprasčiausias testinių duomenų generavimo metodas. Jis atsitiktinai generuoja duomenis ir sustoja, kai yra pasiekiamas problemos sprendinys arba peržengiami algoritmo vykdymo rėžiai [9]. Toks generatorius dažnai naudojamas kaip atskaitos taškas įvertinti kitų, sudėtingesnių generatorių efektyvumą.

B. Kopimo į kalną optimizacija paremtas metodas

Šios optimizacijos veikimas priklauso nuo strategijos, pagal kurią yra pasirenkamas gretimas sprendinys bei strategijos, pagal kurią pasirenkamas naujas pradinis sprendinys [11]. Nagrinėjamu atveju buvo pasirinkta paprasčiausia gretimo sprendinio pasirinkimo strategija. Pagal ją gretimas sprendinys gaunamas iš jau esamo sprendinio atsitiktinai pasirinkus vieną testuojamo metodo parametro reikšmę ir prie jos dvejetainio pavidalo pridėjus arba atėmus vieną bitą. Tai atliekant taip pat yra atsižvelgiama į parametro galimų reikšmių rėžius.

Tuo tarpu naujo pradinio sprendinio pasirinkimui buvo naudota atsitiktinė strategija, kuri dažniausiai ir yra naudojama šioje optimizacijoje. Jos metu sprendinys yra atsitiktinai pasirenkamas iš visų galimų variantų.

C. Modeliuojamojo atkaitinimo optimizacija paremtas metodas

Šios optimizacijos metu yra vykdoma paieška, kurios pirmasis sprendinys yra pasirenkamas atsitiktinai. Visi po to einantys paieškos sprendiniai yra pasirenkami iš sprendinių gretimų prieš tai nagrinėtam sprendiniu. Tam yra naudojama tokia pati strategija kaip ir HC atveju. Pasirinktų sprendinių tinkamumas yra apsprendžiamas pagal tikimybę kuri priklauso nuo parametro vadinamo temperatūra (1) [12].

$$P(c_0, c, t) = exp(-(c - c_0) / t) \quad (1)$$

Čia c_0 – pradinio sprendinio padengiamumas, c – gretimo sprendinio padengiamumas, t - temperatūra

Pradinė temperatūros reikšmė yra didelė ir beveik visi paieškos žingsniai yra tinkami. Vykdant algoritmą temperatūros reikšmė yra pastoviai mažinama dauginant ją iš temperatūros mažinimo koeficiento. Žingsniai duodantys geresnį sprendimą yra visą laiką tinkami. Tuo tarpu žingsnių duodančių sprendimą, kuris yra prastesnis arba lygus prieš tai esančiam, tinkamumas yra apsprendžiamas pagal tikimybę. Kuo mažesnė temperatūra, tuo mažesnė tikimybė, kad blogesnis sprendinys bus tinkamas. Ši algoritmo savybė leidžia išvengti lokalaus optimumo. Kai SA temperatūros reikšmė yra lygi nuliui, paieškos efektyvumas pasidaro lygus HC algoritmui.

D. Standartiniu genetiniu algoritmu paremtas metodas

Šis algoritmas remiasi ląstelių kryžminimosi gyvuose organizmuose – miozės procesu. Gyvame organizme esančias chromosomas šiame algoritme atstoja dvejetainiai skaičiai (pavyzdžiui 101111001). Jie sudaro populiaciją. Populiacijos narių tinkamumą kryžminimui apibūdina tam tikra algoritmo funkcija. Vykstant algoritmui pagal išrinkimo funkciją, nusakančią, kuriuos populiacijos narius kryžminti, išrenkami du atskiri populiacijos individai, kurie po to yra kryžminami pagal kryžminimo funkciją ir vėl išskiriami. Galiausiai gauti du nauji individai mutuojami pagal mutavimo funkciją. Visas procesas yra kartojamas apibrėžtą kiekį kartų, kol yra pasiekiamas tam tikras pabaigimo kriterijus [14]. Toliau detaliau aptariamos algoritme paminėtos funkcijos.

1) *Tinkamumo funkcija.* Tai funkcija skaičiuojanti populiacijos individo tinkamumo įvertį sprendžiamai problemai. Dažniausiai jos rezultatai priklauso nuo ankstesnės algoritmo iteracijos rezultatų. Narinėjamų atvejų tinkamumo įvertis yra lygus paskutiniosios iteracijos metu pasiektai kodo padengiamumo kriterijaus reikšmei.

2) *Išrinkimo funkcija*. Tai funkcija išrenkanti porą populiacijos individų tolimesniam kryžminimui. Ši funkcija dažniausiai priklauso nuo tinkamumo funkcijos rezultatų. Individo tikimybė būti išrinktam kryžminimui yra tuo didesnė, kuo didesnis individo tinkamumo įvertis. Nagrinėjamu atveju yra atsitiktinai išrenkami du geriausią tinkamumo įvertį turintys individai.

3) *Kryžminimo funkcija.* Ši funkcija apibūdina tai, kaip genai (bitai) yra keičiami išrinktų individų poroje. Tai ar kiekvienos iteracijos metu pats kryžminimas įvyks ar ne apsprendžia kryžminimo tikimybė išreiškiama skaičiumi nuo 0 iki 1. Prieš kryžminimą yra sugeneruojamas atsitiktinis skaičius nuo 0 iki 1. Jei šis skaičius yra mažesnis už kryžminimo tikimybę, tada kryžminimas vykdomas, jei ne – yra gražinama nepakitusi individų pora. Nagrinėjamu atveju yra naudojama standartinė kryžminimo funkcija. Jos vykdymo metu pirmiausia atsitiktiniu būdu yra pasirenkamas individo bitas. Tada individo dalis nuo pasirinktojo bito (įskaitant ir jį patį) yra sukeičiama su atitinkama kito poros individo dalimi. Pavyzdžiui turime du individus 101111001 ir 110101010. Atsitiktinai pasirinktas bito numeris nuo 1 iki 9 yra 5. Tada individų pora po kryžminimo atrodys taip: 101101010 ir 110111001.

4) *Mutavimo funkcija.*Tai funkcija apsprendžianti kaip individai turi mutuoti. Mutavimui dažniausiai naudojama paprasta funkcija – pasirinkto bito invertavimas. Mutuojant individą kiekvienam jo bitui yra sugeneruojamas atsitiktinis skaičius nuo 0 iki 1, kuris yra lyginamas su mutacijos tikimybe ir skaičiui esant mažesniam už šią tikimybę individo bitas yra invertuojamas.

