KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Egidijus Babenskas

Vienetų testų generavimo metodo Android aplikacijoms testuoti realizavimas ir tyrimas

Magistro darbas

Darbo vadovas:
dr. Šarūnas Packevičius

Kaunas, 2012

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Egidijus Babenskas

Vienetų testų generavimo metodo Android aplikacijoms testuoti realizavimas ir tyrimas

Magistro darbas

Recenzentas:
doc. dr. Jevgenijus Toldinas
2012-05-25

Vadovas:
dr. Šarūnas Packevičius
2012-05-25

Atliko:
IFM-0/2 gr. stud.
Egidijus Babenskas
2012-05-25

Kaunas, 2012

Implementation and research of unit tests generation method for testing Android applications

SUMMARY

With the development of smart phones and their technical capabilities and increase of their sales in Lithuania and the world applications become more complex and have more functionality, but the issue of quality remains a painful part of the development of software.

Currently 50% out of all smart phones are sold with Android operating system. Having an increasing demand and popularity of Android OS applications in the market, as well as having researched the current market and seen that there is a lack of testing tools to test Android applications, it has been decided that a solution generating unit tests is needed to test Android applications. The main goal of this work is to provide unit test generation solution for the Android OS application testing, implementation and validate it experimentally.

This work proposes a method generating unit tests based on random generation, using OCL constraints and regression testing principles. It is compatible with Google plug-in ADT and Android SDK tools. The tool is designed as a plugin in Eclipse development environment.

Efficiency of the proposed decision of generating unit tests is proved by experimental study. During this study four applications were tested. Using the tool the average of catched mutants is 75%. The minimum value is 69%, while the highest - 88%. On average coverage of code lines is achieved by 85%. The minimum value is 72% and the maximum value of coverage - 97%.

SANTRAUKA

Tobulėjant išmaniesiems telefonams ir jų techninėms galimybėms bei didėjant jų pardavimams Lietuvoje ir pasaulyje, kuriamos aplikacijos tampa sudėtingesnės ir funkcionalesnės, tačiau kokybės problema vis dar išlieka skaudžia programinės įrangos kūrimo dalimi.

Šiuo metu iš visų parduodamų išmaniųjų telefonų apie 50% parduodami su Android operacine sistema. Matant Android OS programų vis didėjantį poreikį rinkoje ir jų populiarumą bei panagrinėjus esamą rinką ir pamačius, jog testavimo įrankių, skirtų testuoti Android aplikacijas, beveik nėra, buvo nuspręsta, jog reikalingas vienetų testų generavimo sprendimas pritaikytas testuoti Android aplikacijas. Šio darbo pagrindinis tikslas ir yra pateikti vienetų testų generavimo sprendimą skirtą Android OS aplikacijos testuoti, jį realizuoti bei pagrįsti eksperimentiškai.

Darbe siūlomas vienetų testų generavimo metodas, kuris remiasi atsitiktiniu generavimu, naudoja OCL apribojimus bei regresinio testavimo principus. Taip pat yra suderinamas su Google kompanijos teikiamu ADT įskiepiu ir Android SDK priemonėmis. Įrankis sukurtas kaip Eclipse programavimo aplinkos įskiepis.

Pasiūlyto vienetų testų generavimo sprendimo efektyvumas įrodomas eksperimentiniu tyrimu. Šio eksperimento metu buvo testuojamos 4 aplikacijos. Naudojantis įrankiu vidutiniškai sugautų mutantų skaičius yra 75%. Mažiausia reikšmė yra 69%, o didžiausia – 88%. Vidutiniškai pasiekiamas 85% kodo eilučių padengimas. Mažiausia reikšmė yra 72%, o didžiausia padengimo reikšmė - 97%.

Turinys

PAVEIKSLĖLIŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

1. ĮVADAS

Šiuo metu ypač didėja išmaniųjų telefonų pardavimai. Žmonės vis dažniau renkasi ne paprastus telefonus, o išmaniuosius su įvairiomis operacinėmis sistemomis, tokiomis kaip Android, iOS, Symbian ar Maemo. Pagal garsios tyrimų kompanijos „Gartner“ pateiktą ataskaitą [9] per 2011 metų paskutinįjį ketvirtį parduota 149 milijonai išmaniųjų telefonų, tai 47 procentais daugiau palyginus su 2010 metų paskutiniu ketvirčiu.

Lietuvoje, kaip teigia Omnitel veiklos ataskaita [25], per trečiąjį 2011 metų ketvirtį lyginant su tuo pačiu laikotarpiu pernai, išmaniųjų telefonų pardavimai augo 8 proc., o per tris pirmuosius metų ketvirčius, lyginant su 2010 m. tuo pačiu laikotarpiu, išaugo 46 procentais. Pagal „Gartner“ kompanijos ataskaitą iš visų per 2011 ketvirtąjį ketvirtį parduodamų išmaniųjų telefonų 50,9 procentai buvo parduodami su Google kompanijos Android operacine sistema (OS).

Tobulėjant išmaniesiems telefonams ir jų techninėms galimybėms, kuriama programinė įranga tampa sudėtingesnės ir funkcionalesnės, tačiau kokybės problema vis dar išlieka skaudžia programinės įrangos kūrimo dalimi. Sukurta programinė įranga dažniausiai yra pateikiama su aibe klaidų, o kokybiška programinė įranga yra daugiau išimtis, nei įprastinis dalykas. Pačios programinės įrangos kūrimo technologijos tobulėja, kurios įgalina paprasčiau kurti programinę įranga, o tuo pačiu turėtų leisti kurti ir kokybiškesnę programinę įrangą, tačiau kaip buvo kuriama programinė įranga su klaidomis, taip ir išlieka kokybės problema, net ir taikant naujausias kūrimo ir projektavimo technologijas [37].

Kaip teigia knygos [32] autorius, nesvarbu kiek daug laiko bus investuojama į projektavimą ir kaip atsakingai bus programuojama, klaidos yra neišvengiamos ir sukurtose aplikacijose jų tikrai bus. Tačiau, jeigu klaidos aptinkamos pradinėje aplikacijų kūrimo stadijoje, tai gali sutaupyti pinigų ir sumažinti aplikacijos priežiūros kaštus. Tai yra didžiausia priežastis rašyti programinės įrangos testus kuriamai aplikacijai.

Matant Android OS programų vis didėjantį poreikį rinkoje ir jų populiarumą bei panagrinėjus esamą rinką ir pamačius, jog testavimo įrankių, skirtų testuoti Android aplikacijas, beveik nėra, buvo nuspręsta, jog reikalingas vienetų testų generavimo sprendimas pritaikytas testuoti Android aplikacijas. Šio darbo pagrindinis tikslas ir yra pateikti vienetų testų generavimo metodą skirtą Android OS aplikacijos testuoti. Šio darbo uždaviniai yra realizuoti projekto metu vienetų testų generavimo įrankį panaudojant kelis testų generavimo metodus, ištirti metodų efektyvumą ir jų tinkamumą testuojant Android aplikacijas ir pasiūlyti apibendrintą vienetų testų generavimo metodą.

Šis realizuotas įrankis palengvins Android OS aplikacijų testavimą, padės greičiau aptikti klaidas kuriant Android OS aplikacijas. Taip pat suteiks galimybę valdyti testus, analizuoti testavimo rezultatus bei pačių testų efektyvumą.

Darbo analitinėje dalyje apžvelgiamas Android OS testavimo principai, išanalizuojama dabartinė esamų panašių įrankių rinka ir pateikiamas palyginimas. Apibrėžiamos kylančios problemos bei analizuojami galimi sprendimo būdai. Projekto metu realizuojamas testų generavimo įrankis, kuris generuoja testus skirtus Android OS aplikacijoms testuoti. Šiame įrankyje realizuojami trys pasirinkti testų generavimo algoritmai: atsitiktinis, genetinis bei paremtas OCL apribojimais. Tyrimo metu atliekama įrankio analizė tiriant realizuotų vienetų testų generavimo algoritmų efektyvumą ir nustatomi galimų generavimo algoritmų trūkumai ir pateikiamas patobulintas bendras vienetų testų generavimo sprendimas skirtas Android OS aplikacijoms testuoti. Šis sprendimas realizuojamas patobulinant projekto metu sukurtą programinę įrangą ir atliekant eksperimentinį tyrimą pagrindžiamas praktiškai išbandant su realiomis Android OS parašytomis mobiliomis aplikacijomis.

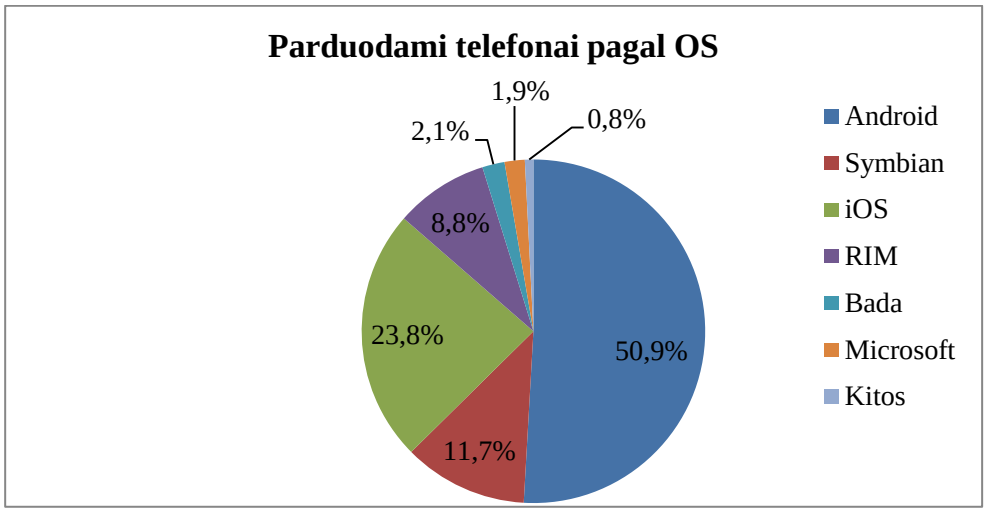
2. ANALITINĖ DALIS

2.1 Android OS aplikacijų testavimo poreikis

Šiuo metu ypač didėja išmaniųjų telefonų pardavimai. Pagal garsios tyrimų kompanijos „Gartner“ pateiktą ataskaitą [9] per 2011 metų paskutinįjį ketvirtį parduota 149 milijonai išmaniųjų telefonų, tai 47 procentais daugiau palyginus su 2010 metų paskutiniu ketvirčiu.

Lietuvoje, kaip teigia Omnitel veiklos ataskaita [25], per trečiąjį 2011 metų ketvirtį lyginant su tuo pačiu laikotarpiu pernai išmaniųjų telefonų pardavimai augo 8 proc., o per tris pirmuosius metų ketvirčius, lyginant su 2010 m. tuo pačiu laikotarpiu, išaugo 46 procentais.

Pagal „Gartner“ kompanijos ataskaitoje [9] pateikiamą diagramą 1 pav. matome, jog iš visų per 2011 ketvirtąjį ketvirtį parduodamų išmaniųjų telefonų 50,9 procentai buvo parduodami su Google kompanijos Android operacine sistema (OS). Taigi kaip matome Android OS sistema populiariausia tarp išmaniųjų telefonų. Android sistema naudojama ne tik mobiliuose telefonuose bet ir planšetiniuose kompiuteriuose.



1 pav. 2011 m. 4 ketvirčio parduodamų telefonų operacinės sistemos [9]

Google kompanijos Android OS sistema buvo pristatyta 2007 metų pabaigoje, taigi ši sistema palyginus su kitomis operacinėmis sistemomis yra jauna. Kadangi ši sistema sulaukė tokio didelio vartotojų susidomėjimo, jai skirtų aplikacijų sukuriama labai daug, o šiuo metu įrankių palengvinančių šių aplikacijų kūrimą nėra labai daug.

Kaip teigia knygos [32] autorius, nesvarbu kiek daug laiko bus investuojama į projektavimą ir kaip atsakingai bus programuojama, klaidos yra neišvengiamos ir sukurtose aplikacijose jų tikrai bus. Ir taip pat nėra svarbu ar kuriama paprasta aplikacija, kuri bus patalpinta į Google Play [10] svetainę, ar kuriama sudėtinga Android aplikacija, kuri bus talpinama į specifinį įrenginį, klaidų vis vien nebus galima išvengti ir tos klaidos dažniausiai kainuoja pinigus. Tačiau, jeigu klaidos aptinkamos pradinėje aplikacijų kūrimo stadijoje, tai

gali sutaupyti pinigų ir sumažinti aplikacijos priežiūros kaštus. Tai yra didžiausia priežastis rašyti programinės įrangos testus kuriamai aplikacijai. Taip pat kaip teigia [32] autorius, testų rašymas leidžia giliau suprasti reikalavimus ir esamą probleminę sritį. Testų parašyti neišeis, jeigu nebus galima suprasti kurios nors programos dalies.

2.2 Android OS aplikacijų testavimas

Kai 2007 metų pabaigoje buvo pristatytas Android operacinė sistema mobiliems telefonams, tai Google skyrė labai mažą dėmesį šių aplikacijų testavimui, tačiau dabar labai išpopuliarėjus Android sistemai, Google suteikia daug didesnių testavimų galimybių. [32]

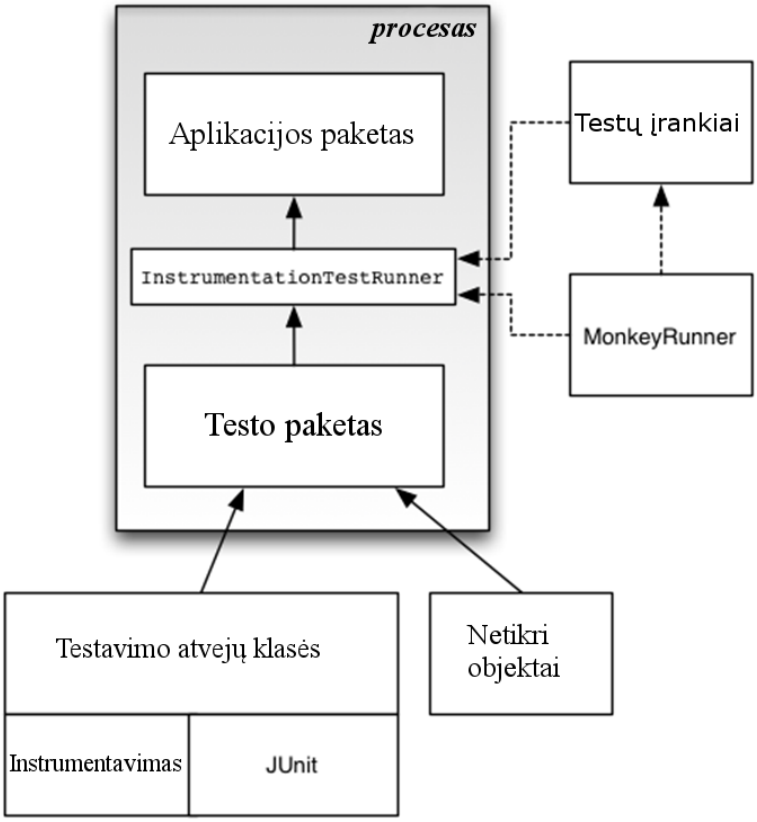
Google pateikiamas Android testavimo karkasas yra dalis aplikacijų kūrimo aplinkos, kuri suteikia architektūrinį priėjimą ir galingus įrankius, kurie padeda testuoti aplikaciją naudojant skirtingas testavimo strategijas [13]

Kai kuriais atvejais yra reikalingi papildomi įrankiai, tačiau šių įrankių integravimas daugeliu atveju yra paprastas ir nesudėtingas.

Pagrindinės Android testavimo karkaso savybės remiantis [13] šaltiniu:

- Android vienetų testai yra paremti JUnit karkasu. Galima naudoti ir paprastus JUnit testus testuoti toms klasėms, kurios visai nenaudoja Android API, o Android Junit testavimo priemones naudoti testuojant Android komponentams.
- Android JUnit išplėtimas suteikia komponentų specifines testavimo klases. Šių klasių teikiamais pagalbiniais metodais galima kurti pagalbinius objektus ir metodus, kurie kontroliuotų testuojamų komponentų gyvavimo ciklą.
- Testų rinkiniai sudaro testų paketą, kuris labai panašus į pagrindinės programos paketą, todėl nereikia mokytis naujų įrankių ar testų projektavimo ir kūrimo technikų
- Instrumentavimo karkasas leidžia kontroliuoti testus bei analizuoti aplikaciją.
- Netikrų objektų versijos dažniausiai naudojamiems Android sistemos objektams.
- SDK įrankiai kūrimui ir testavimui yra prieinami per Eclipse IDE su ADT įskiepiu, ir taip pat pasiekiami per komandinės eilutės sąsają naudojant su kitomis kūrimo platformomis. Šie įrankiai iš testuojamos programos projekto gautą informaciją panaudoja automatiškai, kuriant diegimo skriptus, manifesto failus bei katalogų struktūrą testavimo paketams.
- SDK taip pat suteikia „monkeyrunner“, tai testavimo API su Python programa ir „Application Exerciser Monkey“, komandinės eilutės įrankis stresiniam grafinės sąsajos testavimui, imituojantis pseudo atsitiktinius įrankio įvykius.

Testavimo karkaso apibendrinta diagrama pateikiama 2 paveikslėlyje.



2 pav. Android testavimo karkaso schema [13]

2.2.1 Vienetų testai

Vienetų testai - tai programinės įrangos testai, kuriuos programuotojai parašė naudodamiesi programavimo kalba ir jie turėtų būti parašyti izoliuotiems komponentams ir taip pat suteikti galimybę juos pakartotinai įvykdyti. [13] Todėl vienetų testai ir netikri objektai dažniausiai yra naudojami kartu. Naudojant netikrus objektus izoliuojami testuojami vienetai nuo priklausomybių, stebimos sąveikos ir taip pat suteikia galimybę testus leisti norimą kiekį kartų. Pavyzdžiui, jeigu testas ištrina kokius nors duomenis iš duomenų bazės, tai iš tikro duomenų nereikia ištrinti, nes kitą kartą leidžiant testus jų neras [32].

JUnit yra Android vienetų testų standartas. Tai paprastas atviro kodo karkasas automatiniam vienetų testavimui sukurto Erich Gamma ir Kent Beck [32].

Android (iki Android 4.0) naudoja JUnit 3. Ši versija nenaudoja anotacijų bet naudoja vidinį stebėjimą testų nustatymui.

Tipiškas JUnit testas atrodo taip:

```
package org.achartengine.renderer;

import android.test.*;

public class BasicStrokeTest extends AndroidTestCase {

	protected transient org.achartengine.renderer.BasicStroke test =
null;

	public void setUp() throws Exception {
		test = new org.achartengine.renderer.BasicStroke(
				android.graphics.Paint.Cap.BUTT,
				android.graphics.Paint.Join.BEVEL, 0.1f, new
float[] { 0.1f },
				0.1f);
	}

	public void tearDown() throws Exception {

	}

	public void testPreconditions() {
	}

	public void testGetCap_Cap_OCL_1() {
		try {
			test.getCap();
		} catch (Exception e) {
			fail(e.getMessage());
		}
	}

	public void testGetIntervals_Float0_OCL_1() {
		try {
			test.getIntervals();
		} catch (Exception e) {
			fail(e.getMessage());
		}
	}
}
```

Toliau detalizuojama iš kokių komponentų susideda testinis atvejis.

setUp() metodas:

Šis metodas iškviečiamas testo eigos pradžioje. Perrašomas tuomet, kai norima sukurti objektus ar inicializuoti kintamuosius, kurie bus naudojami pačiuose testuose. Šis metodas kviečiamas prieš kiekvieną testą.

tearDown() metodas:

Šis metodas iškviečiamas testo eigos pabaigoje. Perrašomas tuomet, kai reikia atlaisvinti resursus naudotus inicializacijoje ar testuojant. Šis metodas iškviečiamas po kiekvieno testo. Kaip pavyzdys šiame metode galima atlaisvinti prisijungimą prie duomenų bazės ar tinklo. Taip pat, jeigu testo metu buvo naudojami išoriniai arba riboti resursai, tai taip pat juos reikia atlaisvinti šiame metode.

testPreconditions() metodas:

Nėra jokio būdo ištestuoti prieš sąlygas testų, kurie atrandami stebėjimo tvarka ir negalima užtikrinti jų vykdymo tvarkos. Todėl siūloma sukurti testPreconditions() metodą, kurio paskirtis testuoti prieš sąlygas. Gera praktika testą ir testPreconditions() metodą laikyti kartu.

Aktualūs testai:

Visi vieši metodai kurių vardas prasideda žodžiu „test“ yra suprantamas kaip testas. JUnit 3, skirtingai negu JUnit 4, nenaudoja anotacijų pažymėti testams bet naudoja stebėjimą testams nustatyti. Tačiau yra kelios anotacijos naudojamos Android testavimo karkase, tai @SmallTest, @MediumTest ir @LargeTest, bet jie neapjungia metodų į vieną testą. Šios anotacijos leidžia organizuoti testus į skirtingas kategorijas. Toks testų sutvarkymas leidžia paleisti kiekvienos kategorijos testus atskirai. [32]

Nusistovėjusi taisyklę testų vardus sudaryti panaudojant daiktavardžius bei sąlygas, kuriomis yra testuojama. Pavyzdžiui testValues(), testConversionError() yra rekomenduojami vardai.

Testai turi išbandyti išimtis ir blogas reikšmes, o ne tik testuojant naudoti gerus atvejus. [32]

Testų vykdymo metu testų sąlygos, šalutinis efektas ar metodo grąžinami rezultatai turi būti palyginami su tikėtinais rezultatais. Siekiant palengvinti šias operacijas, JUnit pateikia daug assert* metodų, kurie palygina aktualius gautus rezultatus su tikėtinais ir iškviečia išimtis, jeigu sąlygos nėra įvykdytos. Tuomet testų vykdytojas apdoroja šias išimtis ir pateikia rezultatus vartotojui.

Metodai, kurie yra perrašyti iš JUnit ir palaiko daug skirtingų argumentų:

- assertEquals()
- assertFalse()
- assertNotNull()
- assertNotSame()
- assertNull()
- assertSame()
- assertTrue()
- fail()

Papildomai JUnit assert metodams, Android praplėtė ir pridėjo du papildomus assert metodus:

- MoreAsserts

• ViewAsserts

2.2.2 Netikri objektai

Netikri objektai yra skirti imituoti objektus ir naudojami vietoje realių objektų taip leidžiant izoliuoti vienetus. Paprastai, tai daroma siekiant užtikrinti, kad metodai būtų teisingai iškviesti, tačiau taip pat jie padeda atskirti testus nuo aplinkos, jog juos būtų galima paleisti savarankiškai ir pakartojamai [32].

Android testavimo karkasas palaiko keletą netikrų objektų, kurie yra labai naudingi rašant testus, tačiau norint, jog testus teisingai kompiliuoti reikalinga pridėti keletą priklausomybių.

Keletą klasių, kurias pateikia Android testavimo karkasas iš android.test.mock paketo:

• MockApplication
• MockContentProvider
• MockContentResolver
• MockContext
• MockCursor
• MockDialogInterface
• MockPackageManager
• MockResources

Beveik visi šios platformos komponentai, kad galėtų sąveikauti su Activity turi būti sukurti iškviečiant kurią nors iš šių klasių.

Tačiau tai nėra realiai realizuotos klasės bet kamščiai, kurių kiekvienas metodas generuoja UnsupportedOperationException ir jie gali būti išplėsti realizuojant realius netikrus objektus.

2.2.3 Eclipse ir kitų IDE palaikymas

JUnit yra pilnai palaikomas Eclipse programavimo aplinkos bei Android ADT įskiepio, kuris leidžia sukurti Android testavimo projektus. Be to galima paleisti testus ir analizuoti jų rezultatus nepaliekant IDE. Taip pat suteikia daug subtilesnių pranašumų; leidžiant testus iš Eclipse suteikiama galimybę juos derinti, jeigu jie nesielgia teisingai. Kituose programavimo aplinkose, kaip IntelliJ ar Netbeans taip pat turi integruotus Android įrankius, tačiau jie nėra oficialiai Google palaikomi. Jeigu nenaudojamas joks įrankis, tai galima kurti ir leisti testus naudojantis Ant įrankiu ir pateiktais skriptais [32].

2.3 Nagrinėjamos problemos

2.3.1 Android naudojama skirtinga platforma

Android savyje turi pagrindines bibliotekas, kurių funkcionalumas prieinamas ir Java programavimo kalbos pagrindinėse bibliotekose. Tačiau aplikacijos leidžiamos ne ant Java virtualios mašinos (VM), o naudojama nestandartinė virtuali mašina pavadinta Dalvik VM. [11] Kiekviena Android aplikacija pasileidžia kaip procesas Dalvik VM. Dalvik vykdo tik Dalvik vykdomuosius .dex formato failus. Ši mašina remiasi Linux branduolio pagrindinėmis funkcijomis, tokiomis kaip daugiagijiškumas ir žemo lygio atminties valdymu. [6]

Programos yra rašomos Java kalba ir sukompiliuojamos į mašininį kodą. Tuomet iš Java VM suderinamų .class failų programa konvertuojama į Dalvik suderinamus .dex failus ir tuomet paleidžiama. [6] Dėl šio skirtumo aplikacijos ir įrankiai suderinami su Java programomis ir palengvinantys programavimą Java kalba yra netinkami programuojant Android aplikacijas. Kadangi Android OS palyginus jauna, tai įrankių skirtų programuojantiems šiai sistemai skirtas aplikacijas yra mažai. Tokių įrankių kūrimas taip pat sudėtingas, kadangi sudėtingas arba dažnai nėra įmanomas pakartotinis panaudojimas jau esamų komponentų ir tenka visą funkcionalumą realizuoti patiems.

2.3.2 Objektų kūrimo problema

Kuriant mobilias aplikacijas Android OS sistemai naudojama Java programavimo kalba. [11] Java programavimo kalba yra objektinė, todėl reikalingi objektų sukūrimai. Dažniausiai prieš iškviečiant klasės funkciją reikia sukurti objektą. Objekto sukūrimas pasidaro sudėtingas, jeigu sukūrimui reikia tam tikrų parametrų (konstruktorius yra su parametrais) [14]. Todėl tam reikalinga realizuoti objektų konstruktorių paiešką ir objektų kūrimo metodiką. Galimas sprendimas, kurti tik tas klases, kurios turi numatytąjį konstruktorių t.y. tokį, kuris neturi parametrų. Objektų kūrimui palengvinti galima naudoti Eclipse SDK teikiamas priemones.

