KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ INŽINERIJOS STUDIJŲ PROGRAMA

DARIUS LIEPINAITIS

KLAIDŲ PAIEŠKOS MOBILIŲ ĮRENGINIŲ PROGRAMINĖJE ĮRANGOJE METODO, TAIKANT PROGRAMOS VYKDYMO LAIKINES CHARAKTERISTIKAS, KŪRIMAS IR TYRIMAS

Magistro darbas

Vadovas
Doc. Dr. Š. Packevičius

KAUNAS, 2014

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ INŽINERIJOS STUDIJŲ PROGRAMA

DARIUS LIEPINAITIS

KLAIDŲ PAIEŠKOS MOBILIŲ ĮRENGINIŲ PROGRAMINĖJE ĮRANGOJE METODO, TAIKANT PROGRAMOS VYKDYMO LAIKINES CHARAKTERISTIKAS, KŪRIMAS IR TYRIMAS

Magistro darbas

Vadovas
Doc. Dr. Š. Packevičius

.....

Recenzentas
Prof. Dr. E. Bareiša

.....

Studentas
D. Liepinaitis

.....

KAUNAS, 2014

AUTENTIŠKUMO PATVIRTINIMAS

Patvirtinu, kad įteikiamas baigiamasis darbas „Klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant programos vykdymo laikines charakteristikas, kūrimas ir tyrimas“:

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

Darius Liepinaitis _____

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

THE BUGS SEARCH METHOD FOR MOBILE DEVICES SOFTWARE USING CODE EXECUTION TIMING CHARACTERISTICS

SUMMARY

The number of mobile phones applications is very big and it is increasing. Most of these applications are not successful in gaining users audience and are not used widely. Although these applications give the functionality that is needed by the user. These applications do not meet users' needs, because they do not function as expected by the users and users are not able to use those applications.

After analyzing testing methods of graphical user interface it has been noticed that one of the main problems is that most testing methods are very time-consuming. Big part of time is used when tests are created. Because of this reason it was decided to create testing method that would solve this problem. Then it was decided to create a tool using created graphical user interface functionality testing method for "Windows Phone 8.0" operating system.

The main purpose of this work is to propose new testing method. This method needs to be adapted to mobile devices graphical user interface functionality testing. It also needs to solve time-consumption problem.

This work contains the bugs search method for mobile devices software using code execution timing characteristics. Method is based on principle that when action does not get executed during specified time it is considered that test failed and bugs were found. This method is adapted to mobile devices and automated testing. The testing tool was created as add-on for "Microsoft Visual Studio" integrated development environment.

The created bugs search method for mobile devices software using code execution timing characteristics effectiveness was proved by conducting an experimental study. During experimental study four programs were tested. These programs were created for "Windows Phone" operating system. After conducting experimental study results showed that time-consumption decreased by 7.6 times comparing with functional testing method. Also results showed that average mutants detection is 39.75%.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

1 ĮVADAS

Mobiliųjų telefonų programų šiuo metu yra sukurta labai daug ir jos yra kuriamos toliau. Tačiau didelė dalis šių programų nėra sėkmingos ir nėra 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 reikalingas žmogus 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 [8].

Šio darbo sprendžiama problema yra:

- Programinės įrangos grafinės sąsajos testavimas reikalauja didelių laiko sąnaudų.

Šio darbo tikslas yra:

- Sumažinti programinės įrangos grafinės vartotojo sąsajos testavimo laiko sąnaudas.

Šio darbo uždaviniai yra:

- Išanalizuoti sukurtą klaidų paieškos mobilių įrenginių programinėje įrangoje metodą, taikant programos vykdymo laikines charakteristikas.
- Pateikti rekomendacijas, kada būtų naudinga naudoti sukurtą metodą.

1.1 Dokumento paskirtis

Šiame dokumente yra pateikiamas sukurtas ir analizuojamas klaidų paieškos mobilių įrenginių programinėje įrangoje metodas, taikant programos vykdymo laikines charakteristikas. Taip pat pateikiama egzistuojančių testavimo metodų ir testavimo įrankių analizė. 3 skyriuje yra pateikiamas sukurtas testavimo įrankis. Taip pat pateikiami su šiuo įrankiu atliktų tyrimų rezultatai, tyrimų išsami informacija ir gautos išvados.

1.2 Santrauka

Mobiliųjų telefonų programų šiuo metu yra sukurta labai daug ir jos yra kuriamos toliau. Tačiau didelė dalis šių programų nėra sėkmingos ir nėra 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.

Išanalizavus šiuo metu egzistuojančius pagrindinius grafinės vartotojo sąsajos testavimo metodus buvo pastebėta, kad viena iš pagrindinių testavimo metodų problemų yra didelės testavimo laiko

sąnaudos. Jos yra ypatingai didelės sudarant testus. Dėl šios priežasties buvo nuspręsta sukurti testavimo metodą, kuris išspręstų šią problemą. Naudojantis šiuo metodu buvo nuspręsta sukurti grafinės vartotojo sąsajos funkcionalumo testavimo įrankį, skirtą „Windows Phone 8.0“ operacinei sistemai.

Šio darbo pagrindinis tikslas yra pateikti testavimo metodą, kuris būtų pritaikytas mobilių įrenginių grafinės vartotojo sąsajos funkcionalumo testavimui. Taip pat šis metodas turi išspręsti problemą su didelėmis testavimo laiko sąnaudomis.

Šiame darbe yra sukurtas klaidų paieškos mobilių įrenginių programinėje įrangoje metodas, taikant programos vykdymo laikines charakteristikas. Metodas yra pagrįstas principu, jog nespėjus atlikti veiksmų per tam tikrą laiką, yra laikoma, kad testas nepavyko ir buvo aptiktos klaidos. Šis metodas yra pritaikytas mobiliems įrenginiams ir automatizuotam testavimui. Testavimo įrankis buvo realizuotas kaip „Microsoft Visual Studio“ programavimo aplinkos įskiepis.

Sukurto klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant programos vykdymo laikines charakteristikas, efektyvumas buvo įrodytas eksperimentiniu tyrimu. Eksperimentinio tyrimo metu buvo testuojamos 4 programos, skirtos „Windows Phone“ operacinei sistemai. Atlikus tyrimą, buvo gauta, kad testų sudarymo laiko sąnaudos vidutiniškai sutrumpėjo 7,6 karto, lyginant su funkciniu testavimo metodu. Taip pat buvo gauta, kad vidutinis sugeneruotų mutantų aptikimas yra 39,75 proc.

2 EGZISTUOJANČIŲ SPRENDIMŲ IR METODŲ ANALIZĖ

2.1 Temos tikslingumas ir aktualumas

Programinės įrangos vartotojo sąsajos testavimas yra procesas skirtas patikrinti ar veikia visos programos funkcijos. Kitaip dar sakoma, jog yra atliekamas testavimas ar programa yra funkciškai teisinga. Tai atliekama vykdant įvairias komandas ir lyginant gautą rezultatą su tikėtinu rezultatu. Visa tai yra kartojama daug kartų su skirtingais įėjimo duomenimis.

Ankščiau minėtas testavimas yra labai naudingas, tačiau viską atliekant rankiniu būdu yra labai brangu. 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].

Daugelis programinės įrangos kūrėjų siekia sumažinti programinės įrangos testavimo kaštus ir resursų panaudojimą. Dėl šios priežasties yra siekiama automatizuoti programinės įrangos testavimą [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.

Resursams labai imlus testavimas yra 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 [14,15], bet grafinės sąsajos automatizavimas labai stipriai sumažina testavimo kaštus [8].

2.2 Egzistuojantys sprendimai

Kaip įvade yra nurodyta, šiame darbe yra stengiamasi išanalizuoti sprendimus, kurie egzistuoja automatiniam testavimui ir pritaikyti „Windows Phone 8“ operacinės sistemos programoms. Iš to seka, kad šioje dalyje yra tikėtina analizuoti įrankius, skirtus testuoti programinės įrangos grafinę sąsają, skirtą „Windows Phone 8“ operaciniai sistemai. Tačiau įrankių, kurie dirbtų „Windows Phone 8“ platformoje nepavyko rasti. Taigi bus analizuojami atviro kodo įrankiai, kurie yra patys populiariausi šioje srityje.

Šioje srityje buvo rasti tokie populiariausi įrankiai, kurie toliau yra aprašomi smulkiau:

- „Abbot“,
- „Jacareto“,
- „Pounder“,
- „Marathon“,
- „JFCUnit“.

2.2.1 Abbot

„Abbot“ yra karkasas, skirtas testuoti programinės įrangos grafinei sąsajai. Jis yra naudojamas rašant grafinės sąsajos vienetų tesus, Java kalbos metodų pavidalu, kurie kviečia „Abbot“ karkasą, o šis savo ruoštu vykdo programos grafinę sąsają. Kaip jau ankščiau minėta, šio karkaso testai gali būti parašyti Java programavimo kalboje ir XML scenarijais. Taip pat yra pateikiamas įrankis „Costello“ XML scenarijams redaguoti. Be rankinio testų sudarymo, „Costello“ taip pat suteikia galimybę įrašyti scenarijus, vykdant komandas testuojamoje programoje [16].

2.2.2 Jacareto

„Jacareto“ yra grafinės sąsajos įrašymo ir atkartojimo įrankis, palaikantis animuotas demonstracijas, vartotojo elgsenos analizę ir taip pat grafinės sąsajos testavimą. Šis įrankis palaiko daug papildomų funkcijų, tokių kaip: tam tikro komponento pažymėjimas grafinėje sąsajoje, praplėtimo galimybė palaikyti specifinių semantinių įvykių įrašymą ir atkartojimą. Dar viena iš įdomesnių funkcijų – galimybė integruotis su kitomis programomis. „Jacareto“ yra prieinamas su dviem vartotojo sąsajomis – „CleverPHL“ ir „Picoder“ [17].

2.2.3 Pounder

„Pounder“ įrankis ypatingai orientuojasi į įrašymo ir atkartojimo veiksmus grafinės sąsajos testavime. Testų išsaugojimas yra vykdomas naudojantis XML scenarijų failais ir nėra skirtas redaguoti rankiniu būdu. Lyginant su „Abbot“ ir „Jacareto“ įrankiais, „Pounder“ yra nedidelis įrankis, kadangi jis neapima daug funkcijų [18].

2.2.4 Marathon

„Marathon“ yra žymiai galingesnio komercinio įrankio „MarathonITE“ atviro kodo versija. Šis įrankis, kaip ir kiti, palaiko interaktyvių sesijų įrašymą ir atkartojimą. Pateikiamas ir scenarijų redaktorius. Dar vienas pažymėtinas dalykas yra tai, kad interaktyvios sesijos yra saugomos kaip „Python“ scenarijai [19].

2.2.5 JFCUnit

„JFCUnit“ yra „JUnit“ testavimo karkaso praplėtimas grafinės vartotojo sąsajos testavimui. Kadangi šis įrankis yra veikiantis „JUnit“ karkaso pagrindu, jame testai yra rašomi tokiu pačiu principu kaip ir „JUnit“ testai. Pagrindinis šio įrankio dėmesys yra skiriamas rankiniam grafinės sąsajos testų kūrimui. Tačiau versijoje 2.0 buvo pridėta ir interaktyvių sesijų įrašymo galimybė [20].

2.2.6 Pastabos

Šie išanalizuoti testavimo įrankiai negali veikti „Windows Phone 8“ platformoje dėl architektūrinių sprendimų, t.y. jie veikia tik su Java programavimo kalba parašytomis programomis. Nepaisant to, jie yra geras palyginimas, kadangi jais galima remtis funkcijų atžvilgiu. Jie pateikia informaciją, kokios funkcijos turi būti realizuotos mūsų planuojamame kurti testavimo įrankyje. Kadangi šie įrankiai yra populiariausi šioje srityje, jų patirtimi mes galime pasiremti.

2.3 Testuojami laukai grafinėje sąsajoje

Dar viena problema kylanti testavimo metu yra, kokius laukus gali testuoti grafinės sąsajos testavimo įrankiai. „Windows Phone 8“ platformoje, kaip ir kitose platformose, testuojamų laukų yra daug. Kadangi nepavyko rasti atviro kodo įrankių, skirtų testuoti grafinę sąsają „Windows Phone 8“ platformoje, bus palyginama, kokius laukus gali testuoti anksčiau nagrinėti įrankiai, minėti skyriuje „2.2 Egzistuojantys sprendimai“. Palyginimo rezultatai yra pateikiami 1 lentelėje.

1 lentelė. Testuojamų laukų padengimas, nagrinėtų atviro kodo grafinės sąsajos testavimo įrankių

	Įrankis				
Laukai	**Abbot**	**Jacreto**	**Pounder**	**JFCUnit**	**Marathon**
Teksto laukas	●	●	●		
Pelės judesys (angl. *MouseMove*)	●	●	●		
Pelės tempimas (angl. *MouseDrag*)	●	●	●		
Pelės paspaudimas (angl. *MouseClick*)		●	●		

Komponentas	●	●	●		
Slinkimas (angl. *Scrolling*)	●	●	●		
Failo dialogas		●	●	●	
Pasirenkamo įvedimo laukas (angl. *ComboBox*)		●	●	●	●
Laikas (angl. *Timing*)					

2.4 Grafinės sąsajos testavimo algoritmų apžvalga

Šiame skyriuje bus apžvelgiami įvairūs metodai, skirti testuoti grafinę vartotojo sąsają:

- Modeliu grįstas testavimas (angl. – „*Model Based Testing"*).
- Kombinatorinis sąveikos testavimas (angl. – „*Combinatorial Interaction Testing"*).
- Grafine vartotojo sąsaja grįstos programos (angl. – „*Graphical User Interface (GUI)-based Applications"*).
- Našumo testavimas ir analizė (angl. – „*Performance Testing and Analysis"*).

2.4.1 Modeliu grįstas testavimas

Nauja grįžtamuoju ryšiu grįsta technika yra naudojama visiškai automatizuotuose procesuose, specifiniam grafinės sąsajos testavimui. „Dūmų" testai (angl. – „*smoke tests"*) yra generuojami automatiškai, naudojantis egzistuojančiu įvykių sąveikos grafinės vartotojo sąsajos grafo modeliu, kuris atspindi visas įmanomas įvykių sekas, kurios gali būti atliekamos grafinėje vartotojo sąsajoje [21]. Šis testavimo metodas naudojasi vykdymo metu prieinama informacija, kaip grįžtamuoju ryšiu, modeliu grįstam grafinės vartotojo sąsajos testų atvejų generavimui. Tačiau vykdymo metu prieinama informacija ankščiau buvo naudojama įvairiems testų automatizavimo aspektams ir modeliu grįstas testavimas buvo taikomas įprastinėms programinėms įrangoms, kaip ir įvykiais grįstai programinei įrangai.

Modelis yra programinės įrangos elgsenos abstrakcija iš tam tikros perspektyvos. Pavyzdžiui: programinės įrangos būsenos, konfigūracijos, kintamųjų reikšmių ir pan. Taip pat tai gali būti skirtingų lygių abstrakcija, tokių kaip abstrakčios būsenos, grafinės vartotojo sąsajos būsenos, vidinių kintamųjų būsenos, mašinos būsenos modeliai (angl. – „*state machine models"*). Labiausiai populiarus modelis, naudojamas programinės įrangos testavimui, yra mašinos būsenos modelis. Programinės įrangos elgsena yra modeliuojama naudojantis jos abstrakčiomis arba konkrečiomis būsenomis. Šios būsenos tipiškai yra atvaizduojamos kaip būsenos diagramos. Keletas tipų mašinos būsenos modelių yra naudojama programinės įrangos testavimui [21].

2.4.2 Kombinatorinis sąveikos testavimas

Kombinatorinis sąveikos testavimas yra testavimo metodas, kuris specializuojasi testų prioretizavimų technikose grafinėms vartotojo sąsajoms [22]. Šio testavimo specifinis įnašas apima: pirmąjį vieną modelį programoms, turinčioms grafinę vartotojo sąsają, ir internetinėms programoms, bendra prioretizavimo funkcija pagrįsta abstrakčiu modeliu ir bendrais prioretizavimo kriterijais. Atlikti tyrimai rodo, kad grafinę vartotojo sąsają turinčios programos, taip pat internetinės programos, naudojant šį testavimo modelį, parodė panašius rezultatus, kas leidžia šias abi programų klases modeliuoti ir tirti kartu. Tačiau kiti rezultatai rodo, jog grafinę vartotojo sąsają turinčios internetinės programos elgiasi skirtingai. Šie prieštaringi rezultatai įtakoja, jog modelis turi tobulėti ir turi būti atliekami tolimesni eksperimentai [23].

2.4.3 Grafine vartotojo sąsaja grįstos programos

Rankinis grafinės vartotojo sąsajos juodos dėžės (angl. – *black-box*) testavimas yra varginantis ir sunkus darbas, kadangi programų grafinė vartotojo sąsaja yra netriviali, turinti daug langų ir labai didelį kiekį komponentų. Testų automatizavimas yra pagrindinė priemonė sumažinti grafine vartotojų sąsaja grįstų programų testavimo išlaidas. Testuotojai, norėdami automatizuoti testavimo procesą, rašo programas įvairiomis scenarijų kalbomis (angl. – „*scripting languages"*), tokiomis kaip „JavaScript", „VBScript" ir pan. [24]. Vėliau šios parašytos programos (testavimo scenarijai) atkartoja vartotojo veiksmus grafinę vartotojo sąsają turinčiose programose, naudodamos testavimo karkasus. Šiuos testus užtrunka rašyti ilgiau nei vieną kartą atlikti vartotojo sąsajos testavimą, tačiau išlošimas labai stipriai pasijaučiau, kai tie patys testai yra vykdomi dažnai.