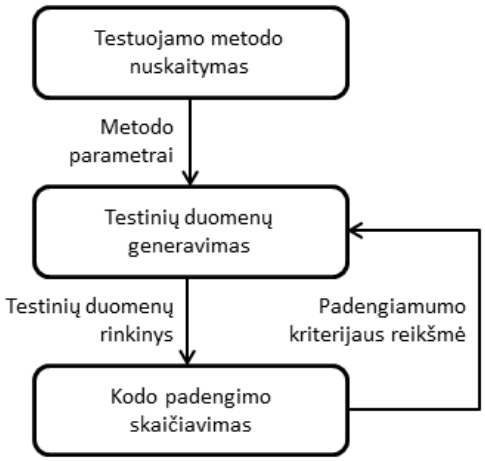
IV. TYRIMO METODIKA IR NAUDOTI ĮRANKIAI

A. *Testinių duomenų generatorius*

Tam, kad skirtingus testinių duomenų generavimo metodus būtų galima tirti esant kuo panašesnėmis sąlygomis bei siekiant kuo labiau supaprastinti jų panaudojimą, buvo sukurta speciali programinė įranga skirta generuoti duomenis vienu iš pasirinktų metodų – testinių duomenų multi-generatorius (TDMG). TDMG susideda iš 3 pagrindinių dalių :

- Duomenų generatorius – tai komponentas, kuriame yra realizuotas duomenų generavimo algoritmas. TDMG buvo sukurtas taip, kad prie jo lengvai galima prijungti kelis skirtingus algoritmus realizuojančius generatorius.
- Kodo padengimo skaičiavimo komponentas. Šis komponentas apskaičiuoja kodo padengiamumą su pasirinktais testiniais duomenimis. TDMG padengiamumui skaičiuoti naudoja JaCoCo [19] biblioteką.
- Valdymo komponentas. Šis komponentas apjungia pirmuosius du komponentus bei kontroliuoja jų veikimą.

Testinių duomenų generavimo, kurį atlieka TDMG, proceso schema pateikiama 1 paveikslėlyje.



22 pav. Testinių duomenų generavimas

Šį procesą galima apibūdinti 4 žingsniais:

1. Nuskaitomas testuojamos programos įėjimo metodas bei jo parametrai.
2. Pasirinktas duomenų generatorius pagal ankstesnės iteracijos metu gautas padengiamumo kriterijaus reikšmę (jeigu tai yra ne pradinė iteracija) sugeneruoja testinių duomenų rinkinį nuskaitytiems metodo parametrams.
3. Apskaičiuojamas testuojamos programos kodo padengiamumas su sugeneruotu testinių duomenų rinkiniu.
4. Grįžtama į antrą žingsnį, kol pasiekiamas minimalus tenkintinas padengiamumas arba kitas generavimo algoritmo užbaigimo kriterijus.

TDMG testinius duomenis gali generuoti remiantis kodo eilučių, instrukcijų arba šakų padengiamumo kriterijais (tai lemia naudojama padengiamumo skaičiavimo biblioteka).

Siekiant išgauti kuo tikslesnę generavimo metodų veikimo trukmę, TDMG į generacijos vykdymo laiką neįtraukia testinių duomenų rinkinių išsaugojimo bei kodo padengiamumo skaičiavimo operacijų.

B. *Tyrimo metodika*

Apžvelgus panašius atliktus tyrimus [15 - 18] buvo pasirinkti keli kitų autorių naudoti testiniai algoritmai, kuriuos pagal veikimo paskirti galima suskirstyti į atskiras grupes (matematiniai, masyvų apdorojimo, žaidimų logikos). Pastarųjų charakteristikos pateikiamos 1 lentelėje. Visų pasirinktų algoritmų įėjimo parametrai yra elementarieji Java kalbos duomenų tipai arba jų masyvai. Tyrimo metu kiekvienam iš pasirinktų testuojamų algoritmų visais generavimo metodais buvo generuojami testiniai duomenys ir stebimas pasiektas padengiamumas bei jam pasiekti sugaištas laikas. Generavimai buvo atlikti naudojantis visais trimis TDMG palaikomais padengiamumo kriterijais. Duomenų generavimo metodų parametrų reikšmės bei generuojamų reikšmių rėžiai naudoti visų generacijų metu pateikiami 2 lentelėje.

Kadangi testiniai algoritmai yra gana paprasti, duomenų generavimas jiems užtrunka labai trumpą laiką (iki 1s.). Siekiant didesnio tyrimo duomenų patikimumo, vienos generacijos metu testiniai duomenys buvo generuojami 100 kartų ir fiksuojamas suminis laikas. Taip pat kiekviena generacija buvo pakartota 5 kartus ir rezultatuose pateikiamas

X LENTELĖ. TESTUOJAMI ALGORITMAI

Algoritmo pavadinimas	Tipas	Kodo eilučių kiekis	Instrukcijų kiekis	Šakų kiekis	Įėjimo duomenys
Sinuso skaičiavimas	Matematinis	26	121	16	*double* tipo kintamasis
Didžiausio bendrojo daliklio (DBD) skaičiavimas	Matematinis	11	32	10	du *integer* tipo kintamieji
Įrašo išrinkimas	Masyvų apdorojimo	23	62	14	*integer* tipo masyvas bei penki *integer* tipo kintamieji (išrinkimo kriterijai)
Sąrašo rikiavimas burbulu	Masyvų apdorojimo	6	49	6	*integer* tipo masyvas
Tetrio logikos klasė	Žaidimas	86	194	30	aštuoni įvairių elementariųjų tipų kintamieji apibūdinantys esamą žaidimo būseną
Gyvatėlės logikos klasė	Žaidimas	70	170	23	septyni įvairių elementariųjų tipų kintamieji apibūdinantys esamą žaidimo būseną

visų generacijų aritmetinis vidurkis.

Tyrimas atliktas kompiuteryje su Intel Core 2 Duo dviejų branduolių 2,40 GHz procesoriumi, 2Gb operatyviąja atmintine, kuriame įdiegta 32 bit Windows 7 operacinė sistema.

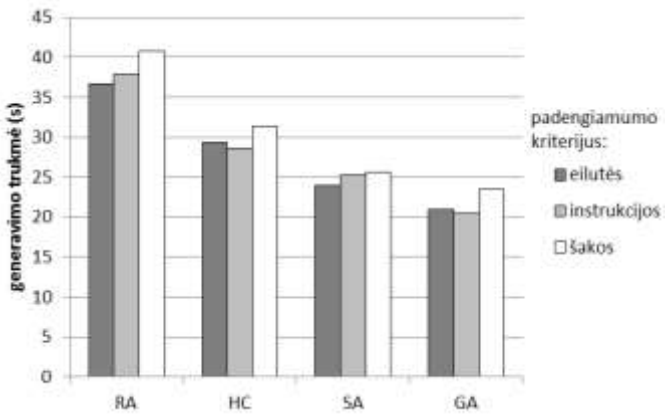
XI LENTELĖ. GENERAVIMO METODŲ PARAMETRŲ REIKŠMĖS

Parametras	**Reikšmė**	**Generavimo metodas**			
		RA	***HC***	***SA***	***GA***
minimalus tenkintinis padengiamumas	90%	X	X	X	X
maksimalus iteracijų kiekis	1000	X	X	X	
maksimalus vidinių iteracijų kiekis	50			X	
temperatūros mažinimo koeficientas	0,9			X	
populiacijos dydis	10				X
kryžminimo tikimybė	0,9				X
mutavimo tikimybė	0,01				X
maksimalus populiacijų skaičius	1000				X
short, int, long rėžiai	–10 - 1000	X	X	X	X
float, double režiai	–10,0 - 1000,0	X	X	X	X
string rėžiai	[a-z], [A-Z], [0-9]	X	X	X	X

