Kauno technologijos universitetas

Informatikos fakultetas

Rankinio testavimo atvejų generavimas iš panaudojimo atvejų specifikacijų

Magistro studijų baigiamasis projektas

Projektą parengė

Ieva Valatkevičiūtė

Projektui vadovavo

Doc. dr. Lina Čeponienė

Kaunas, 2026

Kauno technologijos universitetas

Informatikos fakultetas

Rankinio testavimo atvejų generavimas iš panaudojimo atvejų specifikacijų

Magistro studijų baigiamasis projektas
Veiklos skaitmeninimas ir sistemų architektūros (6211BX009)

Projektą parengė
Ieva Valatkevičiūtė

Projektui vadovavo
Doc. dr. Lina Čeponienė

Projektą recenzavo
Prof. dr. Evaldas Vaičiukynas

Kaunas, 2026

Kauno technologijos universitetas

Informatikos fakultetas

Ieva Valatkevičiūtė

Rankinio testavimo atvejų generavimas iš panaudojimo atvejų specifikacijų

Akademinio sąžiningumo deklaracija

Patvirtinu, kad:

1. baigiamąjį projektą parengiau savarankiškai ir sąžiningai, nepažeisdamas (-a) kitų asmenų autoriaus ar kitų teisių, laikydamasis (-i) Lietuvos Respublikos autorių teisių ir gretutinių teisių įstatymo, Kauno technologijos universiteto (toliau – Universitetas) intelektinės nuosavybės valdymo ir perdavimo nuostatų bei Universiteto akademinės etikos kodekse nustatytų etikos reikalavimų;

2. visi baigiamajame projekte pateikti duomenys ir tyrimų rezultatai yra teisingi ir gauti teisėtai, nei viena projekto dalis nėra plagijuota nuo spausdintinių ar elektroninių šaltinių, o visos baigiamojo projekto tekste pateiktos citatos ir nuorodos yra nurodytos literatūros sąraše;

3. baigiamajame projekte tinkamai laikiausi asmens duomenų apsaugos reikalavimų, nenaudojau neskelbtinų ar konfidencialių duomenų be teisėto pagrindo, o jei juos naudojau, jie yra tinkamai nuasmeninti;

4. jei rengiant baigiamąjį projektą naudojausi dirbtinio intelekto (toliau – DI) ar kitais automatizuotais įrankiais, juos taikiau pagal Universitete nustatytą tvarką, nepažeisdamas (-a) akademinio sąžiningumo principų;

5. nesumokėjau ir nesu įsipareigojęs (-usi) mokėti jokių įstatymų nenumatytų piniginių sumų už baigiamąjį projektą ar jo dalis jokiam fiziniam ar juridiniam asmeniui;

6. suprantu, kad išaiškėjus akademinio nesąžiningumo ar kitų asmenų teisių pažeidimo faktui, man bus taikoma atsakomybė pagal Universitete nustatytą tvarką ir galiu būti pašalintas (-a) iš Universiteto; akademinio nesąžiningumo atvejis gali būti nagrinėjamas ir po studijų baigimo, inicijuojant kvalifikacinio laipsnio atšaukimo procedūrą.

Valatkevičiūtė Ieva. Rankinio testavimo atvejų generavimas iš panaudojimo atvejų specifikacijų. Magistro studijų baigiamasis projektas / projektui vadovavo doc. dr. Lina Čeponienė; Kauno technologijos universitetas, Informatikos fakultetas.

Studijų kryptis ir studijų krypčių grupė: Informacijos sistemos, Informatikos mokslai.

Reikšminiai žodžiai: UML, panaudojimo atvejų specifikacija, rankinis testavimas, testavimo atvejai.

Kaunas, 2026. 89 p.

Santrauka

Šiame darbe pristatoma testavimo atvejų generavimo metodika, skirta automatizuoti rankinių testavimo atvejų kūrimą iš UML panaudojimo atvejų diagramų ir jų specifikacijų. Nors programinės įrangos testavimo procesuose vis dar plačiai taikomas rankinis testavimo atvejų kūrimas, šis procesas reikalauja daug laiko, žmogiškųjų išteklių ir priklauso nuo testuotojo patirties bei interpretacijos. Dėl šios priežasties testavimo atvejų sudarymas tampa sudėtingas projektuose, kuriuose naudojamas didelis kiekis panaudojimo atvejų bei jų scenarijų. Atlikta programinės įrangos testavimo, UML panaudojimo atvejų diagramų bei esamų testavimo atvejų generavimo sprendimų analizė parodė, kad dauguma esamų sprendimų testavimo atvejų generavimui naudoja kitų tipų UML diagramas, pavyzdžiui, veiklos ar sekų diagramas, o panaudojimo atvejų diagramas dažniausiai taiko tik kaip papildomą informacijos šaltinį. Taip pat nustatyta, kad esami sprendimai nevertina kartu naudojamų panaudojimo atvejų specifikacijų lentelių turinio, jose aprašytų scenarijų, todėl negali generuoti detalių rankinių testavimo atvejų, remdamiesi tik UML panaudojimo atvejų diagrama ir jos specifikacijomis.

Siekiant spręsti šias problemas, buvo iškeltas darbo tikslas – palengvinti programinės įrangos testavimo procesą, automatizuojant testavimo atvejų kūrimą pagal UML panaudojimo atvejų diagramas ir jų specifikacijas. Šiam tikslui pasiekti buvo suprojektuota testavimo atvejų generavimo metodika, apibrėžianti UML panaudojimo atvejų diagramų ir jų specifikacijų analizės, scenarijų nustatymo bei testavimo atvejų generavimo principus. Metodikai realizuoti sukurtas įrankis, kuris analizuoja pateiktą UML panaudojimo atvejų diagramą ir panaudojimo atvejų specifikacijų lenteles, nustato galimus testavimo scenarijus bei generuoja rankinius testavimo atvejus „Excel“ formatu. Įrankis realizuotas Python programavimo kalba, naudojant „Tkinter“ grafinę naudotojo sąsają, „python-docx“ biblioteką duomenų nuskaitymui iš „Word“ dokumentų bei „openpyxl“ biblioteką sugeneruotų testavimo atvejų eksportui. Panaudojimo atvejų analizėje ir požymių nustatyme taikytas generatyvinio dirbtinio intelekto modelis „OpenAI GPT-4.1-mini“.

Sukurto sprendimo vertinimui atliktas eksperimentinis tyrimas, sudarytas iš dviejų dalių. Pirmojoje dalyje naudoti trijų skirtingų realių projektų duomenys, gauti iš IT įmonės, ir atliktas sugeneruotų testavimo atvejų palyginimas su rankiniu būdu sudarytais testavimo atvejais. Antrojoje dalyje atlikta IT specialistų apklausa, kurios metu vertintas įrankio veikimo aiškumas, naudingumas ir pritaikomumas praktikoje. Eksperimentinio tyrimo rezultatai parodė, kad sukurtas sprendimas gali būti taikomas testavimo atvejų kūrimo procesui palengvinti projektuose, kuriuose naudojamos standartizuotos UML panaudojimo atvejų specifikacijos.

Valatkevičiūtė Ieva. Manual Test Cases Generation from Use Case Specifications. Master‘s Final Degree Project / supervisor assoc. prof. Lina Čeponienė; Faculty of Informatics, Kaunas University of Technology.

Study field and study field group: Information Systems, Computing.

Keywords: UML, use case specification, manual testing, test cases.

Kaunas, 2026. 89 p.

Summary

This thesis presents a test case generation methodology designed to automate the creation of manual test cases from UML use case diagrams and their specifications. Although manual test case creation is still widely used in software testing processes, this activity requires significant time and human resources and largely depends on the tester’s experience and interpretation. As a result, test case creation becomes increasingly complex in projects that involve a large number of use cases and scenarios. An analysis of software testing, UML use case diagrams, and existing test case generation solutions revealed that most existing approaches use other types of UML diagrams, such as activity or sequence diagrams, for test case generation, while use case diagrams are typically applied only as an additional source of information. It was also determined that existing solutions do not evaluate the content of use case specification tables and the scenarios described within them, and therefore are unable to generate detailed manual test cases based solely on UML use case diagrams and their specifications. To address these issues, the aim of the thesis was defined as facilitating the software testing process by automating test case creation based on UML use case diagrams and their specifications. To achieve this goal, a test case generation methodology was designed, defining the principles of UML use case diagram and specification analysis, scenario identification, and test case generation. To implement the methodology, a tool was developed that analyzes the provided UML use case diagram and use case specification tables, identifies possible testing scenarios, and generates manual test cases in Microsoft Excel format. The tool was implemented using the Python programming language, the Tkinter graphical user interface framework, the python-docx library for reading data from Word documents, and the openpyxl library for exporting generated test cases. The generative artificial intelligence model OpenAI GPT-4.1-mini was applied for use case analysis and feature identification. An experimental study consisting of two parts was conducted to evaluate the developed solution. In the first part, data from three different real-world projects obtained from an IT company were used, and the generated test cases were compared with manually created test cases. In the second part, a survey of IT specialists was carried out to evaluate the clarity, usefulness, and practical applicability of the tool. The results of the experimental study demonstrated that the developed solution can be applied to facilitate the test case creation process in projects that use standardized UML use case specifications.

Turinys

Lentelių sąrašas

Paveikslų sąrašas

Santrumpų ir terminų sąrašas

UML (angl. Unified Modeling Language) – standartizuota modeliavimo kalba, kuri naudojama norint vizualiai specifikuoti programinės įrangos sistemas.

OMG (angl. Object Management Group) – objektų valdymo grupė, atsakinga už UML standarto kūrimą.

SDLC (angl. Software Development Life Cycle) – programinės įrangos kūrimo gyvavimo ciklas.

QA (angl. Quality Assurance) – kokybės užtikrinimas, procesų ir metodų rinkinys, skirtas užtikrinti, kad programinė įranga atitiktų reikalavimus.

Testavimo scenarijus – aukštesnio lygmens testavimo vienetas, kuris aprašo naudotojo veiksmų seką arba situaciją, kurią reikia patikrinti sistemoje.

Testavimo atvejis – konkretus, struktūruotas testavimo vienetas, kuris apibrėžia pradines sąlygas, įvestis, vykdomus veiksmus, laukiamus rezultatus ir parodo, kaip tiksliai atliekamas testavimas.

Įvadas

Šis darbas priklauso studijų programai „Veiklos skaitmeninimas ir sistemų architektūros“.

Programinės įrangos kūrimas yra sudėtingas procesas, apimantis projektavimo, analizės, programavimo ir testavimo etapus. Kiekvienas iš šių etapų yra svarbus siekiant užtikrinti programinės įrangos kokybę ir sistemos atitiktį naudotojų poreikiams. Vienas iš didžiausių iššūkių išlieka efektyvus ir patikimas testavimas, leidžiantis užtikrinti, kad sistema veiktų pagal apibrėžtas specifikacijas ir atitiktų verslo reikalavimus. Daugelyje projektų, kuriuose naudojamos detalios verslo procesų specifikacijos ir sudėtinga verslo logika, testavimo atvejų kūrimas vis dar atliekamas rankiniu būdu, nes šis procesas dažnai apima didelės apimties reikalavimų analizę bei scenarijų sudarymą pagal projektinę dokumentaciją. Tokia praktika reikalauja daug laiko ir žmogiškųjų išteklių, o dėl daugybės galimų naudotojo sąveikų ir sistemos reakcijų ne visada pavyksta užtikrinti pakankamą testavimo padengimą. Dėl šios priežasties rankinių testavimo atvejų kūrimo procesas dideliuose ir sudėtinguose programinės įrangos projektuose tampa vis sudėtingesnis ir reikalaujantis daugiau laiko, todėl IT įmonės vis aktyviau ieško būdų automatizuoti testavimo atvejų sudarymą bei palengvinti testuotojų darbą. Automatizuotos priemonės leidžia greičiau generuoti testavimo scenarijus bei sumažinti žmogiškųjų išteklių poreikį. Vienas iš galimų sprendimų – UML panaudojimo atvejų diagramų ir jų specifikacijų panaudojimas testavimo atvejams generuoti, kadangi šie modeliai leidžia apibrėžti sistemos funkcionalumą ir naudotojų sąveikas su sistema. Visgi daugelis šiuo metu egzistuojančių įrankių turi ribotas galimybes generuoti testavimo atvejus iš UML modelių, todėl ši sritis išlieka aktualiu tyrimo ir tobulinimo objektu. Esami sprendimai dažnai reikalauja sudėtingų konfigūracijų ir papildomų techninių žinių, todėl jų integravimas į organizacijų naudojamus testavimo procesus yra apsunkintas. Dėl šių priežasčių išlieka poreikis kurti lankstesnius ir praktiškai pritaikomus sprendimus, leidžiančius generuoti testavimo atvejus pagal UML panaudojimo atvejų diagramas ir jų specifikacijas bei prisidėti prie efektyvesnio programinės įrangos testavimo proceso.

Darbo tikslas ir uždaviniai

Tyrimo tikslas – palengvinti programinės įrangos testavimo procesą, automatizuojant testavimo atvejų kūrimą pagal UML panaudojimo atvejų diagramas ir jų specifikacijas.

Tyrimo uždaviniai:

1. Išanalizuoti UML panaudojimo atvejų diagramas ir jų specifikacijas testavimo atvejų generavimo kontekste.
2. Atlikti testavimo atvejų, jų struktūros ir sudarymo principų analizę.
3. Ištirti įrankius ir technologijas, skirtus testavimo atvejų generavimui iš UML diagramų.
4. Pasiūlyti naują metodą, skirtą automatizuoti rankinių testavimo atvejų kūrimą pagal UML diagramas ir jų specifikacijas.
5. Realizuoti įrankį, kuris pagal pasiūlytą metodą generuotų rankinius testavimo atvejus iš UML panaudojimo atvejų diagramų ir jų specifikacijų.
6. Eksperimentiškai ištirti pasiūlyto metodo bei įrankio efektyvumą testavimo atvejų generavimo kontekste ir apibendrinti eksperimentinio tyrimo rezultatus.

Darbo rezultatai ir jų svarba

Darbo metu buvo sukurta testavimo atvejų generavimo metodika ir ją realizuojantis programinis įrankis, leidžiantis automatizuotu būdu generuoti testavimo atvejus pagal UML panaudojimo atvejų diagramas ir jų specifikacijas. Sukurtas sprendimas analizuoja pateiktus panaudojimo atvejų duomenis, identifikuoja naudotojų veiksmus, sistemos atsakus, pagal nustatytas generavimo taisykles suformuoja testavimo scenarijus. Įrankis taip pat leidžia atlikti pirminę specifikacijų analizę, nustatyti trūkstamą informaciją bei eksportuoti sugeneruotus testavimo atvejus į struktūrizuotą rezultatų failą.

Darbo rezultatai yra aktualūs projektuose, kuriuose naudojamos standartizuotos UML panaudojimo atvejų specifikacijos, o testavimo procesas reikalauja daug pasikartojančio rankinio darbo. Tokiose sistemose testavimo atvejų rengimas dažnai tampa ilgu ir daug žmogiškųjų išteklių reikalaujančiu procesu, todėl automatizuotas generavimas gali padėti sumažinti laiko sąnaudas testavimo atvejų rengimo procese.

Generatyvinio dirbtinio intelekto naudojimas

Rengiant baigiamąjį projektą buvo naudojami šie generatyvinio dirbtinio intelekto (GDI) įrankiai:

- Teksto struktūrizavimui, formuluočių tikslinimui bei idėjų generavimui atliekant lyginamąją analizę buvo naudojamas ChatGPT (OpenAI GPT-5.5).
- Programinio kodo fragmentų generavimui, logikos realizavimo idėjų paieškai bei techninių problemų sprendimo pasiūlymams buvo taikomas ChatGPT (OpenAI GPT-5.5).
- Siekiant įvertinti, ar generatyvinio dirbtinio intelekto įrankis gali būti naudojamas kaip esamas testavimo atvejų generavimo sprendimas, buvo naudojamas ChatGPT (OpenAI GPT-5.5). Vertinimo metu buvo tikrinamos kelių rūšių užklausos su prašymu generuoti testavimo atvejus pagal pateiktą šabloną (detalesnį aprašymą galima rasti skyriuje Testavimo atvejų generavimo metodikos išvesties duomenų modelis, šablonas ir taisyklės).
- Kuriamo įrankio funkcijoms realizuoti buvo naudojama OpenAI API (GPT-4.1-mini):
    - Pirminė analizė įkėlus panaudojimo atvejų specifikacijos failą – panaudojimo atvejų nuskaitymas iš paveiksliuko, panaudojimo atvejų nuskaitymas iš lentelių, panaudojimo atvejų sulyginimas, ar diagramoje panaudojimo atvejų skaičius lygus lentelėse rastų panaudojimo atvejų skaičiui.
    - Antroji analizė – atitinkamų požymių sužymėjimas kiekvienam rastam panaudojimo atvejui.

Šio projekto autorė nuodugniai įvertino, suprato ir pagal poreikį redagavo GDI sugeneruotą turinį bei prisiima dėl jo pilną atsakomybę.

Darbo struktūra

Magistro baigiamasis darbas sudarytas iš penkių skyrių, literatūros ir priedų sąrašų. Įvade pristatoma darbo problematika, aktualumas, tikslas, uždaviniai ir rezultatai. Probleminės srities analizės skyriuje nagrinėjama rankinio testavimo atvejų kūrimo problematika, UML panaudojimo atvejų diagramų ir jų specifikacijų taikymas testavimo procese bei analizuojami esami testavimo atvejų generavimo sprendimai ir jų galimybės. Metodikos reikalavimų specifikacijos formalizuoto aprašo skyriuje pateikiami kuriamo testavimo atvejų generavimo

įrankio funkciniai reikalavimai, duomenų modelis ir nefunkciniai reikalavimai. Realizacijos projekto skyriuje aprašomas įrankio sąsajos prototipas bei loginė architektūra. Sprendimo realizacijos ir testavimo skyriuje pateikiamas realizuoto įrankio veikimo aprašas, testavimo duomenys bei rezultatai. Eksperimentinio tyrimo skyriuje pristatomas įrankio vertinimas naudojant realių projektų UML panaudojimo atvejų specifikacijas, sugeneruotų ir rankiniu būdu sudarytų testavimo atvejų palyginimas, apklausos rezultatų analizė bei sprendimo taikymo rekomendacijos. Darbo pabaigoje pateikiamos galutinės išvados.

1. Probleminės srities analizė

1.1. Analizės tikslas

Šiame darbe atliekamos analizės tikslas – ištirti UML panaudojimo atvejų diagramų ir jų specifikacijų taikymo galimybes automatizuotame testavimo atvejų generavime, siekiant pagrįsti naujos metodikos kūrimą. Tai bus pasiekta išanalizuojant:

- UML panaudojimo atvejų diagramas, jų specifikacijas ir panaudojamumą testavimo atvejų kūrimo procese;
- testavimo atvejų sudarymo principus ir metodus;
- egzistuojančius įrankius, skirtus testavimo atvejų generavimui pagal UML modelius ir jų galimybes;
- UML specifikacijų pritaikymo galimybes, siekiant užtikrinti tikslesnį testavimo atvejų generavimą;

1.2. Tyrimo objektas, sritis ir problema

Tyrimo objektas – automatizuotas testavimo atvejų sudarymas. Tyrimo srityje analizuojamos UML panaudojimo atvejų diagramos, jų specifikacijos, esamos testavimo atvejų generavimo metodikos, įrankiai bei jų pritaikomumas testavimo atvejų kūrimo kontekste. Atlikta esamų sprendimų analizė parodė, kad daugelis testavimo atvejų generavimo įrankių naudoja kitas UML diagramas, pavyzdžiui, veiklos ar sekų diagramas, o UML panaudojimo atvejų diagramas dažniausiai taiko tik kaip papildomą informacijos šaltinį. Taip pat nustatyta, kad esami sprendimai nevertina kartu naudojamų panaudojimo atvejų specifikacijų lentelių turinio ir jose aprašytų scenarijų, todėl jų pritaikymas detalių rankinių testavimo atvejų generavimui yra ribotas. Dėl šios priežasties išlieka poreikis kurti sprendimus, leidžiančius panaudoti UML panaudojimo atvejų diagramas ir jų specifikacijas testavimo atvejų kūrimo procese.

1.3. Automatizuoto testavimo atvejų sudarymo analizė

1.3.1. Testavimo atvejų analizė

Testavimo atvejai yra scenarijai, kuriuose tikrinama, kaip sistema reaguoja į tam tikras įvestis ar sąlygas. Kiekviename testavimo atvejyje apibrėžiamos išankstinės sąlygos, veiksmo atlikimo žingsniai, įvedami duomenys ir laukiamas rezultatas [1]. Šie atvejai apima ne tik funkcinius sistemos reikalavimus, bet ir patikimumo, našumo bei saugumo aspektus, siekiant užtikrinti visapusišką programinės įrangos kokybę. Kokybės užtikrinimas (angl. Quality Assurance, QA) yra vienas pagrindinių testavimo atvejų kūrimo tikslų, nes jie padeda nustatyti trūkumus ankstyvose programinės įrangos kūrimo stadijose ir išvengti galimų klaidų diegimo metu [2].

Testavimo atvejai yra būtini norint užtikrinti, kad sistema veiktų pagal sudarytą reikalavimų specifikaciją ir naudotojo norus. Testavimo atvejų kūrimas leidžia ankstyvoje stadijoje aptikti klaidas, kurios galėtų sukelti rimtų problemų programinės įrangos diegimo ir palaikymo metu. Taip sumažinamos klaidų taisymo išlaidos bei projekto rizikos. Be to, QA procesai integruoja testavimo atvejus kaip svarbiausią priemonę užtikrinti sistemų stabilumą ir veikimą pagal specifikacijas [2].

Praktikoje testavimo atvejai gali būti kuriami tiek rankiniu, tiek automatizuotu būdu. Automatinis testavimas dažniausiai taikomas pasikartojantiems, aiškiai apibrėžtiems ir dažnai vykdomiems scenarijams, nes leidžia sumažinti testavimo laiką bei žmogiškųjų klaidų tikimybę. Tačiau rankinis testavimas vis dar plačiai naudojamas situacijose, kai reikia vertinti sudėtingesnę verslo logiką, naudotojo patirtį ar nevisiškai formalizuotus reikalavimus. Tokiais atvejais testuotojo interpretacija ir konteksto supratimas išlieka svarbūs, todėl rankiniu būdu kuriami testavimo atvejai vis dar yra reikšminga programinės įrangos testavimo proceso dalis.

Testavimo atvejų tikslas yra užtikrinti, kad sistema atitinka funkcinius ir nefunkcinius reikalavimus, identifikuoti galimas klaidas, patikrinti kodo efektyvumą ir saugumą bei patikrinti, ar sistema elgiasi pagal numatytus reikalavimus [1]. Kiekvienas testavimo atvejis turi būti parengtas taip, kad jį būtų galima tiksliai atkartoti.

Testavimo atvejis dažniausiai susideda iš kelių pagrindinių komponentų: įvesties duomenys, veiksmo atlikimo žingsniai, laukiamas rezultatas, gautas rezultatas ir testavimo aplinka. Papildomai gali būti įtraukti tokie elementai kaip prioritetas (testo svarbos lygis), susiję reikalavimai (tikrinamos funkcijos) bei statusas (pavyzdžiui, „sėkmingas“, „klaidingas“ arba „vykdomas“). Tai leidžia sistemingai analizuoti ir vertinti, ar sistema atitinka lūkesčius [1]

Testavimo atvejų struktūra ir tikslas gali kisti priklausomai nuo testavimo tipo, tačiau jų pagrindinis tikslas yra užtikrinti programinės įrangos kokybę ir patikimumą [1]. Šis procesas glaudžiai susijęs su *QA* metodais, kurie užtikrina, kad sistemos reikalavimai atitinka naudotojo poreikius bei programinės įrangos specifikaciją [2].

1.3.2. Testavimo atvejų sudarymo metodai

1.3.2.1. Ekvivalentiškumo klasių skaidymo (angl. *Equivalence Partitioning*) metodas

Ekvivalentiškumo klasių skaidymas yra viena iš populiariausių metodikų, naudojamų testavimo atvejams sudarinėti [3]. Ši metodika remiasi idėja, kad testuojama sistema turi įvesties duomenis, kurie gali būti suskaidomi į skirtingas klases, o pastarųjų elgsena yra vienoda. Tai leidžia sumažinti reikiamų sukurti testavimo atvejų skaičių, kadangi nėra būtina tikrinti kiekvieną galimą įvestį, jei ji priklauso tai pačiai klasei [3]. Ekvivalentiškumo klasių skaidymas padeda identifikuoti galimus sistemos atsakus į skirtingas įvestis ir apima tiek teigiamus, tiek neigiamus atvejus. Klasės gali būti skirstomos į dvi rūšis:

1. Teigiamosios – tikimasi, kad sistema veiks pagal numatytąsias specifikacijas.
2. Neigiamosios – pateikiama neteisinga įvestis ir tikimasi gauti klaidą.

Šis metodas dažnai naudojamas vertinti riboms, kai reikia patikrinti kraštutinumus. Ekvivalentiškumo klasių skaidymas yra populiari metodika dėl paprastumo ir gebėjimo sutrumpinti testavimo atvejų kūrimo laiką, nesumažinant testavimo apimties [4].

Paveikslėlyje žemiau (žr. 1 pav.) pateikiamas ekvivalentiškumo testavimo atvejo pavyzdys, kuriame galima matyti veiksnius: paleidimo objektas, objekto tipas, žymeklio identifikatorius ir kitus, kurie turi įtakos atliekamam veiksmui [5]. Kiekvienas iš šių veiksnių yra suskirstytas į ekvivalentiškas klases, kurios apima skirtingus galimus scenarijus, leidžiančius sukurti testavimo atvejus pagal įvairius sistemos įvesties duomenis ir veiksmus.

ID	Launcher	Object Type	Marker Identifier	Object Count	Action Result
1	AA/ACA/V360 Routing portlet/V360 Inspection list	Cap/work order/Inspection /Parcel/GIS feature/Asset	Purple box with digital filling	Maximum	Display marker/Show routing/Display object' summary info

Action After Initial Routing	Action on Map	Action on Contents Panel	Selected Routing Method	Action Result (Mouse Over)	Action on Routing Panel
Add object	Add marker	Add object	Click routing button	N/A	N/A

Result	System Response Speed
Add objects information to the routing panel list	Lot slowly but does not halted

1 pav. Pavyzdinis testavimo atvejis sudarytas pagal ekvivalentiškumo metodą [5]

Naudojant šį metodą, galima sukurti atvejus, kurie apima kraštutines situacijas, tokias kaip didelė duomenų apimtis ar didelis naudotojų skaičius, kurie gali sukelti sistemos našumo problemas [4]. Tai padeda ne tik patikrinti, ar sistema teisingai veikia pagal numatytus scenarijus, bet ir simuliuoti galimus realaus pasaulio iššūkius [5]. Išsamus testavimo atvejų aprašymas, įskaitant tokius veiksmus kaip objekto pridėjimas, maršruto pasirinkimas ir sistemos atsakas, leidžia tiksliai įvertinti, kaip sistema reaguoja į skirtingas sąlygas ir užtikrinti jos patikimumą [5].

Ekvivalentiškumo klasių skaidymo metodo analizė svarbi todėl, kad šis metodas parodo, kaip testavimo atvejai gali būti sudaromi remiantis iš anksto apibrėžtomis taisyklėmis ir pasikartojančia logika [5]. Tokie principai aktualūs ir automatizuotam testavimo atvejų generavimui, nes leidžia grupuoti panašius scenarijus ir sumažinti rankiniu būdu kuriamų testavimo atvejų skaičių [5].

1.3.2.2. Panaudojimo atvejų analizės (angl. *Use Case Analysis*) metodas

UML panaudojimo atvejų diagramos ir jų specifikacijos atlieka svarbų vaidmenį testavimo atvejų kūrimo procese [6]. Panaudojimo atvejų diagramos ir jų specifikacijos pateikia aiškų ir suprantamą sistemos funkcionalumo aprašą, kuris padeda testavimo specialistams identifikuoti ir dokumentuoti pagrindinius scenarijus [7]. Kadangi kiekvienas panaudojimo atvejis atitinka konkrečią funkciją ar procesą, kurį sistema atlieka – tai tiesiogiai susiję su testavimo atvejais ir jų vykdymu pažingsniui [8].

Šios diagramos padeda lengviau nustatyti pagrindinius ir alternatyvius srautus, taip pat galimus klaidų scenarijus [9]. Testuotojai gali remtis panaudojimo atvejų specifikacijomis, kad sukurtų testavimo atvejus, kurie atspindi tiek sėkmingą sistemos veikimą, tiek išimtinius atvejus [7].

Panaudojimo atvejų diagramos ir specifikacijos yra ypač naudingos, kai reikia bendradarbiauti tarp skirtingų komandos narių. Jos užtikrina, kad visi dalyviai – nuo kūrėjų

iki testuotojų ir projektų vadovų – turėtų vienodą supratimą apie sistemos funkcijas[9]. Funkcinių testavimo atvejų sudarymas remiantis šiais aprašais leidžia ne tik patikrinti, ar sistema atitinka reikalavimus, bet ir pagerina kokybę bei sumažina galimų klaidų riziką [10].

Be to, panaudojimo atvejų analizė gali būti naudojama kartu su kitais testavimo metodais, pavyzdžiui, regresinio ar apkrovos testavimo atvejais, užtikrinant dar didesnę testavimo apimtį [7]. Taip testavimo procesas tampa visapusiškesnis, nes dėmesys sutelkiamas į tai, kaip sistema turėtų veikti realiose situacijose, kas yra itin svarbu naudotojo patirčiai ir sistemų patikimumui [6]. Žemiau pateiktas pavyzdinis panaudojimo atvejo aprašas (žr. 2 pav.), kuriame aprašyta anksčiau paminėta svarbi informacija apie tam tikrą funkcionalumą, t.y. aktorius, išankstinės sąlygos, rezultatai, detalūs žingsniai, alternatyvūs scenarijai ir kita.

Use Case Definition

Last Updated By:	Last Updated On:
Use Case Name: Withdraw From Checking	UC ID: ATM-01
Actor: Bank Customer	
Objective: To allow a bank customer to obtain cash and have the withdrawal taken from their checking account.	
Preconditions: Bank customer must have an ATM cash card, valid account, valid PIN and their available checking account balance must be greater than, or equal to, withdrawal amount. ATM in idle mode with greeting displayed (main menu).	
Results (Postconditions): The cash amount dispensed must be equal to the withdrawal amount. The ATM must print a receipt and eject the cash card. The checking account is debited by amount dispensed.	

Detailed Description

Action	Model (System) Response
1. Customer inserts ATM cash Card.	1. ATM reads cash card and prompts customer to enter PIN.
2. Customer enters PIN.	2. ATM validates PIN and displays menu with a list of transactions that can be selected.
3. Customer selects Withdraw From Checking transacton.	3. ATM validates account and prompts customer for withdrawal amount.
4. Customer enters withdrawal amount.	4. ATM validates account balance is greater than, or equal to, withdrawal amount. ATM dispenses cash equal to withdrawal amount and prompts customer to take cash.
5. Customer takes cash.	5. ATM asks customer whether they want to continue.
6. Customer indicates not to continue.	6. ATM prints receipt, ejects cash, prompts customer to take card, sends debit message to ATM Control System, returns to idle mode and displays main menu.
7. Customer takes card and receipt.	

Exceptions:
If ATM cannot read cash card, then ATM ejects cash card.
If incorrect PIN is entered, then customer is given two additional chances to enter correct PIN.
If correct PIN not entered on third try, then ATM keeps cash card and informs customer that they must retrieve card from bank personnel during business hours.
If account is not valid, ATM ejects card and informs customer that they must contact bank personnel during business hours regarding their invalid account.
If account balance is less than withdrawal amount, ATM informs customer that the withdrawal amount exceeds their account balance and to reenter a withdrawal amount that does not exceed account balance. If amount reentered still exceeds account balance, ATM ejects card, informs customer that amount requested still exceeds account balance and bank policy does not permit exceptions.

Alternative Courses:
At any time after reaching the main menu and before finishing a transaction, including before selecting a transaction, the customer may press the cancel key. If the cancel key is pressed, the specified transaction (if there is one) is canceled, the customer's cash card is returned, the ATM returns to idle mode and the main menu is displayed.

Original Author: Larry Creel	Original Date: 9-25-X

2 pav. Pavyzdinis, detalus panaudojimo atvejo aprašymas, pagal kurį yra generuojami testavimo atvejai [6]

Pagal pateiktą pavyzdinį panaudojimo atvejo aprašą galima sudaryti detalų testavimo atvejį (žr. 3 pav.) [6]. Sudarytame testavimo atvejyje bus panaudota informacija iš panaudojimo atvejo: pavadinimas, pagrindinio scenarijaus detalūs žingsniai, laukiami rezultatai bei alternatyvūs scenarijai [6]. Taip pat prisideda papildoma informacija: testavimo tikslas, įvedami duomenys ir kita [6].

Test Case Worksheet						
Test Case ID: T-ATM-01			Original Author: Larry Creel		Last Updated By:	
Parent Use Case ID: ATM-01			Original Date: 9-26-XX		Last Updated On:	
Test Objective: To test the function *Withdraw From Checking*, the associated exceptions and alternative courses.						
ITEM NO.	**TEST CONDITION**	**OPERATOR ACTION**	**INPUT SPECIFICATIONS**	**OUTPUT SPECIFICATIONS (EXPECTED RESULTS)**	**PASS OR FAIL**	**COMMENTS**
1	Successful withdrawal.	1-Insert card. 2-Enter PIN. 3-Select Withdraw From Checking transaction. 4-Enter withdrawal amount. 5-Take cash. 6-Indicate not to continue. 7-Take card and receipt.	1-ATM can read card. 2-Valid account. 3-Valid PIN. 4-Account balance greater than, or equal to, withdrawal amount.	1-ATM reads card and prompts customer to enter PIN. 2-ATM validates PIN and displays menu with a list of transactions that can be selected. 3-ATM validates account and prompts customer to enter withdrawal amount. 4-ATM validates account balance greater than, or equal to, withdrawal amount. ATM dispenses cash equal to withdrawal amount and prompts customer to take cash. 5-ATM asks customer whether they want to continue. 6-ATM prints receipt, ejects cash card, prompts customer to take card, sends debit message to ATM Control System. ATM returns to idle mode and displays Main Menu.		Re-execute test and use the Continue option Verify correct debit message received by ATM Control System.
2	Unsuccessful withdrawal due to unreadable card.	1-Insert card. 2-Take card.	1-ATM cannot read card. 2-Valid account. 3-Valid PIN. 4-Account balance greater than or equal to, withdrawal amount.	1-ATM ejects card, prompts customer to take card and displays message "Cash Card unreadable. Please contact bank personnel during business hours." ATM returns to idle mode and displays Main Menu.		

3 pav. Pavyzdinis, detalus testavimo atvejis, sudarytas pagal aukščiau esantį panaudojimo atvejį (žr. 2 pav.) [6]

Iš pateiktų paveikslėlių galima matyti, jog testavimo atvejis sudaromas tokiu pat tikslumu kaip ir panaudojimo atvejis. Tokiu būdu užtikrinama, jog nei vienas žingsnis nebus praleistas sistemos testavimo metu.

Panaudojimo atvejų analizė yra aktuali, kad UML panaudojimo atvejų diagramose ir jų specifikacijose pateikiama informacija tiesiogiai susijusi su testavimo scenarijų sudarymu [6]. Panaudojimo atvejų žingsniai, alternatyvūs scenarijai, naudotojų teisės ir laukiami rezultatai gali būti naudojami kaip pagrindas automatizuotam testavimo atvejų generavimui [10].