Scenarijais grįstas grafinės vartotojo sąsajos regresinis testavimas yra vykdomas sukonstruojant ir vėliau lyginant aukšto lygio grafinės vartotojo sąsajos modelius, prieš pritaikant algoritmus, kurie konstruoja testavimo atvejus grafine vartotojo sąsaja grįstoms programoms [25]. Grafinės vartotojo sąsajos modeliai taip pat gali būti gaunami iš grafinę vartotojo sąsają turinčios programos išeities kodo.

2.4.4 Našumo testavimas ir analizė

Labai praktiška yra automatiškai testuoti interaktyvių grafinę vartotojo sąsają turinčių programų našumą, kai yra išspręsti du uždaviniai:

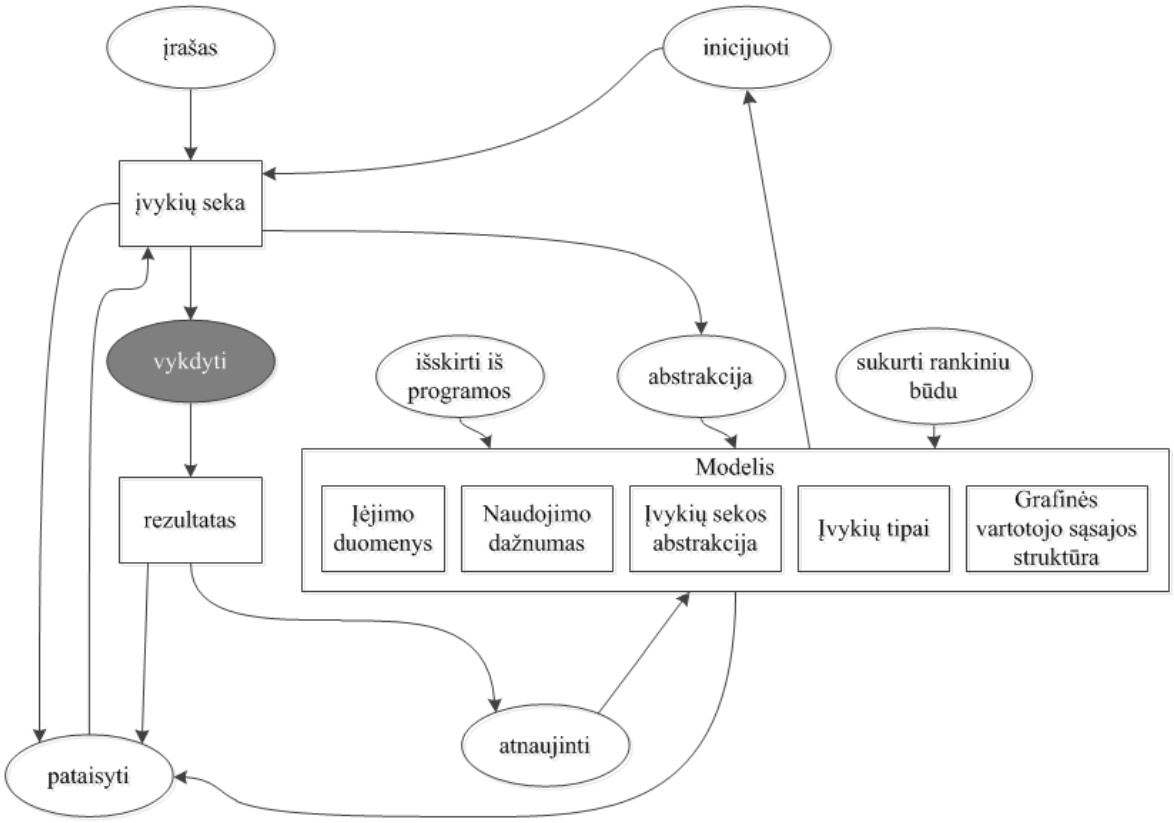
- Išvestas matuojamas įvertis, įvertinti interaktyvių programų našumą,
- Gautas būdas automatiškai atlikti realistines interaktyvias programų sesijas, neįtakojant programų našumo [11].

Grafinės vartotojo sąsajos našumo testavimas ir analizė turi du pagrindinius reikalavimus, kurių neapima tradiciniai grafinės vartotojo sąsajos testavimo metodai:

- Realistinių, ilgų ir sudėtingų interaktyvių vartotojo sesijų atkūrimas,
- Įrankiai, naudojami atlikti testavimui, negali įtakoti arba minimaliai įtakoti testuojamos programos vykdymo spartą, našumą.

Iš aukščiau minėtų problemų ir reikalavimų kyla išvada, kad norint atrasti našumo problemas realiose programose, atkartojamų įvykių sekų ilgis testavimo metu yra labai svarbus. Sekos, kurios atkartoja tik vieną ar kelis įvykius yra dažniausiai naudojamas funkciniam testavimui. Tokios sekos sudaro vienetų testus. Toliau, ilgesnės sekos, kuriose yra daugiau nei keli įvykiai, yra laikomi integraciniais testais, kadangi jos dažniausiai padengia sąveikas tarp kelių komponentų. Tačiau našumo problemų aptikimui sekos turi būti netgi labai žymiai ilgesnės negu integracinių testų sekos. Taip yra todėl, kad po grafine vartotojo sąsaja slypinčios sistemos galėtų pasiekti pusiausvyros būseną, kuri yra pasiekiama, kai programos yra naudojamos realiame pasaulyje. Toliau pateikiamame 1 pav. yra parodomas grafinės vartotojo sąsajos testavimo procesas.

Toliau pateikiamame paveikslėlyje naudojamos dviejų tipų viršūnės: veiklos (elipsės formos) ir duomenų rinkiniai (stačiakampio formos). Kiekvienas lankas jungia veiklą ir duomenų rinkinį. Kaip jau anksčiau minėta, pagrindinis automatizuoto grafinės vartotojo sąsajos testavimo tikslas yra pašalinti žmogaus įsikišimą testavimo metu. Kitaip pasakius, automatiškai vykdyti interaktyvią programą (vykdyti veiklą), atlikti nurodytą vartotojo veiksmų seką (įvykio sekos duomenų rinkinį).



1 pav. Grafinės vartotojo sąsajos testavimo procesas

Labiausiai paplitęs įrašymo ir atkartojimo būdas grafinei vartotojo sąsajai testuoti apima procesą, kas įrašo veiksmų seką, kol vartotojas sąveikauja su programa. Tačiau daugelis grafinės vartotojo sąsajos būdų naudoja modelius abstrahuoti konkrečias įvykių sekas. Modelis atspindi aibę visų galimų įvykių sekų. Turint modelį, galima konkretizuoti tam tikras įvykių sekas, kurios po to gali būti vykdomos. Modelis gali susidėti iš įvykių sekos abstrakcijos, tokios kaip: įvykių srautų grafų, Markovo modelio. Taip pat modelis gali turėti informaciją apie galimus įvykių tipus, grafinės vartotojo sąsajos struktūrą, kuri dažniausiai pateikiama medžio principu, kur aukštesnio lygio viršūnė langas, kuris turi komponentus, kurie atvaizduojami kaip šakos, žemesniame lygyje. Be jau paminėtų dalykų, modelis taip pat gali turėti panaudojimo dažnumus apie komponentus arba įvykius. Naudojant panaudojimo dažnumus galima nusakyti, kurie komponentai arba įvykiai yra svarbiausi praktikoje. Galiausiai modeliai gali atskirti įėjimo duomenis nuo įvykių sekos abstrakcijos. Įėjimo duomenys gali būti – aibė eilučių, kurios gali būti įrašomos į tekstinius laukus [26].

Modelis gali būti sukurtas ir prižiūrimas skirtingais būdais. Vienas būdas gali abstrahuoti modelį nuo vieno ar daugiau konkrečių įvykių sekų. Kitas būdas yra, kad modelis gali būti sukurtas rankiniu būdu, netgi kol dar nėra sukurta pati testuojama programa. Taip pat modelis arba jo dalys (pavyzdžiui, grafinės vartotojo sąsajos struktūra) gali būti ištraukta iš programos, naudojantis statinę arba dinaminę analizę [27]. Galiausiai, grįžtamuoju ryšiu pagrįstas būdas atnaujina modelį, pagal gautus rezultatus vykdant įvykių sekas. Rezultatai gali reaguoti į testo nesėkmes, programinės įrangos lūžimus vykdymo metu arba į gautus rezultatus iš dinaminės analizės atliktos taip pat vykdymo metu.

Rezultatai yra naudojami ne tik modeliu grįstiems būdams. Netgi tradiciniai įrašymo ir atkūrimo būdai gali naudoti rezultatus, gautus atkartojus seką tam, kad ją pataisyti. Dar daugiau, taisant konkrečią įvykių seką galima naudotis informacija iš modelio, pavyzdžiui: galima naudoti informaciją, kaip pasikeitė vartotojo sąsajos struktūra tarp programos versijų [28].

2.5 Testavimo algoritmų sulyginimas pagal kokybines charakteristikas

Žemiau esančioje 2 lentelėje yra pateikiamas išanalizuotų algoritmų palyginimas pagal jų funkcijas.

2 lentelė. Testavimo algoritmų palyginimas pagal galimas atlikti funkcijas

Grafinės vartotojo sąsajos testavimo algoritmai	**Modelis**							
	Įrašyti	**Įvykių seka**	**Atkartoti**	**Rezultatas**	**Vart. sąs. struktūra**	**Sukurti rankiniu būdu**	**Įėjimo duomenys**	**Panaudojimo dažnumas**
Modeliu grįstas testavimas	✓	✓	✓					
Kombinatorinis sąveikos testavimas	✓	✓	✓					
Grafine vartotojo sąsaja grįstos programos	✓	✓	✓			✓		✓
Našumo testavimas ir analizė	✓	✓	✓	✓	✓	✓	✓	✓

2.6 Naujo testavimo metodo reikalingumas

2 lentelė atspindi, jog metodas „Našumo testavimas ir analizė“, pagal teikiamas funkcijas, yra tinkamas mobilių įrenginių programinės įrangos klaidų paieškai. Tačiau atlikus analizę, kuri pateikiama skyriuje 2.4.4 „Našumo testavimas ir analizė“ nustatyta, jog šiam metodui sudaryti testus yra didelių laiko sąnaudų reikalaujantis procesas. Norint sudaryti testus naudojantis šiuo metodu yra reikalingi įvairūs įėjimo duomenys ir reikalingi išėjimo duomenys, gautiems rezultatams palyginti, ar teisingai įvykdyta funkcija.

Dėl aukščiau pateiktų priežasčių yra reikalingas testavimo metodas, kuris tiktų automatizuotam testavimui ir testo sudarymui nebūtų reikalingos didelės laiko sąnaudos.

2.7 „Windows Phone 8“ operacinės sistemos apžvalga

Šioje dalyje bus apžvelgiama „Windows Phone 8“ operacinė sistema, apibendrintos programavimo ir testavimo galimybės.

2.7.1 Apibendrintos programavimo galimybės

Norint programuoti operacinės sistemos „Windows Phone 8“ aplinkoje, yra reikalingas „Windows Phone 8 sdk“ – programavimo įrankių rinkinys. Šį rinkinį sudaro integruota programavimo aplinka (angl. – „*IDE*“) ir programavimo sąsaja (angl. – „*API*“). Programavimo sąsają sudaro 3 komponentai [29]:

- .NET programavimo sąsajos,
- „Windows Phone Runtime“,
- „Direct3D, XAudio2, MF, WASAPI, Win32, COM“.

Pirmajame komponente .NET programavimo sąsajos, galima programuoti valdomomis (angl. – „*managed*“) programavimo kalbomis – C# ir VB.NET. Antrajame komponente „Windows Phone Runtime“ galima programuoti ir valdoma programavimo kalba, ir „gimtąja“ (angl. – „*native*“) programavimo kalba – C++.

Integruota programavimo aplinka yra „Microsoft Visual Studio 2012“. Joje yra pateikiamos funkcijos, reikalingos rašyti išeities kodą, sukompiliuoti jį ir įdiegti sukompiliuotą programinę įrangą į mobilųjį telefoną.

2.7.2 Apibendrintos testavimo galimybės

Testavime yra naudojamas priėjimas prie programos klasių, duomenų ir įvykių. Šias galimybes realizuoja .NET programavimo sąsajos vardų sritis „System.Reflection“.

Kita galimybė analizuoti programos elgseną yra fotografuoti programos vartotojo sąsają ir lyginti pasikeitimus. Ši galimybė nėra tiesiogiai prieinama, ją reikia pačiam pasitobulinti iki sau tinkamos, arba naudoti trečių šalių sukurtus įrankius [30].

2.8 Apibendrinimas

1. Išmaniųjų telefonų programinė įranga turi būti kokybiška, kadangi vartotojai nėra linkę naudotis programine įranga, kuri veikia neteisingai.
2. „Windows Phone 8“ programinei įrangai skirtų grafinės vartotojo sąsajos funkcionalumo testavimo sprendimų rasti nepavyko.
3. Išanalizuoti testavimo metodai funkciniu požiūriu patenkina iškeltus testavimo metodui reikalavimus, tačiau netenkina testavimo laiko sąnaudų reikalavimo. Su išanalizuotais testavimo metodais yra reikalingos nemažos laiko sąnaudos testų sukūrimui ir jų vykdymui.
4. Pasirinktas sprendimas yra kurti testavimo metodą, kuris tenkintų reikalavimus funkciniu požiūriu, nereikalautų didelių laiko sąnaudų testų kūrimui ir būtų pritaikytas automatizuotam testavimui.

3 „WINDOWSPHONEGUITESTTOOL“ ĮRANKIO PROJEKTAS

3.1 Sistemos paskirtis

Šios sistemos pagrindinis tikslas yra palengvinti „Windows Phone OS“ programų grafinės vartotojo sąsajos testavimą programų kūrėjams ir jų testuotojams. Sukurtas įrankis turi būti intuityvus naudotis, lengvai suprantamas ir efektyvus. Šis įrankis skirtas naudoti asmenims, kurie kuria „Windows Phone OS“ programas.

Ši sistema bus gebanti atlikti žemiau pateikiamas funkcijas:

- Rankinis testų sudarymas.
- Sudarytų testų vykdymas.
- Sudarytų testų redagavimas.
- Rezultatų peržiūra.
- Rezultatų palyginimas.

3.2 Sistemos funkcijos

Įrankio teikiamos funkcijos:

- Rankinis testų sudarymas – sudaromi testai, kurie bus vykdomi testavimo metu.
- Sudarytų testų vykdymas – vykdomi testai emuliatoriuje arba išmaniajame telefone.
- Sudarytų testų redagavimas – jau sudarytų testų redagavimas, t.y. egzistuojančių testų modifikavimas.
- Rezultatų peržiūra – peržiūrimi įvykdytų testų rezultatai.
- Rezultatų palyginimas – palyginami dviejų vykdytų testų rezultatai tarpusavyje.

3.3 Funkciniai reikalavimai sistemai

Žemiau yra pateikiami funkciniai reikalavimai sistemai:

- Sistema nurodo, koks pasirinktas projektas.
- Sistema priskiria pasirinktam projektui ypatybes pakeitus konfigūraciją.
- Sistema įvykdo nurodytą testą.
- Sistema automatiškai atlieka konfigūraciją pagal nurodytą įvykdymui testą.
- Sistema įvertina, ar testas įvyko sėkmingai, įvykdžius visus testo žingsnius.
- Sistema leidžia pasirinkti tik vieną seniau vykdytą testavimo scenarijų.
- Sistema leidžia įvykdyti seniau vykdytą testą.
- Sistema leidžia pasirinkti tik vieną testą redagavimui.
- Sistema turi patikrinti, ar redaguotas testavimo scenarijus yra sintaksiškai teisingas.
- Sistema turi leisti peržiūrėti testo rezultatus iš karto po jo įvykdymo.
- Sistema turi leisti peržiūrėti seniau vykdytų testų rezultatus.
- Sistema turi lyginti tik dviejų testų rezultatus.

3.4 Nefunkciniai reikalavimai sistemai

Žemiau yra pateikiami nefunkciniai reikalavimai sistemai:

- Intuityvi vartotojo sąsaja.
- Vartotojo sąsaja neišsiskirianti iš „Microsoft Visual Studio 2012“ vartotojo sąsajos.
- Meniu turi būti nesudėtingas.
- Neįkyri vartotojo sąsaja.
- Sistema turi nuolatos pranešti, koks yra vykdomos užduoties statusas.
- Nereikalingas apmokymas naudotis sistema.
- Galimybė nutraukti komandų vykdymą.
- Sistema turi turėti integruotą pagalbą, kaip naudotis.
- Resursai turi būti panaudojami efektyviai.
- Mažas užduočių vykdymo laikas.
- Patikimumas.