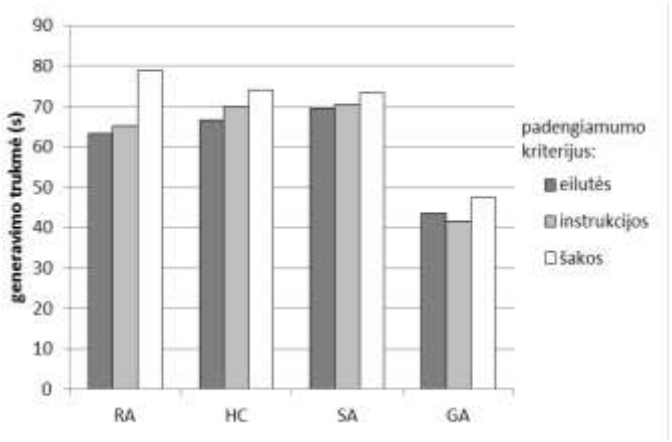
V. TYRIMO REZULTATAI

Generavimo metodų veikimo trukmės naudojant skirtingus padengiamumo kriterijus yra pateikiamos 2-4 pav. (2 pav. – sinuso skaičiavimo algoritmui, 3 pav. – įrašo išrinkimo algoritmui, 4 pav. – tetrio žaidimo logikos klasei).

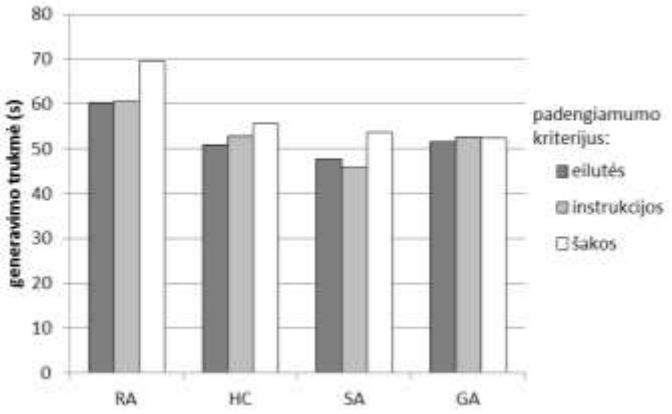
Iš šių rezultatų matyti, jog beveik visais atvejais testinių duomenų generavimas remiantis šakų padengiamumu užtrunka ilgiau negu remiantis instrukcijų arba kodo eilučių padengiamumais.



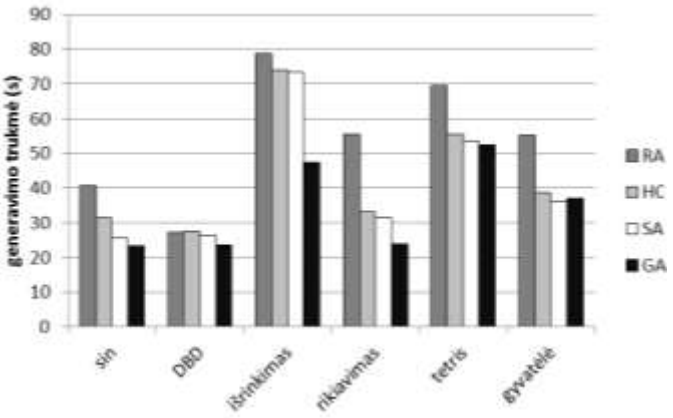
23 pav. Testinių duomenų generavimo sinuso skaičiavimo algoritmui trukmė



24 pav. Testinių duomenų generavimo įrašo išrinkimo algoritmui trukmė



25 pav. Testinių duomenų generavimo tetrio žaidimo logikos klasei trukmė



26 pav. Testinių duomenų generavimo trukmė skirtingiems algoritmams

Testinių duomenų generavimo trukmės palyginimas priklausomai nuo testuojamų programų pateikimas 5 pav. Jame pateikiamos trukmės yra gautos remiantis šakų padengiamumu, kuris iš visų TDMG palaikomų padengiamumo kriterijų teikia objektyviausią įvertį. Iš šio palyginimo matyti, jog RA trukmės skirtumas lyginat su kitais generavimo metodais labai svyruoja (rikiavimo algoritmo atveju net iki 40%) ir neturi jokio dėsningumo. RA veikimas

dažniausiai užtrunka ilgiausiai iš visų arba panašiai kaip HC bei SA, tačiau jis niekada nebūna greičiausias.

SA trukmė lyginant su HC visų generacijų metu yra mažesnė ir dauguma atvejų skiriasi nedaug (iki 6,5%). Didesnis skirtumas tik sinuso skaičiavimo algoritmo atveju (18%).

GA trukmė beveik visų generacijų metu yra trumpiausia. Matematinių algoritmų bei žaidimų logikos klasių atveju GA nežymiai skiriasi nuo HC ir SA. Tačiau dirbant su masyvų apdorojimo algoritmais GA užtrunka net iki 35% trumpiau (įrašo išrinkimo algoritmas).

VI. IŠVADOS

Išbandžius įvairius testinių duomenų generavimo metodus išsiaiškinta, jog tinkamai pasirinkus duomenų generavimo metodą generavimo procesą galima atlikti net iki 43% greičiau (skirtumas tarp RA (55,4 s) ir GA (23,8 s) trukmių masyvo rikiavimo atveju).

Remiantis atliktų tyrimų rezultatais galima teigti, kad testuojant matematinius algoritmus bei žaidimų logikos klases HC, SA bei GA trukmės atžvilgiu yra labai panašūs. Tačiau testuojant masyvų apdorojimo algoritmus labiausiai tam tinkamas yra GA, kuris veikia net iki 35% (skirtumas tarp SA (73,4 ms) ir GA (47,5 ms) trukmių įrašo išrinkimo atveju) greičiau negu kiti generavimo metodai.

VII. TOLIMESNI DARBAI

Tolimesniuose darbuose šia tema yra numatyta ištirti daugiau skirtingų programų tipų. Taip pat planuojama atlikti tyrimą, kurio metu bus stebima testinių duomenų generavimo metodų efektyvumo priklausomybė nuo jų parametrų reikšmių (2 lentelė).

VIII. LITERATŪRA

[1] G. J. Myers, C. Sandler, T. Badgett, T. M. Thomas, “The art of software testing”, ISBN-978-0471469124, Wiley, 2004.

[2] B. Antonia, "Software Testing research: achievements, challenges, dreams," 2007 Future of software engineering: IEEE Computer Society, 2007.

[3] T. Garret, “Implementing automated software testing continously track progress and sdjust sccordingly, methods & tools, practical knowledge for software developer, tester and project manager fall” 2009 (Volume 17 – number 3), ISSN: 1661-402X, 15-30 psl., 2009.

