KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

Tomas Kiselevskis

VEIKLOS MODELIU GRINDŽIAMAS PILKOSIOS DĖŽĖS TESTŲ GENERAVIMAS

Baigiamasis magistro projektas

Vadovas
prof. dr. R. Butleris

Konsultantė
dokt. K. Aleliūnė

KAUNAS, 2015

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

VEIKLOS MODELIU GRINDŽIAMAS PILKOSIOS DĖŽĖS TESTŲ GENERAVIMAS

Baigiamasis magistro projektas
Informacinių sistemų inžinerijos studijų programa (kodas 621E15001)

Vadovas
prof. dr. R. Butleris
2015-05-18

Konsultantė
dokt. K. Aleliūnė
2015-05-18

Recenzentas
dr. Š. Packevičius
2015-05-18

Projektą atliko
Tomas Kiselevskis
2015-05-18

KAUNAS, 2015

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

(Fakultetas)

(Studento vardas, pavardė)

Informacinių sistemų inžinerijos studijų programa, 621E15001

(Studijų programos pavadinimas, kodas)

Baigiamojo projekto „Pavadinimas“

AKADEMINIO SĄŽININGUMO DEKLARACIJA

20 ___ m. _____ ____ d.

Kaunas

Patvirtinu, kad mano, **Tomo Kiselevskio**, baigiamasis projektas tema „VEIKLOS MODELIU GRINDŽIAMAS PILKOSIOS DĖŽĖS TESTŲ GENERAVIMAS“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

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

_____ _____

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

Kiselevskis, T. Gray-box Test Generation Based on Business Model. *Final Degree Project of Master of Information Systems Egineering* / Supervisor Prof. Dr. Rimantas Butleris; Kaunas University of Technology, Faculty of Informatics.

Kaunas, 2015. 51 p.

SUMMARY

GRAY-BOX TEST GENERATION BASED ON BUSINESS MODEL

The aim of this work is to accelerate the error search process of information system development by generating test cases from business model. The work analyses methods and UML diagrams that enables generating test cases from requirement model as well as testing environments that enables testing mobile, desktop and web applications.

The algorithm that generates gray-box test cases from UML activity diagrams was created in accordance to performed analysis. Proposed method enables using requirement stage business model for test case development with input, output data and plausible testing scenario. Plausible testing scenarios are generated from activity diagrams using depth-first search method. Input and output data is generated from OCL constraints and modeled actions on activities and transitions. Test generation method proposed in this work has these benefits:

- Enables using requirement stage business model for gray-box test generation
- Business models become richer in detail, without losing primary view

Proposed method was realized with PHP programming language. Manual and automated gray-box test generation was compared. It has been proven that automated test generation saves time and work necessary for gray-box test creation from business model when UML activity diagrams are created at requirement stage. It has been found that created test generation tool is worth being used in all cases.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

TERMINŲ IR SANTRUMPŲ ŽODYNAS

UML (*angl. Unified Modelling Language*) - modeliavimo ir specifikacijų kūrimo kalba, skirta specifikuoti, atvaizduoti ir konstruoti objektiškai orientuotų programų dokumentus.

IS – informacinė sistema.

SDL (*angl. Specification and Description Language*) – reikalavimų specifikacijų ir apibūdinimo kalba, skirta vienareikšmiškai pateikti reikalavimus keliamus informacinei sistemai.

OCL (*angl. Object Constraint Language)* – kalba aprašanti taisykles pritaikytas UML modeliams. UML standarto dalis.

PHP (*angl. hypertext preprocessor*) – dinaminė interpretuojama programavimo kalba

White-box – baltosios dėžės testavimo metodas, taikomas testuoti vidinei programos struktūrai, turint vidinį kodą.

Black-box – juodosios dėžės testavimo metodas, taikomas norint ištestuoti programinę įrangą nežinant vidinio kodo.

Gray-box – pilkosios dėžės testavimo metodas, taikomas norint ištestuoti programinę įrangą, kai procesai vykstantys vidiniame programos kode yra žinomi dalinai.

ĮVADAS

Šis darbas priklauso informatikos studijų programai, informacinių sistemų inžinerijos specializacijai.

Darbo problematika ir aktualumas

Šiais laikais sukuriama daug skirtingų internetinių, darbalaukio bei mobiliesiems telefonams skirtų informacinių sistemų (IS), kurių teisingam veikimui užtikrinti naudojamos testavimo priemonės. Testavimo procesas yra vienas iš svarbiausių visame IS kūrimo periode, todėl, norint užtikrinti tinkamą IS veikimą, testavimo darbų reikėtų imtis vos tik išsiaiškinus konkrečius vartotojų reikalavimus. Turint omenyje, kad IS kūrimas prasideda nuo reikalavimų specifikacijos sudarymo, imant ją kaip pagrindą testavimo atvejų generavimui, esamas klaidas būtų galima nustatyti ankstyvoje IS kūrimo fazėje. Metodas, leidžiantis atlikti aprašytą funkcionalumą, leistų sumažinti laiko bei finansinius kaštus reikalingus IS realizavimui.

Darbo tikslas ir uždaviniai

Šiuo darbu siekiama pasiūlyti metodą, leidžiantį pagreitinti klaidų aptikimą realizuojamoje IS, generuojant testavimo scenarijus iš reikalavimų specifikacijos, įvertinant skirtingas testų kūrimo aplinkas.

Šiam darbui atlikti keliami uždaviniai:

1. Išanalizuoti esamas testavimo priemones bei metodus, leidžiančius automatiškai generuoti testavimo atvejus bei testuoti darbastalio, internetines bei mobiliesiems telefonams skirtas IS.
2. Nustatyti reikalavimus keliamus testų generavimui
3. Nustatyti diagramas, leidžiančias išpildyti testų generavimui keliamus reikalavimus
4. Pasiūlyti metodą, leidžiantį generuoti testavimo atvejus skirtus skirtingose aplinkose kuriamoms sistemoms
5. Realizuoti siūlomo metodo prototipą.
6. Atlikti eksperimentinius tyrimus realizuoto prototipo veikimui.

Darbo rezultatai ir jų svarba

Šiame darbe išanalizuoti metodai, leidžiantys generuoti testus iš reikalavimo specifikacijoje sumodeliuotų UML diagramų bei priemonės gebančios testuoti įvairioms aplinkoms skirtas informacines sistemas. Iš analizės nustatyti reikalavimai keliami generuojamiems testams. Pasirinktas pilkosios dėžės (*angl. gray-box)* testų generavimo metodas. Pasirinktam metodui nustatytas duomenų gavimas iš UML veiklos procesų diagramų. Įvedimo, išvedimo, logikos veiksmams bei duomenims veiklos modeliuose aprašyti pasirinkta OCL 2.4 (*angl. Object Constraint Language)* specifikacijos kalba. Pagal nustatytus reikalavimus sukurtas prototipas PHP programavimo kalba, pasirinktam sprendimui įvertinti. Realizacija išbandyta su įvairaus sudėtingumo veiklos procesų diagramomis. Lyginant realizuotą sprendimą su kitų autorių darbais išryškinti sukurto prototipo privalumai:

1. Nereikalinga kurti papildomų diagramų testų generavimui, naudojamos reikalavimų modelio veiklos diagramos, taip sutaupomas laikas ir darbas.
2. Veiklos procesų diagramos tampa informatyvesnės, neprarandant pradinio sumodeliuoto vaizdo.

Sukurtas įrankis sumažina laiko sąnaudas, reikalingas norint sugeneruoti testavimo scenarijus iš reikalavimų modelio, sumažina bendrą modeliuojamų diagramų kompleksiškumą.

Darbo struktūra

Darbą sudaro 8 skyriai. Pirmame skyriuje analizuojami testų generavimo iš reikalavimų modelių metodai, vartotojai, kitų autorių sprendimai, sistemos gebančios testuoti įvairioms aplinkoms skirtas IS. Antrame skyriuje pateikiama reikalavimų specifikacija siūlomam sprendimui realizuoti, veiklos procesų ir kt. modeliai. Trečiame skyriuje pateikiamas algoritmo generuojančio testavimo scenarijus iš veiklos procesų diagramų aprašas. Ketvirtą skyrių sudaro sprendimo realizacija, testavimo duomenys, testavimo kontrolinis pavyzdys. Penktame skyriuje pateikiamas realizacijos eksperimentinis tyrimas, metodo pagrindimas. Darbo pabaigoje aprašomi šiuo darbu atskleisti privalumai, pateikiamos išvados, naudotos literatūros sąrašas bei priedai.

1. PROBLEMINĖS SRITIES ANALIZĖ

1.1. Analizės tikslas

Šios analizės tikslas yra išanalizuoti, palyginti ir įvertinti esamas informacinių sistemų testavimo priemones, skirtas darbastalio, mobiliųjų telefonų bei internetinių informacinių sistemų automatiniam testavimui. Ištirti testų generavimą paremtą reikalavimais pateiktais UML modeliuose bei remiantis atlikta analize nustatyti koki reikalavimai ir jų pateikimo būdai būtini sudarinėjant testus.

1.2. Tyrimo objektas, sritis ir problema

Tyrimo objektas: Sistemos reikalavimais pagrįstas testų kūrimas skirtingose aplinkose realizuotoms sistemoms

Tyrimo sritis: Informacinių sistemų testavimo įrankiai, testų generavimo metodai.

Tyrimo problema: ilgas pradinių testų generavimas, reikalavimų ir testų atsekamumo sudėtingumas bei apibendrintos informacijos apie testų kūrimą skirtingose aplinkose trūkumas.

Numatomas sprendimas: reikalavimais pagrįstas testų generavimo metodas, kurio panaudojimas turi būti įvertintas skirtingose aplinkose kuriamoms sistemoms.

1.3. Testavimo proceso iš reikalavimų modelio analizė

Šiuolaikinės kuriamos darbastalio, internetinės, bei išmaniųjų telefonų informacinės sistemos yra pakankamai didelės, todėl norint užtikrinti teisingą jų darbą yra būtinas testavimas.

Informacinės sistemos testavimas atliekamas norint įvertinti informacinės sistemos atitikimą keliamiems reikalavimams:

- Funkciniams reikalavimams
- Nefunkciniams reikalavimams

Kaip teigiama Bartos, J. ir Walek, B. publikacijoje [1], pagrindiniai kriterijai kuriais informacinė sistema gali būti testuojama yra:

- Panaudojamumas: tikrinama, ar informacinė sistema veikia tinkamai ar visos funkcijos atlieka tai ką turėtų atlikti
- Vartotojų kontrolė: tikrinama, ar vartotojo sąsaja yra patogi, intuityvi ir paprasta ar sudėtinga ir sunkiai išmokstama
- Patikimumas: tikrinama, ar informacinė sistema elgiasi vienodai skirtingomis aplinkybėmis, t.y kai yra informacijos perkrovos, klaidos, ar sistema geba pati jas atpažinti ir pranešti apie įvykusias klaidas.
- Palaikomumas: tikrinama, ar informacinė sistema yra tinkamai įdiegta, tikrinamos problemos susijusios su techninės bei programinės įrangos suderinamumu bei konfigūracija

- Perkeliamumas: tikrinama, ar informacinė sistema geba dirbti skirtingose programinės bei techninės įrangos aplinkose

Dėl informacinių sistemų dydžio bei galimų žmogaus klaidų, rankinis testų sudarymas yra brangus ir nevisiškai patikimas. Todėl norint užtikrinti sklandų IS darbą vartotojams, sumažinti programavimo klaidų tikimybes bei rankinio testavimo kaštus yra naudojamos automatinio testavimo priemonės. Automatiniam testavimui atlikti yra naudojama atskira programinė įranga nuo kuriamos IS arba įskiepis į programavimo įrankį. Testuojant yra tikrinami realūs IS veiksmų rezultatai su planuotais rezultatais ir pagal tai vertinama ar informacinė sistema veikia tinkamai. Klaidos atsirandančios IS kyla dėl įvairių priežasčių.

Kaip teigiama Walid Kobrosly ir Stamatis Vassiliadis publikacijoje [2], pagrindinės klaidų priežastys atkleidžiamos testavimo metu yra dėl:

1. Klaidų atsiradusių programavimo metu
2. Specifikacijų aprašymo tikslumo ir vienareikšmiškumo nebuvimo
3. Reikalavimų nesupratimo

Pagrindinės testavimo metu aptiktos klaidos atsiranda programavimo metu. Šios klaidos gali būti efektyviai panaikintos dar ankstyvojoje IS realizavimo fazėje naudojantis reikalavimais pagrįstu testavimu kai testavimo atvejai yra sudaromi nenaudojant vidinio programos kodo. Šis metodas vadinamas Juodosios dėžės metodu (*angl. Black Box*) Tuo naudojantis, palengvinamas suprogramuoto kodo skaitymas, kai yra turimi reikalavimai ir aprašyti pagrindiniai atvejai su kokiais duomenimis tam tikra funkcija turėtų dirbti. Iš to matome, kad naudojantis aprašytais sistemai keliamais reikalavimais, dar prieš kuriant IS galima nusakyti būsimus tam tikrų funkcijų rezultatus ir taip palengvinti bei užtikrinti programavimo kokybę bei taip pat sumažinti IS kūrimo kaštus.

Kaip teigiama [3] publikacijoje pagrindinis reikalavimų modelio tikslas yra tiksliai užfiksuoti funkcinius reikalavimus, kuriuos būtų galima analizuoti, išgryninti bei vykdyti. Tik iš detaliai aprašyto reikalavimų modelio galima kurti testus būsimai IS bei nustatyti užbaigimo kriterijų (išeinamuosius IS duomenis)

Kaip teigiama Predrag Skoković, Marija Rakić-Skoković publikacijoje [4], reikalavimais pagrįsto testavimo procesas gali būti suskirstytas į šiuos procesus:

1. Nustatyti testo užbaigimo kriterijų – kai testas turi specifinius, kiekybinius reikalavimus kada jis laikomas užbaigtu.

2. Suprojektuoti testavimo atvejus – loginiai testavimo atvejai turi 4 charakteristikas:
   - Pradinė sistemos būsena
   - Duomenys
   - Įvedami duomenys
   - Tikėtini gauti rezultatai

3. Sukurti testavimo atvejus – susideda iš 2 dalių:
   - Sugeneruoti reikalingus duomenis
   - Sukurti reikalingus komponentus, kad galima būtų atlikti testavimą

4. Įvykdyti testus – įvykti testus su egzistuojančia informacine sistema ar jos dalimi ir dokumentuoti rezultatus

5. Patikrinti testavimo rezultatus – įvertinti ar tikėtini rezultatai yra toki patys kaip gautieji rezultatai, taip pat įvertinti ar atliktas testas atitinka testo atlikimo kriterijų.

6. Patikrinti testavimo padengimą – sekti kokį kiekį funkcionalumo užtikrina įvykdytas testas.

7. Nustatyti klaidas – visos testavimo metu nustatyti klaidos pateikiamos taisymui. Vedamos aptiktų klaidų statistikos

8. Valdyti testavimo biblioteką – testuotojas turi prižiūrėti, kad atlikti testai būtų atsekami su reikalavimų specifikacijose aprašytais reikalavimais.

Anot publikacijos [4] autorių, reikalavimais pagrįstas testavimas užtikrina pirmą, antrą bei šeštą aprašytą procesus, likę procesai yra valdomi testavimo valdymo įrankių.

Publikacijos [5] autoriai savo darbe teigia, kad norint generuoti testavimo atvejus, reikia atkreipti tinkamą dėmesį į reikalavimų modelio sudarymą bei pakankamą jo detalizavimą, taip, kad būtų įmanoma vykdyti algoritmą pateiktą pseudo kodu 1 pav.

```
Algorithm
1.  Modules[] ←
    extractModuleSpecification(RequirementAnalysis);
2.  for each module i  in modules[]
3.      {
4.      inputs[] ← extractInput(module_i);
5.      outputs[] ← extractOutput(module_i);
6.              {
7.              for each input in inputs[]
8.                      {
9.                      for each output in outputs[]
10.                             {
11.                             inputRange ←
    extractRange(input);
12.                             outputRange ←
    extractRange(output);
13.                             x ← random[] %
    inputRange;
14.                             y ← random[] %
    outputRange;
15.                             limit (x, y) in TCS;
16.                             }
17.                     }
18.             }
19.     }
 TCS- Test case suite
```

1 pav. Pseudo algoritmas testų generavimui [5]

1 lentelėje pateikiami algoritmo naudojamo testų generavimui iš reikalavimų specifikacijos funkcijų paaiškinimai

1 lentelė. Testų generavimo iš reikalavimų specifikacijos pseudo algoritmo parametrų paaiškinimai

Metodo pavadinimas	**Parametrai**	**Paaiškinimas**
extractModuleSpecification	RequirementAnalysis	Funkcija išgauna kiekvieno IS kuriamo modulio specifikaciją iš

Metodo pavadinimas	Parametrai	Paaiškinimas
		reikalavimų specifikacijos ir išsaugo į masyvą.
extractInput	$module_i$	Funkcija gauna visus pasirinktam moduliui priklausančius įeinančius duomenis ir išsaugo juos masyve.
extractOutput	$module_i$	Funkcija gauna visus pasirinktam moduliui priklausančius išeinančius duomenis ir išsaugo juos masyve.
extractRange	input, output	Funkcija gauna visų gautų įėjimų ir išėjimų apribojimus.
Limit	(x, y)	Funkcija išsaugo sugeneruotas generuojamo modulio įvedamų (x) ir išvedamų (y) duomenų ribas

2 lentelė. Testų generavimo iš reikalavimų specifikacijos algoritmo pavaizduoto 1 kintamųjų paaiškinimai

Kintamojo pavadinimas	Kintamojo tipas	Paaiškinimas
Modules	Array	Saugoma kiekviena atskira modulio specifikacija gauta iš reikalavimų specifikacijos
Inputs	Array	Saugomi visi įeinantys duomenys gauti iš pasirinkto modulio
Outputs	Array	Saugomi visi išeinantys duomenys gauti iš pasirinkto modulio
inputRange	Integer	Saugomos gautos įeinančių duomenų ribos.
outputRange	Integer	Saugomos gautos išeinančių duomenų ribos.
X	Integer	Saugomas sugeneruotas atsitiktinis skaičius iš įeinančių duomenų apribojimų.
Y	Integer	Saugomas sugeneruotas atsitiktinis skaičius iš išeinančių duomenų apribojimų.