- Galimybė veikti be išmaniojo telefono, tik su emuliatoriumi.
- Naujų „Microsoft Visual Studio“ versijų palaikymas.
- Naujų „Microsoft Windows Phone“ operacinių sistemų versijų palaikymas.
- Sistemoje esantys duomenys turi būti apsaugoti nuo neteisėtos prieigos.
- Duomenys perduodami tarp išmaniojo telefono/emuliatoriaus neturi būti pažeisti iš išorės.
- Turi būti galimybė ištrinti testų rezultatus.
- Turi būti galimybė ištrinti testus.
- Sistemoje neturi būti naudojami įžeidžiantys terminai arba iliustracijos.
- Sistema privalės būti platinama kaip atvirojo kodo programa, pagal LGPL 2.1 versiją.

3.5 Architektūra

3.5.1 Paskirtis

Architektūros specifikacija apžvelgia kuriamos programinės įrangos architektūrą. Šiame skyriuje yra specifikuojami architektūriniai sprendimai, kurie bus panaudoti kuriant programinę įrangą. Taip pat specifikacija yra reikalinga komunikacijoms tarp kūrėjo, užsakovo ir kitų šios programinės įrangos kūrime ir pristatyme dalyvaujančių žmonių.

Šiame skyriuje kūrėjas specifikuoja programinės įrangos galimybes ir jos kūrimo ypatumus. Visą šią informaciją matantys kiti suinteresuoti žmonės galės prieš pradedant kurti sistemą identifikuoti jiems netinkančias programos vietas

3.5.2 Architektūros pateikimas

Architektūra yra pateikiama naudojantis toliau pateikiamais vaizdais:

- Panaudojimo atvejų vaizdas – pateikiamas naudojant panaudos atvejų diagramą.
- Sistemos statinis vaizdas – pateikiamas naudojant paketų ir klasių diagramas.
- Sistemos dinaminis vaizdas – pateikiamas naudojant veiklos diagramas.
- Duomenų vaizdas – pateikiama naudojant ER diagramą.

3.5.3 Architektūros tikslai ir apribojimai

Sistema bus kuriama naudojantis „Microsoft Visual Studio 2012“ programavimo aplinka, C# programavimo kalba ir „Microsoft Windows Phone SDK 8.0“. Visos šios naudojamos technologijos veikia tik „Windows 8 Professional“ ir aukštesnės versijos 64 bitų operacinėje sistemoje, su įdiegtu „Windows Phone 8.0“ emuliatoriumi.

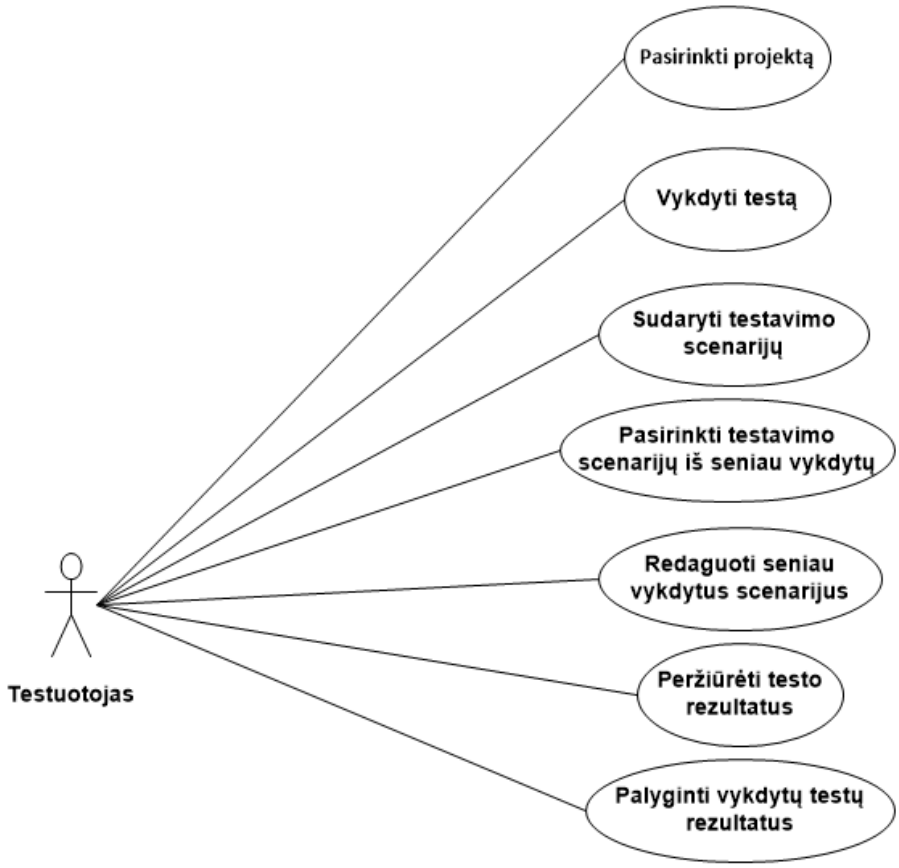
Taip pat ši programinė įranga privalės veikti ir su išmaniuoju telefonu, kuris turi „Windows Phone 8“ operacinę sistemą.

Nepaisant aukščiau išvardintų konkrečių sistemų versijų, ši programinė įranga bus kuriama taip, kad ją būtų nesunku perkelti į naujesnes sistemas. Pavyzdžiui: atsiradus naujai „Windows Phone“ operacinei sistemai ar atsiradus naujai programavimo aplinkai „Microsoft Visual Studio“.

Taip pažymėtina, jog ši programinė įranga bus įskiepis „Microsoft Visual Studio“ aplinkai. Dėl šios priežasties yra pasiekiamas mažas portabilumas. Taip pat portabilumą mažina ir tas faktas, kad šiai sistemai veikti bus reikalinga ir „Microsoft Windows Phone 8.0 SDK“.

3.5.4 Panaudojimo atvejų vaizdas

Panaudojimo atvejų diagrama yra pateikiama 2 pav.



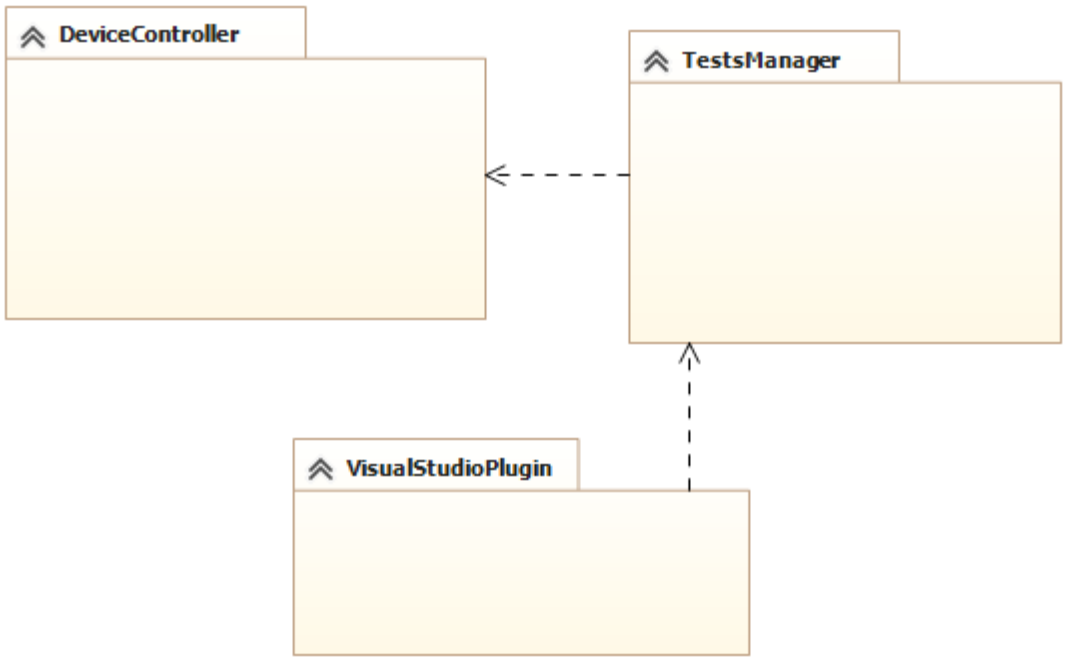
2 pav. Panaudojimo atvejų diagrama

3.5.5 Sistemos statinis vaizdas

Šiame skyriuje yra pateikiama sistemos išskaidymas į paketus ir juos sudarančias klases.

3.5.5.1 Sistemos išskaidymas į paketus

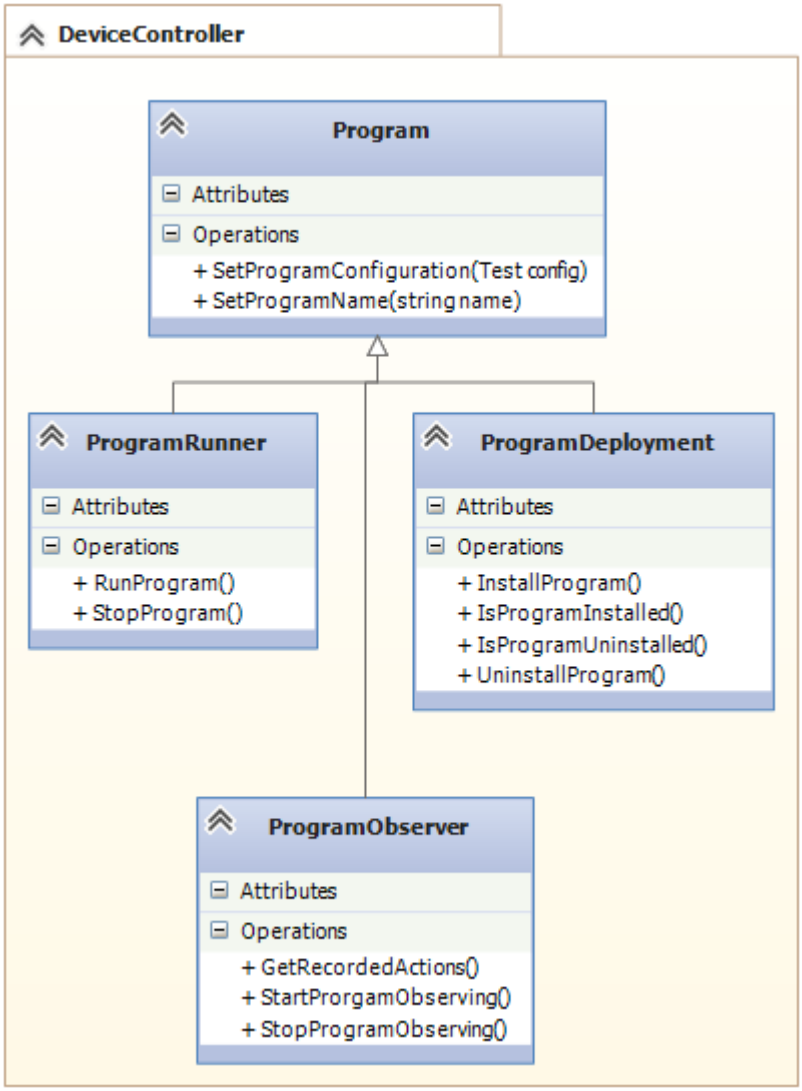
Sistemos išskaidymo į paketus diagrama yra pateikiama 3 pav.



3 pav. Sistemos išskaidymas į paketus

3.5.5.1.1 „DeviceController“ paketas

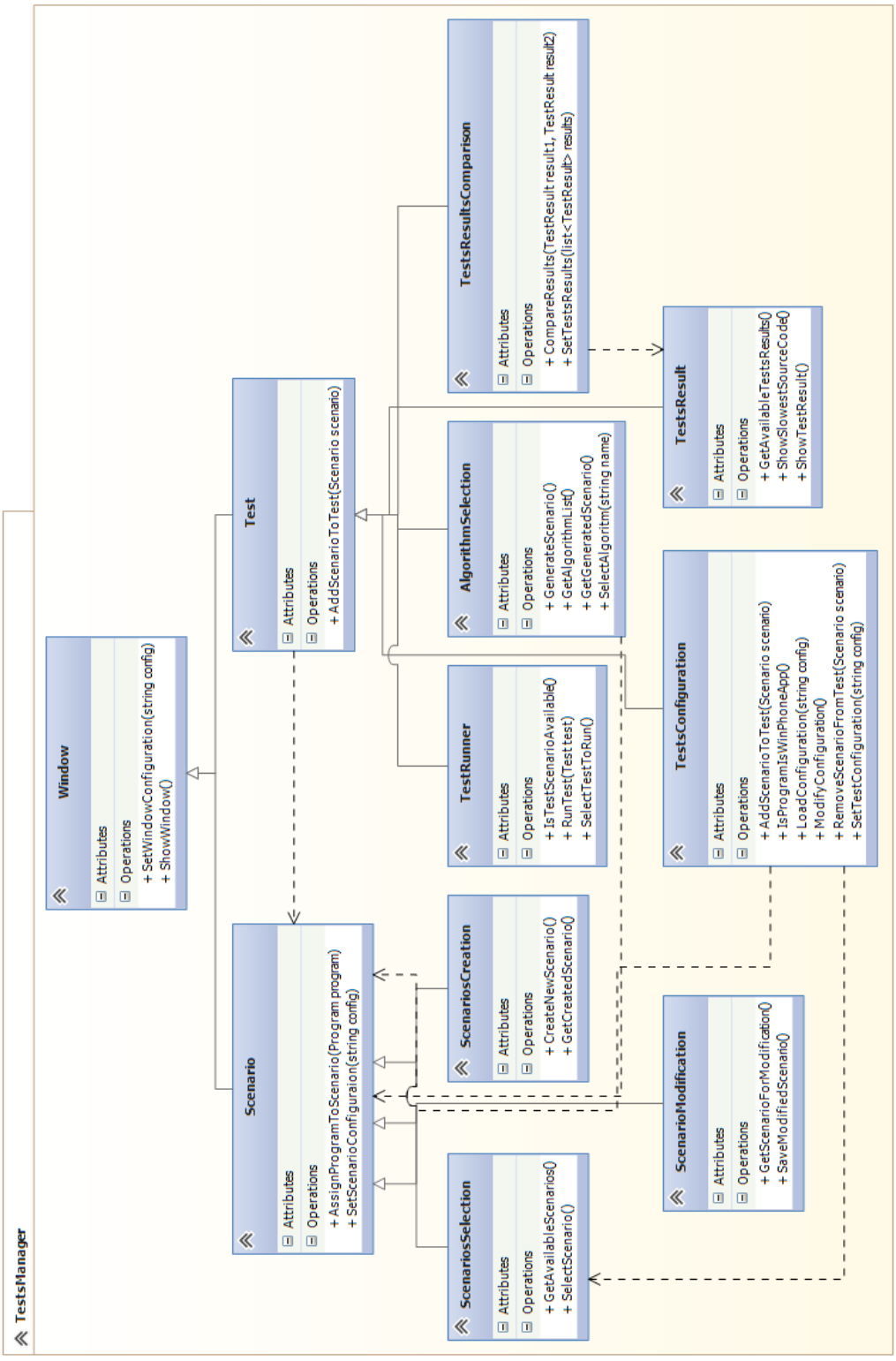
Šis paketas yra atsakingas už testuojamos programos vykdymą, įdiegimą ir stebėjimą emuliatoriuje arba išmaniajame telefone. Paketo „DeviceController“ klasių diagrama yra pateikiama 4 pav.



4 pav. „DeviceController“ paketo klasių diagrama

3.5.5.1.2 „TestsManager“ paketas

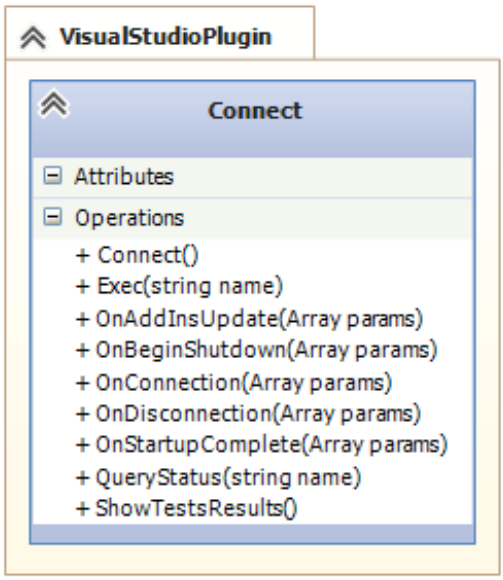
Šis paketas yra atsakingas už testų vykdymą, kūrimą ir modifikavimą, testų rezultatų palyginimą ir peržiūrėjimą. Taip pat šis paketas atsakingas už grafinę vartotojo dalį. Paketo „TestsManager“ klasių diagrama yra pateikiama 5 pav.