1.3.2.3. Ribinių reikšmių analizės (angl. *Boundary Value Analysis*) metodas

Šis metodas yra vienas iš pagrindinių baltosios dėžės testavimo metodų, orientuotas į įvesties duomenų intervalų ribas [11]. Pagrindinis šio metodo tikslas – aptikti klaidas, kurios dažniausiai atsiranda ribiniuose sistemos taškuose, kai įvesties duomenys artėja prie minimalių arba maksimalių leidžiamų reikšmių [12]. Šis metodas ypač naudingas sistemose, turinčiose skaitines įvestis arba kitokius apribojimus, pavyzdžiui, simbolių skaičių ar duomenų dydį [13].

Ribinės reikšmės testavime dažniausiai pasirenkamos žemiau išvardintos ribos:

- Prieš ribą (pvz., minimalus duomuo - 1).
- Ant ribos (pvz., minimalus duomuo ar maksimalus duomuo).
- Už ribos (pvz., maksimalus duomuo + 1).

Pavyzdžiui, jeigu sistema priima reikšmes nuo 10 iki 100, testavimo atvejai bus sudaromi šioms reikšmėms: 9 (žemiau ribos), 10 (ant ribos), 11 (virš ribos), 99 (prie ribos), 100 (ant ribos), 101 (už ribos). Šis metodas taip pat taikomas ne tik skaičiams, bet ir sąrašams bei kitoms struktūroms, kur galima identifikuoti ribas [14].

Šis metodas leidžia:

- Efektyviai aptikti dažniausiai pasitaikančias ribines klaidas [11].
- Sumažinti testavimo apimtį, nes orientuojamasi į pagrindinius taškus [13].

Ribinių reikšmių analizės metodas parodo, kad dalis testavimo atvejų sudaromi pagal aiškiai apibrėžtas sąlygas ir ribas [13]. Tokie scenarijai dažnai yra pasikartojantys, todėl jų sudarymas gali būti automatizuojamas naudojant iš anksto nustatytas taisykles [14].

1.3.2.4. Testavimo atvejų sudarymo metodų analizės apibendrinimas

Atlikta testavimo atvejų sudarymo metodų analizė parodė, kad skirtingi metodai orientuojasi į skirtingus sistemos tikrinimo aspektus – įvesties duomenų grupavimą, ribinių reikšmių tikrinimą, naudotojo veiksmų žingsnius ar tekstinių reikalavimų analizę [7]. Tačiau daugeliu atvejų testavimo scenarijų sudarymas remiasi aiškiai aprašyta ir struktūrizuota informacija apie sistemos veikimą [15]. Analizė taip pat parodė, kad UML panaudojimo atvejų specifikacijos gali būti svarbus informacijos šaltinis automatizuojant testavimo atvejų sudarymą, nes jose aprašomi naudotojo žingsniai, sistemos atsakai, alternatyvūs scenarijai ir kiti testavimui reikalingi elementai [7].

1.4. UML panaudojimo atvejų diagramos ir jų specifikacijos testavimo atvejų generavimo kontekste

1.4.1. UML panaudojimo atvejų diagrama

Vieninga modeliavimo kalba – UML, kurią sukūrė ir standartizavo OMG (*Object Management Group*), yra plačiai naudojama vizualiam programinės įrangos projektavimui ir sistemų modeliavimui [16]. UML panaudojimo atvejų diagrama yra vienas iš pagrindinių ir populiariausių UML diagramų tipų, kuri yra skirta vaizduoti sistemų funkcinius reikalavimus. Ši diagrama parodo, kokius veiksmus sistema turi atlikti, kad naudotojo poreikiai būtų patenkinti . Programinės įrangos kūrimo procesuose panaudojimo atvejų diagrama taip pat plačiai naudojama siekiant užtikrinti, kad visi suinteresuoti asmenys suprastų sistemos funkcionalumą [17].

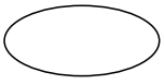
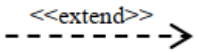
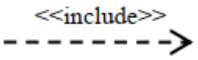
Panaudojimo atvejų diagrama susideda iš kelių elementų:

- Aktoriai – žmonės, sistemos arba išoriniai procesai, kurie sąveikauja su kuriama sistema.
- Panaudojimo atvejai – funkcijos, kurias turi vykdyti sistema.
- Asociacijos – jungtys tarp aktorių ir panaudojimo atvejų, kurios nurodo, kaip jie sąveikauja.

Taip pat diagramoje neretai naudojami specialūs ryšiai – *extend* ar *include* bei asociacijų kardinalumo reikšmės. *Extend* ryšys yra naudojamas, kai vienas panaudojimo atvejis gali būti išplėstas kitu atveju. Šis ryšys dažniausiai naudojamas sąlyginių funkcijų modeliavimui.

Include ryšys parodo, kad vieno panaudojimo atvejo įvykdymas visada apima kito atvejo vykdymą [17]. Išvardintų elementų žymėjimas vaizdiškai pateikiamas lentelėje (žr. 1 lentelė).

1 lentelė. Panaudojimo atvejų diagramos elementų žymėjimas

Elemento pavadinimas	**Elemento žymėjimas PA diagramoje**
Aktorius	
Panaudojimo atvejis	
Asociacija	
Extend ryšys	
Include ryšys	

UML panaudojimo atvejų diagramos leidžia vizualiai pavaizduoti sistemos funkcionalumus, naudotojų atliekamus veiksmus bei jų sąveiką su sistema, todėl šios diagramos plačiai naudojamos tiek mažų, tiek sudėtingų sistemų reikalavimų analizei atlikti. Naudojant panaudojimo atvejų diagramas, galima vizualiai pavaizduoti pagrindinius sistemos reikalavimus, nustatyti ryšius tarp aktorių ir jų veiklų [17].

UML panaudojimo atvejų diagramos testavimo kontekste svarbios dėl jose pateikiamų pagrindinių sistemos funkcionalumų, aktorių bei jų sąveikų su sistema [18]. Tokia informacija leidžia lengviau identifikuoti, kokie scenarijai turi būti testuojami, o diagrama suteikia galimybę šią informaciją analizuoti automatizuotai [18].

1.4.2. UML panaudojimo atvejų specifikacijos

UML panaudojimo atvejo specifikacijos naudojamos detaliau aprašyti kiekvieną panaudojimo atvejį bei pateikti papildomą informaciją apie sistemos atliekamus veiksmus. Specifikacija suteikia išsamų aprašymą, kaip sistema turi elgtis, kai vykdomas atitinkamas panaudojimo atvejis [19].

Panaudojimo atvejo specifikacija apima daug informacijos apie vieną panaudojimo atvejį, pagrindinė informacija, kuri turėtų būti aprašoma kiekviename atvejyje [16]:

1. Tikslas – nurodoma, koks norimas pasiekti rezultatas atliekant atitinkamą panaudojimo atvejį. [7]
2. Aktoriai – nurodomi, kokie aktoriai turi turėti galimybę atlikti atitinkamą scenarijų. [18]
3. Išankstinės sąlygos – nurodoma, kokių reikia pradinių sąlygų, kad atitinkamas scenarijus galėtų būti vykdomas. [7]
4. Pagrindinis scenarijus – nurodomi pagrindinio scenarijaus žingsniai, dažniausiai – sėkmingo scenarijaus. [7]
5. Alternatyvusis scenarijus – nurodomi alternatyvaus scenarijaus žingsniai, pavyzdžiui, klaidos. [18]

Kiekvienas išvardintas punktas panaudojimo atvejo specifikacijoje turėtų būti aprašomas logišku eiliškumu, panaudojimo atvejo specifikacijos pavyzdys atvaizduojamas žemiau pateiktame paveikslėlyje (žr. 4 pav.) [10].

USE CASE	<identification of the use case>
SUMMARY	<brief description of the use case>
ACTOR	<name of actor activating the use case>
PRECONDITION	<prior condition that must be met for the use case to be executed>
DESCRIPTION	**Basic Path** <use case> INITIATES WHEN<actor><activity> 1) [<precondition>]<actor> | <entity><activity>[<restriction>] 2) [<precondition>]<actor> | <entity><activity>[<restriction>] ..... N) [<precondition>]<actor> | <entity><activity>[<restriction>] END <use case> **Alternative Paths** <*Event*> 1) <actor> | <entity><activity>[<restriction>] 2) <actor> | <entity><activity>[<restriction>] .....
OSTCONDITION	<condition that must be met when the execution of the use case culminates>

4 pav. Panaudojimo atvejo specifikacijos aprašymo šablonas [10]

Jeigu panaudojimo atvejo specifikacijoje nėra aiškiai aprašyti naudotojo veiksmai, išankstinės sąlygos, sistemos atsakai ar alternatyvūs scenarijai, tampa sudėtinga tiksliai nustatyti, kokie testavimo scenarijai turi būti sudaromi. Dėl šios priežasties testavimo atvejų sudarymui svarbu, kad panaudojimo atvejų specifikacija būtų pateikta nuosekliai ir struktūrizuotai.

Dažnai panaudojimo atvejų specifikacijos sudaromos pagal *Volere* šabloną, kuris suteikia aiškią ir struktūrizuotą formą reikalavimų aprašymui. Šis šablonas padeda tiksliai apibrėžti visus svarbius panaudojimo atvejo elementus, tokius kaip aktoriai, išankstinės sąlygos, pagrindiniai ir alternatyvūs scenarijai [20]. Naudojant *Volere*, galima užtikrinti, kad specifikacija būtų nuosekli ir išsami, taip palengvinant tolimesnius sistemos kūrimo etapus.

Volere šablonas yra plačiai naudojamas reikalavimų inžinerijoje, nes leidžia sukurti aiškų sistemos elgsenos aprašą, kuris suprantamas tiek analitikams, tiek kūrėjams ir testuotojams. Jo struktūra padeda išvengti dviprasmybių ir užtikrina, kad visi esminiai aspektai būtų tinkamai dokumentuoti [20]. Be to, šis metodas padeda standartizuoti reikalavimų aprašymą skirtinguose projektuose, todėl palengvina jų analizę ir valdymą.

Vienas svarbiausių *Volere* šablono aspektų yra tai, kad jis leidžia tiksliai nustatyti reikalavimų patvirtinimo kriterijus [20]. Tinkamai sudaryta panaudojimo atvejų specifikacija, paremta

Volere struktūra, padeda užtikrinti nuoseklų testavimo scenarijų kūrimą ir efektyvesnį programinės įrangos kokybės užtikrinimą.

Volere šablone kiekvienam panaudojimo atvejui gali būti pateikiama įvairi informacija ir vaizdiniai modeliai. Jame galima aprašyti tekstinį scenarijų, pateikti siužetinę schemą, žemo ar aukšto detalumo prototipą, oficialią panaudojimo atvejo specifikaciją su išimtimis ir alternatyvomis. Taip pat galima naudoti įvairius diagramų tipus, tokius kaip sekų, veiklų ar duomenų srautų diagramos, priklausomai nuo projekto poreikių [20].

Taip pat panaudojimo atvejų specifikacijų aprašymui dažnai naudojamas IEEE Software Requirements Specification (SRS) standartas [21]. Šis standartas skirtas struktūrizuotam programinės įrangos reikalavimų dokumentavimui ir apima funkcinius bei nefunkcinius reikalavimus, sistemos apribojimus ir bendrą sistemos aprašą [22]. Tačiau IEEE SRS standartas labiau orientuotas į bendrą reikalavimų dokumentavimą, todėl jame dažniausiai trūksta detalių scenarijų žingsnių ir naudotojo veiksmų sekų, kurios yra svarbios testavimo atvejų generavimui [21].

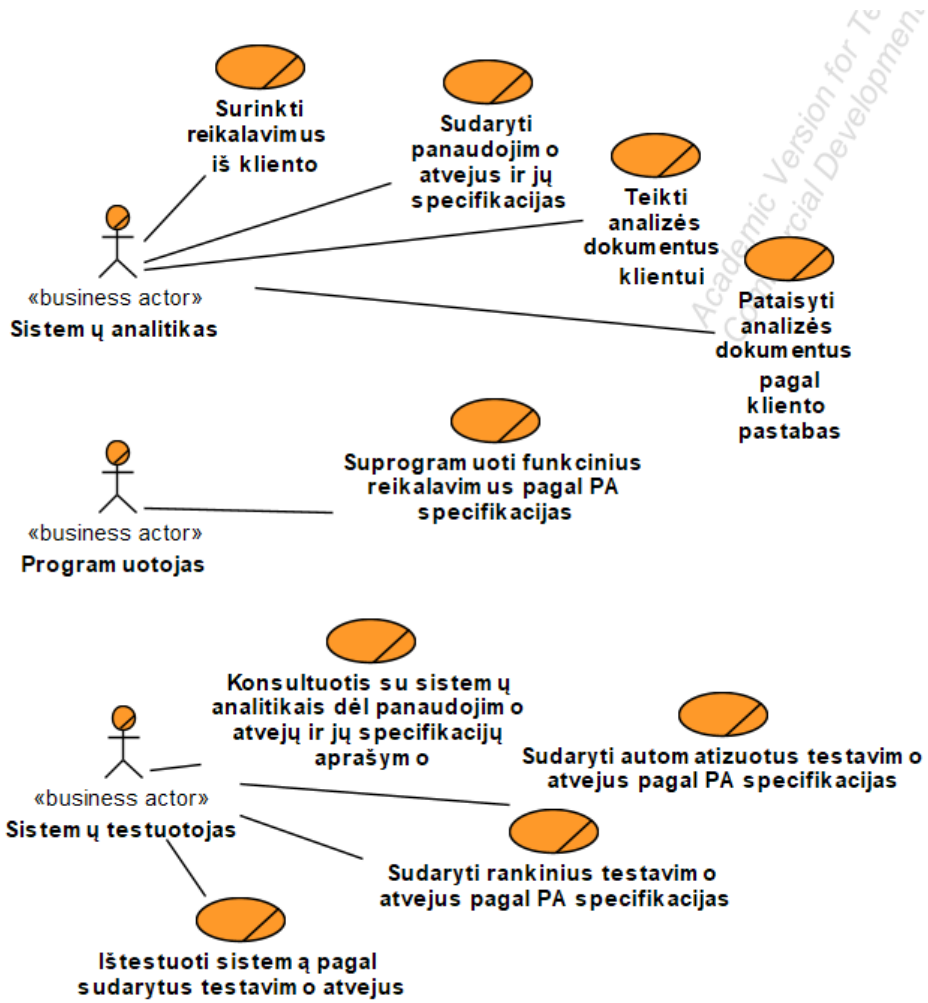
Kitas dažnai naudojamas panaudojimo atvejų aprašymo būdas yra Cockburn panaudojimo atvejų šablonas [23]. Šis šablonas orientuotas į pagrindinių ir alternatyviųjų scenarijų aprašymą, aktorius bei sistemos tikslus [23]. Cockburn šablonas leidžia aiškiai aprašyti sistemos elgseną skirtingose situacijose, tačiau specifikacijų detalumo lygis dažnai priklauso nuo projekto poreikių ir nėra griežtai standartizuotas [24]. Dėl šios priežasties skirtinguose projektuose pateikiama informacija gali skirtis savo struktūra ir detalumu [24].

Realiuose projektuose, panaudojimo atvejų specifikacijos dažnai naudojamos kaip komunikacijos priemonės tarp programinės įrangos kūrimo komandos narių (analitikų, programuotojų, testuotojų ir kitų). Naudodamiesi panaudojimo atvejų specifikacijomis galima visiems suprasti, kaip sistema iš tikrųjų turi veikti [7].

UML panaudojimo atvejų specifikacijos testavimo atvejų generavimo kontekste yra svarbios dėl jose detaliai aprašomų naudotojo veiksmų, sistemos atsakų, alternatyvių scenarijų ir išankstinių sąlygų [18]. Tokia struktūrizuota informacija gali būti naudojama kaip pagrindas automatizuotam testavimo atvejų sudarymui, nes specifikacijose pateikiami pagrindiniai naudotojo ir sistemos žingsniai, reikalingi testavimo scenarijams formuoti [6].

1.5. Tyrimo objekto naudotojų analizė

Programinės įrangos kūrimo procese yra daug svarbių aktorių, vieni iš jų – sistemų analitikai, programuotojai ir sistemų testuotojai [25]. Kiekvienas iš jų turi savo veiklas, kurias turi mokėti, suprasti bei laiku įvykdyti [25]. Pagrindinės išvardintų aktorių veiklos vizualiai vaizduojamos veiklos panaudojimo atvejų diagramoje (žr. 5 pav.).



5 pav. Naudotojų veiklos panaudojimo atvejų diagrama

Informacinių sistemų analitikai yra atsakingi už sistemos reikalavimų surinkimą ir dokumentavimą [26]. Jie glaudžiai bendradarbiauja su suinteresuotosiomis šalimis, kad tiksliai suprastų sistemos poreikius ir juos dokumentuotų į specifinius techninius reikalavimus. Dažniausiai sistemų analitikai naudoja UML panaudojimo atvejų, veiklos, sekų ir kitų rūšių diagrama, tačiau populiariausios yra panaudojimo atvejų diagramos. Šios diagramos suteikia galimybę modeliuoti sistemos funkcijas taip, kad sistemos funkcionalumas būtų aiškus ir lengvai suprantamas visiems sistemos kūrėjams bei naudotojams [26].

Sistemų analitikas ne tik kuria panaudojimo atvejų diagramas, bet ir pateikia išsamias jų specifikacijas, kurioje detalizuoja įvesties duomenis, veiksmų seką pažingsniui bei prie kiekvieno žingsnio pateikia laukiamą rezultatą [26]. Šie elementai yra susiję su testavimo atvejų sudarymu, nes jie suteikia testuotojams aiškų žingsnių planą, kurį reikia patikrinti bei aprašyti testavimo atvejuose. Taip pat, analitikai dažnai dalyvauja testavimo atvejų patikrinime, kurio metu peržiūrima, ar testavimo atvejai tinkamai atspindi panaudojimo atvejų specifikacijas [26].

Informacinių sistemų testuotojai naudoja sistemų analitikų sukurtus panaudojimų atvejus ir jų specifikacijas kaip pagrindą testavimo atvejų sudarymui ir vykdymui [27]. Testuotojai

analizuoja panaudojimo atvejus, jų specifikacijas, kad sukurtų išsamius testavimo atvejus kritinių klaidų aptikimui ankstyvose sistemos kūrimo fazėse. Testuotojai atlieka įvairių tipų testavimą – funkcinius, regresijos, saugumo ir kitus testus, toks platus testavimo planas padeda kuo tiksliau ištestuoti sistemą ir laiku aptikti klaidas. Kadangi panaudojimo atvejai yra orientuoti į galutinio naudotojo veiksmus, jie leidžia testuotojui sukurti realistiškus scenarijus, kuriuose tikrinama, ar sistema atitinka numatytus reikalavimus [27].

Abiejų šių specialistų – analitikų ir testuotojų – darbas yra glaudžiai susijęs su UML panaudojimo atvejais ir testavimo atvejais [27]. Sistemų analitiko pateikti panaudojimo atvejai yra pradinė medžiaga, kurią testuotojai naudoja kurdami testavimo scenarijus. Tai užtikrina, kad testavimas yra orientuotas į svarbiausias sistemos funkcijas, atspindinčias realius naudotojų poreikius [27].

Apibendrinant galima teigti, kad UML panaudojimo atvejų diagramos ir jų specifikacijos sudaro svarbią jungtį tarp sistemų analizės ir testavimo procesų. Sistemų analitikų parengta informacija leidžia testuotojams aiškiau suprasti sistemos funkcionalumą bei sudaryti testavimo atvejus pagal apibrėžtus reikalavimus. Dėl šios priežasties panaudojimo atvejų specifikacijos gali būti naudojamos ne tik sistemos analizės ir projektavimo metu, bet ir kaip pagrindas automatizuotam testavimo atvejų generavimui.

1.6. Testavimo atvejų generavimo esamų sprendimų analizė

Programinės įrangos kūrimo procesuose naudojami įvairūs sprendimai, skirti testavimo veiklų automatizavimui pagal sistemos reikalavimus, UML modelius ar panaudojimo atvejų specifikacijas [28]. Tokie sprendimai dažniausiai taikomi siekiant palengvinti testavimo atvejų sudarymą ir efektyviau valdyti testavimo procesą [28]. Analizuojant esamus sprendimus galima įvertinti, kokie metodai ir technologijos naudojami testavimo atvejų generavimui bei kokios jų pritaikymo galimybės UML panaudojimo atvejų kontekste [18].

1.6.1. *SpecExplorer* įrankio naudojimas testavimo atvejų generavimo kontekste

Tai yra *Microsoft Visual Studio* programinės įrangos papildinys, skirtas modeliu pagrįstam testavimui (angl. *Model-Based Testing*) [29]. Šis įrankis leidžia kurti programinės įrangos elgesio modelius, juos analizuoti bei automatiškai generuoti testavimo atvejus. *Spec Explorer* palaiko vieną iš populiariausių programavimo kalbų *C#*, todėl yra itin tinkamas kūrėjams, kurie dirba su *.NET* technologijomis [30, 31].

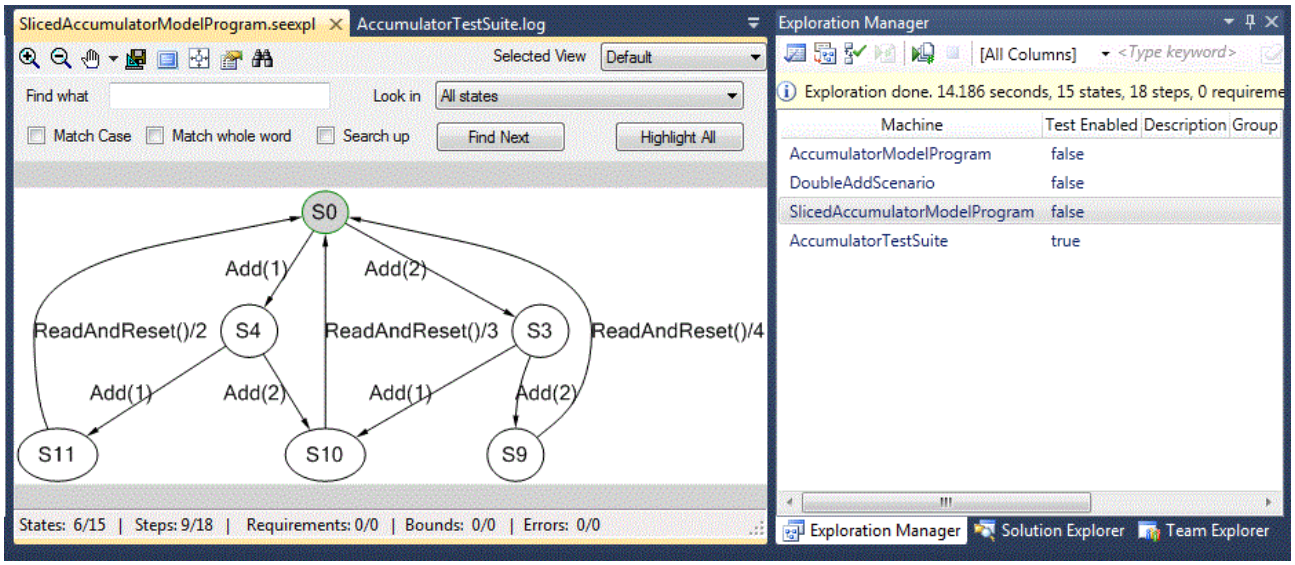
Pagrindinės *Spec Explorer* funkcijos apima modelių kūrimą ir grafinių modelių vizualizaciją [29]. Pavyzdžiui, kūrėjai ir testuotojai, naudodami šį įrankį, gali sukurti modelį, apibrėžiantį sistemos funkcionalumą, identifikuoti galimus sistemos veiksmus ir atsakymus, o tada automatiškai sugeneruoti testavimo atvejus, kurie apima tiek pagrindinius, tiek alternatyvius scenarijus. Ši funkcija sumažina rankinio darbo poreikį ir užtikrina, kad visi testavimo atvejai atitiktų specifikuotus modelius [31].

Vienas iš pagrindinių *Spec Explorer* privalumų – automatinis testavimo atvejų generavimas ir lengvai suprantama grafinė modelių vizualizacija [32]. Testuotojai gali naudotis šiais modeliais ne tik sistemų testavimui, bet ir komandos bendradarbiavimui – grafiškai pateikti modeliai padeda visiems komandos nariams geriau suprasti, kaip sistema turėtų veikti [30].

Tačiau *Spec Explorer* turi ir trūkumų. Šis įrankis nėra aktyviai vystomas, o jo palaikymas galimas iki *Windows 10* ir *Visual Studio* senesnių versijų [30]. Be to, *Spec Explorer* yra orientuotas tik į *C#* kalbą, todėl jo panaudojimas kitose platformose yra ribotas [32].

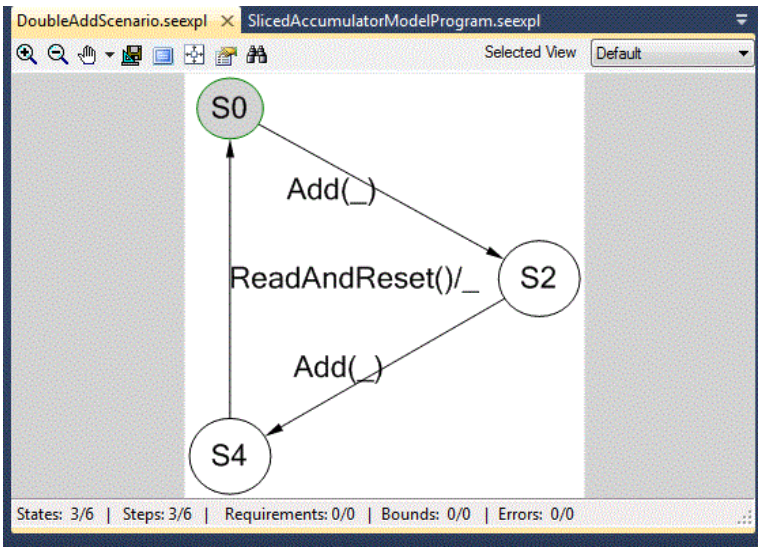
Nepaisant šių apribojimų, *Spec Explorer* išlieka vertingu įrankiu, leidžiančiu automatizuoti testavimo atvejų kūrimą ir efektyviai analizuoti sudėtingas sistemas [32]. Jis yra puikus pasirinkimas programuotojams ir testuotojams, siekiantiems užtikrinti programinės įrangos kokybę naudojant modeliais grįsto testavimo metodiką [30].

Pradedant naudotis *SpecExplorer* įrankiu, dažniausiai sukuriamas projektas, kuriame suformuojamas kuriamos ir testuojamos sistemos modelis. Žemiau pateiktame pavyzdyje (žr. 6 pav.) pateiktas fragmentas iš *SpecExplorer* įrankio, jame galima matyti suformuotą kuriamos sistemos modelį [33].



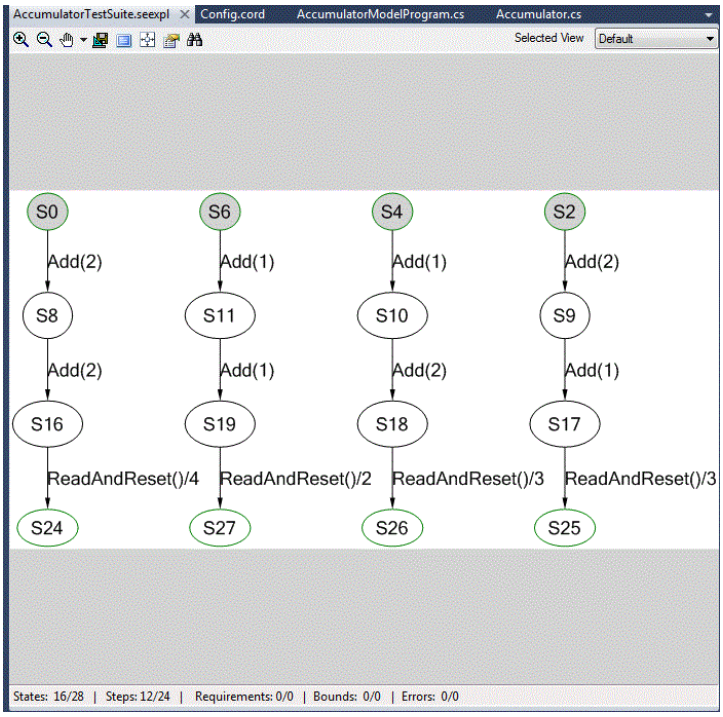
6 pav. Pavyzdinis fragmentas iš *SpecExplorer* įrankio su pavyzdinio projekto modeliu [33]

Vėliau kuriami funkcionalumų scenarijai pagal sukurtą sistemos modelį ir jame pridėtas funkcijas. Žemiau pateikiamas scenarijaus atvaizdavimo pavyzdys (žr. 7 pav.), kuriame yra iškviečiamos dvi pridėjimo funkcijos ir viena skaitymo ir perkrovimo funkcija [33].



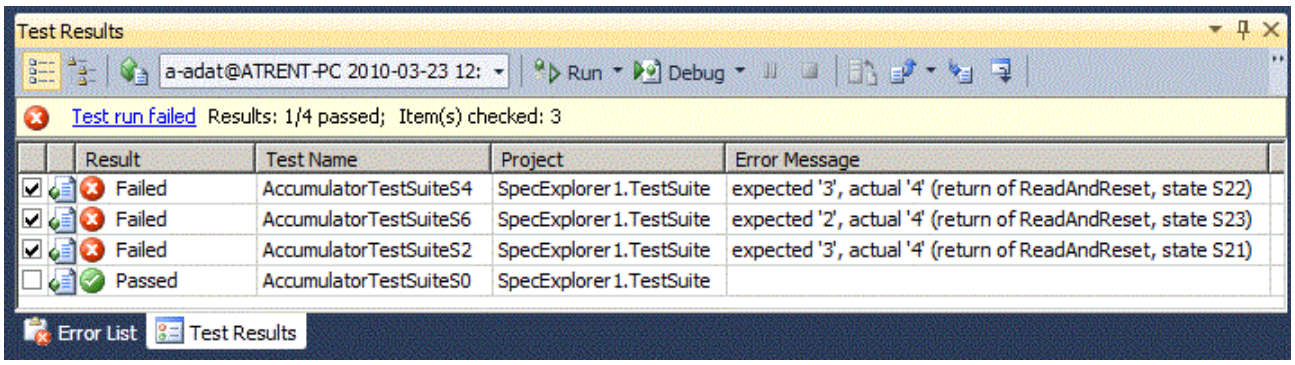
7 pav. Pavyzdinis fragmentas iš SpecExplorer įrankio su pavyzdiniu funkcionalumo scenarijumi [33]

Turint funkcionalumų scenarijus galima kurti testavimo atvejus, jie kuriami naudojant pagrindinį sistemos modelį ir funkcionalumų scenarijus ir jame atvaizduojamus galimus veiksmus. Žemiau pateikiamas pavyzdys su sugeneruotais testavimo atvejais pagal sistemos modelį (žr. 8 pav.) [33].



8 pav. Pavyzdinis *SpecExplorer* įrankio fragmentas su sugeneruotais testavimo atvejais [33]

Paskutinis žingsnis – sistemos testavimas pagal sugeneruotus testavimo atvejus. Tai galima lengvai padaryti pasinaudojus tam skirta *SpecExplorer* funkcija. Paleidus testų rinkinį gaunami rezultatai: Sėkmingas (angl. *Passed*) arba Nesėkmingas (angl. *Failed*). Žemiau pateikiamas pavyzdiniai testavimo rezultatų duomenys (žr. 9 pav.) [33].



9 pav. Pavyzdinis *SpecExplorer* įrankio fragmentas su testavimo rezultatais [33]

SpecExplorer analizė šiame darbe aktuali tuo, kad įrankis leidžia automatiškai generuoti testavimo atvejus pagal aprašytą sistemos veikimą ir scenarijus [33]. Tačiau šiame sprendime testavimo atvejai kuriami pagal programuotojo ar testuotojo rankiniu būdu sudarytą techninį sistemos aprašą [33]. Dėl šios priežasties įrankis nėra tiesiogiai pritaikytas testavimo atvejų generavimui pagal UML panaudojimo atvejų diagramas ir jų specifikacijas [29]. Taip pat įrankis orientuotas į C# ir .NET aplinką, todėl jo panaudojimas kitose technologijose yra ribotas [29].

1.6.2. *ConformIQ* įrankio naudojimas testavimo atvejų generavimo kontekste

Tai yra vienas iš populiariausių įrankių, naudojamų modeliais pagrįstam testavimui (angl. Model-Based Testing) [34]. Šis įrankis leidžia automatizuoti visą testavimo procesą – nuo specifikacijos iki galutinių testavimo atvejų generavimo, jų vykdymo ir rezultatų analizės [34]. ConformIQ išsiskiria tuo, kad palaiko įvairius modelių kūrimo formatus, įskaitant UML panaudojimo atvejus ir veiklos diagramas [34, 35]. Tai leidžia efektyviau kurti ir valdyti testavimo procesą dideliuose ir sudėtinguose projektuose [34, 35].

ConformIQ pagrindinis privalumas – automatizuotas testavimo atvejų generavimas [34]. Įrankis analizuoja pateiktą modelį, identifikuoja visus galimus sistemos veikimo scenarijus ir automatiškai generuoja testavimo atvejus [35]. Šis procesas užtikrina, kad būtų patikrinti tiek pagrindiniai srautai, tiek galimi klaidų scenarijai. Be to, ConformIQ integruojamas su įvairiomis programavimo ir testavimo aplinkomis, įskaitant CI/CD sistemas. Tai užtikrina nuolatinį testavimą ir ankstyvą klaidų aptikimą. Kitas svarbus privalumas – parametrizuoti testai, leidžiantys tikrinti įvairias sistemos sąlygas ir scenarijus naudojant tą patį modelį [35]. Įrankis taip pat geba optimizuoti generuojamus testus, kad būtų pasiekti maksimalūs rezultatai su minimaliais resursais [35].

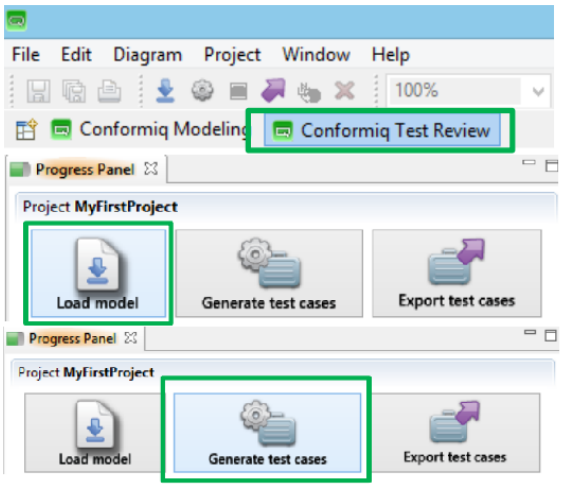
Nepaisant daugybės privalumų, ConformIQ turi ir tam tikrų apribojimų. Šio įrankio naudojimas reikalauja aukšto lygio žinių apie modelių kūrimą ir modeliais pagrįsto testavimo metodologijas [36]. Be to, licencija gali būti gana brangi mažoms įmonėms ar individualiems naudotojams [36].

ConformIQ dažniausiai naudojamas sudėtingose sistemose, kur reikalingas išsamus funkcionalumo patikrinimas [36]. Įrankis palaiko naujausias operacines sistemas ir yra nuolat atnaujinamas, kad atitiktų šiuolaikinius standartus [36].

Testavimo atvejų generavimas naudojant ConformIQ vykdomas tiesiai iš pateikto modelio [36]. Pavyzdžiui, jei naudojama UML panaudojimo atvejų diagrama, įrankis analizuoja visus įmanomus scenarijus, alternatyvius srautus ir ribines situacijas[36].