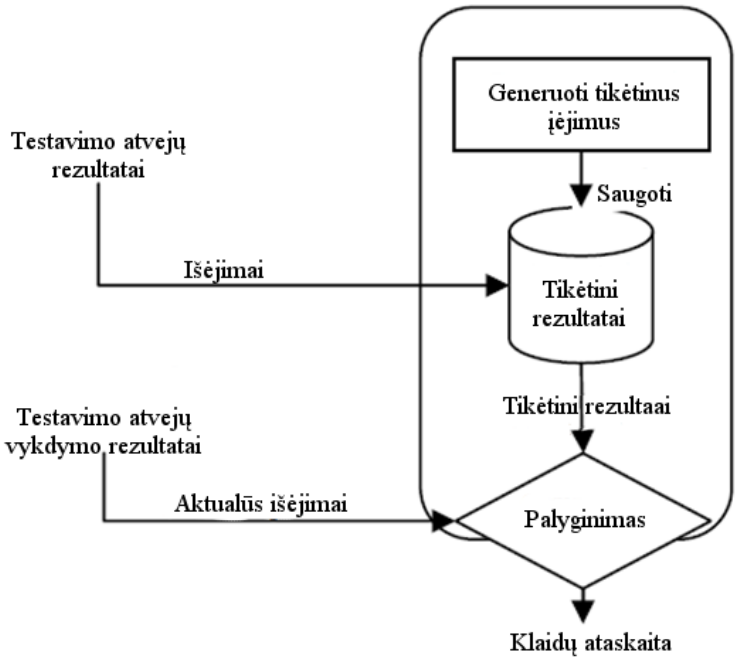
2.3.3 Orakulo problema

Turint testavimo duomenis iškyla problema, kaip patikrinti ar sugeneruotas testas tikrai yra teisingas ir grąžina laukiančius rezultatus, o pats testas yra sėkmingas ar nesėkmingas. Tai atlieka testavimo vykdymo įrankis, kitaip vadinamu orakulu. Testavimo orakulo problema yra plačiai žinoma programinės įrangos testavimo literatūroje ir siūlomi keli testavimo orakulo kūrimo metodai. [31]

Galimas vienas iš testavimo orakulo veikimo algoritmo [31]:

1) Generuojamas laukiamos išėjimo reikšmės
2) Išsaugojamas sugeneruotos reikšmės
3) Vykdomi testavimo atvejai
4) Lyginamos laukiamos ir dabartinės išėjimo reikšmės
5) Apsisprendžiama, ar testas sėkmingas ar ne.

Toks orakulo procesas vaizduojamas 3 pav.

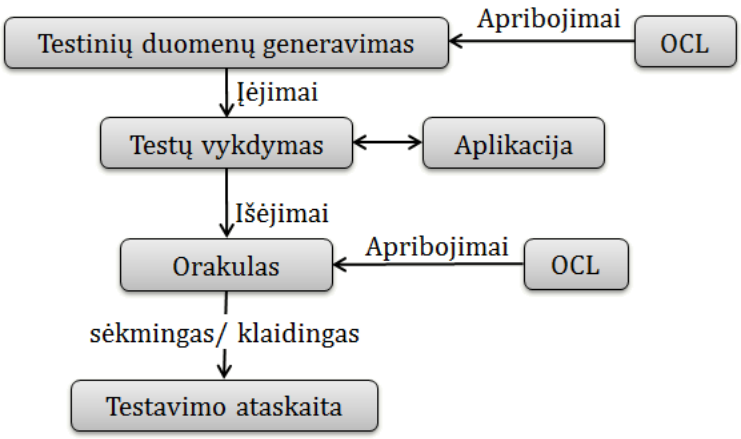


3 pav. Testavimo orakulo procesas [31]

Metodas, reikalaujantis vartotojo įsikišimo, tai kai vartotojas peržiūri sugeneruotus testus ir pažymi juos kaip tinkamus ir teisingus. Tuomet šių testų išėjimo reikšmės laikomos kaip laukiamos testuojamų komponentų reikšmės ir vykdant sugeneruotus testus, gautos naujos išėjimo reikšmės lyginamos su prieš tai išsaugotomis reikšmėmis, ir atradus nesutapimus laikoma, jog testas nesėkmingas. Šis orakulo veikimas remiasi regresinio testavimo principais.

Kitas orakulo būdas remiasi naudojantis OCL apribojimais. Metodas yra vykdomas naudojant sugeneruotus testinius duomenis, o gautas rezultatas yra lyginamas su post sąlygomis. Jeigu gautas rezultatas nepatenkina post sąlygos, tai laikoma, kad testas įvykdytas nesėkmingai ir yra klaida. Jeigu rezultato reikšmė patenkino post sąlygą, tai yra tikimybė, kad testas įvykdytas sėkmingas ir nėra klaidos realizacijoje. [26]

Šio testavimo principinė schema vaizduojama 4 pav.



4 pav. Testavimo principinė schema naudojant OCL apribojimus

Automatiškai generuoti laukiamus išėjimus galima panaudojant pasirinkimo lentelę (Decision table). Pasirinkimo lentelė yra programos reikalavimų reprezentacinis modelis. Šiame modelyje nurodomas sąlygos prie kurių gaunamas atitinkamas programos atsakas. Pasirinkimo lentelė susideda iš sąlygos skilties, kurioje aprašomos sąlygų kombinacijos, ir veiksmų skilties, kuri nusako programos atsaką, kai sąlyga patenkinama. Kiekviena eilutė lentelėje turi variantą, kaip unikalią sąlygos kombinaciją. [31] 1 lentelė vaizduoja tokią pasirinkimo lentelę.

1. Lentelė. Pasirinkimo lentelės pavyzdys [31]

Variantas	**Įėjimo parinkimas**			**Išėjimo parinkimas**		
	Įėjimo kintamasis	Įėjimo veiksmas	Būsena prieš testą	Tikėtini rezultatai	Tikėtinų išėjimai	Tikėtina būsena po testo
	....			...		

2.4 Vienetų testų generavimo metodai

Testų generavime sprendžiama problema, kaip ir pagal ką generuoti testinius duomenis. Tam literatūroje yra aprašoma nemažai generavimo metodų. Remiantis autoriais, kurie aprašo testų generavimą [18] [4] [35] bei mobilių aplikacijų testavimą [17], buvo pasirinkti trys testų generavimo būdai: atsitiktinis generavimas [18], generavimas pasinaudojant OCL apribojimais [2] [4] [26] bei generavimas remiantis genetiniu algoritmu [21] [4]. Algoritmų tinkamumas Android aplikacijoms testuoti bus analizuojamas atliekant tyrimą, po to kai šie metodai bus realizuoti projekto metu.

2.4.1 Atsitiktinio generavimo metodas

Atsitiktinio generavimo metodo metu automatiškai generuojami testų įėjimo duomenys, parenkant testinius atvejus atsitiktinai iš testuojamos programos įėjimo srities. Koncepcija ir realizacija yra palyginus paprasta su kitomis egzistuojančiomis technikomis [18].

Atsitiktinis testavimas gali būti naudojamas testuojant bet kokio tipo programas, kadangi pagrindiniai duomenų tipai galiausiai vis vien yra sveikieji, realūs skaičiai, eilutės ar bitai. Paprasčiausią eilutę galima sugeneruoti, tiesiog atsitiktinai generuojant srautą bitų ir paversti jį į eilutės tipo kintamąjį [35].

Atsitiktinio generavimo algoritmo metu kiekvienam metodui analizuojami įėjimai. Jeigu įėjimas - primityvus kintamasis, jis iškarto sugeneruojamas, jeigu objektas – ieškoma objekto konstruktoriaus ir suradus konstruktorių sugeneruojamas objektas. Testo orakulas leidžia patikrinti pagal metodo tipą: jeigu metodas grąžiną reikšmę, tai tikrinama, ar ta reikšmė grąžinama, jeigu ne – tuomet tikrinama, ar neįvyko išimtis (exception). Pagal sugeneruotus įėjimus ir testo orakulą suformuojamas testas testuojamai klasei.

Atsitiktinis generavimo metodas plačiai naudojamas, kadangi tai paprasčiausiai realizuojamas ir nėra reikalingas žmogaus įsikišimas. [18]

2.4.2 Generavimas panaudojant OCL apribojimus

OCL – tai OMG organizacijos pasiūlytas standartas, kuriuo užrašomi objekto apribojimai. Jos pagrindinis tikslas specifikuoti programas, kurios neturi stipraus matematinio pagrindo ir kurių negalima specifikuoti kitomis formaliosiomis kalbomis, pavyzdžiui kaip Z^2 [26]. Dabar OCL yra dalis UML standarto.

Pagrindiniai atvejai, kuriais galima naudoti OCL yra [2]:

- Užklausų kalba.
- Apibrėžti invariantus klasėms ir tipams klasių modelyje.
- Apibrėžti prieš ir po sąlygas operatoriams ir metodams.
- Specifikuoti žinutes ir veiksmus.
- Specifikuoti apribojimus operatoriams.
- Specifikuoti išvestines taisykles UML modelių atributams.

OCL apribojimams iliustruoti žemiau pateikiama klasė:

```
public class XYMultipleSeriesRenderer {
      public double setXAxisMin (int scale);
};
```

OCL leidžia aprašyti apribojimus šiuo atveju aprašant inv ir post sąlygas. Aprašymo pavyzdys anksčiau pateiktai klasei gali būti:

```
context XYMultipleSeriesRenderer::setXAxisMin(scale:Integer):Real
    inv: scale >= 0
    post: result >= 0
```

Post sąlygos naudojamos orakulo, o inv sąlygos naudojamos generuojant duomenis. Panaudojant OCL apribojimus, neatitinkami apribojimų sugeneruoti duomenys atmetami, taip atrenkami tinkami įėjimo testavimo duomenys [26]. Testų orakului taip pat galima panaudoti OCL apribojimus, pagal juos orakulas žino, kuriuos metodo rezultatus laikyti teisingais, o kuriuos – neteisingais.

Pagrindinis metodo privalumas, jog naudojantis OCL galima nesunkiai formaliai specifikuoti operatorius ir metodus, kuriais remiantis galima generuoti vienetų testus.

2.4.3 Genetiniu algoritmu paremtas metodas

Programos testavimas yra sunkus ir sunaudojantis daug laiko procesas. Tam skiriama apie 50% programų sistemos kūrimo resursų. Apkritai, programų testavimo tikslas yra sukurti minimalų kiekį testinių atvejų, kurie atskleidžia galimas klaidas kiek tai yra įmanoma. [34]

Genetiniai algoritmai (GA) ir jų sudarymo principas buvo pasiūlytas XX-ojo amžiaus 70-aisiais metais Hollando. Genetinis algoritmas ir jo veikimas yra pagrįstas evoliucijos, vykstančios gyvojoje gamtoje, t. y., natūraliosios atrankos proceso imitavimu. [4]

Šio algoritmo principiniai veiksmai remiantis [21] šaltiniu ir pritaikant testų generavimui:

1. [**Pradžia**] Sugeneruoti pradinių testų populiaciją
2. [**Tinkamumas**] Apskaičiuoti, kiek kodo padengia kiekvienas testas
3. [**Kryžminimas**] Sukurti naują populiaciją, atrenkant daugiausiai kodo padengiančius testus ir juos kryžminant tarpusavyje, pagal sugeneruotas įėjimo reikšmes metoduose.
4. [**Pakeitimas**] Tolimesniame darbe naudoti naujai sugeneruotą populiaciją.
5. [**Tikrinti**] Jei padengimo sąlyga tenkinama arba populiacijos kartų skaičius peržengtas, sustabdyti algoritmą ir grąžinti geriausią testą iš einamos populiacijos. Jeigu netenkinama – grįžti į žingsnį 2.

Naudojant genetinį generavimą gaunamas minimalus testinių duomenų kiekis, kuris apima kiek įmanoma daugiau klaidų. Nėra nereikalingų ir besidubliuojančių testinių duomenų.

2.5 Panašių įrankių lyginamoji analizė

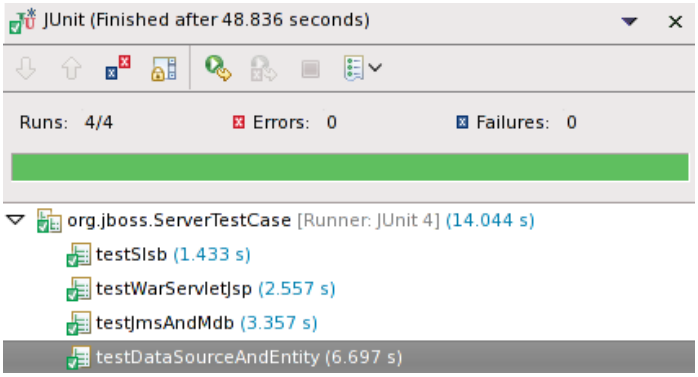
Šiuo metu vienintelis įrankis skirtas testuoti Android OS parašytoms aplikacijoms yra Android SDK palaikomas JUnit testų rašymas. Kitokių įrankių, kurių paskirtis automatiškai generuoti testus Android OS aplikacijoms surasti nepavyko, tikriausiai todėl, kad telefonai su Android OS ne taip seniai pradėjo populiarėti. Todėl buvo nuspręsta panagrinėti kitus testavimo įrankius, kurie skirti kitoms kalboms, kaip Java kalba parašytoms aplikacijoms testuoti, tačiau dėl skirtingų platformų jie Android aplikacijoms testuoti nėra pritaikyti. Tokių įrankių jau yra daugiau, todėl buvo pasirinkti populiaresni įrankiai, kurie nagrinėjami atskiruose skyreliuose.

2.5.1 JUnit

Tai standartinis testavimo karkasas Java kalbai. Anksčiau tai buvo testavimo karkasas, kuris vadinosi SUnit. Vėliau Kent'o komanda su Erich Gamma pritaikė Java kalbai ir buvo pavadinta JUnit. [29]

JUnit buvo sukurtas kaip karkasas, kad būtų galima rašyti automatinius, savipatikrinančius testus Javos kalboje, kurie JUnit yra vadinami testavimo atvejais. Testavimo atvejai grupuojami pagal panašumus ir yra gaunamas taip vadinamas testų rinkinys. JUnit taip pat suteikia galimybes tokius testus įvykdyti. Praleidus testus, rezultatas gaunamas suvestinė, kurioje nurodomi tie testai, kurie negrąžino tinkamo rezultato, t.y. nepraėjo. Jeigu visas testų rinkinys praeina sėkmingai tai tuomet rodoma tiesiog „OK" (Gerai). Testai yra rašomi kiekvienam programos vienetui savarankiškai, todėl jie gali būti paleisti dalimis arba visi iš karto.

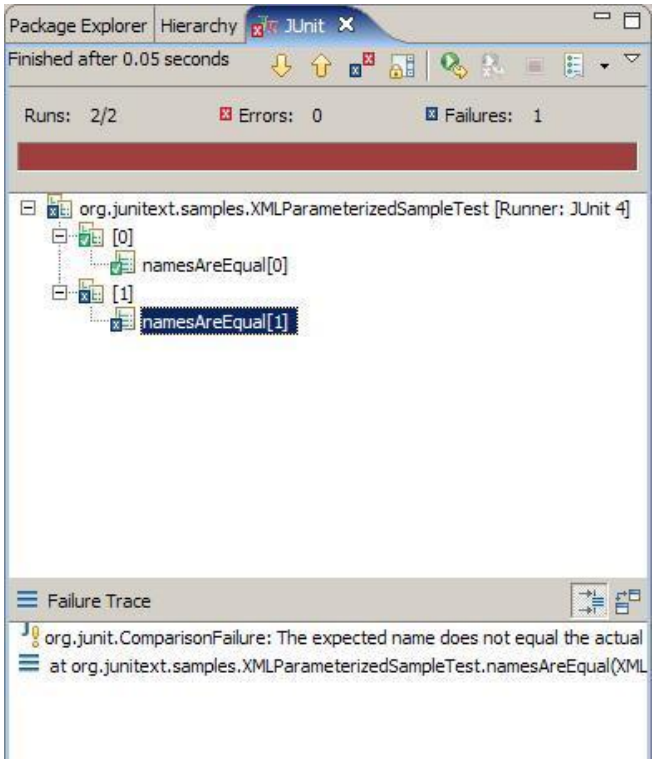
JUnit testų rinkinio sėkmingas įvykdymas vaizduojamas 5 pav.,



5 pav. JUnit testų rinkinio sėkmingas įvykdymas

nesėkmingo testų rinkinio įvykdymo atveju (6 pav.) rodoma kiek testų praėjo, keli testai yra su klaidomis ir kiek testų nebuvo sėkmingai įvykdyti. Paspaudus ant klaidingo testo, iškarto

įrankis parodo išsamesnę informaciją susijusią su klaida, bei tuo pačiu iš kontekstinio meniu, galima pasirinkti, jog įrankis atidarytų tą testo vietą, kurioje yra aprašytas šis testas.



6 pav. JUnit testų rinkinio nesėkmingas įvykdymas

JUnit privalumai:

- Paprastas naudoti
- Nemokamas
- Sugeneruoja nurodytai klasei testų karkasą
- Ataskaitoje aiškiai ir suprantamai pateikiami rezultatai
- Palaikomas standartiškai Android SDK

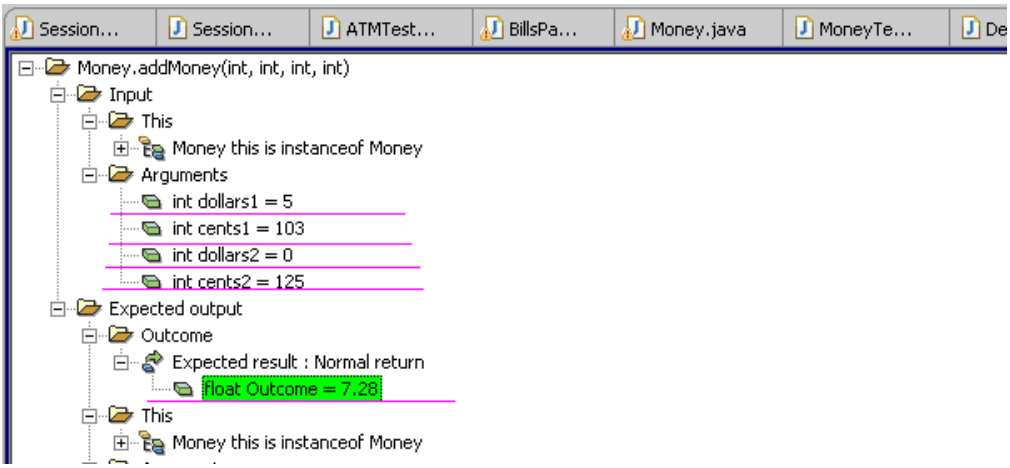
JUnit trūkumai:

- Automatiškai negeneruoja testuojamų duomenų. Vartotojas pats turi rašyti testus.
- Neturi išsamesnio testų vykdymo (orakulo)
- Neturi testų efektyvumo įvertinimo

2.5.2 Parasoft Jtest

Tai „Parasoft“ kompanijos sukurtas įrankis. Kaip teigia „Parasoft“ [27], tai įrankis, kuris integruoja plataus pritaikymo automatinius sprendimus, kurie pagerina programų kūrimą, komandos produktyvumą bei programos kokybę. Šis įrankis apima vienetų testo atvejų generavimą, statinę kodo analizę, vykdymo klaidų aptikimą ir kodo peržiūrą.

Vienetų testai generuojami nurodant norimą metodą, po to vartotojui pateikiamas grafinis langas, kuriame gali suvesti paduodamų parametrų reikšmes bei laukiamą rezultatą. Toks langas pavaizduotas 4 pav.



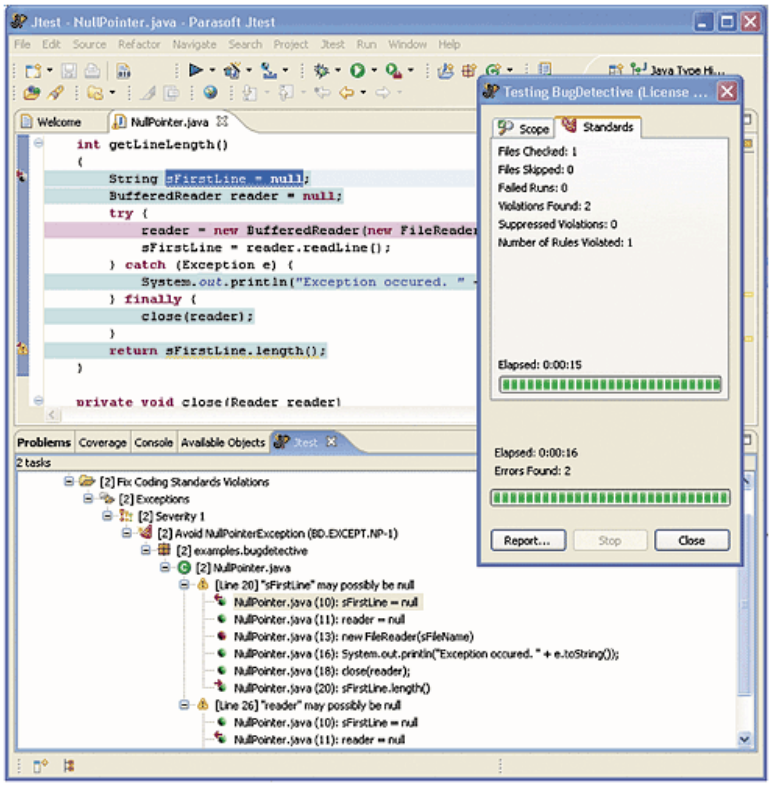
7 pav. JTest parametrų bei rezultato įvedimo langas

Pasinaudojant JTest galima sugeneruoti didelį kiekį testų, tačiau jų rezultatams patikrinti reikalingas vartotojas. Vartotojui tikrinti didelį kiekį testų gali užimti daug laiko, bei yra nepatogus. [39]

ParaSoft's Jtest gali automatiškai generuoti vienetų testus Java klasėms. Kai specifikacija nėra pateikiama, tai Jtest gali automatiškai generuoti testinius įėjimus atlikdamas baltos dėžės testavimą. Kai specifikaciją pateikiama, tai Jtest atlieka juodos dėžės testavimą. [39]

JTest palaiko Struts, Spring ir Hibernate projektus ir taip pat standartinius J2EE EJB/JSP. Tačiau bandant vykdyti didesnius testų rinkinius su silpnesniu kompiuteriu, kaip Pentium4 1GB nešiojamas kompiuteris, JTest įrankis veikia nestabiliai. [16]

JTest atlieka taip pat statinę kodo analizė palygindamas vykdomą kodą ar programinį kodą su programavimo taisyklėmis, kurios reprezentuoja daugiau kaip 300 nuorodų. Kiekvienas toks JTest atrastas nukrypimas nuo taisyklės laikomas potencialiu programos defektu ir įspėjamas vartotojas, tokia analizė vaizduojama 8 pav. [27]



8 pav. Statinės analizės vykdymo suvestinė

JTester privalumai:

- Palaiko Spring, Struts ir Hibernate projektus [16]
- Automatinis vienetų testų generavimas [39]
- Statinė kodo analizė [8]
- Vykdymo klaidų aptikimai, kaip atminties praradimai, išimčių aptikimas. [27]

JTester trūkumai:

- Mokamas
- Nėra funkcionalesnio testų vykdymo (orakulo) [39]
- Reikalingi nemaži kompiuterio resursai didesniems testams vykdyti [16]
- Nepalaiko Android OS

2.5.3 JTestCase

Tai įrankis papildantis JUnit testavimo karkaso galimybes ir suteikia būdą formalizuoti testavimo kodą. Standartiškai naudojant JUnit reikia kiekvienam testavimo atvejui kurti testavimo kodą bei keisti testavimo duomenis. JTestCase suteikia galimybę atskirti testavimo duomenis nuo testavimo kodo. Visi testavimo atvejai daugybės testų saugomi viename XML faile ir gali būti lengvai įkeliami į testus pasinaudojant JTestCase API. [15] Testavimo duomenų aprašas XML formatu vaizduojamas 9 pav.

```
...
<method name="method_name">
 <test-case="test_sum_two_positive_int">
        <params>
          <param_name="addend_1" type="int">1<param>
          <param_name="addend_2" type="int">2<param>
        </params>
        <asserts>
          <assert_name="result" type="int" action="EQUALS">3<assert>
        </asserts>
 </test-case>
</method>
...
```

9 pav. JTestCase įrankio testinių duomenų aprašas

Tokių duomenų panaudojimas vaizduojamas 8 pav.

```
...
TestCaseInstance testCase = (TestCaseInstance) jtestcase.getTestCasesInMethod("myMethod").get(0);
HashMap params = testCase.getTestCaseParams();
int addend_1 = ((Integer) params.get("addend_1")).intValue();
int addend_2 = ((Integer) params.get("addend_2")).intValue();
int result = myCalculator.sum(addend_1,addend_2);
boolean success = testCase.assertTestVariable("result", new Integer(result));
assert("calculator sum failed", success);
...
```

10 pav. JTestCase įrankio testavimo duomenų panaudojimas JUnit testuose

JTestCase privalumai:

- Nemokamas
- Palengvina testinių duomenų saugojimą
- Atskiria testinius duomenis nuo testų
- Gali būti panaudotas kuriant JUnit testus su Android SDK

2.5.4 JUB (JUnit test case Builder)

Tai JUnit testinių atvejų generatorius kartu su daug IDE specifinių papildymų. Jais galima naudotis IDE aplinkoje ir jie saugo sugeneruotą testo atvejo kodą programos teksto saugykloje, kuri yra valdoma IDE.