R. Kavitha, V.R. Kavitha, Dr. N. Suresh Kumar publikacijoje [6] pažymi, kad norint padidinti programinės įrangos testavimo efektyvumą ir našumą būtina prioretizuoti kuriamus testavimo atvejus. Šiame darbe aptariamos testų 3 prioretizavimo rūšys:

- Vartotojo priskiriamas prioretizavimas
- Reikalavimų specifikacijos pokyčiai
- Realizacijos sudėtingumas

Autorių teigimu, tobulėjant informacinėms sistemoms vis sudėtingiau užtikrinti, kad kuriama programinė įranga veiktų tinkamai. Kaip pavyzdys, pateikiamas produktas turintis 20,000

kodo eilučių, kurio pilnam testavimui atlikti reikia 7 savaičių laiko. Tam, kad sumažinti testavimo sąnaudas yra siūloma naudoti testavimo atvejų prioretizavimo technikas, kurių esmė surikiuoti generuojamus testus pagal prioritetą, tam, kad būtų galima anksčiau identifikuoti klaidas ir sumažinti projekto sąnaudas. Kadangi testai yra rikiuojami, tačiau nešalinami, išvengiamos problemos, susijusios su testų minimizavimo technikomis t.y. nenustačius klaidų vykdant aukščiausio prioriteto testavimo atvejus, galima atlikti pilną sistemos testavimą su žemesnį prioriteto lygmenį turinčiais atvejais, todėl šios technikos taikymas nesukelia kritinių klaidų testavimo procese.

Publikacijoje [6] teigiama, jog susitelkiant į būsimų informacinės sistemos naudotojų keliamus reikalavimus pakyla jų IS vertės suvokimas ir pasitenkinimas, todėl vartotojų reikalavimai įvertinti aukščiausiais balais turėtų būti testuojami anksčiausiai.

Kitas aspektas pagal kurį skirstomi testavimo atvejai yra realizacijos sudėtingumas. Tai yra subjektyvus reikalavimo realizavimo sudėtingumo įvertinimas, kurį apibrėžia programuotojai, kuo didesnis skaičius priskiriamas reikalavimui, tuo reikalavimą yra sudėtingiau realizuoti. Šio prioretizavimo pagrindinė priežastis yra tai, kad kuo sudėtingesnis reikalavimo realizavimas, tuo daugiau pasitaiko programavimo klaidų.

Paskutinis aspektas pagal kurį publikacijos [6] autoriai siūlo rikiuoti testavimo atvejus yra reikalavimų pasikeitimai. Šis dydis yra nusakomas programuotojų, pagal tai, kiek kartų reikalavimo realizacija keitėsi nuo pradinės versijos. Jeigu reikalavimas kito daugiau kaip 10 kartų, jam priskiriamas aukščiausias 10 balų prioretizavimo rodiklis. IS reikalavimo pokytis RPi gali būti suskaičiuojamas padalinant konkretaus reikalavimo i pokytį kartais iš maksimalaus visame projekte esančio reikalavimo pokyčio. Jeigu i-tasis IS reikalavimas pasikeitė N kartų, o maksimalus reikalavimų pokytis kartais visoje reikalavimų specifikacijoje yra M, tada reikalavimo prioretizavimo rodiklis RPi gali būti apskaičiuojamas, kaip pateikiama (1) formulėje:

$$(1)\ RPi = N / M * 10$$

Šis rodiklis yra įvedamas dėl to, nes pasak autorių apie 50 procentų visų projektuose identifikuotų klaidų yra aptinkama reikalavimų fazėje, kurios kitimas tolimesnėse fazėse sąlygoja projekto nesėkmę.

Visoms aprašytoms prioretizavimo rūšims yra priskiriamos pradinės reikšmės nuo 1 iki 10 pagal prioritetą nuo žemiausio iki aukščiausio projekto reikalavimų fazėje bei kinta, bei kinta kartu su projektu visose IS realizavimo fazėse.

Turint „n“ reikalavimų su „j“ prioretizavimo reikšmėmis skaičiuojama kiekviena i reikalavimo faktoriaus reikšmė RFRi pagal (2) formulę:

$$(2)\ RFR_i = \sum_{j=1}^{3} \frac{faktoriaus\_reikšmė_j}{3}$$

Pateikta (2) formulė skaičiuoja reikalavimo i faktoriaus reikšmę, kuris parodo reikalavimo svarbą, kuo didesnis skaičius nuo 1 iki 10, tuo reikalavimo svarba yra didesnė.

Kaip teigiama Sunitha E.V, Philip Samuel [7] publikacijoje, veiklos diagramos yra vienos iš geriausių modelių kuriais galima perteikti verslo procesų elgseną, o panaudojus OCL (*angl. Object Constraint Language*) diagramoms galima suteikti daugiau informatyvumo ir išbaigtumo, panaudojus OCL galima detaliai aprašyti:

- Operacijas
- Pradines reikšmes
- Esamus parametrus
- Sąlygas
- Programavimo šabloną (*angl. Instance)*

OCL [8] yra formali specifikacijos kalba, UML plėtinys, turinti lengvai suprantamą sintaksę bei semantiką. Ši kalba naudojama norint suteikti modeliui tikslumo, vienareikšmiškumo. OCL yra naudojama suteikti išraišką modeliams, tačiau pati išraiška negali nieko keisti modelyje, kuriame ji yra panaudota, t.y. modelis, modelio būsena, turi išlikti nepakitus, net jei pati išraiška vaizduoja modelio būsenos kitimą, visos objektų, kintamųjų reikšmės esančios modelyje nekinta. Įvertinus modelyje esančią OCL išraišką yra pateikiamas rezultatas.

Ši modeliavimo kalba turi 4 pagrindinius apribojimų tipus:

- Invariantai *(angl. Invariants)*
- Sąlyga prieš *(angl. Pre-condition)*
- Sąlyga po *(angl. Post-condition)*
- Apsauga *(angl. Guard)*

Invariantai *(angl. Invariants)* – apribojimai, kurie suteikiami visoms klasėms, jie visada turi turėti teigiamą *(angl. true)* reikšmę. Paprasčiausias invarianto pavyzdys - klasės kintamojo apribojimas. Šis tipas taikomas per visas asociacijas turinčias klases, taip pat gali būti taikomas ir paprastosioms, asociacijų neturinčioms klasėms.

Sąlyga prieš *(angl. Pre-condition)* – sąlyga kuri turi turėti teigiamą *(angl. true)* reikšmę, operacijos aktyvavimo metu. Tipinis šios sąlygos veiksmas yra įsitikinti ar vienas objektas yra susietas su kitu, prieš juos susiejant tarpusavyje, taip pat jei ryšys tarp objektų yra, papildomas ryšys nekuriamas.

Sąlyga po *(angl. Post-condition)* – sąlyga, tikrinanti ar asociacija buvo sukurta, kai ryšys tarp objektų jau yra sukurtas.

Apsauga *(angl. Guard)* – sąlyga kuri turi būti teigiama *(angl. true)* prieš modelio būsenai kintant. Apsauga *(angl. Guard)* yra naudojama kai norima užtikrinti, kad modelio būsena nekis, kol nebus patenkinti visi sumodeliuoti apribojimai. Šis apribojimų rinkinys prieš kintant būsenai ir yra vadinamas Apsauga *(angl. Guard).*

Anot publikacijos [9] autorių, OCL panaudojimas UML modeliuose yra labai svarbus, kai iš jų norima generuoti testus, panaudojus OCL apribojimams aprašyti, iš jų galima generuoti įvedimo/išvedimo duomenis, atlikti veiksmus su susijusiais kintamaisiais, sugeneruoti reikiamus parametrus sumodeliuoto veiksmo pradėjimui.

Kaip teigiama Mohd. Ehmer Khan bei Farmeena Khan publikacijoje [10], norint sugeneruoti tinkamus testus, reikia pasirinkti tinkamą modelį tam atlikti, publikacijoje yra nagrinėjami 3 pagrindiniai galimi testų generavimo modeliai:

- Juodosios dėžės (*angl. black-box*)
- Pilkosios dėžės (*angl. gray-box)*
- Baltosios dėžės (*angl. white-box)*

Iš publikacijoje [10] pateiktų duomenų galima teigi, kad testai iš reikalavimų modelio gali būti generuojami tik juodosios bei pilkosios dėžės metodais, nes norint generuoti baltosios dėžės testus reikia žinoti vidinį programos realizavimo kodą.

Kaip teigiama publikacijoje [10], Juodosios dėžės (*angl. black-box)* testai užtikrina tik fundamentalius sistemos tikrinimo aspektus, o to pasekoje gaunamas neefektyvus sistemos testavimas, taip pat sudėtinga įvertinti ar gaunami išėjimo duomenys yra teisingi, nors taikant šį metodą yra reikalingas tik minimalus testuotojo suvokimas, tačiau pilkosios dėžės (*angl. gray-box)* testai turi, tiek juodosios dėžės, tiek baltosios dėžės testavimo privalumus. Šis metodas leidžia generuoti testus iš dalies žinant testuojamos sistemos veikimo procesus, todėl kuriami testai nusako

sistemos reakciją į įvedamus duomenis, pateikia, koks veiksmas ar jų sąrašas bus atliktas bei pateikia išvedimo duomenis, kurie leidžia testuotojui tikrinant sistemą lengvai identifikuoti ar sistemos reakcija į įvestus duomenis, ar atliktus veiksmus, yra teisinga.

Išnagrinėjus 1 pav. esantį algoritmą galima prieiti prie išvados, kad norint generuoti testus iš reikalavimų specifikacijos, reikia:

1. Išanalizuoti reikalavimų specifikaciją
2. Išanalizuoti detalizuotą reikalavimų specifikaciją
3. Sukurti kiekvienam moduliui tarpinius aprašymus
4. Kiekvienam moduliui surinkti įėjimo parametrus ir grąžinimo tipus
5. Gauti kiekvienam parametrui nustatytus apribojimus
6. Naudojantis gautais apribojimais, generuoti testavimo atvejus kiekvienam moduliui

Pritaikius aprašytą algoritmą, panaudojant OCL kalbos galimybes UML modeliuose esančiose diagramose, galima pritaikyti šiuos modelius testinių duomenų generavimui, o kadangi reikalavimų specifikacija susideda iš daug reikalavimų bei reikalavimų modelių, pritaikius prioretizavimo technikas generuojamiems testams iš reikalavimų, galima būtų greičiau aptikti esamas programavimo klaidas.

1.4. Tyrimo objekto naudotojų analizė

3 lentelė. Vartotojai dalyvaujantys testavimo procese

Vartotojo kategorija	Atliekamos funkcijos
Testuotojas	IS testavimas, testų sudarymas, testų ir reikalavimų atsekimas
Programuotojas	IS programavimas, testų rezultatų ir reikalavimų atsekimas

Vartotojų tikslai ir problemos

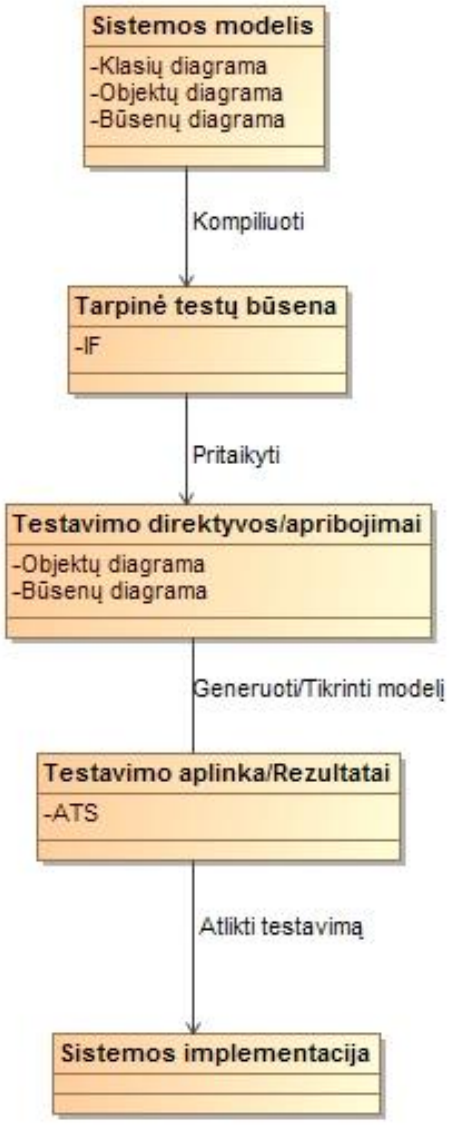
Testų generavimas bei atsekamumas yra labiausiai aktualus 2 tipams žmonių, t.y. testuotojams ir programuotojams. Testuotojų pagrindinis tikslas efektyviai ištestuoti kuriamą IS ir rasti esamas programavimo klaidas kuo ankstesnėje IS kūrimo fazėje, kuo anksčiau nustatoma klaida, tuo mažesni tampa IS kūrimo kaštai. Pagrindinės testuotojams kylančios problemos yra sudėtingas testų, testavimo scenarijų sudarymas, jų atsekamumas bei vėlyvas programavimo klaidų nustatymas, kas kelia testavimo, bei visos programinės įrangos kainą. Programuotojų pagrindinis tikslas sukurti IS ir užtikrinti teisingą jos veikimą, pagal paruoštus reikalavimus. Esminė kylanti problema yra ilgas pradinių testavimo scenarijų kūrimas bei, tai, kad atlikus testavimą ir gavus testavimo rezultatus ir radus klaidas, yra sudėtinga atsekti aprašytus testavimo atvejus su jų keliamais reikalavimais. Žinant testavimo metu užfiksuotą klaidą bei keliamus reikalavimus konkrečiai IS funkcijai, galima efektyviai ir greitai ištaisyti programavimo klaidas.

1.5. Esamų problemos sprendimo metodų analizė

Kadangi darbastalio, mobiliosios bei internetinės informacinės sistemos gali būti kuriamos skirtingomis programavimo kalbomis, norint teisingai jas ištestuoti reikia imti tai kas yra bendra visoms informacinėms sistemoms. Anot Alessandra Cavarra, Thierry JeronLaurent Mounier Jim Davies, Alan Hartman Sergey Olvovsky publikacijoje [11] teigiama, kad kiekvienos informacinės sistemos architektūra, sumodeliuota UML notacija, yra geras pagrindas automatiniam testų kūrimui. Šiam tikslui yra naudojami 3 tipai diagramų:

- Klasių diagrama (*angl. class diagram*)
- Būsenų diagrama (*angl. state machine diagram*)
- Objektų diagrama (*angl. object diagram*)

Kitas komponentas, reikalingas testų generavimui iš UML notacijos, yra testavimo direktyvos (*angl. test directives*), kurios yra sudaromos iš objektų diagramos bei būsenų diagramos. Objektų diagrama, išreiškia testavimo apribojimus ir aprėpties kriterijus, būsenų diagrama, nurodo testavimo tikslus. Šis architektūros principas pavaizduotas *1 pav.*



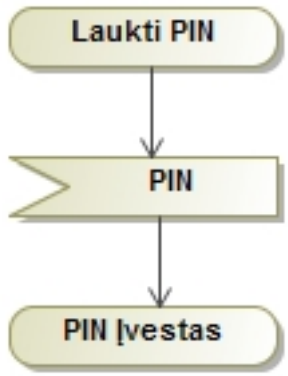
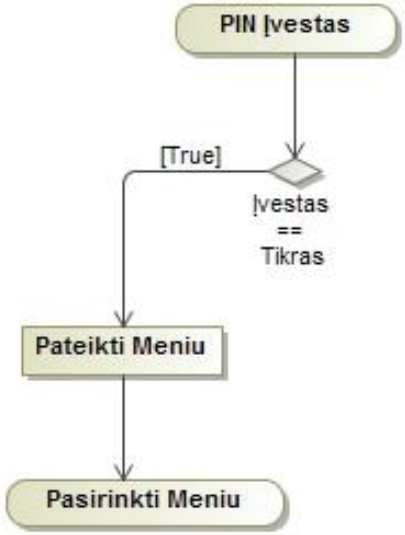
2 pav. „Testų generavimo architektūros principinė diagrama iš UML" [11]

1pav. Vaizduojamame sprendime testavimo direktyvos (*angl. test directives*) yra realizuojamos tarpinio formato (IF) kalba t.y. operacijos, veiksmai ir duomenų tipai yra apibūdinami paprastu IF sakinių rinkiniu, kurie yra suprantami visiems susipažinusiems su imperatyviomis programavimo kalbomis. Šis autorių sprendimas pasirinktas, todėl nes nežiūrint į tai kokioje programavimo kalboje yra realizuojama informacinė sistema, šį IF sakinių rinkinį galima paversti į atitinkamos programavimo kalbos testavimo rinkinį.