[4] L. Baresi, M. Young, “Test oracles”, Technical report CIS-TR-01-02, 2001.

[5] P. B. Nirpal & K. V. Kale, "Comparison of software test data for automatic path coverage using genetic algorithm", International Journal of Computer Science & Engineering Technology (IJCSET), ISSN: 2229-3345, Vol. 1 No. 1, Sep 2012.

[6] S. Ali, L. C. Briand, H. Hemmati, R. K. Panesar-Walawege, “A systematic review of the application and empirical investigation of evolutionary testing”, Software Engineering: IEEE Transactions on, ISSN: 0098-5589, Vol. 36, Issue 6, 742-762 psl., 2010.

[7] D. Saff, M. D. Ernst, “Automatic mock object creation for test factoring”, ACM SIGPLAN/SIGSOFT workshop on program analysis for software tools and engineering (PASTE'04), (Washington, DC, USA), 49-51 psl., 2004.

[8] S. Zhang, D. Staff, Y. Bu, M. D. Ernst, “Combined static and dynamic automated test generation”, ISSTA ‘11, (Toronto, ON, Canada) 17-21 psl., 2011.

[9] Bj Rollison, “Parametrized random test data generation”, PNSQC, 2011.

[10] J. A. Edvardsson, “Survey on automatic test data generation” Second Conference on Computer Science and Engineering in Linkoping, 1999.

[11] A. Arcuri, P. K. Lehre, Xin Yao, “Theoretical runtime analyses of search algorithms on the test data generation for the triangle classification problem”, Software testing verification and validation workshop, 2008. ICSTW ‘08. IEEE International conference., 161-169 psl. 2008.

[12] K. Ghani, J. A. Clark, “Automatic test data generation for multiple condition and MCDC coverage”, Fourth International Conference on Software Engineering Advances (ICSEA ‘09), 152-157 psl., 2009.

[13] A. Gotlieb, T. Denmat B. Botella, “Goal-oriented test data generation for programs with pointer variables”, Raport de recherche n°5528, Institut national de recherche en informatique et en automatique, ISSN: 0249-6399, 2005.

[14] E. K. Prebys, “The genetic algorithm in computer science”, MIT undergraduate journal of mathematics, 165-170 psl., 2007.

[15] M. Harman, P. McMinn, "A theoretical empirical analysis of evolutionary testing and hill climbing for structural test data generation," International symposium on software testing and analysis ’07, London, United Kingdom: ACM, 2007.

[16] J. Miller, M. Reformat, H. Zhang, "Automatic test data generation using genetic algorithm and program dependence graphs", Information and Software Technology, vol. 48, 586-605 psl., 2006.

[17] S. Kanmani, P. Maragathavalli, “Search-based software test data generation using evolutionary testing techniques”, International journal of software engineering (IJSE), 10-22 psl., 2010.

[18] M. Xiao, M. El-Attar, M. Reformat, J. Miller, "Empirical evaluation of optimization algorithms when used in goal-oriented automated test data generation techniques," Empirical Software Engineering, vol. 12, 183-239 psl., 2007.

[19] JaCoCo Java Code Coverage Library, Mountainminds GmbH & Co, [interaktyvus]. http://www.eclemma.org/jacoco/, [žiūrėta 2014-03-02].

ANALYSIS OF AUTOMATIC TEST DATA GENERATION ALGORITHMS

G. Motiejūnas, T. Kuckis, D. Liepinaitis

IX. SUMMARY

This paper analyzes automated test data generation methods. Analyzed methods are Random generator and few methods based on optimization algorithms such as Hill Climbing, Simulated Annealing and Genetic algorithm. During research all mentioned methods are compared to each other by generating test data for various specifics test programs. After research the recommendations for users, who generates test data in automated way, are formulated.

B. Pažyma apie straipsnio „Automatinio testinių duomenų generavimo metodų tyrimas“ pristatymą konferencijoje

KAUNO TECHNOLOGIJOS UNIVERSITETAS

DALYVIO PAŽYMĖJIMAS

2014-04-24 Nr. ST25-F-14-59

Kaunas

Pažymima, kad

Gintautas Motiejūnas, Titas Kuckis, Darius Liepinaitis (KTU)

2014 m. balandžio 24 d. dalyvavo tarpuniversitetinėje konferencijoje „Informacinė visuomenė ir universitetinės studijos“ (IVUS 2014) ir perskaitė pranešimą

Automatinio testinių duomenų generavimo metodų tyrimas

Informatikos fakulteto dekanas prof. Eduardas Bareiša

Programų komiteto pirmininkas prof. Robertas Damaševičius

C. Straipsnis „Klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant laikines charakteristikas, kūrimas ir tyrimas“

Klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant laikines charakteristikas, kūrimas ir tyrimas

Darius Liepinaitis, Gintautas Motiejūnas, Titas Kuckis
Programų inžinerijos katedra. Informatikos fakultetas.
KTU
Kaunas, Lietuva
darius.liepinaitis@stud.ktu.lt, gintautas.motiejunas@stud.ktu.lt, titas.kuckis@stud.ktu.lt

***Anotacija* — Šiame straipsnyje yra aprašomas klaidų paieškos mobilių įrenginių programinėje įrangoje, taikant laikines charakteristikas metodas bei tyrimas. Tyrimo metu šis metodas yra lyginamas su funkcionalumo testu ir siekiama išsiaiškinti, kaip gerai šis metodas suranda klaidas. Atlikus tyrimą yra sudarytos rekomendacijos, kaip galima būtų pasinaudoti sukurtu metodu.**

Raktiniai žodžiai — klaidų paieška; grafinės vartotojo sąsajos testavimas; automatizuotas testavimas; laikinė charakteristika;

I. ĮVADAS

Mobiliųjų telefonų programų šiuo metu yra sukurta labai daug ir jos yra kuriamos toliau. Tačiau didelė dalis šių programų nėra sėkmingos ir plačiai naudojamos, nors iš pirmo žvilgsnio ir teikia vartotojui reikalingas funkcijas. Daugelis iš jų neatitinka vartotojų poreikių, kadangi veikia neteisingai ir vartotojas negali pasinaudoti funkcijomis, kurios turėtų veikti pagal aprašymą.

Vienas žinomiausių būdų užtikrinti programinės įrangos aukštai kokybei yra programinės įrangos testavimas. Programinės įrangos testavimas yra procesas, kurio tikslas nustatyti problemas (defektus), dėl kurių sistema negali atitikti vartotojo lūkesčių. Šis procesas apima sistemos ar taikomosios programos vykdymą kontroliuojamomis sąlygomis ir gautų rezultatų vertinimą, bet neapsiriboja tuo [1].