5 pav. „TestsManager“ paketo klasių diagrama

3.5.5.1.3 „VisualStudioPlugin“ paketas

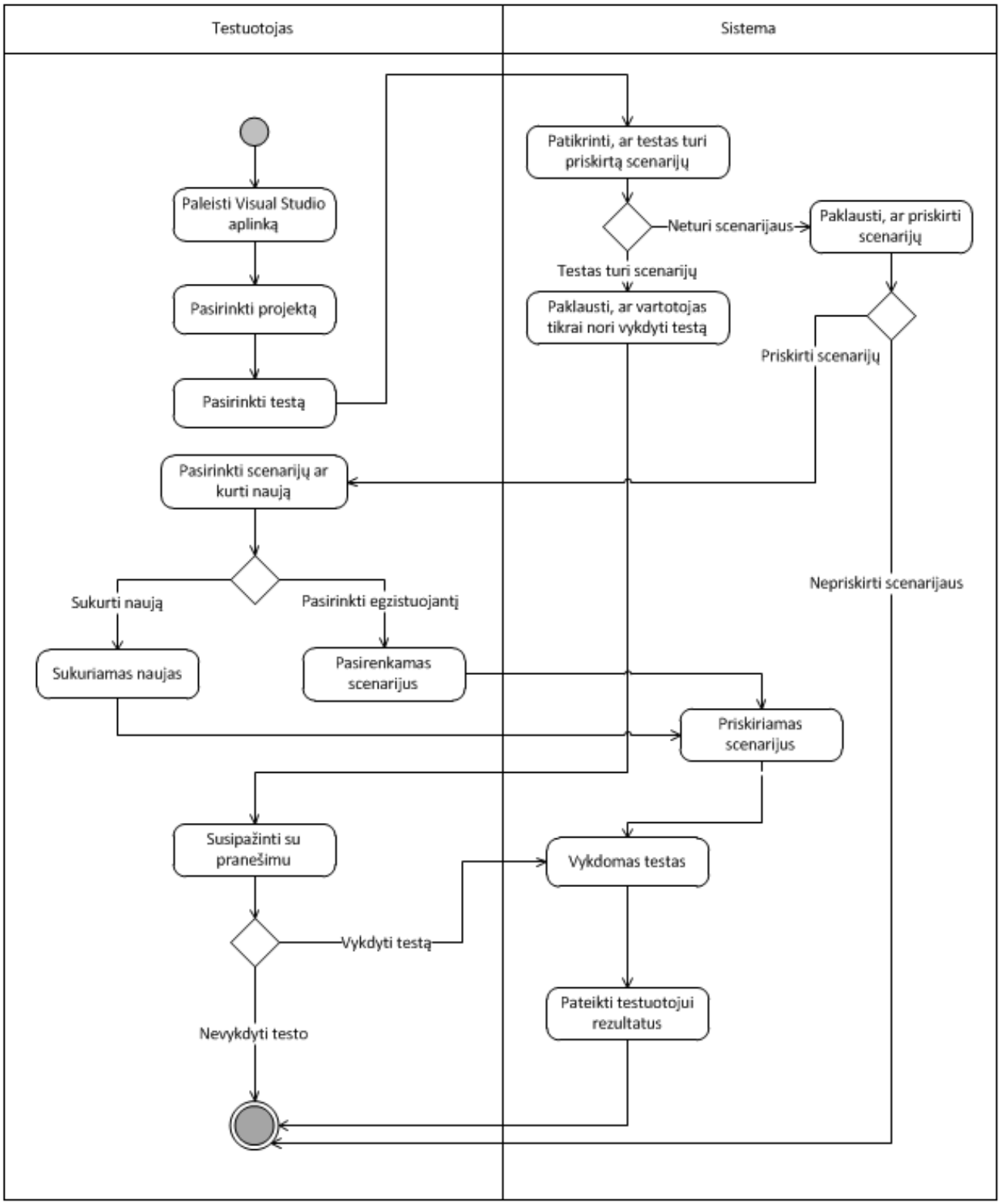
Paketas skirtas sąsajai su „Microsoft Visual Studio“ aplinka. Paketo klasių diagrama yra pateikiama 6 pav.



6 pav. „VisualStudioPlugin“ paketo klasių diagrama

3.5.6 Sistemos dinaminis vaizdas

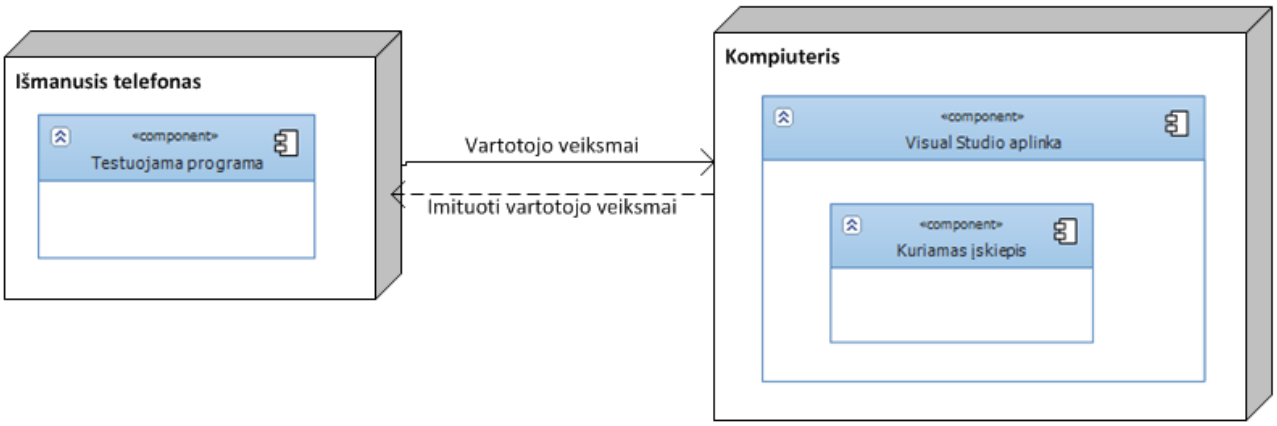
7 pav. yra pateikiama veiklos diagrama, kaip sistemos vartotojas vykdo testą. Klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant programos vykdymo laikines charakteristikas, veiklos diagrama yra pateikiama 11 pav., kuri yra skyriuje 4 „Klaidų paieškos mobilių įrenginių programinėje įrangoje metodas, taikant programos vykdymo laikines charakteristikas“.



7 pav. Vykdyti testą – veiklos diagrama

3.5.7 Išdėstymo vaizdas

Išdėstymo diagrama yra pateikiama 8 pav.



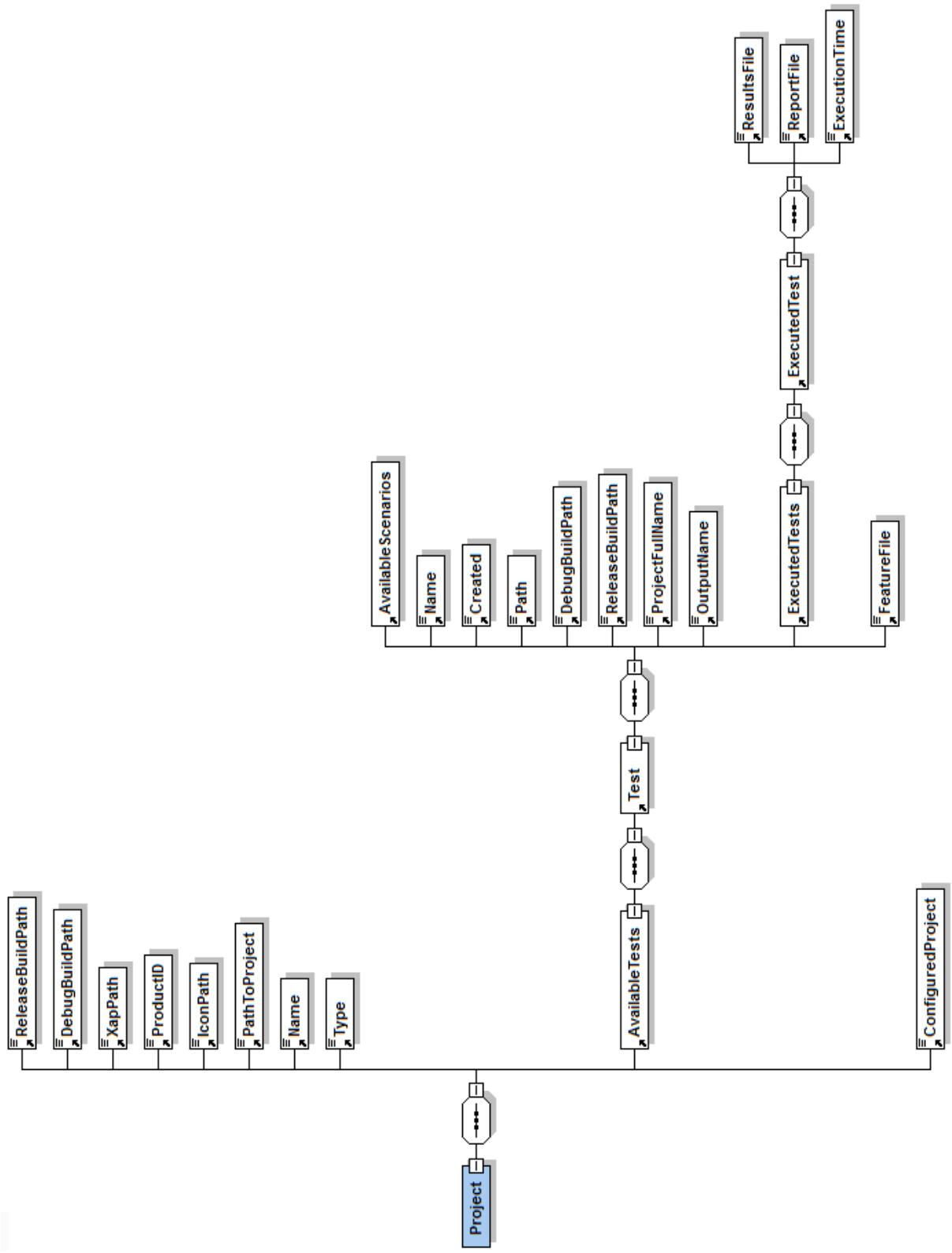
8 pav. Išdėstymo diagrama

Aukščiau pateiktoje diagramoje matoma, jog sistemai funkcionuoti yra reikalingi du techniniai mazgai: išmanusis telefonas ir kompiuteris. Toliau yra pateikiamas jų aprašymas:

- **Išmanusis telefonas** - išmaniajame telefone veikia testuojama programa. Taip pat pažymėtina, kad išmanusis telefonas gali būti pakeistas emuliatoriumi.
- **Kompiuteris** - kompiuteryje veikia mūsų kuriama sistema. Kuriama sistema integruojasi su „Microsoft Visual Studio“ aplinka.

3.5.8 Duomenų vaizdas

Duomenų vaizdas yra pateikiamas 9 pav.



9 pav. Duomenų vaizdo diagrama

Toliau yra pateikiami paaiškinimai kiekvienam laukui:

- „Project“ – sukurtas testo projektas.
  - „ReleaseBuildPath“ – kelias į „Release“ katalogą.
  - „DebugBuildPath“ – kelias į „Debug“ katalogą.
  - „XapPath“ – kelias į programos „.xap“ failą.
  - „ProductID“ – programos produkto identifikacinis numeris.
  - „IconPath“ – programos paveiksliukas.
  - „PathToProject“ – kelias į projekto katalogą.

- „Name“ – projekto pavadinimas.
- „Type“ – projekto tipas.
- „ConfiguredProject“ – indikacija ar projektas jau yra sukonfigūruotas.
- „AvailableTests“ – projektui sukurti testai.
  - „Test“ – testuojamos programos testas.
    - „AvailableScenarios“ – galimi scenarijai.
    - „Name“ – testo pavadinimas.
    - „Created“ – indikacija ar testas jau sukurtas.
    - „Path“ – kelias į sukurto testo katalogą.
    - „DebugBuildPath“ – kelias į „Debug“ katalogą.
    - „ReleaseBuildPath“ – kelias į „Release“ katalogą.
    - „ProjectFullName“ – pilnas testo projekto pavadinimas.
    - „OutputName“ – testo išeities failas.
    - „ExecutedTests“ – įvykdytų testų rezultatai.
      - „ResultsFile“ – rezultatų failas.
      - „ReportFile“ – raporto failas.
      - „ExecutionTime“ – laikas, kada testas buvo įvykdytas.
    - „FeatureFile“ – testo scenarijaus aprašymas.

3.5.9 Kokybė

Pateikta programos architektūra įgalina nesunkiai pereiti prie naujesnių „Microsoft Visual Studio“ versijų, kadangi pasikeitimus reikės padaryti tik bazinėse klasėse. Logika nuo to nepriklauso. Visa tai galioja ir „Microsoft Windows Phone SDK“ versijos atsinaujinimo atveju. Pakeitimai bus reikalingi tik bazinėse klasėse.

Taip pat visi naudojami grafinės sąsajos komponentai yra „Microsoft Visual Studio“ dalis. Tai padidins naudojimosi paprastumą.

3.6 Sistemos testavimas

Šiame skyriuje yra pateikiamas apibendrintas testavimo planas, aprašomos testavimo procedūros bei pateikiami testavimo rezultatai.

3.6.1 Vienetų testavimas

Vykdant vienetų testavimą bus tikrinamas atskirų sistemos vienetų veikimas. Sistemos vienetai, kurie bus testuojami yra:

- Klasės,
- Metodai.

Vienetų testavimas yra vykdomas paduodant tam tikrus įėjimo duomenis ir gavus išėjimo duomenis, juos lyginant su iš anksto žinomais teisingais išėjmo duomenimis. Taip pat vienetų testavimas bus vykdomas dviem būdais:

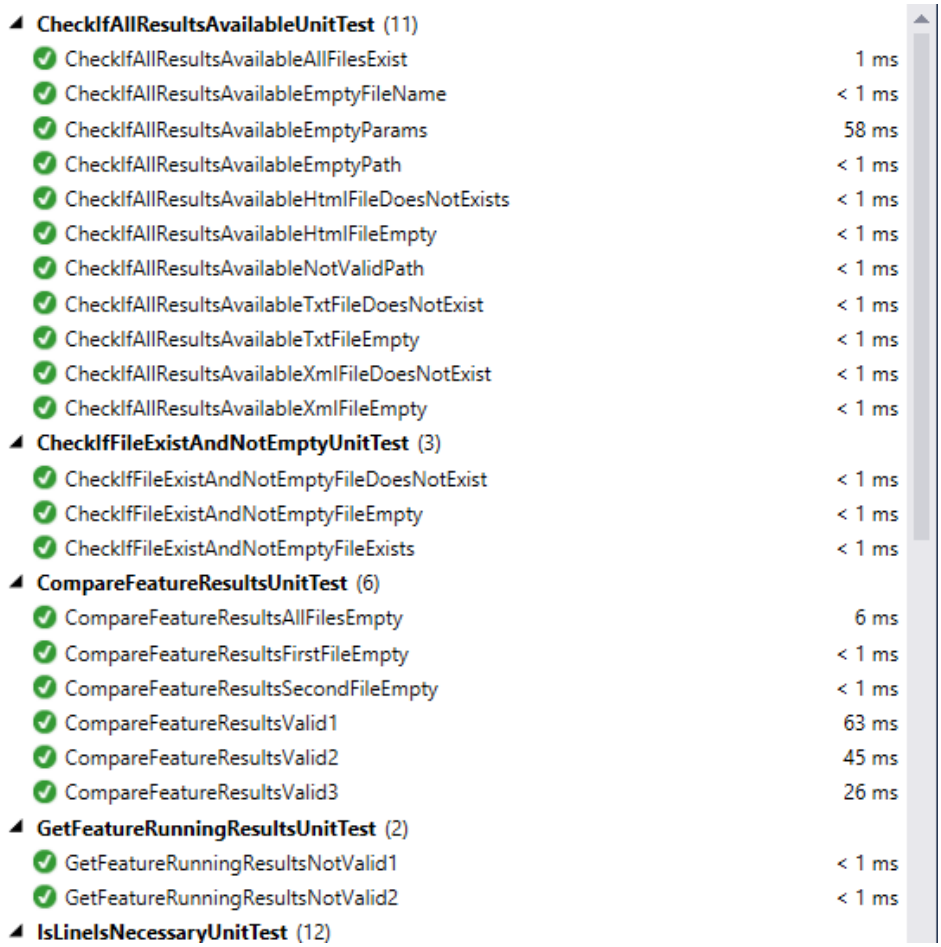
- Baltos dėžės testavimas,
- Juodos dėžės testavimas.

Baltos dėžės testavimas yra vykdomas atsižvelgiant į testuojamo vieneto vidinę struktūrą. Vykdant šį testavimą yra stengiamasi padengti kuo didesnę vieneto dalį, t.y. pereiti kuo daugiau kodo eilučių.

Juodos dėžės testavimas yra vykdomas atsižvelgiant tik į programinės įrangos specifikaciją. Įėjimo ir išėjimo duomenys yra parenkami tik pagal specifikaciją, neatsižvelgiant į tai, kaip yra realizuotas vienetas.

3.6.1.1 Vienetų testavimo rezultatai

Atliekant vienetų testavimą buvo atrasta įvairių klaidų. Visos klaidos yra pataisytos. Sėkmingai įvykdytų testų rezultatai yra pateikiami 10 pav.



10 pav. Testų rezultatų langas

3.6.2 Integracinis testavimas

Integravimo testavimo metu bus testuojami apjungtų vienetų tarpusavio sąveika. Norint vykdyti šį testavimą su apjungtais vienetais, turi būti išpildyta sąlyga – vienetai turi būti sėkmingai įvykdę vienetų testavimą.

