KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

TITAS KUCKIS

TESTINIŲ DUOMENŲ GENERAVIMO ALGORITMŲ EFEKTYVUMO PRIKLAUSOMYBĖS NUO PASIRINKTŲ PARAMETRŲ ANALIZĖ IR TYRIMAS

Magistro darbas

Darbo vadovas
doc. dr. Š. Packevičius

Kaunas, 2014

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

TITAS KUCKIS

TESTINIŲ DUOMENŲ GENERAVIMO ALGORITMŲ EFEKTYVUMO PRIKLAUSOMYBĖS NUO PASIRINKTŲ PARAMETRŲ ANALIZĖ IR TYRIMAS

Magistro darbas

Vadovas
doc. dr. Š. Packevičius
_____
(data, parašas)

Recenzentas
lekt. dr. D. Barisas
_____
(data, parašas)

Studentas
T. Kuckis
_____
(data, parašas)

Kaunas, 2014

AUTENTIŠKUMO PATVIRTINIMAS

Autorius, Titas Kuckis
(vardas, pavardė)

patvirtina, kad Kauno technologijos universitetui pateiktas baigiamasis magistro darbas (toliau vadinama – Kūrinys)
TESTINIŲ DUOMENŲ GENERAVIMO ALGORITMŲ EFEKTYVUMO PRIKLAUSOMYBĖS NUO PASIRINKTŲ PARAMETRŲ ANALIZĖ IR TYRIMAS
(kūrinio pavadinimas)

pagal Lietuvos Respublikos autorių ir gretutinių teisių įstatymą yra originalus ir užtikrina, kad

1. jį sukūrė ir parašė Kūrinyje įvardyti autoriai;
2. Kūrinys nėra ir nebus įteiktas kitoms institucijoms (universitetams) (tiek lietuvių, tiek užsienio kalba);
3. Kūrinyje nėra teiginių, neatitinkančių tikrovės, ar medžiagos, kuri galėtų pažeisti kito fizinio ar juridinio asmens intelektinės nuosavybės teises, leidėjų bei finansuotojų reikalavimus ir sąlygas;
4. visi Kūrinyje naudojami šaltiniai yra cituojami (su nuoroda į pirminį šaltinį ir autorių);
5. neprieštarauja dėl Kūrinio platinimo visomis oficialiomis sklaidos priemonėmis.

_____	_____	_____
(studento vardas, pavardė)	(data)	(parašas)

SUMMARY

Programs and their systems are rapidly entering people's lives, and even people with less knowledge of information technology, are forced to use vide variety of programs. Even in large, well-funded programs, errors are inevitable. The question is how to ensure quality of programs in small, poorly-funded programs. The cheapest, simplest and most frequently used software quality assurance and improvement method is testing. Test automation further reduces the cost of testing, but also brings new problems as the oracle problem, and automatic generation of test data.

This paper describes the most popular algorithms for generating test data. These algorithms are implemented as part of test data generation extension for Eclipse programming environment created in project phase. Created product is tested and its expansion and improvement opportunities are envisaged.

In the experimental part test data generation algorithm efficiency dependence on used parameters are researched. In the conclusions are presented test data generator parameters selection proposals based on experimental findings like observation that ElitismValue parameter in genetic generator is best used with small populations.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLĖLIŲ SĄRAŠAS

1. ĮVADAS

1.1. Dokumento paskirtis

Šis dokumentas yra autoriaus magistrantūros studijų baigiamasis darbas kuriame pateikiamas autoriaus atliktas darbas visų magistrantūros studijų metu. Darbas priklauso Kauno technologijos universiteto magistratūros studijų programų sistemų inžinerijos programai.

1.2. Darbo tikslas ir uždaviniai

Nors programų kūrimas yra sparčiai besiplėtojanti sritis, bet programų kokybės užtikrinimas vis dar susiduria su daugybe problemų kaip:

- klaidų kiekis išleidžiamose programose nemažėja;
- Metodologijos skirtos sumažinti klaidų kiekį programinėje įrangoje yra brangios
- testavimo įrankiai yra brangus;
- testavimo įrankių veikimas nėra visiškai aiškus dėl gausybes parametrų nuo kurių priklauso algoritmo veikimo efektyvumas;
- Specialistų samdymas programų testavimui yra labai brangus ir netgi ne visada įmanomas;
- Testavimo kaina ir žinių trūkumas testavimo srityje yra vieni didžiausių programinės įrangos projektų kokybės užtikrinimą stabdančių veiksnių.

Šiame dokumente aprašomo darbo tikslas yra:

- sumažinti testavimo kainą;
- sumažinti laiko kiekį reikalingą surasti optimalius testinių duomenų generavimo generatoriaus parametrus;
- palengvinti testinių duomenų generavimą.

Darbo tikslui pasiekti keliami uždaviniai:

- sukurti nemokamą testinių duomenų generavimo programinę įrangą
- atlikti testinių duomenų generavimo algoritmų efektyvumo priklausomybės nuo parametrų tyrimą
- pateikti testinių duomenų generatorių pasirinkimo rekomendacijas

1.3. Santrauka

Programoms ir jų sistemos sparčiai besiveržiant į žmonių gyvenimus, net ir mažiau informacines technologijas išmanantys žmonės yra priversti naudoti įvairias programas. Žinant, kad net ir didelėse, gerai finansuotuose programų sistemose, klaidos yra neišvengiamos, kyla klausimas kaip užtikrinti programų kokybę mažose, menkai finansuojamose programose. Pigiausias, dažniausiai naudojamas ir paprasčiausias programų kokybės užtikrinimo ir kėlimo būdų yra testavimas. Testavimo automatizavimas dar labiau sumažina testavimo kainą, bet sukelia naujų problemų kaip orakulo problema ir automatinis testinių duomenų generavimas.

Šiame darbe aprašomi populiariausi testinių duomenų generavimo algoritmai. Šie algoritmai realizuojami kaip dalis projekto metu sukurto testinių duomenų generavimo plėtinio Eclipse programavimo aplinkai. Sukurtas produktas yra ištestuojamas bei numatomos jo išplėtimo ir tobulinimo perspektyvos.

Eksperimentinėje dalyje tiriamas generavimo algoritmų efektyvumo priklausomumas nuo generavimui naudojamų parametrų. Išvadose pateikiami testinių duomenų generatorių parametrų parinkimo pasiūlymai paremti eksperimentinės dalies atradimais kaip kad pastebėjimas, jog ElitismValue parametrą genetiniame generatoriuje verta naudoti su mažomis populiacijomis.

2. PROBLEMINĖS SRITIES ANALIZĖ

2.1. Programų testavimas ir jo poreikis

Šiom dienom programos kontroliuoja ar gelbsti kontroliuoti labai daug kritinių sistemų. Yra daugybė gerai žinomų atvejų kai menkos klaidos programos kode padarė daug materialinės žalos [1] ar net kainavo žmonių gyvybes [2]. Dėl šios priežasties dideliuose projektuose yra naudojami įvairūs programinės įrangos kokybės užtikrinimo metodai [3]. Šių metodų taikymas yra dažnai perdaug brangus mažesnėms, kasdieniškesnėms programoms.

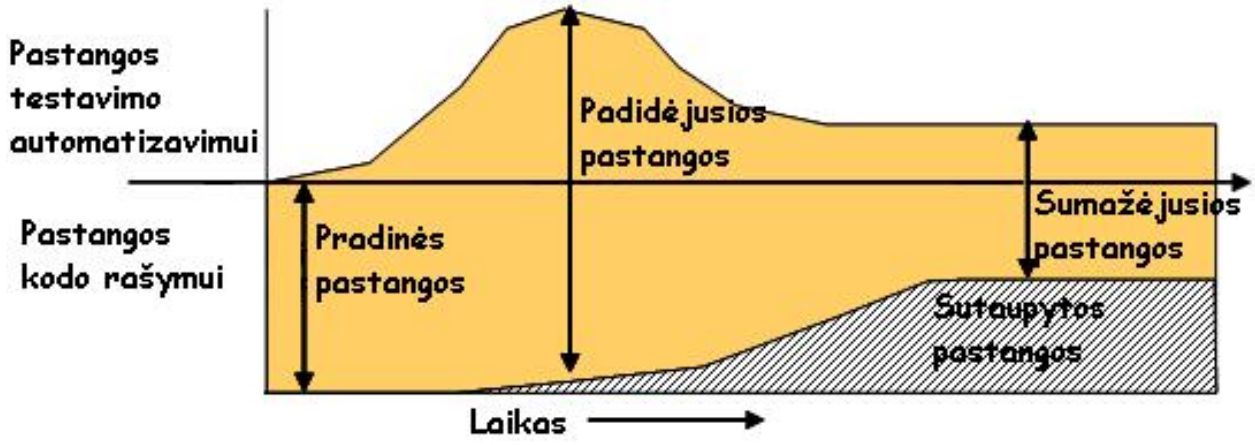
Informacinės technologijos plečiasi nelėtėjančiais tempais ir vis labiau skverbiasi į žmonių kasdienybę. Dėl to net mažiau informacines technologijas išmanantys žmonės yra priversti naudoti įvairias programas. Klaidos tokiose programose sukelia mažiau problemų, nuostolių bei pavojų, bet vis tiek turi neigiamų padarinių kaip klientų praradimas.

Dažniausiai naudojamas ir paprasčiausias programų kokybės užtikrinimo ir kėlimo būdų yra testavimas. Testavimas užtikrina, kad programa atlieka tai ką ji turi atlikti ir neatlieka to kas nebuvo numatyta [4]. Testavimas yra labai imlus laikui bei brangus procesas, kuris kuriant programinę įrangą vidutiniškai užima daugiau negu pusę viso numatyto laiko[5].

2.2. Automatinis testavimas

Testavimas yra galingas įrankis mažinti pastangų ir resursų kaip pinigai bei laikas, reikalingų projektui kiekį. Deja, atliekant testavimą rankomis tenka investuoti daug laiko, pastangų bei lėšų, kad palaikyti pastovų testavimą viso projekto metu. Šiai problemai spręsti naudojamas automatinis testavimas.

Kaip pavaizduota grafike (1 pav.)[6], testavimo automatizavimas pradžioje kainuoja daugiau pastangų, kad jį įdiegti, bet vėliau jo palaikymas sumažėja ir tampa pastovus, o produkto kūrimo kaina sumažėja. Taip yra dėl to, kad rankinis testavimas reikalauja pastovių išlaidų, o automatinis testavimas nors ir kainuoja daugiau pradžioje, laikui bėgant jo palaikymui reikia vis mažiau resursų. Automatinio testavimo palaikymo kaina yra mažesnė, nes jam yra lengviau pridėti naujus testus bei pakartotinai panaudoti senus. Automatiniai testai taip pat yra patikimesni nei rankiniai testai, nes sumažina žmogaus klaidų įterpimo tikimybę.



1 pav. Automatinio testavimo įtaka projekto kūrimo pastangoms

Automatinio testų generavimo problema susideda iš kelėtos dalių, iš kurių geriausiai žinomos yra:

- automatinis testinių duomenų generavimas [7][8] – problema sprendžianti kaip sugeneruoti naudingus testinius duomenis kiek įmanoma greičiau.

- orakulo problema [9] – problema nagrinėjanti kaip žinoti kokie turi būti teisingi programos teikiami rezultatai su specifiniais testiniais duomenim, kad automatiškai palyginti ką programa su jais gražina ir ką ji turėtų gražint.
- automatinis testinių objektų generavimas [10] – sprendžia parametrų kurie susideda iš sudėtingų objektų tokių kaip masyvai, adresai, rodyklės ar kiti objektai susidedantys iš kitų objektų generavimą.

Nors jos visos yra labai svarbios, bet dažniausiai kalbama apie ir daugiausiai tiriama yra orakulo problema. Mažiau dėmesio skiriama automatiniam testinių duomenų generavimui nes jis matomas kaip paprastesnis už kitas dalis ir dėl to ne toks įdomus.

2.3. Testinių duomenų generavimas

Testiniai duomenys generuojami baltosios dėžės principu arba juodos dėžės principu. Generavimas baltos dėžės principu dar vadinamas statiniu. Jis remiasi simboliniu kodo vykdymu, kas reiškia, jog duomenų generavimui jis nepaleidžia pačios programos [11]. Šis būdas išreiškia turimą programą matematinėmis išraiškomis sudarytomis iš įėjimo kintamųjų ir konstantų, deja, tai reikalauja daug resursų bei apriboja testuojamų programų aibę [12]. Juodos dėžės principu remtas testinių duomenų generavimas vadinamas dinaminiu. Šis duomenų generavimo variantas iteraciniu būdu generuoja testinius duomenis ir paleidinėja programą su bandomais duomenimis, kad apspręsti ar sugeneruoti duomenys yra naudingi. Dinaminiai testinių duomenų generatoriai baigia savo darbą kaip pasiekia užbrėžtą tikslą arba viršija numatytas laiko ar iteracijų ribas. Užbrėžtas tikslas dažniausiai yra nusakomas viena iš testo metrikų:

- Kodo eilučių padengimas – tai paprasčiausia, dažniausiai naudojama ir lengvai atvaizduojama testo kokybės metrika. Ši metrika parodo kurios eilutės buvo įvykdytos paleidus testą. Įvairios IDE dėl lengvo šios metrikos atvaizdavimo dažnai parodo šią metriką grafiškai kaip skirtingai nudažytas eilutes. Standartiškai žalia eilutė reiškia padengtą eilutę, o raudona – nepadengtą. Nors eilučių padengimas yra lengviausiai suprantamas, bet tai nėra geriausia metrika, nes ji nenusako padengtų kodo išsišakojimų ir galimų nepadengtų instrukcijų.
- Instrukcijų padengimas – nurodo kurios instrukcijos buvo padengtos testo. Naudojama patikslinti eilučių padengimą, kai galimi daliniai vienos eilutės padengimai.
- Šakų padengimas – nusako kurie programos loginiai išsišakojimai buvo padengti testo.

Dinaminiai duomenų generavimo metodus pagal jų veikimą galima suskirstyti į keletą grupių:

- atsitiktinius [13] – generuojami visiškai atsitiktiniai duomenys, kol randami dominantys.
- į tikslą orientuotus [12][14] – generuojami duomenys bandant padengti specifinę eilutę ar instrukciją. Sunku nuspėti kuris kelias bus padengtas.
- į kelią orientuotus [12][15] – duomenys generuojami specifiniam keliui. Lengviau nuspėti kelią, bet sunkiau randami tinkami duomenys.

Šiame darbe toliau nagrinėsime dinaminius testinių duomenų generatorius.

Dinaminių testinių duomenų generatorių pavyzdžiai yra atsitiktinis, kopimo į kalną (angl. Hill Climbing), atkaitinimo (angl. Simulated Annealing), tabu paieška, skruzdžių kolonijos optimizacija, dirbtinio intelekto sistema, genetiniai ir kiti evoliuciniai algoritmai. Projektinėje darbo dalyje bus nagrinėjami keturi iš jų: atsitiktinis (RA), kopimo į kalną (HC), atkaitinimo (SA) bei genetinis (GA). Šių algoritmų veikimas trumpai aptartas tolesniuose skyriuose, o trumpas palyginimas pateiktas lentelėje (1 lentelė).

1 lentelė Generavimo algoritmų palyginimas

Parametras	Atsitiktinis	Kopimo į kalną	Atkaitinimo	Genetinis
Algoritmo sudėtingumas	Žemas	Vidutinis	Aukštas	Aukštas
Kodo padengimo nuspėjimas	Minimalus (Įmanomas ribojant parametrų rėžius)	Minimalus (Įmanomas ribojant parametrų rėžius)	Minimalus (Įmanomas ribojant parametrų rėžius)	Minimalus (Įmanomas ribojant parametrų rėžius)
Skaičiavimų kiekis	Minimalus	Vidutinis	Aukštas	Aukštas
Atminties naudojimas	Žemas	Žemas	Žemas	Aukštas (Priklauso nuo parametrų)
Algoritmo variantų kiekis	Mažas	Mažas	Mažas	Didelis

2.3.1. Atsitiktinis generatorius

Atsitiktinis generatorius yra paprasčiausias iš nagrinėjamų generatorių. Jo veikimas paremtas atsitiktinių duomenų bandymu tol kol bus išsemti generavimo limitai arba bus pasiektas užbrėžtas tikslas. Dėl to, kad naudojami atsitiktiniai duomenys generatoriaus rezultatai yra nenusėjami ir gali smarkiai kisti tarp skirtingų generavimų. Vienintelis būdas kontroliuoti nuspėjamą padengimą yra keičiant parametrų rėžius. Atsitiktinis duomenų generavimas yra naudojamas ir kitų nagrinėjamų algoritmų veikime, taigi galima sakyti, kad šis generatorius yra pagrindas kitų algoritmu. Šis generatorius taip pat naudojamas generatorių palyginimuose kaip atskaitos taškas pažiūrėti ar bandomas generatorius yra efektyvus.

1. Atsitiktinio generatoriaus veikimas aprašomas keletu paprastų žingsnių. Nors egzistuoja alternatyvių variacijų, bet esmė lieka ta pati:
2. Atsitiktiniu būdu sugeneruojamas duomenų rinkinys.
3. Su sugeneruotu duomenų rinkiniu paleidžiama programa ir stebimos dominančios statistikos.
4. Jei atsitiktinis sprendinys pasižymi mus dominančiom savybėm jis išsaugomas.
5. Jei išsaugoti rinkiniai tenkina mūsų užsibrėžtą tikslą arba pasiektas maksimalus iteracijų kiekis baigiame darbą, kitu atveju grįžtame į pirmą žingsnį.