Anot Tahat, L.H. ; Vaysburg, B. ; Korel, B. ; Bader, A.J. publikacijoje [12] teigiama, kad norint užtikrinti tikslius juodosios dėžės (*angl. Black-box*) testus, kurie yra generuojami iš reikalavimų specifikacijos, pati reikalavimų specifikacija turi turėti vienareikšmišką aprašymo struktūrą. Tam tikslui autoriai siūlo naudoti SDL (*angl. Specification and Description Language*) Specifikacijos ir apibūdinimo kalbą. Ši specifikavimo kalba yra skirta sudėtingoms įvykiais pagrįstoms, realaus laiko ar interaktyvioms informacinių sistemų specifikacijos aprašyti.

SDL specifikacija yra pagrįsta būsenomis ir jų perėjimais. Kiekviena būsena turi rinkinį įėjimų. Tarp būsenų yra tarpiniai perėjimai, kurie gali keisti ar sukurti naujas būsenas, o patys perėjimai tai yra įvykių rinkiniai, kuriuos gali sudarytį išėjimo signalai (*angl. outputs*), užduotys (*angl. tasks*) ir sprendimai (*angl. decisions*).

Autorių požiūriu, reikalavimų specifikacija yra individualių informacinės sistemos reikalavimų rinkinys, kuris apibūdina dalinį funkcinį sistemos veikimą. Kiekvienam reikalavimui yra priskiriamas unikalus identifikavimo numeris, diagrama bei tekstinis reikalavimo apibūdinimas. Diagrama bei tekstinis reikalavimo apibūdinimas turi būti identiški, t.y. ne papildyti vienas kitą, o nusakyti konkretų reikalavimą tiek notacijos, tiek tekstiniu pavidalu. Toks dvigubas vaizdavimas leidžia programuotojams, testuotojams, vadovams ir kt. geriau suprasti aprašytą reikalavimą. Reikalavimų specifikacijos pavyzdys pateiktas 3 pav. Visi sistemai keliami nefunkciniai reikalavimai(greitaveika, saugumas ir kt.) aprašomi klasių diagramoje. Jeigu funkciniam reikalavimui yra keliami ir nefunkciniai reikalavimai, kuriuos jis turi išpildyti, tai nurodoma atskiroje klasių diagramose, susiejant individualų reikalavimą su reikalavimų klasėmis. Toks autorių siūlomas sprendimas, naudoti griežtą reikalavimų specifikacijos struktūrą yra būtinas, norint, kad kuriama testų generavimo aplinka teisingai sugeneruotų testavimo atvejus, nepriklausomai nuo to, kokiai aplinkai kuriama informacinė sistema yra skirta.

R1	**R2**	**R3**
Sistema turi priimti PIN	**Kai PIN priimtas, jeigu PIN sutampa su tikru PIN pateikti Meniu ir laukti pasirinkimo**	**Kai PIN priimtas, jeigu PIN nesutampa su tikru PIN pranešti apie klaidingą PIN ir laukti PIN įvedimo**
Klasė: Saugumas	**Klasė: Saugumas**	**Klasė: Saugumas**
		

3 pav. Reikalavimų specifikacijos pagal SDL pavyzdys [12]

Kitoje publikacijoje [13], siūlomas sprendimas panaudoti veiklos procesų diagramą kaip pagrindą testų generavimui iš UML modelių. Autorių teigimu, šioje diagramoje veiklos procesai atspindi procedūras turinčias būti realizuotas atliekamoje projekto realizacijoje, plaukimo takeliai (*angl. swim lanes)* vaizduoja realizuojančias klases, pati diagrama, turi signalų siuntėjus, gavėjus, sprendimų taškus (*angl. decitions),* lygiagretumo elementus (*angl. forks),* sujungimus *(angl. joins),* objektus. Naudojantis išvardintais elementais galima realizuoti veiklos procesus, kurie gali būti panaudoti testų generavimui. Autorių realizuotam algoritmui veikti reikia iš sumodeliuotos diagramos gauti:

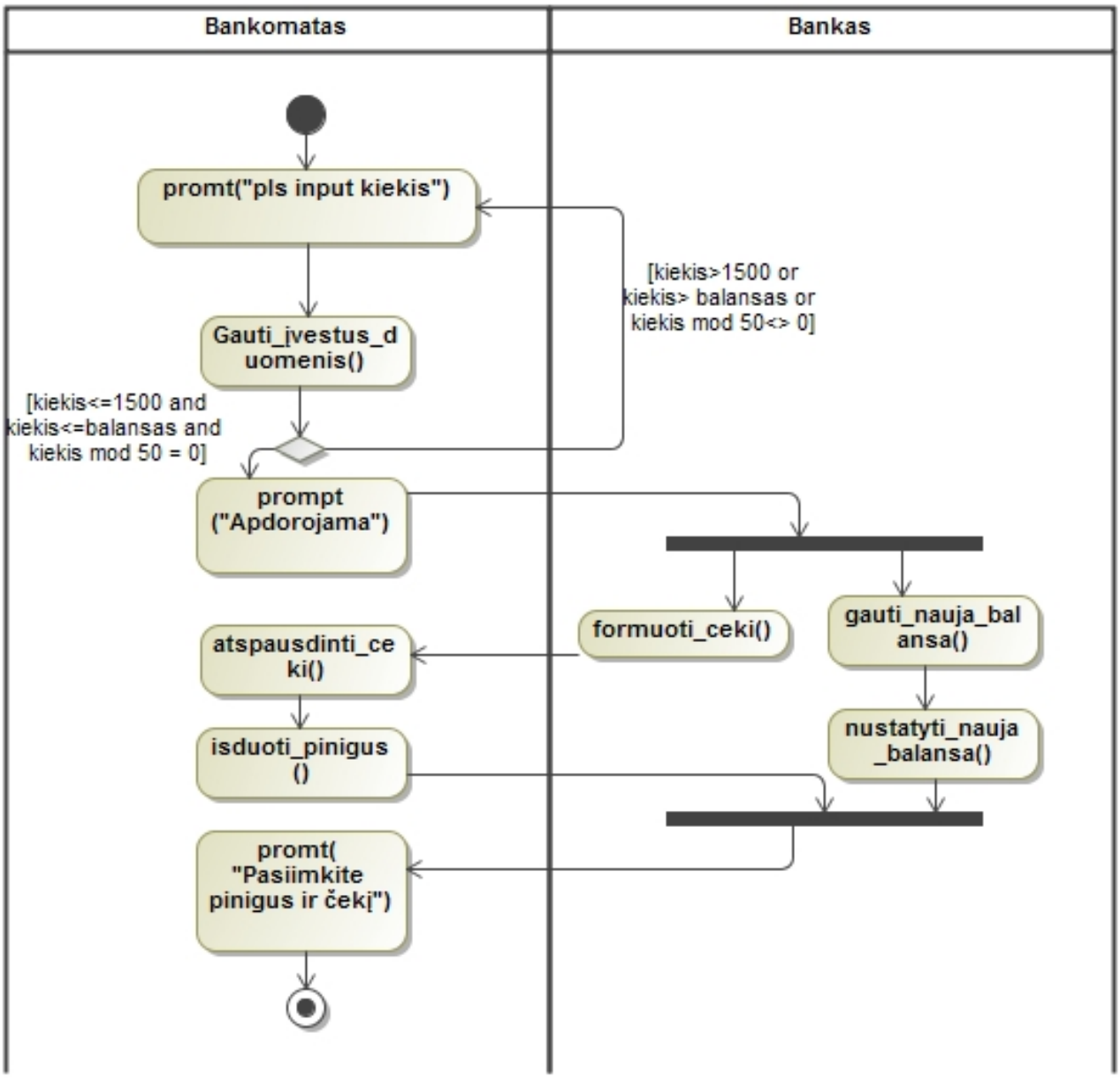
- A={a1,a2,…,am} veiklos procesų būsenų masyvą.
- T={t1,t2,…tn} perėjimų tarp būsenų masyvą.
- C={c1,c2,…,cn} apribojimų (*angl. guard)* masyvą, kur Ci turi atitinkamą ti perėjimų būsenų susietumą.

Iš aukščiau aprašytų duomenų galima generuoti juodosios dėžės (*angl. black-box)* testus, tačiau, kadangi šiais testais galima nustatyti tik informaciją ar funkcijos esančios veiklos procesų diagramoje yra realizuotos sistemoje autoriai siūlo papildyti sumodeliuotus procesus įėjimo ir išėjimo duomenimis. Įvedus šiuos elementus atsiranda galimybė realizuoti pilkosios dėžės (*angl. gray-box)* testus, dėl logikos bei kintamųjų įvedimo į veiklos procesų modelį. Įėjimo duomenų pvz: (prompt (“pls input amount:”)) kintamojo pavadinimas kuriuo manipuliuojama diagramoje baigiasi dvitaškiu („:“), išvedimo duomenų modeliavimo pvz: (prompt(“processing”)), norint tik išvesti duomenis naudojama funkcija žyminti duomenų išvedimą („prompt“), tarp kabučių (“ ”) esantis tekstas išvedamas vartotojui į ekraną. Šios publikacijos autorių aprašytas veiklos procesų diagramos pavyzdys pateiktas 4 pav

Šiame darbe [13], autoriai siūlo analizuoti modelį nuo jo pradžios iki pabaigos, naudojantis paieška į gylį pažymint, kad kiekviena šaka bei sumodeliuoti ciklai yra pereinami tik vieną kartą, kad kiekviena būsena bei perėjimai tarp jų yra praeiti, taip gaunami elementarieji keliai (*angl. basic paths***)**

Gauti elementarieji keliai tenkina pilkosios dėžės (*angl. gray-box*) testavimo metodo keliamus minimalius reikalavimus bei gali būti taikomi testų generavimui. Publikacijoje [13] aprašytas algoritmas atlieką paiešką į gylį pasirinktame veiklos procesų modelyje ir sugeneruoja galimus testavimo scenarijus pagal diagramoje atitinkančius rastus elementariuosius kelius. Tada iš veiksmų sekos kiekvienam iš testavimo scenarijų yra sugeneruojami įėjimo ir išėjimo duomenų modeliai. Apribojimų duomenys imami iš apsaugos (*angl. guard)* laukų. Tuomet, naudojantis (*angl. category partition)* metodu generuojami įėjimo bei išėjimo duomenys. Iš gautų duomenų generuojami testavimo atvejai susidedantys iš:

- Tikslių įvedimo duomenų
- Tikėtinos sistemos veiksmų sekos
- Sistemos reakcijos (išvedimo duomenų)



4 pav. Veiklos procesų diagrama „Išsigryninti pinigus“ [13]

4 lentelėje pateikiama išanalizuotų metodų leidžiančių generuoti testavimo scenarijus iš įvairių modelių palyginimas.

4 lentelė. Išanalizuotų testų generavimo metodų palyginimas

Autoriai	**Metodo santrauka**	**Naudojami duomenys testų generavimui**	**Privalumai**	**Trūkumai**
Alessandra Cavarra Jim Davies Thierry Jeron Laurent Mounier Alan Hartman Sergey	Diagramos transformuojamos į tarpinę IF būsenų kalbą Metodas remiasi objektų būsenų kitimais.	Būsenų diagrama Objektų diagrama Klasių diagrama	Generuojami įvedimo / išvedimo duomenys	Reikalauja didelės IS modeliuotojo kompetencijos.

Autoriai	Metodo santrauka	Naudojami duomenys testų generavimui	Privalumai	Trūkumai
Olvovsky				
Tahat L.H. Vaysburg B. Korel B. Bader A.J.	ITU (International Telecommunication Union) standartas SDL (Specification and Description Language) konceptas	Modifikuotos veiklos diagramos pagal SDL konceptą	SDL konceptas suteikia daug informatyvumo testuotojui apie testuojamą procesą, dėl griežtos specifikacijos	Automatiškai generuojami tik įeinantys duomenys, sunku įvertinti testavimo rezultatą.
Wang Linzhang Yuan Jiesong Yu Xiaofeng Hu Jun, Li Xuandong Zheng Guoliang	Generuojami testavimo atvejai iš procedūriškai sumodeliuotų veiklos diagramų.	Veiklos diagramos	Generuojami įvedimo/išvedimo duomenys.	Reikalingas papildomų veiklos diagramų braižymas, nėra galimybės naudoti reikalavimų stadijos veiklos diagramų.

Iš pateiktų duomenų 4 lentelėje galima spręsti, kad metodo, leidžiančio generuoti testavimo scenarijus su įėjimo ir išėjimo duomenimis iš pradinio reikalavimų modelio diagramų skirtų vartotojams nerasta.

1.6. Panašių sprendimų analizė

Šiandieną yra sukurta nemažai įrankių skirtų informacinių sistemų automatiniam testavimui. Tačiau, ne visi įrankiai turi galimybę testuoti skirtingų tipų IS skirtas mobiliesiems telefonams, taip pat intenetines bei darbastalio IS. Buvo pasirinkti 2 komerciniai įrankiai gebantys testuoti mobiliąsias aplikacijas, internetines bei darbastalio IS.

Pasirinkti įrankiai analizei:

- SmartBear TestComplete [14]
- Telerik [15]

Šių abiejų, analizuojamų sistemų paskirtis palengvinti testavimo procesus žmonėms, kuriantiems internetinių, darbastalio bei mobiliųjų telefonų informacines sistemas. Vienas iš pagrindinių kriterijų norint užtikrinti teisingą informacinės sistemos darbą yra funkcinis testavimas. Tam, kad testavimas būtų efektyvus jis turi turėti teisingai aprašytus testavimo atvejus.

Publikacijos autorių Mustafa, K.M. , Al-Qutaish, R.E. , Muhairat, M.I. publikacijoje [16] teigiama, kad idealaus testavimo atveju kuriamas testavimo atvejis turėtų atitikti šiuos kriterijus:

- Trumpumas – Testas turi būti kuo įmanoma paprastesnis ir aiškesnis
- Savikontrolė – Testas turi pranešti rezultatus, neturi būti reikalingas žmogaus įsikišimas
- Pasikartojamumas – Testas turi būti lengvai pakartojamas be žmogaus įsikišimo.
- Patikimumas – Testas turi pateikti tuos pačius rezultatus. Testo negali įtakoti išorinė aplinka.
- Pakankamumas – Testas turi užtikrinti visus testuojamos sistemos keliamus reikalavimus.

- Būtinumas – Testas turi atlikti tik tai kas būtina užtikrinti teisingą sistemos darbą
- Aiškumas – Kiekvienas testo atvejis turi būti aiškus ir lengvai suprantamas
- Efektyvumas – Testas turi vykti atitinkamą laiko tarpą
- Konkretumas – Kiekviena testavimo klaida turi pateikti nuorodas į klaidingo kodo vietą
- Nepriklausomumas – Kiekvienas testas turi veikti nepriklausomai nuo kitų testų ir tuo galima sudaryti testų grandinę
- Modifikuojamumas – Testas turi būti lengvai praplečiamas ir modifikuojamas
- Atsekamumas – Testas turi būti atsekamas pagal reikalavimus. Reikalavimai turi būti atsekami pagal testus.

1.7. Įrankis SmartBear TestComplete

Anot programos [14] autorių, pagrindiniai kuriami testavimų atvejai šioje programoje yra scenarijumi bei raktažodžiais paremtas testavimas.

Raktažodžiais paremtas testavimas priskiria programos veiksmą pasirinktam raktažodžiui kurį atliks testas. Programa atpažįsta testuojamos sistemos elementus pagal jų pavadinimus, internetinėse sistemose pagal unikalų pavadinimą ID. Veiksmu laikomi įvykiai kaip pelės paspaudimas, pasirinkimas iš meniu punkto, langų atidarymas ar uždarymas, klaviatūros klavišų paspaudimas ir tt. Raktažodžiui priskirtas veiksmas vadinamas operacija, kuriai programoje galima nurodyti parametrus kurie nusako kokie veiksmai bus atliekami šiuo testu. Yra galimybė vienu testu, paleisti kitą raktažodžio testą, priskirti programos veiksmą(*angl. event*) operacijai. Galima nustatyti įvedamus parametrus, bei planuojamus rezultatus: teksto(*angl. string*), skaičiaus (*angl. integer*), teigiamos ar neigiamos reikšmės kintamojo (*angl. boolean)*, programuojamo kodo eilučių tipo, objektų ir tt. Kuriamos testų operacijos gali veikti nepriklausomai arba gali būti grupuojamos į testų grupes. Kiekviena sukurta testavimo operacija yra lengvai modifikuojama. Atlikus testus programa sugeneruoja ataskaitą apie testus kurie netenkina rezultatų bei nurodo galimas programavimo kodo vietas kuriose yra potencialios klaidos. Tada kai testavimo programoje numatytų testavimo atvejų trūksta galima konvertuoti iš raktažodžio arba susikurti naują scenarijumi pagrįstą testavimo atvejį. T.y. programuoti testavimo atvejus rankomis naudojant VBScript, Jscript, DelphiScript,C++Script, C#Script skriptų kalbas.