Vykdant šį testavimą bus naudojamas principas „Iš apačios į viršų“ (angl. - „Bottom-Up“). Naudojantis šiuo principu iš pradžių yra testuojami patys mažiausi sistemos komponentai arba moduliai. Po to jie yra apjungiami į stambesnius modulius ar komponentus. Kitame žingsnyje apjungti moduliai yra vėl apjungiami ir taip yra daroma tol, kol gaunama visa sistema.

3.6.2.1 Integracinio testavimo rezultatai

Integracinis testavimas buvo atliktas pagal aukščiau pateikiamą aprašymą. Testavimo rezultatai yra pateikiami 3 lentelėje.

3 lentelė. Integracinio testavimo rezultatai

Testavimo lygis	Rezultatai
1 lygis	Testavimas sėkmingas, rastos klaidos ištaisytos
2 lygis	Testavimas sėkmingas, rastos klaidos ištaisytos
3 lygis	Testavimas sėkmingas, rastos klaidos ištaisytos
4 lygis	Testavimas sėkmingas, rastos klaidos ištaisytos

3.6.3 Priėmimo testavimas

Priėmimo testavimas bus vykdomas tik sėkmingai užbaigus prieš tai vykdytą testavimą – integravimo testavimą.

Šio testavimo metu bus tikrinama, ar sukurta programinė įranga atitinka specifikaciją ir užsakovo reikalavimus. Vykdant testavimą aptikus neatitikimus tarp specifikacijos ir testuojamos programos,

bus registruojama klaida. Šią klaidą reikės ištaisyti. Jeigu vykdant testavimą, iškils vartotojo poreikis naujiems patobulinimams, kurių nėra specifikacijoje, šis patobulinimas taip pat bus registruojamas. Tačiau patobulinimai nebus registruojami kaip klaidos. Jie bus registruojami atskirai ir jų įgyvendinimas bus vykdomas tik kuriant naują sistemos versiją.

3.6.3.1 Priėmimo testavimo rezultatai

Priėmimo testavimo rezultatai yra pateikiami 4 lent.

4 lentelė. Priėmimo testavimo rezultatai

Panaudojimo atvejis	**Rezultatai**
Pasirinkti projektą	Panaudojimo atvejis veikia tinkamai
Vykdyti testą	Panaudojimo atvejis veikia tinkamai
Sudaryti testavimo scenarijų	Panaudojimo atvejis veikia tinkamai
Pasirinkti testavimo scenarijų iš seniau vykdytų	Panaudojimo atvejis veikia tinkamai
Redaguoti seniau vykdytus scenarijus	Panaudojimo atvejis veikia tinkamai
Peržiūrėti testo rezultatus	Panaudojimo atvejis veikia tinkamai
Palyginti vykdytų testų rezultatus	Panaudojimo atvejis veikia tinkamai

3.7 Apibendrinimas

1. Suprojektuotas, realizuotas ir ištestuotas „Windows Phone OS“ programų automatinio testavimo įrankis.
2. „Windows Phone OS“ programų automatinio testavimo įrankis skirtas „Microsoft Visual Studio“ programavimo aplinkai.

4 KLAIDŲ PAIEŠKOS MOBILIŲ ĮRENGINIŲ PROGRAMINĖJE ĮRANGOJE METODAS, TAIKANT PROGRAMOS VYKDYMO LAIKINES CHARAKTERISTIKAS

4.1 Įžanga

Šiame skyriuje yra aprašomas klaidų paieškos metodas, skirtas mobiliems įrenginiams. Aprašytas metodas leidžia išspręsti testavimo laiko sąnaudų problemą, kuri aprašyta skyriuje 1 „Įvadas“. Siūlomo metodo tikslas yra sumažinti testų kūrimo laiko sąnaudas. Taip pat metodas yra pritaikytas automatizuotam testavimui. Visos šios metodo savybės leis greičiau ir pigiau vykdyti testavimą.

Klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant programos vykdymo laikines charakteristikas, veikimas yra pagrįstas grafinės vartotojo sąsajos testavimu. Visos testuojamos programos funkcijos yra inicijuojamos iš programinės įrangos grafinės vartotojo sąsajos. Visas funkcijų inicijavimas ir jų rezultatų laukimas yra vykdomas kartu matuojant kiekvienos atliktos funkcijos įvykdymo laiką.

4.2 Grafinės vartotojo sąsajos operacijos

Toliau yra pateikiamos metodo palaikomos grafinės vartotojo sąsajos operacijos:

- Mygtuko paspaudimas,
- Teksto įvedimas,
- Teksto atsiradimas,
- Pranešimo atsiradimas,
- Mygtukų atsiradimas,
- Veiksmai su lietimui jautriu ekranu:
    - Braukimas į kairę/dešinę,
    - Braukimas aukštyn/žemyn,
    - Ekrano aktyvių zonų paspaudimas,
    - Kiti veiksmai, kurie yra priklausomi nuo konkretaus telefono modelio (kelių taškų paspaudimas ekrane vienu metu ir pan.).

4.3 Reikalavimai metodui įvykdyti

Norint pasinaudoti šiuo metodu yra reikalinga išpildyti tokias sąlygas:

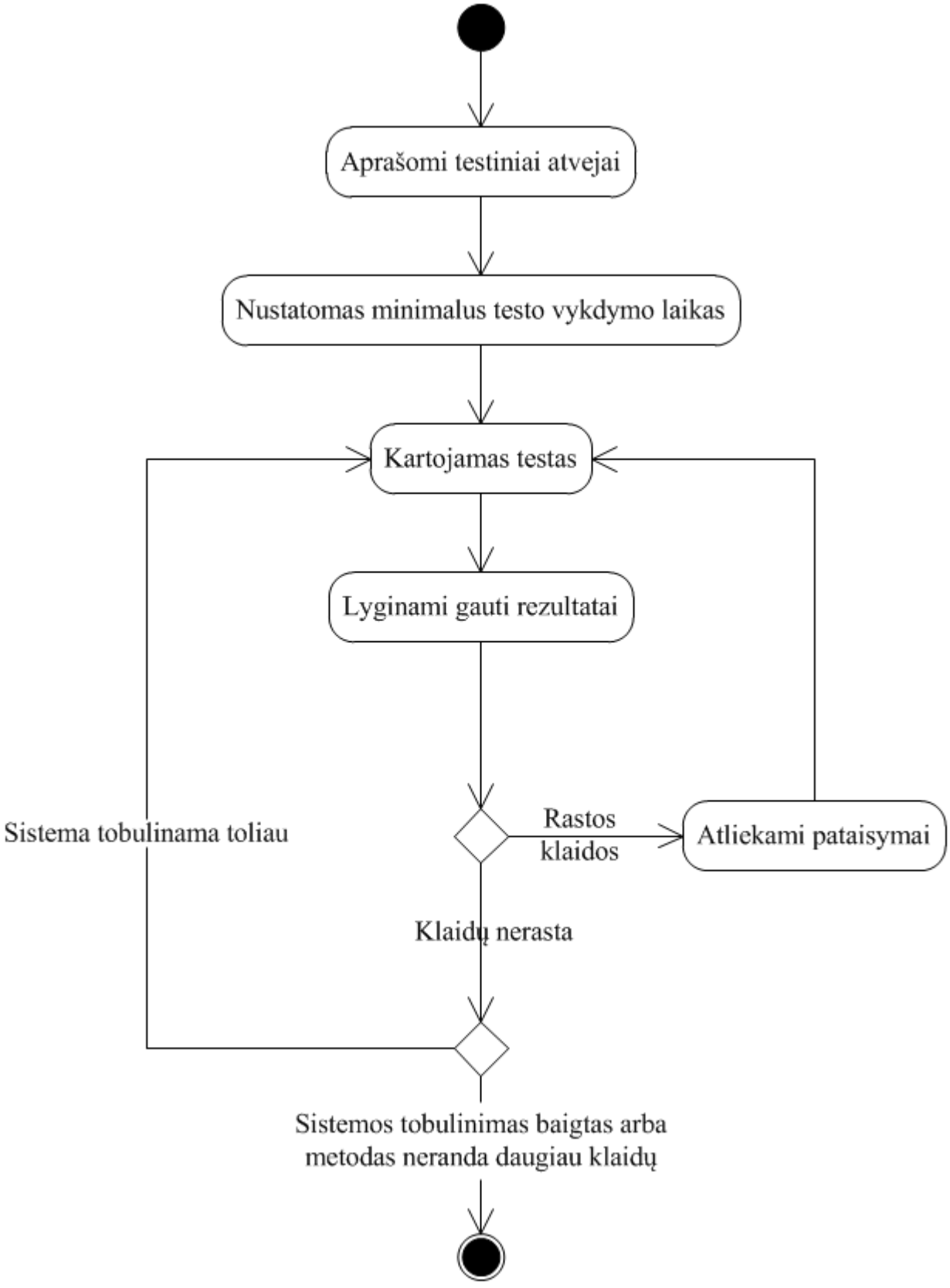
- Testuojama programinė įranga turi būti kuriama mobiliems įrenginiams.
- Testuojama programinė įranga turi turėti grafinę vartotojo sąsają.
- Turi būti prieinamas testuojamos programinės įrangos išeities kodas.

4.4 Metodo veikimas

Toliau 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.

Toliau yra pateikiama klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant programos vykdymo laikines charakteristikas veiklos diagrama.



11 pav. Metodo veiklos diagrama

4.5 Testavimo pabaigos sąlyga

Testavimas pasinaudojant šiuo metodu yra baigiamas tuomet, kai testuojama programinė įranga yra sukurta pilnai ir gaunami rezultatai yra mažesni arba lygūs fiksuotiems rezultatams. Taip pat testavimas turi būti baigiamas, kai metodas neatranda klaidų.

4.6 Metodo pritaikymas

Klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant programos vykdymo laikines charakteristikas, yra taikytas naudoti mobiliųjų telefonų programoms, kurios yra kurtos šioms operacinėms sistemoms:

- „Microsoft Windows Phone“,
- „Google Android“,
- „Apple iOS“.

Aukščiau išvardintos operacinės sistemos yra labiausiai paplitę, todėl 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 skyriuje 4.3 „Reikalavimai metodui įvykdyti“.

4.7 Metodo panaudojimo pavyzdys

Šiame skyriuje yra pateikiamas pavyzdys, kaip būtų galima pasinaudoti šiuo metodu. Pavyzdyje yra pateikiama programa, kuri įvestą tekstą išveda priešinga kryptimi. Taip pat ši programa pasileidžiant 30 sekundžių rodo užrašą „Opening“. Naudojantis pavyzdine programa bus pereitas visas testavimo metodas.

4.7.1 Aprašomi testiniai atvejai

Šiuo žingsniu yra aprašomi testiniai atvejai. Juose yra tiesiog nurodoma, kokias funkcijas iškviesti ir laukiama jų rezultatų. Į rezultatus yra nekreipiama dėmesio, tiesiog yra svarbus faktas, kad buvo grąžinti rezultatai arba ne.

Toliau yra pateikiamas vienas testavimo atvejis, skirtas testuoti testuojamą programą. Testas parašytas pasinaudojant „WindowsPhoneGUITestTool“ įrankiu. Šis įrankis yra išsamiai aprašytas skyriuje 3 „„WindowsPhoneGUITestTool“ įrankio projektas“.

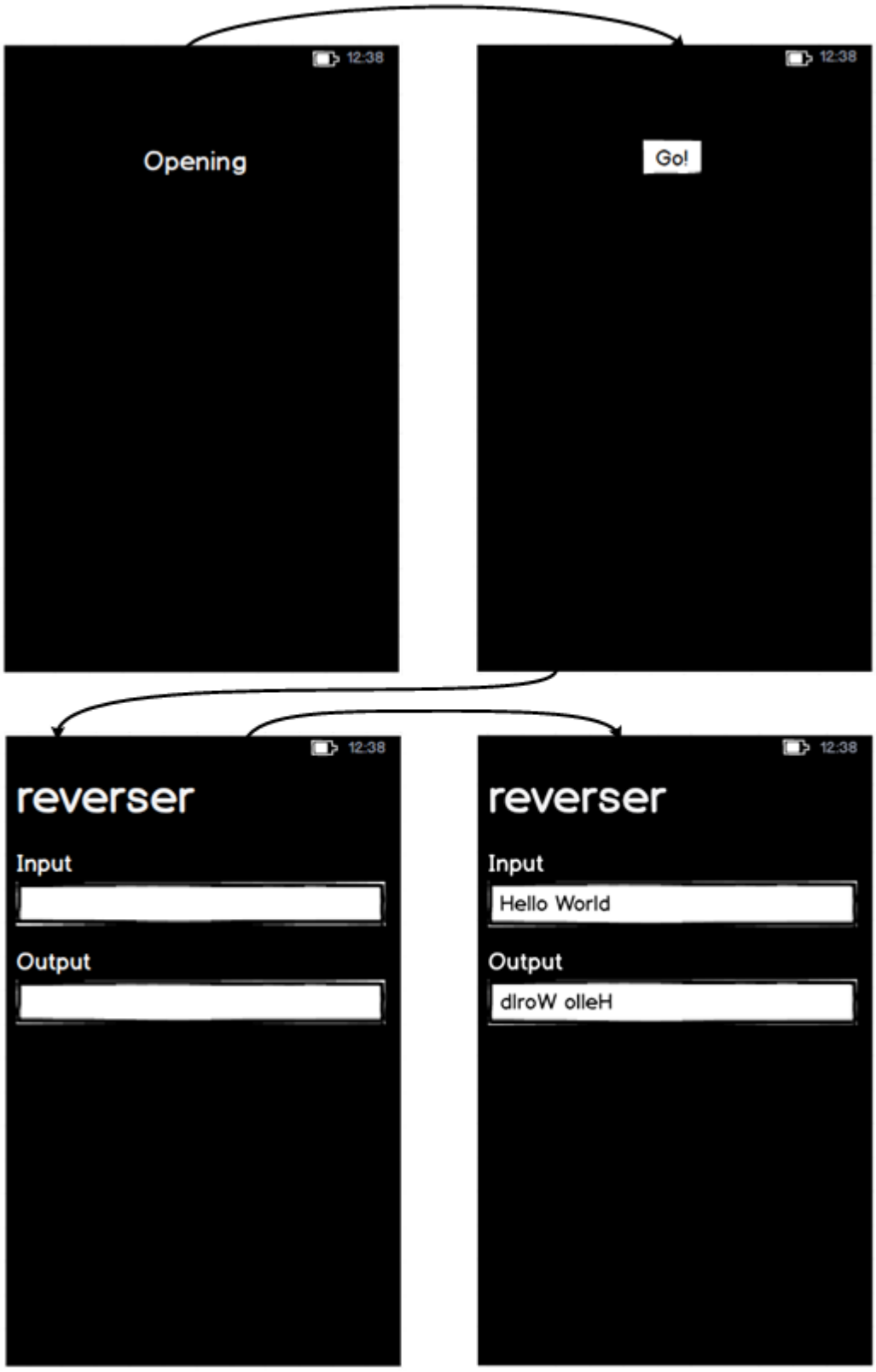
```
Feature: EAppTest
    Example application testing.

Scenario: The Pivot Page output field reverses what I type in input
    Given my app is clean installed and running
    Then I wait for the text "Go!" to appear
    Then I press the control "Go!"
    Then I wait for the text "reverser" to appear
    Then I see the control "TextBoxInput" contains ""
    And I see the control "TextBoxOutput" contains ""
    Then I enter "Hello World" into the control "TextBoxInput"
    Then I wait for some text in the control "TextBoxOutput"
```

Kiekvieno žingsnio paaiškinimas:

- „Given my app is clean installed and running“ – šiame žingsnyje yra įdiegiama ir paleidžiama testuojama programa.
- „Then I wait for the text "Go!" to appear“ – šiame žingsnyje yra laukiama, kol atsiras tekstas „Go!“ pagrindiniame programos lange.
- „Then I press the control "Go!"“ – šiame žingsnyje paspaudžiamas mygtukas, kadangi tekstas „Go!“ yra ant mygtuko.
- „Then I wait for the text "reverser" to appear“ – šiame žingsnyje yra laukiama, kol atsiras tekstas „reverser“ kitame programos lange.
- „Then I see the control "TextBoxInput" contains ""“ – šiame žingsnyje yra patikrinama, ar tuščia reikšmė yra užstatyta teksto įvedimo lauke.
- „And I see the control "TextBoxOutput" contains ""“ – šiame žingsnyje yra patikrinama, ar tuščia reikšmė yra užstatyta teksto išvedimo lauke.
- „Then I enter "Hello World" into the control "TextBoxInput"“ – šiame žingsnyje yra įvedamas tekstas į įvedimo lauką.

- „Then I wait for some text in the control "TextBoxOutput"“ – šiame žingsnyje yra laukiama, kol atsiras kažkoks, visiškai nesvarbu koks, tekstas išvedimo lauke.

12 pav. yra pateikiamas grafinis viso testo atvejo atvaizdavimas.



12 pav. Grafinis testo atvejo atvaizdavimas

4.7.2 Nustatomas minimalus teksto vykdymo laikas

Aprašius testinius atvejus yra nustatomas minimalus testo vykdymo laikas. Tai yra padaroma tiesiog paleidžiant testą, kai yra žinoma, jog programinė įranga veikia teisingai. Rekomenduojama testą įvykdyti kelis kartus, kad būtų nustatomas teisingas minimalus testo vykdymo laikas.

Pavyzdinės programos atveju buvo fiksuota 67 sekundės.