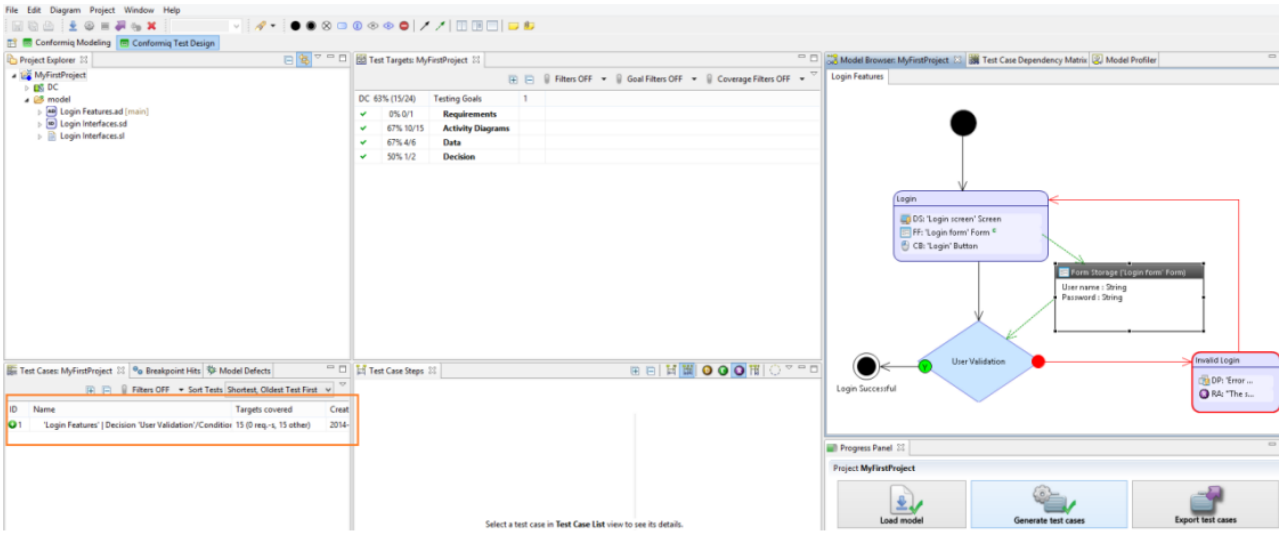
Automatiškai sugeneruoti testavimo atvejai atspindi visus galimus sistemos veikimo variantus, įskaitant netikėtus ar klaidų scenarijus [36].

Pradedant naudotis ConformIQ reikia sukurti sistemos modelį su galimomis funkcijomis. Sukurtas modelis yra naudojamas testavimo atvejams generuoti [36]. Žemiau pateiktame paveikslėlyje (žr. 10 pav.) pateikti žingsniai skirti pradėti testavimo atvejų generavimui.



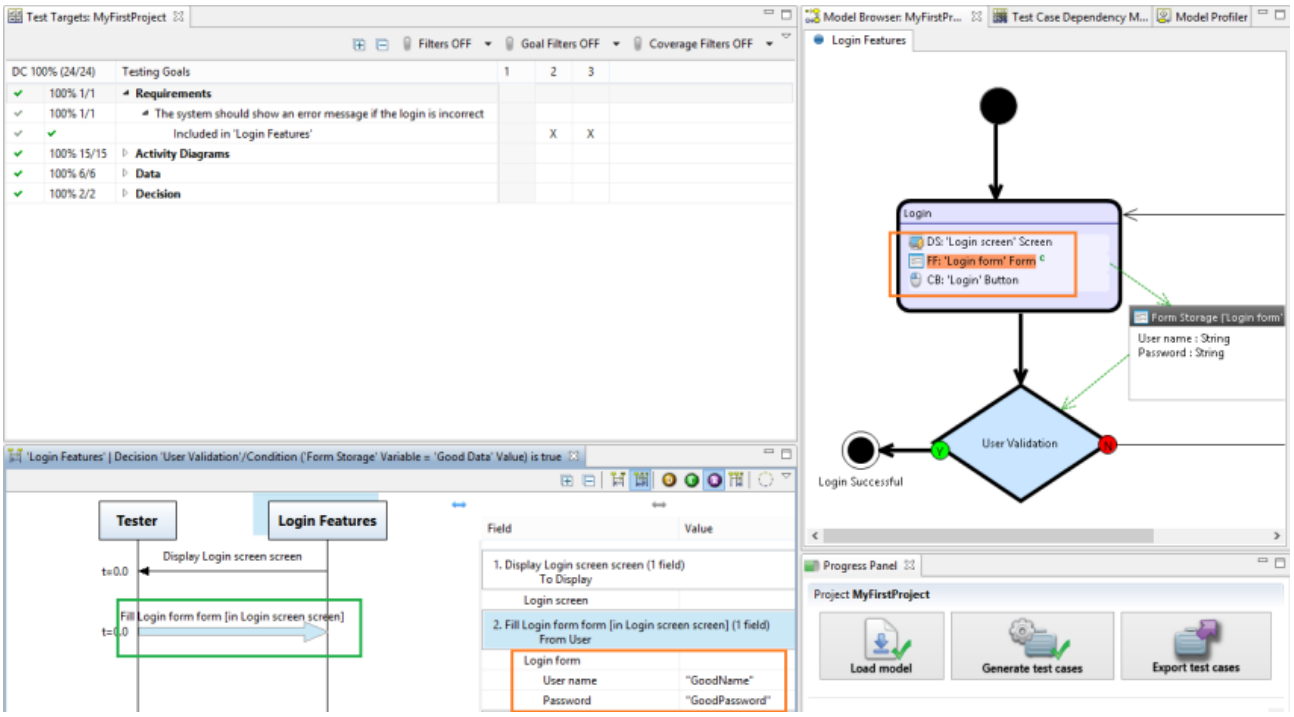
10 pav. Pavyzdinis *ConformIQ* įrankio fragmentas su testavimo atvejų generavimo inicijavimo žingsniais [36]

Sugeneruoti testavimo atvejai naudotojui bus pateikiami skirtuke „Testavimo atvejai“ (angl. *Test cases*). Jei testavimo atvejų generavimo metu pasitaiko klaidų – naudotojui tai pateikiama sistemos modelyje (dešinėje lango pusėje) pažymint veiklas raudona linija [37]. Programinės įrangos fragmentas pateiktas žemiau esančiame paveikslėlyje (žr. 11 pav.).



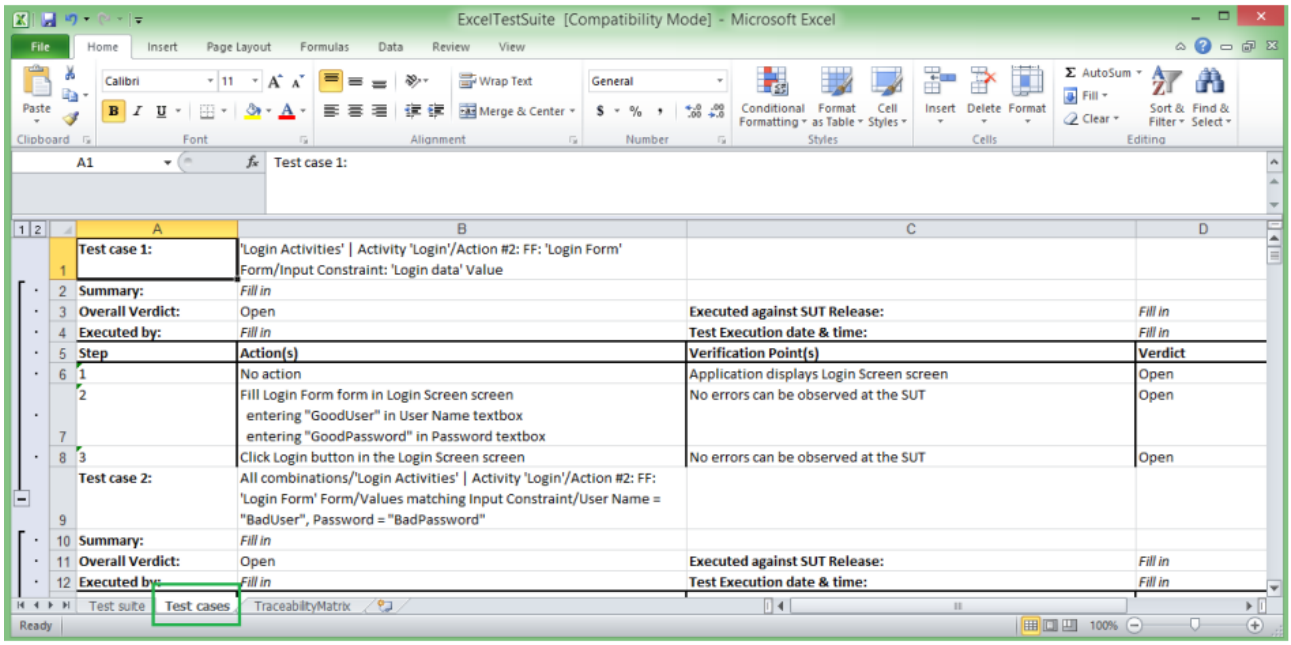
11 pav. Pavyzdinis *ConformIQ* įrankio fragmentas su rodomais sugeneruotais testavimo atvejais ir klaidos pažymėjimu modelyje [37]

Norint peržiūrėti visą sugeneruoto testo informaciją, naudotojas pasirenka norimą testą ir spaudžia ant jo veiksmo pažymėto rodykle [37]. Tuomet naudotojui bus atvaizduota informacija apie testo veiksmus, įvedamus duomenis bei sistemos modelyje bus pažymėta veikla, kuri atliekama testo metu [37]. Programinės įrangos langas su atvaizduojamais rezultatais pateikiamas paveikslėlyje žemiau (žr. 12 pav.).



12 pav. Pavyzdinis *ConformIQ* įrankio fragmentas su pasirinkto testo išsamia informacija [37]

Sugeneruotus testavimo atvejus galima eksportuoti xlsx formatu paspaudus eksportavimo mygtuką [37]. Eksportuotų testavimo atvejų vaizdą galima pamatyti žemiau pateiktame paveikslėlyje (žr. 13 pav.).



13 pav. Pavyzdiniai testavimo atvejai eksportuoti *xlsx* formatu [37]

ConformIQ analizė parodė, kad testavimo atvejų generavimas gali būti atliekamas naudojant įvairias UML diagramas ir detalesnius sistemos veikimo aprašus [34]. Įrankis geba generuoti tiek pagrindinius, tiek alternatyvius scenarijus, tačiau UML panaudojimo atvejų specifikacijos šiame sprendime nėra pagrindinis testavimo atvejų generavimo šaltinis [37]. Dažniausiai naudojamos papildomos UML diagramos, kuriose detaliau aprašoma sistemos logika ir veiksmų seka [37]. Dėl šios priežasties įrankis labiau orientuotas į sudėtingesnį testavimą nei į testavimo atvejų generavimą pagal standartizuotas panaudojimo atvejų specifikacijas [34].

1.6.3. *RT-Tester* įrankio naudojimas testavimo atvejų generavimo kontekste

Tai yra programinės įrangos testavimo įrankis, specialiai pritaikytas realaus laiko ir įterptinėms sistemoms [38]. Šis įrankis leidžia automatizuoti testavimo atvejų kūrimą remiantis UML modeliais, tokiais kaip būsenų (angl. *statechart*) ir sekų (angl. *sequence*) diagramos, kurios pateikia detalų sistemos veikimo ir sąveikų aprašą [38]. *RT-Tester* suteikia galimybę generuoti tikslius testavimo atvejus, kurie tikrina visas sistemos funkcijas, įskaitant pagrindinius ir alternatyvius scenarijus [38].

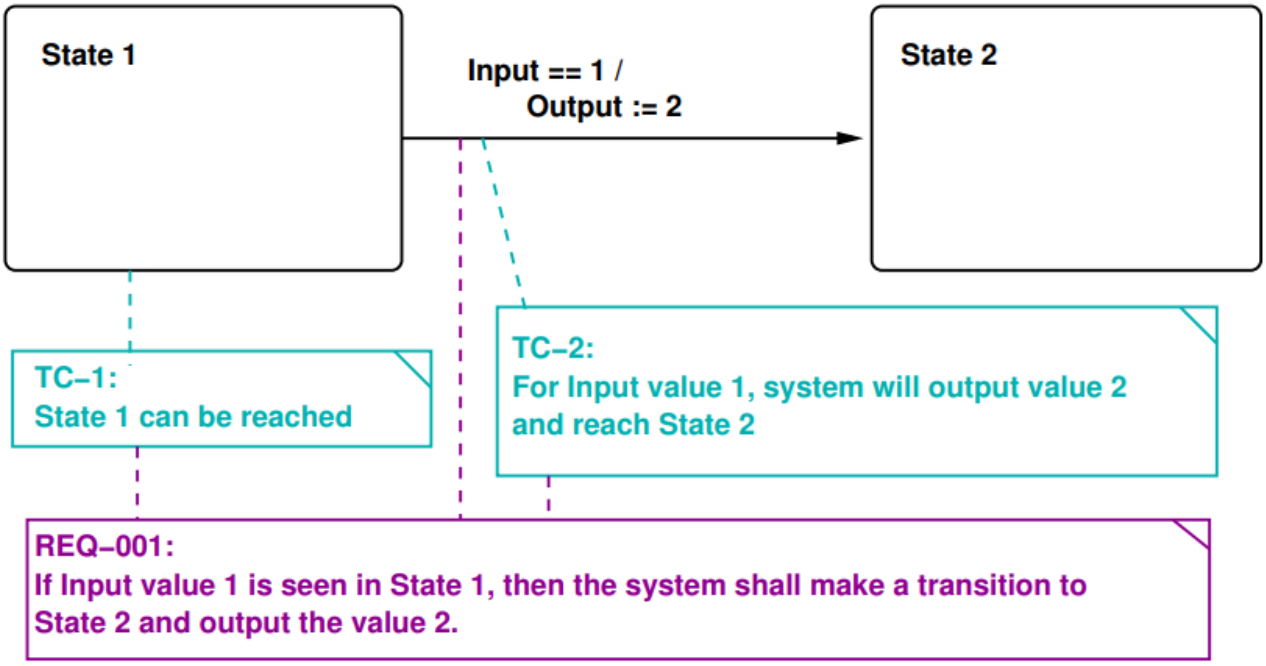
Vienas iš pagrindinių *RT-Tester* privalumų yra jo gebėjimas importuoti UML modelius iš populiarių projektavimo įrankių, tokių kaip *Enterprise Architect* ar *IBM Rational Rhapsody* [39]. Naudodamas šiuos modelius, įrankis analizuoja būsenas, įvykius ir perėjimus, kad sukurtų testavimo atvejus, atitinkančius sistemos specifikacijas [40]. Generuojami testai yra ypač tikslūs, nes jie grindžiami modeliavimo metu apibrėžtomis sistemos taisyklėmis ir apribojimais [40].

RT-Tester taip pat palaiko testų vykdymą ir analizę [40]. Naudotojai gali paleisti testus realaus laiko aplinkoje, o įrankis pateikia išsamius testų rezultatus bei padeda identifikuoti galimas klaidas ar neatitikimus [40]. Šis įrankis plačiai naudojamas pramonės srityse, kur reikalingas griežtas testavimas [40].

Nepaisant šių galimybių, *RT-Tester* turi ir trūkumų. Pavyzdžiui, jis nėra tiesiogiai pritaikytas testavimo atvejų generavimui iš UML panaudojimo atvejų diagramų, nes šios diagramos paprastai nėra pakankamai detalios testų kūrimui [40]. Tokiu atveju būtinas papildomas modelių detalizavimas, naudojant kitas UML diagramos rūšis, tokias kaip būsenų (angl. *statechart*) ar sekų (angl. *sequence*) [41]. Be to, *RT-Tester* labai priklauso nuo modelių kokybės – jei modeliai yra neišsamūs ar netikslūs, sugeneruoti testai gali neatitikti reikalavimų [41].

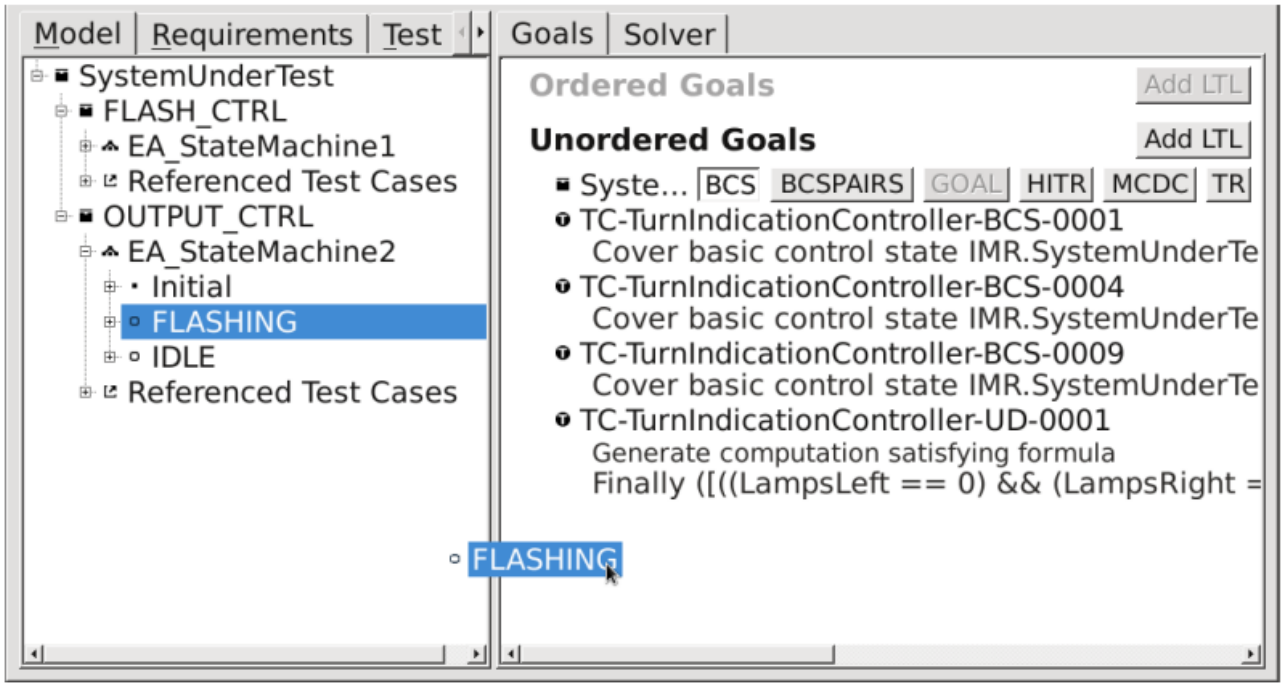
Apibendrinant, *RT-Tester* yra galingas įrankis, skirtas kokybiškam realaus laiko sistemų testavimui, tačiau jo veiksmingumas tiesiogiai priklauso nuo naudotojo pateiktų modelių kokybės ir detalumo. Šis įrankis yra vertinga pagalba kuriant ir vykdant testus sudėtingose sistemose, kur reikalaujama griežtų kokybės ir saugos standartų [41].

Naudojantis RT-Tester pirmiausiai yra suformuojamas sistemos modelis, kuris vėliau bus naudojamas testavimo atvejams generuoti. Vėliau sugeneruojami testavimo atvejai ir reikalavimai jiems, sistemos modelis su komentarais apie testavimo atvejus ir reikalavimus. Sistemos modelio anotavimo pavyzdys pateikiamas paveikslėlyje žemiau (žr. 14 pav.).



14 pav. Pavyzdinis sistemos modelio anotavimas pagal testavimo atvejus ir reikalavimus [42]

Sugeneruoti testavimo atvejai su *drag and drop* funkcija pateikiami naudotojui pasirinkti, kuriuos testavimo atvejus įtraukti į testavimo planą (žr. 15 pav.).



15 pav. Sugeneruoti testavimo atvejai naudotojui pateikiami lange su galimybe norimus testavimo atvejus įtraukti į testavimo planą [42]

RT-Tester analizė parodė, kad testavimo atvejų generavimas šiame įrankyje remiasi detaliu sistemos veikimo aprašymu, kuriame modeliuojamos sistemos būsenos, įvykiai ir veiksmų perėjimai [41]. UML panaudojimo atvejų diagramos šiame sprendime nėra pakankamos testavimo atvejų generavimui, todėl papildomai reikia naudoti kitų tipų UML diagramas ir detalesnius sistemos aprašus [40]. Tai rodo, kad įrankis nėra orientuotas į testavimo atvejų generavimą pagal tekstines panaudojimo atvejų specifikacijas [41].

1.6.4. *TestOptimal* įrankio naudojimas testavimo atvejų generavimo kontekste

TestOptimal yra pažangus automatizuoto testavimo įrankis, leidžiantis generuoti, valdyti ir vykdyti testavimo atvejus iš įvairių šaltinių, įskaitant *UML* panaudojimo atvejus ir reikalavimų specifikacijas [43]. Šis įrankis sukurtas siekiant automatizuoti testavimo procesus, sumažinant rankinio darbo kiekį ir padidinant testavimo efektyvumą [43].

Pagrindinė *TestOptimal* įrankio funkcija – generuoti tikslius testavimo atvejus, pagrįstus modeliais, kuriuose apibrėžtas sistemos funkcionalumas ir galimi klaidų scenarijai [43]. Įrankis leidžia integruoti testavimo scenarijus su kitomis vystymo aplinkomis, užtikrinant sklandų testavimo proceso vykdymą [43].

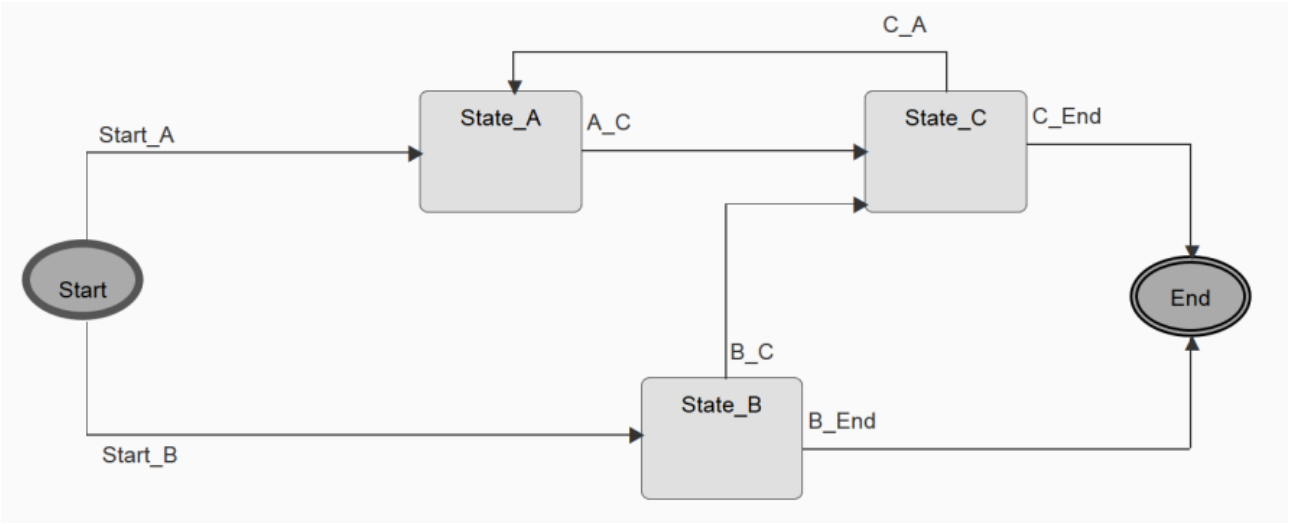
TestOptimal palaiko įvairius testavimo metodus, tokius kaip regresinis testavimas, funkcinis testavimas ir automatizuotas testavimas [43]. Jis suteikia galimybę stebėti testavimo proceso rezultatus realiuoju laiku ir analizuoti klaidas bei kitas susijusias problemas [44].

TestOptimal taip pat leidžia naudoti tikslias *UML* panaudojimo atvejų specifikacijas, kurių pagalba galima automatizuotai kurti testavimo atvejus [44]. Tai ypač naudinga projektams, kuriems svarbu užtikrinti sistemos veikimą pagal nustatytus reikalavimus [44].

TestOptimal tinkamas įvairių tipų testavimui, įskaitant priėmimo, regresinį, funkcijinį ir integracinį testavimą [44].

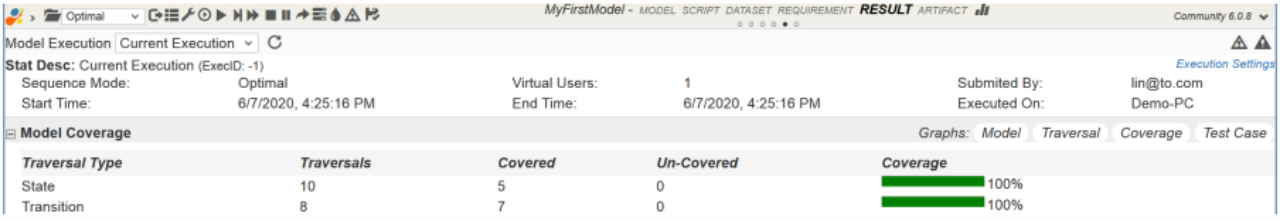
TestOptimal generuoja testavimo atvejus naudodamas modeliu pagrįstą testavimo (*MBT*) metodą [44]. Šio metodo metu sukurti modeliai atspindi sistemos elgseną, o testavimo atvejai yra išvedami iš šių modelių, siekiant apimti galimus scenarijus ir klaidų atvejus [44].

Naudojant *TestOptimal* įrankį pirmiausia sukuriamas sistemos būsenų modelis, jo pavyzdį galima matyti žemiau pateiktame paveikslėlyje (žr. 16 pav.).



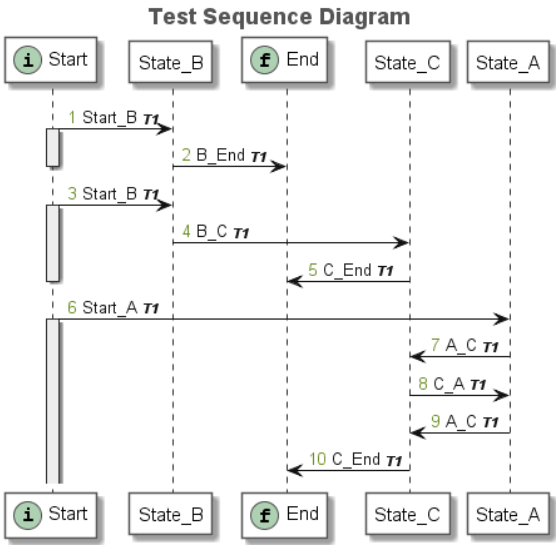
16 pav. Pavyzdinis sistemos būsenų modelis [45]

Sudarius sistemos būsenų modelį, galima generuoti testavimo atvejus. Sugeneravus testavimo atvejus, rodoma išsami informacija apie juos. Žemiau pateikiamas ekranvaizdis iš lango su testavimo atvejų informacija (žr. 17 pav.).



17 pav. Fragmentas iš *TestOptimal* įrankio su sugeneruotų testavimo atvejų informacija [45]

Sugeneruotus testavimo atvejus galima peržiūrėti testavimo sekos diagramos pavidalu. Pavyzdinio testo sekos diagrama vaizduojama žemiau esančiame paveikslėlyje (žr. 18 pav.).



18 pav. Pavyzdinė testo sekos diagrama [45]

TestOptimal analizė aktuali, nes įrankis leidžia automatiškai generuoti testavimo atvejus pagal įvairius sistemos veikimo aprašus ir UML diagramas [45]. Tačiau pagrindinis dėmesys šiame sprendime skiriamas sistemos būsenų ir veiksmų sekų analizei, todėl UML panaudojimo atvejų specifikacijos nėra pagrindinis informacijos šaltinis testavimo scenarijų generavimui [45]. Norint sugeneruoti detalesnius testavimo atvejus, reikalingi papildomi sistemos veikimo aprašai ir techninė informacija apie sistemos veikimo logiką [45].

1.6.5. *TestCaseLab* įrankio naudojimas testavimo atvejų generavimo kontekste

Tai yra testavimo valdymo įrankis, skirtas rankinių testavimo atvejų kūrimui, organizavimui ir vykdymui [46]. Šis įrankis leidžia centralizuotai valdyti testavimo procesą – nuo testavimo atvejų aprašymo iki jų vykdymo ir rezultatų sekimo [46]. TestCaseLab išsiskiria paprasta naudotojo sąsaja bei galimybe integruotis su kitais įrankiais, tokiais kaip Jira ar Trello, kas leidžia efektyviau valdyti projektus ir testavimo veiklas [46].

Vienas iš svarbesnių TestCaseLab funkcionalumų yra integracija su Treeify įrankiu, kuris suteikia galimybę generuoti testavimo atvejus naudojant dirbtinio intelekto metodus [47]. Naudojant šią integraciją, testavimo atvejai gali būti kuriami remiantis tekstiniais reikalavimais ar kitais įvesties duomenimis, kuriuos analizuoja Treeify sistema. Įrankis identifikuoja galimus scenarijus ir automatiškai suformuoja testavimo atvejų juodraščius, kurie vėliau gali būti perkelti į TestCaseLab aplinką tolimesniam redagavimui ir vykdymui [47]. Toks sprendimas leidžia sumažinti rankinio darbo kiekį ir paspartinti testavimo dokumentacijos rengimą.

Pagrindinis TestCaseLab privalumas – lankstus testavimo atvejų valdymas ir galimybė derinti rankinį bei pusiau automatinį testavimo atvejų kūrimą [47]. Įrankis leidžia kurti struktūrizuotus testavimo atvejus, priskirti juos skirtingiems testavimo rinkiniams, sekti jų vykdymo būsenas bei analizuoti rezultatus [47]. Be to, integracijos su išorinėmis sistemomis leidžia užtikrinti testavimo proceso nuoseklumą viso programinės įrangos kūrimo gyvavimo ciklo metu. Taip pat galima eksportuoti testavimo atvejus į įvairius formatus, pavyzdžiui, xlsx, kas palengvina duomenų dalijimąsi [47].

Nepaisant privalumų, *TestCaseLab* turi ir tam tikrų apribojimų. Testavimo atvejų generavimas naudojant *Treeify* nėra visiškai automatizuotas procesas, kadangi sugeneruoti testai dažnai reikalauja papildomo naudotojo įsikišimo ir koregavimo [46]. Be to, šis sprendimas nėra tiesiogiai orientuotas į modeliais pagrįstą testavimą, todėl testavimo atvejai nėra generuojami iš formalių modelių, tokių kaip UML diagramos. Dėl šios priežasties gali trūkti nuoseklumo ir pilno scenarijų padengimo, ypač sudėtingesnėse sistemose.

TestCaseLab dažniausiai naudojamas projektuose, kuriuose svarbus paprastas ir aiškus testavimo proceso valdymas bei komandinis darbas [46]. Įrankis yra tinkamas mažoms ir vidutinėms komandoms, kurios siekia struktūruoti testavimo veiklas, tačiau nereikalauja sudėtingų modeliais pagrįsto testavimo sprendimų. Testavimo atvejų kūrimas šiame įrankyje vykdomas rankiniu būdu arba naudojant integracijas, tokias kaip Treeify, kurios leidžia generuoti pradinius testų variantus [46]. Tačiau galutinis testavimo atvejų tikslumas ir kokybė išlieka priklausomi nuo naudotojo atliekamo peržiūrėjimo ir validavimo.

TestCaseLab analizė parodė, kad įrankis daugiausia skirtas rankinių testavimo atvejų valdymui, organizavimui ir vykdymui [46]. Nors integracija su *Treeify* leidžia generuoti pradinius testavimo scenarijus naudojant dirbtinio intelekto metodus, sugeneruoti rezultatai dažnai reikalauja papildomo naudotojo koregavimo, o testavimo scenarijai nėra generuojami pagal UML panaudojimo atvejų diagramas ar specifikacijas [46].

1.6.6. Dirbtinio intelekto įrankių naudojimas testavimo atvejų generavimo kontekste

Dirbtinio intelekto naudojimas testavimo atvejams generuoti tampa vis populiaresnis, siekiant automatizuoti procesus ir efektyviau valdyti testavimo veiklą [48]. Šis metodas remiasi įvairiais algoritmais, kurie analizuoja didelius duomenų kiekius, atpažindami tendencijas ir kuria testavimo scenarijus remiantis istorine informacija apie sistemos veikimą [48].

Dirbtinio intelekto generuojami testavimo atvejai padeda sumažinti rankinio darbo sąnaudas ir užtikrina, kad bus apimta didesnė scenarijų įvairovė [48]. Tai leidžia greičiau identifikuoti klaidas ir sumažina klaidingų testavimo atvejų tikimybę [49]. Generuojant testavimo atvejus, dirbtinis intelektas dažnai remiasi duomenų analize, kuri gali būti pagrįsta naudotojų elgsena, sistemos veikimo žurnalais arba kitomis istorinėmis duomenų bazėmis [49].

Dideli kalbos modeliai (angl. *Large Language Models*, LLM) gali būti taikomi testavimo atvejams generuoti, kai testavimo informacija pateikiama tekstiniu pavidalu [50]. Šie modeliai geba analizuoti reikalavimų specifikacijas ar panaudojimo atvejų aprašus ir automatiškai atpažinti naudotojo veiksmus bei sistemos reakcijas [50]. Tai leidžia iš tekstinių dokumentų išskirti testavimui reikalingus elementus ir juos panaudoti struktūrizuotų testavimo atvejų kūrimui [50]. Vis dėlto, gautų rezultatų tikslumas priklauso nuo įvesties duomenų aiškumo ir kokybės [50].

Papildomai, buvo įvertinta galimybė testavimo atvejų generavimui naudoti generatyvinio dirbtinio intelekto modelį. Kadangi tokie modeliai geba generuoti tekstinį turinį pagal pateiktas instrukcijas, buvo analizuojama, ar jie gali generuoti struktūrizuotus rankinio testavimo atvejus pagal UML panaudojimo atvejų specifikacijas bei iš anksto apibrėžtas taisykles.

Vertinimui buvo naudojamas generatyvinio dirbtinio intelekto modelis „OpenAI GPT-4.1-mini“. Modeliui buvo pateikiamos UML panaudojimo atvejų specifikacijos, testavimo atvejų šablonas bei papildomos generavimo taisyklės. Pagrindinis tikslas buvo įvertinti, ar modelis gali savarankiškai generuoti nuoseklius ir standartizuotus testavimo atvejus.

Bandymui buvo naudojama nedidelės knygų parduotuvės sistemos UML panaudojimo atvejų specifikacija, sudaryta testavimo tikslams. Specifikacijoje buvo pateikti 6 panaudojimo atvejai, apimantys skirtingus požymius: peržiūros, kūrimo, redagavimo, šalinimo, sisteminius bei tik pagrindinio scenarijaus atvejus. Taip pat dalyje panaudojimo atvejų buvo pateikti alternatyvūs scenarijai ir naudotojų teisės, leidžiančios įvertinti, kaip dirbtinio intelekto modelis laikosi pateiktų generavimo taisyklių. Naudota užklausa pateikiama žemiau esančiame paveikslėlyje (žr. 19 pav.).

```
Sugeneruok testavimo atvejus pagal pateiktą UML panaudojimo atvejų specifikaciją.
Kiekvienas testavimo atvejis turi turėti:
   –  testavimo scenarijaus numerį;
   –  pavadinimą;
   –  roles ir teises;
   –  detalų testavimo scenarijų;
   –  laukiamą rezultatą.

Testavimo scenarijaus numeris sudaromas paimant panaudojimo atvejo numerį, pridedant
raides TS ir testavimo atvejo numerį, juos atskirti tašku, pvz.: PA1.TS1.

Testavimo žingsniai turi prasidėti žodžiu „Naudotojas“, o laukiamas rezultatas – žodžiu
„Sistema“.

Jeigu specifikacijoje pateiktos naudotojo teisės, jos turi būti įtrauktos į testavimo
atvejį.

Jeigu panaudojimo atvejų diagramoje yra aktorius, tai reikia nurodyti prie „Rolės ir
teisės“, tokiu šablonu: Naudotojas yra X (kur X yra nuskaitytas aktorius iš diagramos).

Jeigu yra alternatyvūs scenarijai – turi būti sugeneruoti papildomi testavimo atvejai.

Papildomai, peržiūros tipo atvejams sugeneruok filtravimo, puslapiavimo, rikiavimo
atvejus, o visiems kitiems – netinkamų teisių atvejus.
```

19 pav. Dirbtinio intelekto įrankiui pateiktos užklausos tekstas

Atlikus generavimo bandymus buvo pastebėta, kad modelis geba sugeneruoti dalį testavimo atvejų, tačiau rezultatai ne visuomet atitiko pateiktas taisykles ir reikalaujamą struktūrą. Nustatyti šie pagrindiniai trūkumai:

- ne visuose testavimo žingsniuose buvo laikomasi nustatytos struktūros („Naudotojas“ / „Sistema“);
- „Sistema patikrina“ taisyklės dažnai buvo ignoruojamos arba netinkamai suskaidomos;
- ne visuomet buvo sugeneruojami testavimo atvejai netinkamoms teisėms;
- kai kuriais atvejais buvo sugeneruojami papildomi scenarijai, kurie nebuvo aprašyti specifikacijoje, ir nereikalingi testavimo procese;
- sugeneruotų rezultatų struktūra tarp skirtingų generavimo bandymų nebuvo nuosekli, t. y. kiekvieną kartą sukurti testavimo atvejai struktūriškai nebuvo vienodi.