Kodo generavimas JUB įrankyje yra truputį platesnis negu kituose generatoriuose. Jis atsižvelgia į daugialypius konstruktorius ir perkrautus metodus. Šis įrankis bando apimti visas būtinas svarbias formuluotes ir sudėti teisingas ištestuotų metodų žymes bei išimčių testavimą / valdomąjį kodą į testų metodų kūnus. Jis taip pat leidžia testų kodo generavimą apsaugotiems ir numatytiems metodams (šiuo metu tiktai, kai testo atvejo klasė yra tame pačiame pakete kaip testo klasė). [22]

JUB privalumai:

- Nemokamas
- Nepriklauso nuo IDE
- Platesnis kodo generavimas (daugialypiai konstruktoriai, perkrauti metodai)

JUB trūkumai:

- Nepritaikytas Android SDK aplikacijoms.
- Lyginant su TestGen4J ir JCrasher generuojami testai neefektyviausi [36]

2.5.5 JCrasher

JCrasher yra automatinis atsparumo testavimo įrankis Java kodui. JCrasher nagrinėja Javos klases ir konstruktorių kodo fragmentus ir sukuria skirtingų tipų testus išoriniams programos metodams su atsitiktiniai duomenims. JCrasher bando aptikti klaidas, kurios sukelia programos lūžimus, t.y. iškviečiamos nenumatytos vykdymo išimtys (exception). [5]

Nors apibendrintai atsitiktinio testavimo būdas turi daug apribojimų, jis taip pat turi privalumą, nes yra visiškai automatinis: jam nereikia jokios priežiūros, išskyrus testų atvejus, kurie nepraėjo, tuomet reikalingas tiesioginis patikrinimas

Lyginant su JTest, JCrasher sugeneravo daug mažesnį kiekį bereikalingų testų, ~50%, kai JTest bereikalingų testų buvo apie ~90%. [33]

JCrasher privalumai:

- Nemokamas
- Integruojamas į Eclipse IDE
- Lyginant su TestGen4J ir JCrasher generuojami testai efektyvesni [36]
- Sugeneruoja testus JUnit karkasui
- Apibrėžia euristikas, kuriomis nusako ar Java išimtys (exception) turi būti laikomas programos klaida ar tiesiog buvo pažeistos programos kodo įvedimo sąlygos
- Palaiko efektyvų ankstesnių testų sukurtos būsenos grąžinimą į pradinę padėtį.

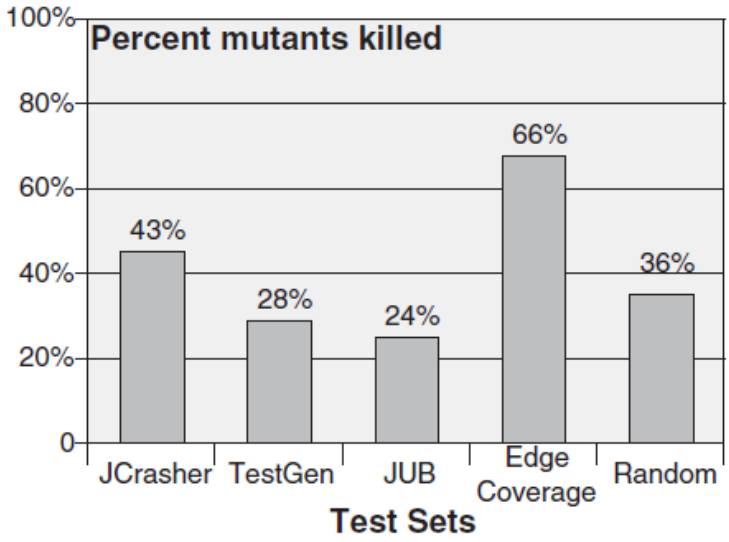
JCrasher trūkumai:

- Nepritaikytas Android SDK aplikacijoms.
- Tai įrankis tikrinantis tik programos atsparumą

2.5.6 TestGen4j

TestGen yra atviro kodo įrankių kolekcija, kuri automatiškai generuoja vienetų atvejų testus. Pirmo leidimo TestGen komponentas yra TestGen4J. TestGen4J automatiškai generuoja JUnit testų atvejus iš Java klasės failų ar programinio kodo. Generuojant testus dėmesys skiriamas ribinėms metodų įvedimo argumentų reikšmėms. Tam naudojamas vartotojo konfigūruojamas XML failas, kuris nusako kiekvieno duomenų tipo ribines sąlygas. Testiniam kodui atskirti nuo testinių duomenų naudojamas JTestCase

TestGen4J palyginimas su JCracher ir JUB vaizduojamas 10 pav.



11 pav. Vaizduojamas TestGen4J, JCracher ir JUB palyginimas [36]

TestGen4 privalumai:

- Nemokamas
- Vartotojas gali nurodyti ribines sąlygas
- Testiniai duomenys atskirti nuo testų

TestGen4 trūkumai:

- Nesuderinamas su Android SDK
- Mažas funkcionalumas

2.5.7 Įrankių savybių palyginimas

Nagrinėtų automatinių testavimo įrankių savybių palyginimas vaizduojamas 2 lentelėje.

2. Lentelė. Programos savybių palyginimas

Palyginimo kriterijai	**JUnit**	**Parasoft JTest**	**JTestCase**	**JUB**	**JCrasher**	**TestGen4J**	**AATest**
Testo karkaso sugeneravimas	+	+	-	+	+	+	+
Testinių duomenų generavimas	-	+	-	+	+	+	+
Testų vykdymas (orakulas)	+	+	-	-	-	-	+
Testo įvykdymo ataskaitos generavimas	+	+	-	-	-	-	+
Kodo padengimo skaičiavimas	-	+	-	-	-	-	+
Testo efektyvumo įvertinimas	-	+	-	-	-	-	+
Integruojamas į Eclipse IDE	+	+	+	+	+	-	+
Pritaikytas Android SDK	+	-	+	-	-	-	+
Testavimo duomenys atskirti nuo testų	-	-	+	-	-	+	-
Nemokamas	+	-	+	+	+	+	+

Iš lentelės matome, kad daugiausiai savybių turi Parasoft kompanijos produktas JTest. Tai vienas iš populiaresnių produktų rinkoje, tačiau turi kelis trūkumus, didžiausias yra tai, jog įrankis komercinis ir mokamas bei nėra suderinamas su Android aplikacijoms. Kiti nagrinėti įrankiai yra nemokami, tačiau atitinkamai yra labiau specializuoti ir dažniausiai kaip praplėtimas JUnit karkasui. Planuojamas automatinių testų generavimo įrankis lentelėje vaizduojamas santrumpa AATest.

2.6 Testų efektyvumo įvertinimo metodai

Vienas iš įrankio dalių yra turimo testo efektyvumo įvertinimas, taip pat efektyvumo įvertinimas yra reikalingas ir atliekant, projekto metu realizuotų vienetų testų generavimo metodų tyrimą. Šioje dalyje iškyla problema, kokias būdais nustatyti turimo testo kokybę. Tam nagrinėjami du būdai: pagal kodo padengimą ir pagal klaidų aptikimą. Šie būdai nagrinėjami detaliau atskiruose skyreliuose.

2.6.1 Pagal kodo padengimą

Testai gerina kodo kokybę, bet tik toms kodo dalims, kurioms tie testai yra vykdomi. Tam ir yra reikalingas kodo padengimas, nes jis parodo, kurios kodo vietos nėra testuojamos ir taip galima sužinoti, kiek vienas ar kitas testas yra geras ir padengia kuo daugiau kodo. [28]

Vienetų kodo testų padengimas [3] yra vienas iš svarbiausių testuojamos programinės įrangos metrikų ir dauguma kuriamų programuotojų komandų reikalauja užtikrinti bent 85% testuojamos programos padengimo [38]

Šiuo atveju įvykdomas testas, kurio efektyvumą norima įvertinti ir suskaičiuojama kiek ir kokias kodo eilutes tas testas padengia. Šis įvertinimas dažniausiai pateikiamas procentais. 100% - padengiama pilnai, mažesnis procentas reiškia, kad padengiama mažiau testuojamo kodo. Kurios kodo eilutės padengiamos parodoma grafiškai. Kodo padengimo pavyzdys vaizduojamas 12 paveikslėlyje. Raudona spalva pažymėtos tos kodo eilutės, kurios nebuvo vykdytos, o žalia spalva – kurios buvo įvykdytos.

Kodo padengimui skaičiuoti populiariausias nemokamas ir laisvai prieinamas įrankis yra EMMA [35] biblioteka.

```
public class Eclemma {

    public static void main(String[] args) {

        String str1 = "Eclemma";
        String str2 = new String("Ecl") + "emma";

        if(str1 == str2)
        {
            System.out.println("Reaally ??");
        }
        else
        {
            System.out.println("No Interning here :(");
        }
    }
}
```

12 pav. Testuojamo kodo padengimas vykdant testus

2.6.2 Pagal klaidų aptikimą

Kitas testų įvertinimo būdas, tai kiek klaidų gali aptikti vertinamas testas. Tokiam testo įvertinimui naudojamas vadinamas mutacinis testavimas.

Mutacinis testavimas - tai klaidomis paremta technika, kuri leidžia išmatuoti vienetų testų efektyvumą. Klaidos yra įterpiamos į programos kodą, taip sukuriant klaidingą programos versiją. Tokia programos versija vadinama mutantu. Tokie mutantai sukuriami iš

originalios programos kodo, panaudojant mutacijos operatoriuos, kurie aprašo programinio kodo sintaksinius pasikeitimus. [24]

Dažniausi Java metoduose mutaciniai operatorių pakeitimai [23] vaizduojami 3 lentelėje.

3. Lentelė. Java metodo lygio mutaciniai operatoriai [24]

Grupė	**Operatorius**	**Paaiškinimas**
Aritmetinės operacijos	AOR	Aritmetinės operacijos pakeitimai
	AOI	Aritmetinės operacijos įterpimas
	AOD	Aritmetinės operacijos ištrynimas
Sąlygos operacijos	COR	Sąlygos operacijos pakeitimas
	COI	Sąlygos operacijos įterpimas
	COD	Sąlygos operacijos ištrynimas
Loginės operacijos	LOR	Loginės operacijos pakeitimas
	LOI	Loginės operacijos įterpimas
	LOD	Loginės operacijos ištrynimas
Priskyrimas	ASR	Priskyrimo operacijos pakeitimas

Metodo esmė, jog yra sukuriamos originalios programos variacijos vadinamos mutantais su atitinkamai pakeistais operatoriais. Abi programos – originalioji ir mutantas testuojamos naudojant tą patį testą, kurio efektyvumą norime nustatyti. Jei testavimo rezultatai skiriasi, tai toks mutantas atmetamas, reiškia testas tokį mutantą nužudo. Jei mutantas neatmetamas, tai testiniai duomenys nėra pakankami klaidos atskleidimui ir mutanto neatskiria nuo originalios programos. Kartais neįmanoma atmesti mutanto, kadangi neįmanomi testiniai duomenys, kurie atskleistų naujai įvestą klaidą, toks mutantas vadinamas ekvivalenčiu. Tokio metodo rezultatas parodo testinių atvejų pilnumą arba kiek procentų mutantų buvo nužudyta.

Testo efektyvumas galima įvertinti procentine išraiška:

Įvertinimas = 100*D/(N-E), kur D – nužudyti mutantai, N – Sugeneruoti mutantai, E-ekvivalentūs mutantai. [24]

Mutaciniam testavimui galima naudoti laisvai prieinamą ir nemokamą dviejų universitetų sukurtą įrankį µJava (muJava) [24], kuris yra skirtas atlikti mutacinį testavimą Java programoms.

2.7 Išvados

1. Šiuo metu išaugo išmaniųjų telefonų pardavimai. Iš visų parduodamų išmaniųjų telefonų apie 50% parduodami su Android operacine sistema.
2. Tobulėjant išmaniesiems telefonams ir jų techninėms galimybėms, kuriama programinė įranga tampa sudėtingesnės ir funkcionalesnės, tačiau kokybės problema vis dar išlieka skaudžia programinės įrangos kūrimo dalimi.
3. Šiuo metu vienintelis įrankis skirtas testuoti Android OS parašytoms aplikacijoms yra Android SDK palaikomas JUnit testų rašymas. Kitokių surasti nepavyko, tikriausiai todėl, kad telefonai su Android OS ne taip seniai pradėjo populiarėti.
4. Pagrindinės nagrinėjamos problemos yra Android naudojama skirtinga platforma, atsitiktinis objektų kūrimas, testavimo orakulas, testinių duomenų generavimas bei sukurtų testų efektyvumo įvertinimas.
5. Vienetų testams generuoti pasirinkti trys metodai: atsitiktinis metodas pasirinktas dėl paprastumo ir dėl to, jog nereikalingas vartotojo įsikišimas; genetinis - nes gaunamas minimalus testinių duomenų kiekis, kuris apima kiek įmanoma daugiau klaidų ir nereikalauja vartotojo įsikišimo; generavimas remiantis OCL apribojimais - jog vartotojas gali nesunkiai individualizuoti testų generavimą pagal testuojamą aplikaciją. Šių metodų efektyvumas bus įvertinamas ištyrus projekto metu realizuotus šiuos metodus.

3. PROJEKTINĖ DALIS

3.1 Sistemos paskirtis

Šios sistemos pagrindinis tikslas yra palengvinti Android OS aplikacijų testavimą programų kūrėjams ir testuotojams. Tuo pačiu sukurtas testavimo įrankis turi būti patogus ir efektyvus. Jis ne tik generuotų automatinius testus, tačiau kartu būtų ir testų valdymo priemonė. Šiuo įrankiu naudotųsi įmonės ar asmenys, kuriantys Android OS aplikacijas. Šis įrankis jiems leistų geriau ištestuoti savo kuriamas aplikacijas ir taip pat sutaupyti testavimo kaštus, nes nereikėtų vienetų testų rašyti rankomis.

Kiti šio projekto tikslai yra išnagrinėti testavimo duomenų generavimo algoritmus bei aplikacijos testavimo (orakulo) metodus, bei juos įgyvendinti kuriamoje sistemoje.

Sistema gebės:

- Generuoti testinius duomenis, pagal įgyvendintus metodus
- Vykdyti testus
- Valdyti testus
- Analizuoti ir pateikti testavimo rezultatus
- Analizuoti testų efektyvumą

3.2 Sistemos funkcijos ir reikalavimai sistemai

Šio įrankio funkcijos:

- Generuoti testus – skirtas sugeneruoti testus, pagal vieną ar kitą generavimo metodą.
- Tvarkyti testus – skirtas testų tvarkymui, tai yra šalinti, redaguoti.
- Vykdyti testus – vykdyti sugeneruotus testus.
- Įvertinti testus – skirta testų efektyvumui skaičiuoti, tai yra paskaičiuojamas kiek programos kodo eilučių padengia sugeneruoti testai.
- Sugeneruoti ataskaitą – skirta ataskaitos sugeneravimui, kurioje pagal vartotojo nurodytus kriterijus yra pateikiama testų ataskaita.

Funkciniai reikalavimai sistemai:

- Leidžiama pasirinkti generavimo metodą
- Leidžiama pasirinkti, kurioms klasėms generuoti testus
- Leidžiama įvesti testų kiekį

- Negeneruoti testų, jeigu programos kodas su kompiliavimo klaidomis , kita kalba parašyta arba tai ne Android aplikacijos
- Vartotojui parodyti informacinį pranešimą, kai baigiamas generavimas
- Vartotojui leidžia pasirinkti, jog netinkamus testus pašalintų arba perkeltų į kitą katalogą
- Vartotojui parodoma suvestinė po atrinkimo (kiek testų tenkino kriterijų ir kiek netenkina)
- Įrankis leidžia sugeneruotiems testams atlikti įvertinimą
- Įrankis pateikia įvertinimo suvestinę
- Įrankis pateikia išsamią ataskaitą apie testus
- Įrankis leidžia įvykdyti testus
- Po testų vykdymo pateikti suvestinę
- Galimybė pašalinti testus
- Galimybė vartotojui redaguoti testus
- Galimybė peržiūrėti testus sąrašu

Nefunkciniai reikalavimai sistemai:

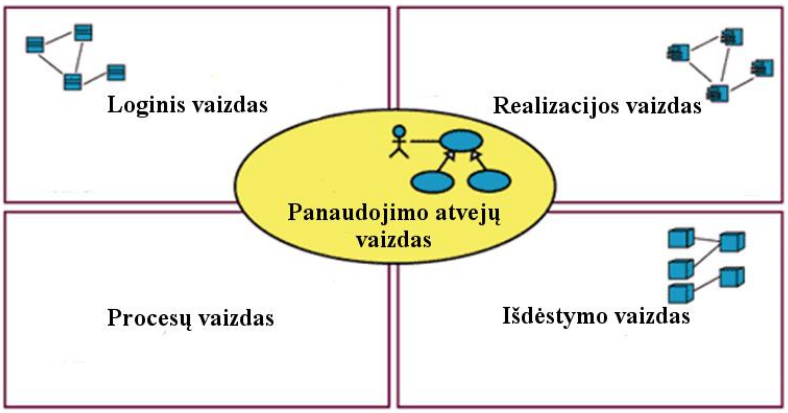
- Paprasta, greitai perprantama, intuityvi ir netrukdanti vartotojo sąsaja.
- Sąsaja turi atitikti Eclipse rekomendacijas
- Įrankio navigacija atitinka Eclipse standartus
- Visi terminai anglu kalba
- Kompiuterio resursai turi būti efektyviai panaudojami
- Testų generavimo laikas turi būti kuo trumpesnis
- Įrankis turi veikti įvairiose OS (kuriose veikia Eclipse IDE)
- Įrankio palaikymas turi būti kuo paprastesnis
- Įrankio atnaujinimas turi būti atliekamas per Eclipse atnaujinimo vedlį
- Galimybė paprastai praplėsti įrankio galimybes
- Laikytis standartinių Eclipse įskiepio saugumo reikalavimų
- Naudoti komponentus tik iš oficialių šaltinių
- Įrankiui taikoma GNU GPL licenzija

3.3 Architektūra

Šiame dokumente sistemos architektūra pateikiama keliais vaizdais: panaudojimo atvejų vaizdu, procesų vaizdų, dinaminiu vaizdu ir išdėstymo. Šie vaizdai yra pateikiami

kaip Rational Rose modeliai naudojant unifikuota modeliavimo kalba (UML). Sistemos architektūra pateikta remiantis RUP (Rational Unified Process) rekomendacijomis.

Šių vaizdų iliustracija ir išdėstymas pateikiamas 13 paveikslėlyje.



13 pav. Sistemos architektūros pateikimo vaizdai

- Panaudojimo atvejų vaizdas (panaudojimo atvejų diagrama)
- Loginis vaizdas (paketai ir klasių diagramos)
- Procesų vaizdas (būsenų, veiklos, sekų, bendradarbiavimo diagramos)
- Išdėstymo vaizdas (išdėstymo diagrama)

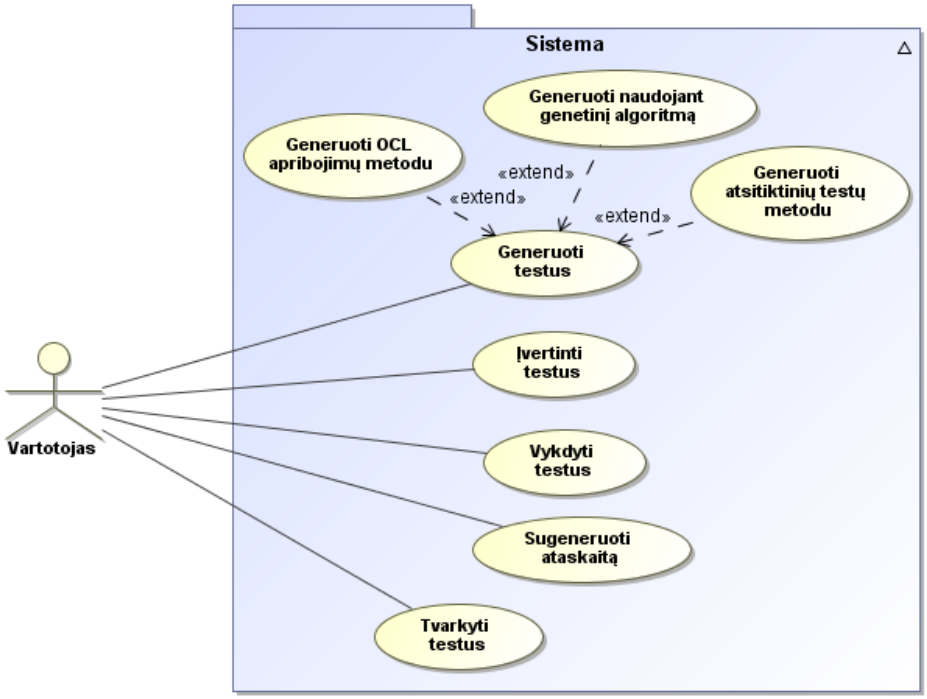
3.3.1 Architektūros tikslai ir apribojimai

Architektūrinius sprendimus įtakojantys reikalavimai ir apribojimai:

- Sistemos turi būti suprojektuota taip, kad ją galima būtų lengva išplėsti.
- Sistema turi būti suprojektuota taip, kad lengvai eitų pridėti naujus generavimo metodus
- Kadangi Android SDK oficialiai palaiko Eclipse IDE, todėl įrankis turi būti kuriamas kaip įskiepis Eclipse IDE.
- Įrankis turi veikti įvairiose OS (kuriose veikia Eclipse IDE)
- Įrankio įdiegimas ir atnaujinimas turi būti atliekamas per Eclipse atnaujinimo vedlį
- Kuriant sistemos sąsaja turi būti laikoma Eclipse GUI kūrimo rekomendacijų:
- Sudarant sistemos architektūra, turi būti atsižvelgta į būtinas programos vykdymo charakteristikas.

3.3.2 Panaudojimo atvejų vaizdas

Kuriamo įrankio panaudojimo atvejai pateikti 14 paveikslėlyje.

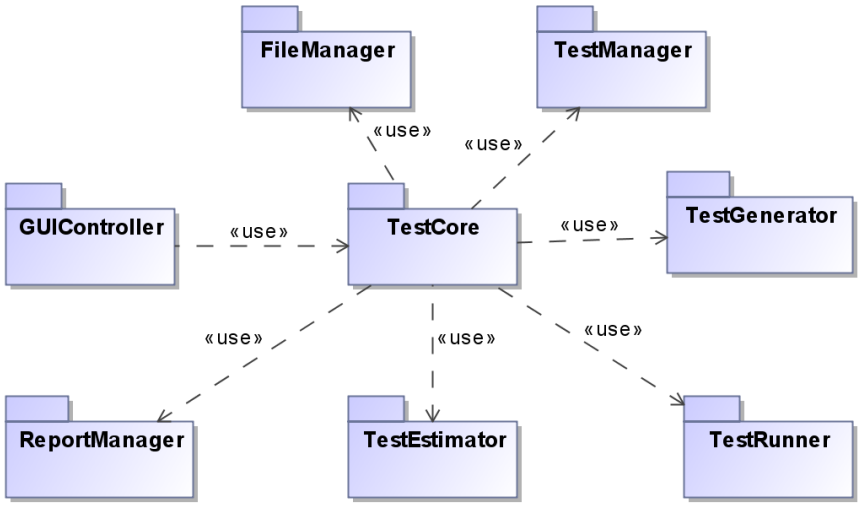


14 pav. Automatinių testų generavimo įrankio panaudojimo atvejų diagrama

3.3.3 Sistemos statinis vaizdas

Šis skyrius aprašo kuriamos sistemos loginę struktūrą. Pateikia sistemos išskaidymą į paketus ir juos sudarančias klases.

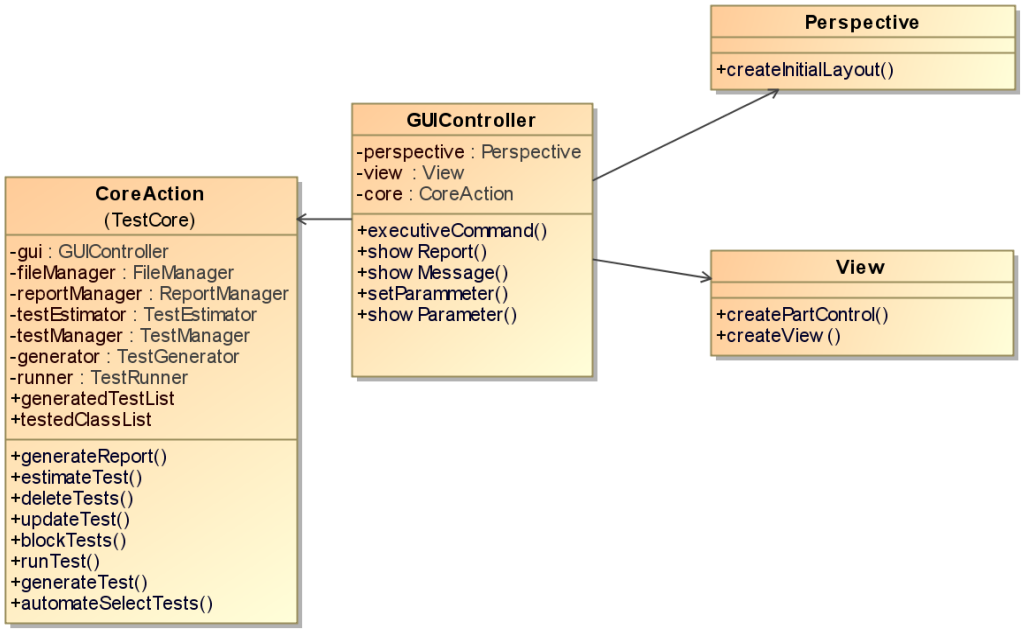
Sistemos suskaidymas į paketus vaizduojamas 15 paveikslėlyje.



15 pav. Automatinių testų generavimo įrankio paketų diagrama

GUIController paketas

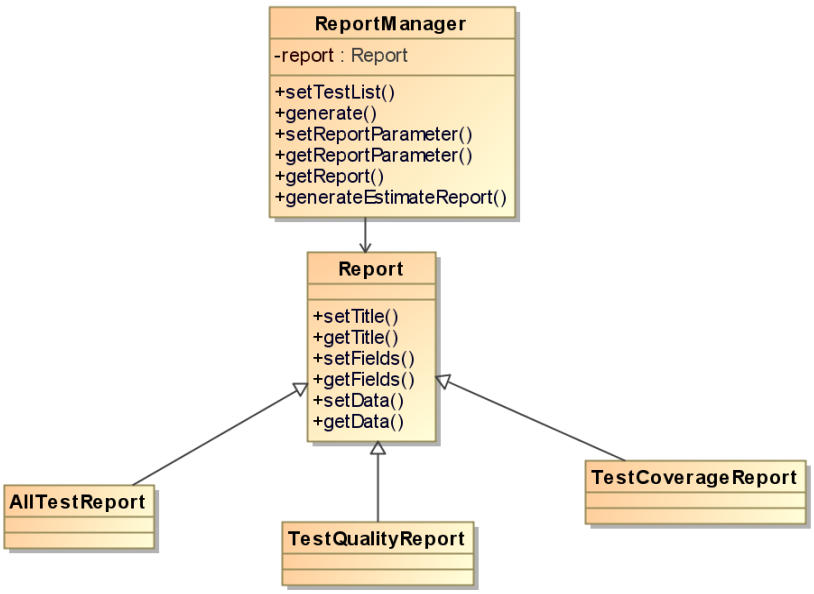
Šis paketas skirtas bendravimui tarp vartotojo ir įrankio. Taip pat šiame pakete bus naudojami Eclipse SDK reikalingi metodai. Šio paketo klasių diagrama pateikiama 16 pav.