2.3.2. Kopimo į kalną generatorius

Šis algoritmas yra optimizacijos euristika veikianti vietinės paieškos principu. Šio algoritmo veiksmingumas labai priklauso nuo to kaip yra pasirenkamas pradinis duomenų rinkinys ir kaip pasirenkami jam gretimi. Dažniausiai pradinis rinkinys yra pasirenkamas atsitiktinai, bet žinant testuojamos programos duomenų rėžius ir specifiką galima sukurti geresnių variantų. Gretimas sprendinys dažniausiai pasirenkamas pastumiant pradinį sprendinį tam tikro dydžio žingsniu naudojantis matematinėmis operacijomis ar bitų manipuliacijom. Bendriniai algoritmo žingsniai:

1. Atsitiktiniu būdu sugeneruojamas duomenų rinkinys kuris tampa pirminiu.
2. Su sugeneruotu duomenų rinkiniu paleidžiama programa ir stebimos dominančios statistikos.
3. Sugeneruojamas gretimas pirminiam duomenų rinkinys, jei visi gretimi duomenų rinkiniai išbandyti grįžtame į pirmą žingsnį.
4. Su sugeneruotu duomenų rinkiniu paleidžiama programa ir stebimos dominančios statistikos.
5. Jei nepirminis duomenų rinkinys demonstruoja geresnes metrikas nei pirminis jis yra išsaugomas ir tampa pirminiu.
6. Jei išsaugoti rinkiniai tenkina mūsų užsibrėžtą tikslą arba pasiektas maksimalus iteracijų kiekis baigiame darbą, kitu atveju grįžtame į trečią žingsnį.

2.3.3. Atkaitinimo generatorius

Atkaitinimo algoritmas yra išplėstas kopimo į kalną algoritmas. Šis algoritmas papildytas tikimybine funkcija, kuri leidžia pradiniu duomenų rinkiniu tapti rinkiniui, kuris nėra geresnis už esamą pradinį rinkinį. Tokia funkcija priklauso nuo parametro vadinamo temperatūra (1)

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

čia c_0 – pradinio duomenų rinkinio naudingumo įvertis, c – gretimo sprendinio duomenų rinkinio naudingumo įvertis, t – temperatūra. Algoritmo eigoje temperatūros vertė pastoviai mažinama. Toks temperatūros mažėjimas reiškia, jog tikimybė, kad prastesnis sprendinys taps pradiniu pastoviai mažėja. Toks algoritmo veikimas turėtų padėti išvengti lokalaus maksimumo.

Bendras algoritmas:

1. Atsitiktiniu būdu sugeneruojamas duomenų rinkinys, kuris tampa pirminiu.
2. Su sugeneruotu duomenų rinkiniu paleidžiama programa ir stebimos dominančios statistikos.
3. Sugeneruojamas gretimas pirminiam duomenų rinkinys, jei visi gretimi duomenų rinkiniai išbandyti grįžtame į pirmą žingsnį.
4. Su sugeneruotu duomenų rinkiniu paleidžiama programa ir stebimos dominančios statistikos.
5. Jei nepirminis duomenų rinkinys demonstruoja geresnes metrikas nei pirminis jis yra išsaugomas ir tampa pirminiu.
6. Jei rinkinys netapo pirminiu yra sugeneruojamas atsitiktinis skaičius $x \in [0, 1]$. Jei tenkinama $x < P(c0, c, t)$ duomenų rinkinys išsaugomas ir tampa pirminiu.
7. Jei išsaugoti rinkiniai tenkina mūsų užsibrėžtą tikslą arba pasiektas maksimalus iteracijų kiekis baigiame darbą, kitu atveju sumažiname temperatūrą ir grįžtame į trečią žingsnį.

2.3.4. Genetinis generatorius

Genetinis algoritmas tai euristinis paieškos metodas bandantis mėgdžioti natūralią atranką. Šis algoritmas priklauso evoliucinių algoritmų grupei kurie naudoja metodus įkvėptus gamtoje stebimų veiksniu kaip natūrali atranka, paveldėjimas, mutacijos, kryžminimasis. Genetinio algoritmo idėja yra iš turimos grupės, naudojantis tinkamumo funkcija, atrinkti naudingus duomenų rinkinius ir juos sukryžminti naudojantis kryžminimo funkcija taip sukuriant naują duomenų rinkinių populiaciją su kuria šis procesas kartojamas kol pasiekiamas užsibrėžtas tikslas arba pasiekiamas maksimalus iteracijų kiekis. Tinkamumo funkcija priklauso nuo užduoties, kurios sprendimui naudojamas algoritmas. Testinių duomenų generavimo atveju tai dažniausiai yra eilučių, instrukcijų ar šakų padengimas. Kryžminimo funkcijos dažniausiai apjungia duomenų rinkinius bitų lygiu arba naudojantis tam tikromis matematinėmis operacijomis. Bendras algoritmas aprašomas taip:

1. Atsitiktiniu būdu sugeneruojama pasirinkto dydžio n grupė duomenų rinkinių.
2. Su duomenų rinkiniais paleidžiama programa ir pagal stebimas vykdymo metrikas apskaičiuojami duomenų rinkinių tinkamumo įverčiai.
3. Jei turima duomenų rinkinių grupe tenkina užsibrėžtą tikslą arba pasiektas maksimalus iteracijų kiekis išsaugome turimus rinkinius ir baigiame darbą.
4. Geriausius tinkamumo įverčius parodę duomenų rinkiniai kryžminami tarpusavyje naudojantis kryžminimo funkcija. Taip sukuriama vaikinė duomenų grupė. Grįžtama į antrą žingsnį.

2.4. Egzistuojantys sprendimai

Rinkoje egzistuoja daug automatinių tetinių duomenų generatorių. Dauguma jų yra uždaro kodo ir nenori atskleisti komercinių paslapčių kaip kad specifiniai naudojamo algoritmo patobulinimai ar netgi pats naudojamas algoritmas. Kiti įrankiai yra dalis kokio nors kito projekto. Taip pat daugelis programoms, o ne duomenų bazėms, skirtų testų naudoja įvairius baltos dėžės metodus, kurie šiame tyrime mūsų nedomina.

Tyrimui buvo nuspręsta pasinagrinėti populiariausius atviro kodo, JAVA [16] programavimo kalbai skirtus testinių duomenų generatorius veikiančius juodos dėžės principu. Deja, pastebėjus tokių įrankių trūkumą buvo nuspręsta pasigilinti į keletą skirtingų įrankių, kurie yra geriau žinomi. Čia bus paminėti geriausią įspūdy palikę įrankiai ir trumpai apibūdinti palyginimo lentelėje (2 lentelė).

Jtest [17] – komercinis uždaro kodo kompanijos Parasoft [18] produktas. Rinkoje pasirodė 1997 metais. Skirtas JAVA kalba parašytoms programoms. Hibridinis įrankis, kurio skirtingos funkcijos veikia skirtingais principais – kažikurios baltos dėžės, kažikurios juodos dėžės principu. Be testinių duomenų generavimo atlieka kodo statinę analizę, programos veikimo analizę gali aptikti programos saugumo spragas.

CodePro AnalytiX [19] – kadaise komercinis įrankis, perimtas Google ir 2010 metais išleistas su Apache License 2.0 [20]. JAVA kalbai skirtas įrankis gebantis generuoti JUnit [21] testus, pateikti projekto metrikas bei atlikti statinę kodo analizę. Integruojasi į Eclipse [22], Rational Developer [23] bei IBM WebSphere Studio [24] programavimo aplinkas.

jPET [25] – Atviro kodo Eclipse plėtinys skirtas generuoti testinius duomenis JAVA programoms. Neatnaujintas nuo 2011 metų. Naudojama GPL Version 3 [23][26] licencija.

2 lentelė Testinių duomenų duomenų generavimo įrankiai

Parametras	**Jtest**	**CodePro AnalytiX**	**jPET**
Licencija	Komercinis	Apache License 2.0	GPL Version 3
Operacinė sistema	Windows/Linux/Mac /Solaris	Windows/Linux/Mac	Windows/Linux/Mac
Funkcionalumas	Didelis	Didelis	Mažas
Naudojama IDE	Eclipse / IBM RAD	Eclipse / Rational Developer / IBM WebSphere Studio	Eclipse
Kaina	Mokama	Nemokamas	Nemokamas
Palaikomos kalbos	Java	Java	Java

2.5. Apibendrinimas

1. Aptarta testavimo svarba šių dienų programų kokybės užtikrinime.
2. Trumpai aptartas automatinis testinių duomenų generavimas, jo nauda, į kokias grupes jis skirstomas bei kokiomis metrikomis jis nusakomas.
3. Pasirinkti ir aprašyti keturi automatinių testinių duomenų generavimo metodai, kurie bus naudojami projektui realizuoti: atsitiktinis(RA), kopimo į kalną(HC), atkaitinimo(SA) bei genetinis(GA).
4. Iš aptartų algoritmų atsitiktinis yra matomas kaip prasčiausias ir dėl to naudojamas kaip etalonas palyginti kitų algoritmų efektyvumą.
5. Aptarti egzistuojantys testinių duomenų generavimo įrankiai.
6. Pastebėtas trūkumas nemokamų, lengvai suprantamų ir pastoviai atnaujinamų bei plečiamų įrankių.

3. PROJEKTINĖ DALIS

Šioje dalyje bus aprašyti studijų metu, kartu su kolega Gintautu Motiejūnu, sukurtos programinės įrangos skirtos generuoti testinius duomenų rinkinius, Testinių Duomenų Multi-Generatorius (toliau TDMG), esminiai architektūriniai aspektai.

3.1. Atsakomybių pasiskirstymas projekte

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

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.

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.

3.2. Sistemos paskirtis

Sistema yra skirta palengvinti testinių duomenų generavimo procesą jį automatizuojant bei palengvinti automatinių testinių duomenų generavimo algoritmų tyrimą suteikiant galimybę sistemą lengvai išplėsti, rinkti duomenų generavimų statistikas bei suteikti galimybę palyginti generatorių pasiektus rezultatus. Visa tai turi būti pasiekta sukuriant patrauklų, lengvai suprantama ir lengvai panaudojamą įrankį skirta plačiai naudojamai Eclipse IDE.

3.3. Sistemos funkcijos ir reikalavimai

Atliekant reikalavimų surinkimą iš užsakovo buvo identifikuoti įrankio funkciniai reikalavimai bei nefunkciniai reikalavimai.

3.3.1. Įrankio 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 kaip: generavimo laikas, kodo padengimas, sugeneruotų testinių atvejų skaičius.
- Sistema turi kaupti bendrą jų statistiką.
- Sistema privalo palaikyti šiuos automatinius testu generavimo metodus:
    - Atsitiktinį (Random)
    - Kopimo į kalną (Hill Climbing)
    - Atkaitinimo (Simulated Annealing)
    - Genetinį (Genetic)
- Sistema turi leisti pasirinkti kelis duomenų generavimo metodus, su kuriais vieno vykdymo metu bus generuojami testiniai 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 metodų efektyvumo įverčiai.

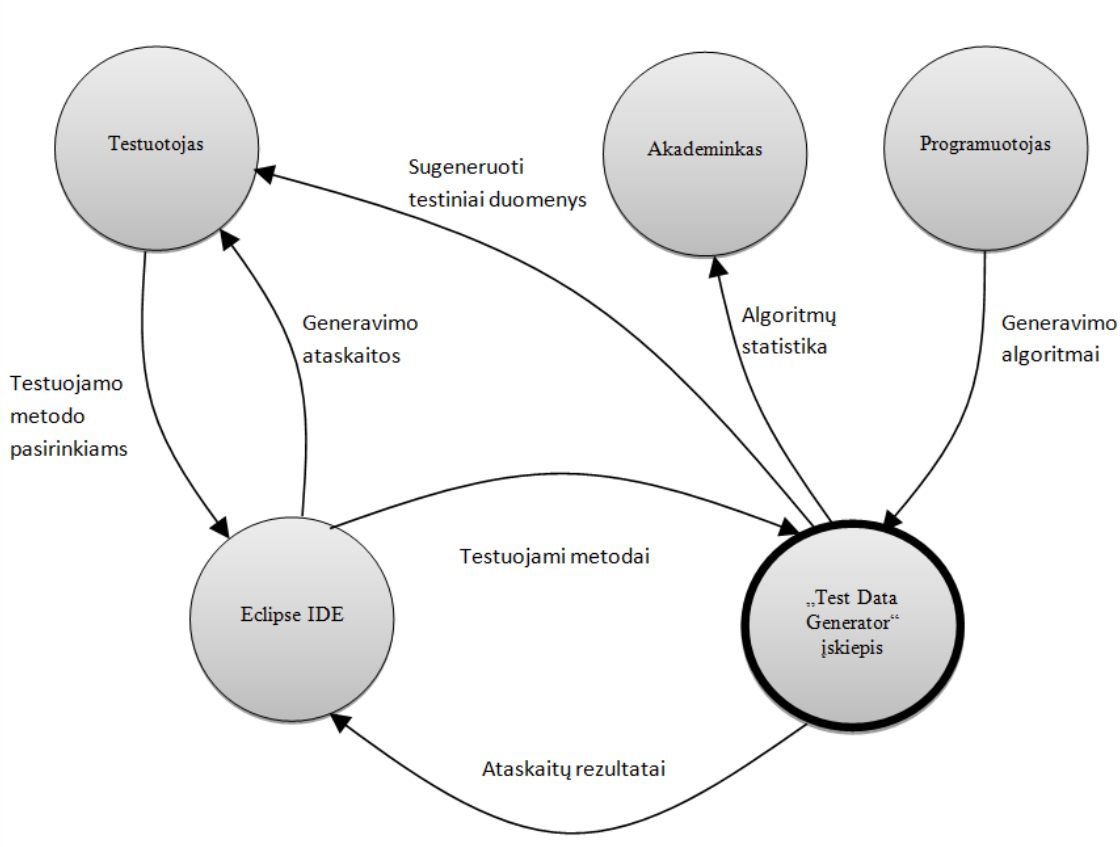
- Sistema turi leisti peržiūrėti sugeneruotas ataskaitas standartiniame Eclipse lange tekstiniu formatu.
- Sistema turi leisti išsaugoti ataskaitas HTML bei TXT formate.
- Sistema turi leisti išsaugoti ataskaitas paprastu tekstiniu formatu.
- Sistema turi būti lengvai išplečiama naujais testinių duomenų generavimo metodų moduliais.
- Sistema turi leisti pašalinti testinių duomenų generavimo modulius.
- Sistema turi leisti keisti atskirų generavimo metodų parametrus
- Sistema turi kaupti bendrą naudojimo statistiką nuo pat programos naudojimo pradžios. Joje išskiriami įvairių charakteristikų atžvilgiu geriausi bei blogiausi generavimo metodai.
- Sistema turi leisti ištrinti naudojimo statistiką.
- 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 statistiką juos ištrinti.

3.3.2. Įrankio 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 netrukdyti kitoms sistemoms, efektyviai naudoti resursus.
- Sistema turi atlikti užduotis per kaip galima greitesnį laiką.
- Sistema turi rodyti, kad užduotis vykdoma.
- Sistema turi leisti ją papildyti naujais testavimo komponentais.
- Paprastas produkto įdiegimas.
- Naujų Eclipse 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.
- Programa platinama kaip nemokama pagal GNU GPL v3 licencija.

3.3.3. Veiklos kontekstas

Sistemos veiklos kontekstas pateiktas diagramoje (2 pav.). Čia matyti, kad testuotojui pasirinkus testuotiną metodą Eclipse jį perduoda plėtiniui, kuris gražina testuotojui sugeneruotus duomenis testuotojui ir ataskaitas atgal į Eclipse. Taip pat matyti, kad programuotojas gali išplėsti sistemą naujais generavimo algoritmais, ir jog sistema pateikia įvairias ataskaitas ir statistikas tuo besidominčiam akademikui. Čia vartotojai testuotojas, akademikas ir programuotojas buvo išskirti aiškumui, jie gali būti tas pats asmuo.



2 pav. Sistemos veiklos kontekstas

3.3.4. Architektūrinius sprendimus ribojantys reikalavimai

Iš surinktų reikalavimų identifikuoti reikalavimai ir apribojimai labiausiai įtakojantys tolesnius architektūrinius sprendimus:

- Sistema turi būti Eclipse IDE plėtinys.
- Sistemos sąsaja turi integruotis ir neišsiskirti iš Eclipse IDE.
- Sistema turi būti lengvai išplečiama naujais testinių duomenų generatoriais.

Iš šių apribojimų taip pat kyla išvestiniai apribojimai:

- Sistema, kaip ir Eclipse IDE kurioje ji veiks, turi be konfliktų veikti visose Eclipse palaikomose operacinėse sistemose.