Taip pat buvo pastebėta, kad, naudojant tą pačią užklausą bei tą pačią specifikaciją, sugeneruoti rezultatai skirdavosi tarp skirtingų generavimo kartų. Dėl šios priežasties nebuvo galima užtikrinti stabilaus testavimo atvejų generavimo proceso.

Atlikta analizė parodė, kad generatyvinio dirbtinio intelekto modelis gali būti naudojamas kaip pagalbinė priemonė testavimo atvejų generavimo procese, tačiau vien tik dirbtinio intelekto modelio nepakanka norint užtikrinti standartizuotą ir taisyklėmis pagrįstą testavimo atvejų generavimą [18].

1.6.7. Apibendrinimas

Išanalizavus esamus sprendimus, pastebėta, kad dauguma jų neturi galimybės generuoti testavimo atvejų tiesiogiai pagal UML panaudojimo atvejų diagramas ir jų specifikacijas. Daugelyje esamų sprendimų UML panaudojimo atvejų diagrama ir jų specifikacija veikia kaip papildoma informacija testavimo atvejams generuoti. Detalesni lyginamosios analizės rezultatai pagal kriterijus pateikti žemiau esančioje lentelėje (žr. 2 lentelė. Esamų sprendimų palyginimas).

2 lentelė. Esamų sprendimų palyginimas

Palyginimo kriterijus	**SpecExplorer[33]**	**Conformiq[37]**	**RT-Tester[38]**	**TestOptimal [45]**	**TestCaseLab[47]**	**DI taikymas testavimo atvejų generavimui**
Ar UML diagramos naudojamos kaip šaltiniai testavimo atvejams generuoti?	Taip	Taip	Taip	Taip	Ne	Taip
Naudojamos UML diagramos	Būsenų ir veiklos diagramos	Veiklos ir sekų diagramos	Veiklos ir sekų diagramos	Sekų diagramos	Ne	Panaudojimo atvejų diagramos, veiklos diagramos
UML panaudojimo atvejų specifikacijos naudojimas	Iš dalies (tik kaip papildoma informacija)	Iš dalies (tik kaip papildoma informacija)	Iš dalies (tik kaip papildoma informacija)	Iš dalies (tik kaip papildoma informacija)	Ne	Taip
Ar atliekama įvesties duomenų analizė (patikra, ar panaudojimo atvejai diagramoje ir lentelėse sutampa)?	Ne	Ne	Ne	Ne	Ne	Taip

Palyginimo kriterijus	SpecExplorer[33]	Conformiq[37]	RT-Tester[38]	TestOptimal [45]	TestCaseLab[47]	DI taikymas testavimo atvejų generavimui
Palaikomų formatų eksportas	TXT, XML, Visual Studio suderinami formatai	XML, CSV, JIRA	TXT, CSV	JSON, XML	CSV, TestCaseLab testavimo atvejų formatas	CSV, XML, JSON, HTML
Skirtas skirtingiems testavimo tipams	Palaikomi funkciniai ir regresijos testai	Palaikomi regresijos ir integracijos testai	Palaikomi funkciniai, sistemų ir regresijos testai	Labiau orientuotas į funkcinius testus	Labiausiai orientuota į funkcinius testus	Gali sugeneruoti funkcinius, regresijos, ir integracijos testus

Atlikus esamų sprendimų analizę pastebėta, kad dauguma įrankių UML panaudojimo atvejų specifikacijas naudoja tik kaip papildomą informacijos šaltinį. Tokiu atveju pagrindinis testavimo atvejų generavimas dažniausiai remiasi kitų tipų UML diagramomis, pavyzdžiui, veiklos, sekų ar būsenų diagramomis, o panaudojimo atvejų specifikacijos naudojamos tik papildomai scenarijų informacijai ar reikalavimams pateikti. Dėl šios priežasties UML panaudojimo atvejų specifikacijos dažniausiai nėra pagrindinis testavimo atvejų generavimo šaltinis, nors jose pateikiama detali informacija apie naudotojo žingsnius, sistemos atsakus, alternatyvius scenarijus ir išankstines sąlygas.

1.7. Siekiamo sprendimo apibrėžimas

Atlikus UML panaudojimo atvejų, jų specifikacijų bei esamų testavimo atvejų generavimo sprendimų analizę, nustatyta, kad trūksta aiškiai apibrėžtos metodikos, leidžiančios generuoti rankinio testavimo atvejus tiesiogiai pagal UML panaudojimo atvejų specifikacijas. Dėl šios priežasties šiame darbe siekiama sukurti metodiką, skirtą struktūrizuotam testavimo atvejų generavimui pagal standartizuotas panaudojimo atvejų specifikacijas.

Metodika paremta UML panaudojimo atvejų diagramų ir jų specifikacijų naudojimu, kai panaudojimo atvejai aprašomi pagal nustatytą šabloną bei apibrėžtas taisykles. Remiantis pateikta informacija, sudaromi standartizuoti testavimo atvejai, apimantys pagrindinius, alternatyviuosius bei papildomus scenarijus pagal pasirinktus požymius. Tokiu būdu siekiama sumažinti rankinio darbo kiekį testavimo procese bei užtikrinti nuoseklesnį testavimo atvejų sudarymą. Siūloma metodika orientuota į projektus, kuriuose naudojamos UML panaudojimo atvejų specifikacijos.

1.8. Analizės išvados

Atlikus tyrimo objekto, jo naudotojų bei esamų problemos sprendimo metodų analizę, buvo priimtos išvados:

1. Išanalizavus UML panaudojimo atvejų diagramas ir jų specifikacijas, pastebėta, kad jose pateikiama informacija, tokia kaip išankstinės sąlygos, naudotojo veiksmai, naudotojo teisės ir laukiami rezultatai, gali būti naudojama testavimo atvejų sudarymui.
2. Atlikus testavimo atvejų, jų struktūros ir sudarymo principų analizę, buvo pastebėta , jog norint sukurti tikslų testavimo atvejį naudojantis UML panaudojimo atvejų diagrama ir jos

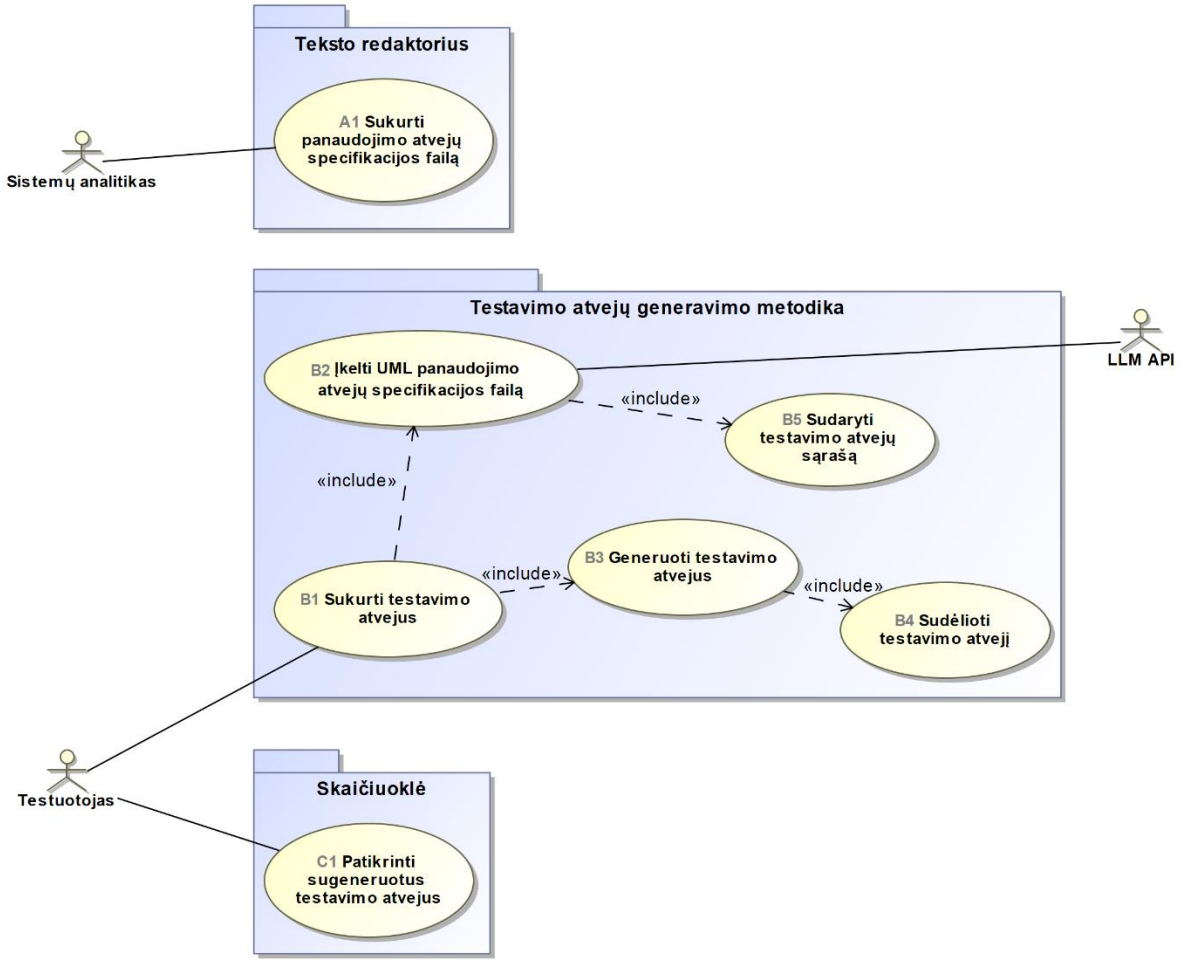
specifikacija, reikia užtikrinti, kad specifikacija būtų išsami ir aiški, aprašanti visus būtinus sistemos reikalavimus – išankstines sąlygas, naudotojo veiksmus, numatomą sistemos rezultatą, teises.

3. Ištyrus esamus metodus, įrankius ir technologijas, skirtas testavimo atvejų generavimui iš UML diagramų, pastebėta, kad esami sprendimai dažniausiai naudoja UML panaudojimo atvejų diagramas tik kaip papildomą informaciją. Būtų naudinga sukurti metodiką, kuri galėtų apdoroti UML panaudojimo atvejų diagramas kaip pagrindinę informaciją.

2. Testavimo atvejų generavimo metodikos reikalavimų specifikacija, formalizuotas aprašas

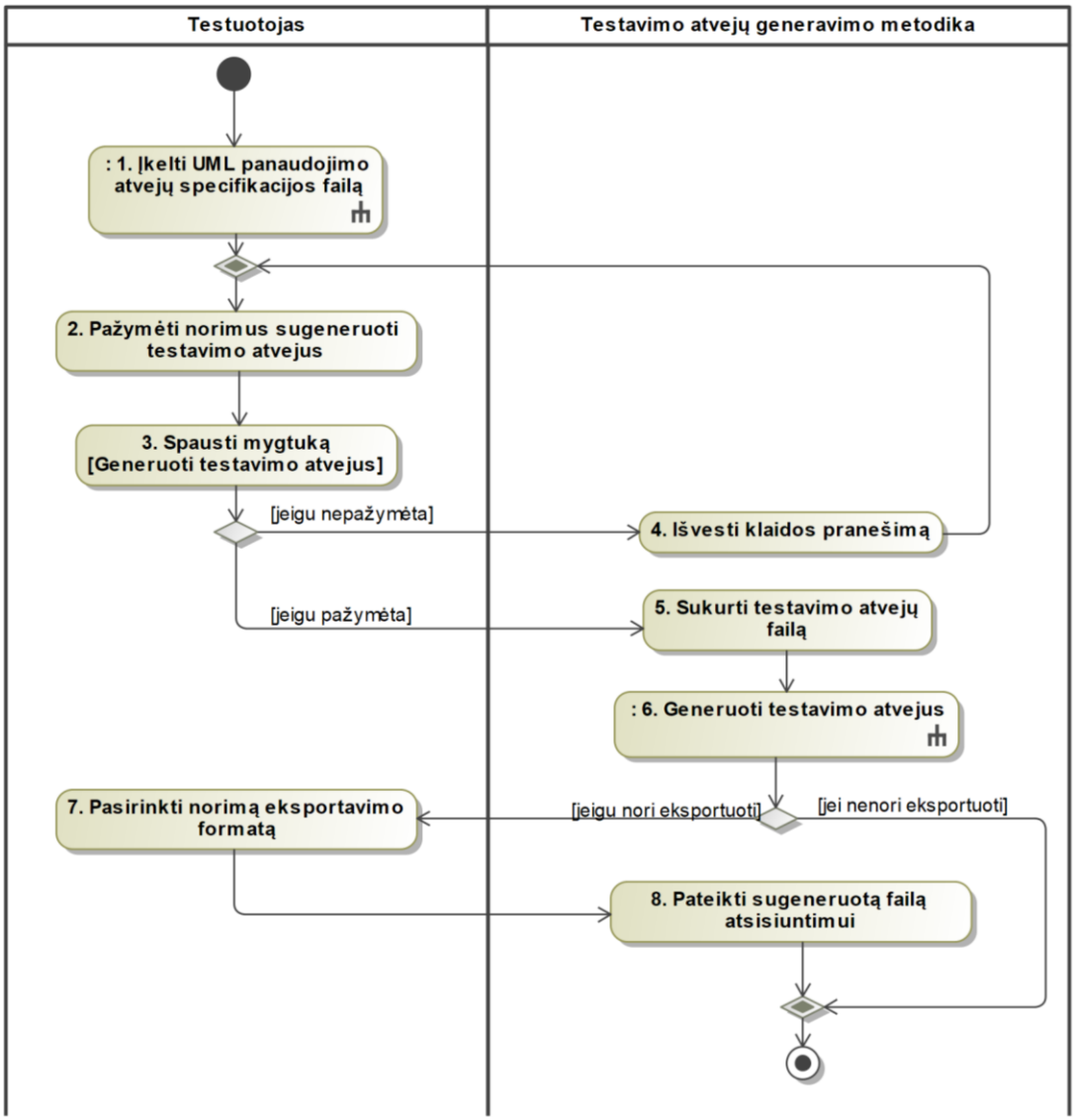
2.1. Testavimo atvejų generavimo metodikos funkciniai reikalavimai

Funkciniai reikalavimai apibrėžia pagrindinius metodikos veikimo principus ir veiksmus, reikalingus testavimo atvejų generavimui pagal UML panaudojimo atvejų specifikacijas. Metodikoje numatoma, kad testuotojas pateikia standartizuotą „.docx“ formato reikalavimų specifikaciją su panaudojimo atvejų diagrama ir specifikacijų lentelėmis. Remiantis pateikta informacija atliekama specifikacijų analizė, identifikuojami panaudojimo atvejai bei pagal nustatytas taisykles formuojami testavimo atvejai. Taip pat metodikoje numatyta galimybė peržiūrėti sugeneruotus testavimo atvejus ir eksportuoti juos tolimesniam naudojimui. Pagrindiniai naudotojo atliekami veiksmai pateikiami UML panaudojimo atvejų diagramoje (žr. 20 pav.).



20 pav. Testavimo atvejų generavimo metodikos panaudojimo atvejų diagrama

Panaudojimo atvejo „B1 Sukurti testavimo atvejus“ veiklos diagrama pateikiama žemiau esančiame paveikslėlyje (žr. 21 pav.). Joje atvaizduojamas aukšto lygio testavimo atvejų generavimo procesas.

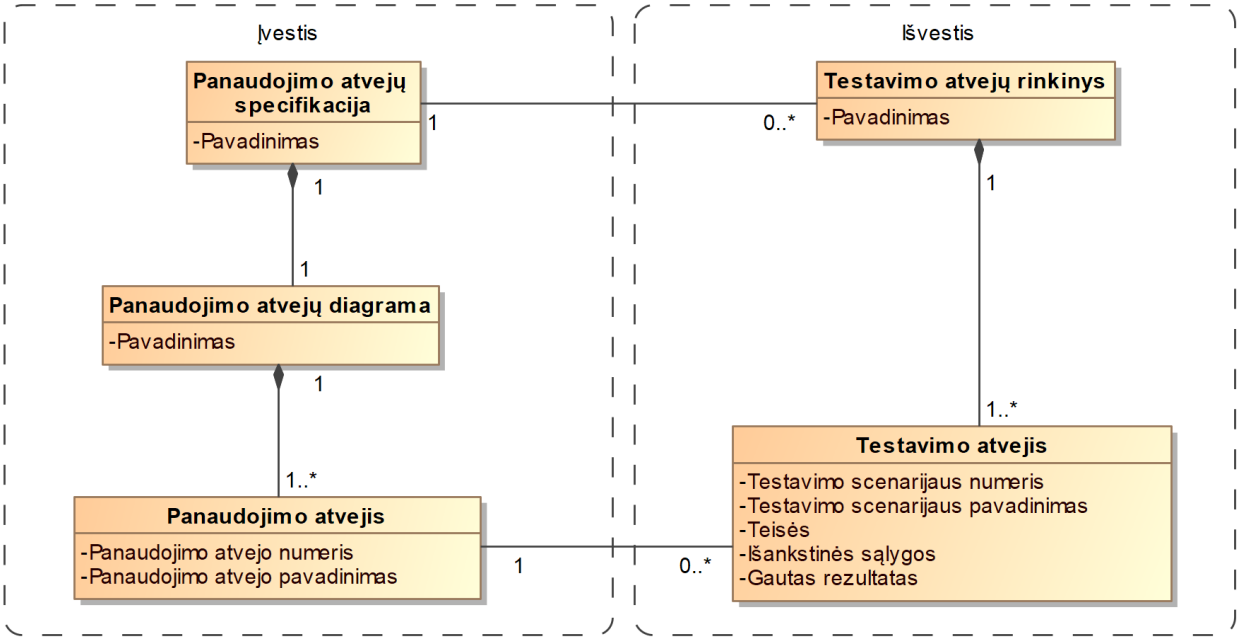


21 pav. Veiklos diagrama, parodanti bendrinį testavimo atvejų kūrimo procesą

Likusiųjų panaudojimo atvejų veiklos diagramos išsamiau aptariamos vėlesniame skyriuje (žr. 2.4.1).

2.2. Testavimo atvejų generavimo metodikos duomenų modelis

Testavimo atvejų generavimo procesui įgyvendinti reikalinga sukurti duomenų modelį, kuris apibrėžtų pagrindinius duomenų tipus, jų savybes ir tarpusavio ryšius. Žemiau pateikiamas aukšto lygio metodikos duomenų modelis (žr. 22 pav.).



22 pav. Bendrasis testavimo atvejų generavimo metodikos duomenų modelis

Įvesties dalyje apibrėžiami UML panaudojimo atvejų specifikacijos duomenys, kurie naudojami testavimo atvejų formavimui. Svarbiausias šios dalies elementas yra panaudojimo atvejis, nes būtent pagal jo informaciją formuojami testavimo scenarijai. Dėl šios priežasties modelyje numatyta, kad vienoje panaudojimo atvejų diagramoje gali būti keli panaudojimo atvejai, o kiekvienam jų gali būti sugeneruotas atskiras testavimo atvejų rinkinys. Išvesties dalyje apibrėžiama sugeneruotų testavimo atvejų struktūra. Modelyje akcentuojama, kad vienam panaudojimo atvejui gali būti sugeneruotas ne vienas testavimo atvejis, nes papildomai gali būti formuojami alternatyvieji, filtravimo, puslapiavimo, rikiavimo ar netinkamų teisių scenarijai. Įvesties ir išvesties dalių duomenų modeliai detalizuojami tolimesniuose skyriuose (žr. 2.4.2 ir 2.4.3).

2.3. Testavimo atvejų generavimo metodikos nefunkciniai reikalavimai

Nefunkciniai reikalavimai nustato, kokius įvesties duomenų formatus sistema gali priimti, kokiais formatais gali eksportuoti sugeneruotus testavimo atvejus, taip pat kitus našumo ir naudojimo apribojimus.

Nefunkciniai testavimo atvejų generavimo metodo reikalavimai:

1. Metodika turi priimti failus, kuriuose yra vienas panaudojimo atvejų diagramos paveikslėlis ir paveikslėlyje atvaizduotų panaudojimo atvejų specifikacijų lentelės.
2. Metodika turi priimti tik standartizuotos struktūros .docx formato failus, kuriuose panaudojimo atvejai pateikiami pagal apibrėžtą lentelės formatą ir atitinka apibrėžtas taisykles (žr. 2.4.2).
3. Metodika turi suformuoti apibrėžtos formos ir atitinkančius apibrėžtas taisykles testavimo atvejus (žr. 2.4.3Testavimo atvejų generavimo metodikos išvesties duomenų modelis, šablonas ir taisyklės).
4. Metodika turi turėti galimybę eksportuoti sugeneruotus testavimo atvejus į .csv, .xlsx, .docx, .pdf formato failus.
5. Metodika turi būti palaikoma Windows 10 ir vėlesnėse operacinės sistemos versijose.

6. Metodika turi užtikrinti, kad visi įkeliami ir eksportuojami failai būtų saugomi ir apdorojami naudojant UTF-8 koduotę.

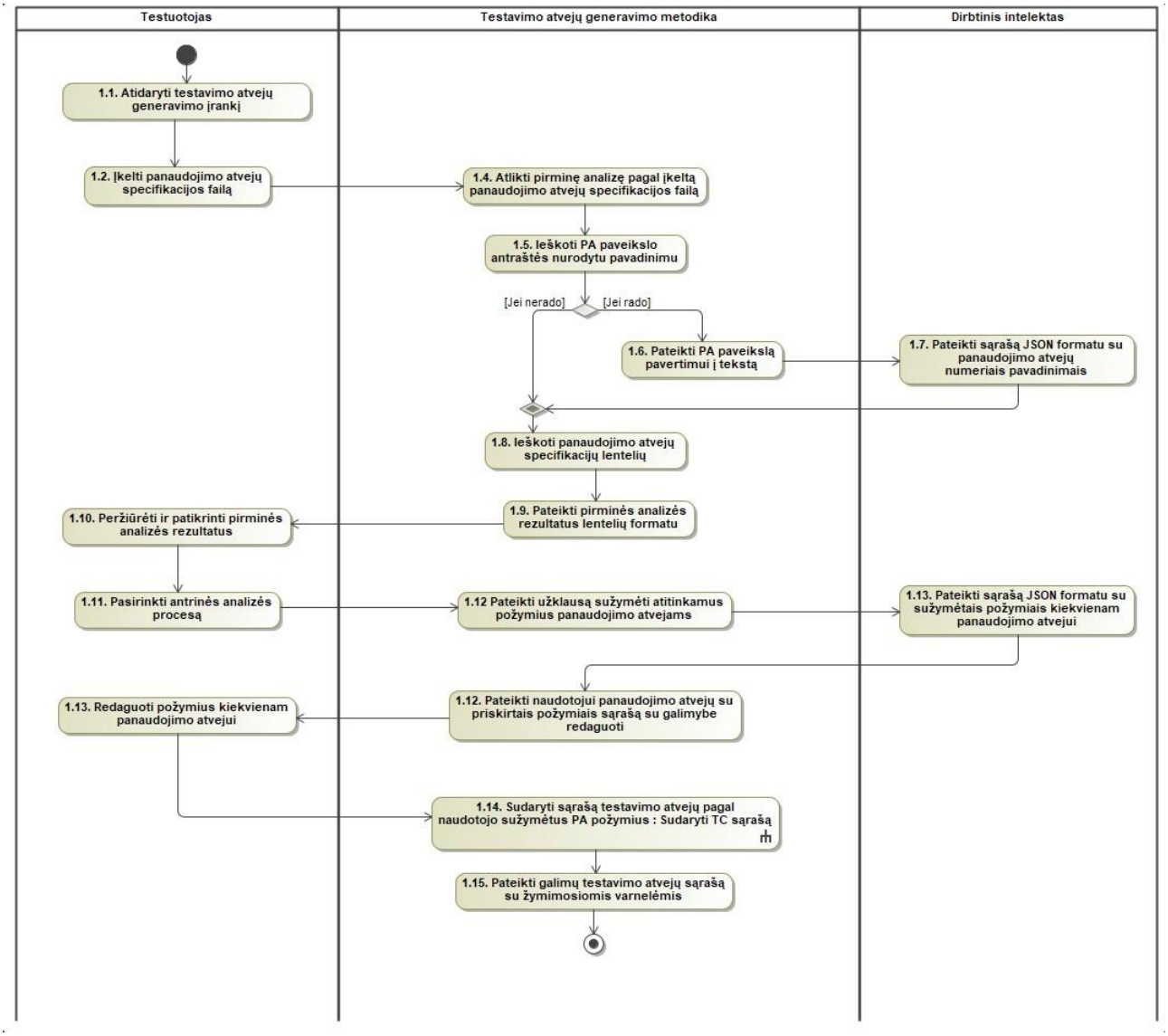
2.4. Formalizuotas testavimo atvejų generavimo metodikos aprašas

2.4.1. Testavimo atvejų generavimo metodikos procesų diagramos

Šiame skyriuje pateikiami pagrindiniai testavimo atvejų generavimo metodikos procesai ir jų tarpusavio ryšiai. Procesų diagramos sudarytos remiantis anksčiau apibrėžtu duomenų modeliu (žr. 2.2) bei naudotojo atliekamais veiksmais metodikos taikymo metu. Metodikos esmę sudaro nuoseklus UML panaudojimo atvejų specifikacijų apdorojimas – nuo reikalavimų specifikacijos analizės ir panaudojimo atvejų identifikavimo iki testavimo atvejų formavimo pagal nustatytas taisykles.

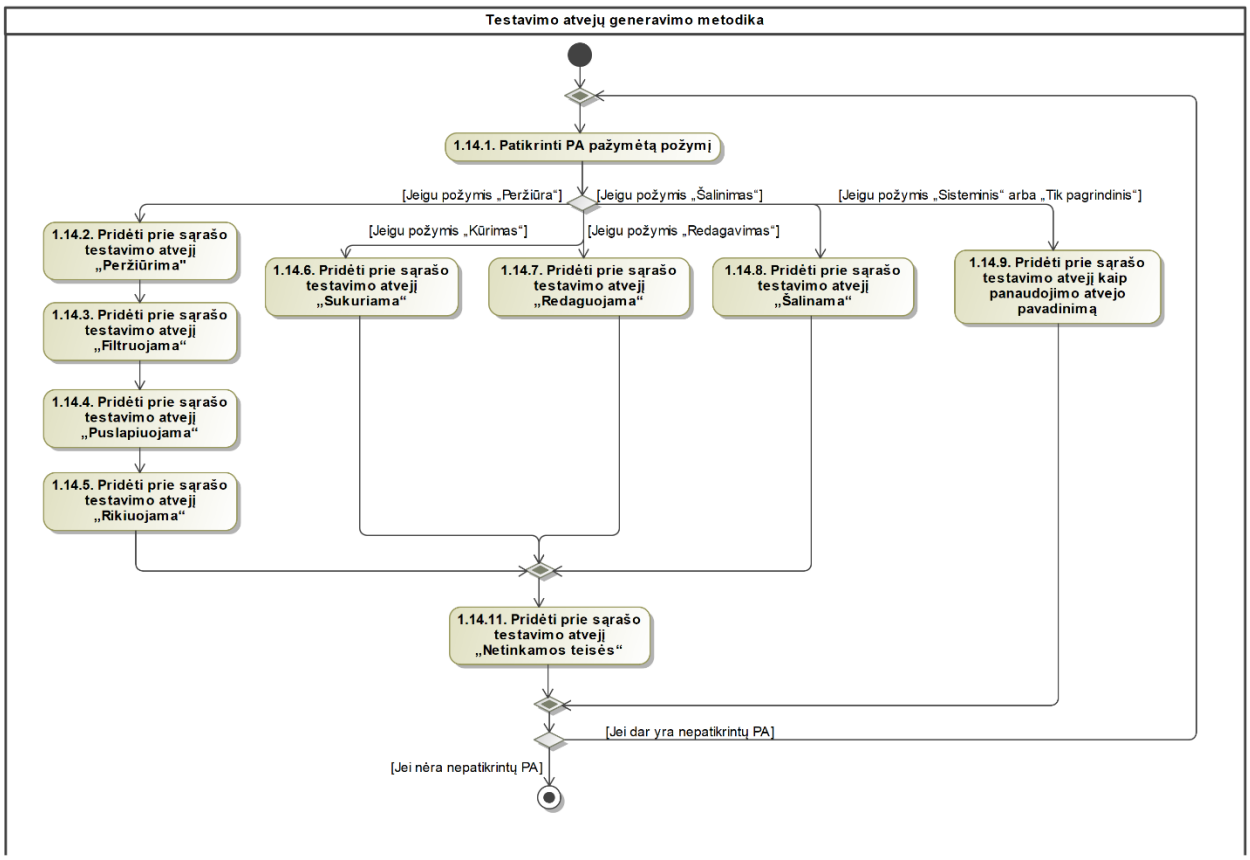
Procesų diagramos detalizuoja, kaip metodikoje apdorojami įvesties duomenys ir kaip iš specifikacijoje pateiktos informacijos formuojami standartizuoti testavimo atvejai. Taip pat diagramose parodoma, kaip atskiri procesai priklauso vieni nuo kitų. Tokiu būdu siekiama apibrėžti visą testavimo atvejų generavimo eigą bei jos ryšį su anksčiau aprašytais duomenų modeliais (žr. 2.2).

Žemiau pateikiamoje veiklos diagramoje pateikiamas veiksmų eiliškumas nuo įrankio atidarymo iki galimų testavimo atvejų sąrašo pateikimo (žr. 23 pav.). Ši veiklos diagrama detalizuoja panaudojimo atvejo „B2 Įkelti UML panaudojimo atvejų specifikacijos failą“ veiksmus.



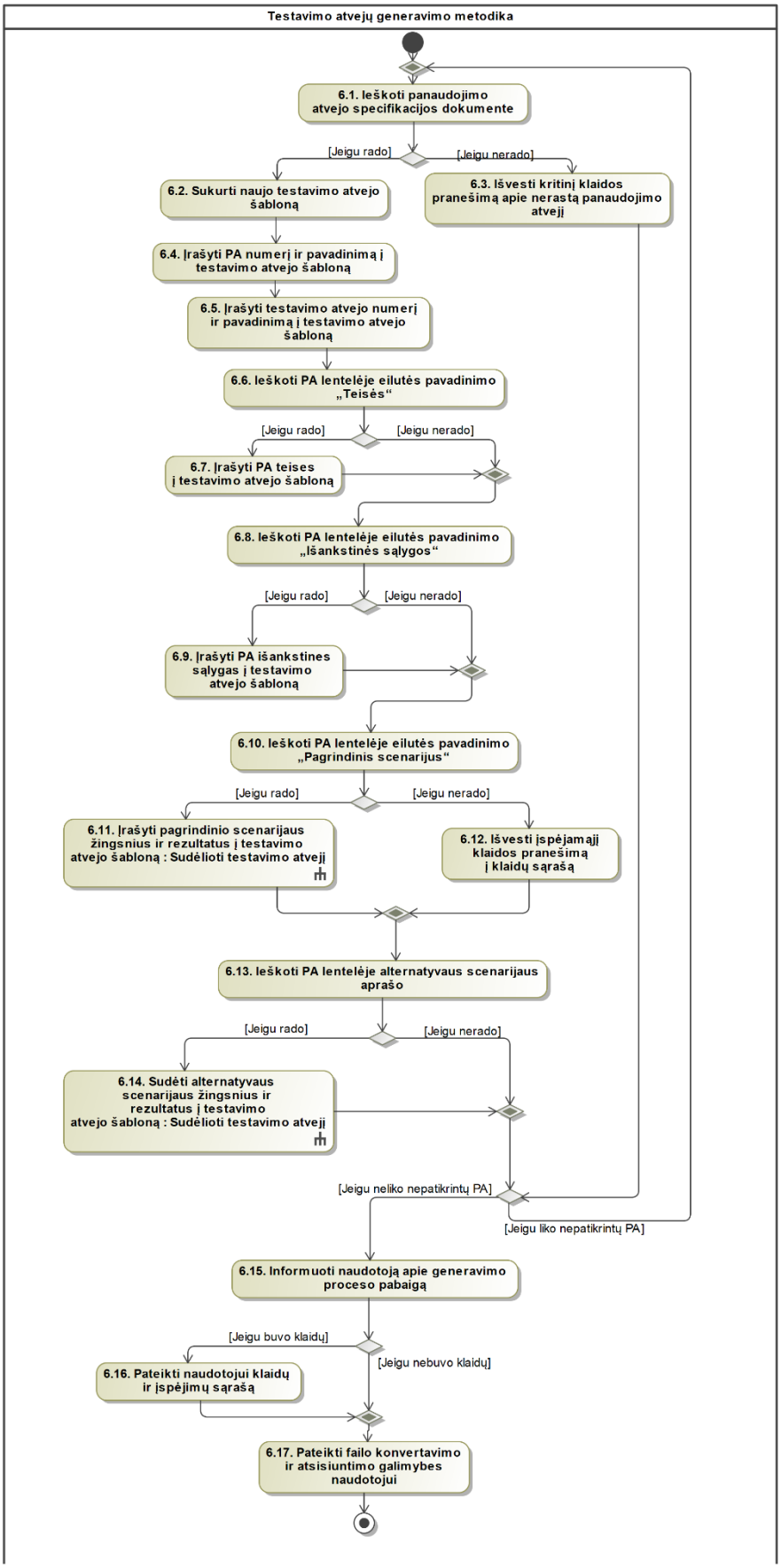
23 pav. Veiklos diagrama, parodanti failo įkėlimo ir pradinio apdorojimo procesą

Žemiau pateikiama veiklos diagrama atvaizduoja metodikos veiksmus sudarant galimų testavimo atvejų sąrašą naudotojui (žr. 24 pav.).