16 pav. GUIController paketo klasių diagrama

ReportManager paketas

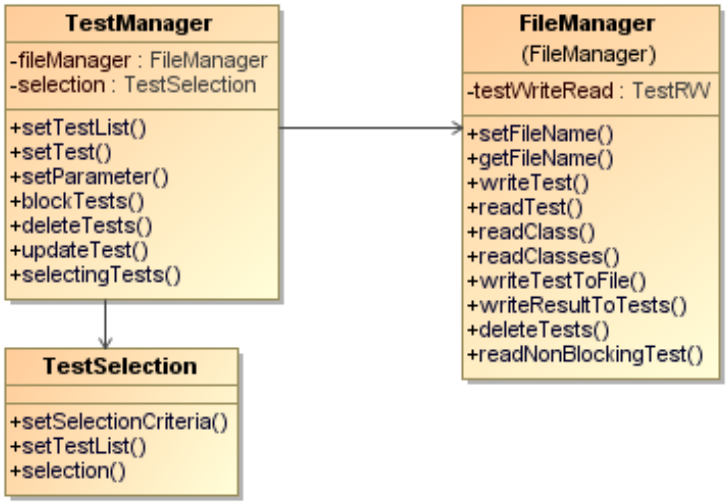
Šis paketas skirtas generuoti įvairias ataskaitas. Šio paketo pagalba bus galima sugeneruoti visų testų ataskaitą, testo aptinkamų klaidų ataskaitą ir testo padengimo ataskaitą. Šio paketo klasių diagrama pateikiama 17 pav.



17 pav. ReportManager paketo klasių diagrama

TestManager paketas

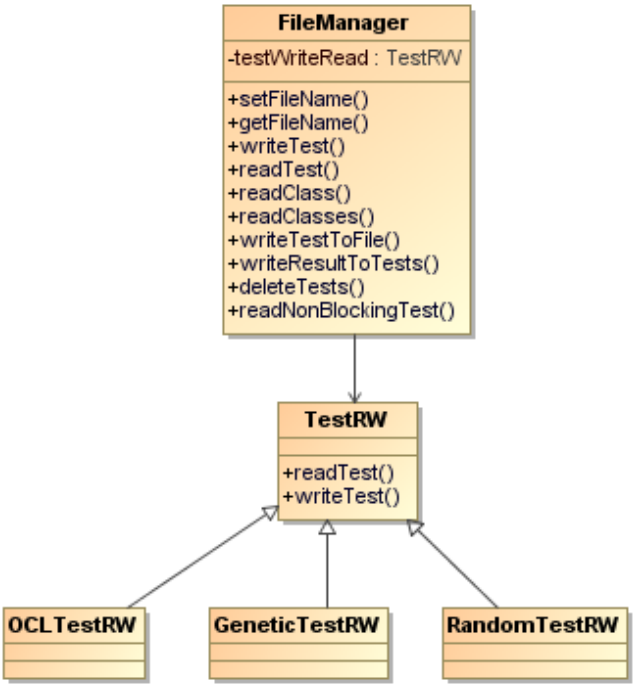
Šis paketas skirtas darbui su testais. Šio paketo pagalba tvarkomi testai, t.y. atnaujinami, pašalinami, laikinai užblokuojami, kad būtų nevykdom, taip pat automatinis testų atrinkimas pagal vartotojo pateiktus kriterijus. Šio paketo klasių diagrama pateikiama 18 pav.



18 pav. TestManager paketo klasių diagrama

FileManager paketas

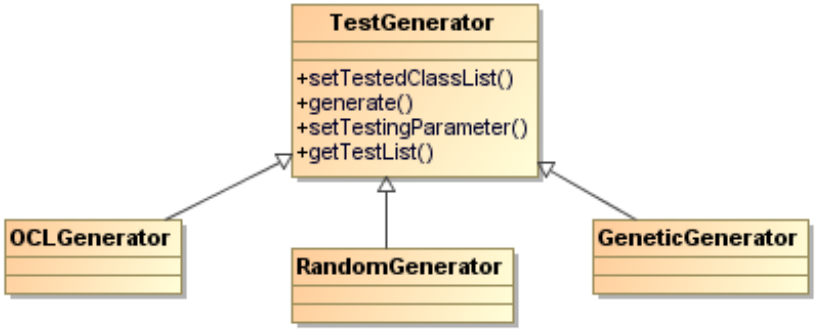
Šis paketas skirtas darbui su failais. Šio paketo pagalba bus nuskaitomos klasės, įrašomi sugeneruoti testai bei taip pat nuskaitomi testai. Šio paketo klasių diagrama pateikiama 19 pav.



19 pav. FileManager paketo klasių diagrama

TestGenerator paketas

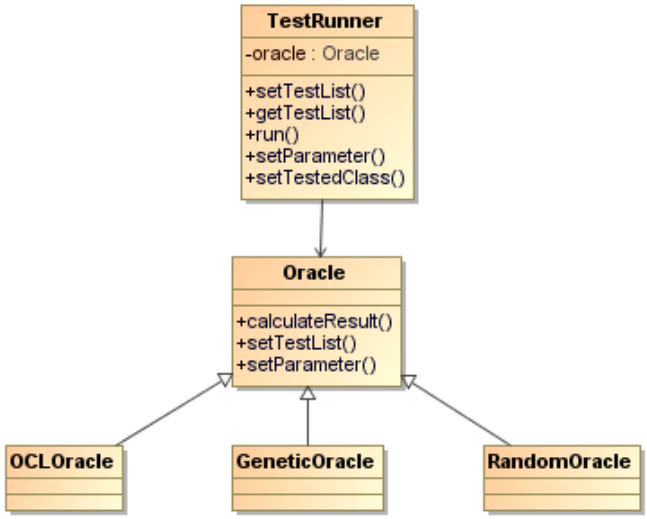
Šis paketas skirtas testų generavimui. Šio paketo pagalba generuojami testai atsitiktiniu, OCL ir genetiniu metodais. Taip pat realizavus TestGenerator bus galima pridėti ir naujų generavimo metodų. Šio paketo klasių diagrama pateikiama 20 pav.



20 pav. TestGenerator paketo klasių diagrama

TestRunner paketas

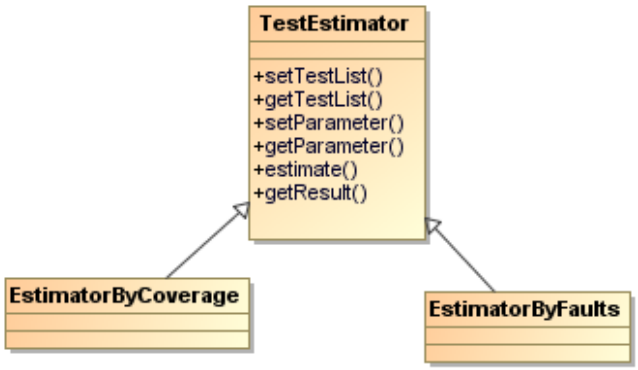
Šis paketas skirtas testų paleidimui. Šio paketo pagalba leidžiami sugeneruoti testai. Taip pat šis paketas turi kiekvieno metodo (OCL, genetinio ir atsitiktinio) realizuotą orakulą, kuris nustatys ar praleistas testas grąžino gerus rezultatus ar blogus, t.y. ar testas praėjo ar ne. Šio paketo klasių diagrama pateikiama 21 pav.



21 pav. TestRunner paketo klasių diagrama

TestEstimator paketas

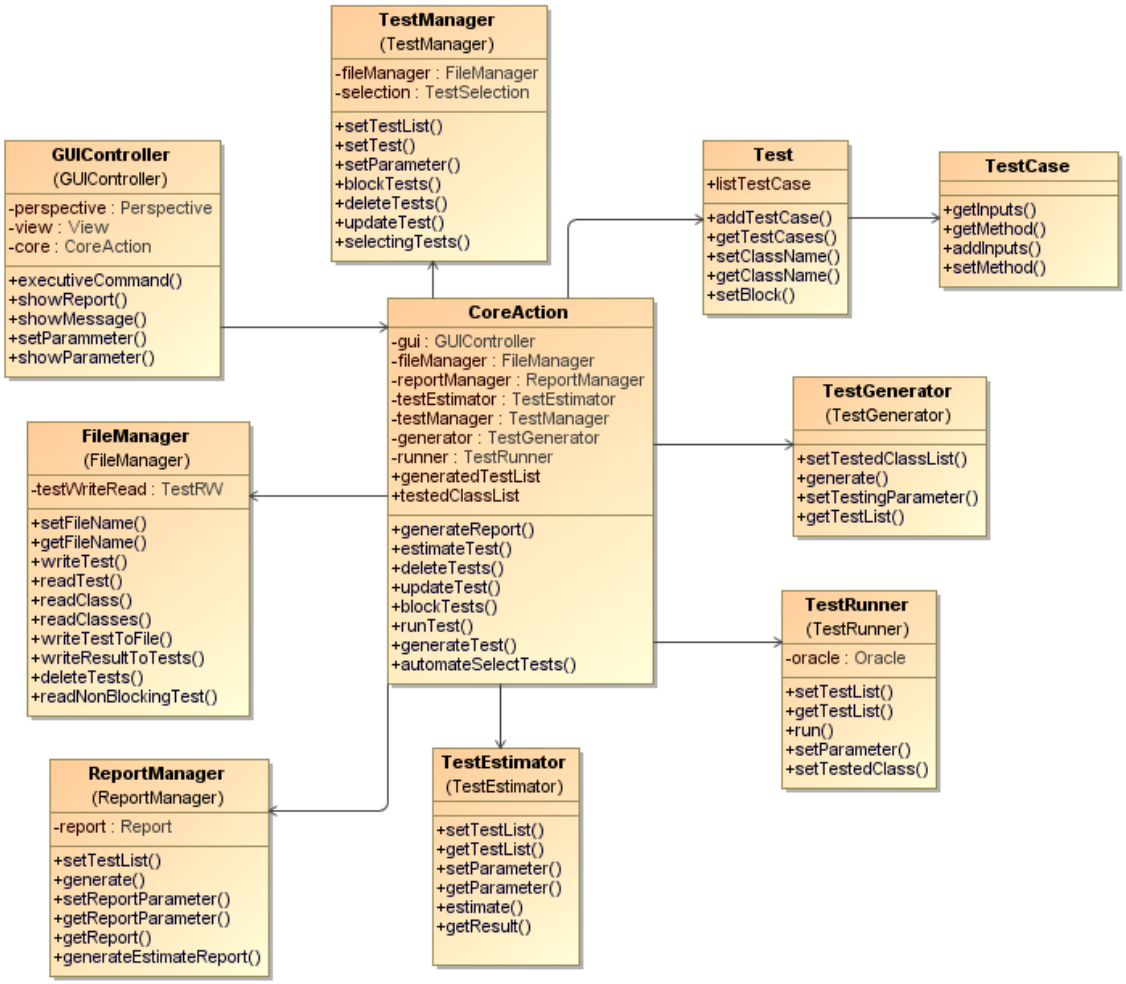
Šis paketas skirtas testų įvertinimui. Šio paketo pagalba bus įvertinami testai dviem metodais, pagal kodo padengimą ir pagal klaidų aptikimą. Šio paketo klasių diagrama pateikiama 22 pav.



22 pav. TestEstimator paketo klasių diagrama

TestCore paketas

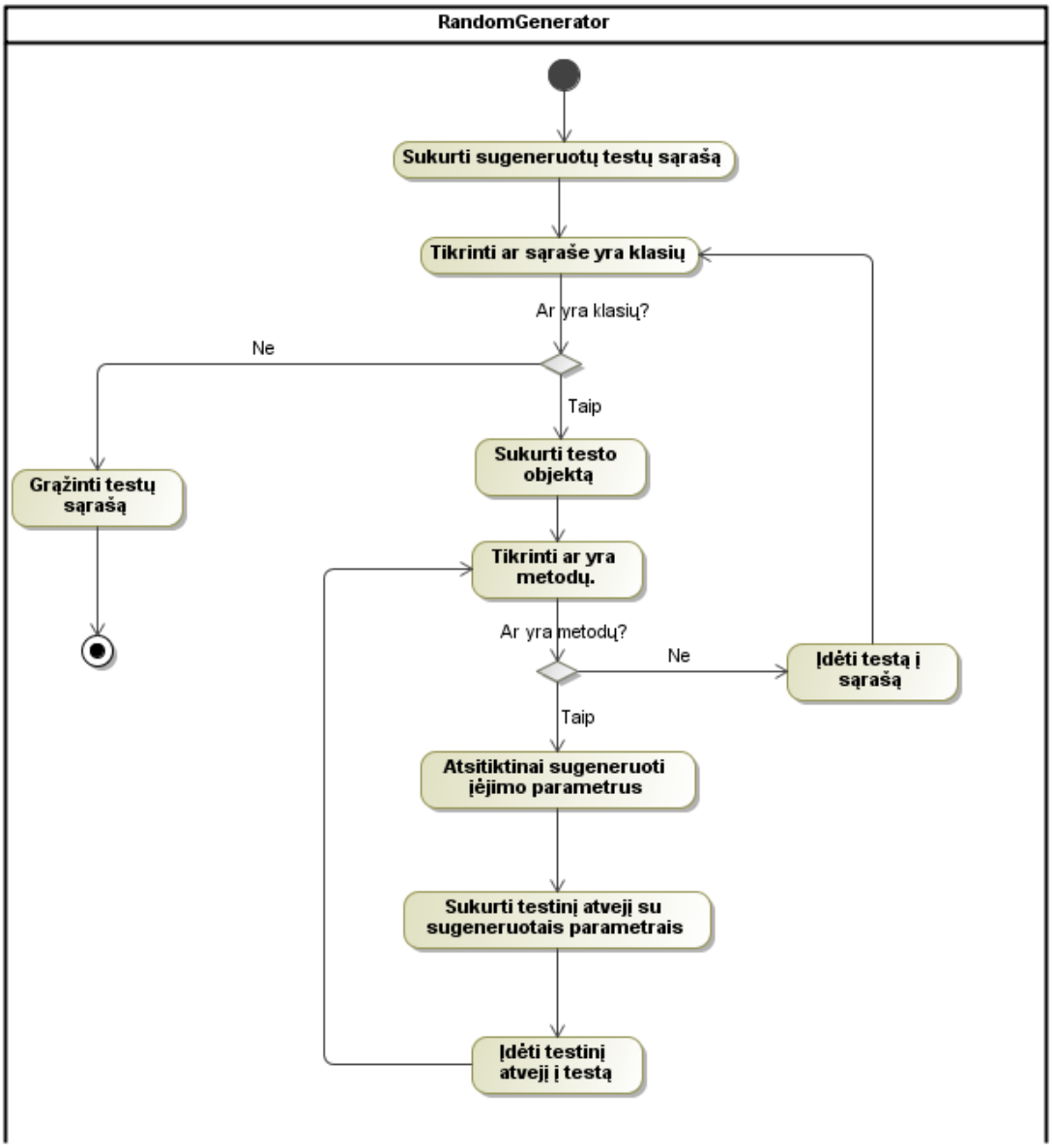
Šis paketas yra įrankio pagrindinis paketas, kurio metodus kvies grafinės sąsajos kontroleris ir kuris valdys visus kitus paketus, t.y. kvies kitų paketų metodus ir funkcijas. Šio paketo klasių diagrama pateikiama 23 pav. Diagramoje dėl aiškumo vaizduojamos ir kitos klasės iš kitų paketų.



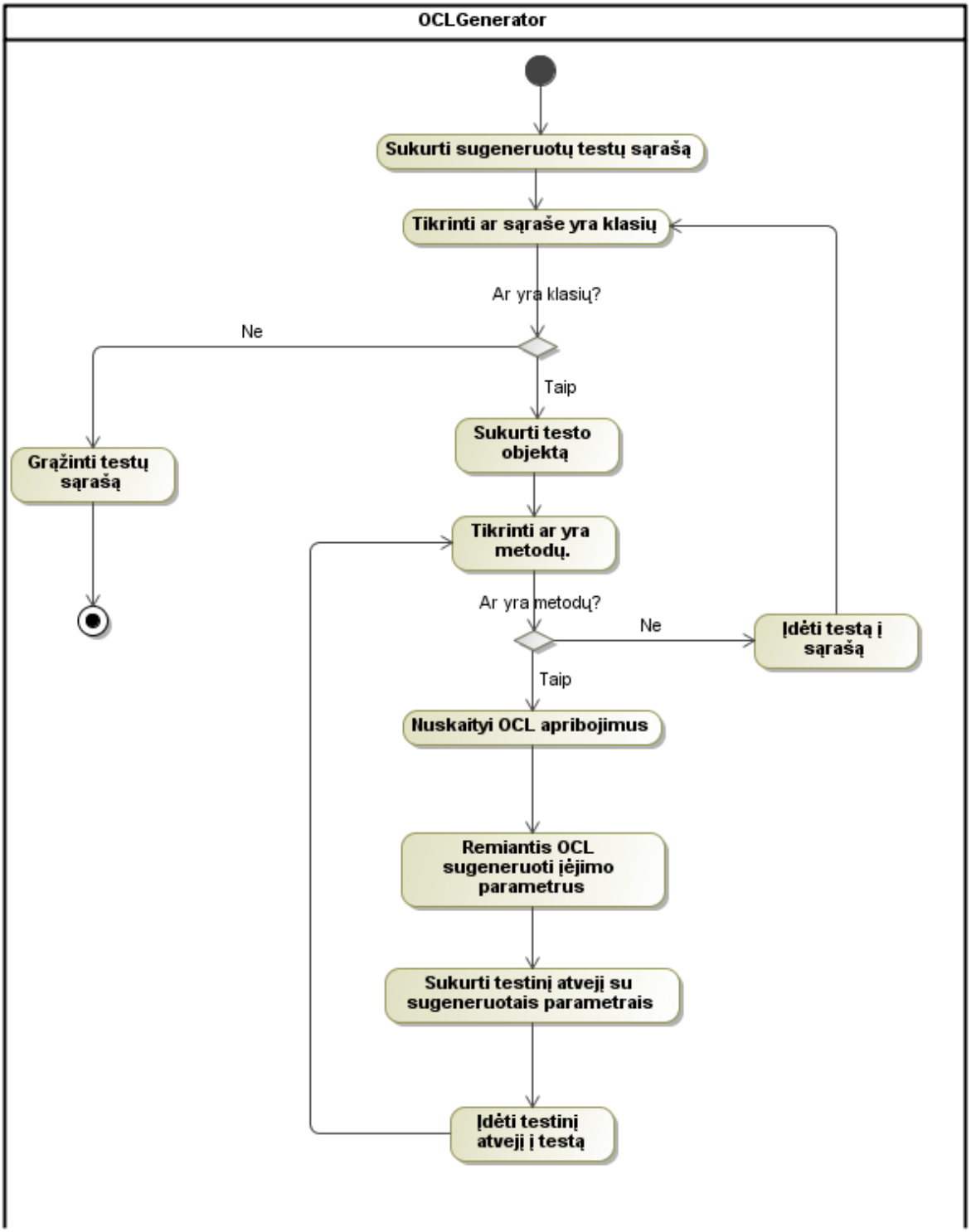
23 pav. TestCore paketo klasių diagrama

3.3.4 Sistemos dinaminis vaizdas

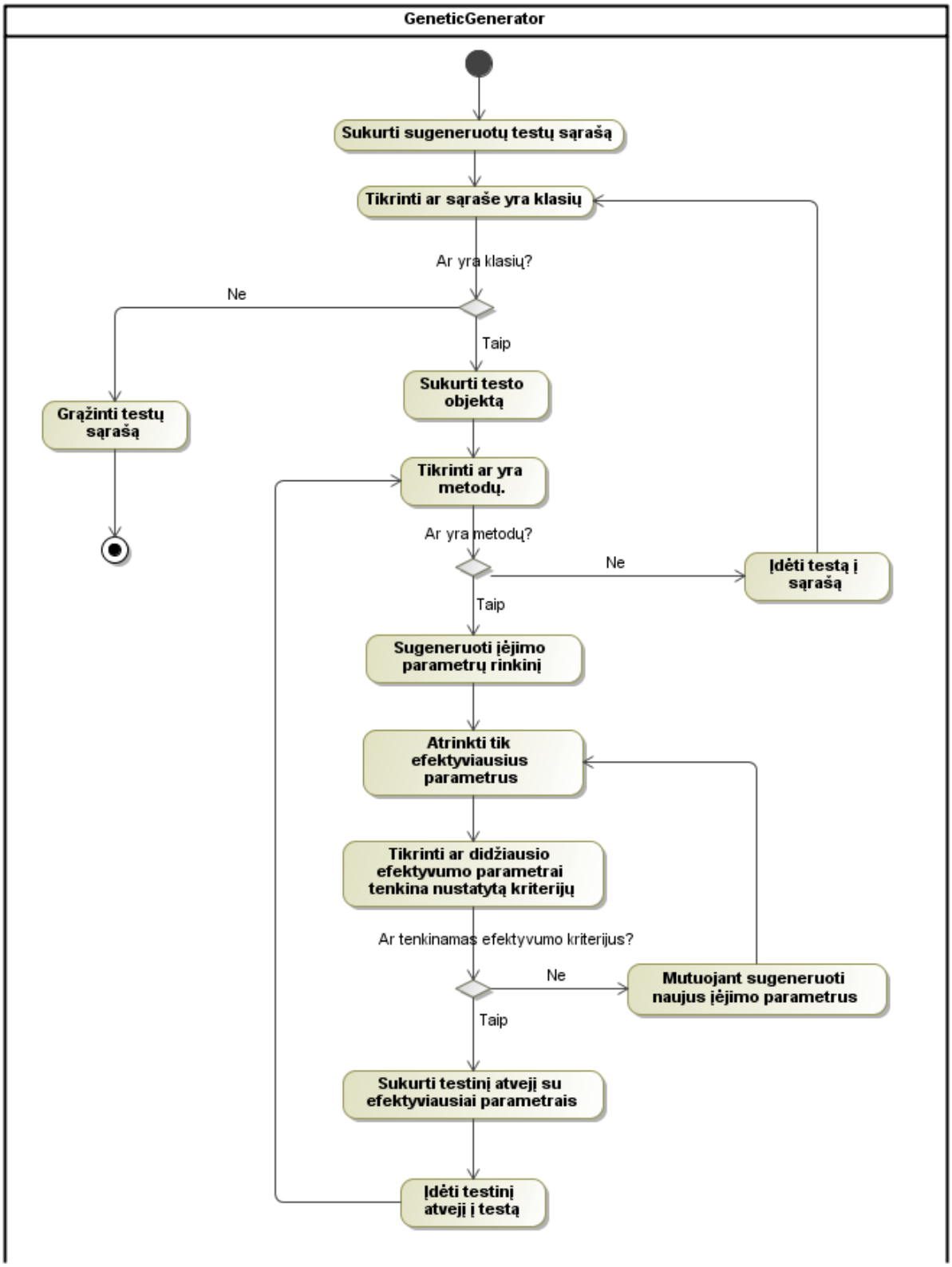
Testų generavimo atsitiktiniu, OCL apribojimų ir genetiniu algoritmais veiklos diagramos pateikiamos žemiau.



24 pav. Atsitiktinio generavimo algoritmo veiklos diagrama



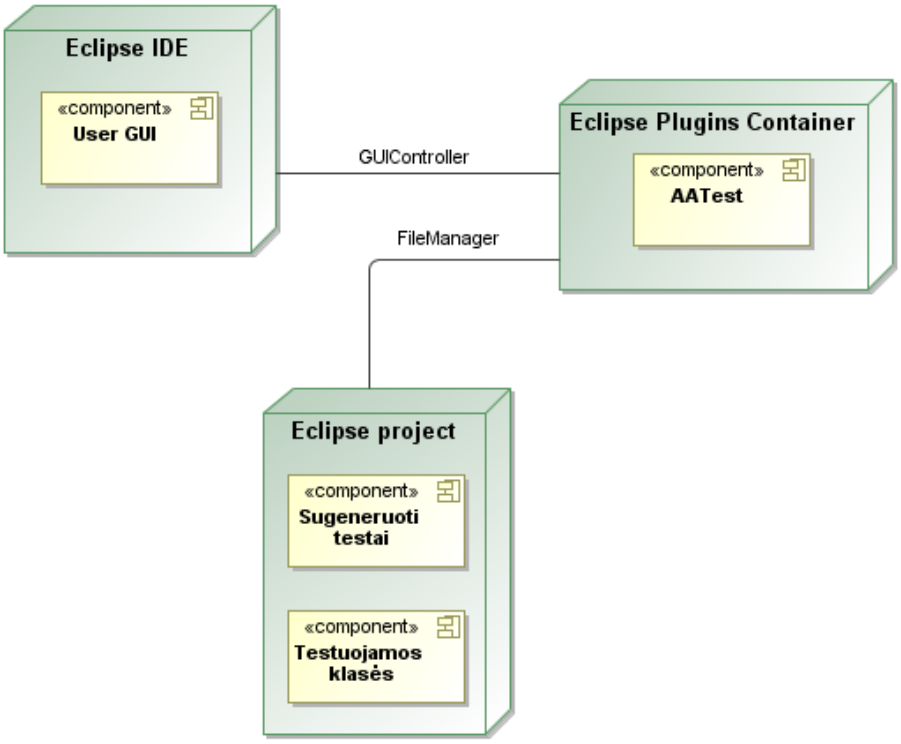
25 pav. OCL apribojimai paremtas generavimo algoritmo veiklos diagrama



26 pav. Genetinio generavimo algoritmo veiklos diagrama

3.3.5 Išdėstymo vaizdas

27 pav. pateikta sistemos išdėstymo diagrama



27 pav. Įrankio išdėstymo diagrama

Eclipse IDE

Tai Eclipse programavimo aplinką, į kurią bus integruotas kuriamas įrankis, ir šios aplinkos pagalba bus atvaizduojami testavimo įrankio įvedimo formos bei išvedami rezultatai. Testavimo įrankis su Eclipse IDE bendraus per GUIController.