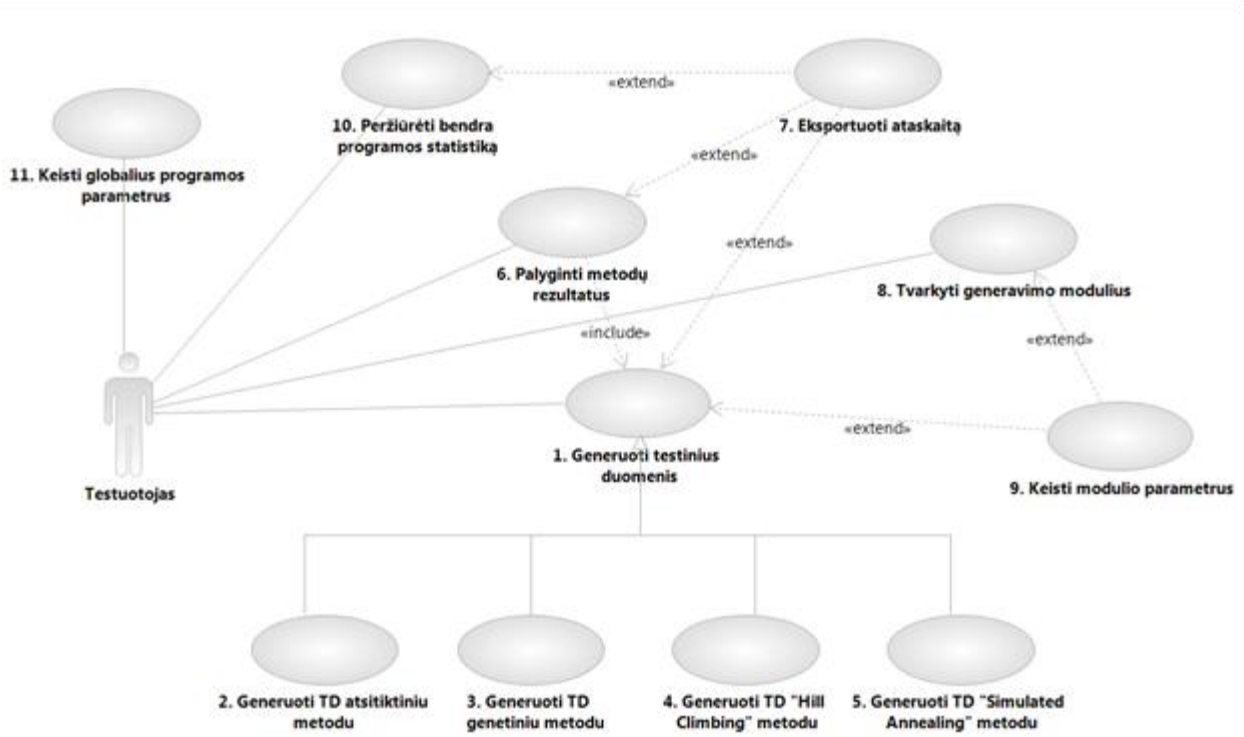
3.4. Architektūra

Sukurta sistema aprašyta pagrindiniais vaizdas kurie pateikti ir paaiškinti šiame skyriuje. Šie vaizdai pateikti unifikuota modeliavimo kalba (UML [27]) naudojantis Rational Rose įrankiu. Toliau skyriuje pateikiama:

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

3.4.1. Panaudos atvejų vaizdas

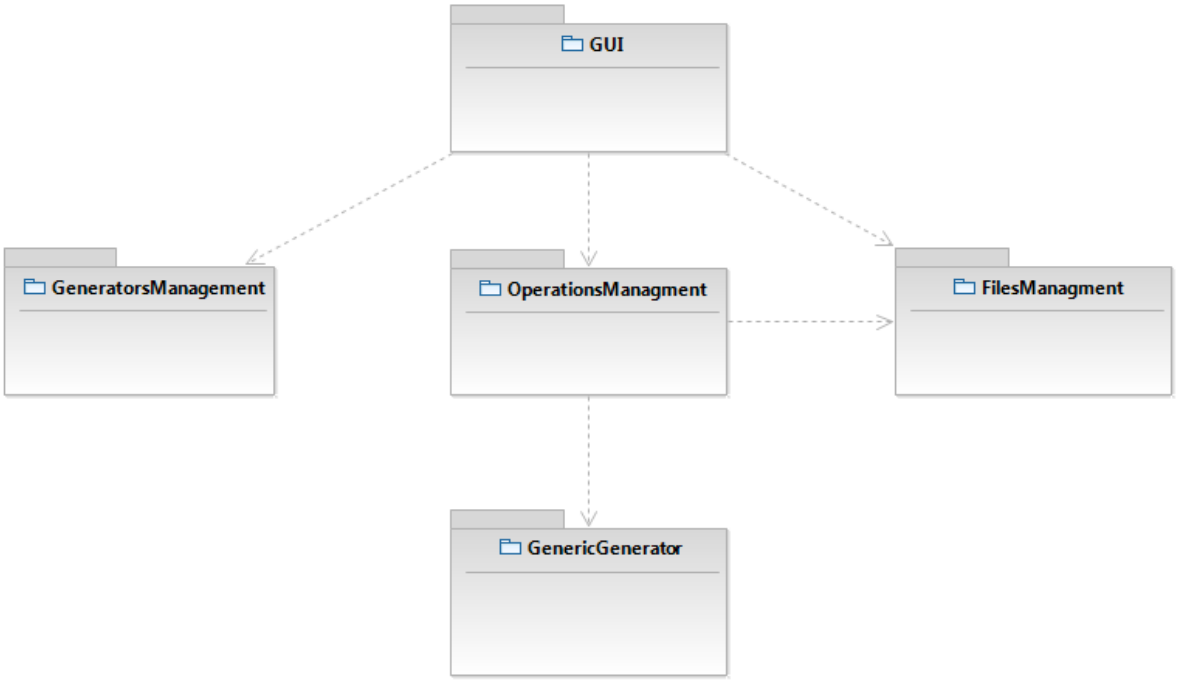
Sistemos panaudos atvejų diagrama pateikta žemiau (3 pav.).



3 pav. Sistemos panaudos atejų diagrama

3.4.2. Sistemos statinis vaizdas

Sistema susideda iš penkių paketų pavaizduotų žemiau (4 pav.).

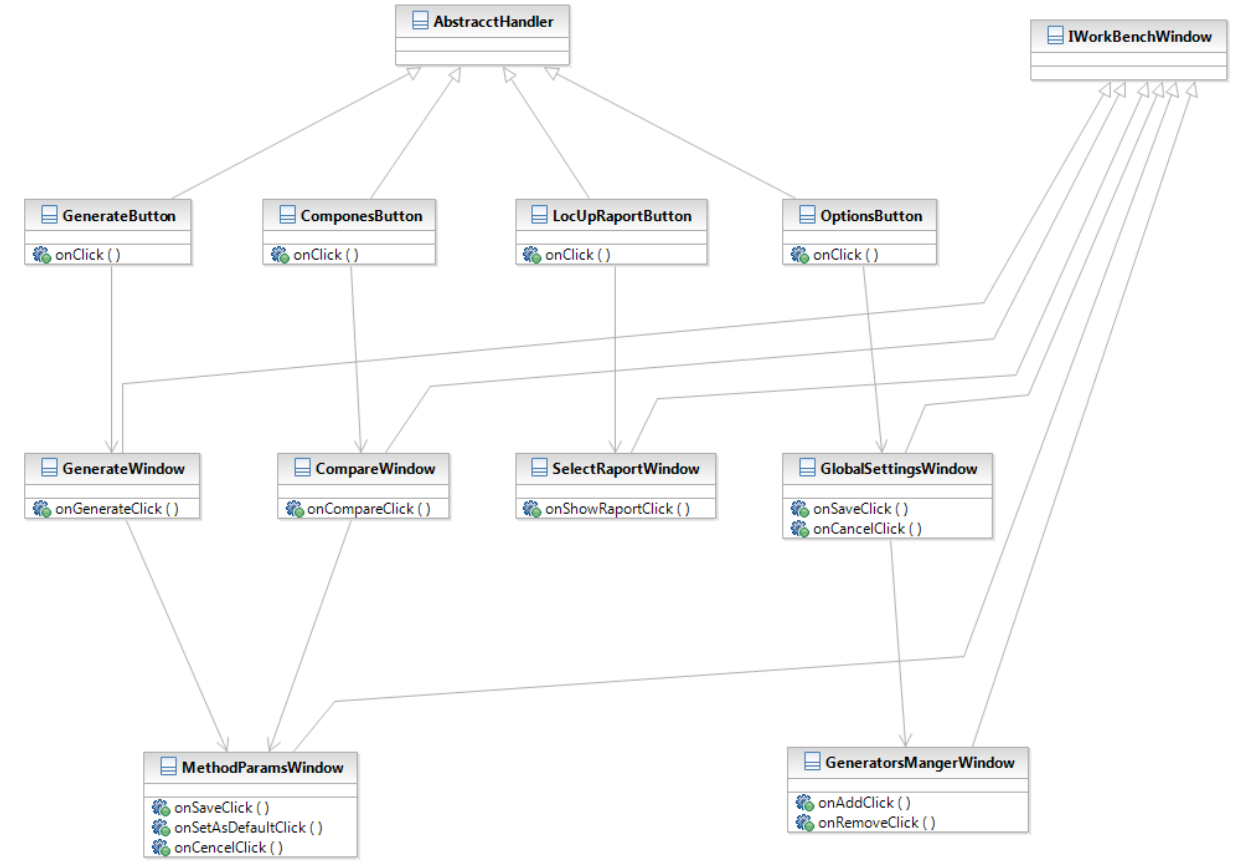


4 pav. Sistemos paketai

Toliau detaliau aprašomi sistemos paketai.

Paketas „GUI“

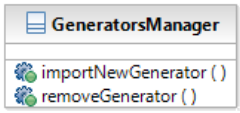
Šis paketas apima klases susijusias su sistemos vartotojo sąsaja. Vartotojo sąsajos rydymui sistemoje naudojami Eclipse komponentai. Detali klasių diagrama pateikta žemiau (5 pav.).



5 pav. Paketo „GUI klasių“ diagrama

Paketas „GeneratorsManagement“

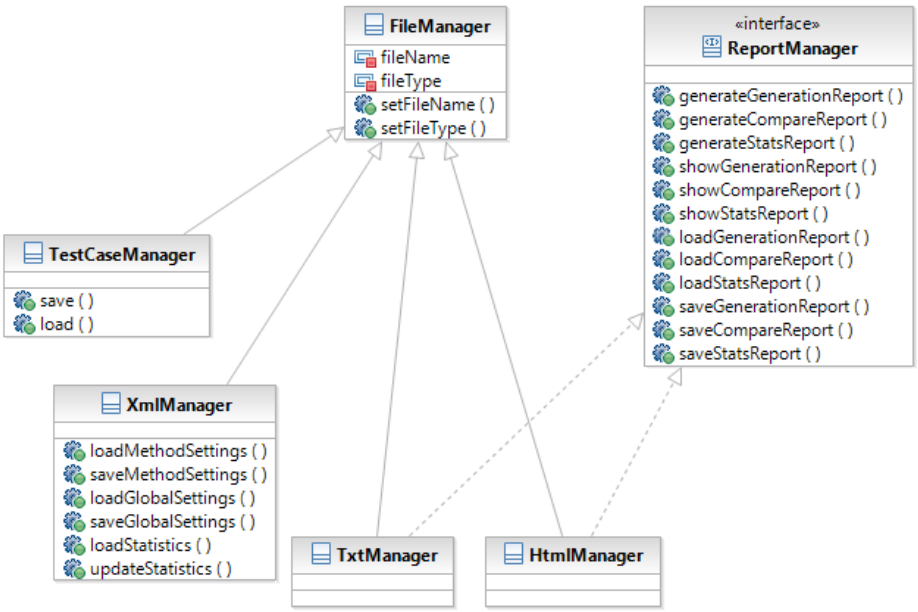
Šis paketas atsako už testinių duomenų generatorių modulių pridėjimą ir pašalinimą iš sistemos. Klasių diagrama pateikiama žemiau (6 pav.).



6 pav. Paketo „GeneratorsManagement“ klasių diagrama

Paketas „FilesManagement“

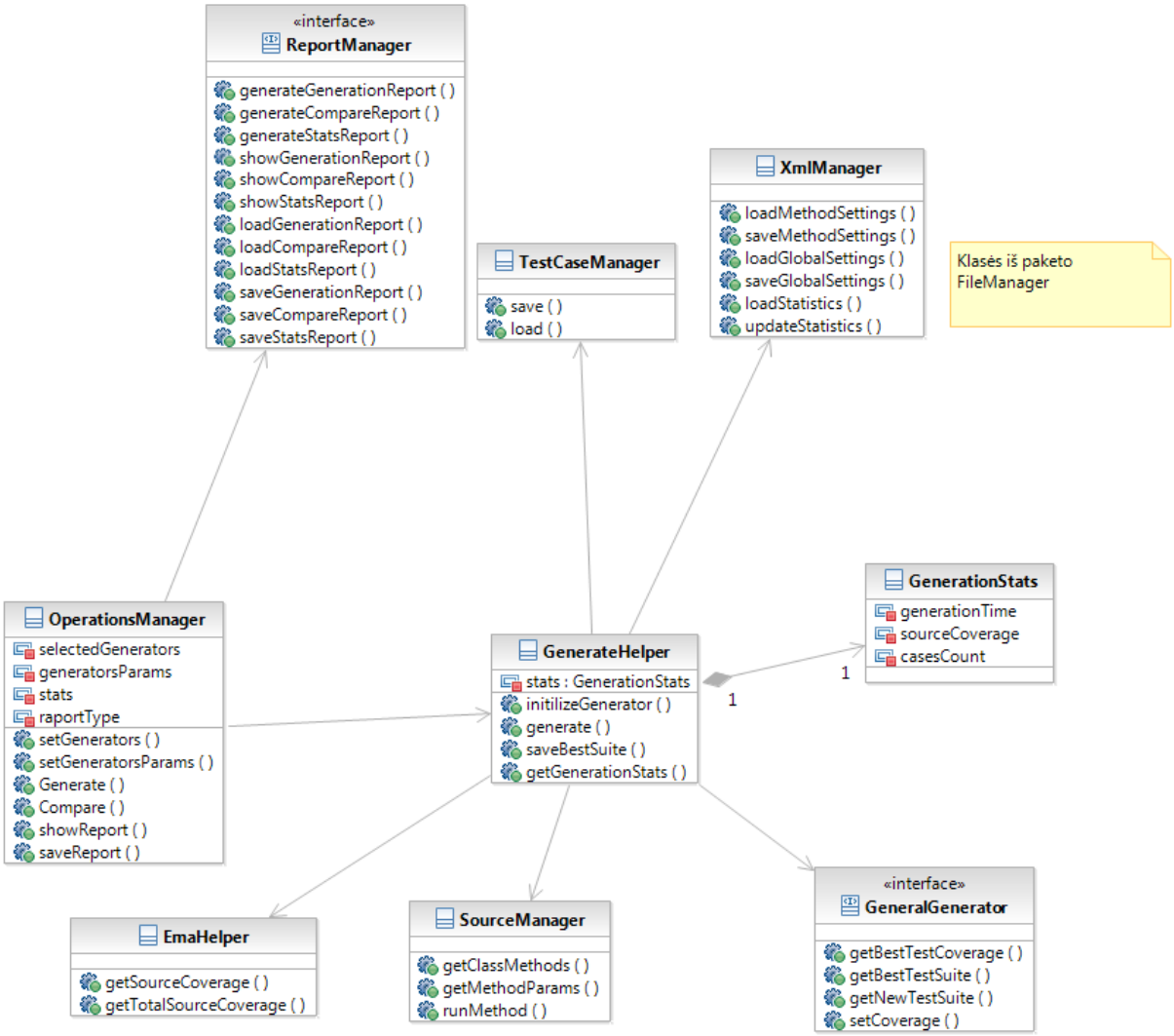
Paketas susideda iš klasių skirtų darbui su failais. Tai klasės atsakingos už ataskaitų bei įvairių nustatymų saugojimą failuose. Klasių diagrama pateikiama žemiau (7 pav.).



7 pav. Paketo „FilesManagement“ klasių diagrama

Paketas „OperationsManagement“

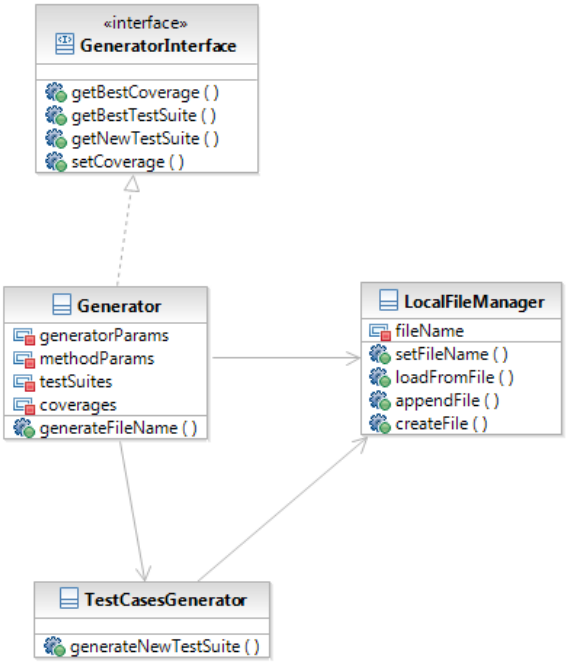
Klasės šiame pakete nusako pagrindinį kurtos sistemos funkcionalumą susijusį su testinių duomenų generavimu. Vartotojo veiksmai per „GUI“ paketą gali iškviesti tam tikras klasės „OperationsManager“ funkcijas, kurios naudodamos tiek vidines paketo klases, tiek ir klases iš paketų „FileManagement“ ir „GeneralGenerator“ įgyvendina norimą funkcionalumą. Kadangi šis paketas yra kaip tarpininkas tarp kitų paketų, žemiau pateiktoje diagramoje (8 pav.), yra pridėta keletas klasių iš paketo „FileManagement“. Taip pat pažymėtina, jog sąsajos klasė „GeneralGenerator“ įgyvendina sistemos išplečiamumą kitais testinių duomenų generatoriais įgyvendinančiais šią sąsajos klasę.



8 pav. Paketo „OperationsManagement“ klasių diagrama

Paketas „GenericGenerator“

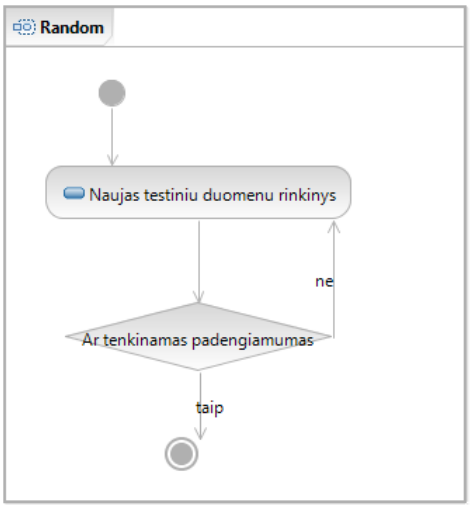
Tai nėra realus paketas, o tik jo atitikmuo. Kiekvienas testinių duomenų generavimo algoritmas bus realizuojamas kaip atskiras modulis, sudarytas iš keleto klasių, kuriose aprašyta generavimo logika. Toks modulis įgyvendins „OperationsManagement“ paketo „GenericGenerator“ sąsają ir jį bus galima paprastai pridėti arba atimti neperkompiliuojant visos programos. Diagrama pateikta žemiau (9 pav.).