Programinės įrangos testavimas reikalauja didelių laiko sąnaudų ir resursų [2]. Jeigu kuriama programinė įranga yra sudėtinga, testavimas gali užtrukti 50 – 75 procentų bendro programinės įrangos kūrimo laiko [3]. Taip pat testavimui yra skiriami dideli žmogiškieji resursai [4], kadangi yra reikalingi žmonės su atskira testuotojo profesija [5]. Šio žmogaus pagrindinė užduotis yra patikrinti, ar visos funkcijos veikia taip, kaip turėtų veikti, ir ar jos grąžina rezultatą per tam tikrą (numatytą) laiką. Dėl šios priežasties nemažai programinės įrangos kūrėjų apriboja vykdomų testų skaičių. Apribojus vykdomų testų skaičių daugeliu atvejų yra sumažinama programinės įrangos kokybė [6].

Siekiant sumažinti programinės įrangos testavimo kaštus ir resursų panaudojimą, programinės įrangos testavimas yra automatizuojamas [7]. Automatizavus testavimą, resursų ir kaštų sumažėjimas pasireiškia ilgalaikėje perspektyvoje, t.y. vykdant automatizuotus testus pakartotinai. Taip pat automatizuotas testavimas testų rezultatus kiekvieną kartą interpretuoja vienodai, skirtingai negu vykdant rankinius testus. Testuotojas vykdydamas rankinius testus gali skirtingai įvertinti tą patį operacijos vykdymo laiką.

Ypatingai daug testuotojų resursų reikalauja grafinės vartotojo sąsajos testavimas [8, 9]. Šiam testavimui atlikti yra reikalinga labai tiksliai aprašyti testavimo atvejus, kad testuotojas galėtų nuspręsti, ar ekrane rodoma informacija yra teisinga [10, 11, 12]. Kaip ir tradicinio testavimo atveju, taip ir grafinės vartotojo sąsajos testavimą yra siekiama automatizuoti [13]. Grafinės sąsajos automatizavimas yra sudėtingesnis negu tradicinio testavimo, bet grafinės sąsajos automatizavimas labai stipriai sumažina testavimo kaštus [14].

Šio straipsnio tikslas yra išanalizuoti sukurtą klaidų paieškos mobilių įrenginių programinėje įrangoje metodą, taikant laikines charakteristikas. Toliau yra pateikiama šio straipsnio struktūra. II skyriuje pateikiama panašių metodų apžvalga. III skyriuje aprašomas sukurtas klaidų paieškos mobilių įrenginių programinėje įrangoje metodas, taikant laikines charakteristikas. IV skyriuje pateikiama tyrimo metodika ir apžvelgiami panaudoti įrankiai tyrimui atlikti. V skyriuje pateikiami atlikto tyrimo rezultatai. VI skyriuje pateikiamos išvados (apibendrinti tyrimo rezultatai). VII skyriuje pateikiami planuojami ateities darbai.

II. PANAŠŪS METODAI

Panašus metodas yra grafinės vartotojo sąsajos našumo testavimas [15]. Šis metodas yra panašus tuo, kad: (1) yra naudojamos laikinės charakteristikos išmatuoti įvykdyto veiksmo laiką, (2) visi veiksmai yra inicijuojami iš grafinės vartotojo sąsajos. Šis metodas skiriasi tuo, kad: (1) jis yra naudojamas kitoje platformoje, lyginant su sukurtu metodu, (2) jis padengia ir funkcionalumo testus, kurių nepadengia sukurtas metodas.

Taip pat panašus metodas yra įrašo atkartojimas (angl. „*record-playback*“) [16]. Šis metodas panašus tuo, kad visi veiksmai inicijuojami iš grafinės vartotojo sąsajos. Skiriasi metodas tuo, kad: (1) norint juo pasinaudoti, reikia iš pradžių testuotojui padaryti visus veiksmus su testuojama programa, o po to tie veiksmai bus atkartojami, (2) šis metodas apima tik funkcionalumo testus.

III. KLAIDŲ PAIEŠKOS MOBILIŲ ĮRENGINIŲ PROGRAMINĖJE ĮRANGOJE METODAS

Klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant laikines charakteristikas, veikimas yra pagrįstas grafinės vartotojo sąsajos testavimu, kartu matuojant kiekvienos atliktos grafinės sąsajos operacijos laiką. Grafinės sąsajos operacijos gali būti įvairios: mygtuko paspaudimas, teksto įvedimas, teksto atsiradimas ir pan. Taip pat visi veiksmai, kurie yra galimi mobiliems įrenginiams su lietimui jautriu ekranu: braukimas į kairę/dešinę, aukštyn/žemyn ir pan.

Šio metodo idėja – nespėjus atlikti veiksmų per tam tikrą laiką, yra laikoma, kad testas nepavyko.

A. *Reikalavimai metodui įvykdyti*

Norint pasinaudoti šiuo metodu yra reikalinga išpildyti tokias sąlygas:

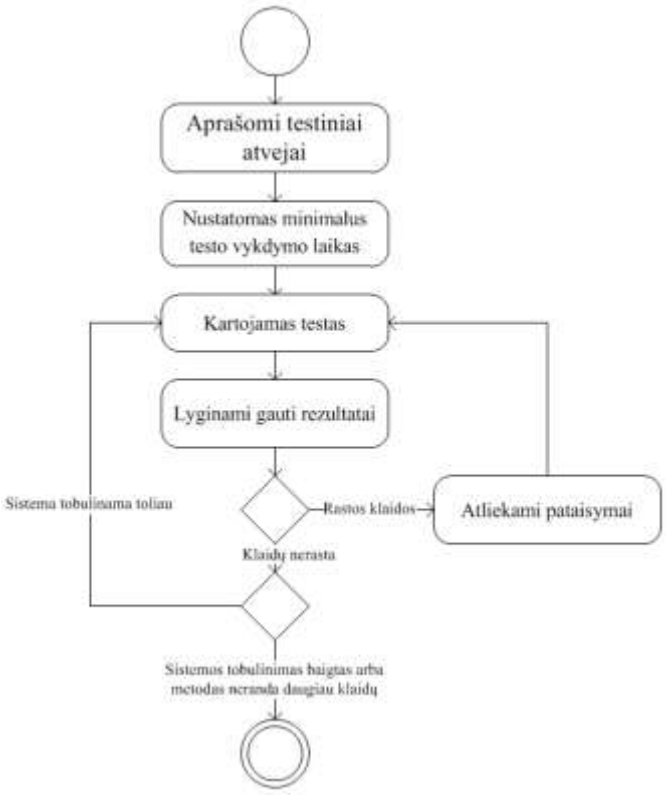
- Testuojama programinė įranga turi būti kuriama mobiliems įrenginiams.
- Testuojama programinė įranga turi turėti grafinę vartotojo sąsają.
- Turi būti prieinamas testuojamos programinės įrangos išeities kodas.