1.8. Įrankis Telerik

Kaip pateikiama programos [15] autorių, šiame įrankyje testavimo atvejai yra atvaizduojami piramidės principu, bendras testavimas (IS), funkcijų testavimas (funkcijos), vartotojo sąsaja (vartotojo veiksmai). Naudojamas automatinis testų generavimo įrašymo įrankis. Naudojantis testuojama IS kiekvienas žingsnis bei panaudotas elementas yra įrašinėjamas į lentelę, kurioje galima nustatyti atskirą testavimo atvejį tiek elementui, tiek atliktam įvykiui. Vieno kuriamo testo įrašinėjamas baigiamas uždarius naudojamą informacinę sistemą. Kiekvienas testavimo atvejis yra nepriklausomas, taip pat galima grupuoti testus sudarant jų grandinę. Galima nustatyti planuojamus testavimo atvejo rezultatus, simuliuoti automatinį teksto įvedimą laukams. Galima kurti testus, tiek funkciniams reikalavimams užtikrinti, tiek atlikti trukmės testus kiekvienai norimai funkcijai ar jų rinkiniui. Testavimo klaidos pateikiamos žingsnių bei elementų atvaizdavimo lentelėje. Yra galimybė lengvai keisti testavimo atvejo duomenis. Dėl to, kad kiekvienas testavimo atvejis pateikiamas pažingsniui lentelėje, yra lengva suprasti testavimo atvejo principus, norint juos

modifikuoti. Įrašyti testavimo atvejai gali būti kartojami. Taip pat galima kurti rankinius testus C# ir VB.NET skriptų kalbomis.

Išanalizuotų testavimo įrankių, automatinių testavimo atvejų, kūrimo palyginimas pateikiamas 24 lentelėje.

5 lentelė. Automatinių testų kūrimo įrankių palyginimas

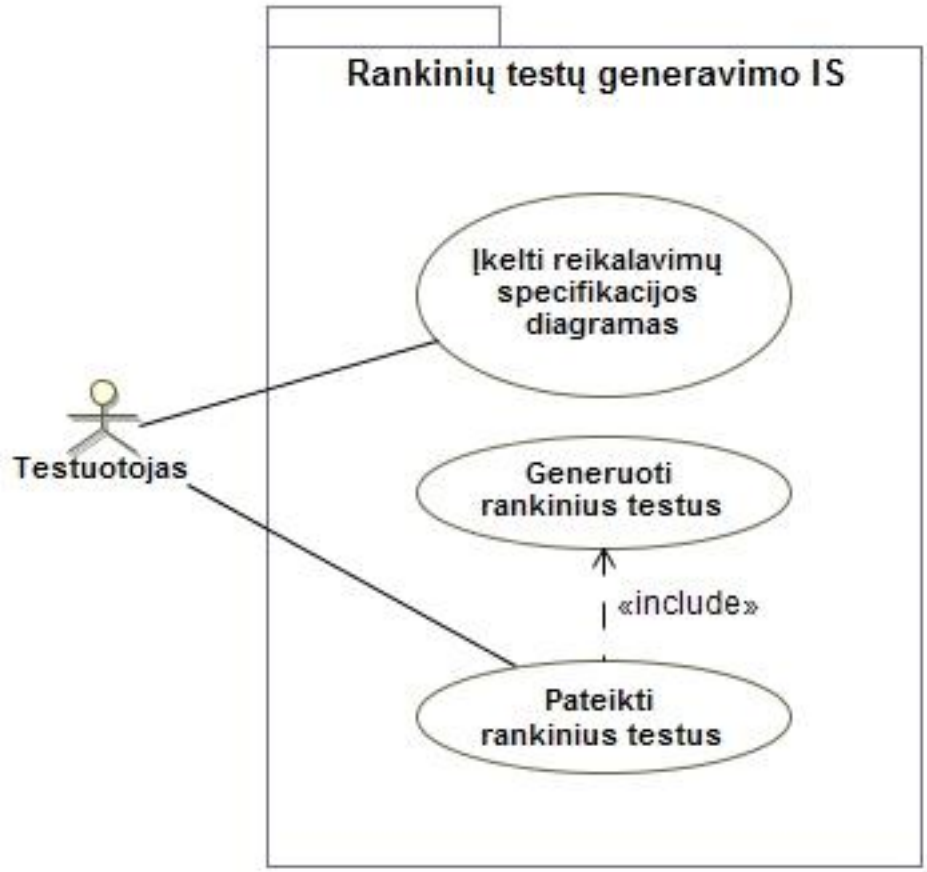
Įrankis / Kriterijus	**SmartBear TestComplete**		**Telerik**	
Trumpumas	Galimi trumpi testai	+	Galimi trumpi testai	+
Savikontrolė	Aiškiai pranešami testavimo rezultatai ir klaidos	+	Aiškiai pranešami testavimo rezultatai ir klaidos	+
Pasikartojamumas	Gali būti kartojami testai	+	Gali būti kartojami testai	+
Patikimumas	Testai pateikia tuos pačius rezultatus, juos kartojant	+	Testai pateikia tuos pačius rezultatus, juos kartojant	+
Pakankamumas	Galima realizuoti visus norimus testavimo atvejus	+	Galima realizuoti visus norimus testavimo atvejus	+
Būtinumas	Aiškiai nustatomi testavimo atvejai	+	Aiškiai nustatomi testavimo atvejai	+
Aiškumas	Aiškus testo vaizdavimas	+	Sudėtingesnis testo vaizdavimo būdas	+/-
Efektyvumas	Priklausomai, nuo testavimo atvejo		Priklausomai, nuo testavimo atvejo	+
Konkretumas	Nuorodos į testavimo aptiktas klaidas pateikiamos	+	Nuorodos į testavimo aptiktas klaidas pateikiamos	+
Nepriklausomumas	Kiekvienas testas yra individualus	+	Kiekvienas testas yra individualus	+
Modifikuojamumas	Greitai keičiami testavimo atvejai	+	Greitai keičiami testavimo atvejai	+
Atsekamumas	Sudėtingiau atsekti testus pagal reikalavimus	+/-	Lengviau atsekamas testas pagal reikalavimus	+

+ - automatinio testo kriterijus tenkinamas
+/- - automatinio testo kriterijus tenkinamas iš dalies

Buvo išanalizuoti Telerik bei SmartBear automatinio testavimo įrankiai. Kiekvienas įrankis efektyviai atlieka darbą testuojant informacines sistemas ir pakankamai gerai užtikrina kriterijus keliamus idealiam testavimo atvejui. Tačiau testų atsekamumas pagal reikalavimus yra geriau realizuotas įrankyje Telerik, dėl detalaus testavimo atvejų žingsnių vaizdavimo. Tačiau, nei vienas iš analizuotų įrankių nesuteikia galimybės generuoti testavimo scenarijų iš reikalavimų modelio pateikto UML notacija.

1.9. Siekiamo sprendimo apibrėžimas

Šiuo darbu siekiama sukurti informacinę sistemą, kuri generuotų testavimo atvejus iš pateiktų UML reikalavimų specifikacijos veiklos diagramų, atsižvelgiant į skirtingose aplinkose kuriamas informacines sistemas bei suteiktų galimybę užtikrinti reikalavimų specifikacijos bei testų atsekamumą.



5 pav. Rankinių testų generavimo IS pradinė panaudojimo atvejų diagrama

Siekiamame sprendime, pradinė panaudojimo atvejų diagrama pateikta 5 pav., kuriama IS priima MagicDraw UML 18.0 (*.mdzip) projektų failus iš kurių bus generuojami rankiniai testai, tai realizuoja *„Įkelti reikalavimų specifikacijos diagramas“* panaudojimo atvejis.

Iš nurodytų MagicDraw UML projekto modelių generuojami rankiniai testavimo atvejai, realizuoja *„Generuoti rankinius testus“* panaudojimo atvejis.

Sugeneruoti testai pateikiami vartotojui, tai realizuoja *„Pateikti rankinius testus“* panaudojimo atvejis. Vartotojui pateikiama informacija apie įkeltam projektui sugeneruotus testavimo atvejus, jų unikalų eilės numerį, taip pat veiklos diagramą kuriai priklauso konkretus testas, taip užtikrinamas reikalavimų atsekimas su testais.

Funkciniai reikalavimai:

- Sistema turi generuoti rankinius testavimo atvejus, nepriklausomai nuo to kokiai platformai yra suprojektuota sistema
- Sistema turi pateikti informaciją siejančią sugeneruotus testavimo atvejus su reikalavimais

Nefunkciniai reikalavimai:

- Sistema turi būti suderinama su MagicDraw UML RUP sumodeliuotų sistemų pateiktų projektų (*.mdzip) failais

1.10. Analizės išvados

Išanalizavus tyrimo objektą bei padarius panašių sprendimų analizę paaiškėjo, kad:

1. UML veiklos diagramas galima naudoti norint atvaizduoti kuriamos IS reikalavimus visiems kuriamų IS tipams (darbastalio, internetinės, mobiliosios aplikacijos)
2. Pritaikius OCL veiklos diagramoms jas galima detalizuoti išlaikant pradinį sumodeliuotą variantą iš vartotojo pusės.
3. Remiantis veiklos diagramomis su aprašytais OCL parametrais galima generuoti testavimo scenarijus.
4. Naudojant veiklos diagramas reikalavimams aprašyti ir iš jų generuojant testavimo scenarijus galima užtikrinti testų su reikalavimais atsekamumą.
5. Pritaikant prioretizavimo technikas generuojamiems testavimo scenarijams galima anksčiau nustatyti galimas programavimo klaidas.
6. Naudojant testavimo įrankius generuojančius automatinius testus skirtus skirtingose aplinkose realizuotoms sistemoms, testus reikia vykdyti toje pačioje testavimo aplinkoje.
7. Metodo, leidžiančio generuoti testavimo scenarijus su įėjimo ir išėjimo duomenimis iš reikalavimų modelio diagramų skirtų vartotojams nerasta, todėl nuspręsta kurti įrankį leidžiantį tai atlikti.

2. VEIKLOS MODELIU GRINDŽIAMAS PILKOSIOS DĖŽĖS TESTŲ GENERAVIMO REIKALAVIMŲ SPECIFIKACIJA IR PROJEKTAS, FORMALUS APRAŠAS

2.1. Reikalavimų specifikacija

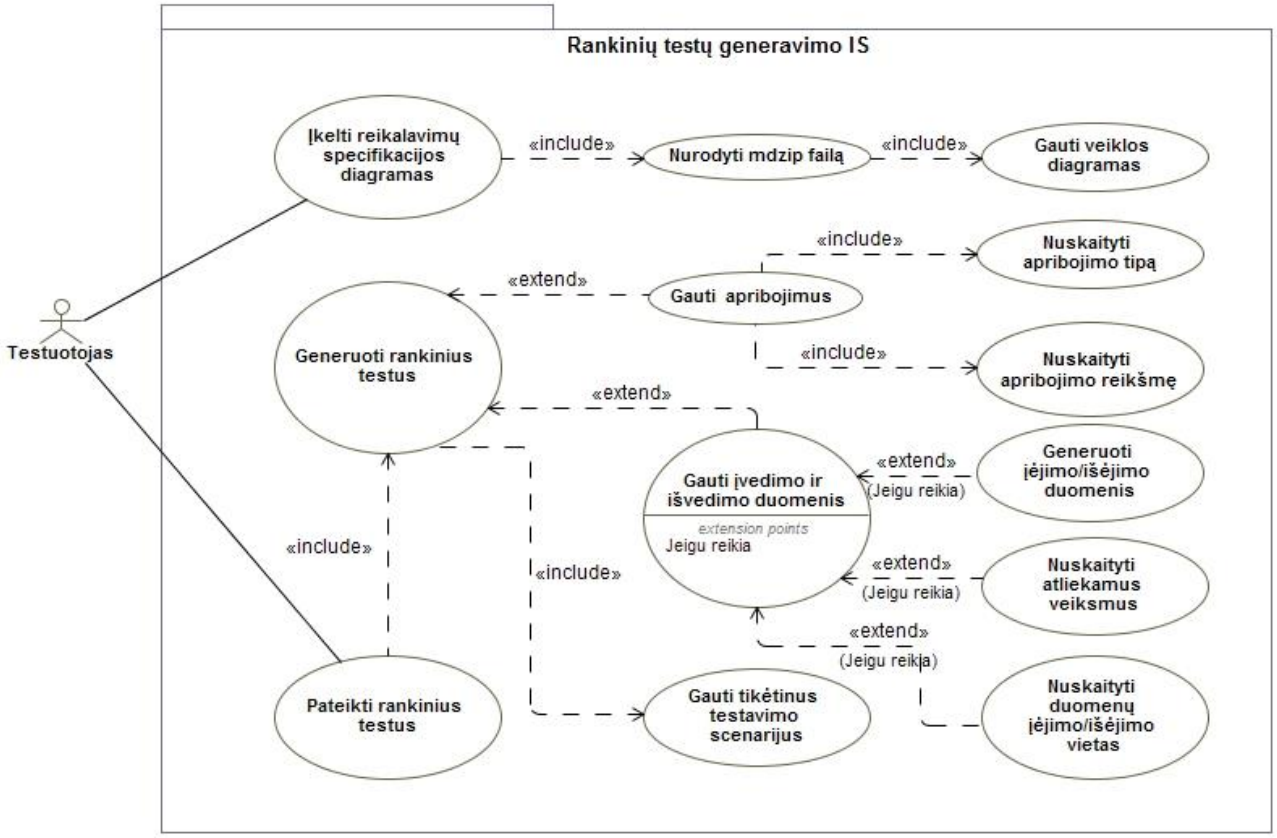
Šiame skyriuje pateikiama veiklos modeliu grindžiamos testų generavimo sistemos reikalavimų specifikacija. 6 pav. vaizduojama siūlomos testų generavimo IS detalizuota panaudojimo atvejų diagrama. Pagrindinis informacijos šaltinis iš kurio generuojami rankiniai testai yra veiklos procesų diagramos sumodeliuotos MagicDraw UML 18.0 ar naujesne versija. Išsaugoti MagicDraw *.mdzip failai tiesiogiai įkeliami į IS generuojančią rankinius testus, šį funkcionalumą realizuoja panaudojimo atvejis (PA) „Įkelti reikalavimų specifikacijos diagramas“. Po reikalavimų specifikacijos įkėlimo IS nuskaito veiklos procesų diagramas, gauna veikloms ar perėjimams tarp jų taikomus apribojimus, tai realizuoja PA „Gauti apribojimus“. Šį panaudojimo atvejį realizuoja:

- PA „Nuskaityti apribojimo tipą“, kuris nustato nuskaitomo apribojimo tipą (Skaitinį ar tekstinį)
- PA „Nuskaityti apribojimo reikšmę“, kuris nuskaito apribojimui sumodeliuotą reikšmę, jeigu tokia yra.

Kitame etape nuskaitomi sumodeliuoti įeinamieji bei išeinamieji duomenys, tai realizuoja PA „Gauti įvedimo ir išvedimo duomenis“. Šį PA realizuoja:

- PA „Nuskaityti atliekamus veiksmus“ kuris nuskaito sumodeliuotus veiksmus su kintamaisiais.
- PA „Nuskaityti duomenų įėjimo/išėjimo vietas“, kuris nustato duomenų sumodeliuotas duomenų įėjimo ir išėjimo vietas.
- PA „Generuoti įėjimo/išėjimo duomenis“, kuris generuoja įėjimo ir išėjimo duomenis pagal anksčiau nuskaitytus apribojimus ir veiksmus su kintamaisiais.