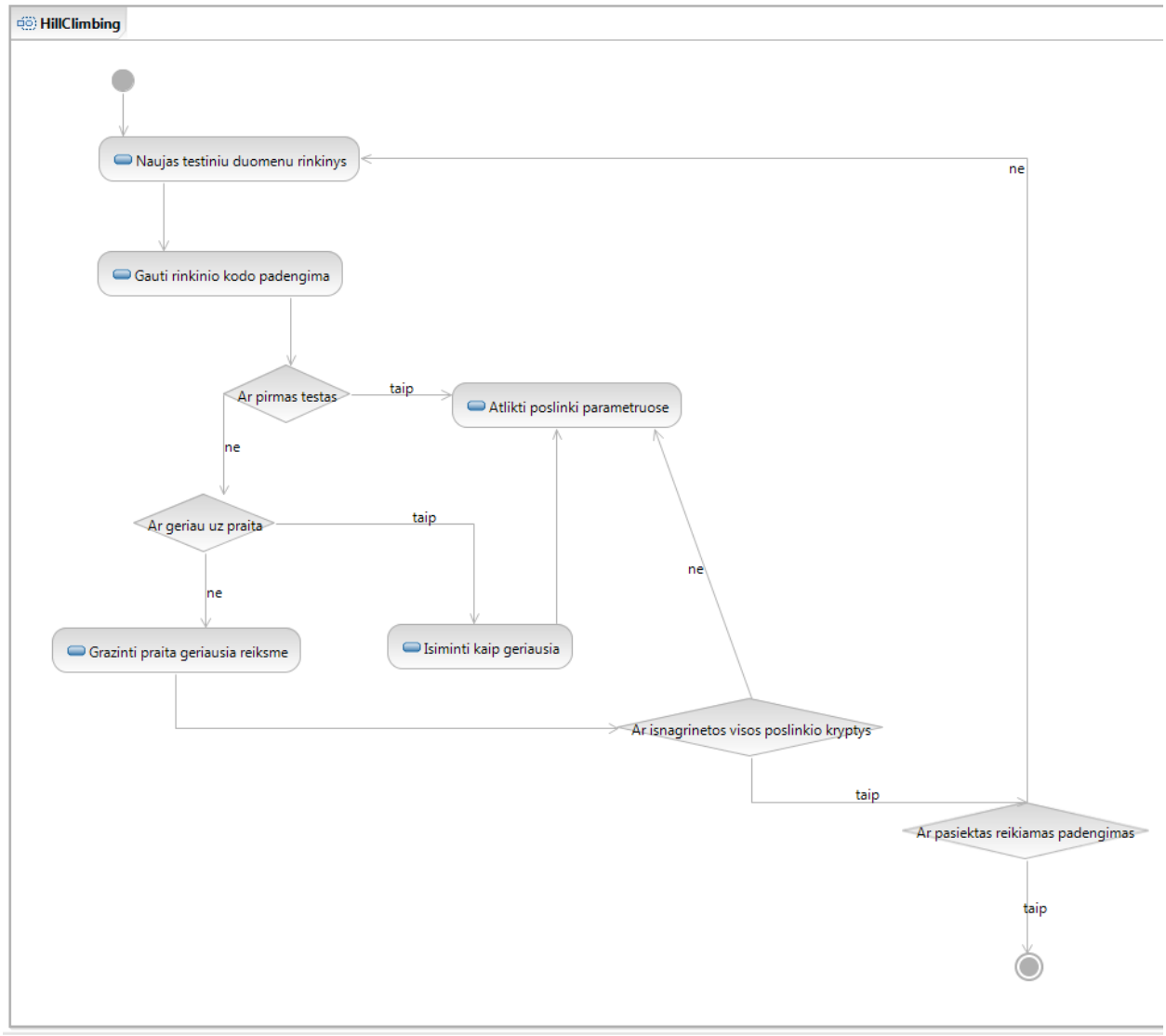
9 pav. Paketo „GenericGenerator“ klasių diagrama

3.4.3. Sistemos dinaminis vaizdas

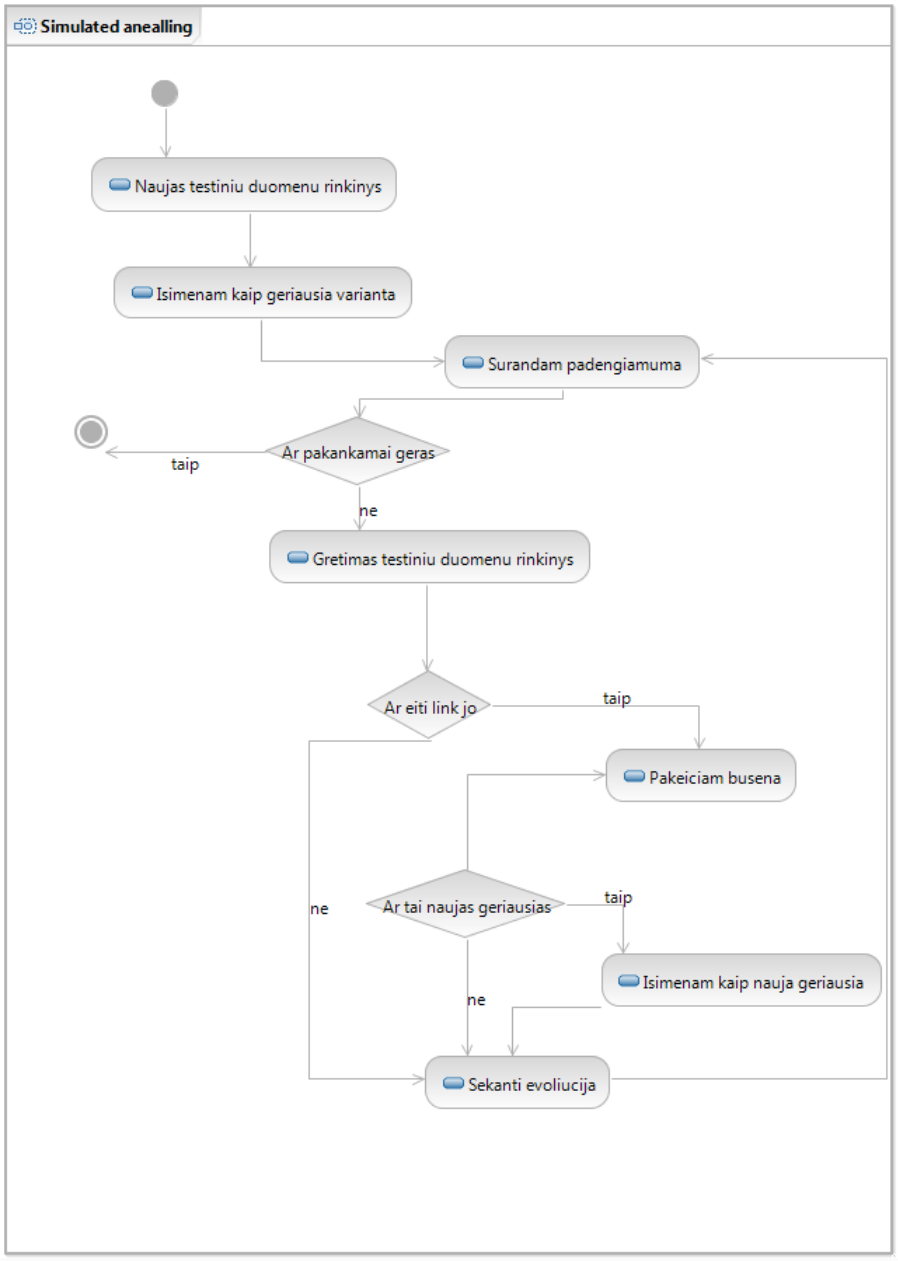
Žemiau pateikiamos atsitiktinio (10 pav.), kopimo į kalną (11 pav.), atkaitinimo (12 pav.) bei genetinio (13 pav.) testinių duomenų generavimo metodų veiklos diagramos. Šių algoritmų veikimas išsamiai paaiškintas skyriuje 2.3 Testinių duomenų generavimas.



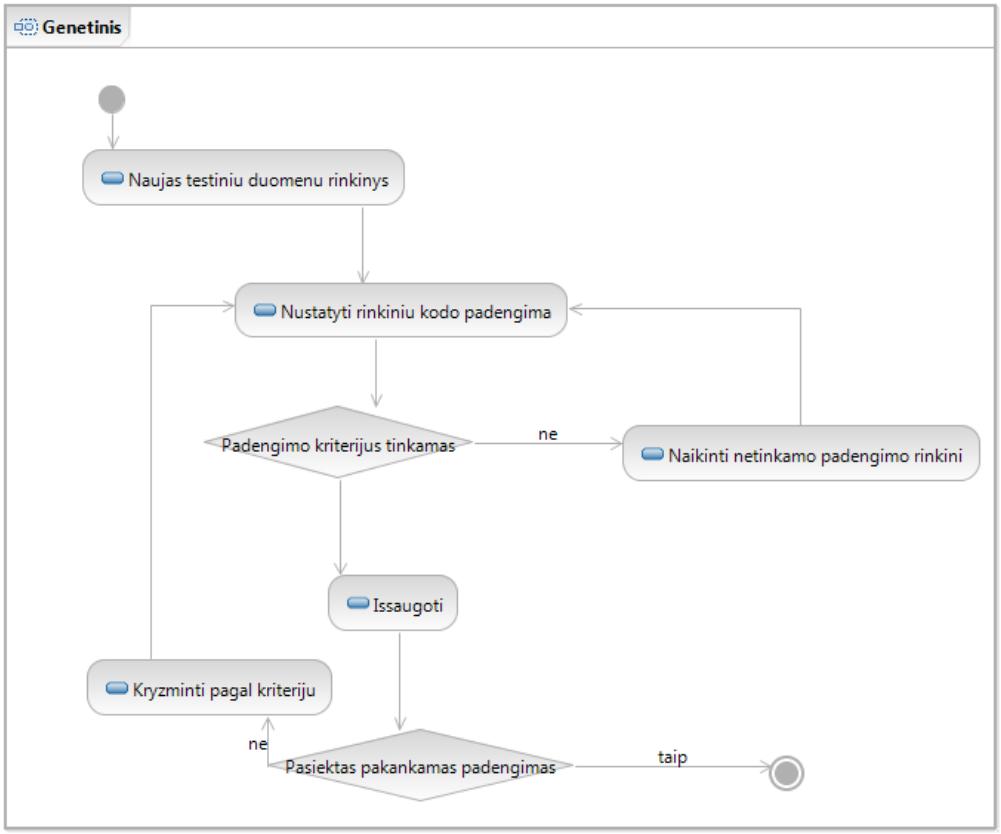
10 pav. Atsitiktinio duomenų generavimo algoritmo veiklos diagrama



11 pav. Kopimo į kalną duomenų generavimo algoritmo veiklos diagrama



12 pav. Atkaitinimo duomenų generavimo algoritmo veiklos diagrama



13 pav. Genetinio duomenų generavimo algoritmo veiklos diagrama

3.4.4. Vartotojo sąsaja

Vartotojo sąsaja yra labai svarbi vartotojui. Nesuprantama, neintuityvi sąsaja išgąsdins vartotojus ir neleis įrankiui populiarėti. Kadangi sistema skirta testuoti programinį kodą Eclipse aplinkoje buvo nuspręsta daryti ją neišsiskiriančią iš Eclipse. Tam pasiekti buvo naudojami Eclipse aplinkos komponentai bei remiamasi Eclipse vartotojo sąsajos nurodymais [28]. Taipogi žinant jog testavimo darbus atlieka specialistai gerai išmanantys savo darbą, buvo nuspręsta nedėti išsamių paaiškinimų kas vyksta kiekviename programos žingsnyje į pačią programą, bet visa tai sudėti į vartotojo gidą.

3.4.5. Kokybė

- Apibrėžta sąsaja testų generavimo realizavimui leidžia ateityje nesunkiai prijungti ir daugiau testų generavimo algoritmų.
- Generatorių bei sistemos nustatymų saugojimas failuose leidžia juos lengvai perkelti bei keisti išoriškai.
- Sukurtas įrankis yra Eclipse plėtinys naudojantis Eclipse aplinkos komponentus, kas padės vartotojams įsisavinti įrankį.
- Sukurtas įrankis yra plėtinys kas reiškia, jog vartotojams nereiks naudoti jo kaip atskiros programos.
- Sukurtas įrankis gana paprastai plečiamas kas leis ateityje pridėti naujų funkcijų kaip Orakulas.

3.5. Sistemos testavimas

Toliau aprašoma sukurtos sistemos testavimo specifikacija, procedūros bei pasiekti rezultatai. Testavimo metu buvo atlikta:

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

3.5.1. Vienetų testavimas

Atskiriems sistemos metodams ir klasėms testuoti buvo naudojami vienetų testai. Vienetų testai nusako įėjimo kintamuosius į metodą ir lygina metodo rezultatą su rezultatu, kurio tikisi sudarytas vieneto testas. Sukurti vienetų testai buvo aprašyti JUnit karkase ir automatizuoti. Projekto reikalavimuose buvo reikalaujama sukurti 46 vienetų testus iš kurių pusė buvo sukurti juodos dėžės principu, o kiti baltos dėžės principu. Sukurtų testų pavyzdys pateikiamas žemiau (3 lentelė). Šis testas skirtas paketo OperationsManagement klasės EmaHelper metodui getSourceCoverage testuoti. Rastos klaidos buvo ištaisomos ir testavimas tęsiamas. Po bet kokių pakeitimų vienetuose automatinis testavimas buvo kartojamas.

3 lentelė Vinetų testai metodui „getSourceCoverage“

Nr	Įėjimas	Laukiamas rezultatas	Klaida
1.5.2.1.1	method: test1 testCasePath: E:/TDMG/temp/temp1tc.tc temp1tc.tc testinis atvejis padengia 50 % test1 metodo kodo	Gražinama reikšmė 50	-
1.5.2.1.2	method: test1 testCasePath: E:/TDMG/temp/temp2tc.tc temp1tc.tc testinis atvejis padengia 100 % test1 metodo kodo	Gražinama reikšmė 100	-
1.5.2.1.3	method: test1 testCasePath: E:/TDMG/temp/temp3tc.tc temp1tc.tc testinis atvejis padengia 0 % test1 metodo kodo	Gražinama reikšmė 0	-

3.5.2. Integracinis testavimas

Integracinis testavimas buvo atliekamas apjungiant vienetų testavimo metu sėkmingai ištestuotus elementus ir testuojant jų tarpusavio sąveiką. Tam buvo sukurta 18 JUnit testų kuriais buvo bandoma rasti klaidas. Rastos klaidos buvo ištaisytos ir testavimas kartojamas nuo vienetų testų, kad įsitikinti, jog nebuvo įvelta naujų klaidų.

3.5.3. Vartotojo sąsajos testavimas

Kadangi vartotojo sąsaja yra aukščiausias ir paskutinis integracinio testavimo lygmuo, ji buvo testuojama paskutinė. Kadangi vartotojo sąsaja nėra sudėtinga jai testai nebuvo rašomi. Sąsaja buvo testuojama rankiniu būdu bandant visas esamas funkcijas. Kad testuotojui būtų lengviau atlikti rankinį testavimą ir sumažėtų žmogaus klaidos tikimybė, testų scenarijai buvo aprašyti scenarijais. Scenarijų pavyzdys pateiktas žemiau (4 lentelė).

4 lentelė Testavimo scenarijai langui „Generators“

Nr	Atliekami veiksmai	Laukiamas rezultatas
2.5.1	Pasirenkamas generatorius „TestGenerator“; Spaudžiamas mygtukas „Remove“; Spaudžiamas patvirtinimo mygtukas „No“.	„Compare“ lango sąraše generatoriaus „TestGenerator“ vis dar yra.
2.5.2	Pasirenkamas generatorius „TestGenerator“; Spaudžiamas mygtukas „Remove“; Spaudžiamas patvirtinimo mygtukas „Yes“.	„Compare“ lango sąraše generatoriaus „TestGenerator“ nebėra.
2.5.3	Nurodomas kelias iki failo „TestGenerator.jar“; Spaudžiamas mygtukas „Add“.	„Compare“ lango sąraše atsirado generatorius „TestGenerator“.
2.5.4	Nurodomas kelias iki failo „Trollollo.jar“; Spaudžiamas mygtukas „Add“.	Rodomas klaidos pranešimas apie netinkama generatoriaus modulį. „Compare“ lango sąraše generatoriaus „Trollollo“ nėra.
2.5.5	Pasirenkamas generatorius „Random“; Spaudžiamas mygtukas „Settings“.	Atidaromas „Random“ generatoriaus individualių parametrų langas „Generator Settings“.
2.5.6	Spaudžiamas mygtukas „Cancel“.	Uždaromas langas „Generators“.

3.5.4. Priėmimo testavimas

Priėmimo testavimas buvo atliekamas pilnai pabaigus ir ištestavus sistemą. Priėmimo testavimo metu programa buvo pateikta užsakovui, pademonstruotas jos veikimas ir leista užsakovui tikrinti programos veikimą visais užsakovą dominančiais būdais. Užsakovui išreiškus nepasitenkinimą programos veikimu buvo planuota imtis tokių veiksmų:

- Jei užsakovo nepasitenkinimą sukėlė neteisingai realizuotas ar suprastas funkcionalumas dėl kurio buvo tariamasi reikalavimų surinkimo metu – tai fiksuojama kaip klaida ir kuo greičiau ištaisoma, kad atitiktų užsakovo reikalavimus.
- Jei užsakovo nepasitenkinimą sukėlė programos veikimas kuris prieštarauja užsakovo išreikštiems norams reikalavimų surinkimo metu – tai neregistruojama kaip klaida, bet registruojama kaip naujas ar skirtingas reikalaujamas funkcionalumas sekančiai programos versijai.