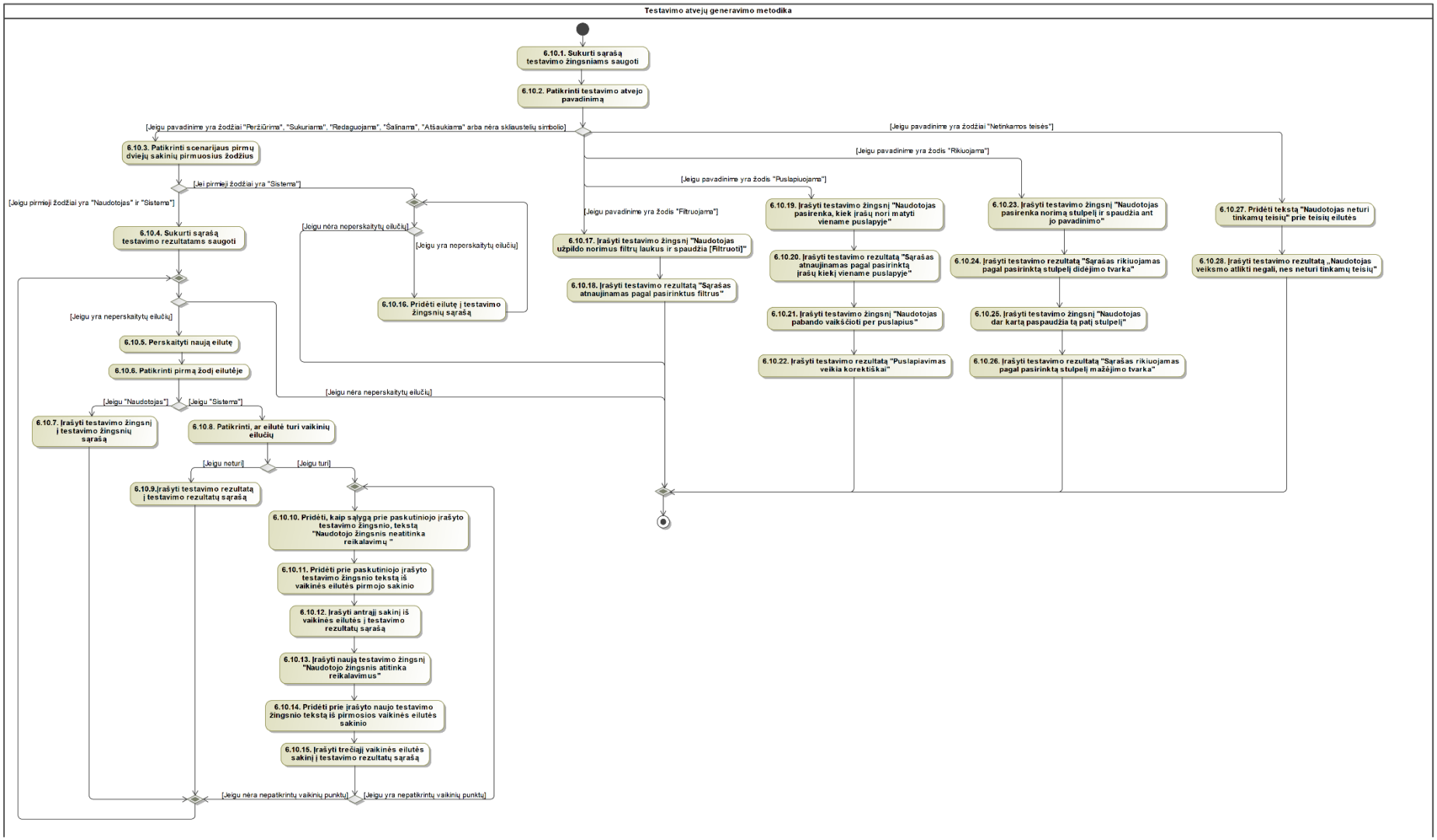
24 pav. Veiklos diagrama, sudaranti galimų sudaryti testavimo atvejų sąrašą pagal pasirinktus panaudojimo atvejų požymius

Žemiau pateikiama veiklos diagrama, kuri detalizuoja panaudojimo atvejį „B3 Generuoti testavimo atvejus“ (žr. 25 pav.). Šioje veiklos diagramoje pateikiamas išsamus pilno testavimo scenarijaus sudarymas su visais atributais ir duomenimis iš panaudojimo atvejo.



25 pav. Veiklos diagrama, kuri parodo, kaip sudaromas testavimo scenarijus su visais nustatytais atributais

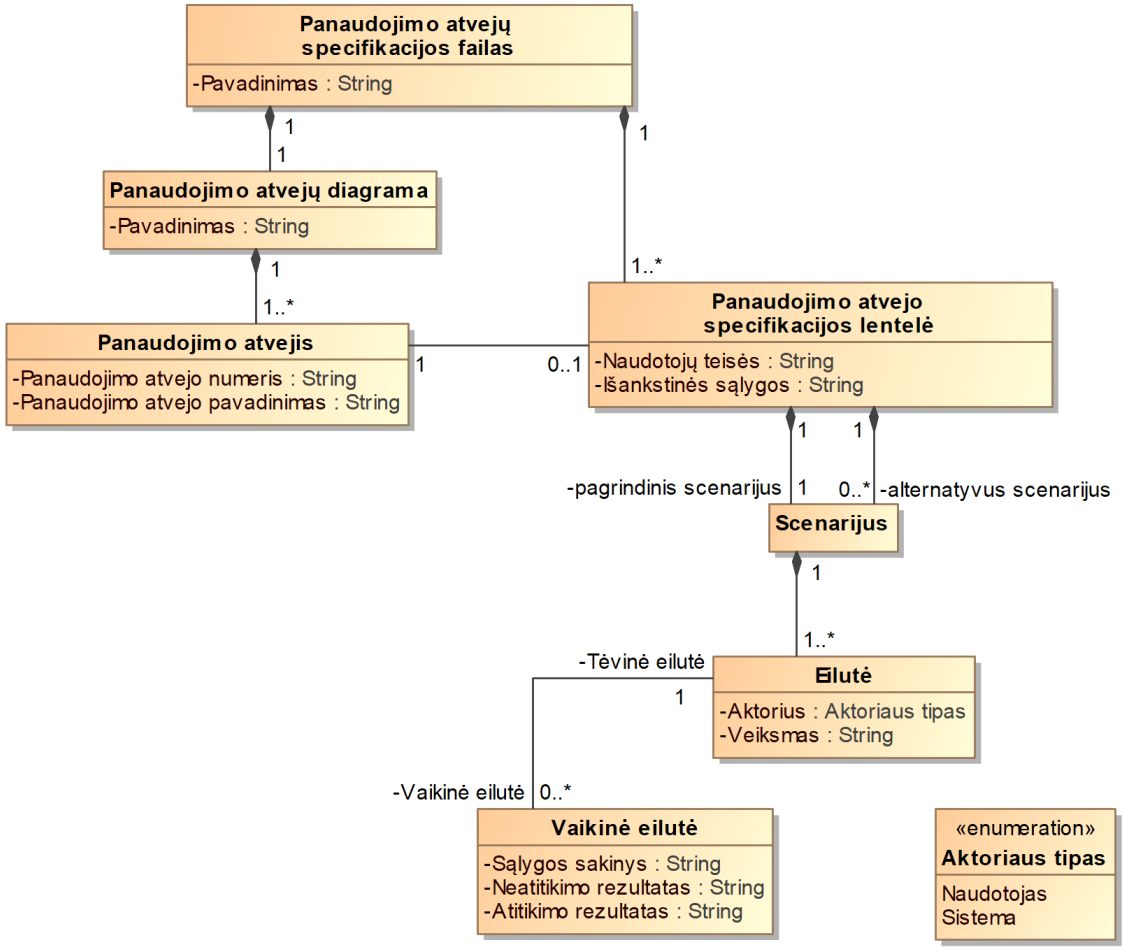
Žemiau pateikiama veiklos diagrama (žr. 26 pav.), kuri išsamiai pateikia vieno testavimo scenarijaus generavimo procesą. Procese atsižvelgiama į visas nustatytas testavimo scenarijų sudarymo taisykles bei naudotojo pažymėtus požymius.



26 pav. Veiklos diagrama, kuri parodo vieno testavimo scenarijaus žingsnių ir rezultatų sudarymo procesą

2.4.2. Testavimo atvejų generavimo metodikos įvesties duomenų modelis, šablonas ir taisyklės

Šiame skyriuje pateikiamas panaudojimo atvejų specifikacijos duomenų modelis, kuriame atvaizduojama informacija, pateikiama panaudojimo atvejų specifikacijos faile (žr. 27 pav.). Šioje klasių diagramoje atvaizduojama panaudojimo atvejų specifikacijos failo struktūra, kuris pateikiamas kaip įvesties duomenys testavimo atvejų generavimo metodikoje. Failas susideda iš vienos panaudojimo atvejų diagramos, kurioje išskirstyti panaudojimo atvejai. Kiekvienas panaudojimo atvejis gali turėti susietą specifikacijos lentelę su išsamesniu panaudojimo atvejo aprašymu. Specifikacijos lentelėje pateikiami scenarijai – vienas pagrindinis ir, jeigu reikia, neribotas skaičius alternatyviųjų. Scenarijus sudaro bent viena eilutė. Eilutėse nurodomas aktorius (naudotojas arba sistema) ir jo atliekamas veiksmas. Jeigu scenarijuje atliekami tikrinimai – eilutės skaidomos į antro lygio (vaikines) eilutes, jose detalizuojama tikrinimo sąlyga bei atitikimo ir neatitikimo rezultatai.



27 pav. Testavimo atvejų generavimo metodikos panaudojimo atvejų specifikacijos duomenų modelis

Siekiant gauti kuo kokybiškesnius rezultatus, buvo nustatytas turimas įkelti panaudojimo atvejų specifikacijos failo šablonas ir jam taikomos taisyklės. Tokiu būdu kuriama sistema struktūriškai atliks paiešką įkeltame faile ir galės sėkmingai apdoroti panaudojimo atvejų

specifikaciją ir sukurti testavimo atvejus. Įkeliamas panaudojimo atvejų specifikacijos failas turi atitikti šiuos reikalavimus:

1. Kiekvienas panaudojimo atvejis turėtų turėti savo atskirą lentelę su minimalia informacija:
2. Panaudojimo atvejo numeris ir pavadinimas; (PA.T1)
3. Naudotojų teisės;
4. Išankstinės sąlygos;
5. Pagrindinis scenarijus;
6. Panaudojimo atvejo specifikacija turi būti pateikiama lentelės formatu su pirmame punkte išvardinta minimalia informacija (žr. 28 pav.)

Panaudojimo atvejo numeris ir pavadinimas	{Panaudojimo atvejo numeris} {Panaudojimo atvejo pavadinimas}
Naudotojų teisės	{Naudotojo teisės}
Išankstinės sąlygos	{Išankstinės sąlygos}
Pagrindinis scenarijus	1. Naudotojas {naudotojo atliekami veiksmai}. 2. Sistema {sistemos atliekami veiksmai}. 3. Sistema patikrina: 3.1. {Sąlyga}. Jei neatitinka, {atliekami veiksmai, kai sąlyga netenkinama}.
Alternatyvus scenarijus nr. 1	1. Naudotojas {naudotojo atliekami veiksmai}. 2. Sistema {sistemos atliekami veiksmai}. 3. Sistema patikrina: 3.1. {Sąlyga}. Jei neatitinka, {atliekami veiksmai, kai sąlyga netenkinama}. Jei atitinka, {atliekami veiksmai, kai sąlyga tenkinama}.

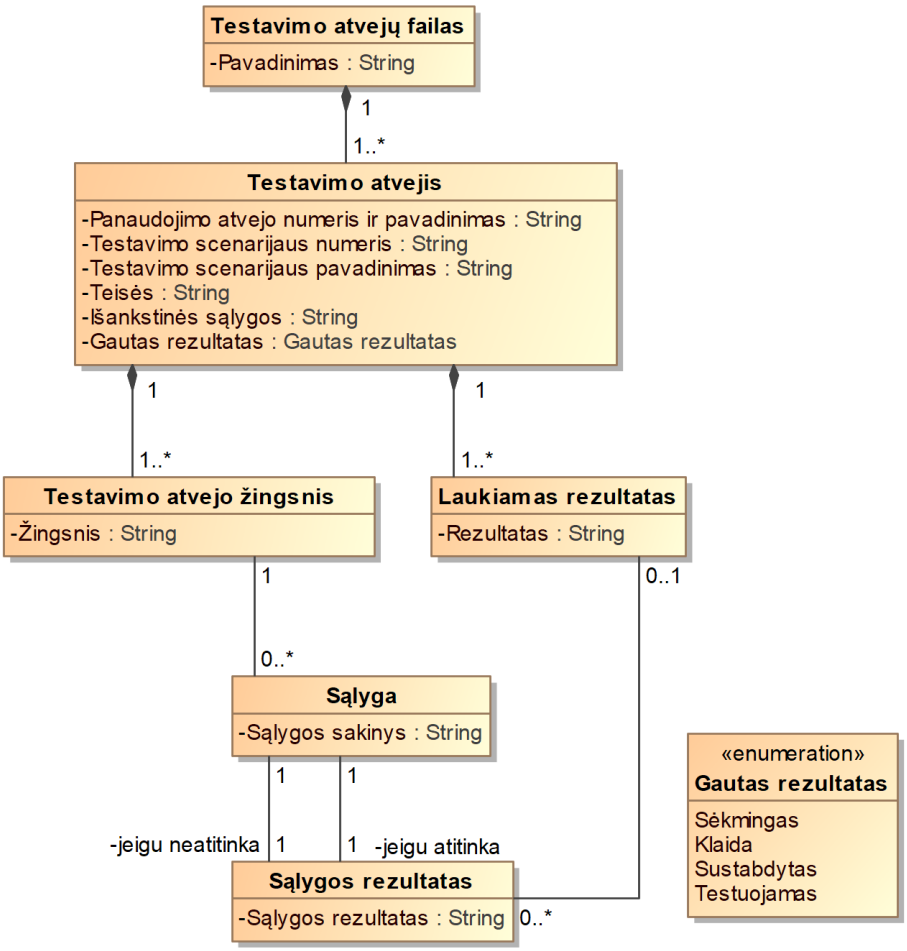
28 pav. Minimalus panaudojimo atvejo specifikacijos lentelės šablonas

7. Atliekamas naudotojo veiksmas/žingsnis pradedamas žodžiu „Naudotojas“.
8. Atliekamas sistemos veiksmas/žingsnis pradedamas žodžiu „Sistema“.
9. Jei žingsnyje atliekamas tikrinimas, tėvinėje eilutėje įrašoma „Sistema patikrina“, tolimesnėse vaikinėse eilutėse turi būti nurodoma sąlyga, po sąlygos nurodomas klaidos atveju atliekamas veiksmas žymimas sakinio pradžioje su „Jei neatitinka“, po neatitikimo rezultato nurodomas sėkmės atveju atliekamas veiksmas žymimas sakinio pradžioje su „Jei atitinka“.
10. Veiksmo atšaukimo scenarijus turi būti aprašomas kaip alternatyvusis scenarijus.
11. Panaudojimo atvejų specifikacijos faile esantis panaudojimo atvejų diagramos paveikslas turi turėti antraštę „Panaudojimo atvejų diagrama“.

2.4.3. Testavimo atvejų generavimo metodikos išvesties duomenų modelis, šablonas ir taisyklės

Šiame skyriuje pateikiamas testavimo atvejų failo duomenų modelis, kuriame atvaizduojama informacija, pateikiama sugeneruotuose testavimo atvejuose (žr. 29 pav.). Šioje klasių diagramoje pavaizduota testavimo atvejų generavimo metodikos sugeneruotų testavimo atvejų struktūra. Testavimo atvejų faile įrašomi visi sugeneruoti testavimo atvejai, kurių gali būti vienas arba daugiau. Kiekvienas testavimo atvejis susijęs su konkrečiu panaudojimo atveju ir jo informacija (teisėmis, išankstinėmis sąlygomis). Papildomai, testavimo atvejį

sudaro testavimo žingsniai ir laukiami rezultatai. Žingsnis gali būti sudarytas iš sąlygos, kuriai esant įrašomas atitinkamas sąlygos rezultatas (atitinka arba neatitinka).



29 pav. Testavimo atvejų generavimo metodikos testavimo atvejo duomenų modelis

Siekiant struktūriškai generuoti testavimo atvejus buvo sudarytas testavimo atvejo šablonas lentelės formatu, kuris bus taikomas visiems testavimo atvejams. Šablonas buvo sudarytas atsižvelgiant į naudotojo patogumą, todėl jame informacija pateikiama stulpeliais, t.y. atvirkščiai nei panaudojimo atvejo lentelė. Nustatytos testavimo atvejo sudarymo taisyklės:

1. Testavimo atvejo numeris sudaromas pagal šabloną: {{Panaudojimo atvejo numeris}TS{Testavimo atvejo eiliškumo numeris}}. Pvz.: Panaudojimo atvejo numeris: PA.KNG.1, tuomet pirmasis testavimo atvejis bus žymimas: PA.KNG.1TS1, o antrasis testavimo atvejis bus žymimas: PA.KNG.1TS2 ir t.t.
   Testavimo atvejo pavadinimas sudaromas pagal šabloną: {{Panaudojimo atvejo pavadinimas} ({Testuojamo funkcionalumo pavadinimas})}. Pvz.: Panaudojimo atvejo pavadinimas: „Peržiūrėti knygų sąrašą“, tai galimi sugeneruoti testavimo atvejai: „Peržiūrėti knygų sąrašą (peržiūrima)“, „Peržiūrėti knygų sąrašą (rikiuojama)“ ir t.t.
2. Testavimo atvejo lentelės šablonas pateikiamas lentelėje žemiau (žr. 30 pav.).

Panaudojimo atvejo numeris ir pavadinimas	Testavimo scenarijaus numeris	Testavimo scenarijaus pavadinimas	Teisės	Išankstinės sąlygos	Testavimo scenarijaus žingsniai	Laukiamas rezultatas	Gautas rezultatas
{Panaudojimo atvejo numeris ir pavadinimas}	*{Testavimo atvejo numeris sudaromas pagal panaudojimo atvejį, žr. taisykles}*	*{Testavimo atvejo pavadinimas sudaromas pagal panaudojimo atvejį, žr. taisykles}*	*{Teisės iš panaudojimo atvejo specifikacijos}*	*{Išankstinės sąlygos iš panaudojimo atvejo specifikacijos}*	*{Pirmas žingsnis, numeruojama}*	*{Pirmo žingsnio laukiamas rezultatas, numeruojama}*	*{Papildomas stulpelis žingsnio gautam rezultatui fiksuoti, pildoma testuotojo}*
					{Antras žingsnis, numeruojama}	*{Antro žingsnio laukiamas rezultatas, numeruojama}*	*{Papildomas stulpelis žingsnio gautam rezultatui fiksuoti, pildoma testuotojo}*
					{Trečias žingsnis, numeruojama}	*{Trečio žingsnio laukiamas rezultatas, numeruojama}*	*{Papildomas stulpelis žingsnio gautam rezultatui fiksuoti, pildoma testuotojo}*

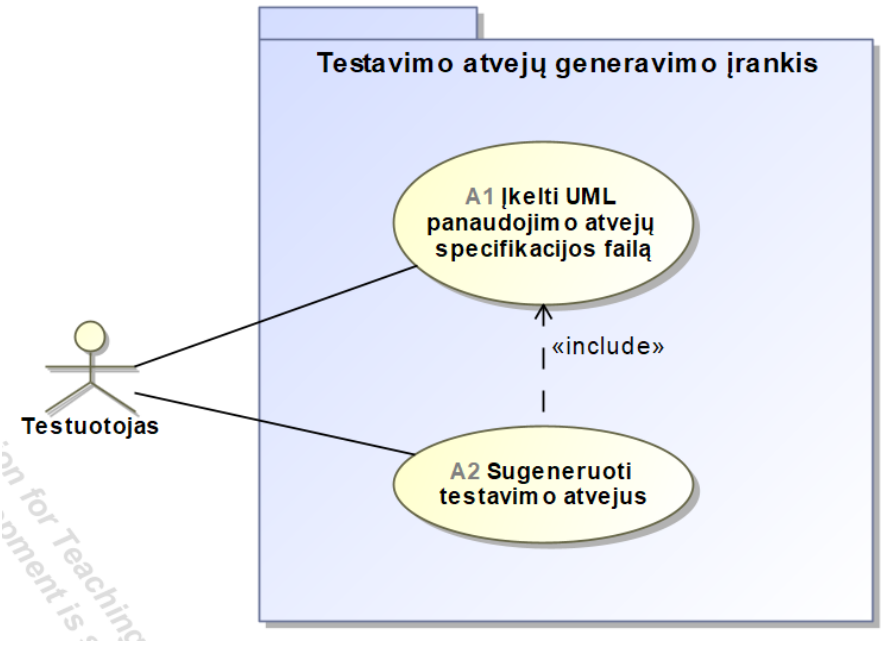
30 pav. Testavimo atvejo lentelės šablonas

Aprašytas testavimo atvejų šablonas ir sudarymo taisyklės leidžia visus sugeneruotus testavimo atvejus pateikti vienodu formatu. Toks informacijos pateikimas padeda matyti sąsają tarp testavimo atvejų ir panaudojimo atvejų bei užtikrina, kad visuose sugeneruotuose testavimo atvejuose būtų pateikiama reikalinga informacija.

3. Testavimo atvejų generavimo įrankio realizacijos projektas

3.1. Testavimo atvejų generavimo įrankio reikalavimai

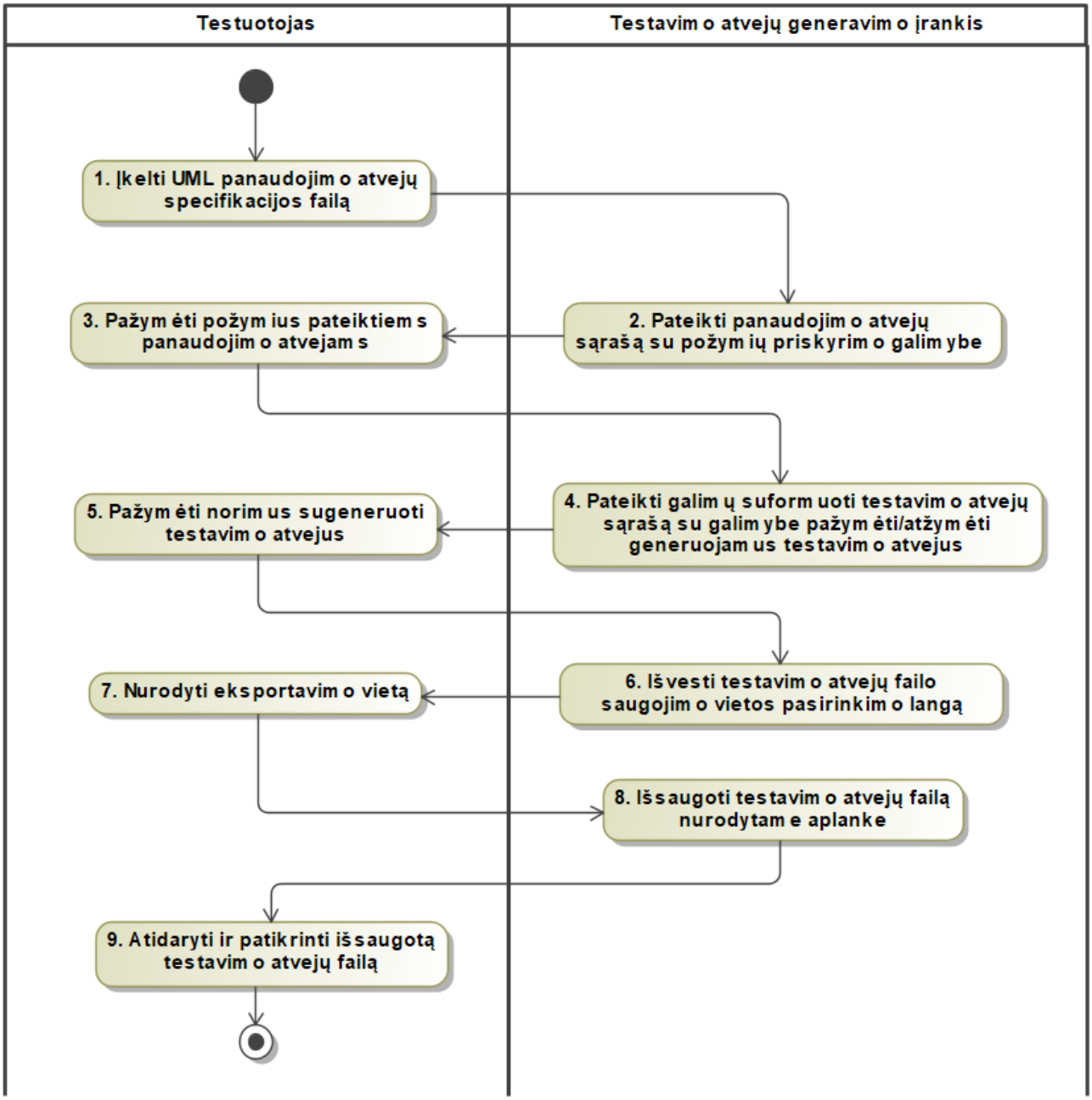
Įrankio naudotojo galimi atlikti veiksmai atvaizduoti panaudojimo atvejų diagramoje (žr. 31 pav.).



31 pav. Testavimo atvejų generavimo įrankio realizacijos panaudojimo atvejų diagrama

Pagrindiniai naudotojo atliekami veiksmai gali būti UML panaudojimo atvejų specifikacijos įkėlimas bei tolimesnis - testavimo atvejų generavimas, kurį sudaro panaudojimo atvejų skirstymas pagal požymius ir norimų sugeneruoti testavimo atvejų pasirinkimas, taip pat, testavimo atvejų generavimas galimas tik tada, kai naudotojas įkelia UML panaudojimo atvejų specifikacijos failą ir paspaudžia atitinkamą mygtuką.

Panaudojimo atvejis „Sugeneruoti testavimo atvejus“ plačiau aprašomas žemiau pateiktoje veiklos diagramoje (žr. 32 pav.).

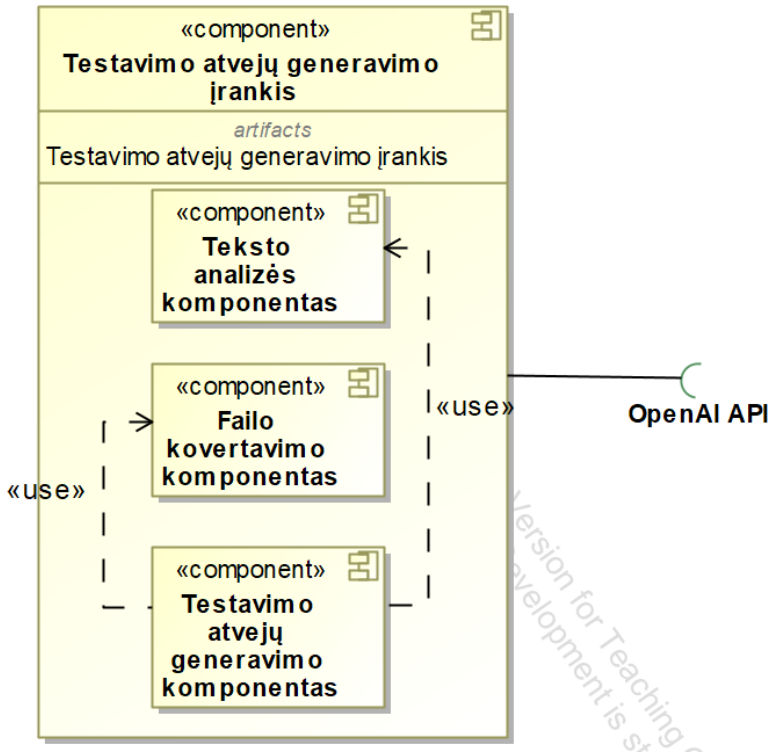


32 pav. Testavimo atvejų generavimo įrankio bendro proceso veiklos diagrama

Veiklos diagramoje atvaizduojami naudotojo (testuotojo) ir įrankio atliekami veiksmai pradedant naudotojo įkeliamu UML panaudojimo atvejų specifikacijos failu iki rezultatų eksportavimo.

3.2. Testavimo atvejų generacimo įrankio loginė architektūra ir diegimo principai

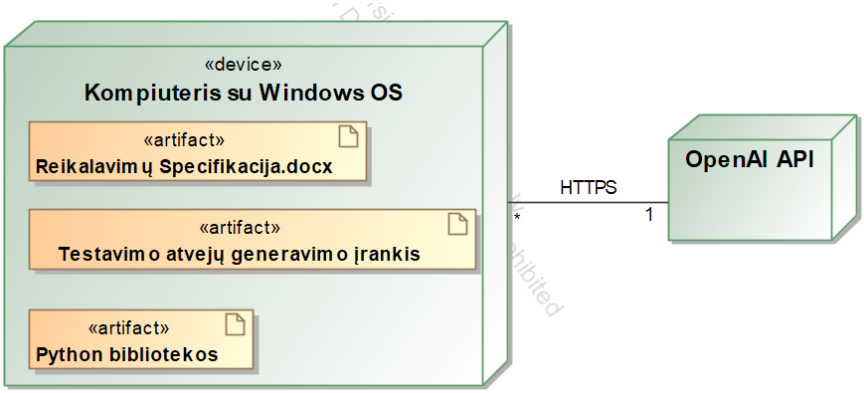
Šiame skyriuje pateikiama sukurto testavimo atvejų generavimo įrankio loginė architektūra bei pagrindiniai sistemos komponentai. Diagramoje pavaizduota, kaip įrankyje atliekamas pateikto dokumento apdorojimas, duomenų analizė ir testavimo atvejų generavimas. Taip pat parodyta, kaip sistemos komponentai tarpusavyje sąveikauja bei paminėtas„OpenAI API“ sprendimas. Loginės architektūros diagrama pateikiama žemiau esančiame paveikslėlyje (žr. 33 pav.).



33 pav. Testavimo atvejų generavimo įrankio loginės architektūros diagrama

Loginė architektūra suskirstyta į komponentus pagal pagrindines įrankio funkcines atsakomybes. Teksto analizės komponentas naudojamas pradiniams duomenims iš pateikto dokumento nuskaityti – jis apdoroja panaudojimo atvejų diagramą, identifikuoja joje esančius panaudojimo atvejus ir nuskaito specifikacijų lentelėse pateiktą informaciją. Testavimo atvejų generavimo komponentas naudoja šiuos duomenis ir pagal nustatytas taisykles formuoja testavimo atvejus: priskiria informaciją atitinkamiems laukams, atskiria scenarijaus žingsnius nuo laukiamų rezultatų bei sugeneruoja papildomus scenarijus pagal pasirinktus požymius. Failo konvertavimo komponentas atsakingas už galutinio rezultato sukūrimą „.xlsx“ formatu, kad sugeneruoti testavimo atvejai būtų pateikiami naudotojui tolimesniam naudojimui. „OpenAI API“ šioje architektūroje naudojamas kaip pagalbinė priemonė panaudojimo atvejų diagramoje esančiai informacijai atpažinti.

Taip pat buvo nubraižyta diegimo diagrama, kurioje parodyta, kokioje techninėje aplinkoje veikia sukurtas įrankis (žr. 34 pav.). Įrankis diegiamas ir naudojamas kompiuteryje su „Windows“ operacine sistema, kartu su reikalingomis „Python“ bibliotekomis. Naudotojas įrankiui pateikia reikalavimų specifikacijos „.docx“ failą, o sugeneruoti testavimo atvejai parengiami tolimesniam naudojimui. Kadangi panaudojimo atvejų diagramos informacijai atpažinti naudojamas „OpenAI API“, įrankis su šia išorine paslauga komunikuoja HTTPS ryšiu.



34 pav. Testavimo atvejų generavimo įrankio diegimo diagrama

Testavimo atvejų generavimo įrankis vykdomas naudotojo kompiuteryje su „Windows“ operacine sistema. Įrankio realizacijai pasirinkta „Python“ programavimo kalba, nes ji leidžia patogiai dirbti su tekstinių dokumentų apdorojimu, duomenų analizavimu ir failų generavimu. Projekto metu naudojamos papildomos bibliotekos „python-docx“ darbui su „.docx“ failais, „openpyxl“ rezultatų eksportavimui į „.xlsx“ formatą bei „customtkinter“ grafinės naudotojo sąsajos realizavimui.

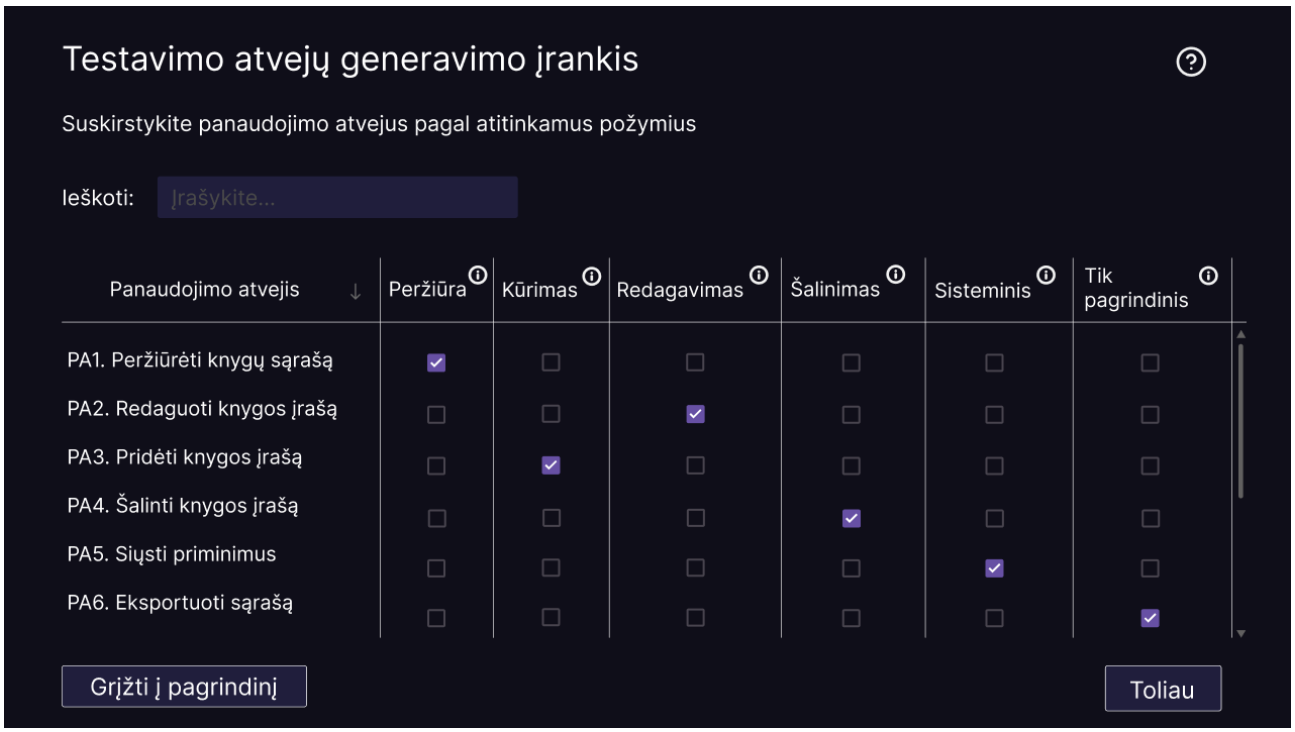
Panaudojimo atvejų diagramų informacijai nuskaityti naudojamas „OpenAI API“ sprendimas, leidžiantis atpažinti diagramoje esančius panaudojimo atvejus. Tokia technologijų kombinacija leidžia automatizuoti reikalavimų specifikacijų analizę ir sugeneruoti struktūrizuotus testavimo atvejus tolimesniam naudojimui.

3.3. Testavimo atvejų generavimo įrankio sąsajos prototipas

Prieš realizuojant galutinį įrankį, buvo sudarytas naudotojo sąsajos prototipas, skirtas geriau apgalvoti sistemos veikimo principus, naudotojo veiksmų eigą bei pagrindinius įrankio funkcionalumus. Šiame skyriuje pateikiamas sukurtas prototipas ir jo pagrindiniai sprendimai. Prototipas suskirstytas į kelis atskirus langus pagal pagrindinius testavimo atvejų generavimo etapus. Toks sprendimas pasirinktas siekiant aiškiai atskirti skirtingus naudotojo atliekamus veiksmus bei sumažinti vienu metu pateikiamos informacijos kiekį.