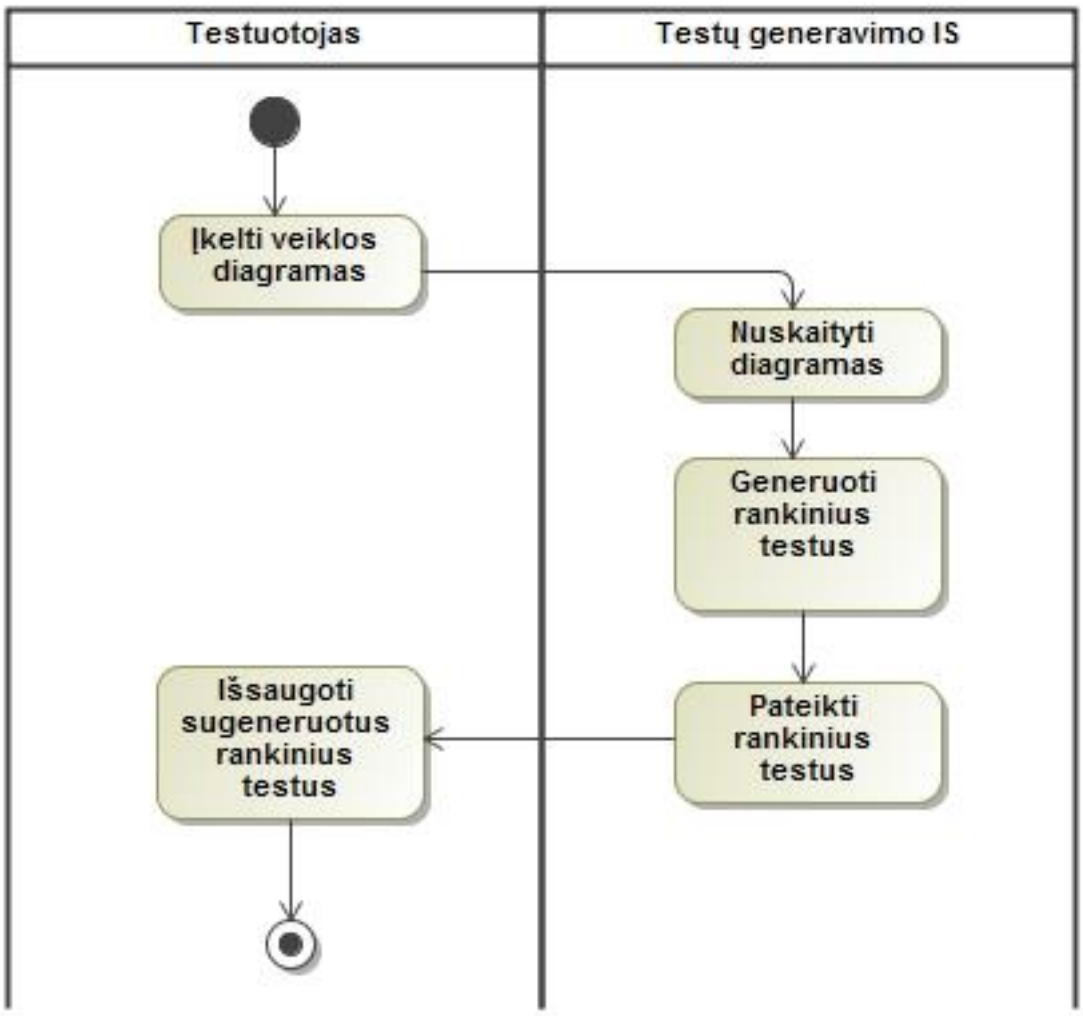
Testavimo scenarijų generavimą realizuoja PA „Gauti tikėtinus testavimo scenarijus“. Tam, kad būtų galimybė užtikrinti reikalavimų atsekamumą su testais bei testų atsekamumą su reikalavimais IS pateikia sugeneruotus testus bei jų sąryšius su veiklos diagramomis, tai realizuoja PA „Pateikti rankinius testus“.



6 pav. Rankinių testų generavimo IS panaudojimo atvejų diagrama

2.2. Formalus sprendimo aprašas

7 pav. Pateikiamas rankinių testų generavimo veiklos procesų modelis, kuriame matyti, kad procesas prasideda nuo Testuotojo kuris įkelia veiklos diagramas į IS, toliau testų generavimo IS nuskaito pateiktas veiklos procesų diagramas. Iš gautų duomenų generuoja rankinius testus. Rankinių testų generavimo įrankis yra paremtas sumodeliuotomis veiklos procesų diagramomis bei jose pateiktais įeinančiais, išeinančiais duomenimis bei jiems priskirtais apribojimais, panaudojant OCL jų aprašymui. Pateikus veiklos procesų diagramas sumodeliuotas pagal 3 skyriuje pateiktus reikalavimus, IS generuoja rankinius testus bei pateikia juos vartotojui.



7 pav. Rankinių testų generavimo IS veiklos proceso modelis

Šio sprendimo esmė yra panaudoti veiklos procesų modelius, kuriamus reikalavimų stadijoje, kuriuos detalizavus, panaudojant OCL, būtų galimybė generuoti rankinius testus pagal aprašytus apribojimus bei įėjimo ir išėjimo duomenis. Toks sprendimas leistų sumažinti bendrą reikalavimų modelio kompleksiškumą, nes nereikėtų kurti papildomų diagramų detalizuojančių veiklos modelius, taip pat nereikėtų modeliuoti ryšių tarp diagramų, ar naujų procesų diagramų testams generuoti.

3. VEIKLOS MODELIU GRINDŽIAMAS PILKOSIOS DĖŽĖS TESTŲ GENERAVIMO ALGORITMO APRAŠAS

Šiame skyriuje aprašomas veiklos modeliu grindžiamo rankinių testų generavimo algoritmas bei veiklos diagramų ryšys su testavimo scenarijais.

3.1. Veiklos diagramos ir jų ryšys su testavimo scenarijais

Veiklos diagramos vaizduoja sistemos elgseną, sekų modelį, vykstantį su jam priskirtais apribojimais, išsišakojimais, priskirtomis sąlygomis. Pagrindinis veiklos modelio tikslas - aiškiai perteikti veiksmų, procesų eiliškumą tam tikromis sąlygomis. Kiekvienas veiklos modelis turi turėti pradžios bei pabaigos taškus ir bent vieną veiklą tarp jų.

Pagrindiniai šioje diagramoje naudojami elementai:

- **Plaukimo takeliai** (*angl. Swimlanes)* naudojami apibrėžti abstraktesnes elementų, esančių juose, priklausomybes. Tai gali būti žmogus, IS, modulis ir kt. bet kokia sumodeliuotą veiklą subendrinanti esybė. Veiklos, esančios plaukimo takelio ribose, turi priklausomybę šiam plaukimo takeliui.
- **Veiklos elementai** (*angl. Action*) naudojami identifikuoti, aprašyti vykstantį veiksmą.
- **Sprendimų taškai** (*angl. Decition)* naudojami norint sumodeliuoti sąlygą veiksmų sekoje.
- **Suliejimo taškai** (*angl. Merge)* naudojami norint pavaizduoti kelių galimų sekų suliejimo vietą.
- **Lygiagretumo elementai** (*angl. Fork)* naudojami norint pavaizduoti keletą lygiagrečiai vykstančių procesų.
- **Suliejimo elementai** (*angl. Join)* naudojami vaizduojant procesų, kurie buvo išskaidyti, suliejimą į vieną procesą. Pastaba, jog sumodeliuotas veiklos procesas po šio elemento neprasidės tol, kol nebus gauti visi įėjimo taškai į šį elementą, t.y. visos lygiagrečiai vykstančios veiklos nebus pasibaigę.
- **Įėjimo taškai** (*angl. Input Pin)* skirti pavaizduoti įeinančius duomenis į veiklos elementą. Pastaba, veiklos elementas nepradedamas vykdyti, kol visi įėjimo taškai, šiam elementui, neturi duomenų.
- **Išėjimo taškai** (*angl. Output Pin)* vaizduoja išeinančius duomenis iš veiklos.
- Vaizduojant veiklos perėjimą iš vienos į kitą elementai sujungiami **veiksmo sekos** (*angl. Control Flow*) ryšiu.

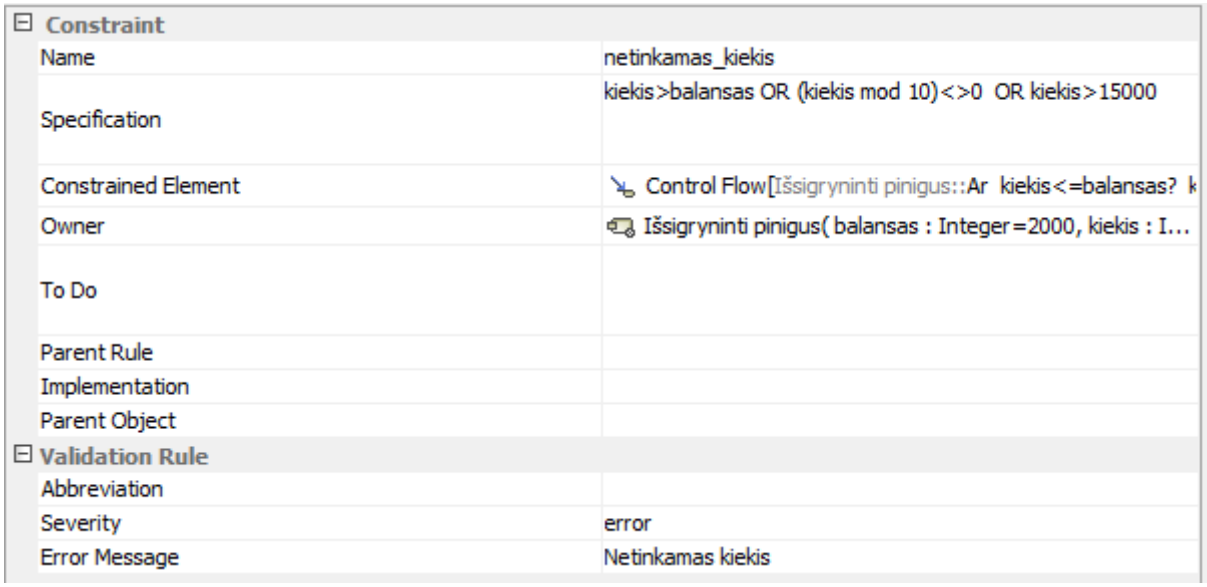
Veiklos modeliai perteikia kuriamos IS veiklos procesus, todėl radus visus tokioje diagramoje sumodeliuotus kelius, galima aiškiai identifikuoti veiksmų sekas vaizduojančias IS funkcionalumą. Jos leidžia žinoti buvusius, esamus bei būsimus IS veiksmus prie veiklos diagramoje sumodeliuotų sąlygų. Šių veiksmų sekų žinojimas suteikia galimybę testuoti kuriamą IS, tikrinant, ar kiekvienas iš sumodeliuotų veiksmų yra įvykdomas atliktoje realizacijoje.

Vaizduojamos IS apribojimai (*angl. Constraint)* turi būti sumodeliuoti pagal OCL 2.4 pritaikymo taisykles MagicDraw 18.0 ir vėlesnėse versijose.

Kiekvienas sukuriamas apribojimas turi:

- Pavadinimą
- Specifikaciją (apribojimo aprašymą)
- Apribojamą elementą
- Savinininką (veiklos diagramą kuriai priklauso apribojimas)

MagicDraw apribojimo modeliavimo pavyzdys pateikiamas 8 pav.



8 pav. Magic Draw apribojimo modeliavimo pavyzdys

Norint sumodeliuoti klaidos pranešimą apribojimui kaip pateikta 8 pav. „Valdation Rule“ nustatomas klaidos (*angl. error)* tipas, bei žinutės vietoje (*angl. Error Message)* įvedas norimas klaidos pranešimas.

Modeliuojant duomenų įvedimą prie veiklos uždedamas Įėjimo taško (*angl. Input Pin)* simbolis, vardo lauke (*angl. Name)* nurodant kintamojo pavadinimą kuris yra įvedamas į vaizduojamą veiklą. Duomenų įvedimo modeliavimo pavyzdys pateikiamas 9 pav.

9 pav. Modeliuojamo duomenų įvedimo pavyzdys

Vaizduojant duomenų išvedimą prie veiklos simbolio pridedamas Išėjimo taško (*angl. Output Pin)* simbolis, vardo lauke nurodant kintamojo pavadinimą, kuris norima, kad būtų išvedamas iš IS. Duomenų išvedimo modeliavimo pavyzdys pateikiamas 10 pav.

10 pav. Modeliuojamo duomenų išvedimo pavyzdys

Veiklos diagramoje modeliuojant veiksmus su aprašytais kintamaisiais veiklos Atlikti (*angl. To Do)* skiltyje aprašomos atliekamos operacijos:

- Sudėties „ + “
- Atimties „ – „
- Dalybos „ / “
- Daugybos „ * “
- String tipo operacijų pridėjimas „ . “

Sumodeliuotose veiklos diagramose Įėjimo taškai parodo, kuriose vietose bus prašoma įvesti duomenis į IS. Išėjimo taškai pateikia informaciją kuriose veiklos vietose bus išvesti pranešimai ar kintamieji vartotojui.

3.2. Algoritmo aprašymas

Pilkosios dėžės rankiniams testams generuoti naudojamos veiklos diagramos. Algoritmo pradžioje iš analizuojamos veiklos diagramos gaunami visi aprašyti kintamieji ir jų reikšmės, gaunamas apribojimų sąrašas, duomenų įėjimo ir išėjimo taškai.

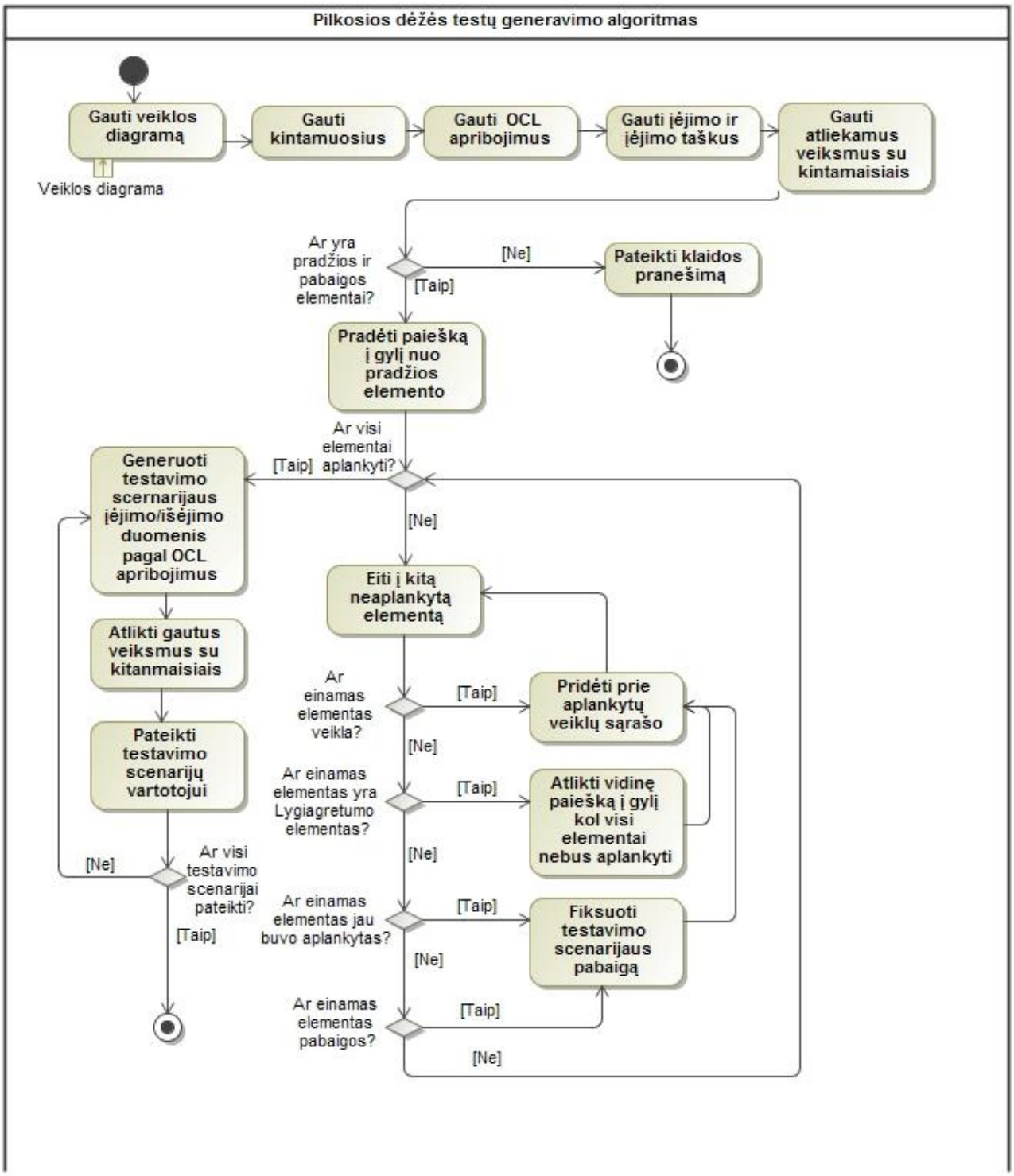
Visų galimų kelių radimui veiklos diagramoje naudojama paieška į gylį, pradedant nuo veiklos modelyje esančio pradžios taško. Algoritmas eina per visas veiklas sujungtas „Control Flow“ ryšiu. Visos algoritmo praeitos veiklos yra saugomos atskirame sąraše. Jeigu randamas lygiagretumo elementas (*angl. Fork)* vykdoma vidinė paieška į gylį, tol kol visi galimi keliai esantys tarp lygiagretumo ir suliejimo (*angl. Join)* elementų nebūna aplankyti. Paieška vykdoma iki pabaigos žymės, jeigu yra nurodytas ryšys į jau aplankytą veiklą arba kitą elementą yra laikoma, kad rastas naujas testavimo scenarijus. Atgalinis ryšys, laikomas kaip testavimo scenarijaus pabaigos žymė.

Kiekvienas kelias, nuo pradžios iki pabaigos elemento, rastas veiklos diagramoje yra laikomas atskiru testavimo scenarijumi, o veiklų seka, laikoma testavimo scenarijaus tikėtina seka.

Algoritmo pabaigoje, kiekvienam rastam keliui, nustatomos duomenų įėjimo bei išėjimo vietos bei pagal nuskaitytus apribojimus ir veiklose sumodeliuotus veiksmus su kintamaisiais yra generuojami įėjimo bei išėjimo duomenys. Jeigu rastame testavimo scenarijuje yra sumodeliuoti apribojimai duomenims ir jie prieštarauja vienas kitam, t.y. neįmanoma nustatyti duomenų su kuriais testavimo scenarijuje aprašyti apribojimai būtų tenkinami, laikoma, kad, visas testavimo scenarijus yra neįmanomas ir jis vartotojui nėra pateikiamas.