Priėmimo testavimo metu užsakovas neišreiškė nepasitenkinimų.

3.6. Atlikti patobulinimai

Programos demonstravimo universitete metu buvo pasiūlyta pakeisti visus generavimo algoritmus taip, kad jie leistų pasirinkti parametrų generavimo rėžius kiekvienam testuojamo metodo parametrui atskirai, vietoje vieno rėžių rinkinio taikomo visiems to paties tipo parametrams. Šis pakeitimas buvo sėkmingai įgyvendintas naujausioje plėtinio versijoje.

3.7. Planuojami patobulinimai

Artimiausiu metu planuojama išplėsti sukurtos sistemos funkcionalumą šiais patobulinimais:

- Nauji testinių duomenų generavimo algoritmai.
- Integruotas Orakulas [29] su galimybe pridėti naujų Orakulų bei pasirinkti norimą naudoti generavimo metu.
- Adaptuoti plėtinį veikimui su InteliJ IDE bei NetBeans IDE.

- Suteikti galimybę, tiems kurie nenaudoja IDE, plėtinį naudoti be jokios IDE per komandinę eilutę.

3.8. Apibendrinimas

1. Sėkmingai surinkti sistemos reikalavimai iš užsakovo.
2. Suprojektuota, realizuota, ir ištestuota testinių duomenų generavimo sistema visiškai patenkinanti surinktus užsakovo reikalavimus.
3. Sukurta sistema išplėsta funkcionalumu pasiūlytu sistemos pristatymo metu.
4. Numatytas tolesnis sukurtos sistemos vystymas ir funkcionalumo plėtimas.
5. Kuriant plėtinį Eclipse programavimo aplinkai buvo susidurta su daug problemų susijusių su tam skirtų įrankių neišbaigtumu, todėl gali teigti, kad Eclipse IDE plėtiniams skirti įrankiai nėra tobuli.

4. EKSPERIMENTINIS TYRIMAS

Eksperimento metu bus bandoma ištirti pasirinktų testinių duomenų generavimo algoritmų efektyvumo priklausomumą nuo naudojamų generatoriaus parametrų. Eksperimentui buvo pasirinkti genetinis (GA) ir atkaitinimo (SA) testinių duomenų generatoriai. Tyrimo metu bus automatizuotai bandomi tiriamų generatorių parametrai ir stebimi jų teikiami rezultatai. Tyrimo rezultate tikimasi pateikti rekomendacijas generatorių parametrų parinkimui.

4.1. Eksperimento eiga

Numatyta eksperimentinio tyrimo eiga:

1. Automatizuotam tyrimui modifikuojamas TDMG Eclipse IDE plėtinys. Naujas funkcionalumas leidžia plėtiniui savaime keisti generatoriaus parametrus ir kartoti duomenų generavimą su naujais parametrais.
2. Dominančiam generatoriui pasirenkami rėžiai parametrų, kurie bus tiriami.
3. Metodui su kuriuo bus atliekamas eksperimentas pasirenkami parametrų rėžiai ir žingsniai, kuriais bus keičiami parametrai.
4. Modifikuotas TDMG, pasirinktu generatoriumi, generuoja testinius duomenis pasirinktam metodui ir renka statistikas. Generatoriaus darbas nutraukiamas kai:
    - Pasiekiamas 95 % kodo eilučių padengimas
    - Generatorius baigia darbą
5. Jei numatyti generatoriaus parametrų rėžiai dar neperžengti – pakeičiami generatoriaus parametrai numatytu žingsniu, grįžtama į ketvirtą žingsnį.
6. Analizuojami gauti rezultatai.
7. Daromos išvados.

Žingsniai 2 – 7 bus atliekami su dviem generatoriais – genetiniu (GA) be atkaitinimo (SA).

4.2. Eksperimento aplinka

Eksperimentas atliekamas toliau pateiktoje techninėje ir programinėje aplinkoje:

- Techninė įranga
    - Intel Core i3 M330 2.13GHz CPU
    - 3.5Gb RAM
- Programinė įranga
    - Xubuntu 12.04 LTS 32Bit (Linux)
    - Eclipse IDE 4.3.1
    - TDMG 1.3 (Modifikuotas tyrimui)

4.3. Tiriami testinių duomenų generatoriai

Tyrimui buvo pasirinkti du iš projekte realizuotų testinių duomenų generatorių. Toliau pateikiama pasirinkimo motyvacija.

- **Genetinis** – genetinis testinių duomenų generavimo algoritmas yra vienas žinomiausių bei dažniausiai naudojamų. Jis taip pat turi pakankamai parametrų, kurie pilnai aprašyti tolesniuose skyriuose, jog sukeltų problemų testuotojui dar gerai nežinančiam šio generatoriaus.
- **Atkaitinimo** – nors šis testinių duomenų generatorius yra mažiau žinomas nei kad jo brolis kopimo į kalną generatorius, bet su tam tikrais parametrais jie dirba identiškai. Dėl šios priežasties buvo pasirinktas daugiau parametrų turintis atkaitinimo generatorius.

Išsamesnis generatorių veikimas aprašomas skyriuose 2.3.3 bei 2.3.4, generatorių veiklos diagramos pateiktos Pav. 12 bei Pav. 13.

4.3.1. Tiriami parametrai ir jų rėžiai

Toliau aprašomi tiriamų generatorių parametrai ir jiems parinkti parametrų rėžiai.

4.3.1.1 Genetinis generatorius

Parametrai pateikiami ir paaiškinti žemiau (5 lentelė):

5 lentelė Genetinio generatoriaus parametrai

Parametras	Paaiškinimas
Generations	Kiek kartų (iteracijų) generavimo algoritmas vykdys.
StartPopulation	Kokio dydžio yra pradinė sugeneruojama populiacija.
ElitismValue	Genetinio algoritmo variacija. Šis parametras nusako kiek procentų, nepakeistų, esamos populiacijos geriausių narių pereis į sekančią iteraciją kaip vaikai.

Tyrimui parinkti parametrų rėžiai bei žingsniai:

- Generations:
  - Rėžiai:
    - [0; 1000]
  - Žingsniai:
    - { 0, 10, 25, 100, 250, 500, 1000}
- StartPopulation:
  - Rėžiai:
    - [10; 1000]
  - Žingsniai:
    - { 10, 100, 250, 500, 1000}
- ElitismValue:
  - Rėžiai:
    - [0; 100]
  - Žingsniai:
    - { 0, 10, 25, 50, 100}

4.3.1.2 Atkaitinimo generatorius

Parametrai pateikiami ir paaiškinti žemiau (6 lentelė):

6 lentelė Atkaitinimo generatoriaus parametrai

Parametras	Paaiškinimas
Iterations	Iteracijų kiekis, kurį generavimo algoritmas vykdys.
StartTemperature	Pradinė temperatūra generavimo pradžioje. Skirta skaičiuoti tikimybei ar algoritmas priims prastesnį rinkinį kaip pradinį.
TemperatureFall	Temperatūros kritimo koeficientas. Skirtas algoritmo eigoje pastoviai mažinti prastesnio rinkinio tapimo pradiniu tikimybę.

Tyrimui parinkti parametrų rėžiai bei žingsniai:

- Iterations:
  - Rėžiai:
    - [0; 1000]
  - Žingsniai
    - { 0, 10, 25, 100, 250, 500, 1000}
- StartTemperature:
  - Rėžiai:
    - [0; 1000]
  - Žingsniai:
    - { 10, 100, 250, 500, 1000}
- TemperatureFall:
  - Rėžiai:
    - [0; 100]
  - Žingsniai:
    - { 0, 10, 25, 50, 100}

4.4. Tyrime stebimi parametrai

Kad įvertinti generatorių efektyvumą buvo nuspręsta sekti:

- Pasiekiamą kodo eilučių padengimą procentais.
- Laiką, kurį generatorius užtruko bandydamas pasiekti 95 % kodo eilučių padengimą.

Stebint šiuos parametrus tikimasi atrasti generatorių parametrus, kurie užtikrina didžiausią kodo eilučių padengimą per trumpiausią laiką.

4.5. Tyrimui naudotas metodas

Kadangi tyrime sekamas kodo eilučių padengimas, buvo nuspręsta naudoti metodą turintį kiek įmanoma daugiau šakų, kad būtų kuo sunkiau padengti daug eilučių. Metodas taip pat parinktas kiek įmanoma paprastesnis. Tam puikiai tiko metodas paverčiantis paprastą skaičių į jo romėnišką formą – integerToRomanNumeral(int). Metodo informacija pateikiama JaCoCo [30] bibliotekos pateikta žemiau (7 lentelė).

7 lentelė Metodo integerToRomanNumeral parametrai

Parametras	Reikšmė
Eilučių kiekis	43
Šakų kiekis	30
Instrukcijų kiekis	194
Ciklomatinis sudėtingumas	16

Dalis tyrimui naudojamo metodo integerToRomanNumeral(int) kodo pateikiama žemiau (14 pav.).

```
public static String integerToRomanNumeral(int input) {
    if (input < 1 || input > 3999)
        return "Invalid Roman Number Value";
    String s = "";
    while (input >= 1000) {
        s += "M";
        input -= 1000;        }
    while (input >= 900) {
        s += "CM";
        input -= 900;
    }
    while (input >= 500) {
        s += "D";
        input -= 500;
    }
    while (input >= 400) {
        s += "CD";
        input -= 400;
    }
    while (input >= 100) {
        s += "C";
        input -= 100;
    }
    while (input >= 90) {
        s += "XC";
        input -= 90;
    }
    while (input >= 50) {
        s += "L";
        input -= 50;
    }
    while (input >= 40) {
        s += "XL";
        input -= 40;
    }
    while (input >= 10) {
        s += "X";
        input -= 10;
```

14 pav. Dalis metodo integerToRomanNumeral kodo

Kadangi metodo parametrai aprėpia labai menką sritį visų sveikojo skaičiaus reikšmių, kurias gali generuoti generatoriai buvo nuspręsta apriboti generatoriaus generuojamų parametrų rėžius. Minėtas sprendimas buvo priimtas dėl to, jog generuojant testinius duomenis visam įmanomam reikšmių diapazonui tikimybė padengti tokią mažą sritį yra labai menka. Taip pat įrankį naudojantis ir savo metodą testuojantis vartotojas žinos metodo ribas ir generavimą apribos. Kadangi visam metodui padengti reikia parametru tarp -1 ir 4000, buvo nuspręsta naudoti rėžius:

- Maksimalus skaičius: 5000;
- Minimalus skaičius: -100.

4.6. Rezultatai

4.6.1. Genetinis generatorius

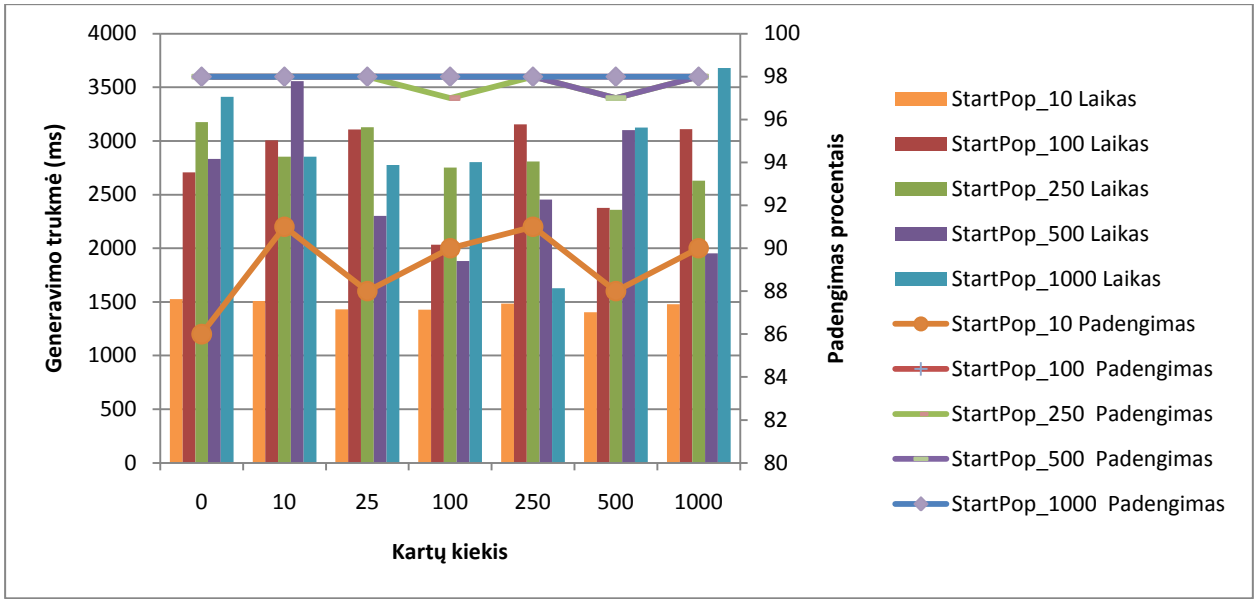
Atlikus genetinio generatoriaus testinių duomenų generavimą su pasirinktais parametrais buvo gauti rezultatai, kurių reprezentacinis pavyzdys pateiktas žemiau (8 lentelė). Pilna rezultatų lentelė pateikta priede A. Pirmuose trijuose stulpeliuose matyti bandomi parametrai, kituose dviejuose generavimo laikas sugaištas atlikti generavimui 100 kartų bei vidutinis pasiektas kodo

padengimas. Generavimas buvo atliekamas 100 kartų, nes genetinio algoritmo pradžia remiasi atsitiktiniais skaičiais.

8 lentelė Sugeneruotų genetinio generatoriaus rezultatų pavyzdys

Generations	ElitismValue	StartPopulation	Laikas (ms)	Padengimas (%)
…	…	…	…	…
0	0	10	1525	86
0	0	100	2707	98
0	0	250	3175	98
0	0	500	2833	98
0	0	1000	3410	98
0	10	10	1580	90
0	10	100	2656	97
0	10	250	2703	98
…	…	…	…	…

Kadangi ElitismValue parametras yra specifinis tiriamai genetinio generatoriaus variacijai, nuspręsta pirma vizualizuoti generatoriaus veikimą, kai šis parametras nenaudojamas (15 pav.). Čia kairėje vertikalioje ašyje matomas sugaištas laikas atlikti duomenų generavimui šimtą kartų, dešinėje vertikalioje matomas vidutinis pasiektas padengimas. Horizontalioje ašyje generavimo laikai bei padengimai sugrupuoti pagal naudota kartų kiekį (Generations parametras). Grafike laužte pažymėtas kiekvieno pradinės populiacijos dydžio parametro (StartPopulation parametras) pasiektas kodo padengimas, o stulpelinė diagrama rodo jų sugaištą laiką.



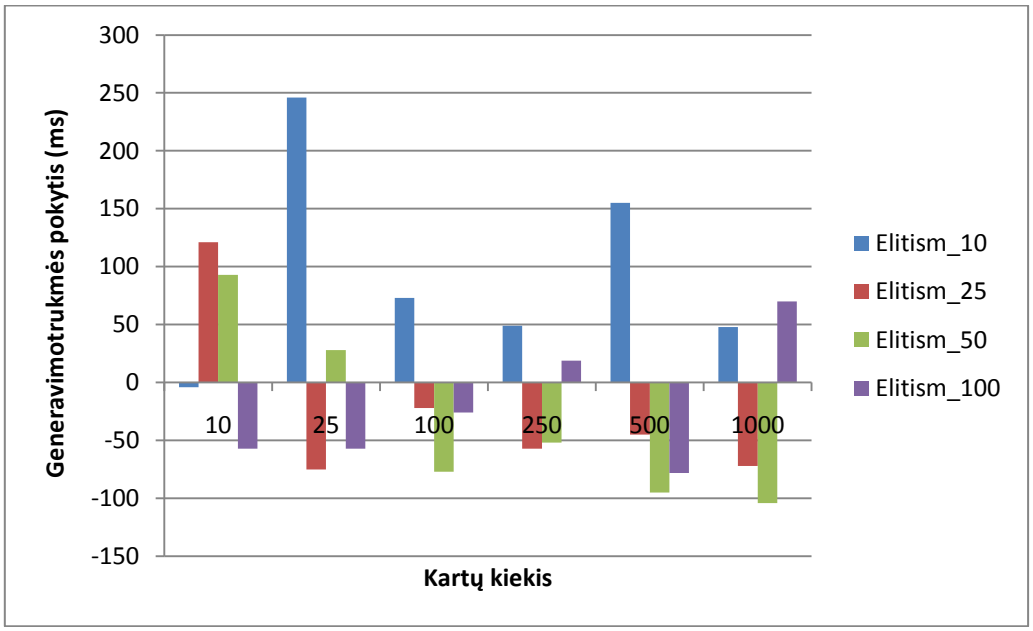
15 pav. GA vykdymo laikas ir kodo padengimas nenaudojant ElitismValue

Kaip matoma iš pateikto grafiko (15 pav.) visi duomenų generavimai su pradine populiacija daugiau nei 100 įvykdė kodo padengimo reikalavimą. Iš to, jog kodo padengimas bei sugaištas laikas menkai kito tarp skirtingų kartų kiekio parametrų (Generations parametras) bei pačių pradinės populiacijos dydžių, galima teigti, jog algoritmas su didesne nei šimto individų pradine populiacija algoritmas arba iškart padengdavo minimalų reikalaujamą kodo kiekį, arba turėdavo užtektinai atrinktų duomenų rinkinių, kad juos sukryžminus, gauti rinkinius

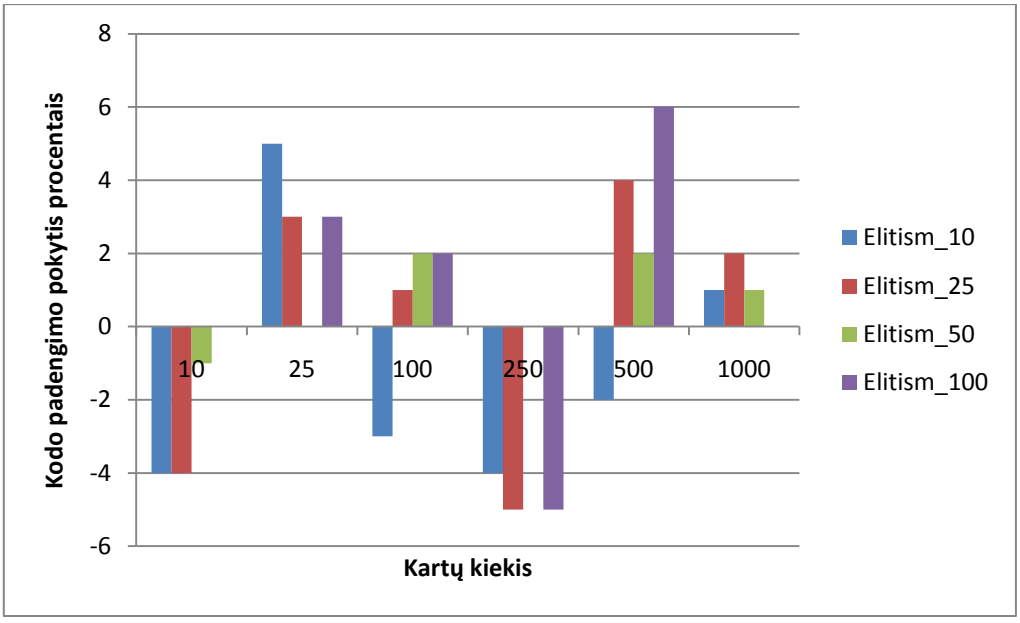
padengiančius likusį kodo eilučių kiekį. Tuo tarpu su pradinė populiacija lygia dešimčiai net su daug kartų algoritmas nesugebėjo pasiekti minimalaus reikalaujamo kodo padengimo.