B. *Metodo aprašymas*

Žemiau yra pateikiami žingsniai, kaip pasinaudoti metodu:

1. Aprašyti testinius atvejus. Testiniai atvejai turi būti aprašomi nesudėtingai, t.y. tik komandos inicijavimas ir rezultatų laukimas.
2. Vykdyti testą ir nustatyti minimalų testo vykdymo laiką. Šiame žingsnyje yra gaunami fiksuoti rezultatai.
3. Padarius pakeitimus testuojamoje programoje, pakartoti testą.
4. Sulyginti gautus rezultatus su fiksuotais rezultatais.
5. Jeigu gauti rezultatai yra didesni nei fiksuoti, yra reikalinga atlikti pataisymus testuojamoje programoje ir vykdyti vėl, pradedant trečiuoju žingsniu.
6. Jeigu gauti rezultatai mažesni arba lygūs fiksuotiems rezultatams, vykdyti tolimesnius testuojamos programos kūrimo darbus.

Taip pat 1 pav. yra pateikiama metodo veiklos diagrama.



27 pav. Klaidų paieškos mobilių įrenginių programinėje įrangoje metodo veiklos diagrama

C. *Testavimo pabaigos sąlyga*

Testavimas turi būti baigiamas tuomet, kai testuojama programa yra sukurta pilnai ir gaunami rezultatai yra mažesni arba lygūs fiksuotiems rezultatams.

D. *Metodo pritaikymas*

Klaidų paieškos mobilių įrenginių programinėje įrangoje metodas, taikant laikines charakteristikas, yra taikytas naudoti mobiliųjų telefonų programoms, kurios yra kuriamos šioms operacinėms sistemoms: (1) Microsoft Windows Phone, (2) Google Android, (3) Apple IOS.

Aukščiau išvardintos operacinės sistemos yra labiausiai paplitę, todėl ir šis metodas yra labiau taikytas joms. Tačiau jeigu programinė įranga yra kuriama kitokioms mobilioms operacinėms sistemoms, šis metodas taip pat galėtų būti pritaikomas jose. Pagrindiniai reikalavimai, kurie turi būti išpildomi yra pateikiami šio skyriaus A dalyje „*Reikalavimai metodui įvykdyti*“.

IV. TYRIMO METODIKA IR NAUDOTI ĮRANKIAI

A. *Naudoti įrankiai*

Metodo tyrimui atlikti buvo sukurtas įrankis „WindowsPhoneGUITestTool“. Šis įrankis buvo sukurtas klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant laikines charakteristikas, pagrindu.

Sukurtas įrankis „WindowsPhoneGUITestTool“ teikia tokias galimybes:

- Testų vykdymas emuliatoriuje ir/arba realiame išmaniajame telefone.
- Testų vykdymo laiko išmatavimas.
- Vykdytų testų rezultatų peržiūra.
- Vykdytų testų rezultatų palyginimas.
- Testų scenarijų redagavimas.

Naudojantis šiomis teikiamomis funkcijomis buvo atlikti eksperimentai.

Įrankio „WindowsPhoneGUITestTool“ testų vykdymo procesas yra pateikiamas žemiau:

1. Startuojama testuojama programa.
2. Į testuojamą programą siunčiama komanda, ką reikia įvykdyti.
3. Laukiamas rezultatas iš testuojamos programos.
4. Kartojami žingsniai 2 ir 3 tol, kol bus įvykdytos visos nurodytos komandos.
5. Susumuojami rezultatai ir išvedami į failą.

B. Tyrimo metodika

Tyrimui buvo pasirinktos 2 programos, priklausančios skirtingoms programų grupėms. Pasirinktos testuoti programų grupės yra: (1) skaičiavimų programos, (2) informacinės programos.

Programos buvo testuojamos Microsoft Windows Phone 8.0 operacinės sistemos aplinkoje, o realizuotos C# programavimo kalba.

Atliekant tyrimą buvo sudaryta po du testus kiekvienai programai. Pirmasis testas buvo sudarytas pagal klaidų paieškos mobilių įrenginių programinėje įrangoje metodą, taikant laikines charakteristikas. Antrasis testas buvo sudarytas pagal funkcionalumo testą, t.y. buvo stengiamasi išgauti kuo geresnį testą, kuris padengtų kuo didesnę aibę įėjimo duomenų. Sudarant testus taip pat buvo skaičiuojamas ir jų sudarymo laikas. Tai buvo vykdoma, pavedant testuotojams sudaryti nurodytoms programoms automatinius testus, siekiant gauti 100% kodo padengimą. Naudojantis projekto valdymo įrankiais buvo nustatytas vidutinis laikas, kurio prireikdavo testuotojams sukurti visus testus. Vėliau buvo fiksuojami minimalūs abiejų programų vykdymo laikai.

Sudarius testus ir fiksavus minimalias vykdymo reikšmes, kiekvienai programai buvo generuojami mutantai. Mutantų generavimui pasirinkta „CREAM“ [17] mutantų generavimo C# programavimo kalbai programinė įranga. Šios programos generuojami mutantų tipai ir jų aprašymai yra pateikiami I lentelėje.

Tyrimas buvo atliktas kompiuteryje, kuris turėjo žemiau pateikiamas charakteristikas:

- Intel Core i7 keturių branduolių, 2.20 GHz procesorius.
- 8Gb operatyvioji atmintinė.
- Windows 8.1 Pro 64bit operacinė sistema.

I LENTELĖ. MUTANTŲ TIPAI

Mutanto tipas	Mutanto aprašymas
ABS	Absoliutinės reikšmės įterpimas
AOR	Aritmetinio operatoriaus pakeitimas
ASR	Masyvo rodyklės pakeitimas skaliariniu kintamuoju
LCR	Loginio jungėjo pakeitimas
LOR	Loginio operatoriaus pakeitimas
ROR	Reliacinio operatoriaus pakeitimas
UOI	Vienanario operatoriaus įterpimas
UOR	Vienanario operatoriaus pakeitimas
DMC	Deleguoto metodo pakeitimas
EHR	Išimties apdorojimo panaikinimas
EOA	Rodyklės priskyrimas ir turinio priskyrimo pakeitimas
EOC	Rodyklės palyginimas ir turinio palyginimo pakeitimas
EXS	Išimties „prarijimas“
IHD	Paslepiamas kintamojo ištrynimas
IHI	Paslepiamas kintamojo įterpimas
IOD	Perdengiamojo metodo ištrynimas
IOK	„Override“ raktažodžio pakeitimas
IOP	Perdengiamojo metodo iškvietimo vietos pakeitimas
IPC	Tiesioginis tėvinės klasės konstruktoriaus ištrynimo iškvietimas
ISK	„Base“ raktažodžio ištrynimas
JID	Kintamojo inicializavimo ištrynimas
JTD	„This“ raktažodžio ištrynimas
OAO	Argumentų tvarkos sukeitimas
OMR	Perdengiamo metodo turinio pakeitimas
PRM	Savybės (angl. „*Property*“) pakeitimas į kintamąjį
PRV	Rodyklės priskyrimas su kitu tinkamu tipu

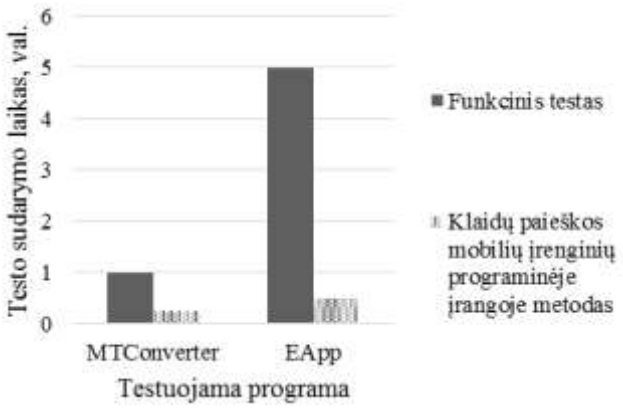
V. TYRIMO REZULTATAI

Testų sudarymo laikų rezultatai pateikiami II lentelėje.