4.7.3 Kartojamas testas

Kai yra atliekami pataisymai programoje, kartojamas testo vykdymas. Tiesiog iš naujo yra paleidžiamas testas ir surenkami rezultatai.

Pavyzdinės programos atveju, atlikus pataisymus ir iš naujo paleidus testus, buvo gautas rezultatas 97 sekundės.

4.7.4 Lyginami gauti rezultatai

Pakartojus testo vykdymą, lyginami gauti rezultatai. Tikrinama, ar testas įvyko per mažesni laiką negu fiksuota arba per tokį patį, ar per didesnį. Jeigu testas įvyko per tokį patį laiką arba per mažesnį yra laikoma, kad testas klaidų neaptiko ir tęsiami tolimesni sistemos tobulinimo darbai. Jeigu testas įvyko per didesnį laiką, yra fiksuojama klaida ir ji yra taisoma.

Pavyzdinės programos atveju testas vyko ilgiau nei minimalus fiksuotas laikas, todėl yra laikoma, kad buvo aptikta klaida.

4.7.5 Atliekami pataisymai

Šiame žingsnyje yra tiesiog atliekami klaidų ištaisymai, aptikti rezultatų lyginime.

Mūsų programos atveju buvo aptikta klaida, kad užrašas „Opening“ buvo rodomas 60 sekundžių vietoje 30 sekundžių. Klaida buvo ištaisyta ir vėl kartojamas testas. Viskas kartojosi vėl nuo žingsnio „Kartojamas testas“.

4.7.6 Testavimo pabaiga

Testavimas yra baigiamas tada, kaip jau minėta skyriuje 4.5 „Testavimo pabaigos sąlyga“, kai testavimas neduoda rezultatų, t.y. nerandamo klaidos arba sistema yra baigiama tobulinti.

Pavyzdinės programos atveju testavimas buvo baigtas, kadangi sistemos tobulinimas buvo baigtas.

4.8 Apibendrinimas

1. Sukurtas metodas, kuris sprendžia testavimo laiko sąnaudų problemą.
2. Siūlomas metodas yra pritaikytas funkciniam grafinės vartotojo sąsajos testavimui.
3. Siūlomas metodas yra pritaikytas automatizuotam testavimui.
4. Siūlomas metodas yra pritaikytas veikti su mobiliais įrenginiais.

5 EKSPERIMENTINĖ DALIS

Šioje dalyje yra aprašoma klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant programos vykdymo laikines charakteristikas, eksperimentinio tyrimo analizė. Šios analizės metu buvo atliekami eksperimentai su testuojamomis programomis, siekiant įsitikinti, ar pasiūlytas metodas veikia, ir gauti rezultatai buvo lyginami su funkcinio testo metodu. Taip pat dalis šio tyrimo metu gautų rezultatų buvo publikuoti straipsnyje „Klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant laikines charakteristikas, kūrimas ir tyrimas“ [31] (šis straipsnis buvo pripažintas geriausiu sekcijoje „IT testavimas ir sauga“. Sekcijos geriausio straipsnio pažymėjimas yra pateikiamas priede 9.3 „Geriausio straipsnio diplomas“).

5.1 Naudojamos metrikos

Šiame eksperimentiniame tyrime naudojamos žemiau pateikiamos metrikos:

- Testų sudarymo laikas – duomenys gaunami skaičiuojant per kiek laiko yra sudaromas testas.
- Sugeneruotų mutantų skaičius – duomenys yra gaunami naudojant mutacinį testavimą.
- Aptiktų mutantų skaičius – duomenys yra gaunami įvykdžius testus.

5.2 Naudoti įrankiai

Metodo tyrimui atlikti buvo pasinaudota „WindowsPhoneGUITestTool“ įrankiu. Šis įrankis buvo sukurtas klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant programos vykdymo laikines charakteristikas, pagrindu. Įrankio detalus aprašymas yra pateikiamas skyriuje 3 „„WindowsPhoneGUITestTool“ įrankio projektas“.

Be to tyrimui atlikti buvo pasinaudota „Cream“ [32] įrankiu, mutantų generavimui testuojamose programose. Šiuo įrankiu naudojantis buvo sugeneruojami įvairūs mutantai. Žemiau yra pateikiama 5 lentelė, kurioje yra sąrašas visų mutantų tipų, kuriuos gali sugeneruoti įrankis ir trumpas jų aprašymas.

5 lentelė. „Cream“ įrankio generuojami mutantai

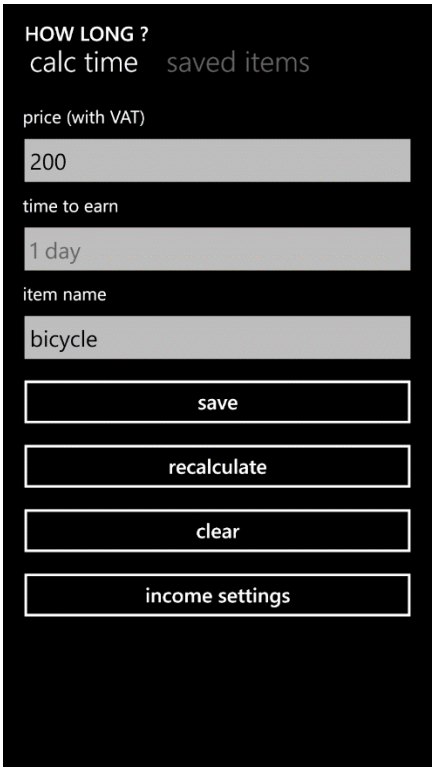
Mutanto tipas	Mutanto aprašymas
ABS	Absoliutinės reikšmės įterpimas
AOR	Aritmetinio operatoriaus pakeitimas
ASR	Masyvo rodyklės pakeitimas skaliariniu kintamuoju
LCR	Loginio jungėjo pakeitimas
LOR	Loginio operatoriaus pakeitimas
ROR	Reliacinio operatoriaus pakeitimas
UOI	Vienanario operatoriaus įterpimas
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

5.3 Naudojamos testavimui „Windows Phone“ programos

Eksperimentinio tyrimo metu buvo panaudotos realios „Microsoft Windows Phone“ operacinės sistemos programos.

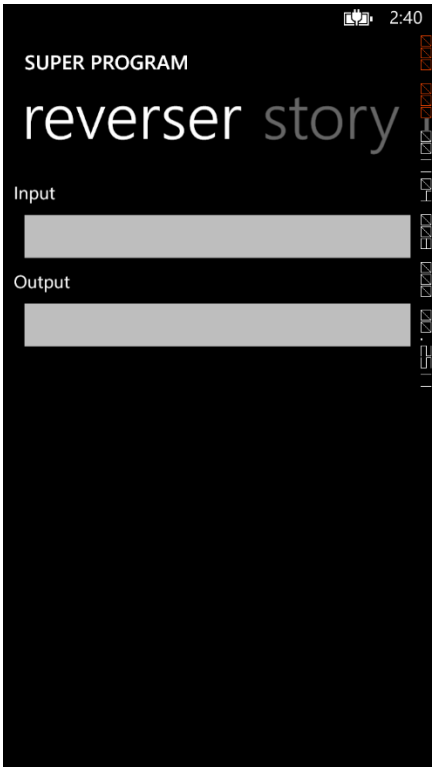
Žemiau yra pateikiamas sąrašas programų, su kuriomis buvo vykdomi eksperimentai:

- „MTConverter“ – matematinių skaičiavimų programa, skirta apskaičiuoti, kiek užtrunka laiko uždirbti įvairius daiktus. Taip pat saugo visų skaičiavimų istoriją. 13 pav. pateikiamas programos pradinis langas.



13 pav. „MTConverter“ programos pradinis langas

- „EApp“ – įvestą tekstą išveda priešinga kryptimi. Taip pat saugo visų įvestų tekstų istoriją. 14 pav. pateikiamas programos pradinis langas.

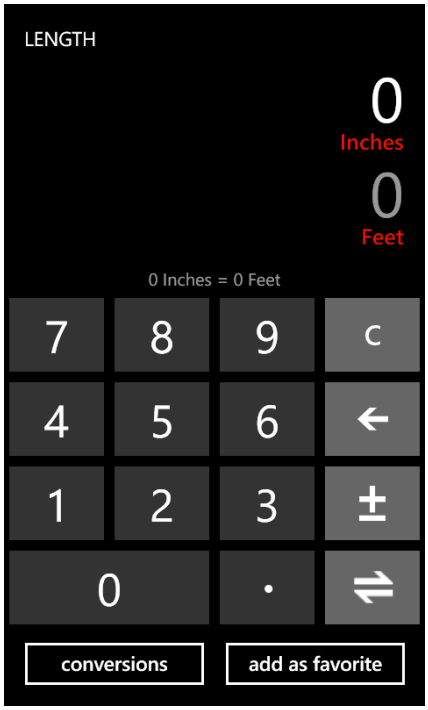


14 pav. „EApp“ programos pradinis langas

- „PhotoHub“ – atvaizduoja telefone arba telefono atmintyje esančias nuotraukas. 15 pav. yra pateikiamas programos pradinis langas.

15 pav. „PhotoHub“ programos pradinis langas

- „UnitConverter“ – programa konvertuoja įvarius matavimo vienetus iš vienų į kitus. 16 pav. yra pateikiamas programos pradinis langas.



16 pav. „UnitConverter“ programos pradinis langas

Eksperimentiniame tyrime naudojamų programų charakteristikos yra pateikiamos 6 lentelėje.

6 lentelė. Testuojamų „Microsoft Windows Phone“ programų charakteristikos

Programos pavadinimas	Klasių skaičius	Kodo eilučių skaičius
MTConverter	26	1723
EApp	6	537
PhotoHub	7	335
UnitConverter	30	4472

5.4 Eksperimento aplinka

Eksperimentiniam tyrimui atlikti buvo naudojamas kompiuteris su reikalinga programine įranga. Žemiau yra pateikiami kompiuterio techniniai parametrai:

- Procesorius – Intel Core i7 keturių branduolių, 2.20 GHz.
- Operatyvioji atmintinė – 8Gb.
- Operacinė sistema – Microsoft Windows 8.1 Pro 64bit.

Toliau yra pateikiamas panaudotos programinės įrangos sąrašas:

- Microsoft Visual Studio 2012,
- Microsoft Windows Phone 8.0 SDK,
- Microsoft Windows Phone 8.0 emuliatorius.

5.5 Eksperimento eiga

Eksperimentinio tyrimo scenarijus:

1. Pasirenkama testuojama programa.
2. Sudaromas testas klaidų paieškos mobilių įrenginių programinėje įrangoje metodu, taikant programos vykdymo laikines charakteristikas.
3. Suskaičiuojama, kiek laiko užtruko sudaryti testą klaidų paieškos mobilių įrenginių programinėje įrangoje metodu, taikant programos vykdymo laikines charakteristikas.
4. Sudaromas testas funkciniu metodu.
5. Suskaičiuojama, kiek laiko užtruko sudaryti testą funkciniu metodu.

6. Sugeneruojami mutantai.
7. Vykdomi sudaryti testai su sugeneruotais mutantais.
8. Skaičiuojama, kiek buvo sugauta mutantų su sukurtais testais.
9. Analizuojami gauti rezultatai.

5.6 Eksperimento rezultatai

Šiame skyriuje yra pateikiami atlikto eksperimentinio tyrimo gauti rezultatai ir jų analizė.

5.6.1 Rezultatų duomenys

7 lentelėje yra pateikiami viso eksperimento rezultatai.

7 lentelė. Eksperimento rezultatai

Programa	Testų sudarymo laikas		Sugeneruotų mutantų skaičius	Aptiktų mutantų skaičius		Sugeneruotų ir aptiktų mutantų santykis, %	
	Funkcinis metodas, val.	Sukurtas metodas, val.		Funkcinis metodas	Sukurtas metodas	Funkcinis metodas	Sukurtas metodas
MTConverter	5	0,5	895	637	210	71%	35%
EApp	1	0,25	14	7	4	50%	29%
PhotoHub	0,8	0,2	24	18	12	75%	50%
UnitConverter	7,5	0,6	1173	862	525	73%	45%

8 lentelėje yra pateikiama „MTConverter“ programai sugeneruoti ir aptikti mutantai abiem testavimo metodais.

8 lentelė. „MTConverter“ programai sugeneruoti mutantai

Mutanto tipas	Sugeneruotų mutantų skaičius	Aptikti mutantai	
		Funkcinis metodas	Sukurtas metodas
ABS	40	24	14
AOR	20	20	20
ASR	112	75	35
LCR	8	5	4
ROR	180	146	56
UOI	274	195	92
UOR	39	24	16
EOC	36	23	14
ISK	1	1	1
JID	21	16	8
JTD	126	80	31
OAO	37	27	18

9 lentelėje yra pateikiama „EApp“ programai sugeneruoti ir aptikti mutantai abiem testavimo metodais.

9 lentelė. „EApp“ programai sugeneruoti mutantai

Mutanto tipas	Sugeneruotų mutantų skaičius	Aptikti mutantai	
		Funkcinis metodas	Sukurtas metodas
UOI	6	4	3
IOP	6	2	0
ISK	1	1	1
JID	1	0	0

10 lentelėje yra pateikiama „PhotoHub“ programai sugeneruoti ir aptikti mutantai abiem testavimo metodais.

10 lentelė. „PhotoHub“ programai sugeneruoti mutantai

Mutanto tipas	Sugeneruotų mutantų skaičius	Aptikti mutantai	
		Funkcinis metodas	Sukurtas metodas
AOR	4	4	4
ROR	5	3	1
UOI	9	6	4
UOR	3	3	1
JID	1	1	1
JTD	1	0	0
OAO	1	1	1

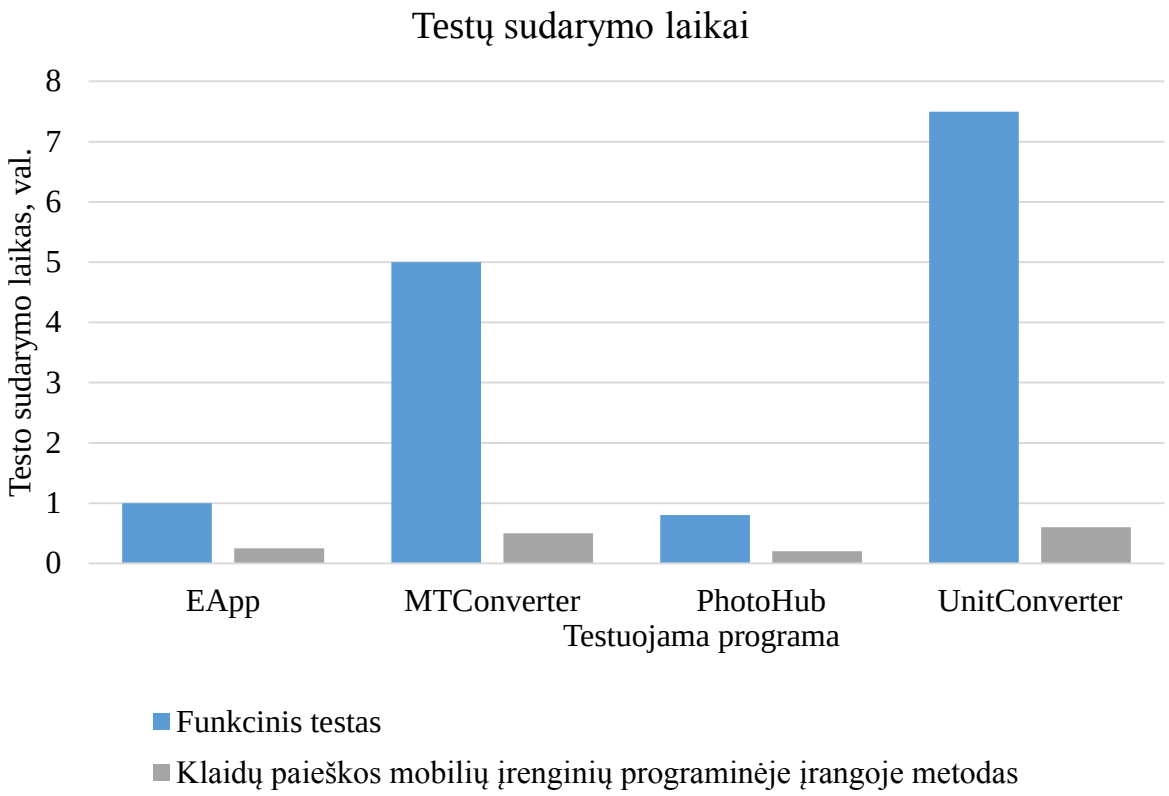
11 lentelėje yra pateikiami „UnitConverter“ programai sugeneruoti ir aptikti mutantai abiem testavimo metodais.

11 lentelė. „UnitConverter“ programai sugeneruoti mutantai

Mutanto tipas	Sugeneruotų mutantų skaičius	Aptikti mutantai	
		Funkcinis metodas	Sukurtas metodas
AOR	76	71	56
ASR	56	37	24
LCR	38	21	15
ROR	100	79	46
UOI	453	319	211
UOR	9	5	3
EOA	1	0	0
EOC	60	52	35
IOP	18	18	11
ISK	10	8	4
JID	22	17	7
JTD	294	210	99
OAO	12	8	5
PRM	2	1	0
PRV	22	16	9

5.6.2 Rezultatų analizė

Toliau esančiame 17 pav. yra pateikiama, kiek laiko buvo sugaišta sukurti testams.