Kadangi su didele pradine populiacija kodas buvo padengiamas akivaizdžiai per pirmą ar kelias pirmas algoritmo iteracijas, o ElitismValue parametro įtaka turi pasijausti tik su daug iteracijų, analizuoti ElitismValue parametro įtaką su didelėmis pradinėmis populiacijomis neverta. Kadangi padengimas su populiacijos dydžiu lygiu dešimčiai netenkino minimalaus padengimo reikalavimo, tai yra geriausias parametras su kuriuo verta tirti ElitismValue parametro įtaką algoritmo veikimui.

Žemiau pateikiame grafike (16 pav.) rodoma kaip skirtingos ElitismValue parametro reikšmės įtakoja generavimo laiką su skirtingomis kartų reikšmėmis kai pradinis populiacijos dydis yra dešimt lyginant su generavimo laikais nenaudojant ElitismValue. Sekančiame grafike (17 pav.) pateikiama ElitismValue įtaka kodo padengimui procentais lyginant su padengimu nenaudojant ElitismValue.

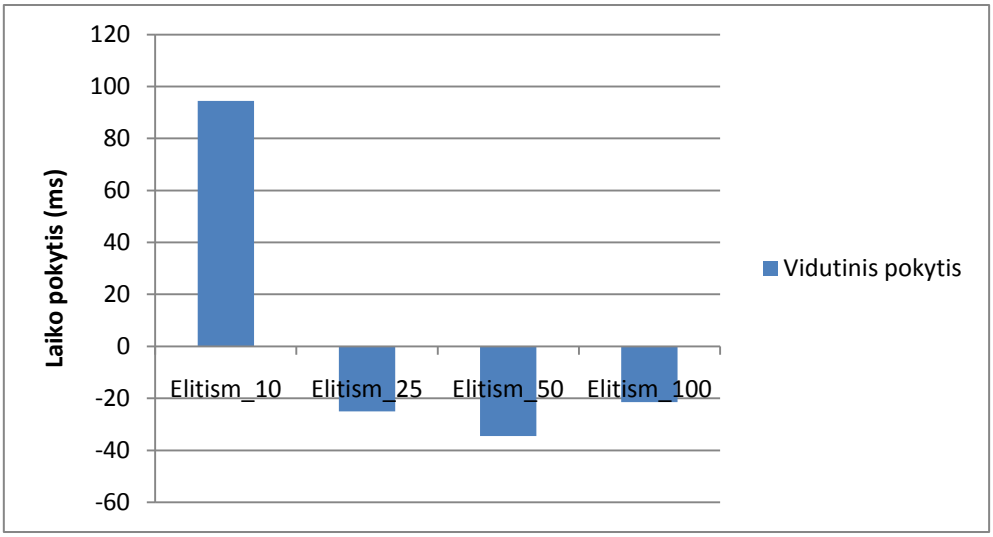


16 pav. Skirtingų ElitismValue reikšmių įtaka generavimo laikui (mažiau – geriau)

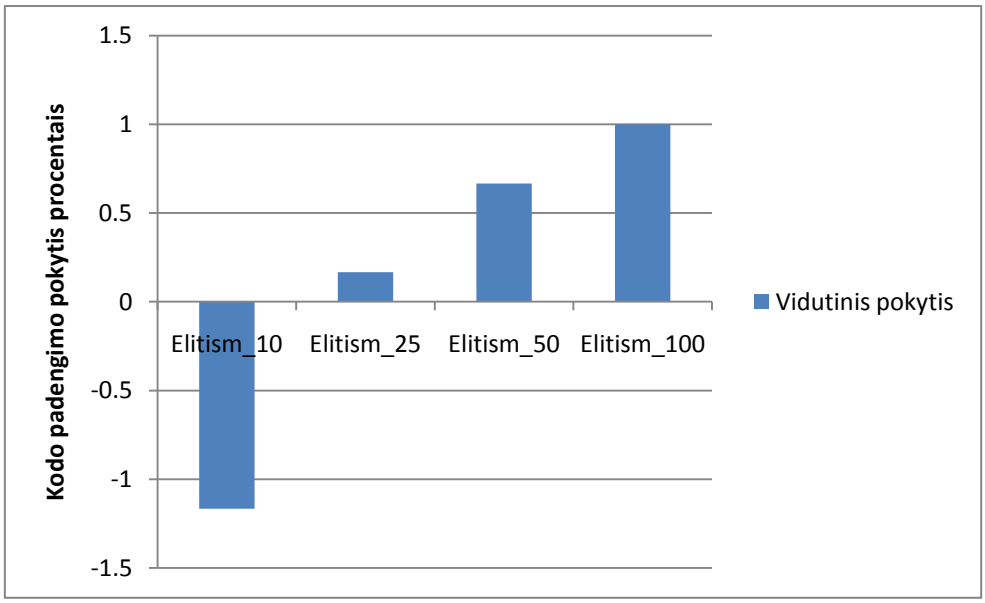


17 pav. Skirtingų ElitismValue reikšmių įtaka kodo padengimui (daugiau – geriau)

Toliau grafikuose pateikiama skirtingų ElitismValue parametro reikšmių vidutinė įtaka generavimo laikui (18 pav.) bei vidutiniam kodo padengimui (19 pav.) lyginant su generavimu be ElitismValue.



18 pav. Skirtingų ElitismValue reikšmių vidutinė įtaka generavimo laikui



19 pav. Skirtingų ElitismValue reikšmių vidutinė įtaka kodo padengimui

Kaip matoma iš grafikų (18 pav. bei 19 pav.) ElitismValue parametro reikšmė, parinkta nuo 25 iki 100, vidutiniškai pagerino vykdymo laiką bei kodo padengimą lyginant su pasiektais nenaudojant ElitismValue parametro.

4.6.2. Atkaitinimo generatorius

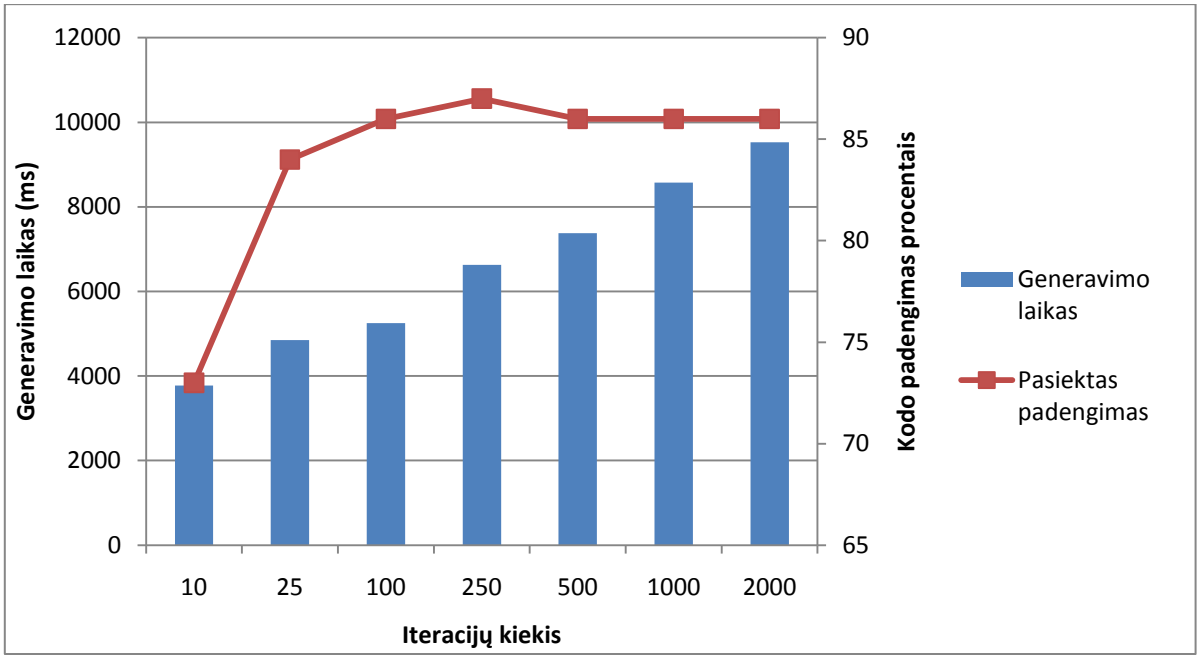
Atkaitinimo generatoriaus testinių duomenų generavimo su pasirinktais parametrais rezultatų reprezentacinis pavyzdys pateiktas žemiau (9 lentelė). Pilna rezultatų lentelė pateikta priede B. Pirmuose trijuose stulpeliuose matyti bandomi parametrai, kituose dviejuose generavimo laikas sugaištas atlikti generavimui 100 kartų bei vidutinis pasiektas kodo

padengimas. Generavimas buvo atliekamas 100 kartų, nes atkaitinimo algoritmas remiasi atsitiktinai sugeneruotais skaičiais.

9 lentelė Sugeneruotų atkaitinimo generatoriaus rezultatų pavyzdys

Iterations	StartTemperature	TemperatureFall	Laikas (ms)	Padengimas (%)
…	…	…	…	…
10	100	1	2376	78
10	100	2	2176	78
10	100	5	2225	69
10	100	10	1900	72
10	100	20	2301	70
25	0	0	4850	84
25	0	1	3601	79
25	0	2	4300	77
…	…	…	…	…

Atkaitinimo generatorius yra išplėstas kopimo į kalną generatorius. Jo veikimas naudojant StartTemperature bei TemperatureFall parametrus lygius nuliui atitinka kopimo į kalną generatoriaus veikimą. Tokio veikimo rezultatai pateikti grafike žemiau (20 pav.). Kaip matoma iš grafiko, generatorius nepasiekė užbrėžto kodo padengimo tikslo. Taip pat yra aiškiai matoma, jog nenaudojant temperatūros parametrų didesnis iteracijų kiekis reiškia garantuotą didesnį kodo padengimą. Vykdymo laikas auga tiesiškai priklausomai nuo iteracijų kiekio.

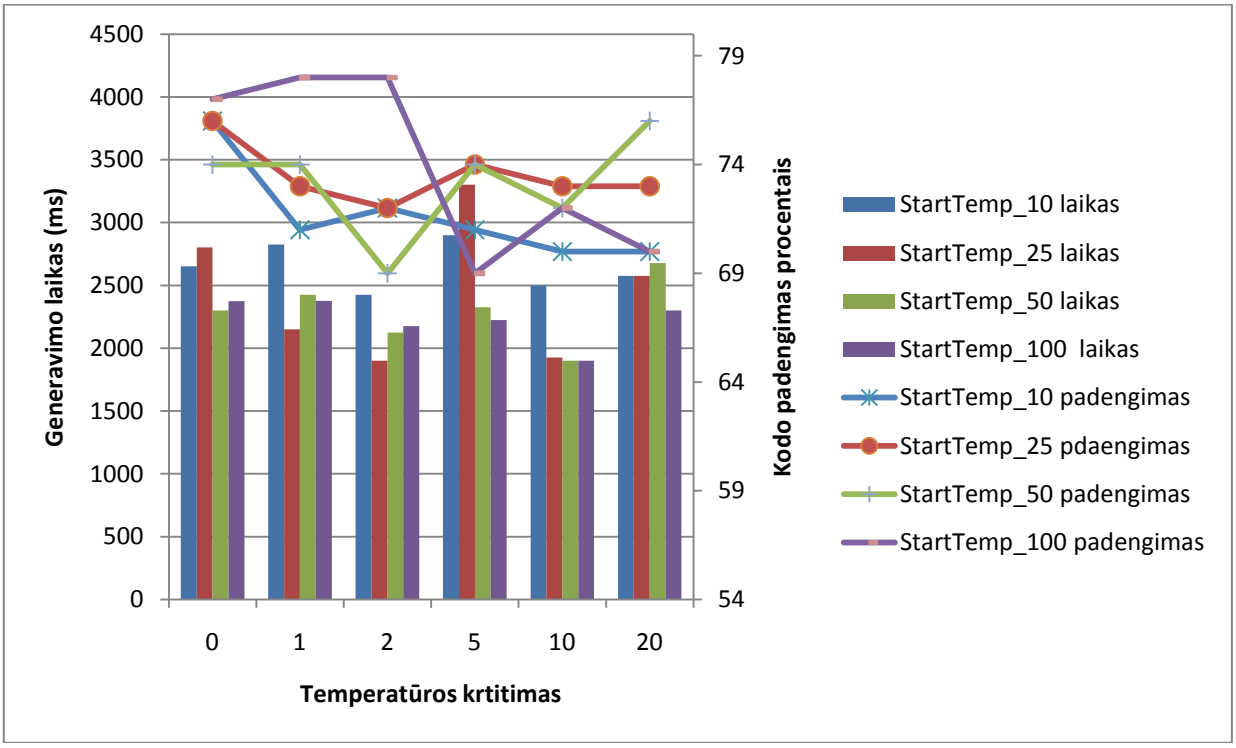


20 pav. Atkaitinimo generatoriaus veikimas be temperatūros parametrų

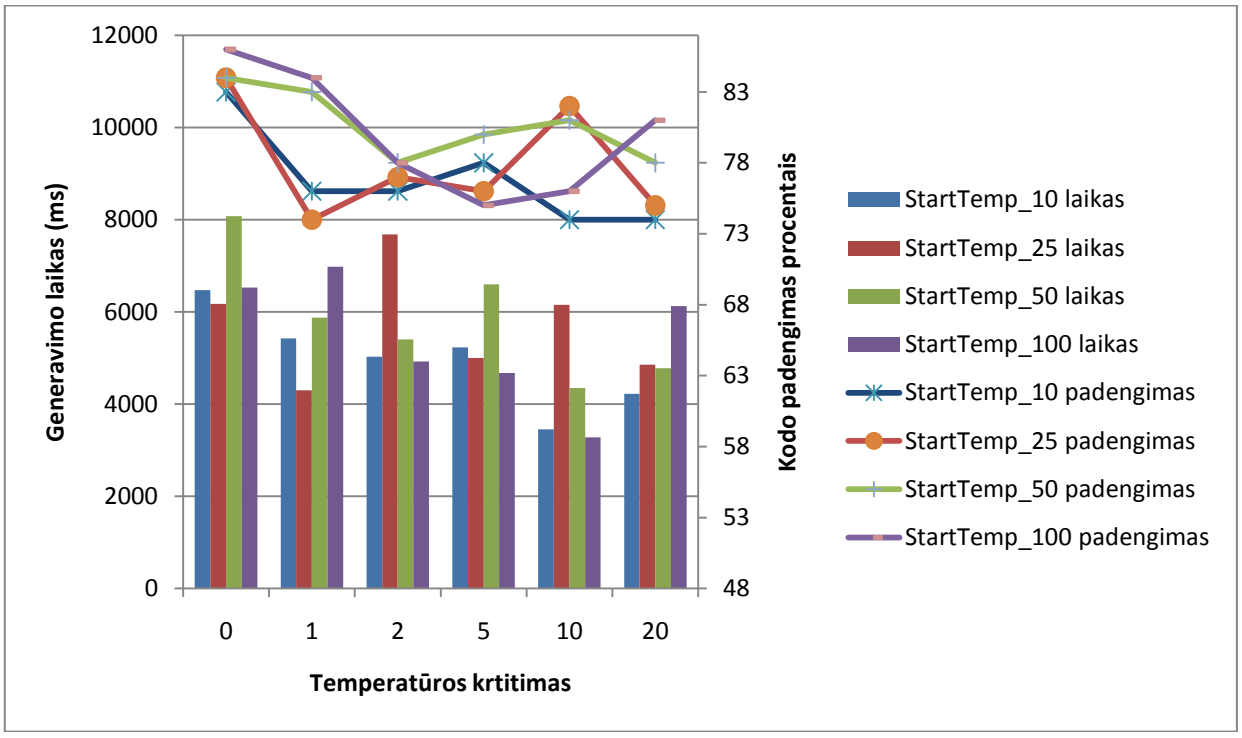
Žemiau pateikiamuose grafikuose pateiktos atkaitinimo generatoriaus generavimo statistikos su kintančiais temperatūrų parametrais ir fiksuotu iteracijų kiekiu.