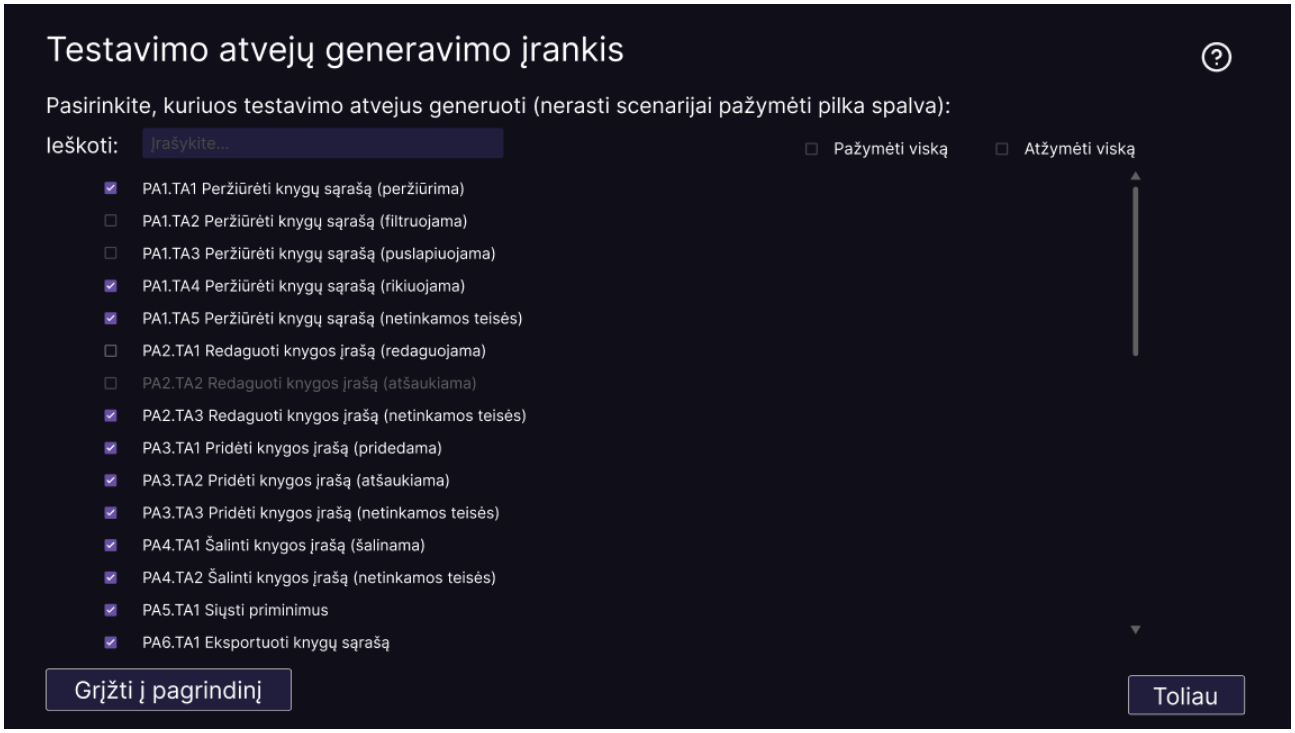
Pagrindiniame lange, naudotojas galės įkelti norimą reikalavimų specifikacijos failą (mygtukas [Įkelti reikalavimų specifikacijos failą]) ir peržiūrėti informaciją apie įrankį bei įkeliamo failo taisykles.

Sėkmingai įkėlus teisingo formato reikalavimų specifikacijos failą – pateikiamas panaudojimo atvejų požymių priskyrimo langas. Panaudojimo atvejų požymių priskyrimo lange informacija pateikiama lentelės forma, nes tai leidžia patogiau peržiūrėti didesnį panaudojimo atvejų kiekį ir greitai priskirti kiekvienam panaudojimo atvejui atitinkamą požymį. Taip pat papildomai integruota paieška ir informacinės etiketės prie požymių pavadinimų, kad naudotojas galėtų greičiau surasti reikiamą informaciją ir suprasti požymių paskirtį (žr. 35 pav.).



35 pav. Panaudojimo atvejų požymių priskyrimo lango eskizas

Pasirinkus mygtuką [Toliau] naudotojui suformuojamas galimų testavimo atvejų sąrašo langas. Šis langas išskirtas siekiant suteikti naudotojui galimybę prieš generavimą peržiūrėti visus siūlomus scenarijus ir pasirinkti tik aktualius testavimo atvejus. Kadangi kai kurie scenarijai gali būti pertekliniai arba netaikomi konkrečiam funkcionalumui, naudotojui paliekama galimybė valdyti galutinį generuojamų testavimo atvejų rinkinį (žr. 36 pav.).



36 pav. Testavimo atvejų pasirinkimo lango eskizas

Sugeneravus testavimo atvejus naudotojui pateikiamas langas su pranešimu „Pasirinkti testavimo atvejai sėkmingai sugeneruoti“. Šiame lange naudotojas gali pasirinkti eksportuojamo failo formatą, atsisiųsti failą (mygtukas [Atsisiųsti]) arba nutraukti procesą ir grįžti į pagrindinį langą (mygtukas [Grįžti į pagrindinį]).

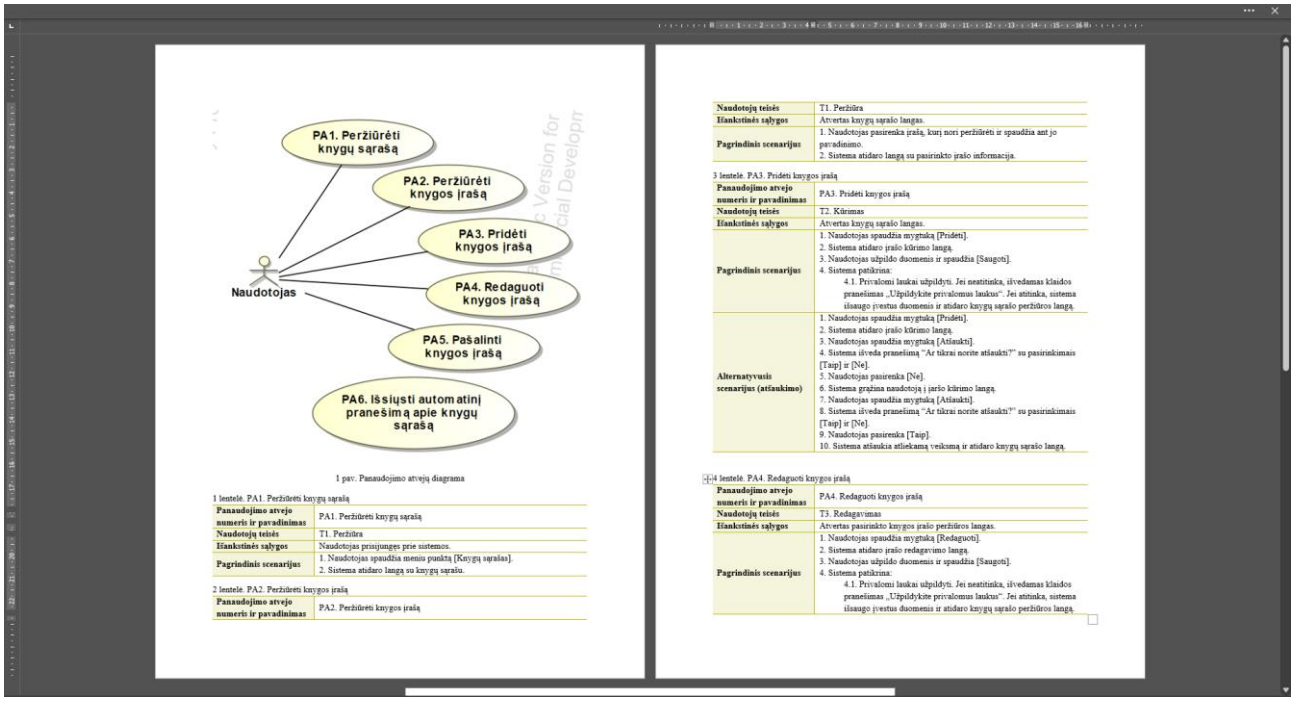
Realizuojant įrankį buvo pritaikyti pagrindiniai prototipe numatyti sąsajos sprendimai – informacijos suskaidymas į atskirus langus pagal generavimo etapus, panaudojimo atvejų pateikimas lentelėmis bei galimybė naudotojui pasirinkti generuojamus scenarijus. Toks sąsajos išdėstymas pasiteisino praktinio realizavimo metu, nes leido aiškiau organizuoti testavimo atvejų generavimo procesą ir sumažino vienu metu pateikiamos informacijos kiekį.

4. Testavimo atvejų generavimo įrankio realizacija ir testavimas

Testavimo atvejų generavimo įrankis sukurtas kaip darbalaukio sprendimas, skirtas testavimo atvejų generavimui pagal reikalavimų specifikacijos dokumentus. Įrankio realizacijai pasirinkta *Python* programavimo kalba, o kūrimo darbams naudojama *Visual Studio Code* integruota kūrimo aplinka. Sprendime taikomos *Python* bibliotekos, leidžiančios apdoroti dokumentus (naudojama *python-docx* biblioteka), struktūrizuoti duomenis (naudojama *openpyxl* biblioteka) ir realizuoti grafinę naudotojo sąsają (naudojama *tkinter* biblioteka**)**. Įrankio veikimas pagrįstas tekstinės informacijos apdorojimu ir atpažinimu (panaudojimo atvejams iš diagramos perskaityti naudojamas *OpenAI API*).

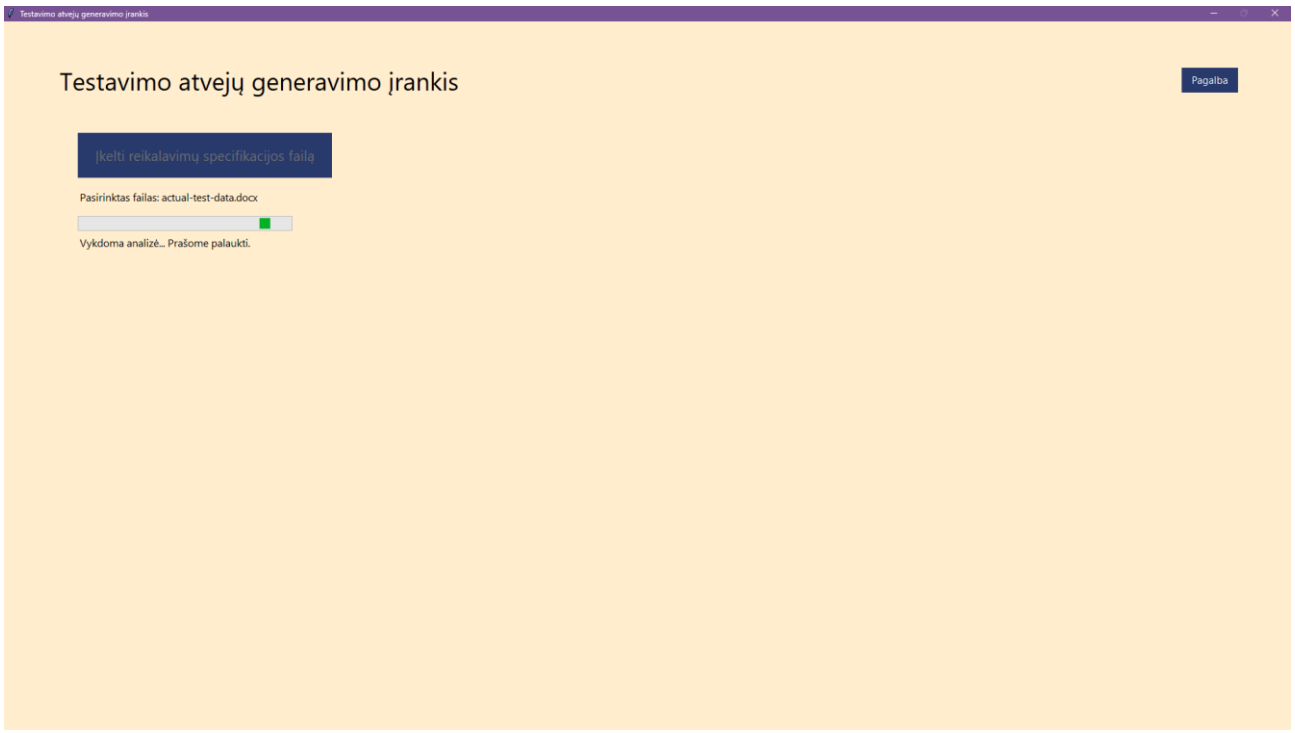
4.1. Testavimo atvejų generavimo įrankio realizacijos ir veikimo aprašas

Testavimo atvejų generavimo įrankis veikia nuosekliu etapiniu principu, kuriame kiekvienas sistemos veiksmas priklauso nuo ankstesnio etapo rezultatų. Darbui su įrankiu reikalingas panaudojimo atvejų specifikacijos failas, kuriame būtų bent minimali reikalavimus atitinkanti informacija (žr. 2.4.2 Testavimo atvejų generavimo metodikos įvesties duomenų modelis, šablonas ir taisyklės). Galimas duomenų failo pavyzdys pateiktas žemiau esančiame paveikslėlyje (žr. 37 pav.).



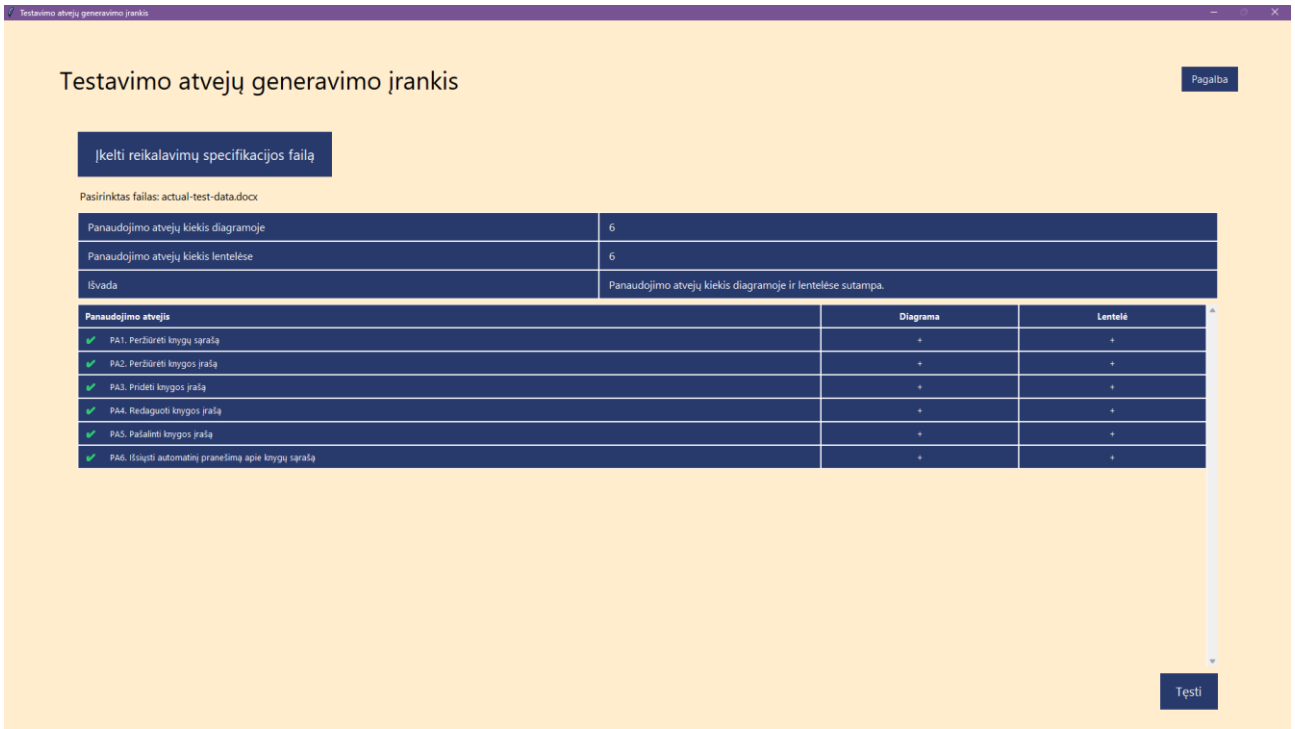
37 pav. Galimo panaudojimo atvejų specifikacijos failo pavyzdys

Pradiniame etape naudotojas įkelia reikalavimų specifikacijos dokumentą, kurį sistema validuoja ir analizuoja, identifikuodama panaudojimo atvejų diagramą bei lentelėse aprašytą testavimui reikalingą informaciją. Naudotojas apie atliekamos analizės procesą informuojamas progreso juostelėje (žr. 38 pav.).



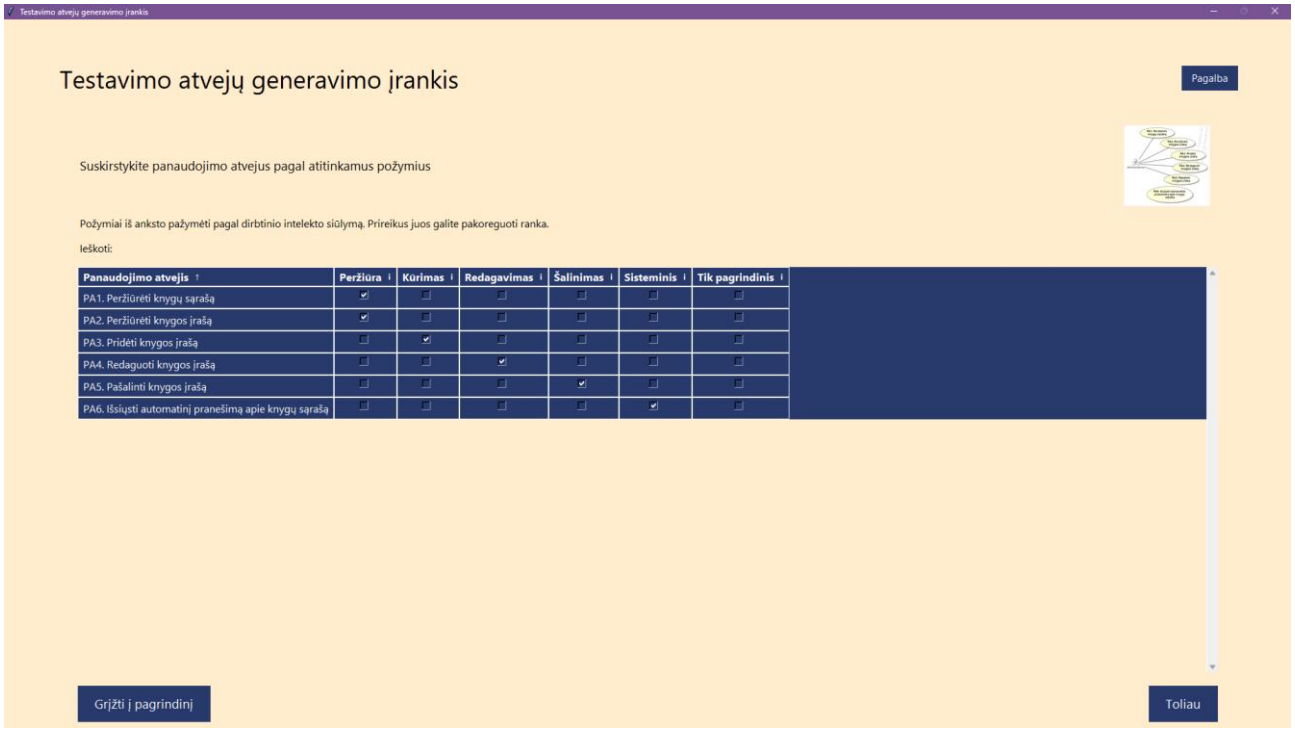
38 pav. Pradinis įrankio langas

Atlikus pirminę analizę, pateikiamos dvi lentelės, pirmoji lentelė parodo, kiek panaudojimo atvejų buvo rasta diagramoje, ir kiek lentelėse, taip pat pateikia išvadą, kuri padeda suprasti, ar specifikacijoje pateikti duomenys yra korektiški. Antroje lentelėje pateikiami visi rasti panaudojimo atvejai (žr. 39 pav.).



39 pav. Atliktos pirminės analizės rezultatų pateikimo langas

Sėkmingai apdorojus duomenis, sistema išskiria panaudojimo atvejus ir pateikia juos naudotojui klasifikavimui pagal iš anksto apibrėžtus požymius, tokius kaip peržiūra, kūrimas, redagavimas, šalinimas, sisteminis veikimas arba tik pagrindinis scenarijus (žr. 40 pav.).



40 pav. Panaudojimo atvejų požymių pasirinkimo langas

Toliau, remiantis naudotojo atlikta klasifikacija, sistema automatiškai sugeneruoja galimų testavimo atvejų sąrašą, apimantį pagrindinius, alternatyvius bei netinkamų teisių scenarijus. Šiame etape naudotojas gali pasirinkti, kurie testavimo atvejai bus generuojami, taip užtikrinant lankstų ir tikslinį testavimo apimties valdymą (žr. 41 pav.).

41 pav. Testavimo atvejų pasirinkimo langas

Galutiniame etape sistema struktūrizuoja pasirinktus testavimo atvejus pagal specifikacijos duomenis ir suformuoja testavimo atvejų dokumentą, kuris išsaugomas naudotojo kompiuteryje. Sugeneruotų testavimo atvejų failo pavyzdys pateikiamas žemiau esančiame paveikslėlyje (žr. 42 pav.).

4.2. Didžiųjų kalbų modelių taikymas (angl. Large Language model) teksto atpažinimui

Siekiant realizuoti automatinį panaudojimo atvejų (PA) teksto nuskaitymo funkcionalumą iš diagramos, buvo išbandyti kelių lokaliai veikiančių LLM (angl. *Large Language Model*) modeliai. Buvo atsižvelgta ir įvertinta, ar modeliai geba tinkamai interpretuoti iš optinio simbolių atpažinimo (angl. *Optical Character Recognition*) gautus duomenis.

Buvo išbandyti šie modeliai: Qwen2.5:72B, LLaVA, LLaMA3.2-Vision ir Gemma3-Vision. Visi modeliai buvo vykdomi lokaliai, be interneto prieigos, siekiant užtikrinti duomenų konfidencialumą. Testavimo metu nustatyta, kad nė vienas modelis nesugebėjo tinkamai apdoroti lietuviško teksto – lietuviškos raidės buvo iškraipomos, o gautos išvestys – neinformatyvios arba neįskaitomos.

Atsižvelgiant į išbandytų įrankių pateiktus rezultatus, nuspręsta naudoti *OpenAI* sprendimą, pasižymintį geresniu įvairių kalbų, įskaitant lietuvių, palaikymu. Pasirinktas modelis veikia su *zero data retention* funkcija, kuri užtikrina, kad įkelti duomenys nėra saugomi ir po apdorojimo automatiškai ištrinami. Tai leidžia išlaikyti aukštą duomenų saugumo lygį ir atitikti konfidencialumo reikalavimus.

Sprendime integruota *OpenAI API*, kuri naudojama perduodant diagramos paveikslėlį bei specialiai suformuotus užklausų šablonus (angl. prompts). Modelis, gavęs paveikslėlį,

analizuoja diagramoje esantį tekstą ir grąžina atpažintų panaudojimo atvejų sąrašą JSON formatu.

Siekiant užtikrinti stabilesnius rezultatus, užklausose modeliui pateikiamos papildomos taisyklės, apibrėžiančios atsakymo struktūrą, leidžiamą formatą bei reikalavimą grąžinti tik diagramoje esančius panaudojimo atvejus. Naudoti užklausų šablonai pateikiami žemiau esančioje lentelėje (žr. 3 lentelė).

3 lentelė. Programiniame kode naudotų užklausų šablonai siunčiami į OpenAI API

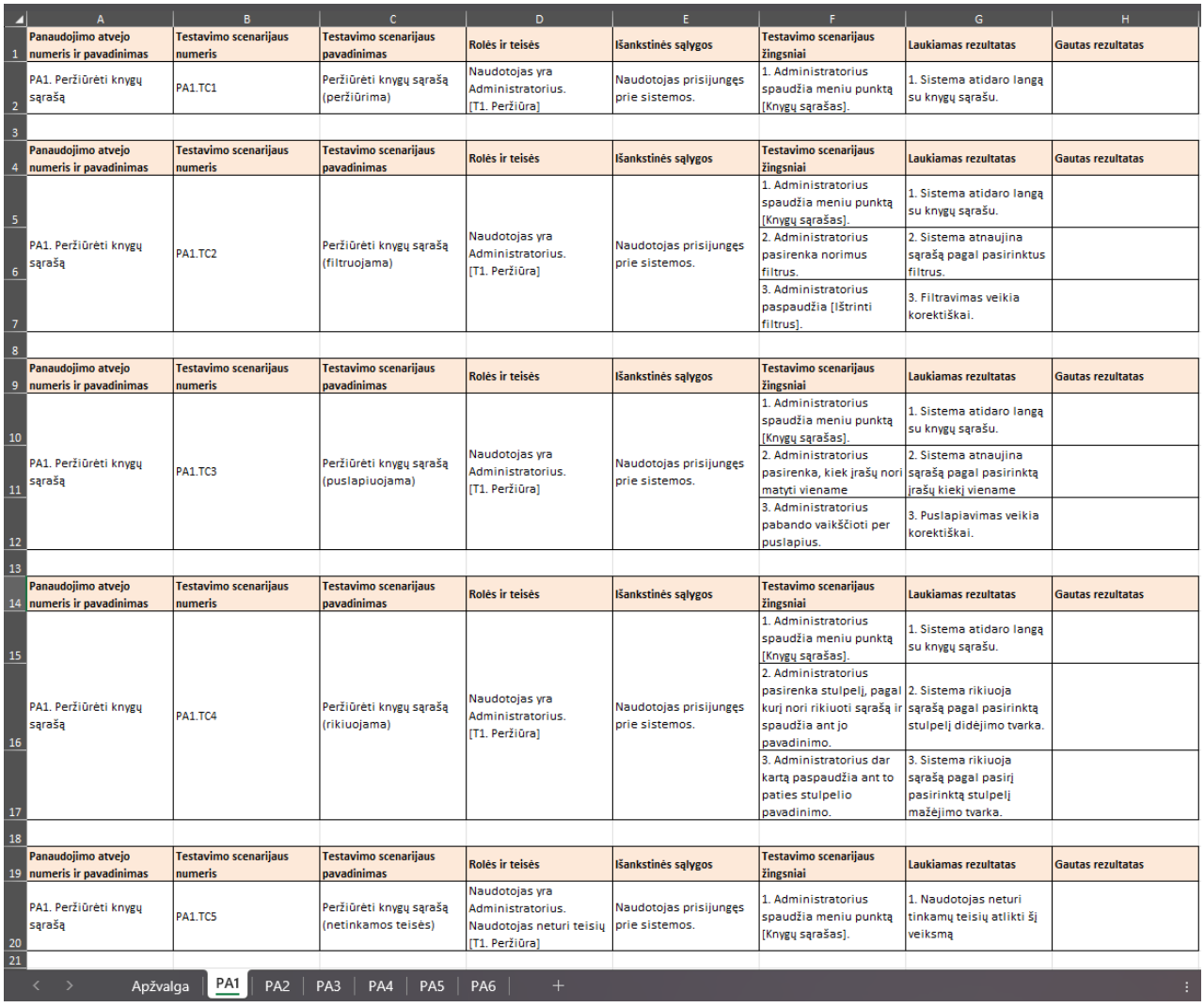
Naudota užklausa
Nuskaityk UML panaudojimo atvejų diagramą ir grąžink visus matomus panaudojimo atvejus JSON formatu. Kiekvienam panaudojimo atvejui nustatyk pavadinimą ir VISUS su juo tiesiogiai susietus aktorius iš diagramos. Jei aktorių diagramoje nėra arba jų nustatyti nepavyksta, grąžink tuščią masyvą 'actors': [] ir nepildyk 'actor' lauko. Neprigalvok aktorių, kurių diagramoje nesimato. Jei aktorius yra vienas, gali grąžinti ir lauką 'actor', ir lauką 'actors'. Jei aktorių yra keli, lauką 'actors' grąžink kaip masyvą su visais aktoriais. Atsakyk TIK grynu JSON, be jokio papildomo teksto, tokiu formatu: {"use_cases":[{"id":1,"name":"PA1. ...","actors":["Administratorius","Vadybininkas"]}, {"id":2,"name":"PA2. ...","actors":[]}]} Naudok lietuviškus pavadinimus. 'id' numeruok nuo 1 iš eilės. Aktorių varduose pateik tik aktoriaus pavadinimą.
Tu klasifikuoji UML panaudojimo atvejus į VIENĄ iš šių kategorijų: perziura, kurimas, redagavimas, salinimas, sisteminis, tik_pagrindinis. Taisyklės: - perziura: peržiūra, atidarymas, sąrašo rodymas, paieška, filtravimas, rikiavimas. - kurimas: naujo objekto sukūrimas, pridėjimas, registravimas. - redagavimas: esamo objekto keitimas, atnaujinimas, redagavimas. - salinimas: šalinimas, trynimas, pašalinimas. - sisteminis: foniniai, automatiniai ar sistemos atliekami veiksmai be tiesioginio naudotojo įvedimo. - tik_pagrindinis: kai labiausiai tinka tik pagrindinio scenarijaus generavimas ir kitos kategorijos netinka. Kiekvienam panaudojimo atvejui grąžink tik vieną geriausiai tinkamą kategoriją. Atsakyk TIK JSON formatu: {"suggestions":[{"id":1,"name":"...","category":"perziura"}]} Leidžiamos category reikšmės: perziura, kurimas, redagavimas, salinimas, sisteminis, tik_pagrindinis.

Su pirmąja užklausa, kuri paminėta lentelėje (žr. **3** lentelė) modeliui siunčiamas diagramos paveikslėlis ir prašoma grąžinti visus matomus panaudojimo atvejus bei su jais susietus aktorius JSON formatu. Jeigu aktorius nėra nurodytas diagramoje, tuomet testavimo atvejyje rolė nėra nurodoma ir įrašomas simbolis „–“.

Antroji užklausa modeliui siunčia panaudojimo atvejų sąrašas su specifikacijos duomenimis ir prašo kiekvieną PA priskirti vienai kategorijai: peržiūra, kūrimas, redagavimas, šalinimas, sisteminis arba tik pagrindinis.

Realizacijos eigoje, atliekant pirmuosius testus, pastebėta, kad pirminės analizės metu modelis kai kuriais atvejais tą patį panaudojimo atvejį perskaitydavo kelis kartus dėl panašių simbolių interpretavimo. Dažniausiai pasitaikydavo lietuviškų raidžių atpažinimo netikslumų, pavyzdžiui, raidė „Į“ būdavo interpretuojama kaip „J“ arba „I“. Dėl šios priežasties sprendime papildomai realizuotas pakartotinis rezultatų tikrinimo etapas po pirminės analizės. Jo metu visi panaudojimo atvejų pavadinimai buvo normalizuojami – tekstas paverčiamas

mažosiomis raidėmis, pašalinami pertekliniai tarpai, specialieji simboliai bei suvienodinamos dažniausiai neteisingai atpažįstamos lietuviškos raidės. Po normalizavimo panaudojimo atvejų pavadinimai buvo lyginami tarpusavyje, tikrinant, ar skiriasi tik pavieniai simboliai, lietuviškos raidės ar nereikšmingos žodžių formos. Tokiais atvejais įrašai buvo laikomi tuo pačiu panaudojimo atveju ir paliekamas tik vienas įrašas. Taikytas papildomas tikrinimas leido sumažinti pasikartojančių panaudojimo atvejų kiekį bei pagerinti bendrą diagramos analizės tikslumą.



Panaudojimo atvejo numeris ir pavadinimas	Testavimo scenarijaus numeris	Testavimo scenarijaus pavadinimas	Rolės ir teisės	Išankstinės sąlygos	Testavimo scenarijaus žingsniai	Laukiamas rezultatas	Gautas rezultatas
PA1. Peržiūrėti knygų sąrašą	PA1.TC1	Peržiūrėti knygų sąrašą (peržiūrima)	Naudotojas yra Administratorius. [T1. Peržiūra]	Naudotojas prisijungęs prie sistemos.	1. Administratorius spaudžia meniu punktą [Knygų sąrašas].	1. Sistema atidaro langą su knygų sąrašu.	

Panaudojimo atvejo numeris ir pavadinimas	Testavimo scenarijaus numeris	Testavimo scenarijaus pavadinimas	Rolės ir teisės	Išankstinės sąlygos	Testavimo scenarijaus žingsniai	Laukiamas rezultatas	Gautas rezultatas
PA1. Peržiūrėti knygų sąrašą	PA1.TC2	Peržiūrėti knygų sąrašą (filtruojama)	Naudotojas yra Administratorius. [T1. Peržiūra]	Naudotojas prisijungęs prie sistemos.	1. Administratorius spaudžia meniu punktą [Knygų sąrašas].	1. Sistema atidaro langą su knygų sąrašu.	
					2. Administratorius pasirenka norimus filtrus.	2. Sistema atnaujina sąrašą pagal pasirinktus filtrus.	
					3. Administratorius paspaudžia [Ištrinti filtrus].	3. Filtravimas veikia korektiškai.	

Panaudojimo atvejo numeris ir pavadinimas	Testavimo scenarijaus numeris	Testavimo scenarijaus pavadinimas	Rolės ir teisės	Išankstinės sąlygos	Testavimo scenarijaus žingsniai	Laukiamas rezultatas	Gautas rezultatas
PA1. Peržiūrėti knygų sąrašą	PA1.TC3	Peržiūrėti knygų sąrašą (puslapiuojama)	Naudotojas yra Administratorius. [T1. Peržiūra]	Naudotojas prisijungęs prie sistemos.	1. Administratorius spaudžia meniu punktą [Knygų sąrašas].	1. Sistema atidaro langą su knygų sąrašu.	
					2. Administratorius pasirenka, kiek įrašų nori matyti viename	2. Sistema atnaujina sąrašą pagal pasirinktą įrašų kiekį viename	
					3. Administratorius pabando vaikščioti per puslapius.	3. Puslapiavimas veikia korektiškai.	

Panaudojimo atvejo numeris ir pavadinimas	Testavimo scenarijaus numeris	Testavimo scenarijaus pavadinimas	Rolės ir teisės	Išankstinės sąlygos	Testavimo scenarijaus žingsniai	Laukiamas rezultatas	Gautas rezultatas
PA1. Peržiūrėti knygų sąrašą	PA1.TC4	Peržiūrėti knygų sąrašą (rikiuojama)	Naudotojas yra Administratorius. [T1. Peržiūra]	Naudotojas prisijungęs prie sistemos.	1. Administratorius spaudžia meniu punktą [Knygų sąrašas].	1. Sistema atidaro langą su knygų sąrašu.	
					2. Administratorius pasirenka stulpelį, pagal kurį nori rikiuoti sąrašą ir spaudžia ant jo pavadinimo.	2. Sistema rikiuoja sąrašą pagal pasirinktą stulpelį didėjimo tvarka.	
					3. Administratorius dar kartą paspaudžia ant to paties stulpelio pavadinimo.	3. Sistema rikiuoja sąrašą pagal pasirį pasirinktą stulpelį mažėjimo tvarka.	

Panaudojimo atvejo numeris ir pavadinimas	Testavimo scenarijaus numeris	Testavimo scenarijaus pavadinimas	Rolės ir teisės	Išankstinės sąlygos	Testavimo scenarijaus žingsniai	Laukiamas rezultatas	Gautas rezultatas
PA1. Peržiūrėti knygų sąrašą	PA1.TC5	Peržiūrėti knygų sąrašą (netinkamos teisės)	Naudotojas yra Administratorius. Naudotojas neturi teisių [T1. Peržiūra]	Naudotojas prisijungęs prie sistemos.	1. Administratorius spaudžia meniu punktą [Knygų sąrašas].	1. Naudotojas neturi tinkamų teisių atlikti šį veiksmą	

Apžvalga | PA1 | PA2 | PA3 | PA4 | PA5 | PA6

42 pav. Galimo sugeneruotų testavimo atvejų failo pavyzdys

Sugeneruotų testavimo atvejų turinys tiesiogiai priklauso nuo įkeltos specifikacijos kokybės, todėl sistema užtikrina, kad generuojami testavimo scenarijai atitiktų pateiktus reikalavimus ir būtų nuoseklūs visame testavimo procese.