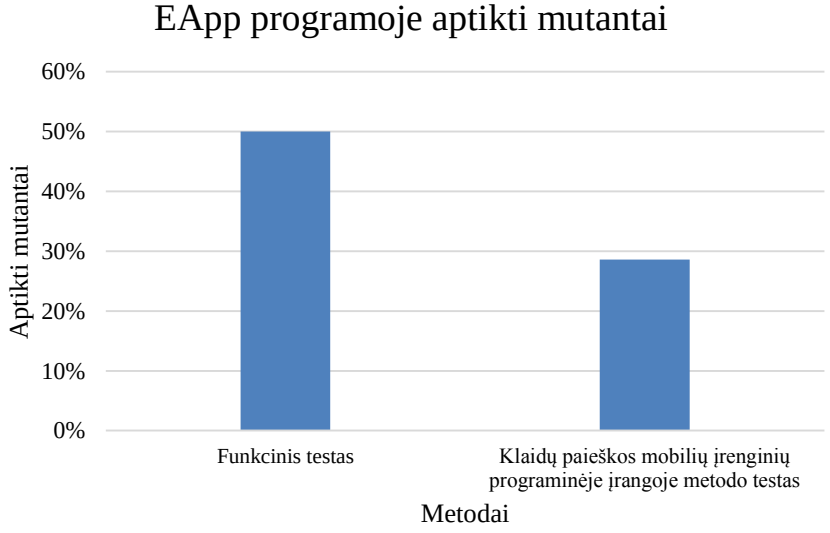
17 pav. Testų sudarymo laikai

Iš aukščiau pateiktos diagramos matoma, jog testų sukūrimo laikas sukurtu metodu lyginant su funkciniu metodu yra žymiai mažesnis:

- „EApp“ – 4 kartus mažesnis,
- „MTConverter“ – 10 kartų mažesnis,
- „PhotoHub“ – 4 kartus mažesnis,
- „UnitConverter“ – 12,5 karto mažesnis.

Iš šių rezultatų gauta, jog su sukurtu metodu testų sukūrimo laikas vidutiniškai sutrumpėjo 7,6 karto. Galima teigti, kad kuo didesnė programa, tuo geresni gaunami testo sukūrimo laikai, lyginant su programos kodo eilučių kiekiu.

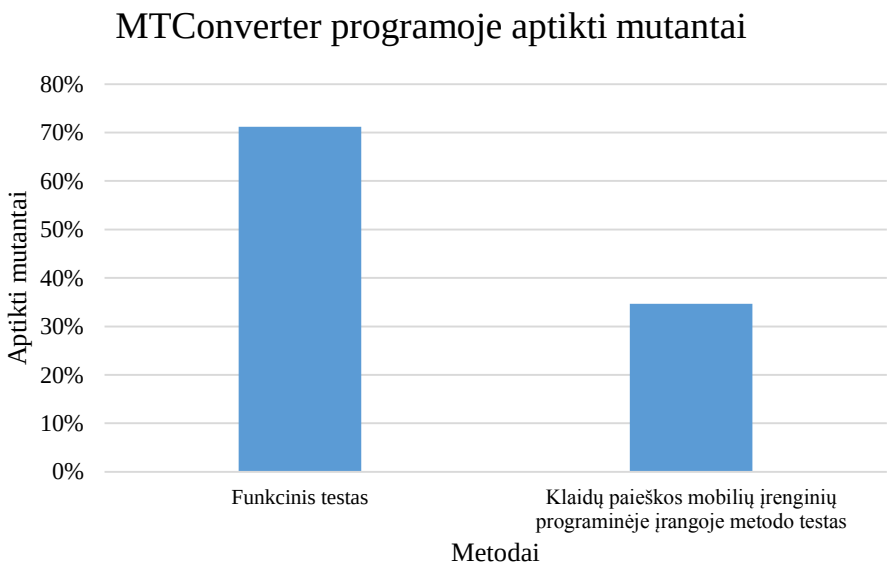
18 pav. yra pateikiama „EApp“ programoje aptiktų mutantų palyginimas funkciniu ir sukurtu metodais.



18 pav. „EApp“ programoje aptikti mutantai

Iš pateiktos diagramos matoma, jog „EApp“ programoje sugeneruotų mutantų aptikimas sukurtu metodu yra prastesnis nei funkciniu metodu. Pastebėtas 21 procento skirtumas mutantų aptikimo efektyvume programai „EApp“.

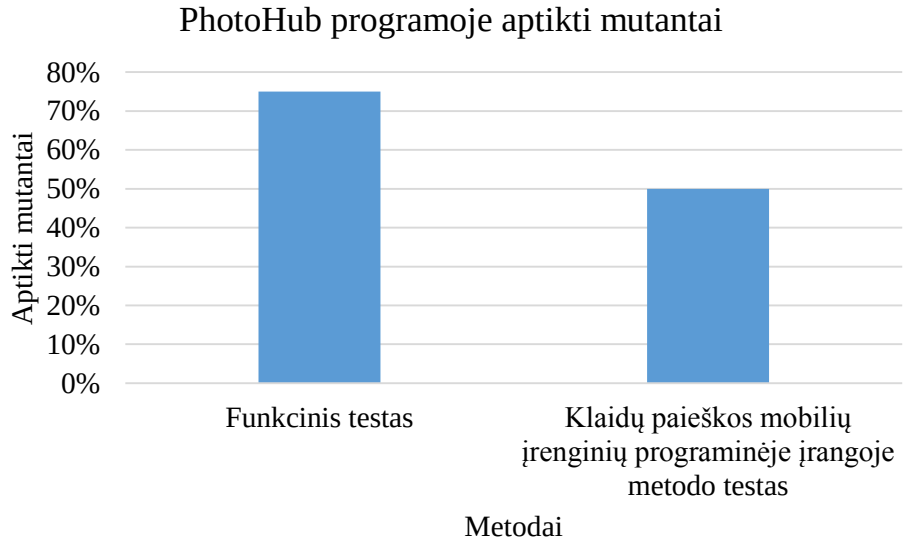
19 pav. yra pateikiama „MTConverter“ programoje aptiktų mutantų palyginimas funkciniu ir sukurtu metodais.



19 pav. „MTConverter“ programoje aptikti mutantai

Iš pateiktos diagramos yra matoma, jog „MTConverter“ programoje sugeneruotų mutantų aptikimas sukurtu metodu yra prastesnis nei funkciniu metodu. Pastebėtas 36 procentų skirtumas mutantų aptikimo efektyvume programai „MTConverter“.

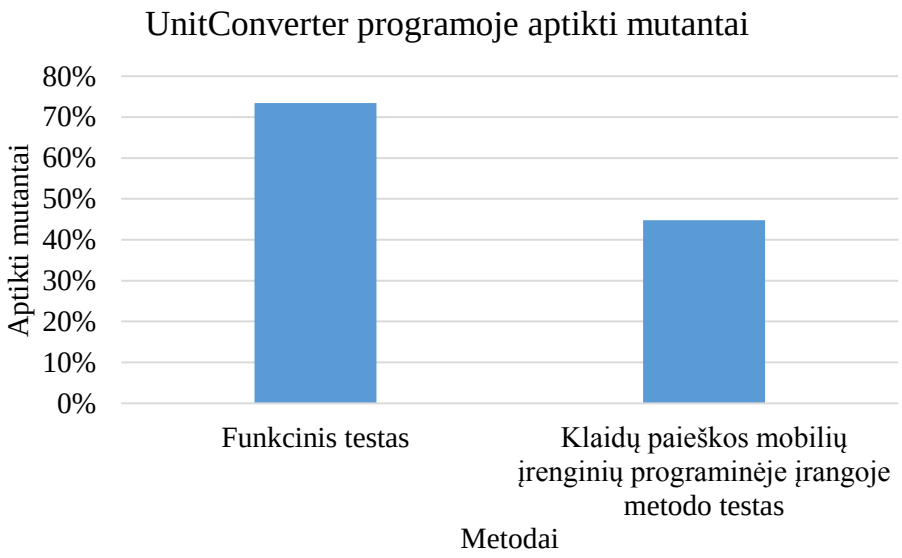
20 pav. yra pateikiama „PhotoHub“ programoje aptiktų mutantų palyginimas funkciniu ir sukurtu metodais.



20 pav. „PhotoHub“ programoje aptikti mutantai

Iš pateiktos diagramos yra matoma, jog „PhotoHub“ programoje sugeneruotų mutantų aptikimas sukurtu metodu yra prastesnis nei funkciniu metodu. Pastebėtas 21 procento skirtumas mutantų aptikimo efektyvume programai „PhotoHub“.

21 pav. yra pateikiama „UnitConverter“ programoje aptiktų mutantų palyginimas funkciniu ir sukurtu metodais.



21 pav. „UnitConverter“ programoje aptikti mutantai

Iš pateiktos diagramos yra matoma, jog „UnitConverter“ programoje sugeneruotų mutantų aptikimas sukurtu metodu yra prastesnis nei funkciniu metodu. Pastebėtas 28 procentų skirtumas mutantų aptikimo efektyvume programai „UnitConverter“.

Apibendrinant mutantų aptikimo rezultatus galima teigti, kad mutantų aptikimas nepriklauso nuo programos dydžio.

5.7 Apibendrinimas

1. Atliktas eksperimentinis tyrimas, kuris įvertina sukurtą testavimo metodą.
2. Naudojantis sukurtu testavimo metodu nustatyta, kad testų kūrimo laikas sutrumpėja vidutiniškai 7,6 karto lyginant su funkcinio testo kūrimu.
3. Naudojantis sukurtu testavimo metodu sugautų mutantų skaičius yra vidutiniškai 39,75%.
4. Pastebėta, kad kuo testuojama programinė įranga yra didesnė, tuo gaunami geresni testų sudarymo laiko rezultatai, t.y. laiko sąnaudos gaunamos žymiai mažesnės lyginant su funkciniu metodu.
5. Pastebėta, kad mutantų aptikimo negalima susieti nei su testų sudarymo laiku, nei su programos dydžiu.

6 IŠVADOS

1. Išmaniųjų telefonų programinė įranga turi būti kokybiška, kadangi vartotojai nėra linkę naudotis programine įranga, kuri veikia neteisingai.
2. Suprojektuotas, realizuotas ir ištestuotas „Windows Phone OS“ programų automatinio testavimo įrankis, skirtas „Microsoft Visual Studio“ programavimo aplinkai.
3. Sukurtas klaidų paieškos mobilių įrenginių programinėje įrangoje metodas, taikant programos vykdymo laikines charakteristikas, kuris turi tokias savybes:
   - Išsprendžia testavimo laiko sąnaudų problemą.
   - Pritaikytas funkciniam grafinės vartotojo sąsajos testavimui.
   - Pritaikytas automatizuotam testavimui.
   - Pritaikytas veikti su mobiliais įrenginiais.
4. Sukurtas testavimo metodas buvo ištirtas eksperimentiškai. Šio tyrimo metu buvo testuojamos 4 skirtingos „Windows Phone“ operacinei sistemai pritaikytos programos ir gauti tokie rezultatai:
   - Testų sudarymo laikas sutrumpėjo vidutiniškai 7,6 karto lyginant su funkciniu testavimo metodu (mažiausia reikšmė – 4 kartai, didžiausia reikšmė – 12,5 karto).
   - Vidutinis mutantų aptikimas testuojamose programose buvo 39,75% (mažiausia reikšmė – 29%, didžiausia reikšmė - 50%).
5. Buvo pastebėta, kad kuo testuojama programinė įranga yra didesnė, tuo gaunami geresni testų sudarymo laiko rezultatai, t.y. laiko sąnaudos gaunamos žymiai mažesnės lyginant su funkciniu metodu.
6. Atlikus eksperimentinį tyrimą buvo pastebėta, kad metodą yra efektyviausia naudoti programinės įrangos kūrimo pradžioje.

7 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, et. al., “Gast: Generic Automated Software Testing”, Springer Berlin Heidelberg, ISSN: 0302-9743, 84-100, 2002.

[8] P. Li, et. al., “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] A. Adamoli, et. al., “Automated GUI performance testing”, Software Quality Journal, Vol. 19, Issue: 4, ISSN: 0963-9314, 801-839, 2011.

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

[16] Abbot framework for automated testing of Java GUI components and programs [interaktyvus]. 2012 m. – [žiūrėta 2012-11-07]. Prieiga per internetą: <http://abbot.sourceforge.net/doc/overview.shtml>

[17] Jacareto [interaktyvus]. 2012 m. – [žiūrėta 2012-11-07]. Prieiga per internetą: <http://sourceforge.net/apps/mediawiki/jacareto/index.php >

[18] Pounder [interaktyvus]. 2012 m. – [žiūrėta 2012-11-07]. Prieiga per internetą: <http://pounder.sourceforge.net/>

[19] Marathon [interaktyvus]. 2012 m. – [žiūrėta 2012-11-07]. Prieiga per internetą: <http://marathontesting.com/>

[20] Introduction to JFCUnit [interaktyvus]. 2012 m. – [žiūrėta 2012-11-07]. Prieiga per internetą: <http://jfcunit.sourceforge.net/>

[21] Xuan Yuan, Atif M Memon, Generating Event Sequence-Based Test Cases Using Gui Runtime State Feedback. IEEE Transactions on Software Engineering – 2010 sausis, Vol. 36, Nr. 1, p. 81-95.

[22] R. C. Bryce, A. M. Memon, Test Suite Prioritization by Interaction Coverage. Workshop on Domain specific approaches to software test automation: in conjunction with the 6th ESEC/FSE joint meeting – 2007, p. 1-7.

[23] R. C. Bryce, S. Samapath, A. M. Memon, Developing a Single Model and Test Prioritization Strategies for Event-Driven Software. Ieee Transactions On Software Engineering, Vol. 37, Nr. 1, 2011 sausis, p. 48-64.
[24] M. Grechanik, Q. Xie, C. Fu, Maintaining and evolving gui-directed test scripts. Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference – 2009 gegužė, p. 408-418.
[25] Q. Xie, A. M. Memon, Model-based testing of community-driven open-source GUI applications. Software maintenance, ICSM – 2006 rugsėjis, p. 145 - 154.
[26] A. Ruiz, Y. W. Price, GUI Testing Made Easy. In Proceedings of the testing: Academic & industrial conference—practice and research techniques, 2008, p. 99-103.
[27] D. H. NGuyen, P. Strooper, J. G. Suess, Model-based testing of multiple gui variants using the gui test generator. ACM Proceedings of the 5th Workshop on Automation of Software Test – 2010, p. 24 – 30.
[28] J. Stecker, A. M. Memon, Relationships between test suites, faults, and fault detection in gui testing. Software Testing, Verification, and Validation, 2008 1st International Conference – 2008 balandis, p. 12 – 21.
[29] System.Reflection Namespace [interaktyvus]. 2012m – [žiūrėta 2012-11-21]. Prieiga per internetą <http://msdn.microsoft.com/en-us/library/windowsphone/develop/system.reflection(v=vs.105).aspx>
[30] Nokia Developer [interaktyvus]. 2012m – [žiūrėta 2012-11-21]. Prieiga per internetą: <http://www.developer.nokia.com/Community/Wiki/How_to_take_screenshot_on_Windows_Phone#Introduction>
[31] D. Liepinaitis, G. Motiejūnas, T, Kuckis, Klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant laikines charakteristikas, kūrimas ir tyrimas, XIX tarpuniversitetinė magistrantų ir doktorantų konferencija „Informacinė visuomenė ir universitetinės studijos“ (IVUS 2014), ISSN: 2029–4832, 2014, p. 55-59.
[32] CREAM, Creator of Mutants, [interaktyvus]. http://galera.ii.pw.edu.pl/~adr/CREAM, [žiūrėta 2014-03-03].