Iš pateiktų grafikų (21 pav. bei 22 pav.) matyti jog temperatūros parametrų naudojimas su mažomis iteracijų reikšmėmis neturi akivaizdžios įtakos tiriamo metodo padengimui bei

vykdymo laikui. Ganėtinai chaotiški laiko bei padengimo svyravimai yra labai menki ir gali būti paaiškinti tuo jog algoritmas remiasi generuojamais atsitiktiniais skaičiais.



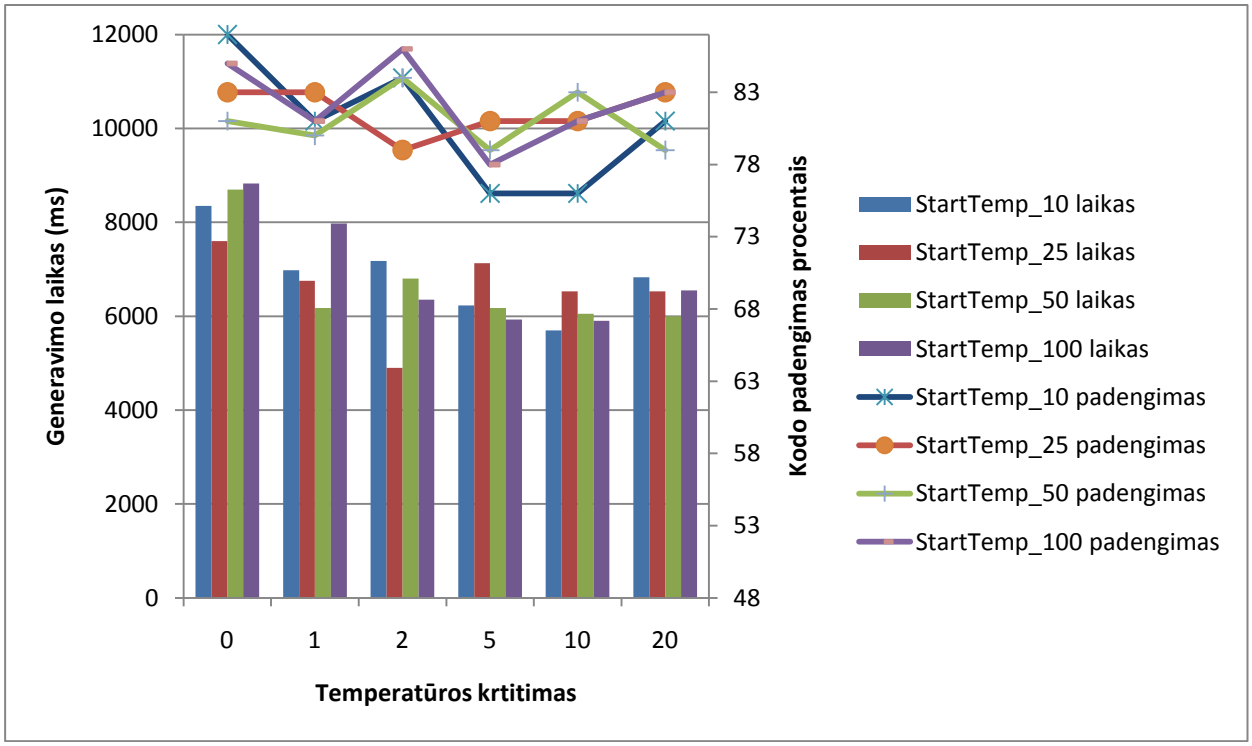
21 pav. Temperatūrų įtaka su iteracijų kiekiu fiksuotu ties 10



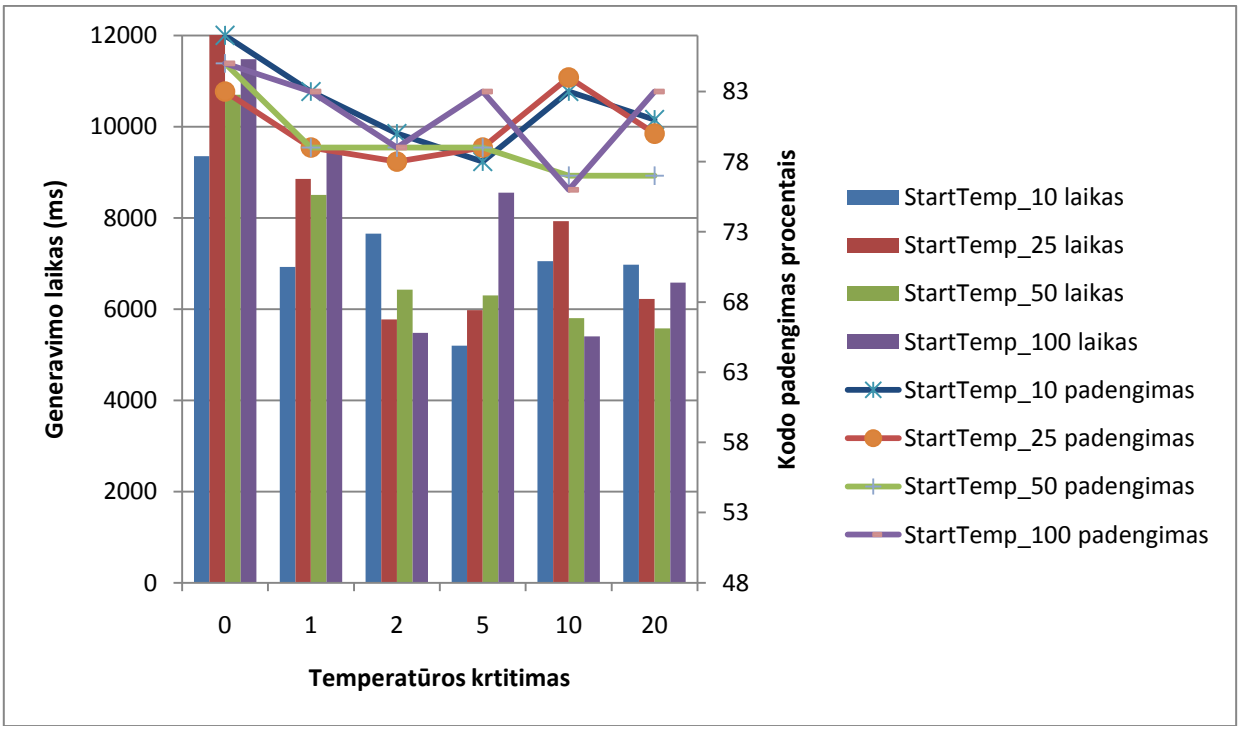
22 pav. Temperatūrų įtaka su iteracijų kiekiu fiksuotu ties 100

Iš žemiau pateiktų grafikų (23 pav. bei 24 pav.) matyti, kad šiame tyrime nematyti jokio nenuginčijamo ryšio tarp temperatūros parametrų ir pasiekiamų padengimo bei laiko rezultatų. Išskyrus, tai jog 24 pav. ir mažiau 23 pav. įmanoma įžvelgti galimą ilgesnį duomenų generavimo

laiką, bet aukštesnį minimalų pasiektą padengimą, kai temperatūros kritimas yra lygus nuliui. Vienintelis pastebimas nenuginčijamas dalykas tarp keturių grafikų (21 pav., 22 pav.,23 pav. bei 24 pav.) yra, tai jog visais atvejais daugiau iteracijų reiškė geresnį minimalų pasiektą padengimą. Šitai taip pat sutampa su pastebėjimu, jog nenaudojant temperatūros parametrų didesnis iteracijų kiekis garantuoja didesnį kodo padengimą (20 pav.).



23 pav. Temperatūrų įtaka su iteracijų kiekiu fiksuotu ties 1000



24 pav. Temperatūrų įtaka su iteracijų kiekiu fiksuotu ties 2000

4.6.3. Išvados ir apibendrinimas

Atlikus tyrimą iš gautų rezultatų galima padaryti sekančias išvadas:

1. ElitismValue parametro naudojimas genetiniam generatoriui pagerina kodo padengimą bei darbo laiką, kai naudojama maža pradinė populiacija.
2. Naudojant didelę pradinę genetinio generatoriaus populiaciją pasiekiami geresni rezultatai nei naudojant mažą pradinę populiaciją su dideliu kartų kiekiu.
3. Didesnis atkaitinimo generatoriaus iteracijų kiekis beveik visada garantuoja geresnį kodo padengimą.
4. Kadangi atkaitinimo generatorius remiasi atsitiktiniais skaičiais jo parametrų įtaką generavimo efektyvumui nustatyti yra labai sunku.

5. IŠVADOS

Šiame darbe buvo:

1. Trumpai aptartas automatinis testinių duomenų generavimas, jo nauda, į kokias grupes jis skirstomas bei kokiomis metrikomis jis nusakomas.
2. Suprojektuota, realizuota, ir ištestuota testinių duomenų generavimo sistema visiškai patenkinanti surinktus užsakovo reikalavimus.
3. Atliktas testinių duomenų generatorių efektyvumo nuo generavimui naudojamų parametrų tyrimas.

Darbo metu buvo padarytos išvados:

1. Programų kokybės užtikrinimas vis dar aktuali problema.
2. Testinių duomenų generavimas vis dar nėra visiškai išspręsta problema.
3. Eclipse IDE plėtiniams kurti skirti įrankiai nėra išbaigti.
4. Eksperimento metu pastebėta, jog genetiniam generatoriui ElitismValue parametro naudojimas kodo padengimą bei darbo laiką, kai naudojama maža pradinė populiacija.
5. Naudojant didelę pradinę genetinio generatoriaus populiaciją pasiekiami geresni rezultatai nei naudojant mažą pradinę populiaciją su dideliu kartų kiekiu.
6. Didesnis atkaitinimo generatoriaus iteracijų kiekis beveik visada garantuoja geresnį kodo padengimą.
7. Abu tirti generatoriai remiasi atsitiktinai sugeneruotais skaičiais kas reiškia, kad rezultatai bei generavimo metrikos gali kisti nenuspėjamai net su tais pačiais generavimo parametrais.

6. LITERATŪRA

[1] Dowson, M. (March 1997). „The Ariane 5 Software Failure“. Software Engineering Notes 22 (2): 84.

[2] Nancy G. Leveson and Clark S. Turner. An investigation of the Therac-25 accidents. IEEE Computer, 26(7):18-41, July 1993.

[3] Stefan Wagner, Klaus Lochmann, Sebastian Winter, Andreas Goeb, Michael Klaes, „Quality models in practice: A preliminary analysis,“ esem, pp.464-467, 2009 3rd International Symposium on Empirical Software Engineering and Measurement, 2009

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

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

[6] Gerard Meszaros. 2006. Xunit Test Patterns: Refactoring Test Code. Prentice Hall PTR, Upper Saddle River, NJ, USA

[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, 742-762 psl., 2010.

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

[10] 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.

[11] 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.

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

[13] Bj Rollison, „Parametrized random test data generation“, PNSQC, 2011.

[14] 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.

[15] R. A. DeMillo and A. J. Offutt. Constraint-based automatic testdata generation. IEEE Transactions on Software Engineering, ąų(9):900-910, September 1991

[16] JAVA prieiga internete, http://www.java.com [žiūrėta 2014.04.30].

[17] Jtest prieiga internete, http://www.parasoft.com/jtest [žiūrėta 2014.04.30].

[18] Parasoft prieiga internete, http://www.parasoft.com/ [žiūrėta 2014.04.30].

[19] CodePro Analytix, prieiga internete https://developers.google.com/java-dev-tools/codepro/doc/ [žiūrėta 2014.04.30].

[20] Apache License 2.0, prieiga internete http://www.apache.org/licenses/LICENSE-2.0.html [žiūrėta 2014.04.30].

[21] JUnit, prieiga internete http://junit.org/ [žiūrėta 2014.04.30].

[22] Eclipse, prieiga internete https://www.eclipse.org/ [žiūrėta 2014.04.30].

[23] Rational Developer, prieiga internete http://www-03.ibm.com/software/products/en/developer [žiūrėta 2014.04.30].

[24] IBM WebSphere Studio, prieiga internete https://www.ibm.com/developerworks/websphere/zones/devtools/ [žiūrėta 2014.04.30].

[25] jPET, prieiga internete http://costa.ls.fi.upm.es/pet/download [žiūrėta 2014.04.30].

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

[27] UML http://www.omg.org/spec/UML/ [žiūrėta 2014.04.30].

[28] Eclipse User Interface Guidelines prieiga internete https://www.eclipse.org/articles/Article-UI-Guidelines/Contents.html [žiūrėta 2014.04.30].

[29] Miller, Edward; and Howden, William E.; Software Testing and Validation Techniques, Long Beach, CA: IEEE Computer Society Press, 1978 (first edition), pp. 16-19

[30] JaCoCo Java Code Coverage Library, Mountainminds GmbH & Co, prieiga internete http://www.eclemma.org/jacoco/, [žiūrėta 2014.04.30].

7. TERMINŲ IR SANTRUMPŲ ŽODYNAS

IDE (angl. *Integrated Development Environment*)	Integruota programinės įrangos kūrimo aplinka.
GA (angl. *Genetic Algorithm*)	Genetinio algoritmo pagrindu veikiantis duomenų generavimo metodas.
GPL (angl. *GNU General Public License*)	GNU bendroji viešoji licencija
HC (angl. *Hill Climbing*)	Kopimo į kalną optimizacijos algoritmo pagrindu veikiantis testinių duomenų generavimo metodas.
HTML (angl *Hyper text Markup Language*)	Turinio aprašymo standartas.
TXT	Tekstinio failo formatas.
RA (angl. *Random Algorithm*)	Atsitiktinio generavimo algoritmo pagrindu veikiantis duomenų generavimo metodas.
SA (angl. *Simulated Annealing*)	Modeliuojamo atkaitinimo optimizacijos algoritmo pagrindu veikiantis duomenų generavimo metodas.
UC (angl. *Use Case*)	Panaudojimo atvejis.
UML (angl. *Unified Modeling Language*)	unifikuota modeliavimo kalba
TDMG (angl. *Test Data Multi-Generator*)	Automatinis testinių duomenų multi-generatorius.

8. PRIEDAI

Prieduose A bei B pateikiamos pilnos tyrimo rezultatų lentelės.

Prieduose taip pat yra pateikiami šio darbo autoriaus magistro studijų metu su kolegomis parašyti straipsniai, viename iš darbų naudojamas projekto dalyje su kolega sukurtas įrankis:

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

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

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

Priedas A

Testų generavimo rezultatai tiriant genetinį generatorių

Generations	ElitismValue	StartPopulation	Laikas (ms)	Vid. Padengimas %
0	0	10	1525	86
0	0	100	2707	98
0	0	250	3175	98
0	0	500	2833	98
0	0	1000	3410	98
0	10	10	1580	90
0	10	100	2656	97
0	10	250	2703	98
0	10	500	4004	98
0	10	1000	2380	97
0	25	10	1559	84
0	25	100	3433	98
0	25	250	2381	98
0	25	500	2782	98
0	25	1000	2703	98
0	50	10	1451	88
0	50	100	2882	98
0	50	250	2158	98
0	50	500	3978	98
0	50	1000	2801	97
0	100	10	1702	92
0	100	100	2982	98
0	100	250	3385	98
0	100	500	2878	97
0	100	1000	2306	98
10	0	10	1509	91
10	0	100	3005	98
10	0	250	2855	98
10	0	500	3557	98
10	0	1000	2855	98
10	10	10	1505	87
10	10	100	2658	98
10	10	250	2104	98
10	10	500	2428	97
10	10	1000	2781	98
10	25	10	1630	87
10	25	100	2784	98
10	25	250	2403	97
10	25	500	2805	98
10	25	1000	2752	98
10	50	10	1602	90
10	50	100	2828	98
10	50	250	2759	98
10	50	500	2629	98

10	50	1000	2808	98
10	100	10	1452	91
10	100	100	2455	98
10	100	250	2106	98
10	100	500	3781	97
10	100	1000	2734	98
25	0	10	1432	88
25	0	100	3108	98
25	0	250	3127	98
25	0	500	2303	98
25	0	1000	2777	98
25	10	10	1978	93
25	10	100	2555	97
25	10	250	3882	98
25	10	500	2380	98
25	10	1000	2929	98
25	25	10	1357	91
25	25	100	2834	98
25	25	250	3734	98
25	25	500	3159	98
25	25	1000	2510	98
25	50	10	1460	88
25	50	100	2375	97
25	50	250	2725	98
25	50	500	3080	98
25	50	1000	2859	97
25	100	10	1375	91
25	100	100	3106	98
25	100	250	2955	98
25	100	500	3407	98
25	100	1000	2450	98
100	0	10	1428	90
100	0	100	2033	98
100	0	250	2752	97
100	0	500	1881	98
100	0	1000	2803	98
100	10	10	1501	87
100	10	100	2577	98
100	10	250	2427	97
100	10	500	2335	98
100	10	1000	2080	98
100	25	10	1406	91
100	25	100	3484	97
100	25	250	2782	97
100	25	500	2584	98
100	25	1000	2426	98