Minimalūs fiksuoti laikai testuojamoms programoms yra: (1) EApp programai - 69 sekundės, (2) MTConverter programai - 80 sekundžių. Rezultatų palyginimas pateikiamas 28 pav.

II LENTELĖ. TESTŲ SUDARYMO LAIKAI

Programos pavadinimas	Testo sudarymo laikas	
	Funkcinis testas	Klaidų paieškos mobilių įrenginių programinėje įrangoje metodas
MTConverter	1 valanda	15 minučių
EApp	5 valandos	30 minučių



28 pav. Testuojamų programų testų sudarymo laikų palyginimas

Programos EApp funkcinio testo ir klaidų paieškos mobilių įrenginių programinėje įrangoje metodo sugeneruotų mutantų ir aptiktų bei neaptiktų mutantų rezultatai pateikiami III ir IV lentelėse. Šiose lentelėse yra pateikiami tik tie mutantai, kurie buvo sugeneruoti.

Iš viso EApp programai buvo sugeneruota 14 mutantų. Funkcinio testo metu buvo aptikta 7 mutantai, o klaidų paieškos mobilių įrenginių programinėje įrangoje metodo testo metu buvo aptikta tik 4 mutantai. Palyginimo rezultatai yra pateikiami 29 pav.

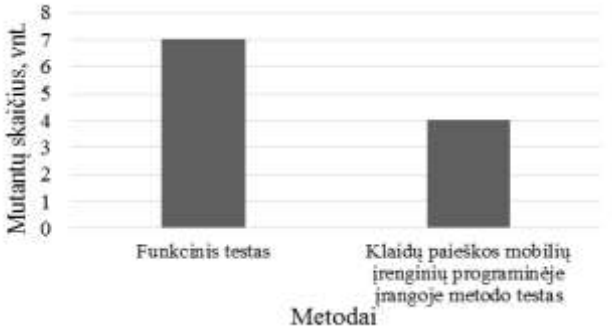
Programos MTConverter funkcinio testo ir klaidų paieškos mobilių įrenginių programinėje įrangoje metodo atliktų testų rezultatai yra pateikiami V ir VI lentelėse. Šiose lentelėse pateikiami tik tie mutantai, kurie buvo sugeneruoti, t.y. mutantai, kurie nebuvo sugeneruoti, nepateikiami.

III LENTELĖ. FUNKCINIO TESTO EAPP PROGRAMOS REZULTATAI

Mutanto tipas	Mutantų skaičius	Neaptiktų mutantų skaičius	Aptiktų mutantų skaičius
UOI	6	2	4
IOP	6	4	2
ISK	1	0	1
JID	1	1	0

IV LENTELĖ. KLAIDŲ PAIEŠKOS MOBILIŲ ĮRENGINIŲ PROGRAMINĖJE ĮRANGOJE METODO TESTO EAPP PROGRAMOS REZULTATAI

Mutanto tipas	Mutantų skaičius	Neaptiktų mutantų skaičius	Aptiktų mutantų skaičius
UOI	6	3	3
IOP	6	6	0
ISK	1	0	1
JID	1	1	0



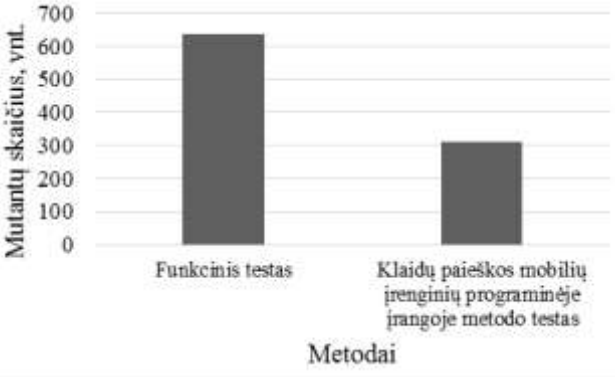
29 pav. Aptiktų mutantų EApp programoje skirtingais metodais palyginimas

V LENTELĖ. FUNKCINIO TESTO MTCONVERTER PROGRAMOS REZULTATAI

Mutanto tipas	Mutantų skaičius	Neaptiktų mutantų skaičius	Aptiktų mutantų skaičius
ABS	40	16	24
AOR	20	0	20
ASR	112	37	75
LCR	8	3	5
ROR	180	34	146
UOI	274	79	195
UOR	39	15	24
EOC	36	13	23
IOP	1	0	1
IPC	0	0	0
ISK	1	0	1
JID	21	5	16
JTD	126	46	80
OAO	37	10	27

VI LENTELĖ. KLAIDŲ PAIEŠKOS MOBILIŲ ĮRENGINIŲ PROGRAMINĖJE ĮRANGOJE METODO TESTO MTCONVERTER PROGRAMOS REZULTATAI

Mutanto tipas	Mutantų skaičius	Neaptiktų mutantų skaičius	Aptiktų mutantų skaičius
ABS	40	26	14
AOR	20	0	20
ASR	112	77	35
LCR	8	4	4
ROR	180	124	56
UOI	274	182	92
UOR	39	23	16
EOC	36	22	14
IOP	1	0	1
ISK	1	0	1
JID	21	13	8
JTD	126	95	31
OAO	37	19	18



30 pav. Aptiktų mutantų MTConverter programoje skirtingais metodais palyginimas

Programai MTConverter buvo sugeneruota 895 mutantai. Funkcinio testo metu buvo aptikta 637 mutantai. Klaidų paieškos mobilių įrenginių programinėje įrangoje, taikant laikines charakteristikas, metodu buvo aptikta 310 mutantų. Palyginimo rezultatai yra pateikiami 30 pav.

Iš gautų rezultatų matoma, kad sukurtas metodas aptiko EApp programos atveju beveik du kartus mažiau klaidų, o MTConverter programos atveju - daugiau nei du kartus mažiau klaidų, lyginant su funkcionalumo testo atveju.

VI. IŠVADOS

Išanalizavus dviejų programų tipų programas buvo išsiaiškinta, kad naudojantis klaidų paieškos mobilių įrenginių programinėje įrangoje, taikant laikines charakteristikas, metodu, klaidų aptikimas yra prastesnis: (1) EApp programos atveju 1,75 karto prastesnis, (2) MTConverter programos atveju 2,05 karto prastesnis. Iš šių rezultatų galime teigti, kad funkcionalumo testo sukurtas metodas negali pakeisti.