4.3. Testavimo atvejų generavimo įrankio testavimo modelis, duomenys, rezultatai

Testavimo atvejų generavimo įrankio testavimui numatyta naudoti kelis skirtingus duomenų failus, apimančius įvairius panaudojimo scenarijus ir leidžiančius patikrinti visus testavimo atvejų generavimo mechanizmus. Testavimo metu reikalavimų specifikacijos failai yra įkeliami į įrankį, kuris juos apdoroja pagal naudotojo pasirinktus požymius ir sugeneruoja

testavimo atvejų rinkinį, išsaugomą naudotojo kompiuteryje. Sugeneruoti testavimo atvejai yra vertinami pagal jų atitikimą pateiktai specifikacijai.

4.3.1. Testavimo atvejų generavimo įrankio testavimo duomenys

Testavimo atvejų generavimo įrankiui testuoti naudojami failai su skirtingais duomenų rinkiniais, kurie buvo nustatyti pagal galimų testavimo atvejų sąrašo sudarymo veiklos diagramą (žr. 24 pav.). Testavimo duomenys apima tokius atvejus:

1. Visa dokumente pateikta informacija yra tiksli ir iš jos galima suformuoti testavimo atvejus
2. Dokumente pateikta ne pilna informacija (pvz., nėra PA diagramos, PA lentelių)
3. Tuščias dokumentas

4 lentelė. Testavimo duomenų struktūra

Projekto pavadinimas	**Testavimo atvejų skaičius**	**Padengiami panaudojimo atvejų požymiai**	**Papildomos pastabos**
Bankomato sistema	6	„Tik pagrindinis“ ir „Sisteminis“	
Knygų parduotuvė	6	„Peržiūra“, „Kūrimas“, „Redagavimas“, „Šalinimas“ ir „Sisteminis“.	Pridedamas alternatyvusis (atšaukimo) scenarijus.
Studentų informacinė sistema	5	„Peržiūra“ ir „Tik pagrindinis“	

Naudojant tokio tipo testavimo duomenis buvo siekiama patikrinti, ar testavimo atvejų generavimo įrankis korektiškai veikia skirtingose situacijose bei teisingai apdoroja įvairių tipų panaudojimo atvejų požymius.

4.3.2. Testavimo atvejų generavimo įrankio testavimo rezultatai

Testavimo metu sugeneruojami skirtingų požymių testavimo atvejai. „Tik pagrindinis“ požymis suformuoja tik pagrindinį scenarijų, kuris pateikiamas panaudojimo atvejo lentelėje, suformuotas testavimo atvejis turi pagrindinę informaciją be jokių papildomų scenarijų (žr. 43 pav.).

Panaudojimo atvejo numeris ir pavadinimas	Testavimo scenarijaus numeris	Testavimo scenarijaus pavadinimas	Teisės	Išankstinės sąlygos	Testavimo scenarijaus žingsniai	Laukiamas rezultatas	Gautas rezultatas
PA2. Įnešti pinigų	PA2.TC1	Įnešti pinigų	[T2. Operacijos]	Klientas autentifikuotas, bankomatas veikia.	1. Naudotojas pasirenka [Įnešti pinigų].	1. Sistema atidaro pinigų įnešimo modulį.	
					2. Naudotojas įdeda grynuosius pinigus.	2. Sistema patikrina sumą.	
					3. Naudotojas patvirtina operaciją.	3. Sistema atnaujina likutį ir pateikia patvirtinimą.	

43 pav. Sugeneruoto testavimo atvejo, pagal tipą „Tik pagrindinis“, pavyzdys

„Sisteminis“ testavimo atvejis suformuojamas suliejant testavimo atvejo žingsnio ir laukiamo rezultato langelius ir įrašant tik sistemos žingsnius (žr. 44 pav.).

Panaudojimo atvejo numeris ir pavadinimas	Testavimo scenarijaus numeris	Testavimo scenarijaus pavadinimas	Teisės	Išankstinės sąlygos	Testavimo scenarijaus žingsniai	Laukiamas rezultatas	Gautas rezultatas
PA6. Išsiųsti automatinį pranešimą apie knygų sąrašą	PA6.TC1	Išsiųsti automatinį pranešimą apie knygų sąrašą	[-]	Pasiekta 00:00 valanda.	1. Sistema 00:00 valandą inicijuoja knygų sąrašo pokyčių patikrą.		
					2. Sistema nuskaito paskutinės patikros „žymą“ (timestamp).		
					3. Sistema pateikia užklausą DB ir surenka knygų įrašus, pasikeitusius nuo paskutinės patikros (nauji / redaguoti / pašalinti).		
					4. Sistema suformuoja pokyčių suvestinę.		
					5. Sistema išsiunčia pranešimą administratoriams.		
					6. Sistema atnaujina paskutinės patikros žymą ir įrašo rezultatą į žurnalą.		

44 pav. Sugeneruoto testavimo atvejo, pagal tipą „Sisteminis“, pavyzdys

„Peržiūra“ testavimo atvejis sudaromas iš pagrindinio scenarijaus žingsnių, poruojant naudotojo veiksmus su atitinkamais sistemos atsakais ir pateikiant juos atskiruose žingsnių ir laukiamų rezultatų langeliuose (žr. 45 pav.).

Panaudojimo atvejo numeris ir pavadinimas	Testavimo scenarijaus numeris	Testavimo scenarijaus pavadinimas	Teisės	Išankstinės sąlygos	Testavimo scenarijaus žingsniai	Laukiamas rezultatas	Gautas rezultatas
PA1. Peržiūrėti knygų sąrašą	PA1.TC1	Peržiūrėti knygų sąrašą (peržiūrima)	[T1. Peržiūra]	Naudotojas prisijungęs prie sistemos.	1. Naudotojas spaudžia meniu punktą [Knygų sąrašas].	1. Sistema atidaro langą su knygų sąrašu.	

45 pav. Sugeneruoto testavimo atvejo, pagal tipą „Peržiūra“, pavyzdys

Jeigu norimų sugeneruoti testavimo atvejų žymėjimo lange naudotojas pasirinko papildomus „Peržiūra“ testavimo atvejus (filtruojama, puslapiuojama ir rikiuojama), suformuojami papildomi testavimo atvejai pagal iš anksto nustatytus (naudotojo nekoreguojamus) žingsnius ir laukiamus rezultatus. Filtravimo testavimo atvejis sudaromas panaudojant pirmus du žingsnius iš panaudojimo atvejų specifikacijos pagrindinio scenarijaus, o kiti sekantys žingsniai ir laukiami rezultatai užpildomi iš anksto numatytomis reikšmėmis (žr. 46 pav.).

Panaudojimo atvejo numeris ir pavadinimas	Testavimo scenarijaus numeris	Testavimo scenarijaus pavadinimas	Teisės	Išankstinės sąlygos	Testavimo scenarijaus žingsniai	Laukiamas rezultatas	Gautas rezultatas
PA1. Peržiūrėti knygų sąrašą	PA1.TC2	Peržiūrėti knygų sąrašą (filtruojama)	[T1. Peržiūra]	Naudotojas prisijungęs prie sistemos.	1. Naudotojas spaudžia meniu punktą [Knygų	1. Sistema atidaro langą su knygų sąrašu.	
					2. Naudotojas pasirenka norimus filtrus.	2. Sistema atnaujina sąrašą pagal pasirinktus	
					3. Naudotojas paspaudžia [Ištrinti filtrus].	3. Filtravimas veikia korektiškai.	

46 pav. Sugeneruoto papildomo (filtravimo) testavimo atvejo pavyzdys

Puslapiavimo testavimo atvejis taip pat sudaromas panaudojant pirmus du žingsnius iš panaudojimo atvejų specifikacijos pagrindinio scenarijaus, o kiti sekantys žingsniai ir laukiami rezultatai užpildomi iš anksto numatytomis reikšmėmis (žr. 47 pav.).

Panaudojimo atvejo numeris ir pavadinimas	Testavimo scenarijaus numeris	Testavimo scenarijaus pavadinimas	Teisės	Išankstinės sąlygos	Testavimo scenarijaus žingsniai	Laukiamas rezultatas	Gautas rezultatas
PA1. Peržiūrėti knygų sąrašą	PA1.TC3	Peržiūrėti knygų sąrašą (puslapiuojama)	[T1. Peržiūra]	Naudotojas prisijungęs prie sistemos.	1. Naudotojas spaudžia meniu punktą [Knygų	1. Sistema atidaro langą su knygų sąrašu.	
					2. Naudotojas pasirenka, kiek įrašų nori matyti viename puslapyje.	2. Sistema atnaujina sąrašą pagal pasirinktą įrašų kiekį viename	
					3. Naudotojas pabando vaikščioti per puslapius.	3. Puslapiavimas veikia korektiškai.	

47 pav. Sugeneruoto papildomo (puslapiavimo) testavimo atvejo pavyzdys

Rikiavimo testavimo atvejis sudaromas panaudojant pirmus du žingsnius iš panaudojimo atvejų specifikacijos pagrindinio scenarijaus, o kiti sekantys žingsniai ir laukiami rezultatai užpildomi iš anksto numatytomis reikšmėmis (žr. 48 pav.).

Panaudojimo atvejo numeris ir pavadinimas	Testavimo scenarijaus numeris	Testavimo scenarijaus pavadinimas	Teisės	Išankstinės sąlygos	Testavimo scenarijaus žingsniai	Laukiamas rezultatas	Gautas rezultatas
PA1. Peržiūrėti knygų sąrašą	PA1.TC4	Peržiūrėti knygų sąrašą (rikiuojama)	[T1. Peržiūra]	Naudotojas prisijungęs prie sistemos.	1. Naudotojas spaudžia meniu punktą [Knygų	1. Sistema atidaro langą su knygų sąrašu.	
					2. Naudotojas pasirenka stulpelį, pagal kurį nori rikiuoti sąrašą ir spaudžia	2. Sistema rikiuoja sąrašą pagal pasirinktą stulpelį didėjimo tvarka.	
					3. Naudotojas dar kartą paspaudžia ant to paties stulpelio pavadinimo.	3. Sistema rikiuoja sąrašą pagal pasirį pasirinktą stulpelį mažėjimo tvarka.	

48 pav. Sugeneruoto papildomo (rikiavimo) testavimo atvejo pavyzdys

„Kūrimas“ testavimo atvejis formuojamas naudojant pagrindinio scenarijaus žingsnius bei, jei numatyta specifikacijoje, papildomus alternatyvius scenarijus, kurie pateikiami kaip atskiri testavimo atvejai (žr. 49 pav.).

Panaudojimo atvejo numeris ir pavadinimas	Testavimo scenarijaus numeris	Testavimo scenarijaus pavadinimas	Teisės	Išankstinės sąlygos	Testavimo scenarijaus žingsniai	Laukiamas rezultatas	Gautas rezultatas
PA3. Pridėti knygos įrašą	PA3.TC1	Pridėti knygos įrašą (sukuriama)	[T2. Kūrimas]	Atvertas knygų sąrašo langas.	1. Naudotojas spaudžia mygtuką [Pridėti].	1. Sistema atidaro įrašo kūrimo langą.	
					2. Naudotojas užpildo duomenis ir spaudžia [Saugoti]. Sistema patikrina: Privalomi laukai užpildyti	2. Jei neatitinka, išvedamas klaidos pranešimas „Užpildykite privalomus laukus“.	
					3. Naudotojas užpildo duomenis ir spaudžia [Saugoti]. Sistema patikrina: Privalomi laukai užpildyti	3. Jei atitinka, sistema išsaugo įvestus duomenis ir atidaro knygų sąrašo peržiūros langą.	

49 pav. Sugeneruoto testavimo atvejo, pagal tipą „Kūrimas“, pavyzdys

„Redagavimas“ testavimo atvejis sudaromas analogiškai „Kūrimas“ atvejui, naudojant pagrindinio scenarijaus informaciją ir atvaizduojant naudotojo atliekamus veiksmus bei sistemos atsakus (žr. 50 pav.).

Panaudojimo atvejo numeris ir pavadinimas	Testavimo scenarijaus numeris	Testavimo scenarijaus pavadinimas	Teisės	Išankstinės sąlygos	Testavimo scenarijaus žingsniai	Laukiamas rezultatas	Gautas rezultatas
PA4. Redaguoti knygos įrašą	PA4.TC1	Redaguoti knygos įrašą (redaguojama)	[T3. Redagavimas]	Atvertas pasirinkto knygos įrašo peržiūros langas.	1. Naudotojas spaudžia mygtuką [Redaguoti].	1. Sistema atidaro įrašo redagavimo langą.	
					2. Naudotojas užpildo duomenis ir spaudžia [Saugoti]. Sistema patikrina: Privalomi laukai užpildyti	2. Jei neatitinka, išvedamas klaidos pranešimas „Užpildykite privalomus laukus“.	
					3. Naudotojas užpildo duomenis ir spaudžia [Saugoti]. Sistema patikrina: Privalomi laukai užpildyti	3. Jei atitinka, sistema išsaugo įvestus duomenis ir atidaro knygų sąrašo peržiūros langą.	

50 pav. Sugeneruoto testavimo atvejo, pagal tipą „Redagavimas“, pavyzdys

„Šalinimas“ testavimo atvejis suformuojamas remiantis pagrindinio scenarijaus žingsniais, pateikiant naudotojo inicijuojamą šalinimo veiksmą ir sistemos atsaką į jį (žr. 51 pav.).

Panaudojimo atvejo numeris ir pavadinimas	Testavimo scenarijaus numeris	Testavimo scenarijaus pavadinimas	Teisės	Išankstinės sąlygos	Testavimo scenarijaus žingsniai	Laukiamas rezultatas	Gautas rezultatas
PA5. Pašalinti knygos įrašą	PA5.TC1	Pašalinti knygos įrašą (šalinama)	[T4. Šalinti]	Atvertas pasirinkto knygos įrašo peržiūros langas.	1. Naudotojas spaudžia mygtuką [Šalinti].	1. Sistema išveda informacinį pranešimą su tekstu „Ar tikrai norite šalinti šį įrašą?“.	
					2. Naudotojas spaudžia [Ne].	2. Sistema uždaro informacinio pranešimo langą, įrašas nepašalinamas.	
					3. Naudotojas spaudžia mygtuką [Šalinti].	3. Sistema išveda informacinį pranešimą su tekstu „Ar tikrai norite šalinti šį įrašą?“.	
					4. Naudotojas spaudžia [Taip].	4. Sistema uždaro informacinio pranešimo langą, įrašas pašalinamas.	

51 pav. Sugeneruoto testavimo atvejo, pagal tipą „Šalinimas“, pavyzdys

Papildomai, naudotojas, renkantis norimus sugeneruoti testavimo atvejus, gali pasirinkti netinkamų teisių testavimo atvejį. Šiuo atveju panaudojamas tik pirmas žingsnis iš panaudojimo atvejų specifikacijos, o laukiamas rezultatas užpildomas iš anksto numatyta reikšme (žr. 52 pav.).

Panaudojimo atvejo numeris ir pavadinimas	Testavimo scenarijaus numeris	Testavimo scenarijaus pavadinimas	Teisės	Išankstinės sąlygos	Testavimo scenarijaus žingsniai	Laukiamas rezultatas	Gautas rezultatas
PA1. Peržiūrėti knygų sąrašą	PA1.TC5	Peržiūrėti knygų sąrašą (netinkamos teisės)	Naudotojas neturi teisių [T1. Peržiūra]	Naudotojas prisijungęs prie sistemos.	1. Naudotojas spaudžia meniu punktą [Knygų	1. Naudotojas neturi tinkamų teisių atlikti šį	

52 pav. Sugeneruoto papildomo (netinkamos teisės) testavimo atvejo pavyzdys

Pateikti pavyzdžiai parodo, kad sukurtas įrankis geba korektiškai sugeneruoti skirtingų tipų testavimo atvejus pagal panaudojimo atvejų specifikacijose pateiktą informaciją. Testavimo metu nustatyta, kad pagrindiniai scenarijai, naudotojo veiksmai ir sistemos žingsniai buvo tinkamai susieti bei pateikti atskiruose žingsnių ir laukiamų rezultatų laukeliuose. Taip pat sėkmingai sugeneruoti sisteminiai, peržiūros, kūrimo, redagavimo ir šalinimo tipo testavimo atvejai bei papildomi scenarijai pagal pasirinktus požymius. Testavimo rezultatai parodė, kad įrankis gali generuoti testavimo atvejus ir pritaikyti skirtingas generavimo taisykles pagal pasirinktą testavimo atvejo tipą.

5. Testavimo atvejų generavimo įrankio eksperimentinis tyrimas

5.1. Eksperimento planas

Šiame skyriuje aprašomas sukurto testavimo atvejų generavimo įrankio eksperimentinis tyrimas, skirtas įvertinti įrankio gebėjimą generuoti testavimo atvejus pagal UML panaudojimo atvejų diagramas ir jų specifikacijas.

Eksperimentas sudarytas iš dviejų pagrindinių dalių, siekiant įvertinti tiek sukurto įrankio rezultatus, tiek jo praktinį pritaikomumą testavimo procese.

Pirmojoje eksperimento dalyje atliekamas vertinimas, naudojant realius eksperimentinius duomenis, gautus iš įmonės:
1. vertinimui paruošti trijų skirtingų projektų duomenys;
2. į sukurtą įrankį įkeliamos UML panaudojimo atvejų diagramos ir jų specifikacijos;
3. įrankis automatiškai sugeneruoja testavimo atvejus;
4. sugeneruoti rezultatai lyginami su įmonės testuotojų rankiniu būdu sukurtais testavimo atvejais, parengtais pagal tą pačią specifikaciją.

Antrojoje eksperimento dalyje atliekamas vertinimas, pasitelkiant tikslinės auditorijos apklausą:
1. testavimo specialistams pristatomas sukurtas įrankis ir jo veikimo demonstracija;
2. apklausos dalyviai vertina įrankio aiškumą, patogumą naudoti bei praktinę naudą testavimo procesui;
3. vertinamos įrankio taikymo galimybės realioje darbo aplinkoje.

5.2. Testavimo atvejų generavimo eksperimentas

Šiame skyriuje aprašoma pirmoji eksperimento dalis, kurios metu atliekamas sukurto įrankio sugeneruotų testavimo atvejų palyginimas su įmonėje rankiniu būdu sukurtais testavimo atvejais.

5.2.1. Eksperimentiniai duomenys

Eksperimentiniam tyrimui buvo pasitelkti trys skirtingi projektai ir jų duomenys, kurie buvo gauti iš įmonės. Žemiau pateikiama visų eksperimentinių duomenų informacija ir struktūra.

Pirmasis eksperimentinių duomenų rinkinys yra finansų (investicijų) srities projekto dalis – projektų modulio funkcionalumas. Šis modulis skirtas investicinių projektų administravimui: projektų kūrimui, duomenų valdymui, susijusių dokumentų ir atsakingų asmenų administravimui bei projektų būsenų stebėsenai. Modulio sudėtingumas pasireiškia per tarpusavyje susijusių duomenų valdymą, skirtingų naudotojų teisių taikymą bei integracijas su kitais sistemos moduliais. Šio modulio kūrimui buvo priskirta atskira komanda, kurią sudarė 1 sistemų analitikas, 3 programuotojai ir 1 sistemų testuotojas. Modulio kūrimas vyko 2022–2023 metų laikotarpiu kaip viena iš šešių bendros sistemos dalių, kurios tarpusavyje integruojasi ir sudaro pilną finansų valdymo sistemą.

Panaudojimo atvejų specifikacijos failą sudaro panaudojimo atvejų diagrama ir kiekvieno panaudojimo atvejo lentelė. Projekto specifikacijos failas yra sudarytas pagal numatytus

specifikacijos failo reikalavimus kuriamam įrankiui. Išsami eksperimentinių duomenų struktūra pateikiama žemiau esančioje lentelėje (žr. 5 lentelė).

5 lentelė. Finansų valdymo sistemos projektų modulio panaudojimo atvejų specifikacijos failo struktūra

Rodiklis		**Kiekis**
Panaudojimo atvejų diagrama		1
Panaudojimo atvejų skaičius diagramoje		17
Panaudojimo atvejų skaičius lentelėse		17
Panaudojimo atvejų, turinčių aprašytus pagrindinius scenarijus dokumente, skaičius		17
Panaudojimo atvejų, turinčių aprašytus alternatyviuosius scenarijus dokumente, skaičius		4
Panaudojimo atvejų, turinčių sistemos tikrinimus, skaičius		3
Taisyklių, esančių sistemos tikrinimo sakiniuose, skaičius		7
Panaudojimo atvejų, turinčių nurodytas roles ar/ir teises skaičius		15
Panaudojimo atvejų skaičius pagal požymį	Peržiūra	8
	Kūrimas	3
	Redagavimas	1
	Šalinimas	2
	Sisteminis	1
	Tik pagrindinis	2

Antrasis eksperimentinių duomenų rinkinys yra to paties finansų (investicijų) projekto dalis ir apima sutarčių modulio funkcionalumą. Šis modulis skirtas sutarčių kūrimui, redagavimui, saugojimui ir jų gyvavimo ciklo valdymui, įskaitant sutarčių būsenų sekimą, susijusių dokumentų administravimą bei sąsajas su projektais ir kitais sistemos objektais. Modulio sudėtingumas pasireiškia per sudėtingą verslo logiką, skirtingų sutarčių tipų apdorojimą, naudotojų teisių kontrolę bei integracijas su kitais sistemos moduliais. Šio modulio kūrimui buvo priskirta komanda, kurią sudarė 1 sistemų analitikas, 2 programuotojai ir 1 sistemų testuotojas. Modulio kūrimas vyko 2022–2024 metų laikotarpiu kaip viena iš bendros sistemos dalių. Eksperimentiniams tyrimams naudojamas šio modulio panaudojimo atvejų specifikacijos failas, sudarytas pagal nustatytus reikalavimus, apimantis panaudojimo atvejų diagramą ir kiekvieno panaudojimo atvejo aprašą lentelės forma (žr. 6 lentelė).

6 lentelė. Finansų valdymo sistemos sutarčių modulio panaudojimo atvejų specifikacijos failo struktūra

Rodiklis	**Kiekis**
Panaudojimo atvejų diagrama	1
Panaudojimo atvejų skaičius diagramoje	20
Panaudojimo atvejų skaičius lentelėse	20
Panaudojimo atvejų, turinčių aprašytus pagrindinius scenarijus dokumente, skaičius	20
Panaudojimo atvejų, turinčių aprašytus alternatyviuosius scenarijus dokumente, skaičius	12
Panaudojimo atvejų, turinčių sistemos tikrinimus, skaičius	23
Taisyklių, esančių sistemos tikrinimo sakiniuose, skaičius	54

Rodiklis		Kiekis
Panaudojimo atvejų, turinčių nurodytas roles ar/ir teises skaičius		20
Panaudojimo atvejų skaičius pagal požymį	Peržiūra	5
	Kūrimas	4
	Redagavimas	3
	Šalinimas	1
	Sisteminis	0
	Tik pagrindinis	7

Trečiasis eksperimentinių duomenų rinkinys yra verslo valdymo srities projektas, orientuotas į mažosios bendrijos veiklos apskaitos informacinės sistemos kūrimą. Projekto užsakovas – mažoji bendrija. Projekto vykdytojas – programinės įrangos kūrimo paslaugas teikianti įmonė. Projekto pradžia – 2022 metai, pabaiga – 2024 metai. Projektui buvo priskirta nedidelė IT specialistų komanda, sudaryta iš projekto vadovo, sistemų analitiko, 3 programuotojų ir testuotojo. Projekto metu kuriama informacinė sistema, skirta veiklos apskaitos procesų valdymui ir automatizavimui pagal užsakovo pateiktus reikalavimus. Sistema vystoma etapais, siekiant užtikrinti nuoseklų funkcionalumų įgyvendinimą ir jų tarpusavio suderinamumą.

Šių duomenų panaudojimo atvejų specifikacijos failą sudaro panaudojimo atvejų diagrama ir kiekvieno panaudojimo atvejo lentelė. Projekto specifikacijos failas yra sudarytas pagal numatytus specifikacijos failo reikalavimus kuriamam įrankiui. Išsami eksperimentinių duomenų struktūra pateikiama žemiau esančioje lentelėje (žr. 7 lentelė).

7 lentelė. MB veiklos apskaitos panaudojimo atvejų specifikacijos failo struktūra

Rodiklis		Kiekis
Panaudojimo atvejų diagrama		1
Panaudojimo atvejų skaičius diagramoje		9
Panaudojimo atvejų skaičius lentelėse		9
Panaudojimo atvejų, turinčių aprašytus pagrindinius scenarijus dokumente, skaičius		9
Panaudojimo atvejų, turinčių aprašytus alternatyviuosius scenarijus dokumente, skaičius		9
Panaudojimo atvejų, turinčių sistemos tikrinimus, skaičius		0
Taisyklių, esančių sistemos tikrinimo sakiniuose, skaičius		0
Panaudojimo atvejų, turinčių nurodytas roles ar/ir teises skaičius		0
Panaudojimo atvejų skaičius pagal požymį	Peržiūra	1
	Kūrimas	2
	Redagavimas	1
	Šalinimas	1
	Sisteminis	0
	Tik pagrindinis	4

Apibendrinant eksperimentinius duomenis, tyrimui buvo panaudoti trys skirtingų sričių projektų panaudojimo atvejų specifikacijų rinkiniai, apimantys finansų valdymo ir verslo apskaitos sistemas. Duomenyse buvo pateikti tiek pagrindiniai, tiek alternatyvieji scenarijai,

sistemos tikrinimai bei naudotojų rolės ir teisės. Skirtingi duomenų rinkiniai leido įvertinti įrankio veikimą įvairiose situacijose – nuo paprastesnių panaudojimo atvejų, turinčių tik pagrindinius scenarijus, iki sudėtingesnių specifikacijų su alternatyviais scenarijais ir papildomais sistemos tikrinimais.

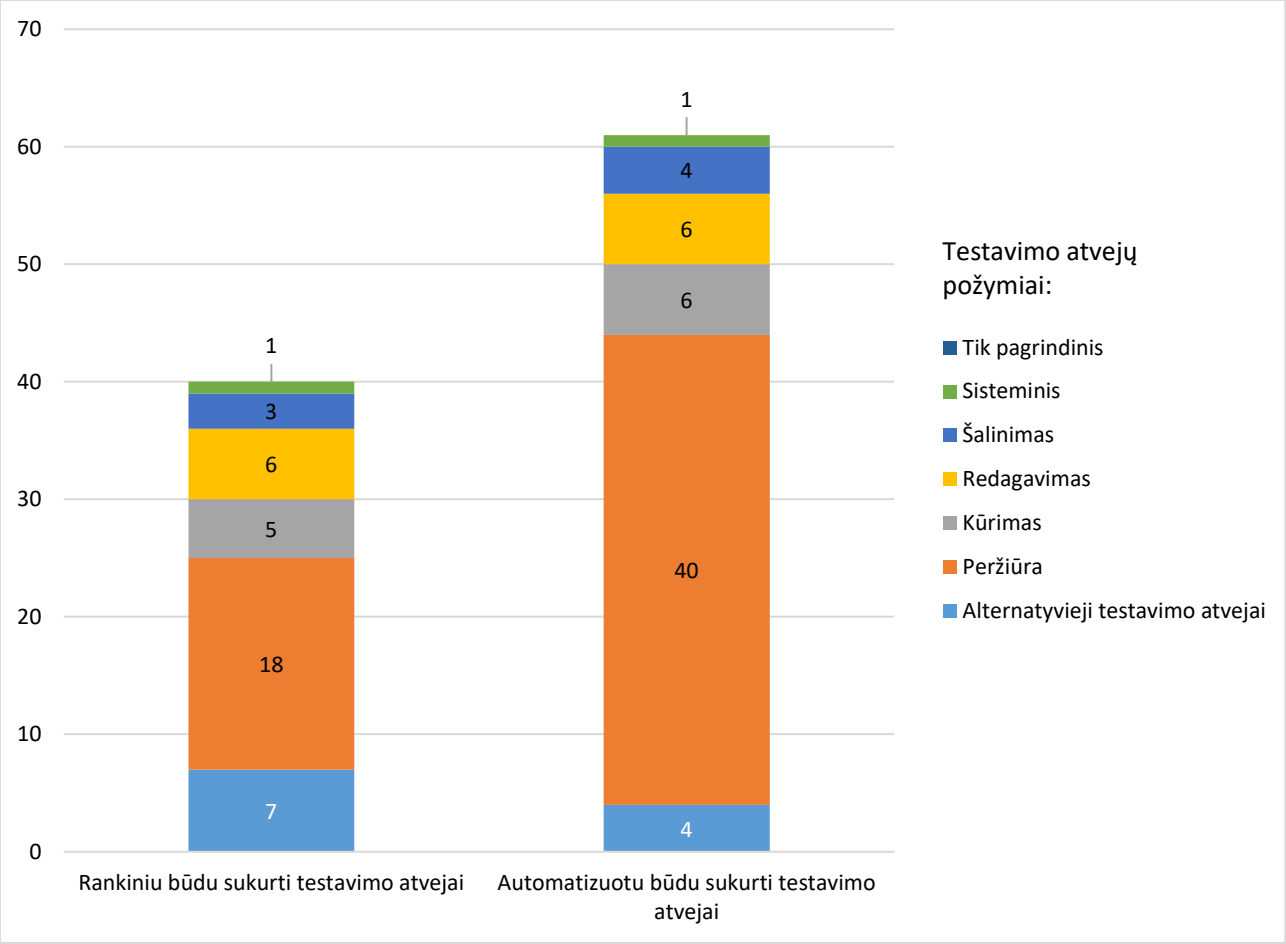
5.2.2. Automatizuoto testavimo atvejų generavimo eksperimento rezultatai

Atliekant eksperimentą su pirmuoju duomenų rinkiniu – projektų moduliu finansų valdymo sistemai, testavimo atvejų generavimui buvo pasirinkti visi įrankio siūlomi testavimo scenarijai pagal identifikuotus panaudojimo atvejų požymius. Kiekvienam panaudojimo atvejui buvo priskirti atitinkami požymiai (pvz., peržiūra, kūrimas, redagavimas, šalinimas). Generavimo metu įrankis automatiškai suformavo testavimo atvejus, remdamasis tiek pagrindiniais, tiek alternatyviais scenarijais. Finansų valdymo sistemos projektų modulio testavimo rezultatai pateikiami žemiau esančioje lentelėje (žr. **8 lentelė**).

8 lentelė. Finansų valdymo sistemos projektų modulio testavimo atvejų generavimo rezultatų lentelė

Rodiklis			**Rankiniu būdu sukurti testavimo atvejai**	**Automatizuotu būdu sukurti testavimo atvejai**
Bendras testavimo atvejų skaičius			40	61
Pagrindiniai scenarijai			17	17
Alternatyvieji scenarijai			4	4
Padengtų sistemos tikrinimų skaičius			3	3
Padengtų taisyklių, sistemos atliekamų tikrinimų žingsniuose, skaičius			7	7
Testavimo atvejai pagal tipą	Peržiūra	Peržiūrima	8	8
		Filtruojama	1	8
		Puslapiuojama	5	8
		Rikiuojama	1	8
		Netinkamos teisės	3	8
	Kūrimas	Sukuriama	3	3
		Netinkamos teisės	2	3
	Redagavimas	Redaguojama	3	3
		Netinkamos teisės	3	3
	Šalinimas	Šalinama	2	2
		Netinkamos teisės	1	2
	Sisteminis		1	1
	Tik pagrindinis		0	0

Remiantis lentelėje pateiktais duomenimis, buvo sudaryta diagrama, kurioje vizualiai palyginami rankiniu ir automatizuotu būdu sukurti testavimo atvejai pagal skirtingus požymius (žr. 53 pav.).



53 pav. Finansų valdymo sistemos projektų modulio sugeneruoti testavimo atvejai pagal požymius

Diagrama parodo, kad didžiausi skirtumai tarp rankiniu ir automatizuotu būdu sukurtų testavimo atvejų šiame projekte pasireiškė peržiūros tipo scenarijuose bei alternatyviųjų scenarijų kategorijoje. Automatizuotas generavimas sukūrė daugiau peržiūros tipo testavimo atvejų, nes įrankis kiekvienam peržiūros tipo panaudojimo atvejui papildomai generavo filtravimo, rikiavimo, puslapiavimo ir netinkamų teisių scenarijus. Tokia logika buvo pritaikyta visiems panaudojimo atvejams, kurie buvo priskirti peržiūros kategorijai.

Eksperimento metu pastebėta, kad dalyje projektų modulio sąrašų realioje sistemoje nebuvo galimybės atlikti filtravimo, rikiavimo arba puslapiavimo veiksmų. Dėl šios priežasties kai kurie sugeneruoti scenarijai praktiniame testavime galėtų būti laikomi pertekliniais. Visgi eksperimento metu buvo pasirinkti visi įrankio pasiūlyti testavimo scenarijai, siekiant įvertinti pilną sugeneruojamų testavimo atvejų apimtį. Praktiniame naudojime testuotojas gali pasirinkti tik aktualius scenarijus ir atmesti nereikalingus.

Kūrimo, redagavimo ir šalinimo scenarijų kategorijose skirtumai buvo mažesni, šiose kategorijose įrankis sugeneravo aprašytus pagrindinius ir alternatyvius scenarijus. Papildomi skirtumai atsirado dėl automatiškai generuojamų netinkamų teisių scenarijų, kurie rankiniu būdu ne visada buvo kuriami kiekvienam panaudojimo atvejui.

Taip pat pastebėta, kad rankiniu būdu buvo sukurta daugiau alternatyviųjų scenarijų, susijusių su išankstinėmis sąlygomis bei platesne verslo logika. Tokie scenarijai dažnai buvo paremti skirtingomis projekto būsenomis, sąsajų tarp kitų sistemos modulių ar situacijomis, kurios nebuvo tiesiogiai aprašytos panaudojimo atvejų specifikacijose.

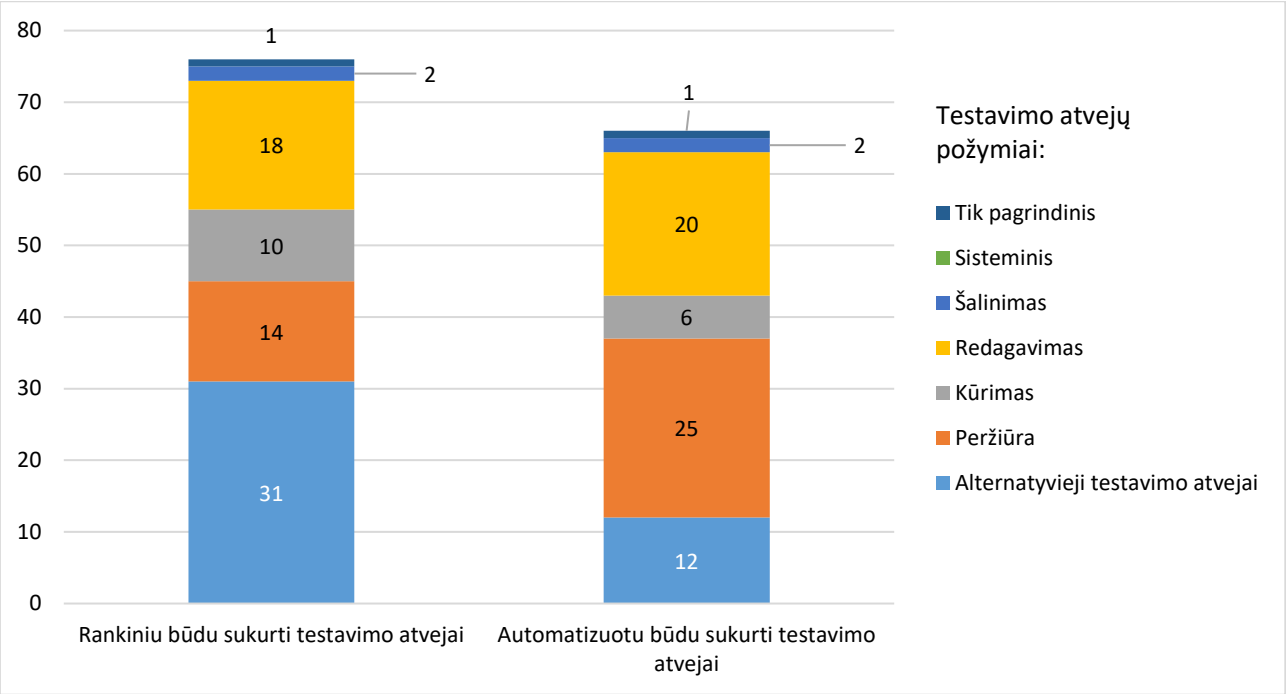
Gauti rezultatai parodė, kad automatizuotas sprendimas geriausiai veikia standartizuotuose ir aiškiai apibrėžtuose scenarijuose, tačiau platesnės verslo logikos vertinimas vis dar reikalauja papildomo testuotojo įsitraukimo.