Algoritmu generuojami skaitiniai(*angl. integer)* bei tekstiniai (*angl. string)* tipo duomenys. Skaitinėms reikšmėms generuoti iš apribojimų nustatomos maksimalios ir minimalios reikšmės ir paimamas atsitiktinis skaičius tenkinantis rastą intervalą. Tekstinėms reikšmėms tenkinti yra nuskaitomas tekstinis apribojimas ir pateikiama tokia pati tekstinė reikšmė, kuri yra apribojime, o tekstinėms reikšmėms nepatenkinti yra generuojama atsitiktinė raidžių kombinacija kuri yra, sumodeliuotos, tinkamos tekstinės reikšmės ilgio.

Detalus pilkosios dėžės testų generavimo algoritmas pateiktas 11 pav.



11 pav. Detalus pilkosios dėžės testų generavimo algoritmas

3.3. Apribojimai veiklos diagramoms

- Apribojimai gali būti taikomi ryšiams tarp veiklų, veikloms
- Kiekvienas veiklos modelis turi turėti pradžios bei pabaigos taškus ir bent vieną veiklą tarp jų.

4. SPRENDIMO REALIZACIJA IR TESTAVIMAS

4.1. Sprendimo realizacijos ir veikimo aprašas

Sistema realizuota PHP (*angl. hypertext preprocessor*) kalba.

Sprendimui realizuoti panaudotos reikalavimų modelio veiklų diagramos su pritaikytais OCL(*angl. Object Consctraint Language*) 2.4 parametrais. Realizuotas pilkosios dėžės (*angl. gray-box*) testavimo metodas, turintis tiek baltosios dėžės (*angl. white-box*), logikos parametrus , tiek juodosios dėžės (*angl. black-box*) testavimo metodo tikrinimą ar sistemoje reikalavimas, šiuo atveju, veiklų sekos, yra realizuotos.

Realizacijoje naudojamas pilkosios dėžės testų generavimo algoritmas, algoritmo baziniai veiksmai pateikiami žemiau:

1. Identifikuoti duomenų įėjimus (Veiklos diagrama)
2. Identifikuoti duomenų išėjimus (Veiklos diagrama)
3. Nustatyti pagrindinius kelius (Veiklos diagrama)
4. Nustatyti sub-funkcijas (SF)X (Veiklos diagramos apribojimai panaudojant OCL)
5. Generuoti įėjimo duomenis funkcijai SF X
6. Generuoti išėjimo duomenis funkcijai SF X
7. Generuoti testavimo atvejus kiekvienam rastam keliui

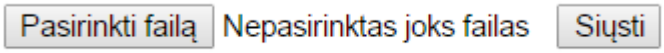
Pradinėje algoritmo fazėje identifikuojami duomenų įėjimai bei išėjimai kurie yra sumodeliuoti veiklos procesų diagramoje. Kitame etape nustatomos visos atskiros veiklos šakos bei identifikuojami apribojimai realizuoti veikloms ar perėjimams tarp veiklų. Sekančiame etape pagal nuskaitytus apribojimus sugeneruojami įėjimo, bei išėjimo duomenys, taip, kad būtų tenkinami apribojimai. Iš šių surinktų duomenų, generuojami testavimo atvejai pagal pavyzdį pateiktą 6 lentelėje. Pilkosios dėžės testavimo atvejo modelis susideda iš 3 elementų:

- Įvedamų duomenų
- Tikėtinos veiklų sekos
- Išėjimo duomenų

6 lentelė. Pilkosios dėžės generuojamo testo šablonas

Įvedami duomenys	**Tikėtina seka**	**Tikėtini išėjimo duomenys**
Jeigu yra nustatomi įvedami duomenys, jie vaizduojami šiame lauke.	*Šiame stulpelyje vaizduojama tikėtina sugeneruotų veiklų seka, nuo pirmos iki paskutinės (iš viršaus į apačią)*	*Šiame stulpelyje vaizduojami tikėtini išėjimo duomenys (rezultatai, pranešimai)*

Norint pradėti darbą įkeliamas „MagicDraw“ projekto failas spaudžiant mygtuką „Pasirinkti failą“, pavaizduota sistemos realizuota vartotojo sąsaja 12 pav. po to spaudžiamas mygtukas „Siųsti“.



12 pav. Realizuotos sistemos failo įkėlimas

Įkeltas „MagicDraw“ (*.mdzip) failas nuskaitomas

Pagrindiniai XML dokumentų nuskaitymo būdai – DOM ir SAX.

DOM (*angl. Document Object Model*) – visa dokumento struktūra nuskaitoma į atmintį ir visi veiksmai atliekami su atmintyje nuskaitytu dokumento elementų medžiu.

SAX (*angl. Simple API for XML*) – įvykiais grindžiamas XML dokumento nuskaitymas. Tokiu būdu nuskaitant dokumentą yra iškviečiamos funkcijos prieš pradedant ir baigiant skaityti dokumentą bei prieš pradedant skaityti ir baigiant skaityti elementą. Tokiu būdu galima nuskaityti reikiamą informaciją iš dokumento neįkeliant viso dokumento į atmintį.

Pasirinktas dokumento nuskaitymo būdas - DOM elementų medis. Naudojantis gautu medžiu sudaromi objektai, reikalingi testų generavimo algoritmui pateiktam 3.2 skyriuje, objektų aprašymai pateikiami 7-12 lentelėse. Panaudojus gautus objektus, generuojami testavimo scenarijai pagal šabloną pateiktą 4 lentelėje ir jie atvaizduojami naršyklėje.

Edge – naudojamas atpažinti ryšius tarp veiklų, aprašymas pateikiamas 7 lentelėje.

7 lentelė. Edge atributai

Tipas	**Aprašymas**
source	Veiklos ID numeris
target	Susietos sekančios veiklos ID numeris

Validavimo taisyklių žinutės – naudojamas išrinkti validavimo klaidų žinutes, pateikiamas aprašymas 8 lentelėje.

8 lentelė. Validavimo taisyklių žinučių atributai

Tipas	**Aprašymas**
Base_Constraint	Apribojimo ID identifikacijos numeris
errorMessage	Saugomas klaidos žinutės tekstas

Elemento „*node“* tipai ir su jais susiję veiksmai pateikiami 9 lentelėje.

9 lentelė. Elemento „Node“ tipai ir su jais susiję veiksmai

***node* tipas**	**Atliekamas veiksmas**
uml:CallBehaviorAction	Išsaugomas veiklos pavadinimas.
uml:InitialNode	Išsaugomas veiklos diagramos pradžios elementas.
uml:ActivityFinalNode	Išsaugomas veiklos diagramos pabaigos elementas.

Elemento „*Parameter“* tipai ir jų aprašymai pateikiami 10 lentelėje.

10 lentelė. Elemento „Parameter“ tipai ir su jais susiję veiksmai

Tipas	**Atliekamas veiksmas**

Tipas	**Atliekamas veiksmas**
type	Išsaugomas parametro tipas
Id	Išsaugomas parametro ID identifikacijos numeris
name	Išsaugomas parametro vardas
value	Išsaugoma parametro reikšmė

Elemento „*Constraint“* tipai ir jų aprašymai pateikiami 11 lentelėje.

11 lentelė. Elemento „Constraint“ tipai ir su jais susiję veiksmai

Tipas	**Atliekamas veiksmas**
constrainedElement:idref	Išsaugomas apriboto element ID identifikacijos numeris
Id	Išsaugomas apribojimo ID identifikacijos numeris
name	Išsaugomas apribojimo vardas
body	Išsaugoma apribojimo reikšmė

Elemento *„To Do“* tipai ir jų aprašymai pateikiami 12 lentelėje.

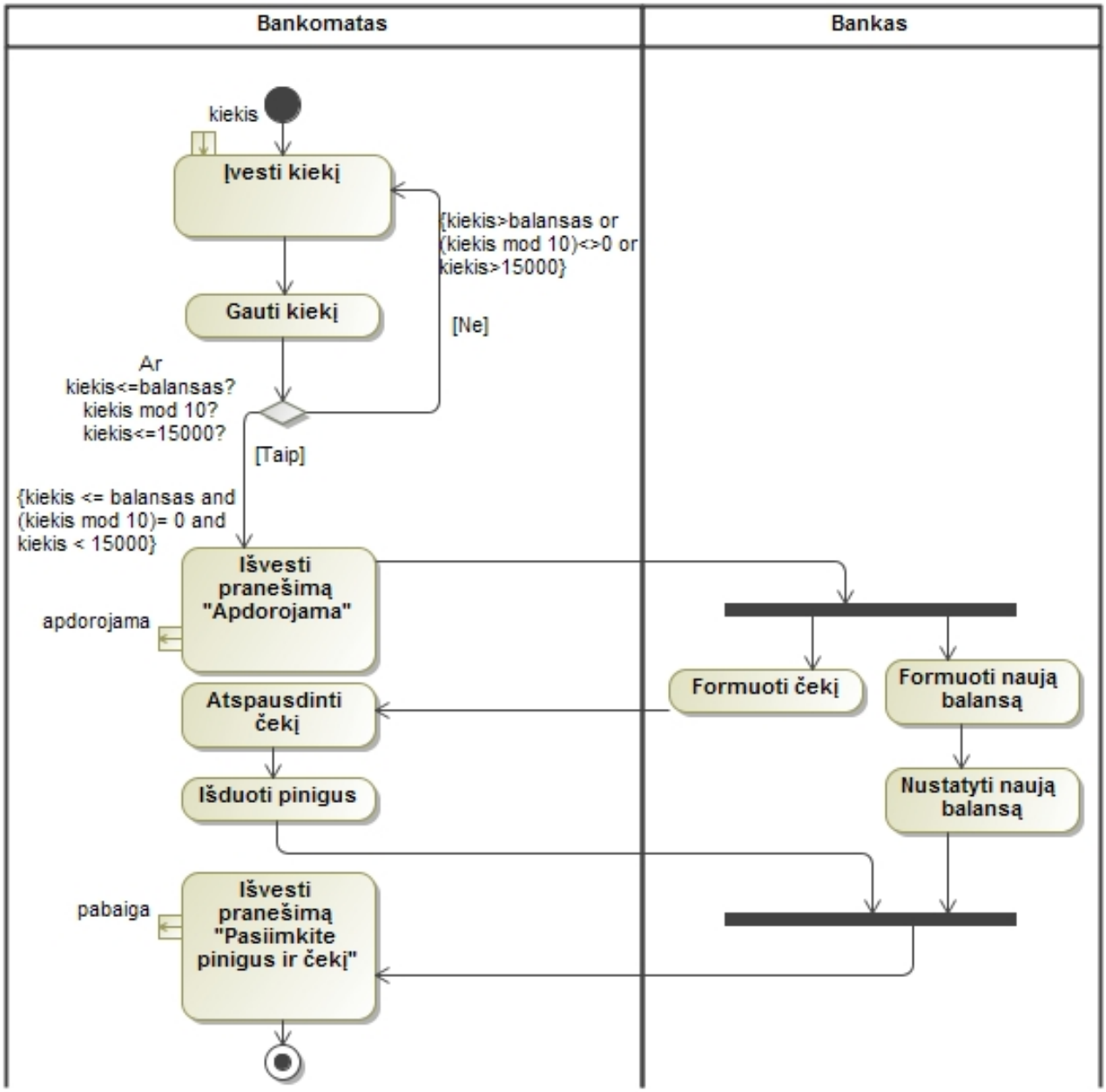
12 lentelė. Elemento „To Do“ tipai ir su jais susiję veiksmai

Tipas	**Atliekamas veiksmas**
baseElement:idref	Išsaugomas aprašytoo element ID identifikacijos numeris
Id	Išsaugomas veiksmo ID identifikacijos numeris
todo	Išsaugomas aprašytas veiksmas

4.2. Testavimo modelis, duomenys, rezultatai

Testų generavimo įrankis, skirtas generuoti testus pagal veiklos diagramose sumodeliuotas diagramas.

Šios realizuotos sistemos testavimui yra sudaromas testavimo planas, kuriame atsispindi, kokius veiksmus reikia atlikti testuojant sistemą ir pilkosios dėžės testų generavimo algoritmą, taip pat, kokia yra laukiama sistemos reakcija į atliekamus veiksmus. Pradinis algoritmo veikimas patikrintas pagal mokslinės publikacijos [13] pavyzdį pateiktą 4 pav. šio modelio atitikmuo pritaikius OCL 2.4 pateiktas 13 pav. Duomenys gauti pradinio testavimo metu pateikti 14 pav.



13 pav. Veiklos diagrama „Pasiimti grynų pinigų“

TA NR:1	Veiklos diagrama: Išsigryninti pinigus	
Įvedami duomenys	**Tikėtina seka**	**Tikėtini išėjimo duomenys**
260	Bankomatas.Įvesti kiekį	
	Bankomatas.Gauti kiekį	
	Bankomatas.Ar kiekis<=balansas? kiekis mod 10? kiekis<=15000?	
	Bankomatas.Išvesti pranešimą "Apdorojama"	Apdorojama
	Bankas.Formuoti naują balansą	
	Bankas.Nustatyti naują balansą	
	Bankas.Formuoti čekį	
	Bankomatas.Atspausdinti čekį	
	Bankomatas.Išduoti pinigus	
	Bankomatas.Išvesti pranešimą "Pasiimkite pinigus ir čekį"	Pasiimkite pinigus ir čekį
	Pabaigos žymė	

TA NR:2	Veiklos diagrama: Išsigryninti pinigus	
Įvedami duomenys	**Tikėtina seka**	**Tikėtini išėjimo duomenys**
22253	Bankomatas.Įvesti kiekį	
	Bankomatas.Gauti kiekį	
	Bankomatas.Ar kiekis<=balansas? kiekis mod 10? kiekis<=15000?	Netinkamas kiekis
22253	Bankomatas.Įvesti kiekį	

14 pav. Automatiškai sugeneruoti pilkosios dėžės testavimo atvejai iš sekų diagramos pateiktos 13 pav.

Įvykdžius pirminį algoritmo testavimą, algoritmas sugeneravo teisingus duomenis, duomenys pateikti 14 pav., lyginant su duomenimis pateiktais Wang Linzhang, Yuan Jiesong, Yu Xiaofeng, Hu Jun, Li Xuandong and Zheng Guoliang publikacijoje [13]

13 lentelė. Sistemos testavimo planas

1. Įkelti *Magic Draw* projektą		
1.1	Nepasirenkamas joks failas.	Neįkeliamas failas.
1.2	Pasirenkamas *.mdzip formato failas.	Įkeliamas *MagicDraw UML* projekto failas, sugeneruojami testavimo atvejai pagal projekte pateiktas veiklos diagramas.
2. Algoritmo testavimas Pradinė sąlyga visiems modeliams Pradžios (Initial Node) ir pabaigos (Activity Final Node) žymomis		
2.1	Įkeliama diagrama su *action* tipo ir *decition* tipo elementu išsiskirstančiu į 2 kelius, kuriuose yra bent po vieną *action* tipo elementą	Algoritmas sugeneruoja 2 testavimo atvejus

2.2	Įkeliama diagrama su *Fork* tipo elementu bei *Join* tipo elementų tarp kurių 2 susieti *action* tipo elementai	Algoritmas atspausdina visas veiklas esančias tarp sumodeliuotų *Fork* ir *Join* tipo elementų
2.3	Įkeliama diagrama su *action* tipo ir *decition* tipo elementu išsiskirstančiu į 2 kelius, kuriuose yra bent po vieną *action* tipo elementą, kurie yra skirtinguose plaukimo takeliuose (angl. *Swimlanes*)	Algoritmas sugeneruoja 2 testavimo atvejus ir atspausdina kokiuose takeliuose yra kiekviena iš sumodeliuotų elementų
2.4	Įkeliama diagrama su *action* tipo ir *decition* tipo elementu išsiskirstančiu į 2 kelius, kuriuose yra bent po vieną *action* tipo elementą, kurie yra skirtinguose plaukimo takeliuose (angl. *Swimlanes*). *Decition* elementų šakai pritaikomas apribojimas skaičiumi (*integer* tipo). Pirmam *action* tipo elementui įvedamas *pameter* tipo elementas	Algoritmas sugeneruoja 2 testavimo atvejus ir atspausdina kokiuose takeliuose yra kiekviena iš sumodeliuotų elementų, taip pat pateikia sugeneruotą skaičių pagal pritaikytą apribojimą ir atvaizduoja jį prie jam priskirto elemento.
2.5	Įkeliama diagrama su *action* tipo ir *decition* tipo elementu išsiskirstančiu į 2 kelius, kuriuose yra bent po vieną *action* tipo elementą, kurie yra skirtinguose plaukimo takeliuose (angl. *Swimlanes*). *Decition* elementų šakai pritaikomas apribojimas tekstu (*string* tipo). Pirmam *action* tipo elemetui įvedamas *pameter* tipo elementas	Algoritmas sugeneruoja 2 testavimo atvejus ir atspausdina kokiuose takeliuose yra kiekviena iš sumodeliuotų elementų, taip pat pateikia sugeneruotą tekstą pagal pritaikytą apribojimą ir atvaizduoja jį prie jam priskirto elemento.
2.6	Įkeliama diagrama su *action* tipo ir *decition* tipo elementu išsiskirstančiu į 2 kelius, kuriuose yra bent po vieną *action* tipo elementą, kurie yra skirtinguose plaukimo takeliuose (angl. *Swimlanes*). *Decition* elementų šakai pritaikomas apribojimas tekstu (*integer* tipo), apribojimui įvedama klaidos žinutė. Pirmam *action* tipo elementui įvedamas *pameter* tipo elementas	Algoritmas sugeneruoja 2 testavimo atvejus ir atspausdina kokiuose takeliuose yra kiekviena iš sumodeliuotų elementų, taip pat pateikia sugeneruotą skaičių pagal pritaikytą apribojimą ir atvaizduoja jį prie jam priskirto elemento, atvaizduoja klaidos tipo pranešimą prie priskirto elemento ryšio.
3. Parametrų testavimas **Atliekami testavimo punktai 2.4-2.6 su pakeitimais apribojimo tipo elementams**		

3.1	Panaudojama apribojimo elemento struktūra (sąlyga) AND (sąlyga)	Išpildomi abu apribojimai, generuojant skaičių(integer tipo), tekstą(string tipo)
3.2	Panaudojama apribojimo elemento struktūra (sąlyga) OR (sąlyga)	Išpildomas bent vienas apribojimas, generuojant skaičių(integer tipo), tekstą(string tipo)
3.3	Panaudojama apribojimo elemento struktūra (skaičius *integer*) DIV (skaičius *integer*)	Įvykdoma DIV funkcija
3.4	Panaudojama apribojimo elemento struktūra (skaičius *integer*) MOD (skaičius *integer*)	Įvykdoma MOD funkcija

Išvados:

Pasinaudojus testavimo planu aptiktos klaidos realizuotoje programinėje įrangoje:

- Sukurta IS negali apdoroti duomenų iš skirtingų struktūrų „MagicDraw“ failų.