Lyginant, kiek yra užtrunkama gauti pirmiesiems testavimo rezultatams, t.y. kaip greitai galima atrasti pirmąsias klaidas, sukurtas metodas yra galimas naudoti. Funkcionalumo testą kuriant EApp programai buvo užtrukta 1 valandą ir gautos 7 klaidos, tačiau sukurto metodo atveju per 15 minučių buvo gauta 4 klaidos. MTConverter programos atveju funkcionalumo testą sudaryti buvo užtrukta 5 valandas ir surastos 637 įvestos klaidos, tačiau su sukurtu metodu užtrukus 30 minučių, buvo surastos 310 klaidų.

Iš visų gautų rezultatų galime teigti, kad skaičiavimų ir informacinėms programoms yra naudinga naudoti klaidų paieškos mobilių įrenginių programinėje įrangoje, taikant laikines charakteristikas, metodą, pradedant programų kūrimą ir naudoti tol, kol naudojantis juo yra neatrandamos klaidos. Po to yra prasminga naudoti funkcionalumo arba kitus testus.

VII. ATEITIES DARBAI

Tolimesni planuojami darbai šia tema yra galimi, apimant didesnį kiekį programų tipų. Taip pat yra norima patikrinti šį metodą ne tik su Microsoft Windows Phone OS bet ir su kitomis mobiliųjų įrenginių operacinėmis sistemomis.

Taip pat yra planuojama tobulinti metodą, įtraukiant laiko matavimą kiekvienai įvykdytai funkcijai, t.y. planuojama, jog bus matuojamas bendras programos vykdymo laikas ir kiekvienos įvykdytos funkcijos laikas atskirai.

VIII. LITERATŪRA

[1] K. Lochmann, A. Goeb, “A Unifying Model for Software Quality”, WoSQ '11 Proceedings of the 8th international workshop on Software quality, 3-10, 2011.

[2] A. Bertolino, “Software Testing Research: Achievements, Challenges, Dreams”, FOSE '07 2007 Future of Software Engineering, 86-103, 2007.

[3] B. Hailpern, P. Santhanam, “Software debugging, testing, and verification”, IBM Systems Journal, ISSN: 0018-8670, Vol. 41, Issue 1, 4-12, 2002.

[4] M. Lyu, Handbook of Software Reliability Engineering, ISBN0-07-039400-8, McGraw-Hill, 3-22, 1996.

[5] R. Patton, Software Testing, ISBN: 0-672-31983-7, SAMS, 9-53, 2001.

[6] S. Berner, R.Weber, R. Keller, “Observations and lessons learned from automated testing”. ICSE '05 Proceedings of the 27th international conference on Software engineering, 571–579, 2005.

[7] P. Koopman, A. Alimarine, J. Tretmans, R. Plasmeijer, “Gast: Generic Automated Software Testing”, Springer Berlin Heidelberg, ISSN: 0302-9743, 84-100, 2002.

[8] P. Li, T. Huynh, M. Reformat, J. Miller, “A practical approach to testing gui systems”, Empirical Software Engineering, Vol. 12, Nr. 4, 331 – 357, 2007.

[9] T. H. Chang, T. Yeh, R. C. Miller, “GUI Testing Using Computer Vision”, CHI '10 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 1535-1544, 2010

[10] J. Prabhu, N. Malmurugan, “A survey on Automated GUI testing Procedures”, European Journal of Scientific Research. ISSN: 1450-216X, Vol. 64, Nr. 3, 456-462, 2011.

[11] Q. Xie, “Developing cost-effective model-based techniques for GUI testing” ICSE '06 Proceedings of the 28th international conference on Software engineering, 997-1000, 2006.

[12] Q. Xie, A. M. Memon, “Designing and Comparing Automated Test Oracles for GUI-Based Software Applications”, ACM Transactions Software Engineering and Methodology, Vol. 16, No. 1, 2007.

[13] P. A. Brooks, A. M. Memon, “Automated GUI Testing Guided by Usage Profiles”, ASE '07 Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, 333-342, 2007.

[14] P. Li, T. Huynh, M. Reformat, J. Miller, “A practical approach to testing GUI systems”, Empirical Software Engineering, Vol. 12. Issue: 4, ISSN: 1382-3256, 331-357, 2007.

[15] A. Adamoli, D. Zaparanuks, M. Jovic, M. Hauswirth, “Automated GUI performance testing”, Software Quality Journal, Vol. 19, Issue: 4, ISSN: 0963-9314, 801-839, 2011.

[16] Atif M. Memon, "GUI Testing: Pitfalls and Process", Computer, Vol. 35, no. 8, 87-88, 2002.

[17] CREAM, Creator of Mutants, [interaktyvus]. http://galera.ii.pw.edu.pl/~adr/CREAM, [žiūrėta 2014-03-03].

THE ERRORS SEARCH METHOD FOR MOBILE DEVICES SOFTWARE USING TIMING CHARACTERISTICS

D. Liepinaitis, G. Motiejūnas, T. Kuckis

IX. SUMMARY

This paper describes errors search in mobile devices software using time characteristics method and its research. During research created method is compared with functionality

test. When comparing methods the goal is to see how well created method is performing compared with functionality test. After research, recommendations were created, how to use created method.

D. Pažyma apie straipsnio „Klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant laikines charakteristikas, kūrimas ir tyrimas“ pristatymą konferencijoje

KAUNO TECHNOLOGIJOS UNIVERSITETAS

DALYVIO PAŽYMĖJIMAS

2014-04-24 Nr. STIS-F-14-60

Kaunas

Pažymima, kad

Darius Liepinaitis, Gintautas Motiejūnas, Titas Kuckis (KTU)

2014 m. balandžio 24 d. dalyvavo tarpuniversitetinėje konferencijoje „Informacinė visuomenė ir universitetinės studijos“ (IVUS 2014) ir perskaitė pranešimą

Klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant laikines charakteristikas, kūrimas ir tyrimas

Informatikos fakulteto dekanas prof. Eduardas Bareiša

Programų komiteto pirmininkas prof. Robertas Damaševičius

E. Pažyma apie geriausią sekcijos straipsnį konferencijoje

KAUNO TECHNOLOGIJOS UNIVERSITETAS

GERIAUSIO STRAIPSNIO DIPLOMAS

2014-04-24 Nr. ST25-F-14-88

Kaunas

Pažymima, kad

Dariaus Liepinaičio, Gintauto Motiejūno ir Tito Kuckio (KTU)

pristatytas straipsnis

Klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant laikines charakteristikas, kūrimas ir taikymas

2014 m. balandžio 24 d. vykusioje tarpuniversitetinėje konferencijoje „Informacinė visuomenė ir universitetinės studijos“ (IVUS 2014) yra išrinktas geriausiu „IT testavimo ir saugos“ sekcijos straipsniu.

Programų komiteto pirmininkas prof. Robertas Damaševičius