Eclipse Plugins Container

Tai Eclipse plug‘ino kontaineris į kurį bus integruotas kuriamas testavimo įrankis AATest.

Eclipse Project

Tai Eclipse projektas, kuriame bus saugomos testuojamos klases, bei šiame projekte bus įrašomi šiame projekte esančioms klasėms sugeneruoti testai. Testavimo įrankis su failų sistema bendraus per FileManager.

3.3.6 Duomenų vaizdas

Visus rezultatus kuriamas įrankis saugos Eclipse projekte failų sistemoje, todėl duomenų bazė projektuojamam įrankiui nėra reikalinga.

3.3.7 Vartotojo sąsaja

Kuriant vartotojo sąsaja iškyla problema, kokia ji turi būti, kad įrankis taptų populiarus, lengvai perprantamas ir patiktų vartotojui. Vartotojai šiuo metu pagrinde bus programuotojai ir testuotojai, taigi tai žmonės turėję nemažai įrankių ir žino kokia vartotojo sąsaja yra patogi ir kokia nepatogi. Panagrinėjus vartotojo sąsajos kūrimo principus [25] nuspręsta, jog neverta kurti naujo atskiro įrankio, o šį įrankį integruoti kaip įskiepi vienai iš populiariausių nemokamų programavimo aplinkų Eclipse. Šis įrankis nemokamas ir palaiko įskiepiu kūrimą.

Kuriant įskiepi bus laikomasi Eclipse‘s sąsajos standartų ir vadovaujamasi „Eclipse‘s vartotojo sąsajos nurodymais“ (User Interface Guidelines)

3.3.8 Kokybė

- Apibrėžta sąsaja testų generavimo realizavimui bei orakului, leidžia ateityje nesunkiai prijungti ir daugiau testų generavimo algoritmų bei orakulų realizacijų.
- Kuriant įrankį, kaip Eclipse įskiepį nesukels nepatogumų vartotojams įsisavinant įrankio sąsaja, nes ji bus tokia pati kaip ir Eclipse IDE.
- Naudojant kaip Eclipse įskiepį, nereikės vartotojui instaliuoti papildomą programą, ten importuoti testuojamas klases, o patogiai galės dirbti su vienu įrankiu – Eclipse IDE.
- Įrankio instaliavimas vyks per Eclipse atnaujinimo įrankį, todėl vartotojui nereikės ieškoti iš kur parsisiųsti įrankį ir kaip jį suinstaliuoti.
- Sukurta įrankio architektūra leis ateityje nesunkiai praplėsti įrankį, pvz. pridėti Android aplikacijų grafinės sąsajos testavimą ir panašiai.

3.4 Sistemos testavimas

Šiame skyrelyje trumpai pateikiamas apibendrintas testavimo planas, aprašomos testavimo procedūros, bei pateikiami testavimo rezultatai.

3.4.1 Vienetų testavimas

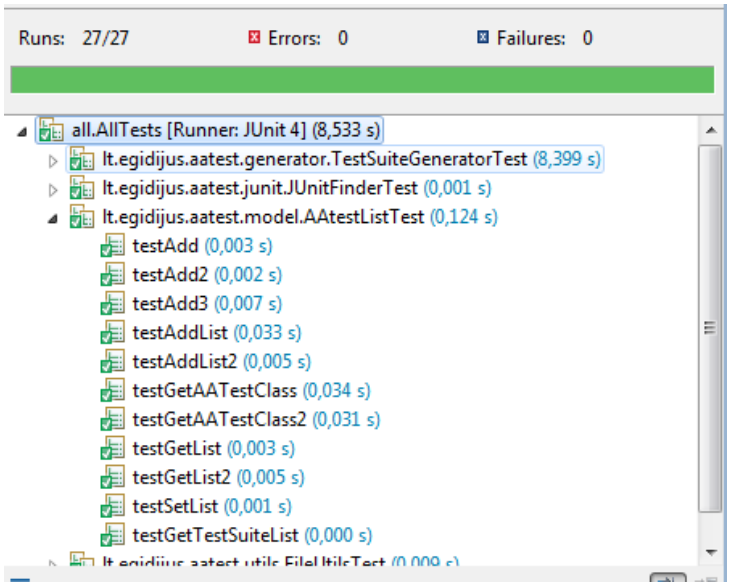
Atskiriems programinės įrangos moduliams/komponentams bus naudojamas vienetų testavimas. Testai bus rašomi dviejų tipų: „baltos dėžės“ – kai bus atsižvelgiama į testavimo įrankio programinį kodą ir „juodos dėžės“ – kai testai bus sudaromi pagal specifikaciją,

Komponentai bus testuojami paduodant jiems įėjimo duomenis ir stebint jų išėjimus ir juos lyginant su laukiamais rezultatais.

Šie testai bus rašomi pasinaudojant Java standartine priemone JUnit. Tokio testo pavyzdys pateikiamas žemiau:

```
import junit.framework.*;
public class calculateCoverage extends TestCase {
  public void testCalculateCoverage(String testName) {
        assertEquals(calculator.calculateCoverage(testename), 95);
  }
}
```

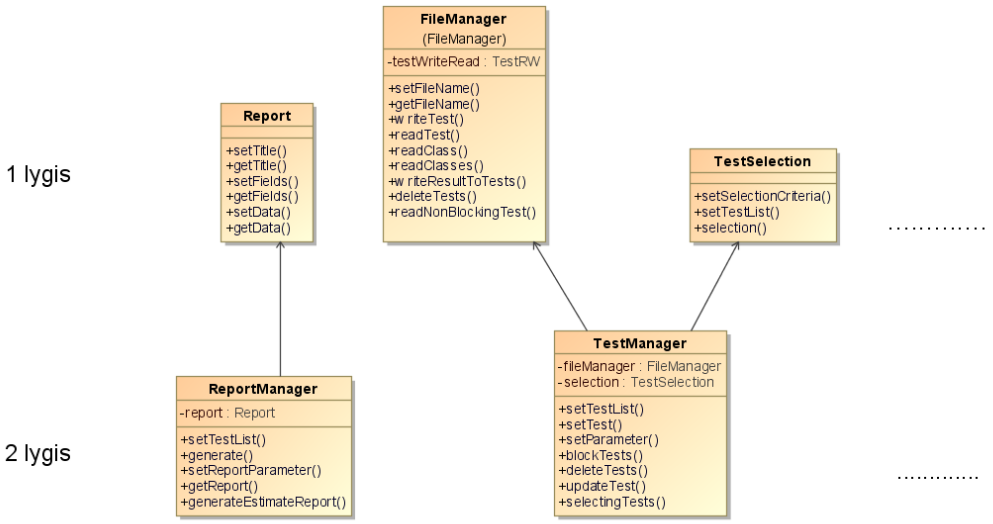
Atliekant vienetų testavimą visos surastos klaidos pataisytos ir sėkmingų testų rezultatas pateikiamas žemiau esančiame paveikslėlyje.



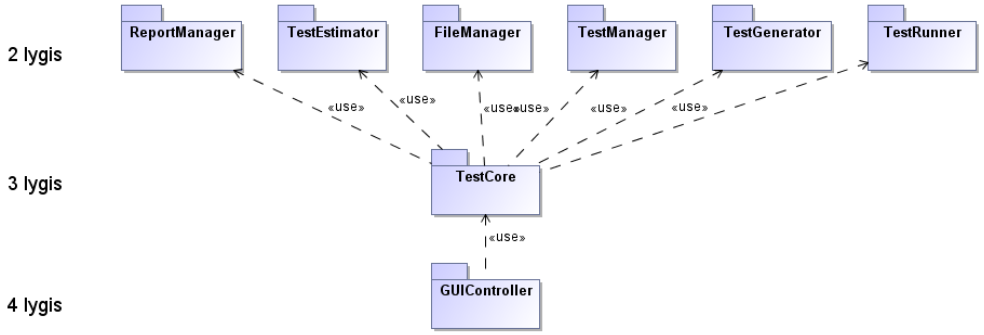
28 pav. JUnit testų rezultato langas

3.4.2 Integracinis testavimas

Vienetai yra apjungiami į bendrą sistemą ir testuojama jų sąveika. Šiam testavimui naudojamas „bottom-up“ integracinis testavimas bei palaipsninis principas. Pirmiausia testuojami komponento lygyje (1-2 lygis) Tokio integravimo pavyzdinė schema pateikta 29 pav. Vėliau palaipsniui apjungiant komponentus testuojama toliau. Tokia schema vaizduojama 30 pav.



29 pav. Integravimo pavyzdinė schema



30 pav. Integravimo pavyzdinė schema

Integracinis testavimas pradedamas nuo žemiausio lygio, o apjungiant pasinaudojama valdikliais „drivers“. Tokiems valdikliams kurti naudojama Eclipse įskiepis „EasyMock“

Integracinis testavimas buvo atliekamas pagal aprašytą procedūrą. Kiekvieno integracinio testavimo lygio rezultatai pateikiami žemiau esančioje lentelėje.

4. Testavimo rezultatai pagal integravimo lygius

Testavimo lygis	**Rezultatas**
1 lygis	Testavimas sėkmingas, visos rastos klaidos ištaisytos
2 lygis	Testavimas sėkmingas, visos rastos klaidos ištaisytos
3 lygis	Testavimas sėkmingas, visos rastos klaidos ištaisytos
4 lygis	Testavimas sėkmingas, visos rastos klaidos ištaisytos

3.4.3 Sąsajos testavimas

Įrankio sąsajos testavimas bus atliekamas dviem etapams:

1 etapas – kuriems tikslinga, tiems sąsajos elementams bus parašomi automatiniai sąsajos veiksmų scenarijai pasinaudojant SWTBot įrankiu.

Tokio testo pavyzdys:

```
workbenchBot.menu("Generuoti").menu("Generuoti atsitiktiniu
metodu").click();
workbenchBot.tree().select("Projektas").expandNode("src").select("main.java
");
```

2 etapas – testuotojas pagal sąsajos testavimo scenarijus testuos aplikaciją ir testavimo rezultatus žymės lentelėse.

Tokio testavimo scenarijai ir jų rezultatai pateikiami žemiau esančiose lentelėse.

Sąsajos testavimas buvo atliekamas pagal testavimo plane aprašytą procedūra. Testavimo scenarijų rezultatai pateikiami žemiau esančiose lentelėse.

5. Lentelė. Parametrų suvedimo, testų ataskaitos testavimo rezultatai

Testas	**Laukiamas rezultatas**	**Rezultatas**
Pasirenkamas "Generuoti .. metodu“ iš kontekstinio menu	Iškviečiamas parametrų pasirinkimo langas	Pavyko
Nesuvedami parametrai ir spaudžiama „Generuoti testus“	Išmetama klaida, jog nenurodyti parametrai	Pavyko
Užpildomi parametrai ir spaudžiama „Generuoti testus“	Testai sugeneruojami ir išmetama rezultatų suvestinė, kurioje turi būti nurodomi kiek testų sugeneruota ir kiek užtruko generavimas, kurioms klasėms testų sugeneruoti nepavyko.	Pavyko

6. Lentelė. Įvertinimo rezultatų, įvykdytų testų langų testavimo rezultatai

Testas	**Laukiamas rezultatas**	**Rezultatas**
Pasirenkamas "Įvertinti testus“ iš kontekstinio menu	Išmetamas sėkmingo įvertinimo įvykdymo pranešimas ir parodomas įvertintų testų sąrašas, kuriame nurodomas testas bei jo įvertinimo reikšmė	Pavyko
Ant nurodyto testo/testų pasirenkamas „Paleisti testus“	Išmetamas pranešimas apie sėkmingą įvykdymą ir atvaizduojamas testų sąrašas, kuriame nurodomas testas bei įvykdymo sėkmingumas	Pavyko
Ant ne testinės klasės spaudžiamas „Įvertinti/Įvykdyti testus“	Parodomas klaidos pranešimas, jog toks veiksmas negalimas	Pavyko

7. .Lentelė. Testų sąrašo, redagavimo langų testavimo rezultatai

Testas	Laukiamas rezultatas	Rezultatas
Ant testo sąrašo paspaudžiama du kartu pelyte	Atidaromas testo redagavimas	Pavyko
Testo sąraše spaudžiamas dešinys pelės klavišas	Parodomas kontekstinis menu	Pavyko
Redagavimo režime taisome testo kodą	Testo kodą leidžia taisyti	Pavyko

3.4.4 Priėmimo testavimas

Naudojant ši metodą programinė įranga bus demonstruojama užsakovui, užsakovas ją išbandys norint nuspręsti ar ji atitinka jo poreikius, specifikaciją. Testavimas atliekamas peržiūrint kiekvieną programinės įrangos panaudojimo atvejį. Aptikus neatitikimus vartotojo norams bus patikrinama ar testų generavimo įrankio realizuotos funkcijos atitinka specifikaciją, esant specifikacijos atitikimui bus registruojamas sistemos pakeitimas, kas, labai tikėtina, bus perkelta į sekančią programinės įrangos versiją.

Priėmimo testavimas buvo atliekamas pagal aprašytą procedūrą. Buvo užsakovui demonstruojama programinė įranga ir peržiūrimi panaudojimo atvejai. Panaudojimo atvejų peržiūros rezultatai aprašyti žemiau.

Panaudojimo atvejis: Generuoti testus

- Patikrintas funkcionalumas. PA veikia tinkamai.

Panaudojimo atvejis: Įvertinti testus

- Patikrintas funkcionalumas. PA veikia tinkamai.

Panaudojimo atvejis: Vykdyti testus

- Patikrintas funkcionalumas. PA veikia tinkamai.

Panaudojimo atvejis: Sugeneruoti ataskaitą

- Patikrintas funkcionalumas. PA veikia tinkamai.

Panaudojimo atvejis: Tvarkyti testus

- Patikrintas funkcionalumas. PA veikia tinkamai.

3.4.5 Testavimo etapo išvados

Projekto vykdymo metu buvo sėkmingai sukurtas testavimo planas, bei vėliau pagal sukurtą testo planą buvo atliekamas programinės įrangos testavimas. Įvykdžius testavimo planą ir pagal gautus rezultatus galima teigti, kad realizuotos visos sistemos funkcijos ir jos veikia tinkamai, todėl programinė įranga gali būti pateikta naudojimui.

3.5 Sistemos įdiegimas

Įrankis įdiegtas įmonėje VĮ „Registrų centras“ Informacinių technologijų centro IS konstravimo departamento programuotojų dirbančių su Android projektais kompiuteriuose. Įdiegimo faktą patvirtinantis aktas pridėtas prieduose.

3.6 Išvados

1. Suprojektuotas, realizuotas ir ištestuotas Android OS aplikacijų testų generavimo įrankis atitinkantis apibrėžtus reikalavimus.
2. Ateityje atsiradus greitesniam padengimo skaičiavimui Eclipse programavimo aplinkoje pertvarkyti genetinio algoritmo realizaciją.
3. Realizuotas įrankis įdiegtas VĮ „Registrų centras“ įmonės programuotojų, dirbančių su Android projektais kompiuteriuose bei įrankis yra visiems laisvai prieinamas internetu.

4. TYRIMO DALIS

Šioje dalyje atliekama projektinėje dalyje sukurtos programinės įrangos analizė. Jos metu buvo realizuoti trys testų generavimo metodai, kuriais pasinaudojant yra sugeneruojami Android aplikacijoms vienetų testai. Šioje dalyje pateikiamas šių testų generavimo algoritmų tyrimas bei išvados. Tyrimo rezultatai publikuoti straipsnyje „Automatinis testų generavimas testuojant Android OS aplikacijas" [7]. Apibrėžiamos pastebėtos problemos naudojant šiuos metodus bei pasiūlomas patobulintas ir apibendrintas vienetų testų generavimo algoritmo sprendimas pritaikytas Android OS sukurtoms aplikacijoms testuoti. Taip pat aprašoma šio algoritmo praktinė realizacija patobulinant projekto metu sukurtą programinę įrangą.

4.1 Testų generavimo algoritmų palyginimo tyrimas

Projektinėje dalyje buvo sukurta programinė įranga, kuri generuoja vienetų testus, skirtus Android OS aplikacijoms testuoti. PĮ realizacijos metu buvo realizuoti trys vienetų testų generavimo algoritmai: atsitiktinis generavimo metodas, panaudojant OCL apribojimus bei genetinis. Buvo atliekamas tyrimas, kurio tikslas išsiaiškinti , kurie iš realizuotų algoritmų yra labiausiai tinkami generuoti Android aplikacijoms skirtus testus.

Pagrindiniai tyrimo uždaviniai yra: ištestuoti pasirinktas kelias realias Android aplikacijas pasinaudojant projekto metu sukurtu įrankiu; atlikti rezultatų analizę bei pateikti patobulinimus pastebėtoms problemos spręsti.

Algoritmų vertinimo kriterijai [40]:

- Testų generavimo laikas – skaičiuojamas laikas per kurį yra sugeneruojami testai.
- Kodo padengimo skaičiavimo laikas – skaičiuojamas laika per kurį buvo atliekamas kodo padengimas.
- Kodo padengimas sugeneruotais testais – šis kriterijus parodo, kiek procentų testuojamos programos kodo eilučių yra padengiami sugeneruotais testais.
- Aptiktų ir visų mutantų skaičius – šie duomenys gaunami naudojant mutacinį testavimą. Visi mutantai – tai nežymus originalios programos pakeitimai. Jeigu testai surado, kad yra klaida mutante, toks mutantas yra vadinamas aptiktu mutantu.

4.1.1 Tyrimo aplinka

Bandymai buvo atliekami ant vieno kompiuterio su reikalinga programine įranga. Kompiuterio techniniai parametrai: CPU - Intel Core 4 branduolių, RAM- 4GB, Windows 7 64bit. Programinė įranga: Eclipse Helios, Android SDK r16, ADT r16. Testuojama buvo ant realaus mobilaus telefono, kurio techninės charakteristikos: CPU – 800Mhz, RAM– 512Mb, Android 2.3.

4.1.2 Naudojamos Android aplikacijos

Tyrime testavimui buvo naudotos realios mobiliosios Android aplikacijos, kurios buvo parsisiųstos iš Google kompanijos palaikomos programų saugyklos. [12]:

AChartEngine – tai grafikų braižymo biblioteka skirta naudoti Android aplikacijoms [1]

Alaus-radaras – tai lietuvių sukurta programėlė, kuri padeda susirasti barą pagal jame pilstomo alaus rūšis [19]

Luminance – tai dėlionės žaidimas, kuris buvo sukurtas ant XNA/XBOX360 ir parašytas C# kalba, tačiau dabar autoriai perrašo Android OS sistemai. [20]

Robobrain-sdk – tai paprastas, lengvasvoris 2D žaidimų variklis skirtas Android aplikacijoms kurti. [30]

Tyrime naudojamų aplikacijų paketų charakteristikos pateikiamos 8 lentelėje.

8. Lentelė. Testuojamų Android aplikacijų charakteristikos

Aplikacija	Paketas	Klasių skaičius	Kodo eilučių skaičius	Sugeneruotų mutantų skaičius
AChartEngine	org.achartengine.render	6	451	221
Luminance	ca.luminance.input	8	259	135
Alaus-radaras	alaus.radaras.submition	6	82	44
Robobrain-sdk	org.robobrain.sdk.graphics	10	488	367

4.1.3 Tyrimui naudojami įrankiai

Kodo padengimui skaičiuoti buvo naudojamas Google kūrėjų teikiamas kodo padengimo komandinės eilutės įrankis. Kadangi kodo padengimo skaičiavimas buvo numatytas projekto metu realizuotame įskiepyje, tai komandinės eilutės įrankiui buvo realizuotas Eclipse grafinės sąsajos interpretatorius ir atliekant tyrimui atskiros programinės įrangos neprireikė. Google teikiamas komandinės eilutės įrankis naudoja EMMA [35] padengimo skaičiavimo biblioteką. Jos teikiamos kodo padengimo ataskaitos pavyzdys pateikiamas prieduose 9.1 skyrelyje.

Mutaciniam testavimui atlikti nebuvo rasta nė vieno tinkamo įrankio, kuris galėtų modifikuoti Dalvik virtualiai mašinai skirtus sukompiliuotus failus. Todėl buvo šiek tiek modifikuotas MuJava [24] įrankis MuEclipse skirta Eclipse IDE aplinkai ir pritaikytas atlikti pakeitimus Android aplikacijose. Naudoti mutaciniai operatoriai aprašyti skyriuje „1.1.1 Pagal klaidų aptikimą“ 3 lentelėje.

4.1.4 Tyrimo scenarijus

Tyrimas buvo atliekamas pagal žemiau pateikiamą tyrimo scenarijų. Jo metu buvo palyginami projekto metu realizuoti trys vienetų testų generavimo algoritmai testuojant realias Android mobiliąsias aplikacijas.

Tyrimo scenarijus:

1. Pasirenkama testuojama programa. Programa turi būti Eclipse IDE projektas ir neturi būti kompiliavimosi klaidų.
2. Programos pasirinkto paketo klasėms užrašomi OCL apribojimai.
3. Sugeneruojami vienetų testai pasirinktoms klasėms ar paketams.
4. Skaičiuojamas kodo padengimas, leidžiant sugeneruotus testus ir peržiūrima padengimo ataskaita.
5. Sugeneruojami mutantai
6. Testuojami mutantai. Mutantams naudojami tokie patys testais su tokiais pat testavimo duomenimis.
7. Analizuojami padengimo ir mutacinio testavimo rezultatai. Jeigu reikia tobulinami OCL apribojimai ir procesas kartojamas nuo 3 etapo.

4.1.5 Tyrimo eiga ir rezultatai

Tyrimas buvo atliekamas dviem etapais:

1. Sukurtu įrankiu buvo sugeneruojami testai atsitiktiniu metodu visai testuojamai aplikacijai ir paskaičiuojamas tos aplikacijos padengimas.

2. Buvo pasirinktas kiekvienos aplikacijos po vieną paketą. Pasirinkti paketai aprašyti skyrelyje „4.1.2 Naudojamos Android aplikacijos“ 8 lentelėje. Šiems paketams testai sugeneruoti trimis metodais t.y. atsitiktiniu, naudojant OCL apribojimus bei genetiniu algoritmu. Jų tyrimas buvo atliekamas pagal anksčiau aprašytą tyrimo scenarijų. Testavimo metu OCL generavimo metodui buvo sukurti apribojimų failai. OCL failo pavyzdys pateikiamas prieduose 9.2 skyrelyje. Genetiniam algoritmui buvo parinkta 10 testų populiacija.

1. Pirmojo etapo tyrimo rezultatai

Pirmo etapo tyrimo rezultatai pateikiami 9 lentelėje. Šioje lentelėje pateikiamos 4 testuojamų aplikacijų rezultatai, kai testai generuojami visam programiniam kodui. Kaip matome iš lentelės be jokio vartotojo įsikišimo galima padengi nuo 25% iki 47% mobilios aplikacijos kodo eilučių.

9. Lentelė. Atsitiktinio metodo tyrimo rezultatai generuojant testus visai aplikacijai

Aplikacija	**Klasių kiekis**	**Generavimo laikas**	**Padengimo skaičiavimo laikas**	**Projekto kodo eilučių padengimas testais**
AChartEngine	55	1.54 s.	24.16 s.	40% (963/2416)
Alaus-radaras	144	4.52 s.	55.88 s	25% (550/2198)
Robobrain-sdk	46	1.75 s.	35.98 s.	47% (603/1276)
Luminance	80	2.77 s.	60.13 s.	34% (1421/4156)

2. Antrojo etapo tyrimo rezultatai

Antrojo etapo rezultatai pateikiami 10 -12 lentelėse, kuriose atvaizduojami tyrimo rezultatai, kai buvo testuojamas tik konkretus aplikacijos paketas skirtingais metodais.

Atsitiktinio metodo sugeneruoti testai ir jų tyrimo rezultatai pateikiami 10 lentelėje. Kaip matome iš lentelės pasiekti galima nuo 57% iki 94% kodo eilučių padengimo. Sugautų mutantų santykis yra nuo 32% iki 50%.

10. Lentelė. Atsitiktinio metodo tyrimo rezultatai

Aplikacija	**Testų generavimo laikas, s**	**Kodo eilučių padengimas, %**	**Sugauti mutantai**	**Visų ir sugautų mutantų santykis, %**
AChartEngine	0.58 s	94% (424,8/451)	111	50,23%
Alaus-radaras	0.42 s	71% (58,6/82)	15	34,09%
Luminance	0.61 s	77% (198,8/259)	64	47,41%
Robobrain-sdk	0.67 s	57% (276,8/488)	120	32,70%

Naudojant genetinį metodo sugeneruoti testai ir jų tyrimo rezultatai pateikiami 11 lentelėje. Kaip matome iš lentelės pasiekti galima nuo 62% iki 95% kodo eilučių padengimo. Sugautų mutantų santykis yra nuo 34% iki 53%.