100	50	10	1351	92
100	50	100	2704	98
100	50	250	2560	98
100	50	500	3626	97
100	50	1000	2805	98
100	100	10	1402	92
100	100	100	2602	97
100	100	250	2651	98
100	100	500	2008	98
100	100	1000	2159	98
250	0	10	1484	91
250	0	100	3156	98
250	0	250	2808	98
250	0	500	2453	98
250	0	1000	1629	98
250	10	10	1533	87
250	10	100	3135	98
250	10	250	2827	98
250	10	500	2579	98
250	10	1000	3032	98
250	25	10	1427	86
250	25	100	3400	98
250	25	250	2677	98
250	25	500	2927	97
250	25	1000	3535	97
250	50	10	1432	91
250	50	100	3327	98
250	50	250	3085	98
250	50	500	2952	98
250	50	1000	2875	98
250	100	10	1503	86
250	100	100	2903	98
250	100	250	2534	98
250	100	500	2954	98
250	100	1000	2376	98
500	0	10	1403	88
500	0	100	2376	98
500	0	250	2358	97
500	0	500	3100	97
500	0	1000	3126	98
500	10	10	1558	86
500	10	100	2959	98
500	10	250	3230	98
500	10	500	2353	98
500	10	1000	2555	98
500	25	10	1358	92

500	25	100	2425	98
500	25	250	2177	98
500	25	500	3183	98
500	25	1000	2731	98
500	50	10	1308	90
500	50	100	2727	97
500	50	250	2858	97
500	50	500	2481	98
500	50	1000	2505	98
500	100	10	1325	94
500	100	100	2701	98
500	100	250	3056	98
500	100	500	3603	98
500	100	1000	2151	98
1000	0	10	1480	90
1000	0	100	3109	98
1000	0	250	2630	98
1000	0	500	1954	98
1000	0	1000	3681	98
1000	10	10	1528	91
1000	10	100	2454	98
1000	10	250	3110	98
1000	10	500	2232	98
1000	10	1000	2555	98
1000	25	10	1408	92
1000	25	100	2352	98
1000	25	250	3803	98
1000	25	500	2929	98
1000	25	1000	2454	97
1000	50	10	1376	91
1000	50	100	2452	97
1000	50	250	2432	98
1000	50	500	2207	97
1000	50	1000	2555	98
1000	100	10	1550	90
1000	100	100	2927	98
1000	100	250	2435	98
1000	100	500	2356	98
1000	100	1000	2805	98

Priedas B

Testų generavimo rezultatai tiriant atkaitinimo generatorių

Iterations	StartTemperature	TemperatureFall	Laikas (ms)	Vid. Padengimas %
10	0	0	3776	73
10	0	1	3600	72
10	0	2	3176	77
10	0	5	3625	74
10	0	10	3476	74
10	0	20	2801	72
10	10	0	2651	76
10	10	1	2826	71
10	10	2	2425	72
10	10	5	2900	71
10	10	10	2500	70
10	10	20	2575	70
10	25	0	2801	76
10	25	1	2150	73
10	25	2	1900	72
10	25	5	3301	74
10	25	10	1925	73
10	25	20	2576	73
10	50	0	2301	74
10	50	1	2426	74
10	50	2	2125	69
10	50	5	2325	74
10	50	10	1901	72
10	50	20	2676	76
10	100	0	2375	77
10	100	1	2376	78
10	100	2	2176	78
10	100	5	2225	69
10	100	10	1900	72
10	100	20	2301	70
25	0	0	4850	84
25	0	1	3601	79
25	0	2	4300	77
25	0	5	4101	74
25	0	10	4425	81
25	0	20	3801	76
25	10	0	3951	77
25	10	1	3876	75
25	10	2	3626	74
25	10	5	2975	77
25	10	10	2875	75
25	10	20	3225	74
25	25	0	3600	81
25	25	1	3525	78

25	25	2	3026	70
25	25	5	3051	71
25	25	10	1775	71
25	25	20	3501	77
25	50	0	3901	80
25	50	1	3351	78
25	50	2	3200	71
25	50	5	4201	76
25	50	10	3601	79
25	50	20	3101	77
25	100	0	4375	78
25	100	1	3951	78
25	100	2	3401	73
25	100	5	4525	76
25	100	10	5626	71
25	100	20	5025	78
100	0	0	5251	86
100	0	1	6101	80
100	0	2	8426	83
100	0	5	4851	74
100	0	10	5701	80
100	0	20	9801	79
100	10	0	6475	83
100	10	1	5425	76
100	10	2	5025	76
100	10	5	5226	78
100	10	10	3451	74
100	10	20	4226	74
100	25	0	6176	84
100	25	1	4301	74
100	25	2	7676	77
100	25	5	5001	76
100	25	10	6151	82
100	25	20	4851	75
100	50	0	8075	84
100	50	1	5876	83
100	50	2	5401	78
100	50	5	6600	80
100	50	10	4350	81
100	50	20	4776	78
100	100	0	6526	86
100	100	1	6976	84
100	100	2	4926	78
100	100	5	4675	75
100	100	10	3276	76
100	100	20	6125	81

250	0	0	6626	87
250	0	1	5825	77
250	0	2	8026	81
250	0	5	7276	83
250	0	10	4301	77
250	0	20	4726	77
250	10	0	6626	81
250	10	1	5601	83
250	10	2	3576	79
250	10	5	4901	80
250	10	10	4975	80
250	10	20	6126	81
250	25	0	6401	85
250	25	1	4876	77
250	25	2	6325	85
250	25	5	4501	74
250	25	10	5275	80
250	25	20	4500	77
250	50	0	7900	86
250	50	1	4175	76
250	50	2	4150	77
250	50	5	6276	85
250	50	10	5750	79
250	50	20	4701	76
250	100	0	8100	86
250	100	1	6951	81
250	100	2	3776	77
250	100	5	2575	71
250	100	10	6626	81
250	100	20	3075	72
500	0	0	7376	86
500	0	1	4950	80
500	0	2	6225	81
500	0	5	5526	80
500	0	10	6326	81
500	0	20	4901	77
500	10	0	7301	85
500	10	1	6026	84
500	10	2	5175	81
500	10	5	7426	83
500	10	10	5476	80
500	10	20	5726	83
500	25	0	8175	86
500	25	1	4376	78
500	25	2	5250	79
500	25	5	6050	85

500	25	10	3651	76
500	25	20	4976	80
500	50	0	8425	85
500	50	1	6700	84
500	50	2	6251	83
500	50	5	6800	85
500	50	10	5150	78
500	50	20	5375	78
500	100	0	7426	84
500	100	1	6900	85
500	100	2	4650	78
500	100	5	6701	79
500	100	10	3600	69
500	100	20	4975	76
1000	0	0	8576	86
1000	0	1	6275	83
1000	0	2	4150	73
1000	0	5	5201	78
1000	0	10	5525	81
1000	0	20	6350	83
1000	10	0	8350	87
1000	10	1	6976	81
1000	10	2	7176	84
1000	10	5	6226	76
1000	10	10	5700	76
1000	10	20	6826	81
1000	25	0	7600	83
1000	25	1	6751	83
1000	25	2	4900	79
1000	25	5	7126	81
1000	25	10	6526	81
1000	25	20	6526	83
1000	50	0	8700	81
1000	50	1	6176	80
1000	50	2	6800	84
1000	50	5	6175	79
1000	50	10	6051	83
1000	50	20	6000	79
1000	100	0	8826	85
1000	100	1	7976	81
1000	100	2	6350	86
1000	100	5	5926	78
1000	100	10	5900	81
1000	100	20	6551	83
2000	0	0	9525	86
2000	0	1	7976	85

2000	0	2	7725	84
2000	0	5	8426	81
2000	0	10	6950	81
2000	0	20	6125	80
2000	10	0	9351	87
2000	10	1	6926	83
2000	10	2	7651	80
2000	10	5	5200	78
2000	10	10	7051	83
2000	10	20	6975	81
2000	25	0	13351	83
2000	25	1	8851	79
2000	25	2	5775	78
2000	25	5	5975	79
2000	25	10	7925	84
2000	25	20	6225	80
2000	50	0	10701	85
2000	50	1	8501	79
2000	50	2	6425	79
2000	50	5	6301	79
2000	50	10	5800	77
2000	50	20	5576	77
2000	100	0	11476	85
2000	100	1	9526	83
2000	100	2	5475	79
2000	100	5	8551	83
2000	100	10	5400	76
2000	100	20	6576	83

Priedas C

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 pastarųjų 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.

(1) (1)

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.

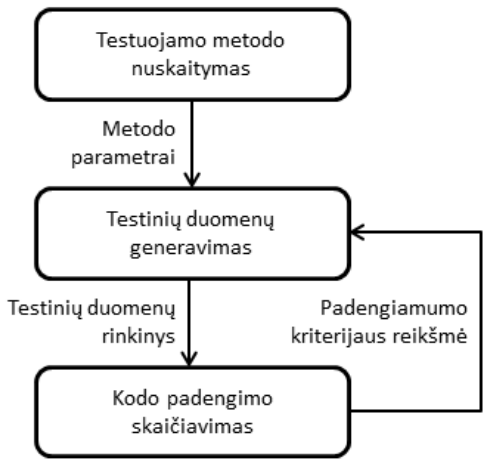
IV. Tyrimo metodika ir naudoti įrankiai

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.



1 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 visų generacijų aritmetinis vidurkis.

I 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

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.

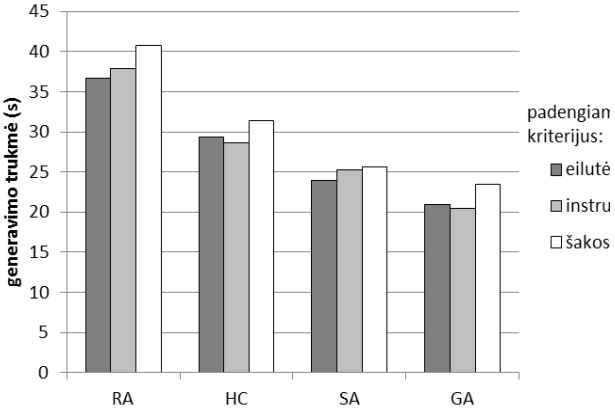
II LENTELĖ. GENERAVIMO METODŲ PARAMETRŲ REIKŠMĖS

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

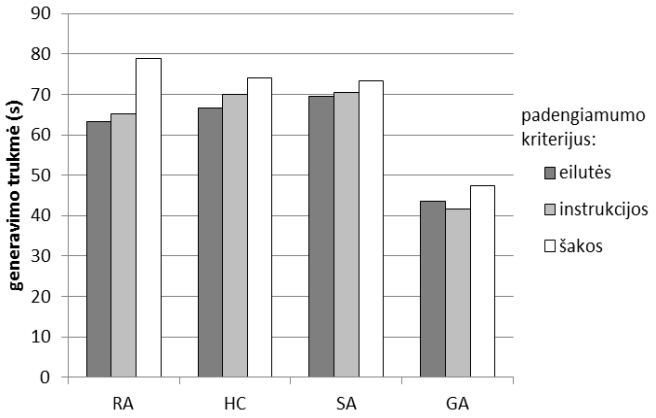
V. TYRIMO REZULTATAI

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).

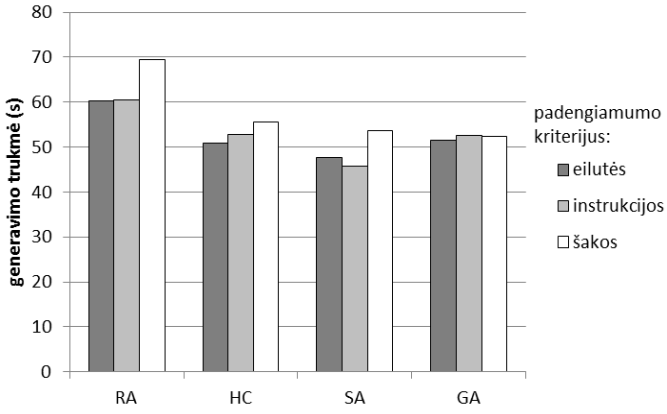
Iš šių rezultatų matyti, jog beveik visais atvejais testinių duomenų generavimas remiantis šakų padengiamumu užtrunka ilgiau negu remiantis instrukcijų arba kodo eilučių padengiamumais.



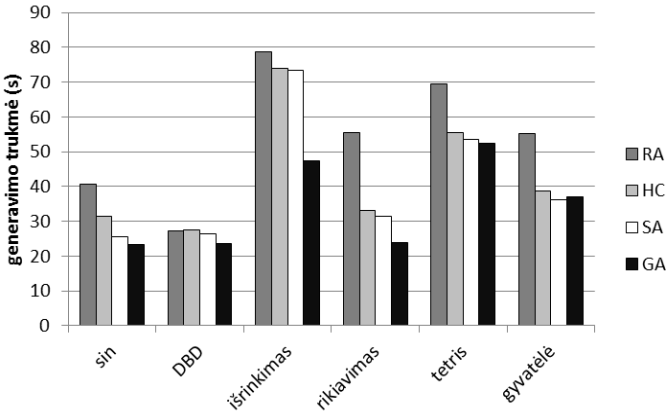
2 pav. Testinių duomenų generavimo sinuso skaičiavimo algoritmui trukmė



3 pav. Testinių duomenų generavimo įrašo išrinkimo algoritmui trukmė



4 pav. Testinių duomenų generavimo tetrio žaidimo logikos klasei trukmė



5 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.

Priedas D

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:

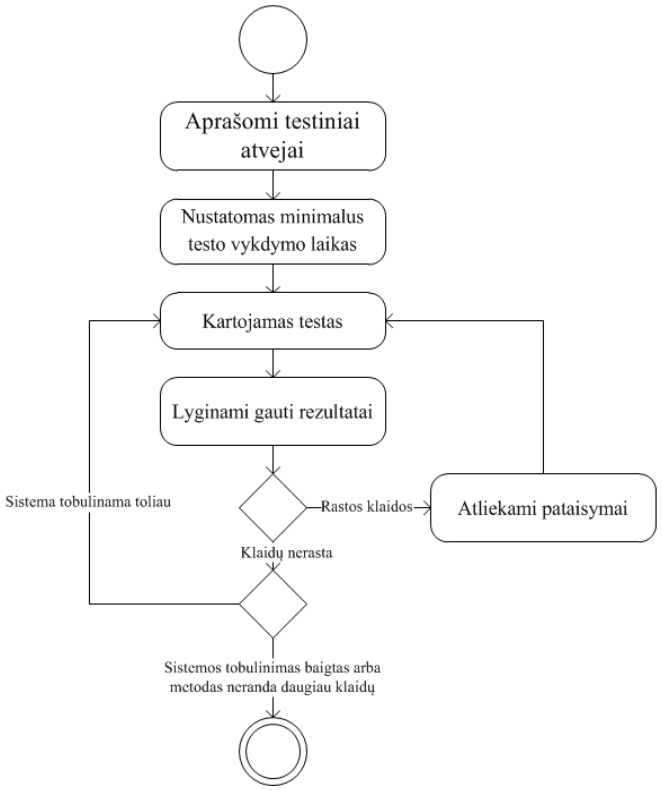
- Testuojama programinė įranga turi būti kuriama mobiliems įrenginiams.
- Testuojama programinė įranga turi turėti grafinę vartotojo sąsają.
- Turi būti prieinamas testuojamos programinės įrangos išeities kodas.

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.



6 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

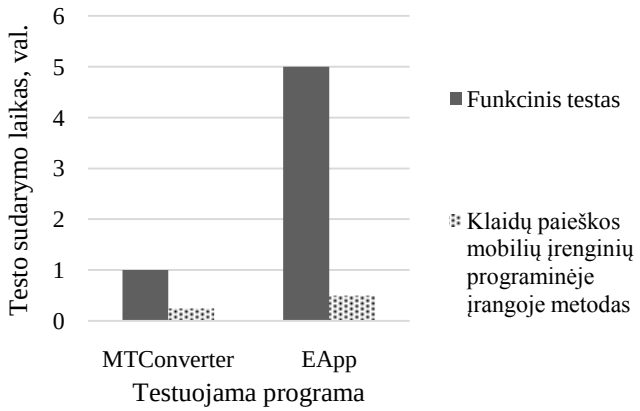
V. TYRIMO REZULTATAI

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 7 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



7 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 8 pav.

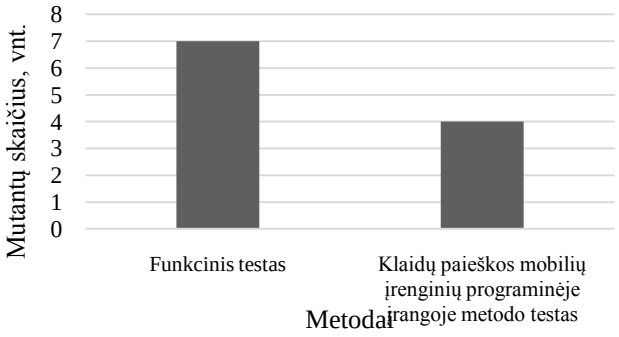
Programos MTConverter funkcinio testo ir klaidų paieškos mobilių įrenginių programinėje įrangoje metodo atliktų testų rezultatai yra pateikiami V ir VI lentelėse. Šiose lentelėse pateikiami tik tie mutantai, kurie buvo sugeneruoti, t.y. mutantai, kurie nebuvo sugeneruoti, nepateikiami.

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



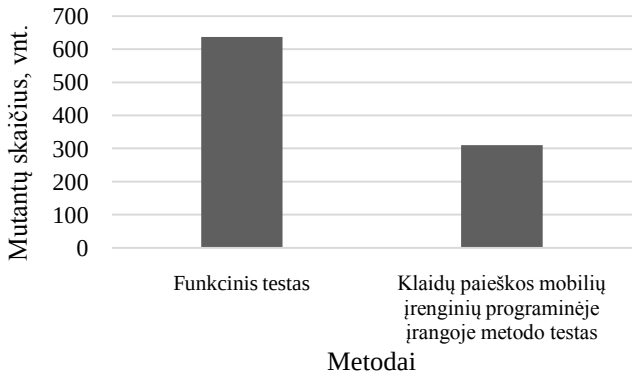
8 pav. Aptiktų mutantų EApp programoje skirtingais metodais palyginimas

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



9 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 9 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.

Priedas E

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

Priedas F

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. STI5-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

Priedas G

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**