Šiuo metu sukurtas IS prototipas generuoja testavimo scenarijus iš veiklos diagramų sumodeliuotų MagicDraw UML 18.0 aplinkoje, kai projekte yra tik viena veiklos diagrama. Ateityje planuojama realizuoti galimybę nuskaityti įvairių tipų MagicDraw UML projektų failus, su veiklos diagramų pasirinkimų galimybe, kurioms norima sugeneruoti testavimo scenarijus, bei pritaikyti testavimo scenarijų prioretizavimo technikas, tiek tarp veikloje esančių testavimo scenarijų, tiek tarp skirtingų veiklos diagramų.

5. EKSPERIMENTINIS VEIKLOS MODELIU GRINDŽIAMOS PILKOSIOS DĖŽĖS TESTŲ GENERAVIMO TYRIMAS

Šiame skyriuje pateikiamas realizuoto rankinių testų generavimo prototipo eksperimentinis tyrimas.

5.1. Eksperimento planas

Realizuoto prototipo įvertinimui pateiktos 2 skirtingos veiklos diagramos, turinčios skirtingą veiklų, kintamųjų bei operacijų kiekį atliekamą su aprašytais kintamaisiais. Tam, kad įvertinti prototipo darbą, naudojantis pateiktomis diagramomis, bus atliktas rankinis bei automatinis pilkosios dėžės testų sudarymas.

Prieš pradedant rankinį testų kūrimą kiekvienas eksperimente dalyvaujantis žmogus yra supažindamas su veiklos diagramomis, jų elementais, pilkosios dėžės testų generavimo metodu, pagrindinėmis naudojamomis matematinėmis reikšmėmis, operatoriais bei funkcijomis.

Rankinis testų sudarymas laikomas užbaigtu, kai:

- Rasti visi galimi testavimo scenarijai
- Nuosekliai aprašyta kiekviena rasta tikėtina testavimo scenarijaus seka
- Tinkamose vietose nustatyti duomenų įėjimo bei išėjimo taškai
- Įrašyti duomenys atitinkantys aprašytus apribojimus įėjimo bei išėjimo taškuose
- Įrašyti sumodeliuoti įvedimo ir išvedimo pranešimai

Testų generavimui naudojamas ir eksperimento dalyviams pateikiamas 6 lentelėje pavaizduotas testų generavimo šablonas.

Eksperimentui atlikti naudojamas 2 veiklos diagramos pateiktos 13 pav. ir 15 pav., kaip priedas apie veiklos diagramą yra pateikiama papildoma specifikacija apie apribojimus kiekvienam iš veiklos modelių, specifikacijos pateiktos 14 ir 15 lentelėse.

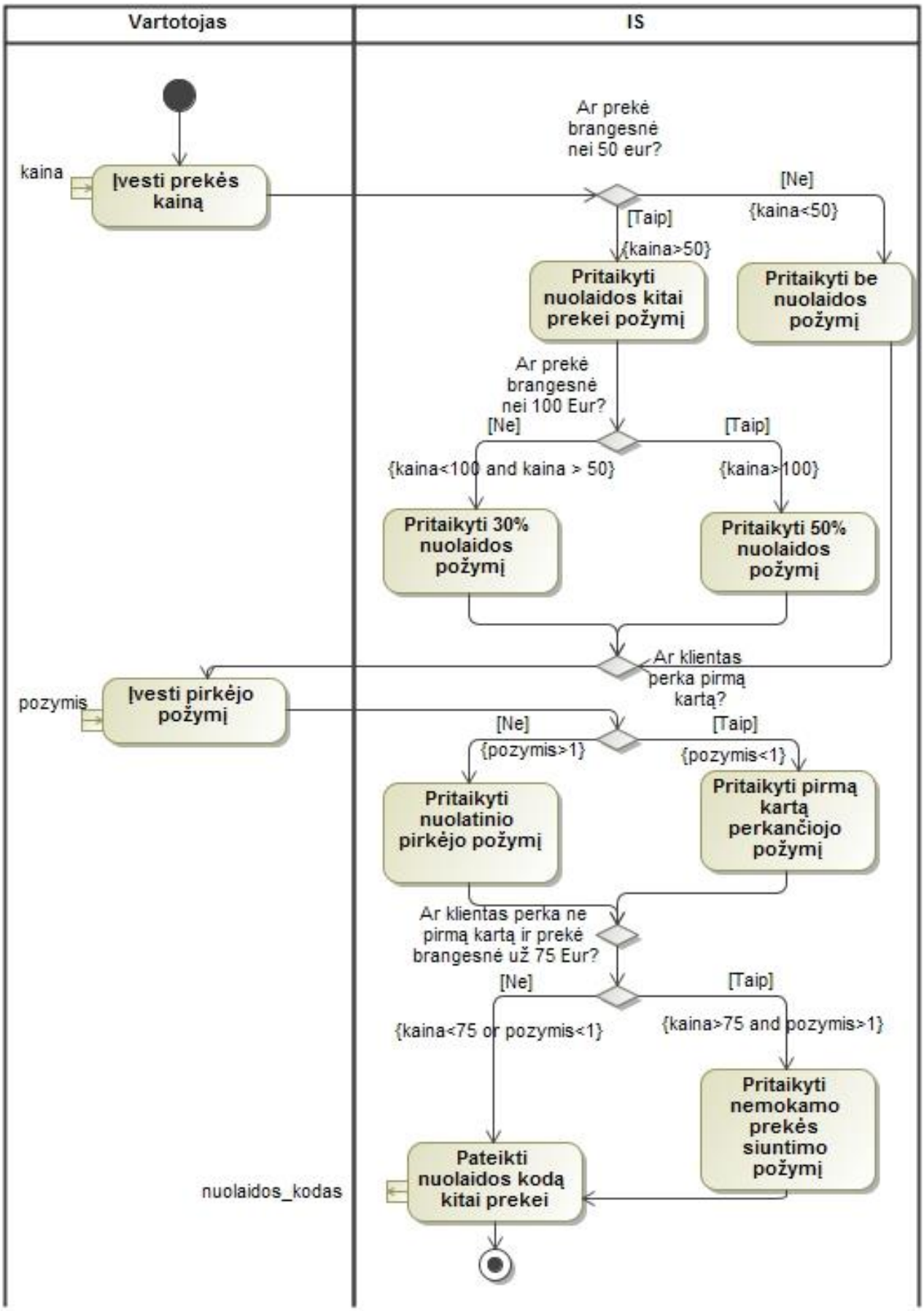
14 lentelė. 13 pav. pateikiamos veiklos diagramos „Pasiimti grynų pinigų“ paaiškinimai, papildomi duomenys.

Apribojimas, kintamasis, sprendimo taškas	Paaiškinimas
Balansas = 2000	Balansas turi reikšmę diagramoje 2000
Kiekis <= Balansas	Kiekis turi būti mažesnis nei Balansas reikšmė
Kiekis mod 10	Kiekis turi dalintis iš 10 be liekanos
Kiekis <=15000	Kiekis turi būti mažesnis nei 15000

15 lentelė. 15 pav. Pateikiamos „Generuoti nuolaidos kodą“ veiklos diagramos paaiškinimai, papildomi duomenys

Apribojimas, kintamasis, sprendimo taškas	Paaiškinimas
Nuolaidos_kodas=””	Aprašytas tuščias String tipo kintamasis. Jis yra generuojamas iš veiklos diagramoje aprašytų veiksmų pagal aprašytus apribojimus.
Kaina < 50	Jei kaina mažesnė už 50, prie kintamojo nuolaidos_kodas prirašoma reikšmė „JDD882“
Kaina > 50	Jei kaina didesnė už 50, prie kintamojo nuolaidos_kodas prirašoma reikšmė „JDS“
Kaina > 100	Jei kaina didenė už 100, prie kintamojo nuolaidos_kodas prirašoma reikšmė „393“
Kaina < 100	Jei kaina mažesnė už 100, prie kintamojo

Apribojimas, kintamasis, sprendimo taškas	**Paaiškinimas**
	nuolaidos_kodas prirašoma reikšmė „358“
Pozymis	Nurodoma reikšmė, kiek kartų klientas apsipirko
Pozymis=0	Jei klientas perka pirmą kartą pritaikomas pirmą kartą perkančiojo požymis. Prie kintamojo nuolaidos_kodas prirašoma reikšmė „N1“
Pozymis>0	Jei klientas perka ne pirmą kartą pritaikomas nuolatinio pirkėjo požymis. Prie kintamojo nuolaidos_kodas prirašoma reikšmė „N2“
Pozymis>0 AND kaina >75	Jei klientas yra nuolatinis ir prekės kaina didesnė nei 75, prie kintamojo nuolaidos_kodas prirašoma reikšmė „5598“
Pozymis=0 OR kaina <75	Jei klientas perka pirmą kartą arba prekės kaina mažesnė nei 75, prie kintamojo nuolaidos_kodas prirašoma reikšmė „5599“



15 pav. Eksperimentinė „Generuoti nuolaidos kodą“ veiklos diagrama

16 lentelė. Eksperimento veiklos diagramose pateikiamų duomenų kiekis

Veiklos diagrama	**Veiklų kiekis**	**Įeinamų duomenų kiekis**	**Išeinančių duomenų kiekis**	**Sprendimų būsenų kiekis**	**Galimų testavimo scenarijų kiekis**	**Operacijų kiekis**
13 pav.	9	1	2	1	2	5
15 pav.	9	2	1	4	6	13

5.2. Eksperimento rezultatai

Šiame skyriuje pateikiami eksperimento rezultatai. 17 lentelėje pateikiami laikai minutėmis, kurie buvo reikalingi sugeneruoti testavimo scenarijus pagal pateiktus veiklos modelius bei jų specifikacijas eksperimento dalyviams bei naudojantis įrankiu. Eksperimento metu sugeneruoti testavimo scenarijai pateikti darbo priedų 8.1 skyriuje.

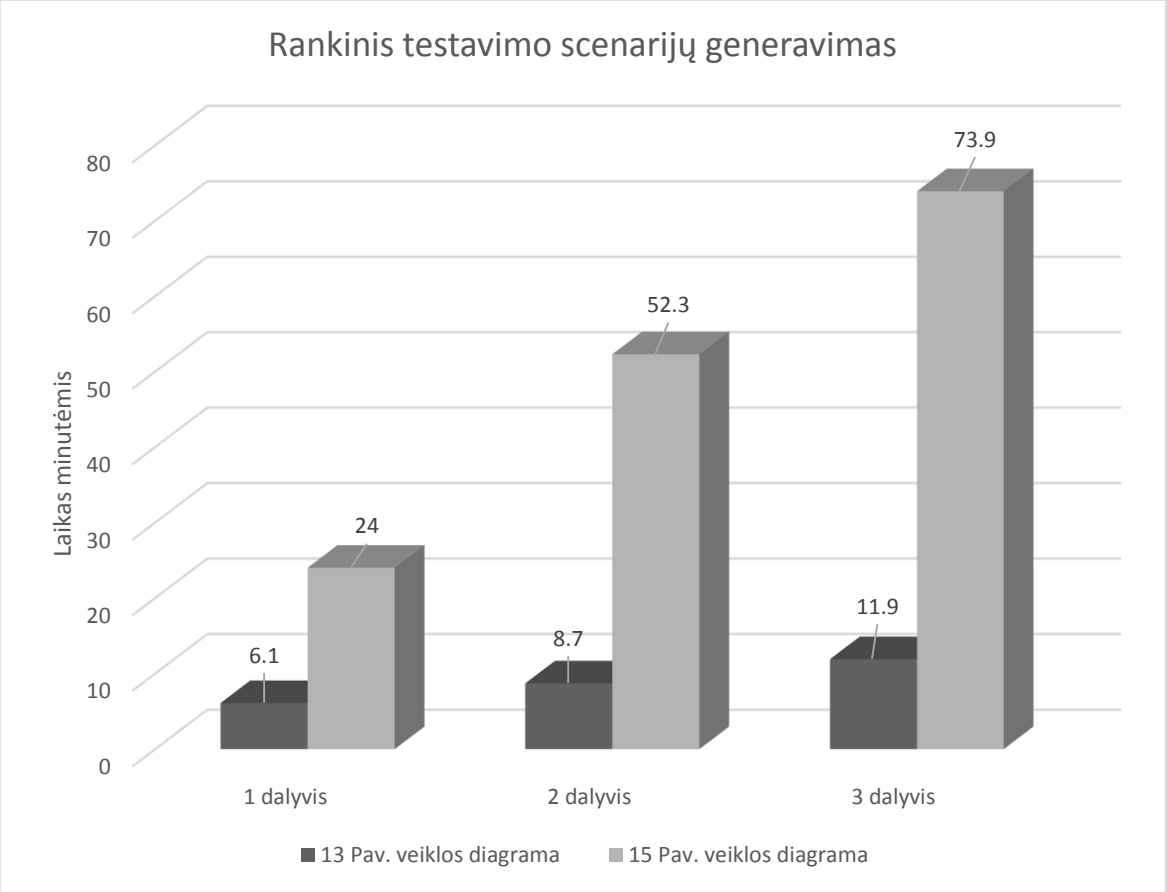
17 lentelė. Eksperimento meto gautų laikų duomenys

Dalyvis / Veiklos diagrama	1 dalyvis – ekspertas (Rankinis testavimo scenarijų sudarymas)	2 dalyvis (Rankinis testavimo scenarijų sudarymas)	3 dalyvis (Rankinis testavimo scenarijų sudarymas)	Vidutinis rankinių testavimo scenarijų sudarymo laikas	Automatinis testavimo scenarijų generavimas
13 Pav. „Pasiimti grynų pinigų“ veiklos diagrama	6.1 min	8.7 min	11.9 min	8.9 min	0.01 min
15 Pav. „Generuoti nuolaidos kodą“ veiklos diagrama	24 min	52.3 min	73.9 min	50.07 min	0.02 min

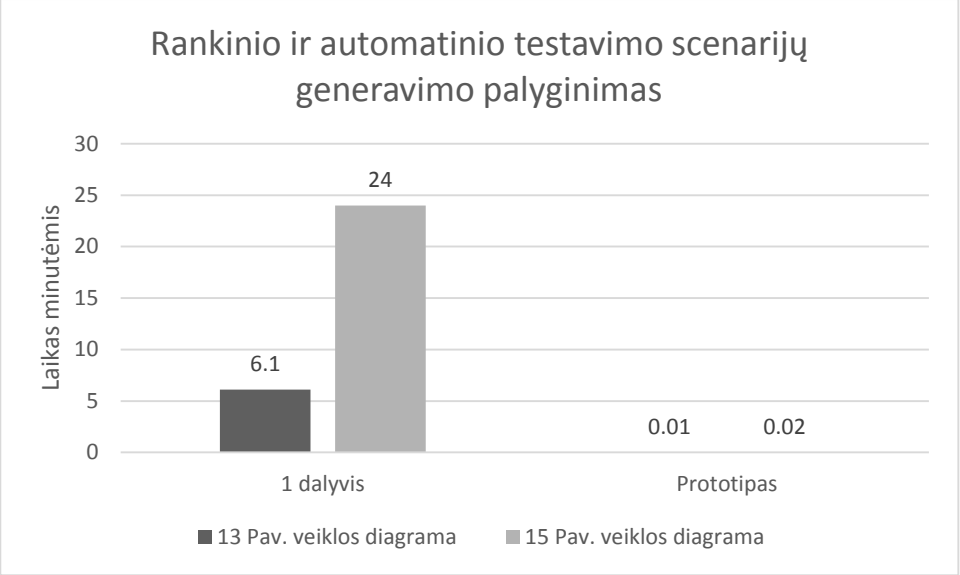
5.3. Sprendimo veikimo ir savybių analizė, kokybės kriterijų įvertinimas

Eksperimente dalyvavo 3 dalyviai, 1 dalyvis - IS ekspertas bei 2 žmonės neturintys informacinių technologijų išsilavinimo. Iš gautų rezultatų 17 lentelėje galima spręsti, kad rankinis testų sudarymas užtrunka ilgai, lyginant su automatiniu testų generavimu, net testavimo scenarijus generuojant IS ekspertui. Kitų eksperimento dalyvių rezultatus įtakojo klaidų darymas, ieškant visų galimų testavimo scenarijų, generuojant įėjimo bei išėjimo duomenis.