11. Lentelė. Genetinio metodo tyrimo rezultatai

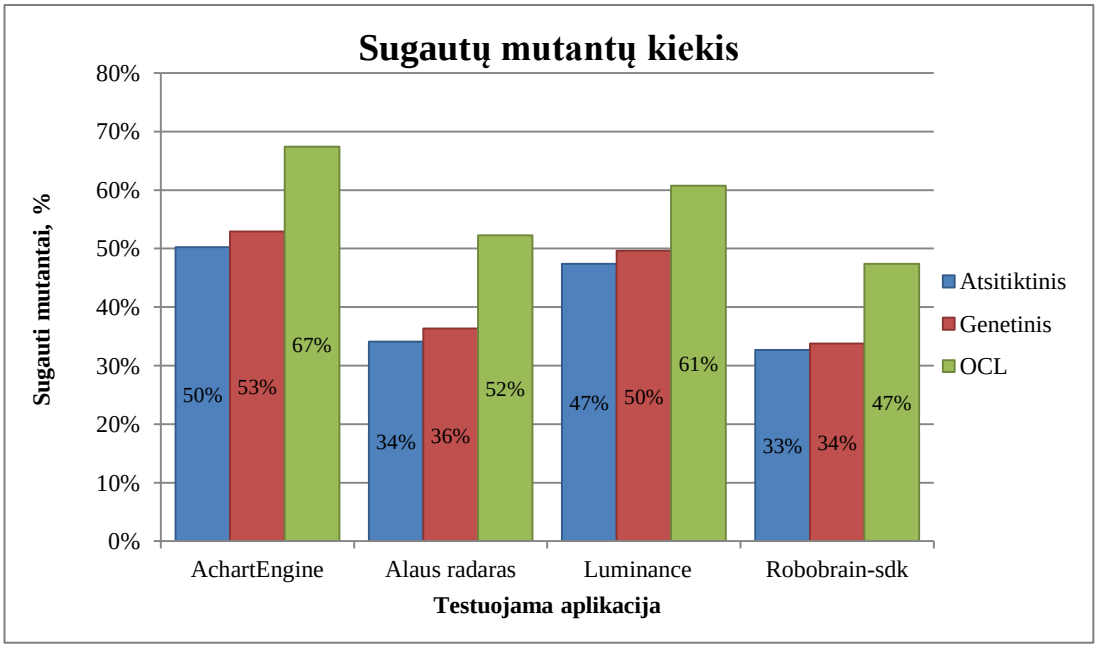
Aplikacija	**Testų generavimo laikas, s**	**Kodo eilučių padengimas, %**	**Sugauti mutantai**	**Visų ir sugautų mutantų santykis, %**
AChartEngine	1783.91 s. (30min)	95% (428,5/451)	117	52,94%
Alaus-radaras	821.87 s. (14 min)	75 % (61,5/82)	16	36,36%
Luminance	1352.12 s. (23 min)	79 % (198,8/259)	67	49,63%
Robobrain-sdk	2621.71 s. (44 min)	62% (302,56/488)	124	33,79%

Naudojant OCL apribojimais paremtu metodu sugeneruoti testai ir jų tyrimo rezultatai pateikiami 12 lentelėje. Kaip matome iš lentelės pasiekti galima nuo 68% iki 97% kodo eilučių padengimo. Sugautų mutantų santykis yra nuo 48% iki 68%.

12. Lentelė. Paremtu OCL metodo tyrimo rezultatai

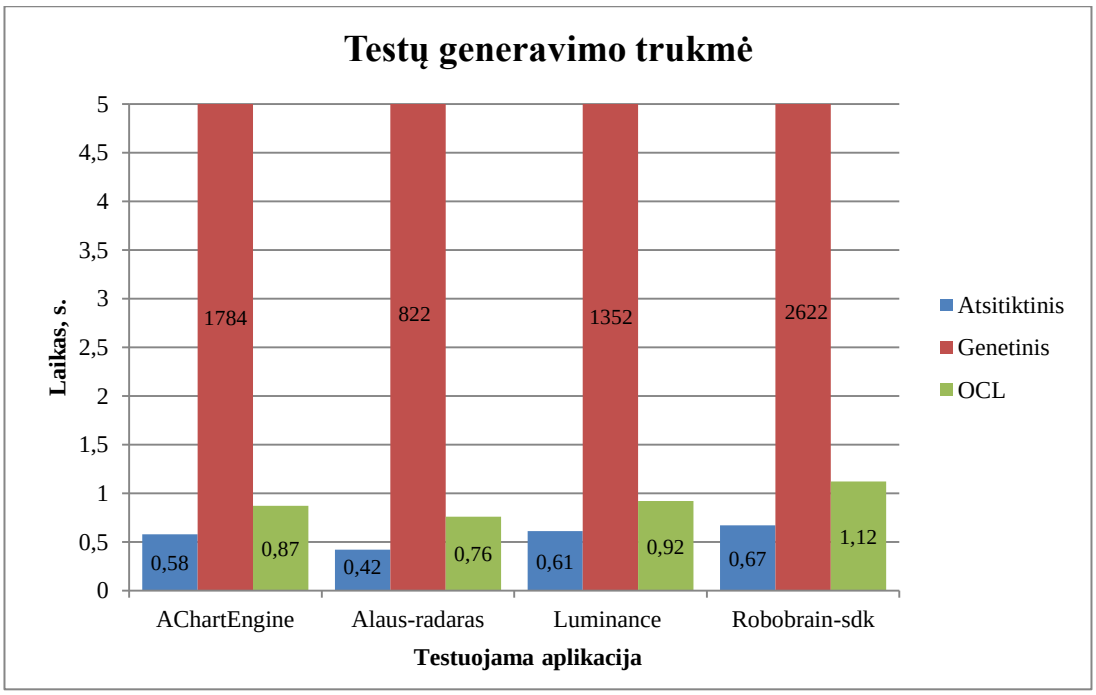
Aplikacija	**Testų generavimo laikas, s**	**Kodo eilučių padengimas, %**	**Sugauti mutantai**	**Visų ir sugautų mutantų santykis, %**
AChartEngine	0.87 s	97% (438,8/451)	149	67,42%
Alaus-radaras	0.76 s	78 % (63,96/82)	23	52,27%
Luminance	0.92 s.	85 % (220,15/259)	82	60,74%
Robobrain-sdk	1.12 s	68% (331,84/488)	174	47,41%

Sugautų mutantų santykio palyginimo pagal projektus diagrama pateikiama 31 paveikslėlyje. Kaip matoma iš diagramos geriausias rezultatas yra pasiekiamas, kai naudojamas OCL apribojimais paremtas generavimas. To ir buvo galima tikėtis, kadangi testus generuojant remiantis OCL yra reikalinga, jog kiekvienam paketui būtų sukurti OCL apribojimų failai, kurie leidžia sugeneruoti efektyvesnius testus. Padengimo skirtumas tarp atsitiktinio generavimo metodo ir genetinio yra labai nedidelis, vidutiniškai apie 2,5%.



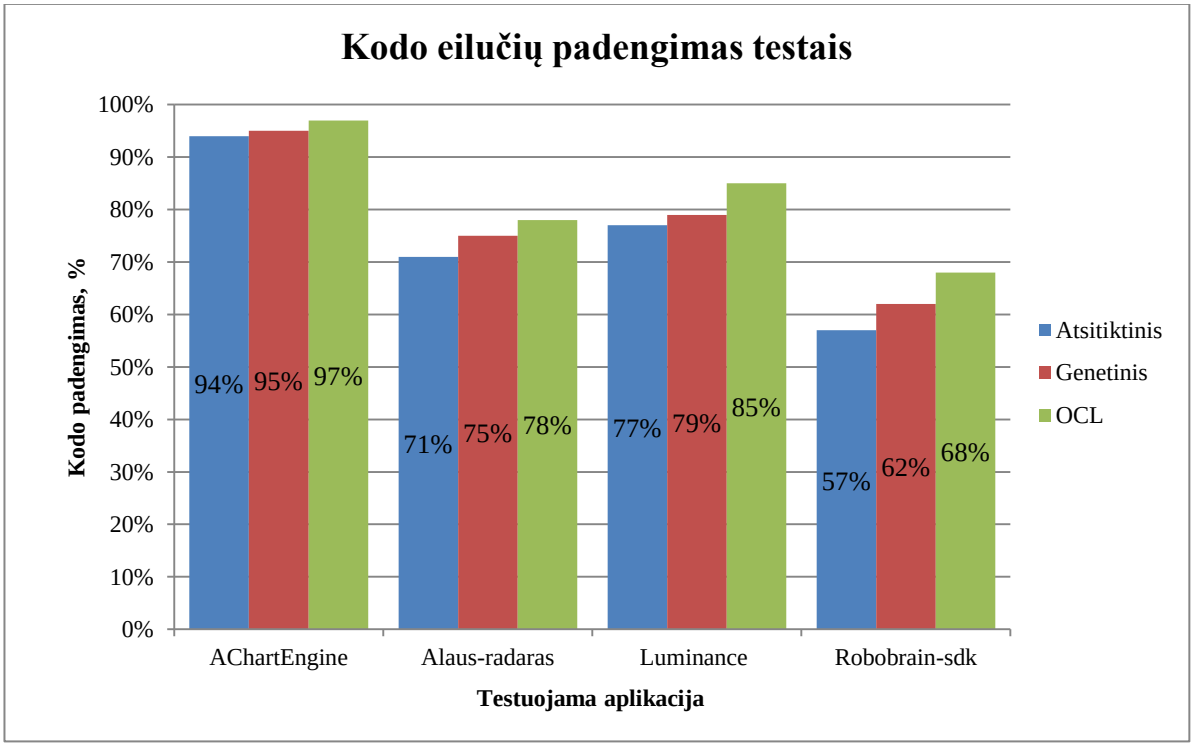
31 pav. Sugautų mutantų kiekis skirtingais metodais

Testų generavimo trukmės palyginimo pagal projektus diagrama pateikiama 32 paveikslėlyje. Kaip matoma iš diagramos atsitiktinio ir genetinio metodų generavimo trukmė vidutiniškai apie 0.8 sekundės. Generuojant genetiniu metodu testų generavimas trunka labai ilgai, vidutiniškai tai 30 minučių sugeneruoti testus vienam paketui, nors buvo parinkta tik 10 testų populiacija, kai norint pasiekti geresnių rezultatų, testų populiacija turėtų būti žymiai didesnė. Tokį ilgą generavimą įtakojo tai, jog po kiekvieno sugeneruoto testo buvo reikalingas testų padengimo skaičiavimas. Šiuo metu nėra testų padengimo įrankio pritaikyto Eclipse IDE, todėl buvo panaudotas Android teikiamas komandinės eilutės įrankis, su kuriuo skaičiuojant testų padengimą užtrunką apie 45 s.



32 pav. Testų generavimo trukmė

Kodo eilučių padengimo testais palyginimo pagal projektus diagrama pateikiama 33 paveikslėlyje. Kaip matoma iš diagramos atsitiktinio ir genetinio metodais padengimas pasiekiamas panašus. Genetiniu algoritmu padengimas yra vidutiniškai 3% didesnis už naudojant atsitiktinį metodą, šis skaičius nėra didelis, kadangi dėl ilgo testų generavimo nebuvo parinkta didelė generuojamų testų populiacija. Geriausias padengimo rezultatas kaip ir buvo tikėtasi pasiekiamas testus generuojant remiantis OCL apribojimais.



33 pav. Programinio kodo eilučių padengimas testais

4.1.6 Tyrimo išvados

Atlikus tyrimą galima teigti, jog naudojantis atsitiktinio testų generavimo metodu be vartotojo įsikišimo testais galima padengti iki 50% testuojamos programos kodo visai aplikacijai, o sugautų mutantų kiekis siekia iki 67 procentų. Panaudojus OCL apribojimus vartotojas gali testų generavimą individualizuoti pagal testuojamą programos kodą, taip padidindamas jos padengimą testais, bei taip pat ir sugaunamų mutantų skaičių. Tačiau genetinis metodas, dėl naudojamo lėto padengimo skaičiavimo (vieno testo padengimas vidutiniškai trunka ~45s) kol kas nėra labai tinkamas, nes vienos klasės su 10 testų populiacija sugeneravimas trunka apie 5 minutes.

4.2 Tyrimo metu pastebėtos problemos

Atlikus projekto metu sukurtos programinės įrangos realizuotų algoritmų tyrimą ir išanalizavus rezultatus pastebimos šios problemos:

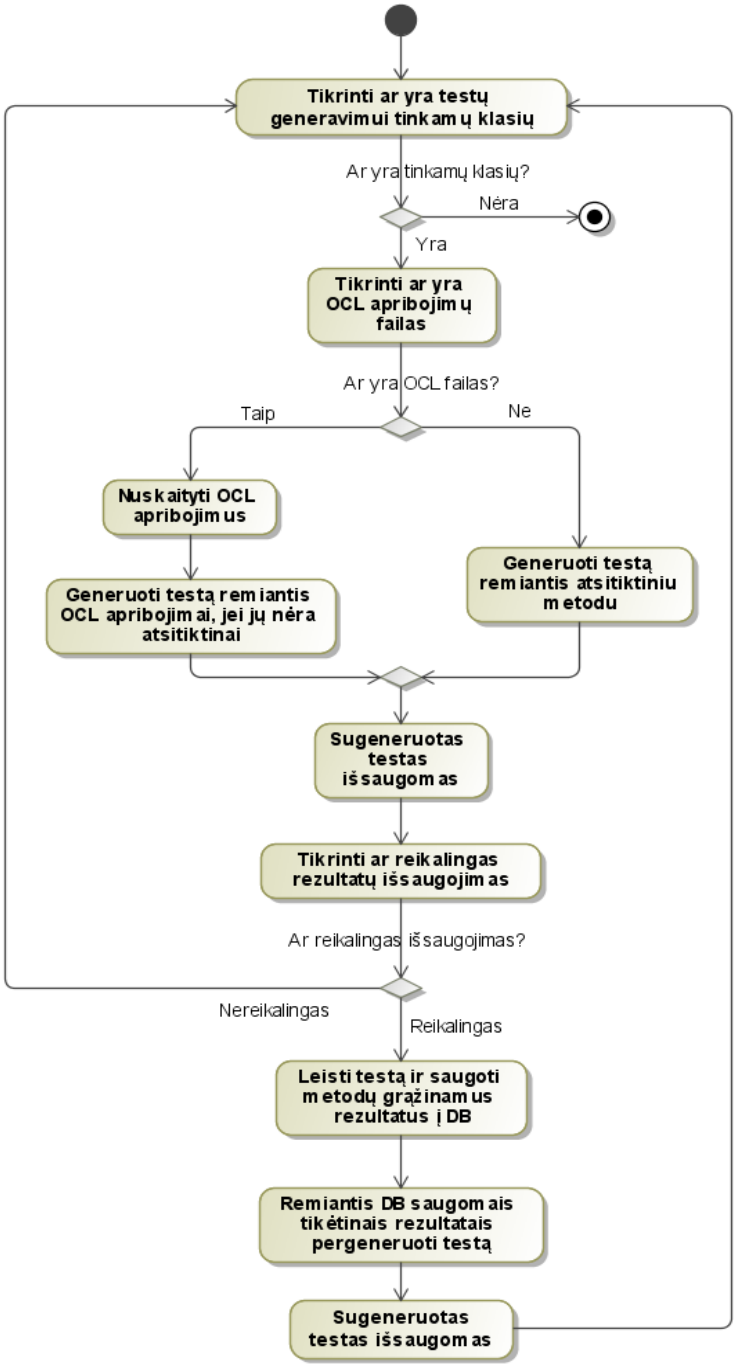
- Ne visuomet sugeneruojami Android aplikacijose naudojami specifiniai objektai, todėl tokie sugeneruoti testai turi klaidų ir negali būti vykdomi. Sprendžiant šią problemą, kaip teigia [32] knygos autorius reikia naudoti netikrus objektus. Netikri objektai yra skirti vienetų testų rašymui ir juos naudojant vietoje realių objektų yra izoliuojami vienetų testai.
- Naudojant atsitiktinį bei genetinį algoritmus yra silpnas testų orakulas. Sprendžiant šią problemą, kaip teigia [31] straipsnio autoriai galima panaudoti gautų rezultatų išsaugojimą bei sekantį kartą leidžiant testus, šiuos gautus rezultatus laikyti kaip tikėtinais ir su jais lyginti dabar gautus testų rezultatus. Toks testavimo būdas vadinamas regresiniu testavimu.
- Naudojant įrankį aplikacijoje sukuriama neteisingų testų. Tokių atvejų pasitaiko, kai aplikacijoje yra persipynusi grafinės sąsajos bei pačios aplikacijos logika. Norint, jog tokių testų pasitaikytų kuo mažiau, reikia, kaip teigia [32] [29] knygų autoriai stengtis aplikaciją kurti kuo labiau testuojamą. Tai yra programuoti laikantis objektiškai orientuotų sistemų principų bei atskirti grafinę atvaizdavimo sąsaja nuo veiksmų logikos ir stengtis kurti kuo labiau izoliuotus komponentus.

4.3 Siūlomas testų generavimo sprendimas

Atsižvelgus į projekto metu sukurtos programinės įrangos tyrimo rezultatus ir išvadas bei atliktą literatūros analizę siūlomas testų generavimo metodas, kuriam keliami šie pagrindiniai reikalavimai:

1. Galimybė testus generuoti atsitiktiniu būdu nereikalaujant vartotojo įsikišimo.
2. Generuojant testus objektai konstruojami automatiškai be arba su parametrais.
3. Specifinių objektų kūrimui naudoti Android SDK teikiamus netikrus objektus.
4. Jeigu metodui yra aprašytas OCL apribojimų failas, generuoti testus remiantis OCL apribojimais.
5. Galimybė sugeneruotų testų rezultatus išsaugoti ir sugeneruoti naujus testus lyginant gaunamus rezultatus su prieš tai išsaugotais, panaudojant regresinio testavimo principus.
6. Suderinamas su Android SDK JUnit testais bei ADT įrankiu.

Siūlomas testų generavimo metodo principinė schema vaizduojama UML veiklos diagramos pagalba 34 paveikslėlyje.



34 pav. Testų generavimo metodo principinė schema

4.4 Sprendimo realizacija

Pagal siūlomą testų generavimo sprendimą buvo patobulinta projekto metu sukurtas Eclipse IDE įskiepis.

Šiame įrankyje atlikti realizacijos pakeitimai:

- ✓ Pridėta įrankyje funkcija, leidžianti išsaugoti sugeneruotus rezultatus į duomenų bazę.

- ✓ Pridėta įrankyje funkcija leidžianti sugeneruoti testus pagal duomenų bazėje esančius tikėtinus rezultatus.
- ✓ Pridėtas duomenų bazės funkcionalumas. Šiame įrankyje naudojama Android palaikoma SQLite [11] duomenų bazė.
- ✓ Atlikti pakeitimai objektų kūrime, jog būtų sugeneruojami netikri objektai iš Android SDK teikiamo android.test.mock paketo.

4.5 Išvados

1. Atliktas tyrimas įvertinantis projekto metu realizuotus tris vienetų testų generavimo metodus.
2. Naudojantis atsitiktinio testų generavimo metodu testais galima padengti iki 50% testuojamos programos kodo bei sugautų mutantų skaičius vidutiniškai 40%.
3. Panaudojus OCL apribojimus vartotojas gali testų generavimą individualizuoti pagal testuojamą programos kodą taip padidindamas jos padengimą testais bei sugaunamą mutantų skaičių.
4. Genetinis metodas dėl naudojamo lėto padengimo skaičiavimo (vieno testo padengimas vidutiniškai trunka ~45s) kol kas nėra labai tinkamas, nes vienos klasės su 10 testų populiacija sugeneravimas trunka apie 5 minutes.
5. Sugeneruotų vienetų testų efektyvumas priklauso nuo testuojamos mobilios Android aplikacijos testuojamumo lygio. Todėl apie testavimą reikia galvoti pradinėse programų kūrimo stadijose.
6. Siekiant padidinti sugaunamų mutantų skaičių išnagrinėjus tyrimo rezultatus, galima teigti, jog pagrindinės problemos kyla kuriant specifinius Android objektus bei dėl silpno testų orakulo.
7. Remiantis tyrimo rezultatais pasiūlytas apibendrintas vienetų testų generavimo sprendimas, kuris:
    - ✓ Pritaikytas Android aplikacijoms. Naudoja Android SDK bei netikrus objektus teikiamus Android SDK
    - ✓ Naudoja atsitiktinį generavimą
    - ✓ Naudoja OCL apribojimus
    - ✓ Naudoja regresinį testavimą
8. Realizuoti patobulinimai, projekto metu sukurtai programinei įrangai pagal tyrimo metu pasiūlytus pastebėtų problemų sprendimus.

5. EKSPERIMENTINĖ DALIS

Tyrimo dalyje buvo išnagrinėtas projekto metu realizuotų algoritmų veikimas ir efektyvumas, bei pagal tyrimo rezultatus ir išvadas pasiūlytas apibendrintas Android OS aplikacijoms vienetų testų generavimo metodas. Šioje dalyje bus atliekamas eksperimentinis tyrimas, kurio tikslas įsitikinti ar pasiūlytas vienetų testų generavimo sprendimas veikia ir kiek jis yra efektyvesnis už projekto metu realizuotus algoritmus. Eksperimento metu buvo generuojami testai tyrimo dalyje aprašytoms testinėms Android aplikacijoms ir atliekamas testų efektyvumo matavimas pagal apibrėžtas metrikas.

5.1 Naudojamos metrikos

Šiame eksperimentiniame tyrime naudojamos tos pačios metrikos, kaip ir lyginant projekto metu realizuotus vienetų testų algoritmus. Tos pačios metrikos pasirinktos todėl, jog būtų galima palyginti šio eksperimento metu gautus rezultatus su projekto metu realizuotų algoritmų tyrimo rezultatais.

Naudojamos metrikos:

- Testų generavimo laikas – skaičiuojamas laikas per kurį yra sugeneruojami testai.
- Kodo padengimo skaičiavimo laikas – skaičiuojamas laika per kurį buvo atliekamas kodo padengimas.
- Kodo padengimas sugeneruotais testais – šis kriterijus parodo, kiek procentų testuojamos programos kodo eilučių yra padengiami sugeneruotais testais.
- Aptiktų ir visų mutantų skaičius – šie duomenys gaunami naudojant mutacinį testavimą. Visi mutantai – tai nežymus originalios programos pakeitimai. Jeigu testai surado, kad yra klaida mutante, toks mutantas yra vadinamas aptiktu mutantu.

5.2 Eksperimentinio tyrimo procesas

5.2.1 Eksperimento aplinka

Tyrimui atlikti buvo naudojama ta pati techninė ir programinė įranga, kaip ir projekto metu realizuotų algoritmų palyginimo metu. Bandymai buvo atliekami ant vieno kompiuterio su reikalinga programine įranga. Kompiuterio techniniai parametrai: CPU - Intel Core 4 branduolių, RAM- 4GB, Windows 7 64bit. Programinė įranga: Eclipse Helios, Android SDK r16, ADT r16. Testuojama buvo ant realaus mobilaus telefono, kurio techninės charakteristikos: CPU – 800Mhz, RAM– 512Mb, Android 2.3.

5.2.2 Testuojamos Android mobiliosios aplikacijos

Eksperimentiniam tyrimui naudojamos realios Android aplikacijos, kurios aprašytos 4.1.2 skyrelyje „Naudojamos Andoid aplikacijos“.

5.2.3 Tyrimo scenarijus

Tyrimo scenarijus panašus į anksčiau atliktą tyrimą, tačiau dėl realizuotų patobulinimų šiek tiek pakeistas ir papildytas.

Tyrimo scenarijus:

1. Pasirenkama testuojama programa. Programa turi būti Eclipse IDE projektas ir neturi būti kompiliavimosi klaidų.
2. Programos pasirinkto paketo klasėms užrašomi OCL apribojimai.
3. Sugeneruojami vienetų testai pasirinktoms klasėms ar paketams.
4. Testai leidžiami pirmą kartą ir gauti rezultatai išsaugomi. Gauti rezultatai patikrinami ir pažymimi, kurie yra teisingi, o kurie yra klaidingi.
5. Sugeneruojami vienetų testai panaudojant išsaugotus rezultatus ir juos laikant kaip tikėtinus.
6. Skaičiuojamas kodo padengimas ir peržiūrima padengimo ataskaita.
7. Sugeneruojami mutantai
8. Testuojami mutantai. Mutantams naudojami tokie patys testais su tokiais pat testavimo duomenimis.
9. Analizuojami padengimo ir mutacinio testavimo rezultatai. Jeigu reikia tobulinami OCL apribojimai ir procesas kartojamas nuo 3 etapo.

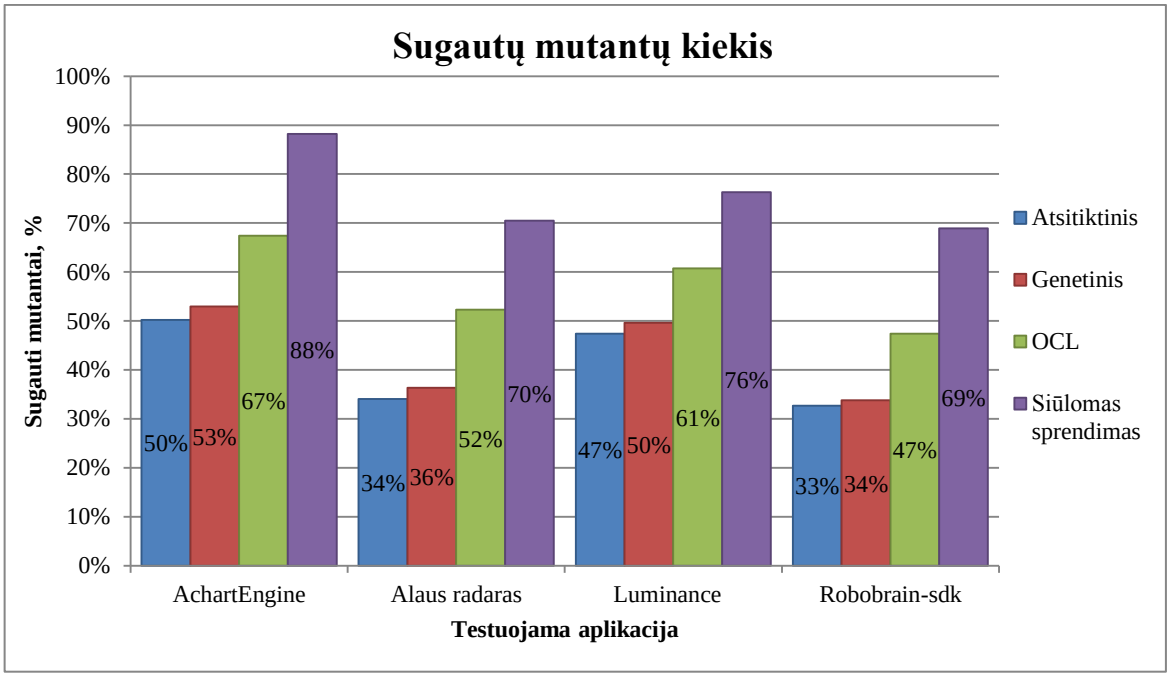
5.3 Eksperimento rezultatai

Naudojant siūlomą patobulintą testų generavimo metodą, jų tyrimo rezultatai pateikiami 13 lentelėje. Kaip matome iš lentelės naudojant šį metodą pasiekti galima nuo 72% iki 87% kodo eilučių padengimo. Sugautų mutantų santykis yra nuo 68% iki 85%.