8 TERMINŲ IR SANTRUMPŲ ŽODYNAS

ATM	„Automated Teller Machine“ yra kompiuterizuotas telekomunikacijų įrenginys, kuris teikia finansinių transakcijų galimybę be finansinių institucijų atstovų.
Black-box testing	Juodos dėžės testavimas. Testavimo būdas, kai testuojant yra naudojami vartotojo reikalavimai ir specifikacijos.
Bug	Žiūrėkite „Defect“.
Business Rule	Tvirtinimas, kuris nusako arba apibrėžia kai kuriuos verslo aspektus. Jis yra naudojamas įvertinti verslo struktūrą arba kontroliuoti, arba įtakoti verslo elgseną.
Class under test	Programinės įrangos klasė, kuri yra testuojama arba testai yra sugeneruoti jos testavimui.
Defect	Klaida komponente arba sistemoje, kuri gali įtakoti komponento arba sistemos funkcijos neįvykdymą. Taip pat gali įtakoti komponento arba sistemos trikį .
Failure	Esminis komponento arba sistemos nukrypimas nuo tikėtino rezultato arba paslaugos.
False positive	Testo sėkmingas įvykdymas, nors problema vis dar egzistuoja ir ji nėra surasta testo metu.
Fault	Žiūrėkite „defect“.
MD5	„Message-Digest“ algoritmas. MD5 tai yra kriptografinė maišos funkcija su 128 bitų maišos reikšme.
MDA	Modeliu pagrįsta architektūra.
MDE	Modeliu pagrįsta inžinerija.
Method under test	Programinės įrangos klasės metodas, kuris yra testuojamas arba testai yra sugeneruoti jo testavimui.
Mutation Analysis	Metodika nuspręsti testo rinkinio visapusiškumą, matuojant testo apimtį, kuria testo rinkinys gali atskirti programą nuo nežymių kitų programos variantų.
Mutation Testing	Žiūrėkite „Mutation Analysis“.
PIM	Nuo platformos nepriklausomas modelis. Modelis, kuris neturi detalių apie specifinę platformą.
PSM	Nuo platformos priklausomas modelis. Modelis, kuris turi detalių apie konkrečią platformą.
Regression Testing	Seniau testuotos programos testavimas, norint patikrinti, ar neatsirado defektų nemodifikuotose programos vietose padarius pakeitimus. Atliekamas, kai programinė įranga ar jos aplinka pasikeitė.
Successful test	Testas, kuris gali aptikti klaidą.
SUT	Testojama programinė įranga.
Test automation	Programinės įrangos testavimo veikos ir rezultatų tikrinimas.
Test Case	Aibė įėjimo duomenų, vykdymo prieš sąlygos, tikėtini rezultatai ir vykdymo po sąlygos, sukurtos tam tikram tikslui arba testo sąlygoms.
Test Fail	Programinei įrangai buvo įvykdytas testas ir jis negražino tikėtinų rezultatų, klaida buvo rasta.
Test Oracle	Šaltinis nustatyti tikėtinus rezultatus palyginimui su rezultatais po įvykdyto testo. Orakulas gali būti egzistuojanti sistema, vartotojo vadovas arba tam tikro individo specializuotos žinios, bet ne išeities kodas.
Test Pass	Programinei įrangai buvo įvykdytas testas ir buvo grąžinti tikėtini rezultatai, klaida nebuvo rasta.
Test-driven development	Testavimu pagrįstas kūrimas yra programinės įrangos kūrimo technika, kuri naudoja trumpas kūrimo iteracijas pagrįstas jau parašytais testo atvejais, kurie apibrėžia norimus patobulinimus arba naujas funkcijas. Kiekviena iteracija sukuria išeities kodą, reikalingą įvykdyti iteracijos testą be klaidų.
Triangulation	Daugiakampių dekompozicija į trikampius.

UML	Unifikuota modeliavimo kalba.
UTF	Vienetų testavimo karkasas.
White-box testing	Baltos dėžės testavimas. Testavimo būdas, kai testuojant yra nagrinėjamas programos išeities kodas.
XML	XML (angl. – „*Extensible Mark-up Language*“) yra bendros paskirties duomenų struktūrų bei jų turinio aprašomoji kalba.

9 PRIEDAI

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

Straipsnis yra pateiktas ir pristatytas 19-oje tarpuniversitetinėje magistrantų ir doktorantų konferencijoje „Informacinė visuomenė ir universitetinės studijos“ (IVUS 2014). Šis straipsnis buvo išrinktas geriausiu „IT testavimo ir saugos“ sekcijos straipsniu.

Straipsnis yra pateikiamas kitame puslapyje.

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;

1 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. 2 skyriuje pateikiama panašių metodų apžvalga. 3 skyriuje aprašomas sukurtas klaidų paieškos mobilių įrenginių programinėje įrangoje metodas, taikant laikines charakteristikas. 4 skyriuje pateikiama tyrimo metodika ir apžvelgiami panaudoti įrankiai tyrimui atlikti. 5 skyriuje pateikiami atlikto tyrimo rezultatai. 6 skyriuje pateikiamos išvados (apibendrinti tyrimo rezultatai). 7 skyriuje pateikiami planuojami ateities darbai.

2 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.

3 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:

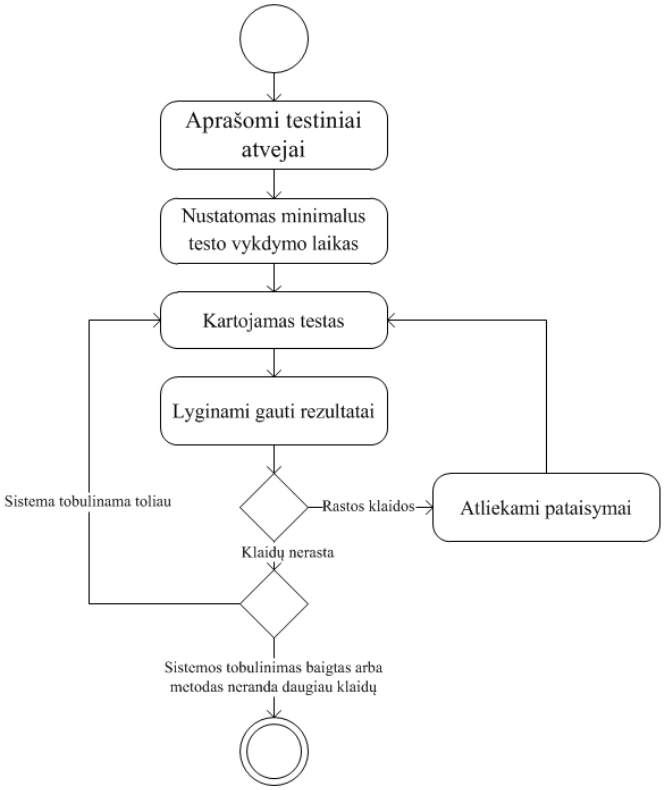
- Testuojama programinė įranga turi būti kuriama mobiliems įrenginiams.
- Testuojama programinė įranga turi turėti grafinę vartotojo sąsają.
- Turi būti prieinamas testuojamos programinės įrangos išeities kodas.

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.



22 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*“.

4. 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

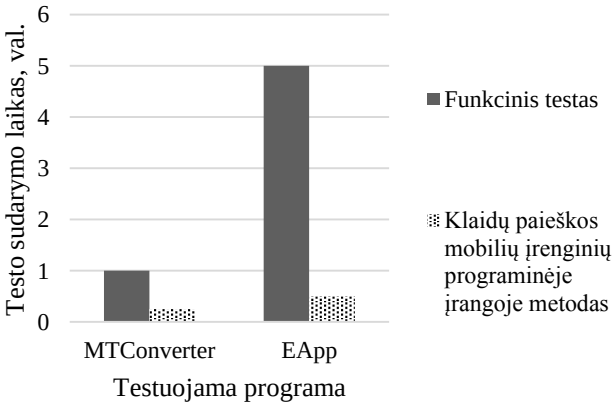
5. TYRIMO REZULTATAI

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 23 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



23 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 24 pav.

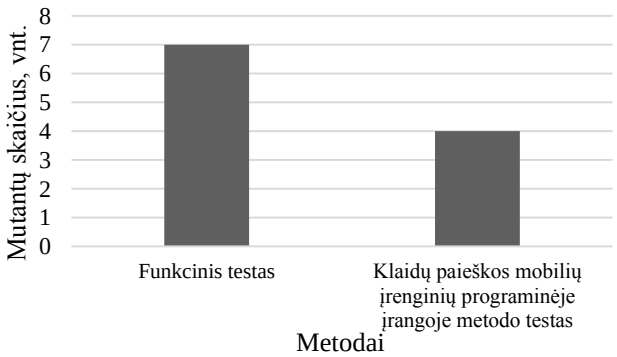
Programos MTConverter funkcinio testo ir klaidų paieškos mobilių įrenginių programinėje įrangoje metodo atliktų testų rezultatai yra pateikiami V ir VI lentelėse. Šiose lentelėse pateikiami tik tie mutantai, kurie buvo sugeneruoti, t.y. mutantai, kurie nebuvo sugeneruoti, nepateikiami.

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



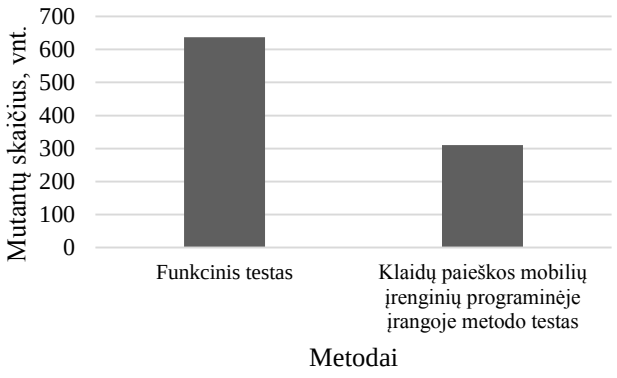
24 pav. Aptiktų mutantų EApp programoje skirtingais metodais palyginimas

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



25 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 25 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.

6. 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.

7. 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.

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

[3] 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] 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

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.

9.2 Dalyvio pažymėjimas pristačius straipsnį „Klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant laikines charakteristikas, kūrimas ir tyrimas“

Kitame puslapyje yra pateikiamas dalyvio pažymėjimas, gautas pristačius straipsnį „Klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant laikines charakteristikas, kūrimas ir tyrimas“.

KAUNO
TECHNOLOGIJOS
UNIVERSITETAS

DALYVIO PAŽYMĖJIMAS

2014-04-24 Nr. ST25-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**

9.3 Geriausio straipsnio diplomas

Kitame puslapyje yra pateikiamas geriausio straipsnio diplomas sekcijoje „IT testavimas ir sauga“ pristatytam straipsniui „Klaidų paieškos mobilių įrenginių programinėje įrangoje metodo, taikant laikines charakteristikas, kūrimas ir tyrimas“, vykusioje konferencijoje „Informacinė visuomenė ir universitetinės studijos“ (IVUS 2014).

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**

9.4 Straipsnis „Automatinio testinių duomenų generavimo metodų tyrimas“

Straipsnis yra pateiktas ir pristatytas 19-oje tarpuniversitetinėje magistrantų ir doktorantų konferencijoje „Informacinė visuomenė ir universitetinės studijos“ (IVUS 2014).

Straipsnis yra pateikiamas kitame puslapyje.

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].

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

$$= \chi. \quad (1)$$

$$\alpha + \beta \quad (1)$$

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

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

D. Standartiniu genetiniu algoritmu paremtas metodas

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

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

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

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

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

IV. Tyrimo metodika ir naudoti įrankiai

A. Testinių duomenų generatorius

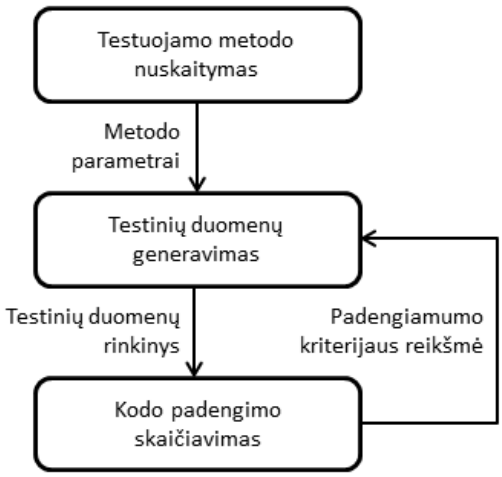
Tam, kad skirtingus testinių duomenų generavimo metodus būtų galima tirti esant kuo panašesnėmis sąlygomis bei siekiant kuo labiau supaprastinti jų panaudojimą, buvo sukurta speciali programinė įranga skirta generuoti duomenis vienu iš pasirinktų metodų – testinių duomenų multi-generatorius (TDMG). TDMG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

- Duomenų generatorius – tai komponentas, kuriame yra realizuotas duomenų generavimo algoritmas.

TDMG buvo sukurtas taip, kad prie jo lengvai galima prijungti kelis skirtingus algoritmus realizuojančius generatorius.

- Kodo padengimo skaičiavimo komponentas. Šis komponentas apskaičiuoja kodo padengiamumą su pasirinktais testiniais duomenimis. TDMG padengiamumui skaičiuoti naudoja JaCoCo [19] biblioteką.
- Valdymo komponentas. Šis komponentas apjungia pirmuosius du komponentus bei kontroliuoja jų veikimą.

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



26 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 0,1s.). 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.

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.

XII 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

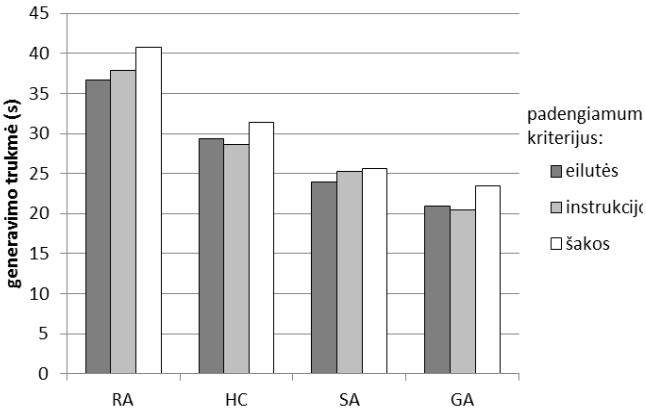
XIII LENTELĖ. GENERAVIMO METODŲ PARAMETRŲ REIKŠMĖS

Parametras	Reikšmė	Generavimo metodas			
		RA	*HC*	*SA*	*GA*
minimalus tenkintins padengiamumas	95%	X	X	X	X
maksimalus iteracijų kiekis	1000	X	X	X	
maksimalus vidinių iteracijų kiekis	50			X	
temperatūros mažinimo koeficientas	0,9			X	
populiacijos dydis	500				X
kryžminimo tikimybė	0,9				X
mutavimo tikimybė	0,01				X
maksimalus populiacijų skaičius	1000				X
short, int, long rėžiai	–10 - 1000	X	X	X	X
float, double režiai	–10,0 - 1000,0	X	X	X	X
string rėžiai	[a-z], [A-Z], [0-9]	X	X	X	X

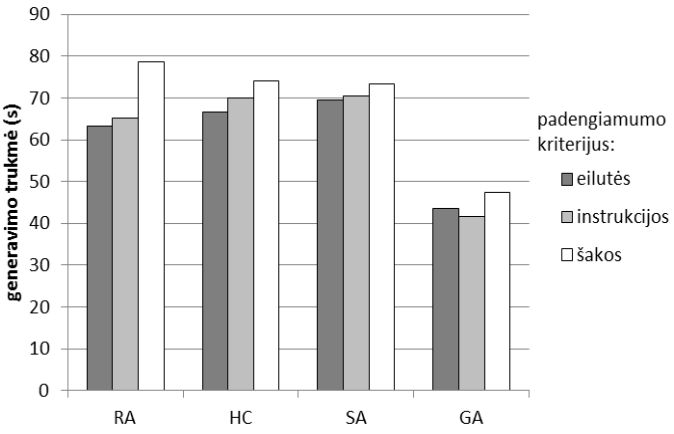
V. TYRIMO REZULTATAI

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

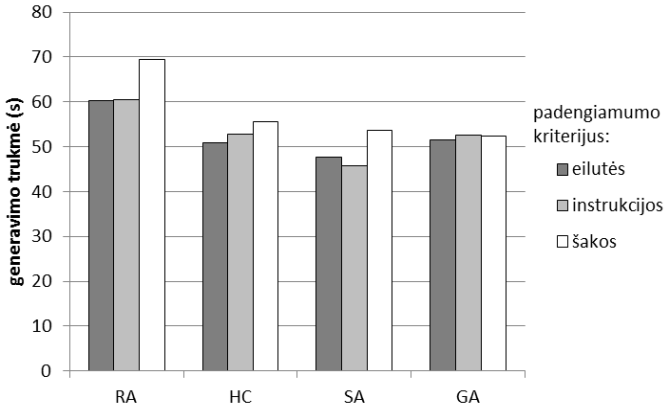
Iš šių rezultatų matyti, jog beveik visais atvejais testinių duomenų generavimas remiantis šakų padengiamumu užtrunka ilgiau negu remiantis instrukcijų arba kodo eilučių padengiamumais.



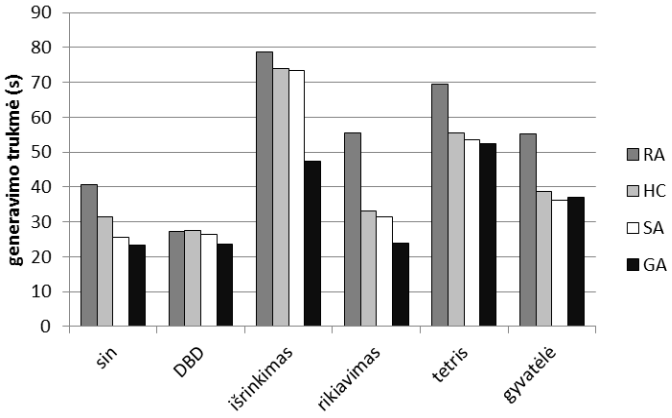
27 pav. Testinių duomenų generavimo sinuso skaičiavimo algoritmui trukmė



28 pav. Testinių duomenų generavimo įrašo išrinkimo algoritmui trukmė



29 pav. Testinių duomenų generavimo tetrio žaidimo logikos klasei trukmė



30 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.

9.5 Dalyvio pažymėjimas pristačius straipsnį „Automatinio testinių duomenų generavimo metodų tyrimas“

Kitame puslapyje yra pateikiamas dalyvio pažymėjimas, gautas pristačius straipsnį „Automatinio testinių duomenų generavimo metodų tyrimas“.

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**