Sprendžiant iš 16 bei 17 lentelėse pateiktų duomenų galima teigti, kad lyginant 13 pav. bei 15 pav. pateiktas veiklos diagramas didžiausią įtaką rankiniam testų sudarymui laiko atžvilgiu daro sprendimų būsenų kiekis, galimų testavimo scenarijų bei atliekamų operacijų su aprašytais kintamaisiais kiekis. Iš to galima daryti išvadą, kad realizuotą prototipą yra naudinga naudoti net ir esant paprastoms veiklos procesų diagramoms. Didėjant sprendimo būsenų kiekiui, galimų testavimo scenarijų bei operacijų kiekiui, rankinis testavimo scenarijų sudarymas tampa ženkliai ilgesnis, o realizuotam prototipui parametrai aprašyti 16 lentelėje reikšmingos įtakos laiko atžvilgiu neturi.

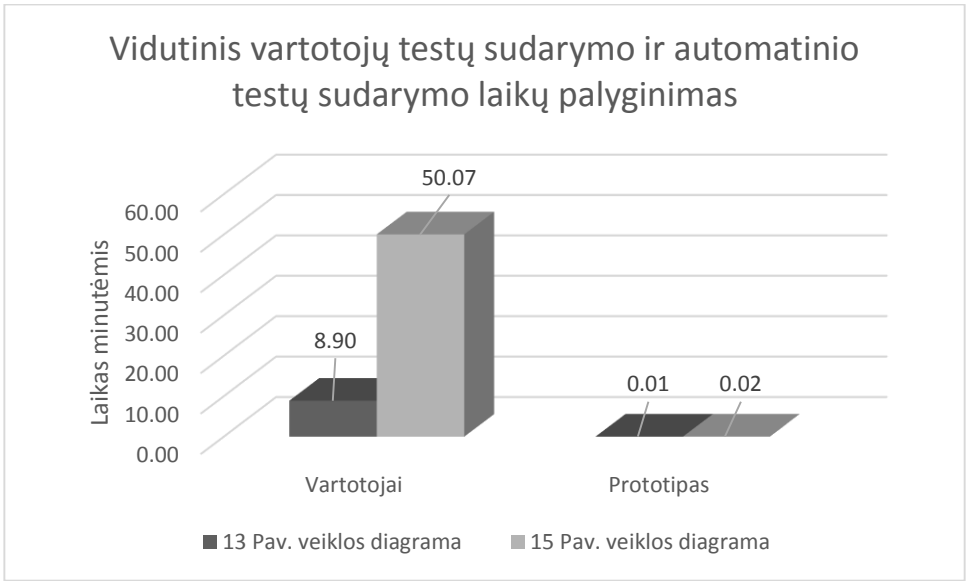


16 pav. Rankinio testavimų atvejų sudarymo laikų grafikas



17 pav. Trumpiausio rankinio ir automatinio testų generavimo laiko atžvilgiu palyginimas

Lyginant trumpiausio laiko testavimo scenarijams sudaryti iš pateiktų veiklos procesų diagramų ir naudojantis įrankiu, duomenys pateikti 17 pav., iš 13 pav. veiklos diagramos rankinius testus automatiškai sugeneruoti užtruko 610 kartų greičiau, lyginant su rankiniu testų sudarymu, o 15 pav. veiklos diagramos pilkosios dėžės testavimo scenarijai, įrankiu buvo sugeneruoti 1200 kartų greičiau nei rankiniu būdu sudaryti testavimo scenarijai. Atsižvelgiant į rankiniu bei automatiniu būdu sudarytų testavimo scenarijų kokybę, klaidų neaptikta. Rankiniu bei automatiniu sudarymo būdu sudaryti testavimo scenarijai buvo identiški ir atitiko darbe aprašytą pilkosios dėžės testavimo metodą.



18 pav. Vidutinio vartotojų testų sudarymo ir automatinio testų sudarymo laikų palyginimas

Lyginant vidutinį laiką, pateiktą 18 pav., reikalingą sudaryti testavimo scenarijus rankiniu būdu ir automatiškai sugeneruotus testavimo scenarijus, nustatyta, kad vidutiniškai pagal 13 pav. pateiktą veiklos diagramą, rankiniu būdu testavimo scenarijai vidutiniškai sudaryti per 8.9 min. t.y. 890 kartų ilgiau, nei automatiniu būdu sudaryti testavimo scenarijai 0.01 min. , o pagal 15 pav. pateiktą veiklos diagramą, rankiniu būdu sudaryti testavimo scenarijus, vidutiniškai užtruko 50.07 min. t.y. 2503 kartus ilgiau nei automatiniu būdu 0.02 min.

6. REZULTATŲ APIBENDRINIMAS IR IŠVADOS

1. Testavimo scenarijų generavimas iš reikalavimų modelio yra naudingas, nes leidžia ankstyvojoje projekto fazėje nustatyti galimas klaidas, taip sumažinant projektui reikalingus kaštus.
2. Analizuotuose egzistuojančiuose sprendimuose testavimo scenarijų generavimas iš reikalavimų modelio yra sudėtingas, reikalauja didelės modeliuotojo kompetencijos korektiškiems testams sugeneruoti.
3. Įvertinus reikalavimus testų generavimui nustatyta, kad pilkosios dėžės testavimo scenarijai susideda iš įeinančių, išeinančių duomenų bei tikėtinos testavimo sekos.
4. Išanalizavus diagramas leidžiančias generuoti testavimo scenarijus nustatyta, kad veiklos modeliai apriboti OCL gali būti naudojami pilkosios dėžės testų generavimui.
5. Iš atliktos analizės nustatyta, kad nėra sprendimo leidžiančio generuoti testavimo scenarijus iš veiklos modelių be papildomų diagramų bei ryšių sudarymo, kas reikalauja didelės modeliuotojo kompetencijos, todėl nuspręsta naudoti reikalavimų modelio pradines veiklos diagramas pritaikant OCL apribojimus, testavimo scenarijams atrinkti bei įėjimo ir išėjimo duomenims generuoti.
6. Realizuotas prototipas leidžiantis generuoti testavimo scenarijus iš veiklos diagramų apribotų OCL, naudojant duomenis iš MagicDraw įrankiu sukurto projekto XML failo.
7. Atlikus eksperimentą nustatyta, kad sukurtą testavimo scenarijų generavimo prototipą verta naudoti norint greičiau sugeneruoti testavimo scenarijus, kai projekto reikalavimų etape yra sudaromos UML diagramos.
8. Sukurtas prototipas leidžia generuoti testavimo scenarijus neatsižvelgiant į tai kokioje aplinkoje bus naudojama kuriama informacinė sistema.

7. LITERATŪRA

[1] Bartos, J., Walek, B., A methodology for testing of information system under uncertainty, Telecommunications and Signal Processing (TSP), 2013 36th International Conference on Digital Object Identifier: 10.1109/TSP.2013.6613883, 2013 , p. 20 - 22.

[2] Kobrosly, W., Vassiliadis, S., A survey of software functional testing techniques, Southern Tier Technical Conference, 1988., Proceedings of the 1988 IEEE, 1988, p. 127 – 134.

[3] Friedman, John Lee and Jon, Requirements Modeling and Automated Requirements-Based Test Generation, SAE International, MathWorks, 2013.

[4] Predrag Skoković, Marija Rakić-Skoković, Requirements-Based Testing Process in Practice, International Journal of Industrial Engineering and Management (IJIEM), 2010, p. 155 – 161.

[5] Harsh Bhasin, Esha Khanna, Sudha, Black Box Testing based on Requirement Analysis and Design Specifications, International Journal of Computer Applications (0975 – 8887), 2014, Volume 87 – No.18.

[6] Kavitha, R.; Comput. Sci. & Eng., VCET, Madurai, India; Kavitha, V.R.; Kumar, N.S., Requirement based test case prioritization, Communication Control and Computing Technologies (ICCCCT), 2010 IEEE International Conference, 2010 p. 826-829.

[7] Sunitha E.V, Philip Samuel, Enhancing UML Activity Diagrams using OCL, Computational Intelligence and Computing Research (ICCIC), 2013 IEEE International Conference, 2013.

[8] „OMG Object Constraint Language," [Tinkle]. Available: http://www.omg.org/spec/OCL/2.4/PDF/.

[9] Ali, S., Certus Software V&V Center, Simula Res. Lab., Lysaker, Norway; Zohaib Iqbal, M.; Arcuri, A.; Briand, L.C., Generating Test Data from OCL Constraints with Search Techniques, Software Engineering, IEEE Transactions (Volume:39 , Issue: 10), 2013 p. 1376 - 1402.

[10] Khan, Mohd. Ehmer, Khan, Farmeena, A Comparative Study of White Box, Black Box and Grey Box Testing Techniques, (IJACSA) International Journal of Advanced Computer Science and Applications, 2012, Vol. 3, No.6.

[11] Alessandra Cavarra, Charles Crichton, Jim Davies, Alan Hartman, Laurent Mounier, Using UML for Automatic Test Generation, INTERNATIONAL SYMPOSIUM ON SOFTWARE TESTING AND ANALYSIS ISSTA, 2002.

[12] Tahat, L.H., Vaysburg, B., Korel, B., Bader, A.J., Requirement-based automated black-box test generation, Computer Software and Applications Conference, 2001. COMPSAC 2001. 25th Annual International, 2001, p. 485-495.

[13] Wang Linzhang, Yuan Jiesong, Yu Xiaofeng, Hu Jun, Li Xuandong, Zheng Guoliang, Generating Test Cases from UML Activity Diagram based on Gray-Box Method, Software Engineering Conference, 11th Asia-Pacific, 2004 p. 284 – 291.

[14] „SmartBear," [Tinkle]. Available: http://smartbear.com/homed/.

[15] „Telerik," [Tinkle]. Available: http://www.telerik.com/.

[16] Mustafa, K.M., Al-Qutaish, R.E., Muhairat, M.I., Classification of Software Testing Tools Based on theSoftware Testing Methods, Computer and Electrical Engineering, 2009. ICCEE '09. Second International Conference on Volume: 1 Digital Object Identifier: 10.1109/ICCEE.2009.9, 2009 , p. 229 – 233.

8. PRIEDAI

8.1. priedas. Eksperimento metu sugeneruoti testavimo scenarijai iš veiklos diagramų.

TA NR:1	**Veiklos diagrama: Išsigryninti pinigus**	
Įvedami duomenys	**Tikėtina seka**	**Tikėtini išėjimo duomenys**
1020	Bankomatas.Įvesti kiekį	
	Bankomatas.Gauti kiekį	
	Bankomatas.Ar kiekis<=balansas? kiekis mod 10? kiekis<=15000?	
	Bankomatas.Išvesti pranešimą "Apdorojama"	Apdorojama
	Bankas.Formuoti naują balansą	
	Bankas.Nustatyti naują balansą	
	Bankas.Formuoti čekį	
	Bankomatas.Atspausdinti čekį	
	Bankomatas.Išduoti pinigus	
	Bankomatas.Išvesti pranešimą "Pasiimkite pinigus ir čekį"	Pasiimkite pinigus ir čekį
	Pabaigos žymė	

TA NR:2	**Veiklos diagrama: Išsigryninti pinigus**	
Įvedami duomenys	**Tikėtina seka**	**Tikėtini išėjimo duomenys**
10885	Bankomatas.Įvesti kiekį	
	Bankomatas.Gauti kiekį	
	Bankomatas.Ar kiekis<=balansas? kiekis mod 10? kiekis<=15000?	Netinkamas kiekis
10885	Bankomatas.Įvesti kiekį	

TA NR:1	**Veiklos diagrama: nuolaidos kodo generavimas**	
Įvedami duomenys	**Tikėtina seka**	**Tikėtini išėjimo duomenys**
5758	Vartotojas.Įvesti prekės kainą	
	IS.Ar prekė brangesnė nei 50 eur?	
	IS.Pritaikyti nuolaidos kitai prekei požymį	
	IS.Ar prekė brangesnė nei 100 Eur?	
	IS.Pritaikyti 50% nuolaidos požymį	
22	Vartotojas.Įvesti pirkėjo požymį	
	IS.Ar klientas perka pirmą kartą?	
	IS.Pritaikyti nuolatinio pirkėjo požymį	
	IS.Ar klientas perka ne pirmą kartą ir prekė brangesnė už 75 Eur?	

TA NR:1	**Veiklos diagrama: nuolaidos kodo generavimas**	
	IS.Pritaikyti nemokamo prekės siuntimo požymį	
	IS.Pateikti nuolaidos kodą kitai prekei	JDS393N25598
	Pabaigos žymė	

TA NR:2	**Veiklos diagrama: nuolaidos kodo generavimas**	
Įvedami duomenys	**Tikėtina seka**	**Tikėtini išėjimo duomenys**
23428	Vartotojas.Įvesti prekės kainą	
	IS.Ar prekė brangesnė nei 50 eur?	
	IS.Pritaikyti nuolaidos kitai prekei požymį	
	IS.Ar prekė brangesnė nei 100 Eur?	
	IS.Pritaikyti 50% nuolaidos požymį	
0	Vartotojas.Įvesti pirkėjo požymį	
	IS.Ar klientas perka pirmą kartą?	
	IS.Pritaikyti pirmą kartą perkančiojo požymį	
	IS.Ar klientas perka ne pirmą kartą ir prekė brangesnė už 75 Eur?	
	IS.Pateikti nuolaidos kodą kitai prekei	JDS393N15599
	Pabaigos žymė	

TA NR:3	**Veiklos diagrama: nuolaidos kodo generavimas**	
Įvedami duomenys	**Tikėtina seka**	**Tikėtini išėjimo duomenys**
40	Vartotojas.Įvesti prekės kainą	
	IS.Ar prekė brangesnė nei 50 eur?	
	IS.Pritaikyti be nuolaidos požymį	
0	Vartotojas.Įvesti pirkėjo požymį	
	IS.Ar klientas perka pirmą kartą?	
	IS.Pritaikyti pirmą kartą perkančiojo požymį	
	IS.Ar klientas perka ne pirmą kartą ir prekė brangesnė už 75 Eur?	
	IS.Pateikti nuolaidos kodą kitai prekei	JDD882N15599
	Pabaigos žymė	

TA NR:4	**Veiklos diagrama: nuolaidos kodo generavimas**	
Įvedami duomenys	**Tikėtina seka**	**Tikėtini išėjimo duomenys**
97	Vartotojas.Įvesti prekės kainą	
	IS.Ar prekė brangesnė nei 50 eur?	
	IS.Pritaikyti nuolaidos kitai prekei požymį	
	IS.Ar prekė brangesnė nei 100 Eur?	
	IS.Pritaikyti 30% nuolaidos požymį	

TA NR:4	**Veiklos diagrama: nuolaidos kodo generavimas**	
22	Vartotojas.Įvesti pirkėjo požymį	
	IS.Ar klientas perka pirmą kartą?	
	IS.Pritaikyti nuolatinio pirkėjo požymį	
	IS.Ar klientas perka ne pirmą kartą ir prekė brangesnė už 75 Eur?	
	IS.Pritaikyti nemokamo prekės siuntimo požymį	
	IS.Pateikti nuolaidos kodą kitai prekei	JDS358N25598
	Pabaigos žymė	

TA NR:5	**Veiklos diagrama: nuolaidos kodo generavimas**	
Įvedami duomenys	**Tikėtina seka**	**Tikėtini išėjimo duomenys**
59	Vartotojas.Įvesti prekės kainą	
	IS.Ar prekė brangesnė nei 50 eur?	
	IS.Pritaikyti nuolaidos kitai prekei požymį	
	IS.Ar prekė brangesnė nei 100 Eur?	
	IS.Pritaikyti 30% nuolaidos požymį	
0	Vartotojas.Įvesti pirkėjo požymį	
	IS.Ar klientas perka pirmą kartą?	
	IS.Pritaikyti pirmą kartą perkančiojo požymį	
	IS.Ar klientas perka ne pirmą kartą ir prekė brangesnė už 75 Eur?	
	IS.Pateikti nuolaidos kodą kitai prekei	JDS358N15599
	Pabaigos žymė	

TA NR:6	**Veiklos diagrama: nuolaidos kodo generavimas**	
Įvedami duomenys	**Tikėtina seka**	**Tikėtini išėjimo duomenys**
40	Vartotojas.Įvesti prekės kainą	
	IS.Ar prekė brangesnė nei 50 eur?	
	IS.Pritaikyti be nuolaidos požymį	
4	Vartotojas.Įvesti pirkėjo požymį	
	IS.Ar klientas perka pirmą kartą?	
	IS.Pritaikyti nuolatinio pirkėjo požymį	
	IS.Ar klientas perka ne pirmą kartą ir prekė brangesnė už 75 Eur?	
	IS.Pateikti nuolaidos kodą kitai prekei	JDD882N25599
	Pabaigos žymė	