13. Lentelė. Pasiūlyto metodo tyrimo rezultatai

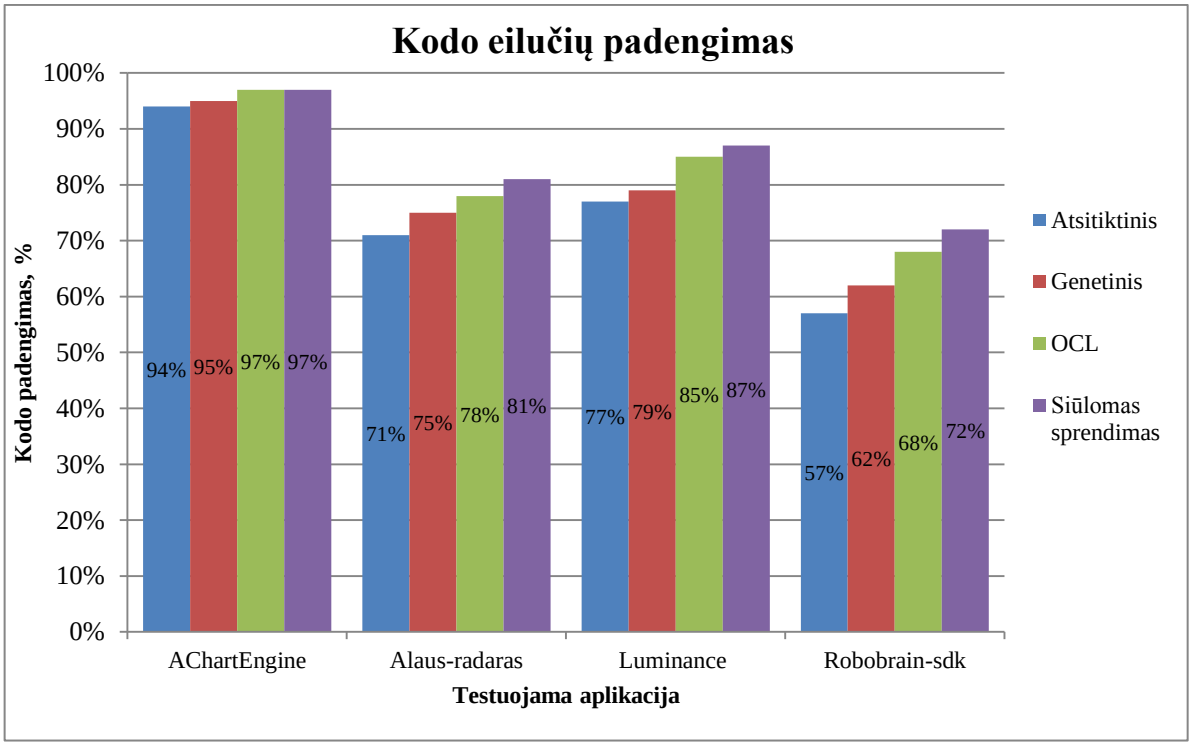
Aplikacija	Testų generavimo laikas, s	Kodo eilučių padengimas, %	Sugauti mutantai	Visų ir sugautų mutantų santykis, %
AChartEngine	0.98 s	97% (438,8/451)	195	88,24%
Alaus-radaras	0.72 s	81 % (66,42/82)	31	70,45%
Luminance	1.02 s.	87 % (225,36/259)	103	76,30%
Robobrain-sdk	1.29 s	72% (351,62/488)	253	68,94%

Sugautų mutantų santykio palyginimas su projekto metu realizuotais algoritmais pateikiamas 35 paveikslėlyje. Kaip matome iš diagramos realizavus tyrimo dalyje pasiūlytus patobulinimus sugautų mutantų kiekis padidėjo vidutiniškai 19 procentų.



35 pav. Metodų palyginimas pagal sugautų mutantų kiekį

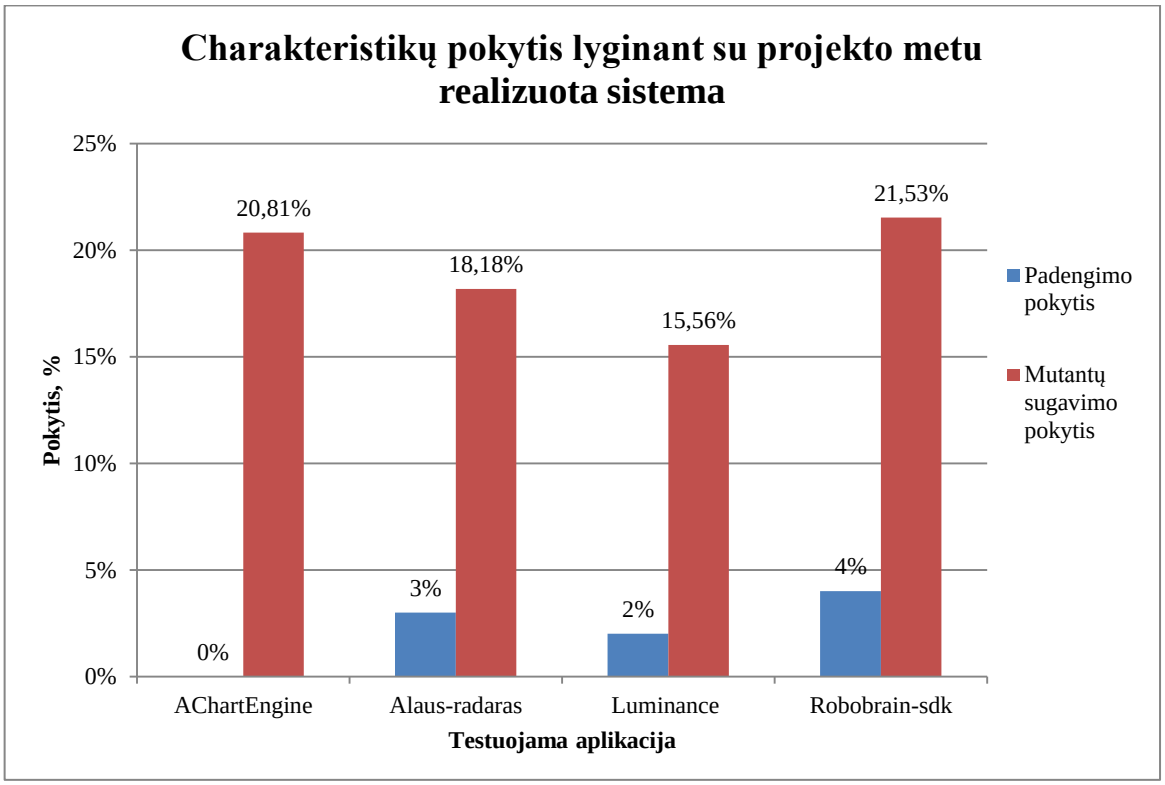
Kodo eilučių padengimo palyginimas su projekto metu realizuotais algoritmais pateikiamas 36 paveikslėlyje. Kaip matome iš diagramos realizavus tyrimo dalyje pasiūlytus patobulinimus kodo eilučių padengimas padidėjo vidutiniškai 4 procentais.



36 pav. Metodų palyginimas pagal kodo eilučių padengimą

Kodo eilučių padengimo bei mutantų sugavimo pokyčio palyginimas su projekto metu realizuota sistema diagrama pateikiamas 37 paveikslėlyje. Lyginama yra su projekto metu realizuotų algoritmų gautais geriausiais rezultatais.

Kaip matome iš diagramos AChartEngine aplikacijoje kodo padengimas realizavus tyrimo dalyje siūlomus patobulinimus nepasikeitė, tai galima paaiškinti, jog testuojamame pakete nebuvo nesugeneruotų Android specifinių komponentų, todėl realizuoti patobulinimai neturėjo įtakos kodo eilučių padengimui testais. Kodo padengimo pokytis kitose aplikacijose padidėjo vidutiniškai 3 procentais. Mutantų sugavimo procentinis pokytis kaip matome iš diagramos padidėjo vidutiniškai 19 procentų, tad galima teigti, jog patobulinimai buvo teisingi ir efektyvūs.



37 pav. Charakteristikų pokyčių su projekto metu realizuota sistema

5.4 Eksperimento išvados

1. Atliktas tyrimas, įvertinantis testų efektyvumą realizavus pasiūlytus patobulinimus projekto metu sukurtai programinei įrangai.
2. Po atlikto patobulinimo, kodo eilučių padengimas vidutiniškai padidėjo 3 procentais.
3. Po atlikto patobulinimo, mutantų sugavimo procentinis pokytis padidėjo vidutiniškai 19% lyginant su projekto metu realizuota sistema.
4. Nebuvo pasiekti rezultatai 100%, nes:
   - ✓ Ne visus OCL apribojimus galima realizuoti esama programinės įrangos versija.

- ✓ Kai kurių specifinių objektų nebuvo įmanoma automatiškai sukurti, todėl tokios kodo vietos liko nepadengtos testais. Dažniausiai tokie objektai, tai Android grafinės sąsajos elementai.
- ✓ Testuojamos programos nebuvo idealiai ir pilnai specifikuota OCL apribojimais, dėl programos veikimo specifikos nežinojimo.

5. Galimi patobulinimai ateityje:
   - ✓ OCL interpretatoriaus praplėtimas ir patobulinimas
   - ✓ Grafinės sąsajos elementų testavimas ir automatinis kūrimas

6. IŠVADOS

1. Šiuo metu Lietuvoje ir pasaulyje išaugo išmaniųjų telefonų pardavimai. Iš visų parduodamų išmaniųjų telefonų apie 50% parduodami su Android operacine sistema.
2. Tobulėjant išmaniesiems telefonams ir jų techninėms galimybėms, kuriama programinė įranga tampa sudėtingesnės ir funkcionalesnės, tačiau kokybės problema vis dar išlieka skaudžia programinės įrangos kūrimo dalimi.
3. Projekto metu sukurta programinė įranga leidžianti generuoti vienetų testus, skirtus Android OS aplikacijoms testuoti.
4. Generuojant testinius atvejus pagrindinės problemos:
   - ✓ Objektų kūrimas
   - ✓ Android naudojama skirtinga platforma Dalvik
   - ✓ Testinių duomenų generavimas
   - ✓ Testų orakulas
5. Pasiūlytas vienetų testų generavimo sprendimas, kuris:
   - ✓ Pritaikytas Android aplikacijoms. Naudoja Android SDK bei netikrus objektus.
   - ✓ Naudoja atsitiktinį generavimą
   - ✓ Naudoja OCL apribojimus
   - ✓ Naudoja regresinį testavimą
6. Pasiūlyto vienetų testų generavimo sprendimo efektyvumas įrodytas eksperimentiniu tyrimu. Šio eksperimento metu buvo testuojamos skirtingų 4 aplikacijų paketai ir buvo pasiekti tyrimo rezultatai:
   - ✓ Vidutiniškai 75% sugautų mutantų. (Mažiausia reikšmė – 69%, didžiausia – 88%)
   - ✓ Vidutiniškai 85% kodo eilučių padengimo. (Mažiausia reikšmė – 72%, didžiausia – 97)
7. Tyrimas buvo atliekamas su realiomis Android aplikacijomis, kas įrodo, jog įrankis gali būti naudojamas praktiškai.
8. Ateityje yra galimybė papildyti vienetų testų siūlomą generavimo metodą numatant galimybę testuoti Android grafinės sąsajos elementus bei patobulinti OCL interpretatorių.

7. LITERATŪRA

[1] **4viewsoft** AChartEngine Charting library for Android, [Žiūrėta 2012 05 02], Prieiga per internetą: <http://code.google.com/p/achartengine/>

[2] **Dephnat N. C., Garis A., Montejano G.** Defining OCL constraint for the Proxy Design Profile, *IEEE/ACS International Conference on Computer Systems and Applications, Winona, Minnesota, 2007.*

[3] **Morrison G. C.** Automated Coverage Calculation and Test Case Generation, *Stellenbosch University*, 2012.

[4] **Cheon Y., Kim M. Y., Perumandla A.** A Complete Automation of Unit Testing for Java Programs, *International Conference on Software Engineering Research and Practice, Las Vegas, Nevada, USA, 2005.*

[5] **Csallner C., Smaragdakis Y.** An automatic robustness tester for Java, [Žiūrėta 2012 04 01], Prieiga per internetą: <http://ranger.uta.edu/~csallner/jcrasher/index.html>

[6] **Ehringer D.** The Dalvik virtual machine architecture, *Google I/O 2010, 2010.*

[7] **Babenskas E., Packevičius Š.** Automatinis testų generavimas testuojant Android OS aplikacijas, *XVII Tarpuniversitetinė magistrantų ir doktorantų konferencija „Informacinės Technologijos 2012“*, 2012.

[8] **Friesen J.** Jtest statically and dynamically analyzes your Java code, [Žiūrėta 2012.03.11], Prieiga per internetą: <http://www.javaworld.com/javaworld/jw-12-2002/jw-1206-java101.html>

[9] **Goasduff L., Pettey C.** Gartner Says Worldwide Smartphone Sales Soared in Fourth Quarter of 2011 With 47 Percent Growth, [Žiūrėta 2012.03.09], Prieiga per internetą: <http://www.gartner.com/it/page.jsp?id=1924314>

[10] **Google** Android Apps on Google Play, [Žiūrėta 2012 05 06], Prieiga per internetą: <https://play.google.com/store>

[11] **Google** Android developers, [Žiūrėta 2012 04 26], Prieiga per internetą: <https://developer.android.com>

[12] **Google** Project Hosting on Google Code, [Žiūrėta 2012 05 02], Prieiga per internetą: <http://code.google.com/hosting/>

[13] **Google** Testing Fundamentals, [Žiūrėta 2012 04 09], Prieiga per internetą: <http://developer.android.com/guide/topics/testing/testing_android.html>

[14] **Yamazaki Y., Sakurai K., ir kiti**. A Unit Testing Framework for Aspects without Weaving, *Proceedings of the 1st Workshop on Testing Aspect Oriented Programs - in Conjunction with AOSD'2005, Chicago, IL, USA, 2005.*

[15] **Wang Y., Lelli F., Kölle C.** JTestCase, [Žiūrėta 2012 04 01], Prieiga per internetą: <http://jtestcase.sourceforge.net/>

[16] **Bell J.** JDJ Product Review — Parasoft Jtest 8.0, [Žiūrėta 2012 01 01], Prieiga per internetą: <http://java.sys-con.com/node/299985>

[17] **Harty J.** A Practical Guide to Testing Wireless Smartphone Applications. *Morgan & Claypool Publishers*, 2009.

[18] **Jaygarl H., Chang C. K., Kim S.** Practical Extensions of a Randomized Testing Tool, *2009 33rd Annual IEEE International Computer Software and Applications Conference, Seattle,Washington, 2009.*

[19] **Mačiulis S. ir kiti** Alaus radaras, [Žiūrėta 2012 05 02], Prieiga per internetą: <http://code.google.com/p/alaus-radaras/>

[20] **Saskatchewan universitetas CMPT371 komanda** Luminance - Puzzle Game, [Žiūrėta 2012 05 04], Prieiga per internetą: <http://code.google.com/p/cmpt371t1/>

[21] **Last M., Eyal S., Kandel A.** Effective Black-Box Testing with Genetic Algorithms, *2005.*

[22] **Tyborowski M.** JUB (JUnit test case Builder), [Žiūrėta 2012 04 12], Prieiga per internetą: <http://jub.sourceforge.net/>

[23] **MA Y. S., Offutt J.** Description of Method-level Mutation Operators for Java, *2005.*

[24] **Ma Y. S., Offutt J., Kwon Y. R.** MuJava: A Mutation System for Java, *Proceedings of the 28th international conference on Software engineering, New York, USA, 2006.*

[25] **Omnitel** 2011 III ketvirčio veiklos ataskaita, [Žiūrėta 2012.03.09], Prieiga per internetą: <http://www.omnitel.lt/apie-omnitel/apie-bendrove/ziniasklaidai/veiklos-rezultatai/2011-iii-ketvirtis/55339>

[26] **Packevičius Š., Ušaniov A., Bareiša E.** The Use of Model Constraints as Imprecise Software Test Oracles, *Information Technology and Control, 2007.*

[27] **Parasoft** Parasoft® Jtest®: Java Testing, Static Analysis, Code Review, [Žiūrėta 2012 03 05], Prieiga per internetą: <http://www.parasoft.com/jsp/products/jtest.jsp>

[28] **Harold E. R.** Measure test coverage with Cobertura, [Žiūrėta 2012 04 01], Prieiga per internetą: <http://www.ibm.com/developerworks/java/library/j-cobertura/>

[29] **Rainsberger J.B.** JUnit Recipes: Practical Methods for Programmer Testing. *Manning Publications*, 2005.

[30] **Robobrain-sdk** An Android 2D Game Engine, [Žiūrėta 2012 05 04], Prieiga per internetą: <http://code.google.com/p/robobrain-sdk/>

[31] **Shahamiri S. R., Kadir W. M. N. W., Mohd-Hashim S. Z.** A Comparative Study on Automated Software Test Oracle Methods, *2009 Fourth International Conference on Software Engineering Advances, 2009.*

[32] **Milano D. T.** Android Application Testing Guide. *Birmingham, UK*, 2001.

[33] **Xie T.** Improving Effectiveness of Automated Software Testing in the Absence of Specifications, *Doctoral Dissertation, University of Washington, 2005.*

[34] **Kim T.H.** Application of Genetic Algorithm in Software Testing *Dept. of Multimedia Engineering, Hannam University, 2009.*

[35] **Roubtsov V.** EMMA: a free Java code coverage tool, [Žiūrėta 2012 05 06], Prieiga per internetą: <http://emma.sourceforge.net/>

[36] **Wang S., Offutt J.** Comparison of Unit-Level Automated Test Generation Tools, *IEEE International Conference on Software Testing, Verification, and Validation Workshops, Denver, Colorado 2009.*

[37] **Whittaker J.A., Voas J.M.** 50 years of software: key principles for quality, *IT Professional, Melbourne, 2002.*

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

[39] **Xie T., Notkin D.** Tool-Assisted Unit Test Selection Based on Operational Violations, *18th IEEE International Conference on Automated Software Engineering, Montreal, Que., Canada 2003.*

[40] **Hongh Z., Patrick A.V.** Software Unit Test Coverage and Adequacy, *ACM Computing Surveys*, 1997.

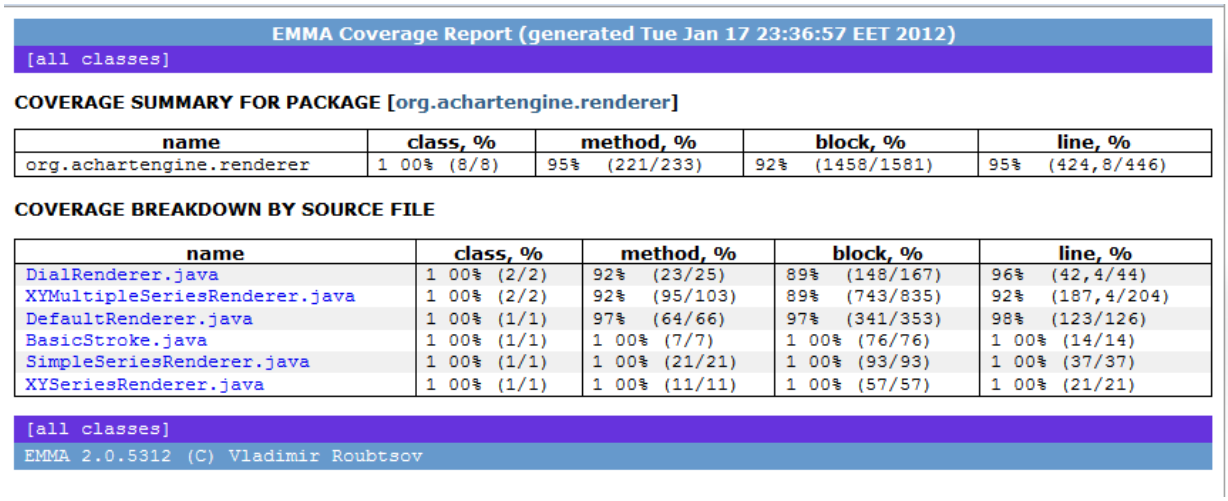
8. TERMINŲ IR SANTRUMPŲ ŽODYNAS

1. **ADT (Android Development Tools)** – tai Google kompanijos palaikomas Eclipse IDE įskiepis, kuris palengvina ir integruoja įrankius skirtus Android aplikacijoms kurti. [11]
2. **Android OS** – Google kompanijos sukurta operacinė sistema skirta mobiliesiems išmaniesiems telefonams. [13]
3. **Baltos dėžės testavimas** – Tai testavimo būdas, kai testuojant yra nagrinėjamas programos kodas.
4. **Dalvik** – tai yra virtuali mašina, kuri yra naudojama Android OS.
5. **D-ART (Distance-bases Adaptive Random Testing)** – Atstumu paremtas prisitaikantis atsitiktinis testavimas
6. **Defektais remtas testavimo principas (*angl. Fault-based testing)*** – tai testavimo strategija, kuomet sukuriami testavimo atvejai skirti atskleisti iš anksto numatytas arba labiausiai tikėtinas klaidas. Pavyzdžiui tikrinama, kaip programa susitvarko su dalyba iš nulio klaida.
7. **EMMA** – tai atviro kodo įrankis, skirtas paskaičiuoti ir pateikti ataskaitą kiek Java kodo yra padengiama testai. [35]
8. **IDE (Integrated Development Environment)** – Integruotos kūrimo aplinkos. Tai programos, turinčios daug priemonių, palengvinančių ir pagreitinančių programinės įrangos kūrimo procesą.
9. **Juodos dėžės testavimas** – Tai testavimo būdas, kai testuojant yra naudojami vartotojo reikalavimai ir specifikacijos.
10. **Mutacinis testavimas (*angl. mutation testing*)** – tai klaidomis paremta technika, kuri leidžia išmatuoti vienetų testų efektyvumą. Klaidos yra įterpiamos į programos kodą, taip sukuriant klaidingą programos versiją. Tokia programos versija vadinama mutantu. Vėliau leidžiami testai ir matuojama kiek mutantų buvo nužudyta. [24]
11. **Mutantas** – taip vadinamas mutaciniame testavime modifikuotos programavos versija, kurioje atliktas nedidelis pakeitimas, kaip pakeistas operatoriaus ženklas, ištrintas kreipinys ir panašiai.
12. **Netikri objektai (Mockup object)** - tai objektai yra skirti imituoti objektus ir naudojami vietoje realių objektų taip leidžiant izoliuoti ir atskirti testuojamus vienetus. [32]
13. **OCL (Object Contraint Language)** – objektų apribojimų kalba.

14. **OMG (Object Management Group)** – organizacija atsakinga už CORBA (Common Object Request Broker Architecture), Unified Modeling Language (UML), ir Model-Driven Architecture (MDA).
15. **OS** – operacinė sistema.
16. **Regresinis testavimas (*regresion testing*)** - testavimo rūšis, kurios pagrindinė paskirtis – patikrinti, ar naujas kodas išlaiko ankstesnėse versijose veikiantį funkcionalumą.
17. **SQLite –** tai reliacinė duomenų bazių valdymo sistema. SQLite yra C kalba parašyta biblioteka, sukurianti duomenų bazę faile. Duomenys valdomi naudojant SQL.
18. **Testų orakulas** – tai priemonė, kurios pagalba nustatoma, ar testas pateikė laukiamus rezultatus. Tai yra orakulas pasako ar testas praėjo, ar nepraėjo. [26].
19. **Vienetų testavimas (Unit testing)** - tai metodas, kai testuojami atskiri programos vienetai. Dažniausiai testuojami programos metodai. [4].

9. PRIEDAI

9.1 EMMA kodo padengimo skaičiavimo įrankio pateikiamos ataskaitos pavyzdys



EMMA Coverage Report (generated Tue Jan 17 23:36:57 EET 2012)

[all classes]

COVERAGE SUMMARY FOR PACKAGE [org.achartengine.renderer]

name	class, %	method, %	block, %	line, %
org.achartengine.renderer	1 00% (8/8)	95% (221/233)	92% (1458/1581)	95% (424,8/446)

COVERAGE BREAKDOWN BY SOURCE FILE

name	class, %	method, %	block, %	line, %
DialRenderer.java	1 00% (2/2)	92% (23/25)	89% (148/167)	96% (42,4/44)
XYMultipleSeriesRenderer.java	1 00% (2/2)	92% (95/103)	89% (743/835)	92% (187,4/204)
DefaultRenderer.java	1 00% (1/1)	97% (64/66)	97% (341/353)	98% (123/126)
BasicStroke.java	1 00% (1/1)	1 00% (7/7)	1 00% (76/76)	1 00% (14/14)
SimpleSeriesRenderer.java	1 00% (1/1)	1 00% (21/21)	1 00% (93/93)	1 00% (37/37)
XYSeriesRenderer.java	1 00% (1/1)	1 00% (11/11)	1 00% (57/57)	1 00% (21/21)

[all classes]

EMMA 2.0.5312 (C) Vladimir Roubtsov

9.2 AChartEngine aplikacijos klasei XYSeriesRenderer naudotas OCL failo pavyzdys

XYSeriesRenderer.ocl turinys:

```
context XYSeriesRenderer::getVisualTypeForIndex(index:Integer):Type
        inv: index > 0

context XYSeriesRenderer::getFillBelowLineColor():Integer
        post: result > 0

context XYSeriesRenderer::getLineWidth():Real
        post: result > 0

context XYSeriesRenderer::setLineWidth(lineWidth:Real)
        inv: lineWidth > 0
```

9.3 Straipsnis „Automatinis testų generavimas testuojant Android OS aplikacijas“

Pateiktas ir pristatytas straipsnis 17-oji Tarpuniversitetinėje magistrantų ir doktorantų konferencijoje „Informacinės technologijos 2012“ bei išspausdintas knygoje “Informacinės technologijos 2012. XVII Tarpuniversitetinė magistrantų ir doktorantų konferencija. Konferencijos medžiaga“, ISSN 2029-249X. Sekcijoje „Informacinių technologijų taikymas“ straipsnis konferencijoje pripažintas geriausiu.

AUTOMATINIS TESTŲ GENERAVIMAS TESTUOJANT ANDROID OS APLIKACIJAS

Egidijus Babenskas[1], **Šarūnas Packevičius** [2]

[1]*Kauno technologijos universitetas, Programų inžinerijos katedra, Studentų gatvė 50, Kaunas, Lietuva,* [1]*egidijus.babenskas@gmail.com,* [2]*sarunas.packevicius@ktu.lt*

Santrauka (abstract). Tobulėjant išmaniesiems telefonams ir jų techninėms galimybėms bei didėjant jų pardavimams Lietuvoje ir pasaulyje, kuriamos aplikacijos tampa funkcionalesnės, todėl tokių programų testavimas tampa sudėtingesnis ir yra reikalingos priemonės leidžiančios palengvinti šių aplikacijų testavimą. Panagrinėjus esamą rinką ir nustačius, jog šiuo metu nėra paprasto ir visiems prieinamo įrankio, siūlomas sprendimas sukurti tokį įrankį. Šiame straipsnyje pateikiami panašūs įrankiai esantys rinkoje, nagrinėjami vienetų testų generavimo algoritmai bei siūlomas testavimo įrankio sprendimas. Pateikiamas eksperimentinis tyrimas, kurio metu nagrinėjama ar pasirinkti testų generavimo metodai yra tinkami bei kiek procentų kodo galima padengti pasinaudojus sukurtu įrankiu.

Raktiniai žodžiai: vienetų testai, testavimas, automatinis testų generavimas, Android, generavimo metodai, mobiliosios aplikacijos, atsitiktinis, OCL, genetinis

1. Įžanga

Šiuo metu ypač didėja išmaniųjų telefonų pardavimai. Žmonės vis dažniau renkasi ne paprastus telefonus, o išmaniuosius su įvairiomis operacinėmis sistemomis, tokiomis kaip Android, iOS, Symbian ar Maemo. Pagal garsios tyrimų kompanijos „Gartner“ pateiktą ataskaitą [3] per 2011 metų paskutinįjį ketvirtį parduota 149 milijonai išmaniųjų telefonų, tai 47 procentais daugiau palyginus su 2010 metų paskutiniu ketvirčiu.

Lietuvoje, kaip teigia Omnitel veiklos ataskaita [7], per trečiąjį 2011 metų ketvirtį lyginant su tuo pačiu laikotarpiu pernai išmaniųjų telefonų pardavimai augo 8 proc., o per tris pirmuosius metų ketvirčius, lyginant su 2010 m. tuo pačiu laikotarpiu, išaugo 46 procentais. Pagal „Gartner“ kompanijos ataskaitą [3] iš visų per 2011 ketvirtąjį ketvirtį parduodamų išmaniųjų telefonų 50,9 procentai buvo parduodami su Google kompanijos Android operacine sistema (OS). Kaip matome šiuo metu populiarėja išmanieji telefonai su Android OS.

Tobulėjant išmaniesiems telefonams ir jų techninėms galimybėms, kuriama programinė įranga tampa sudėtingesnės ir funkcionalesnės, tačiau kokybės problema vis dar išlieka skaudžia programinės įrangos kūrimo dalimi. Sukurta programinė įranga dažniausiai yra pateikiama su aibe klaidų, o kokybiška programinė įranga yra daugiau išimtis, nei įprastinis dalykas. Pačios programinės įrangos kūrimo technologijos tobulėja, kurios įgalina paprasčiau kurti programinę įranga, o tuo pačiu turėtų leisti kurti ir kokybiškesnę, tačiau kaip buvo kuriama programinė įranga su klaidomis, taip ir išlieka kokybės problema, net ir taikant naujausias kūrimo ir projektavimo technologijas. [9]

Šiame straipsnyje nagrinėjami trys metodai: atsitiktinio generavimo, panaudojant OCL apribojimus ir genetinis. Šiais algoritmais remiantis generuojami testiniai atvejai iš Android OS aplikacijos kodo. Apžvelgiami rinkoje esantys panašūs įrankiai atkreipiant dėmesį į jų suderinamumą bei gebėjimą generuoti testinius atvejus.

Matant Android OS programų vis didėjantį poreikį rinkoje ir jų populiarumą bei panagrinėjus esamą rinką ir pamačius, jog testavimo įrankių, skirtų testuoti Android aplikacijas, beveik nėra, buvo nuspręsta, jog reikalingas testavimo įrankis, kuris automatiškai leistų generuoti vienetų testus. Vienetų testavimas – tai metodas, kai testuojami atskiri programos vienetai (dažniausiai metodai)[1]. Įrankis palengvins Android OS aplikacijų testavimą, padės greičiau aptikti klaidas kuriant Android OS aplikacijas. Taip pat suteiks galimybę valdyti testus, analizuoti testavimo rezultatus bei pačių testų efektyvumą.

Realizavus siūlomą įrankį buvo atliktas testavimo įrankio eksperimentinis efektyvumo tyrimas, kurio tikslas buvo nustatyti, ar pasirinkti testų generavimo metodai yra tinkami bei kiek procentų kodo galima padengti pasinaudojus sukurtu įrankiu. Šio eksperimento aprašymas bei jo rezultatai detaliau nagrinėjami šiame straipsnyje. Pabaigoje pateikiamos išvados bei ateities planai.

2. Testų generavime naudojami metodai

Testų generavime sprendžiama problema, kaip ir pagal ką generuoti testinius duomenis bei koks naudojamas orakulas. Orakulas – tai priemonė, kurios pagalba nustatoma, ar testas pateikė laukiamus rezultatus [8]. Remiantis autoriais [4] [5] [1], kurie aprašo testų generavimą bei mobilių aplikacijų testavimą, buvo pasirinkti trys testų generavimo būdai: atsitiktinis generavimas [5], generavimas pasinaudojant OCL apribojimais [1] [8] bei generavimas remiantis genetiniu algoritmu [6] [1].

2.1. Atsitiktinio generavimo metodas

Atsitiktinio generavimo metodo metu automatiškai generuojami testų įėjimo duomenys, parenkant testinius atvejus atsitiktinai iš testuojamos programos įėjimo srities. Koncepcija ir realizacija yra palyginus paprasta su kitomis egzistuojančiomis technikomis. [5]

Atsitiktinio generavimo algoritmo metu kiekvienam metodui analizuojami įėjimai. Jeigu įėjimas - primityvus kintamasis, jis iškarto sugeneruojamas, jeigu objektas – ieškoma objekto konstruktoriaus ir suradus konstruktorių sugeneruojamas objektas. Testo orakulas tikrina pagal metodo tipą: jeigu metodas grąžiną reikšmę, tai tikrinama, ar ta reikšmė grąžinama, jeigu ne – tuomet tikrinama, ar neįvyko išimtis (exception). Pagal sugeneruotus įėjimus ir testo orakulą suformuojamas testas testuojamai klasei. Atsitiktinis generavimo metodas plačiai naudojamas, kadangi tai paprasčiausiai realizuojamas ir nėra reikalingas žmogaus įsikišimas. [5]

2.2. Generavimas panaudojant OCL apribojimus

OCL – tai OMG organizacijos pasiūlytas standartas, kuriuo užrašomi objekto apribojimai. OCL pagalba galima užrašyti apribojimus. [8]

Pavyzdžiui turime klasę:

```
public class XYMultipleSeriesRenderer {
    public double setXAxisMin (int scale);
};
```

OCL leidžia aprašyti apribojimus aprašant inv ir post sąlygas. Aprašymo pavyzdys gali būti:

```
context XYMultipleSeriesRenderer::setXAxisMin(scale:Integer):Real
inv: scale >= 0
post: result >= 0
```

Post sąlygos naudojamos orakulo, o inv sąlygos naudojamos generuojant duomenis. Panaudojant OCL apribojimus, neatitinkami apribojimų sugeneruoti duomenys atmetami, taip atrenkami tinkami įėjimo testavimo duomenys [8]. Testų orakului taip pat panaudojami OCL apribojimai, pagal juos orakulas žino, kuriuos metodo rezultatus laikyti teisingais, o kuriuos – neteisingais.

2.3. Genetinis algoritmas

Genetinis algoritmas ir jo sudarymo principas XX-ame amžiuje buvo pasiūlytas Hollando. Jo veikimas yra pagrįstas evoliucijos, vykstančios gyvojoje gamtoje, t. y., natūraliosios atrankos proceso imitavimu. [1] Šio algoritmo principiniai veiksmai remiantis [6] šaltiniu ir pritaikant testų generavimui:

1. [Pradžia] Sugeneruoti pradinių testų populiaciją
2. [Tinkamumas] Apskaičiuoti, kiek kodo padengia kiekvienas testas
3. [Nauja populiacija] Sukurti naują populiaciją, atrenkant daugiausiai kodo padengiančius testus ir juos kryžminant tarpusavyje, pagal sugeneruotas įėjimo reikšmes metoduose.
4. [Pakeitimas] Tolimesniame darbe naudoti naujai sugeneruotą populiaciją.
5. [Tikrinti] Jei padengimo sąlyga tenkinama arba populiacijos kartų skaičius peržengtas, sustabdyti algoritmą ir grąžinti geriausią testą iš einamos populiacijos. Jeigu netenkinama – grįžti į žingsnį 2.

Naudojant genetinį generavimą gaunamas minimalus testinių duomenų kiekis, kuris apima kiek įmanoma daugiau klaidų ir yra efektyvūs. Nėra nereikalingų ir besidubliuojančių testinių duomenų.

3. Susijusių įrankių apžvalga

Šiuo metu pagrindinis įrankis skirtas testuoti Android OS parašytoms aplikacijoms yra Android SDK palaikomas JUnit testų rašymas. Kitokių įrankių, kurių paskirtis automatiškai generuoti testus Android OS aplikacijoms, surasti nepavyko. Buvo nuspręsta panagrinėti kitus artimus Android OS testavimo įrankius, kurie skirti Java kalba parašytoms aplikacijoms testuoti.

3.1. JUnit

JUnit tai testavimo karkasas, sukurtas rašyti automatinius, save patikrinančius testus Javos kalboje. Testai yra rašomi kiekvienam programos vienetui savarankiškai, todėl jie gali būti paleisti dalimis arba visi iš karto. JUnit privalumai: paprastas naudoti, nemokamas, sugeneruoja nurodytai klasei testų karkasą, palaikomas standartiškai Android SDK. JUnit trūkumai: automatiškai negeneruoja testuojamų duomenų, nėra išsamesnio testų vykdymo (orakulo), neturi testų efektyvumo įvertinimo.

3.2. Parasoft Jtest

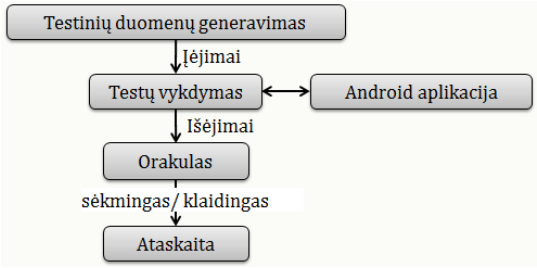
Tai „Parasoft“ kompanijos įrankis, kuris integruoja plataus pritaikymo automatinius sprendimus ir apima vienetų testo atvejų generavimą, statinę kodo analizę, vykdymo klaidų aptikimą ir kodo peržiūrą. JTester privalumai: palaiko Spring, Struts ir Hibernate projektu, automatinis vienetų testų generavimas [2], statinė kodo analizė [2], vykdymo klaidų aptikimai [2]. JTester trūkumai: mokamas, nėra funkcionalesnio testų vykdymo (orakulo) [10], reikalingi nemaži kompiuterio resursai didesniems testams vykdyti [10].

3.3. TestGen4j

TestGen4j yra atviro kodo įrankių kolekcija, kuri automatiškai generuoja vienetų atvejų testus. Generuojant testus dėmesys skiriamas ribinėms metodų įvedimo argumentų reikšmėms. TestGen4j privalumai: nemokamas, vartotojas gali nurodyti ribines sąlygas, testiniai duomenys atskirti nuo testų TestGen4j trūkumai: nesuderinamas su Android SDK, silpnos generavimo konfigūravimo galimybės.

4. Testavimo įrankis

Siūlomo sprendimo įgyvendinimui buvo realizuotas Eclipse įskiepis pavadintas AATest. Šio įskiepio tikslas – palengvinti Android OS aplikacijų testavimą. Jis ne tik generuoja automatinius testus, tačiau kartu yra ir testų valdymo priemonė. Testavimo proceso naudojantis įrankiu principinė schema pateikiama 1 paveikslėlyje.



1 pav. Testavimo procesas naudojantis įrankiu

Pirmiausia sugeneruojami testiniai duomenys, naudojant aprašytus metodus bei suformuojamas testas. Testai leidžiami naudojant testuojamą programą. Pagal aprašytas sąlygas orakulas pasako, ar testas praėjo sėkmingai, ar nesėkmingai. Pabaigoje parodoma praleistų testų ataskaita.

Įrankyje realizuoti trys aptarti testinių atvejų generavimo metodai su galimybe lengvai papildyti naujais metodais. Šio įrankio pagrindinės vartotojo funkcijos: testų generavimas pagal pasirinktą metodą, testų tvarkymas, testų vykdymas ant emuliatoriaus arba realaus prietaiso, kodo padengimo testais skaičiavimas bei pateikti ataskaitas vartotojui. Visi sugeneruoti testai suderinami su Android testavimo priemonėmis, todėl vartotojas juos gali pats papildyti, pataisyti ir taip praplėsti. Įrankis platinamas nemokamai ir visiems prieinamas.

5. Testavimo įrankio efektyvumo tyrimas

Realizavus įskiepį buvo atliekamas eksperimentinis tyrimas, kurio tikslas – nustatyti, kiek procentų Android OS aplikacijos kodo galima padengti generuojant testus be vartotojo įsikišimo bei palyginti realizuotus algoritmus.

Bandymai buvo atliekami ant to paties kompiuterio su reikalinga programine įranga. Kompiuterio techniniai parametrai: CPU - Intel Core 4 branduolių, RAM- 4GB, Windows 7 64bit, Eclipse Helios, Android SDK r16, ADT r16. Testuojama buvo ant realaus mobilaus telefono, kurio techninės charakteristikos: CPU – 800Mhz, RAM– 512Mb, Android 2.3.

Android aplikacijos buvo pasirinktos nemokamos ir lengvai prieinamos iš Google programų talpyklos. Pirmiausia buvo generuojami testai visoms aplikacijos klasėms ir skaičiuojamas jų padengimas. Eksperimento rezultatai pateikiami 1 lentelėje. Kaip matome priklausomai nuo programos, padengimas svyruoja nuo 25 iki 47 procentų.

Lentelė Nr.1 Padengimo rezultatai skirtingiems projektams

Aplikacija	Klasių kiekis	Android versija	Generavimo laikas	Padengimo skaičiavimo laikas	Projekto kodo eilučių padengimas testais
AChartEngine	55	2.1	1.54 s.	24.16 s.	40% (963/2416)
Alaus-radaras	144	2.1	4.52 s.	55.88 s	25% (550/2198)
Robobrain-sdk	46	2.1	1.75 s.	35.98 s.	47% (603/1276)
Luminance	80	2.1	2.77 s.	60.13 s.	34% (1421/4156)

Kitam bandymui buvo pasirinktos aplikacijos: AChartEngine (org.achartengine.render), Alaus-radaras (alaus.radaras.submition), Luminance (ca.luminance.input), Robobrain-sdk (org.robobrain.sdk.graphics) ir jų vienas paketas ištestuotas visais metodais. OCL metodui buvo sukurti apribojimų failai, o genetiniam buvo parinkta 10 testų populiacija. Bandymo rezultatai pateikiami 2 lentelėje.

Lentelė Nr.2 Padengimo rezultatai naudojant skirtingus metodus

Aplikacija	Atsitiktinis metodas		Genetinis metodas		OCL metodas	
	Testų generavimo laikas, s	**Kodo padengimas, %**	**Testų generavimo laikas, s**	**Kodo padengimas, %**	**Testų generavimo laikas, s**	**Kodo padengimas, %**
AChartEngine	0.58 s	94%	1783.91 s	95%	0.87 s	97%
Alaus-radaras	0.42 s	71%	821.87 s	75 %	0.76 s	78 %
Luminance	0.61 s	77%	1352.12 s	79 %	0.92 s.	85 %
Robobrain-sdk	0.67 s	57%	2621.71 s	62%	1.12 s	68%

Kaip matome, atsitiktiniu metodu galima padengti nuo 57 % iki 94 % programos kodo, naudojant genetinį algoritmą padengimą galima padidinti 5%, o aprašius OCL apribojimus padidinti apie 10 %. Kaip matome genetinis algoritmas truko labai ilgai, nors buvo naudojama tik 10 testų populiacija, o norint geresnių rezultatų populiacija turėtų būti žymiai didesnė. Taip nutiko todėl, jog yra reikalingas padengimo skaičiavimas po kiekvieno sugeneruoto testo, o vieno testo padengimą suskaičiuoti užtrunka ~ 45 sekundes.

6. Išvados ir ateities darbai

Programos testavimas yra svarbus ir daug laiko sunaudojantis procesas. Tam skiriama apie pusė programų sistemos kūrimo resursų. Išmanieji telefonai su Android OS populiarėja, tačiau kartu sudėtingėja ir kuriamos aplikacijos. Siūlomas įrankis padės sumažinti su testavimu susijusius kaštus ir palengvins programuotojų bei testuotojų darbą.

Atlikus eksperimentinį tyrimą galima teigti, jog naudojantis atsitiktinio testų generavimo metodu be vartotojo įsikišimo testais galima padengti iki 50% testuojamos programos kodo. Panaudojus OCL apribojimus vartotojas gali testų generavimą individualizuoti pagal testuojamą programos kodą, taip padidindamas jos padengimą testais. Tačiau genetinis metodas, dėl naudojamo lėto padengimo skaičiavimo (vieno testo padengimas vidutiniškai trunka ~45s) kol kas nėra labai tinkamas, nes vienos klasės su 10 testų populiacija sugeneravimas trunka apie 5 minutes.

Ateities darbai: papildyti atsitiktinį vienetų testų generavimą panaudojant prisitaikantį atsitiktinį generavimą [5] bei atsiradus greitesniam padengimo skaičiavimui Eclipse programavimo aplinkoje pertvarkyti genetinio algoritmo realizaciją.

7. Literatūros sąrašas

[1] **Cheon Y., Kim M. Y., Perumandla A.** A Complete Automation of Unit Testing for Java Programs, *International Conference on Software Engineering Research and Practice, Las Vegas, Nevada, USA, 2005.*

[2] **Friesen J.** Jtest statically and dynamically analyzes your Java code, [Žiūrėta 2012.03.11], Prieiga per internetą: http://www.javaworld.com/javaworld/jw-12-2002/jw-1206-java101.html

[3] **Goasduff L., Pettey C.** Gartner Says Worldwide Smartphone Sales Soared in Fourth Quarter of 2011 With 47 Percent Growth, [Žiūrėta 2012.03.09], Prieiga per internetą: http://www.gartner.com/it/page.jsp?id=1924314

[4] **Harty J.** A Practical Guide to Testing Wireless Smartphone Applications. *Morgan & Claypool Publishers*, 2009.

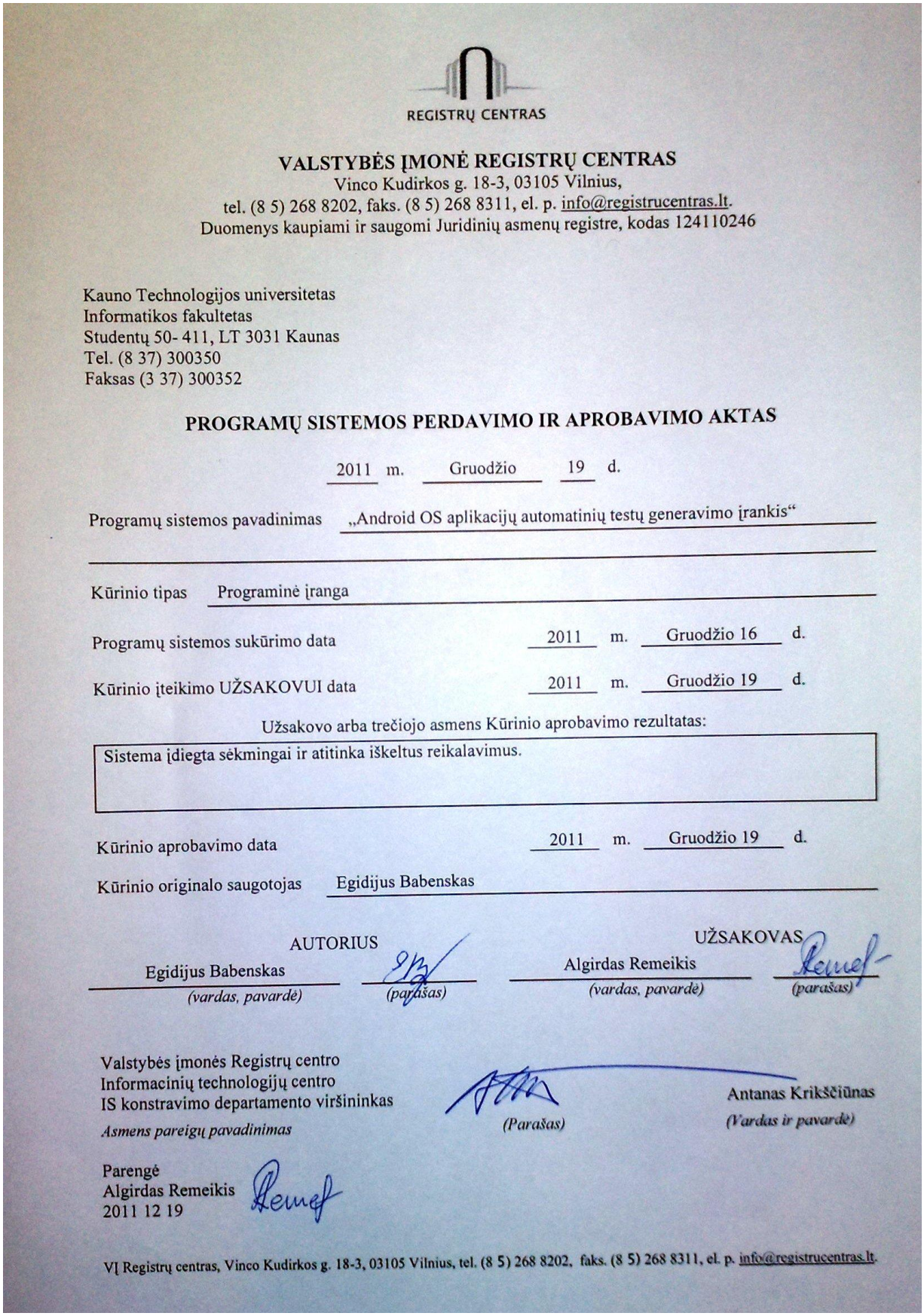
[5] **Jaygarl H., Chang C. K., Kim S.** Practical Extensions of a Randomized Testing Tool, *2009 33rd Annual IEEE International Computer Software and Applications Conference, Seattle,Washington, 2009.*
[6] **Last M., Eyal S., Kandel A.** Effective Black-Box Testing with Genetic Algorithms, *2005.*
[7] **Omnitel** 2011 III ketvirčio veiklos ataskaita, [Žiūrėta 2012.03.09], Prieiga per internetą: http://www.omnitel.lt/apie-omnitel/apie-bendrove/ziniasklaidai/veiklos-rezultatai/2011-iii-ketvirtis/55339
[8] **Packevičius Š., Ušaniov A., Bareiša E.** The Use of Model Constraints as Imprecise Software Test Oracles, *Information Technology and Control, 2007.*
[9] **Whittaker J.A., Voas J.M.** 50 years of software: key principles for quality, *IT Professional, Melbourne, 2002.*
[10] **Xie T., Notkin D.** Tool-Assisted Unit Test Selection Based on Operational Violations, *18th IEEE International Conference on Automated Software Engineering, Montreal, Que., Canada 2003.*

Automated Test Generation in Android Application Testing

While having smart phones and their technical capabilities being improved and their sales are increasing in Lithuania and all over the world, the developed applications become rich in functionality, therefore the testing of these applications become complex and the tools are needed to facilitate the testing of such applications. The finding of the existing market shows that currently there is no simple tool available to everybody so decision is proposed to create such a tool. This article covers the similar tools in the market, the examination of unit test generation algorithms and the proposed testing tool. A pilot study is provided, where it is examined whether the chosen test generation algorithms are appropriate and the percentage of the code being covered using the proposed and developed tool.

9.4 Programų sistemos perdavimo ir aprobavimo aktas

Originalas prisegtas prie spausdinto varianto vietoje šito puslapio.



REGISTRŲ CENTRAS

VALSTYBĖS ĮMONĖ REGISTRŲ CENTRAS
Vinco Kudirkos g. 18-3, 03105 Vilnius,
tel. (8 5) 268 8202, faks. (8 5) 268 8311, el. p. info@registrucentras.lt.
Duomenys kaupiami ir saugomi Juridinių asmenų registre, kodas 124110246

Kauno Technologijos universitetas
Informatikos fakultetas
Studentų 50- 411, LT 3031 Kaunas
Tel. (8 37) 300350
Faksas (3 37) 300352

PROGRAMŲ SISTEMOS PERDAVIMO IR APROBAVIMO AKTAS

2011 m. Gruodžio 19 d.

Programų sistemos pavadinimas „Android OS aplikacijų automatinių testų generavimo įrankis“

Kūrinio tipas Programinė įranga

Programų sistemos sukūrimo data 2011 m. Gruodžio 16 d.

Kūrinio įteikimo UŽSAKOVUI data 2011 m. Gruodžio 19 d.

Užsakovo arba trečiojo asmens Kūrinio aprobavimo rezultatas:

Sistema įdiegta sėkmingai ir atitinka iškeltus reikalavimus.

Kūrinio aprobavimo data 2011 m. Gruodžio 19 d.

Kūrinio originalo saugotojas Egidijus Babenskas

AUTORIUS

Egidijus Babenskas (vardas, pavardė) (parašas)

UŽSAKOVAS

Algirdas Remeikis (vardas, pavardė) (parašas)

Valstybės įmonės Registrų centro
Informacinių technologijų centro
IS konstravimo departamento viršininkas
Asmens pareigų pavadinimas

(Parašas)

Antanas Krikščiūnas
(Vardas ir pavardė)

Parengė
Algirdas Remeikis
2011 12 19

VĮ Registrų centras, Vinco Kudirkos g. 18-3, 03105 Vilnius, tel. (8 5) 268 8202, faks. (8 5) 268 8311, el. p. info@registrucentras.lt.