Atliekant eksperimentą su antruoju duomenų rinkiniu, sutarčių moduliu finansų valfymo sistemai, testavimo atvejų generavimas buvo vykdomas analogišku principu – kiekvienam panaudojimo atvejui priskirti atitinkami požymiai. Generavimo procese įrankis automatiškai formavo testavimo atvejus, remdamasis pateiktomis specifikacijomis, pagrindiniais ir alternatyviais scenarijais. Rezultatai pateikiami žemiau esančioje lentelėje (žr. **9 lentelė**).

9 lentelė. Finansų valdymo sistemos sutarčių modulio testavimo atvejų generavimo rezultatai

Rodiklis			**Rankiniu būdu sukurti testavimo atvejai**	**Automatizuotu būdu sukurti testavimo atvejai**
Bendras testavimo atvejų skaičius			76	66
Pagrindiniai scenarijai			20	20
Alternatyvieji scenarijai			12	12
Padengtų sistemos tikrinimų skaičius			23	23
Padengtų taisyklių, sistemos atliekamų tikrinimų žingsniuose, skaičius			54	54
Testavimo atvejai pagal tipą	Peržiūra	Peržiūrima	8	5
		Filtruojama	0	5
		Puslapiuojama	3	5
		Rikiuojama	1	5
		Netinkamos teisės	2	5
	Kūrimas	Sukuriama	6	3
		Netinkamos teisės	4	3
	Redagavimas	Redaguojama	9	10
		Netinkamos teisės	9	10
	Šalinimas	Šalinama	1	1
		Netinkamos teisės	1	1
	Sisteminis		0	0
	Tik pagrindinis		1	1

Pagal gautus rezultatus buvo sudaryta diagrama, leidžianti palyginti rankiniu ir automatizuotu būdu sukurtų testavimo atvejų pasiskirstymą pagal tipus (žr. 54 pav.).



54 pav. Finansų valdymo sistemos sutarčių modulio sugeneruoti testavimo atvejai pagal požymius

Diagrama rodo, kad šiame duomenų rinkinyje rankiniu būdu sudarytų testavimo atvejų buvo daugiau nei sugeneruotų automatizuotu būdu. Didžiausias skirtumas pastebimas alternatyviųjų scenarijų kategorijoje. Rankiniu būdu buvo parengta daugiau alternatyviųjų scenarijų, nes testuotojai vertino ne tik specifikacijose aprašytus scenarijus, bet ir situacijas, susijusias su išankstinėmis sąlygomis, sąveika su kitais sistemos moduliais bei platesne verslo logika. Tuo tarpu automatizuotas įrankis alternatyviuosius scenarijus generavo remdamasis tik aiškiai specifikacijose apibrėžtomis taisyklėmis ir scenarijais, todėl jų skaičius buvo mažesnis.

Taip pat matoma, kad automatizuotu būdu buvo sugeneruota daugiau peržiūros tipo testavimo atvejų. Šį skirtumą nulėmė tai, kad įrankis papildomai generavo standartizuotus scenarijus, skirtus filtravimo, rikiavimo, puslapiavimo bei netinkamų teisių tikrinimo funkcionalumui. Tokie scenarijai buvo kuriami visiems panaudojimo atvejams, priskirtiems peržiūros kategorijai.

Kūrimo, redagavimo ir šalinimo scenarijų kategorijose skirtumai buvo mažesni, nes visi panaudojimo atvejai ir jų scenarijai buvo aprašyti specifikacijose. Dėl šios priežasties automatizuotas įrankis šiose kategorijose sugeneravo testavimo scenarijus, atitinkančius rankiniu būdu sudarytus scenarijus. Pastebėta, kad redagavimo scenarijų automatizuotu būdu buvo sugeneruota daugiau nei rankiniu būdu, nes įrankis kiekvienam redagavimo tipo panaudojimo atvejui papildomai kūrė netinkamų teisių scenarijus, kurie rankinio testavimo metu ne visada buvo įtraukiami.

Gauti rezultatai parodė, kad automatizuotas sprendimas pateikia geriausius rezultatus tada, kai panaudojimo atvejų specifikacijos yra aprašytos pagal apibrėžtas taisykles (žr. Testavimo atvejų generavimo metodikos įvesties duomenų modelis, šablonas ir taisyklės).

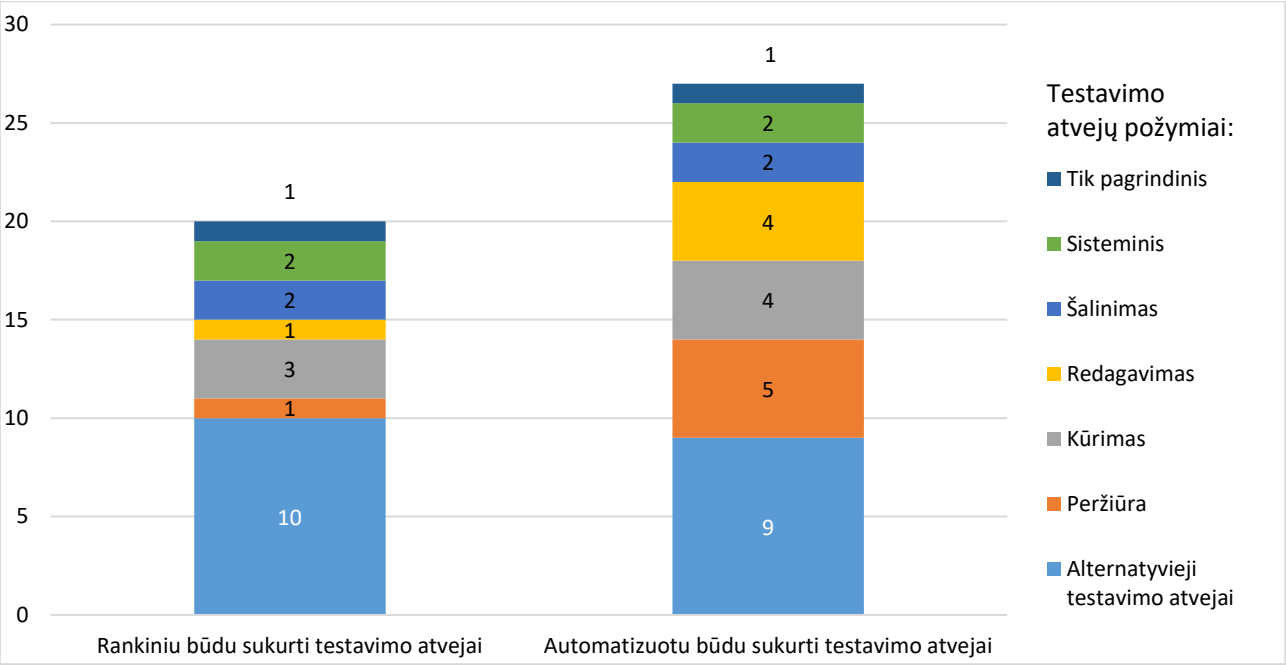
Tuo tarpu sudėtingesni alternatyvieji scenarijai, susiję su platesne verslo logika ar išankstinėmis sąlygomis, vis dar reikalauja papildomo testuotojo vertinimo.

Atliekant eksperimentą su trečiuoju duomenų rinkiniu, testavimo atvejų generavimas buvo vykdomas taikant tą pačią logiką – kiekvienam panaudojimo atvejui priskirti atitinkami požymiai. Generavimo procese buvo siekiama maksimaliai išnaudoti įrankio funkcionalumą, todėl įtraukti visi galimi testavimo scenarijai, kuriuos galima suformuoti remiantis specifikacijoje pateikta informacija ir taisyklėmis. Rezultatai pateikiami žemiau esančioje lentelėje (žr. **10 lentelė**).

10 lentelė. MB veiklos apskaitos testavimo atvejų generavimo rezultatai

Rodiklis			**Rankiniu būdu sukurti testavimo atvejai**	**Automatizuotu būdu sukurti testavimo atvejai**
Bendras testavimo atvejų skaičius			20	27
Pagrindiniai scenarijai			9	9
Alternatyvieji scenarijai			9	9
Papildomi scenarijai, pasiūlyti įrankio			-	9
Padengtų sistemos tikrinimų skaičius			0	0
Padengtų taisyklių, sistemos atliekamų tikrinimų žingsniuose, skaičius			0	0
Testavimo atvejai pagal tipą	Peržiūra	Peržiūrima	1	1
		Filtruojama	0	1
		Puslapiuojama	0	1
		Rikiuojama	0	1
		Netinkamos teisės	0	1
	Kūrimas	Sukuriama	3	2
		Netinkamos teisės	0	2
	Redagavimas	Redaguojama	1	2
		Netinkamos teisės	0	2
	Šalinimas	Šalinama	2	1
		Netinkamos teisės	0	1
	Sisteminis		2	2
	Tik pagrindinis		1	1

Remiantis lentelėje pateiktais duomenimis, sudaryta diagrama, kurioje pateikiamas testavimo atvejų pasiskirstymas pagal tipus, lyginant rankinį ir automatizuotą generavimo būdus (žr. 55 pav.).



55 pav. MB veiklos apskaitos sugeneruoti testavimo atvejai pagal požymius

Diagrama parodo, kad pagrindinių bei alternatyviųjų scenarijų skaičius yra labai panašus, tai susiję su tuo, kad šiame projekte panaudojimo atvejų specifikacijos buvo aprašytos laikantis apibrėžtų taisyklių (žr. Testavimo atvejų generavimo metodikos įvesties duomenų modelis, šablonas ir taisyklės) ir turėjo mažiau sudėtingų verslo taisyklių ar papildomų išankstinių sąlygų.

Didžiausi skirtumai matomi peržiūros ir redagavimo tipo scenarijuose. Automatizuotas įrankis šiose kategorijose sugeneravo papildomus standartizuotus scenarijus, susijusius su filtravimu, rikiavimu, puslapiavimu bei netinkamų teisių tikrinimu. Tokie scenarijai buvo generuojami automatiškai pagal nustatytas taisykles. Tuo tarpu rankiniu būdu kuriami testavimo atvejai šių situacijų neskirdavo į atskirus scenarijus, nes testuotojas jas vertino kaip mažiau reikšmingas arba apjungdavo į vieną bendresnį testavimo atvejį.

Svarbu pažymėti, kad eksperimento metu buvo pasirenkami visi įrankio sugeneruoti scenarijai, siekiant įvertinti pilną automatizuoto generavimo apimtį. Praktiniame naudojime testuotojas gali atsirinkti tik aktualius scenarijus ir neįtraukti nereikalingų ar konkrečiame funkcionalume netaikomų variantų.

Taip pat pastebėta, kad nors alternatyviųjų scenarijų skaičius išliko panašus, jų turinys skyrėsi. Rankiniu būdu sudaryti scenarijai dažniau apėmė platesnį situacijos interpretavimą, kurį testuotojai identifikavo remdamiesi savo praktine patirtimi. Tuo tarpu automatizuotas įrankis generavo tik tuos scenarijus, kurie buvo aprašyti panaudojimo atvejų specifikacijose.

Rezultatai parodė, kad šiame projekte automatizuotas generavimas parodė geriausius rezultatus, kai specifikacijos buvo aprašytos pagal nustatytas taisykles ir turėjo mažiau sudėtingų verslo logikos situacijų.

Apibendrinus visų trijų eksperimentų rezultatus, pastebėta, kad automatizuotu būdu sugeneruotų testavimo atvejų skaičius daugeliu atvejų buvo didesnis nei rankiniu būdu

sukurtų testavimo atvejų. Tai daugiausia lėmė papildomi scenarijai, kuriuos įrankis generavo pagal nustatytas taisykles, pavyzdžiui, filtravimo, puslapiavimo, rikiavimo ar netinkamų teisių tikrinimo scenarijai.

Visuose eksperimentuose pagrindinių scenarijų bei specifikacijose aprašytų alternatyviųjų scenarijų padengimas sutapo su rankiniu būdu parengtais testavimo atvejais. Tačiau nustatyta, kad rankiniu būdu sudaryti testavimo atvejai dažniau apėmė scenarijus, susijusius su išankstinėmis sąlygomis ar papildomais verslo logikos veiksmais, kurių automatizuotas įrankis neidentifikavo.

5.3. Testavimo atvejų naudos analizės eksperimentas

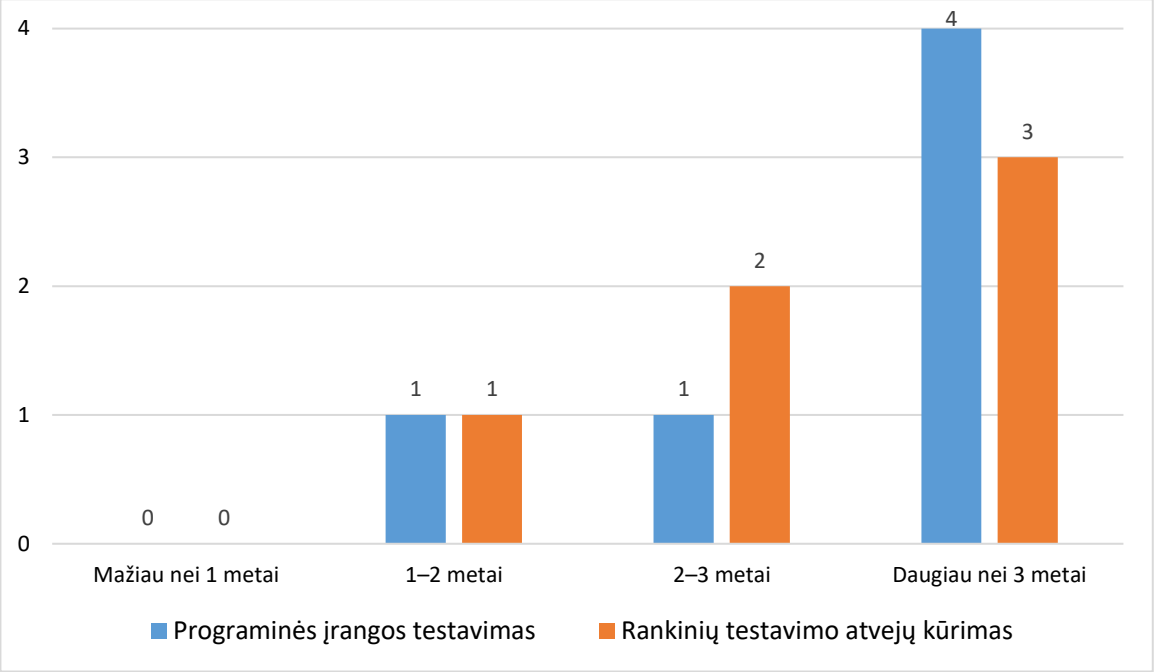
Šiame skyriuje aprašoma antroji eksperimento dalis, kurios metu atliekamas sukurto įrankio vertinimas, pasitelkiant apklausos metodą. Sudarytas klausimynas, o surinkti duomenys analizuojami siekiant įvertinti įrankio naudojimo patirtį, aiškumą ir praktinį pritaikomumą. Apklausos duomenys buvo renkami 2026 m. balandžio mėnesį, apklausoje dalyvavo 6 respondentai, kur visi respondentai turėjo praktinės patirties programinės įrangos testavime bei rankinių testavimo atvejų kūrime. Apklausoje dalyvavę respondentai dirbo informacinių technologijų įmonėse, kuriančiose programinės įrangos sistemas. Jų darbas buvo susijęs su programinės įrangos testavimu, kokybės užtikrinimu bei rankinių testavimo atvejų kūrimu.

Apklausą sudaro 7 klausimai, kurie yra suskirstyti į dvi dalis pagal jų pobūdį. Pirmieji du klausimai yra skirti respondentų patirčiai įvertinti:
1. Kiek metų patirties turite programinės įrangos testavime?
2. Kiek patirties turite rankinių testavimo atvejų kūrime?

Šiems klausimams buvo pateikti atsakymų variantai su laikotarpio intervalais, siekiant struktūruotai įvertinti respondentų patirties lygį.

Visi apklausoje dalyvavę respondentai turi praktinės patirties programinės įrangos testavimo srityje bei rankinių testavimo atvejų kūrime (žr. 56 pav.). Dauguma respondentų nurodė turintys daugiau nei 3 metų patirtį, todėl galima teigti, kad apklausoje dalyvavo su testavimo procesu susipažinę specialistai, galintys objektyviai įvertinti pateiktą sprendimą ir jo praktinį pritaikomumą (žr. 2 priedas.

Testavimo ekspertų apklausos rezultatai pagal klausimą).



56 pav. Apklausos repsondentų patirtis testavimo srityje

Likę penki klausimai pateikiami kaip teiginiai, skirti įvertinti respondentų požiūrį į pateiktą įrankį:

1. Rankinių testavimo atvejų kūrimas yra daug resursų reikalaujantis procesas, todėl jo automatizavimas būtų naudingas.
2. Automatizuoto rankinių testavimo atvejų generavimo įrankio veikimo principai man buvo aiškūs ir suprantami.
3. Pateiktas automatizuoto rankinių testavimo atvejų generavimo įrankis gali palengvinti rankinių testavimo atvejų kūrimo procesą.
4. Jeigu man reikėtų kurti rankinius testavimo atvejus pagal UML panaudojimo atvejų specifikacijas, naudočiau šį įrankį.
5. Rekomenduočiau šį įrankį kitiems specialistams, kuriems tenka kurti rankinius testavimo atvejus.

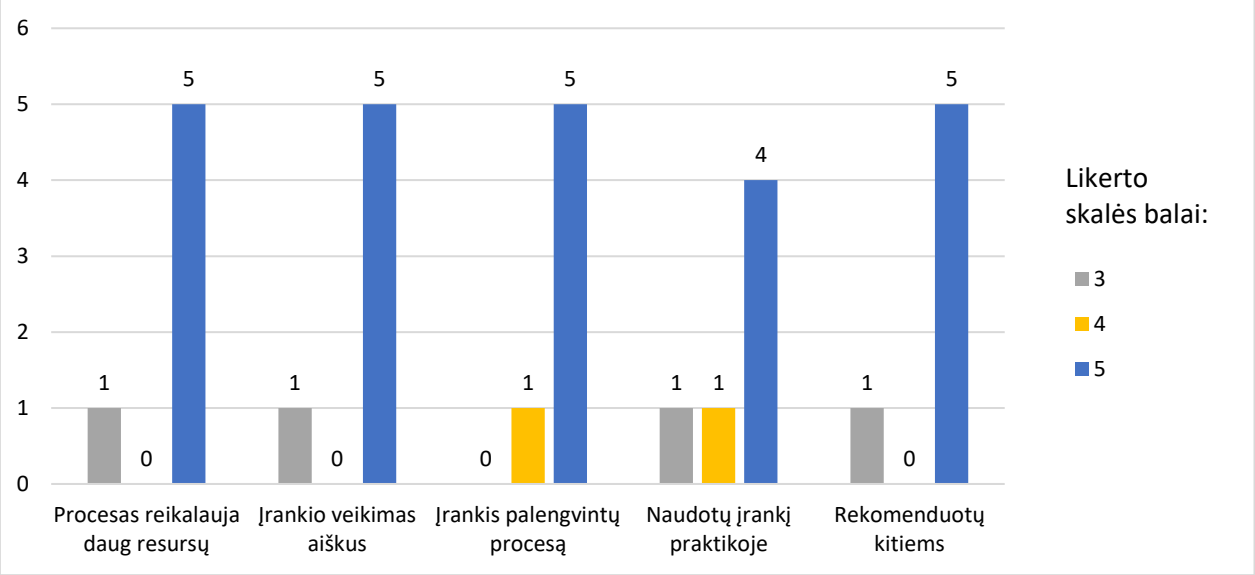
Šiems teiginiams vertinti buvo naudojama Likerto skalė nuo 1 iki 5, kur 1 reiškia „nesutinku“, o 5 – „sutinku“. Tokia skalė pasirinkta siekiant kiekybiškai įvertinti respondentų nuomonę ir sudaryti galimybę atlikti rezultatų palyginamąją analizę.

Apibendrinant gautus rezultatus, galima teigti, kad respondentai teigiamai vertino įrankio potencialą – dauguma sutiko, kad rankinių testavimo atvejų kūrimo procesas yra resursų reikalaujantis ir jo automatizavimas būtų naudingas. Taip pat didelė dalis respondentų nurodė, kad įrankio veikimo principai buvo suprantami, o pats įrankis galėtų palengvinti jų kasdienį darbą. Kiek nuosaikesni vertinimai pastebimi klausimuose, susijusiuose su praktiniu įrankio naudojimu ateityje ir rekomendavimu kitiems specialistams, kas gali būti siejama su tuo, kad įrankis yra prototipo stadijoje ir dar nėra pilnai integruotas į realius darbo procesus.

Kai kurie respondentai papildomai pateikė komentarus, kuriuose išskyrė įrankio privalumus, tokius kaip laiko taupymas ir struktūruotas testavimo atvejų generavimas, tačiau taip pat

paminėjo galimus tobulinimo aspektus, susijusius su funkcionalumo išplėtimu ir lankstumo didinimu.

Respondentų atsakymai, pateikti penkiabalėje *Likert* skalėje, buvo atvaizduoti apibendrintoje diagramoje (žr. 57 pav.).



57 pav. Apibendrinti apklausos rezultatai

Apklausos rezultatai rodo, kad respondentai teigiamai vertino automatizuoto testavimo atvejų generavimo idėją bei matė tokio sprendimo praktinį pritaikomumą testavimo procese. Respondentų vertinimai leidžia daryti prielaidą, kad rankinių testavimo atvejų kūrimas testavimo specialistams vis dar yra pakankamai daug laiko ir resursų reikalaujanti veikla, todėl automatizavimo sprendimai šioje srityje yra aktualūs.

Taip pat galima pastebėti, kad respondentams įrankio veikimo principas buvo suprantamas, o tai svarbu vertinant tokio sprendimo pritaikymo galimybes praktikoje. Dauguma respondentų nurodė, kad naudotų tokį įrankį savo veikloje bei rekomenduotų jį kitiems specialistams, todėl galima teigti, kad pasiūlytas sprendimas buvo vertinamas kaip potencialiai naudingas testavimo proceso efektyvinimui.

Visgi dalis respondentų kai kuriuos aspektus vertino atsargiau. Tai gali būti susiję su tuo, kad testavimo atvejų kūrimas praktikoje dažnai priklauso ne tik nuo formaliai aprašytų scenarijų, bet ir nuo projekto specifikos, verslo logikos bei individualios testuotojo patirties. Dėl šios priežasties automatizuotas generavimas gali būti vertinamas kaip pagalbinė priemonė testavimo procese, tačiau ne kaip visiškas rankinio darbo pakeitimas.

Išsamūs atsakymų pasiskirstymai pagal kiekvieną klausimą pateikiami pridėtame priede (žr. 1 priedas.

Testavimo ekspertų apklausos rezultatai pagal klausimą).

5.4. Sprendimo veikimo ir savybių analizė, kokybės kriterijų įvertinimas

Sukurto įrankio veikimas tiesiogiai priklauso nuo pateikiamų panaudojimo atvejų specifikacijos failo struktūros. Įrankis generuoja testavimo atvejus tuomet, kai specifikacijos failas yra suformatuotas pagal numatytus reikalavimus – pateikta UML panaudojimo atvejų diagrama bei nuosekliai aprašytos panaudojimo atvejų specifikacijos lentelės pagal apibrėžtą šabloną (žr. 28 pav. Minimalus panaudojimo atvejo specifikacijos lentelės šablonas). Sprendime naudojama standartizuota panaudojimo atvejų specifikacijos struktūra, paremta Volere principu, todėl įrankis gali identifikuoti panaudojimo atvejus, išanalizuoti scenarijus ir sugeneruoti naudotojo pasirinktus testavimo atvejus.

Panaudojimo atvejų diagramos nuskaitymas sprendime naudojamas ne tik testavimo atvejų generavimo procesui, bet ir pačių specifikacijų kokybės patikrai. Įrankis palygina diagramoje ir specifikacijos lentelėse esančius panaudojimo atvejus, todėl naudotojas gali greitai pastebėti trūkstamus ar nesutampančius įrašus, neteisingus pavadinimus bei galimus dokumentacijos netikslumus. Tokia papildoma analizė padeda įvertinti, ar specifikacijos failas yra parengtas nuosekliai ir tinkamas tolimesniam testavimo atvejų generavimui.

Svarbi sprendimo savybė yra galimybė naudotojui pasirinkti, kuriuos testavimo atvejus generuoti. Tai leidžia pritaikyti generavimo procesą pagal konkretaus projekto poreikius, pavyzdžiui, generuoti tik pagrindinių scenarijų, tik peržiūros funkcionalumo arba tik tam tikrų veiksmų testavimo atvejus. Tokia galimybė ypač naudinga didesniuose projektuose, kuriuose sugeneruojamų testavimo atvejų kiekis gali būti labai didelis.

Siekiant palengvinti šį procesą, įrankis automatiškai pasiūlo galimus panaudojimo atvejų požymius ir pažymi rekomenduojamas kategorijas. Naudotojas gali šiuos pasiūlymus peržiūrėti ir koreguoti prieš generuojant testavimo atvejus. Toks sprendimas padeda sumažinti rankinio darbo kiekį bei pagreitina generavimo procesą, ypač dirbant su didesnės apimties specifikacijomis.

Tačiau įrankio veikimas yra ribojamas tuo, kad jis nėra tinkamas netvarkingai parengtoms specifikacijoms. Jei specifikacijos faile nesilaikoma struktūros (pvz., neteisingai suformuotos lentelės, trūksta scenarijų, naudojami nevienodi pavadinimai ar terminai), įrankis gali neteisingai interpretuoti duomenis arba apskritai nesugebėti jų apdoroti. Taip pat apribojimai pasireiškia dirbant su sudėtingesniais scenarijais, kuriuose yra daug išsišakojimų ar neapibrėžtų veiksmų – tokiais atvejais sugeneruoti testavimo atvejai gali reikalauti papildomos peržiūros ir koregavimo. Dėl šių priežasčių įrankis šiuo metu labiausiai tinkamas naudoti struktūruotoje ir standartizuotoje reikalavimų aprašymo aplinkoje.

5.5. Sprendimo taikymo rekomendacijos

Sukurtą įrankį rekomenduojama taikyti projektuose, kuriuose reikalavimai aprašomi naudojant UML panaudojimo atvejų diagramas ir standartizuotas panaudojimo atvejų specifikacijų lenteles, pateiktas 2 skyriuje (žr. Testavimo atvejų generavimo metodikos reikalavimų specifikacija, formalizuotas aprašas). Siekiant užtikrinti korektišką įrankio veikimą, specifikacijų dokumente turi būti išlaikyta nuosekli lentelių struktūra – naudojami vienodi laukų pavadinimai, aiškiai atskirti pagrindiniai ir alternatyvūs scenarijai bei pateikta viena panaudojimo atvejų diagrama.

Prieš generuojant testavimo atvejus rekomenduojama atlikti pirminę specifikacijų analizę naudojant įrankį. Šiame etape įrankis palygina diagramoje ir lentelėse esančius panaudojimo atvejus bei leidžia identifikuoti trūkstamus ar nesutampančius įrašus. Jei nustatoma, kad dalis panaudojimo atvejų nėra aprašyti lentelėse arba skiriasi jų pavadinimai, rekomenduojama pirmiausia pakoreguoti specifikacijos dokumentą ir tik tuomet vykdyti testavimo atvejų generavimą.

Projektuose, kuriuose naudojamos kitokios specifikacijų struktūros, rekomenduojama adaptuoti įrankio duomenų nuskaitymo logiką. Tokiu atveju reikėtų koreguoti:

- lentelių laukų atpažinimo taisykles;
- scenarijų identifikavimo logiką;
- naudojamų raktažodžių ir sekcijų pavadinimų analizę;
- panaudojimo atvejų numeravimo bei pavadinimų nuskaitymo taisykles.

Jeigu projekte naudojami sudėtingesni scenarijai su dideliu kiekiu išsišakojimų arba nestandartiniais veiksmų aprašymais, rekomenduojama papildomai peržiūrėti sugeneruotus testavimo atvejus prieš juos naudojant testavimo procese. Tokiais atvejais gali būti reikalingas papildomas scenarijų detalizavimas arba rankinis testavimo atvejų koregavimas.

Įrankį rekomenduojama naudoti ankstyvose programinės įrangos kūrimo stadijose, kai yra rengiamos arba tikslinamos panaudojimo atvejų specifikacijos. Tokiu būdu įrankis gali būti naudojamas ne tik testavimo atvejų generavimui, bet ir pačių specifikacijų kokybės vertinimui. Diagramų ir lentelių palyginimas leidžia anksčiau pastebėti dokumentacijos netikslumus, trūkstamus scenarijus ar neaprašytus panaudojimo atvejus, todėl sumažėja rizika, kad klaidos bus pastebėtos tik testavimo arba programavimo etape.

Didelės apimties projektuose rekomenduojama naudoti įrankio galimybę pasirinkti generuojamų testavimo atvejų tipus. Ši funkcija leidžia generuoti tik aktualius scenarijus, pavyzdžiui, tik pagrindinius arba tik tam tikrų kategorijų testavimo atvejus. Papildomą naudą suteikia automatiniai įrankio pasiūlymai, kurie padeda naudotojui greičiau priskirti panaudojimo atvejams tinkamus požymius ir sumažina rankinio žymėjimo poreikį.

Išvados

1. Išanalizavus UML panaudojimo atvejų diagramas ir jų specifikacijas testavimo atvejų generavimo kontekste, nustatyta, kad specifikacija, parengta pagal Volere šabloną ir turinti tokius elementus kaip išankstinės sąlygos, naudotojo veiksmai bei teisės yra pakankamai informatyvūs, kad būtų galima suformuoti tiek rankinius, tiek automatinius testavimo atvejus.
2. Analizuojant testavimo atvejų struktūrą ir jų sudarymo principus, paaiškėjo, jog siekiant sukurti kokybišką testavimo atvejį remiantis UML panaudojimo atvejų diagrama ir jos specifikacija, būtina užtikrinti specifikacijos detalumą, joje turėtų būti aprašytos visos svarbiausios dalys – išankstinės sąlygos, naudotojo veiksmai, sistemos rezultatas ir teisės.
3. Išanalizavus esamus metodus, įrankius ir technologijas, skirtus testavimo atvejų generavimui iš UML diagramų, paaiškėjo, kad dauguma jų UML panaudojimo atvejų diagramas vertina tik kaip papildomą ar pagalbinį informacijos šaltinį. Siekiant turėti galimybę generuoti rankinius testavimo atvejus iš panaudojimo atvejų diagramų ir jų specifikacijų, būtų tikslinga sukurti metodiką, kurioje šios diagramos ir jų specifikacijos būtų naudojamos kaip pagrindinis informacijos šaltinis testavimo atvejų generavimui.
4. Ištyrus esamus metodus ir remiantis UML panaudojimo atvejų specifikacijomis, buvo pasiūlyta nauja metodika, kuri leidžia generuoti testavimo atvejus naudojantis UML panaudojimo atvejų diagramomis ir jų specifikacijomis. Metodikoje apibrėžta, kokia struktūra turi būti pateikiamos specifikacijos ir pagal kokias taisykles iš jų formuojami testavimo atvejai. Tai leidžia automatiškai generuoti vieningos struktūros testavimo scenarijus pagal pateiktus panaudojimo atvejų duomenis.
5. Naudojant *Python* programavimo kalbą sukurtas įrankis, kuris sugeneruoja testavimo atvejus iš pateikto UML panaudojimo atvejų diagramų ir jų specifikacijų failo DOCX formatu. Panaudojimo atvejų identifikavimui iš diagramos pasitelkiama *OpenAI* sąsaja, o testavimo atvejai suformuojami XLSX formatu, taikant specifikavimo dalyje nustatytas taisykles.
6. Eksperimentinio tyrimo metu trijuose projektuose buvo sugeneruoti 154 testavimo atvejai, kurie buvo lyginami su 136 rankiniu būdu sukurtais testavimo atvejais. Nustatyta, kad įrankis pilnai padengia pagrindinius ir specifikacijose aprašytus alternatyviuosius scenarijus, tačiau neidentifikuoja scenarijų, susijusių su išankstinėmis sąlygomis ar platesniu verslo logikos vertinimu, todėl rankiniu būdu sukurtų testavimo atvejų kiekis dažniausiai būna didesnis nei automatizuotu būdu sugeneruotų testavimo atvejų. Testavimo srities ekspertų apklausoje dalyvavę penki iš šešių respondentų įrankio funkcionalumo aiškumą bei naudingumą įvertino aukštais (keturiais arba penkiais) balais, o keturi iš šešių apklaustųjų nurodė, kad naudotų jį praktikoje, taigi galima teigti, kad įrankis galėtų palengvinti testuotojų darbą rankinių testavimo atvejų kūrimo procese.

Literatūros sąrašas

1. KANER, C. *What Is a Good Test Case?* 2003. Prieiga per internetą: https://kaner.com/pdfs/GoodTestSlides.pdf [žiūrėta 2026-03-20].
2. BHANUSHALI, A. Ensuring Software Quality Through Effective Quality Assurance Testing: Best Practices and Case Studies. *International Journal of Advances in Scientific Research and Engineering*, 2023, t. 9, nr. 10. Prieiga per internetą: https://www.researchgate.net/publication/375342628_Ensuring_Software_Quality_Through_Effective_Quality_Assurance_Testing_Best_Practices_and_Case_Studies [žiūrėta 2026-03-25].
3. SYLVAIN, H. Test Suite Generation for Boolean Conditions with Equivalence Class Partitioning. Iš: *Proceedings of the IEEE/ACM 10th International Conference on Formal Methods in Software Engineering (FormaliSE '22)*. New York: Association for Computing Machinery, 2022, p. 23–33. Prieiga per internetą: https://doi.org/10.1145/3524482.3527659 [žiūrėta 2026-03-25].
4. ANDERSSON, A. ir MALMELING, A. *Automated Test Case Generation from Software Requirements Using ChatGPT: A Comparison of Generative Methods in Black-Box Unit Testing*. 2024. Prieiga per internetą: https://hv.diva-portal.org/smash/get/diva2:1917788/FULLTEXT01.pdf [žiūrėta 2026-03-25].
5. WU, H. An Effective Equivalence Partitioning Method to Design the Test Case of the Web Application. Iš: *2012 International Conference on Systems and Informatics (ICSAI 2012)*. IEEE, 2012, p. 2524–2527 [žiūrėta 2026-02-14].
6. SWAIN, S. K.; MOHAPATRA, D. P. ir MALL, R. Test Case Generation Based on Use Case and Sequence Diagram. *International Journal of Software Engineering*, 2010, t. 3, nr. 2, p. 21–52.
7. PAHWA, N. ir SOLANKI, K. *UML Based Test Case Generation Methods: A Review*. 2014. Prieiga per internetą: https://www.researchgate.net/publication/269669135_UML_based_Test_Case_Generation_Methods_A_Review [žiūrėta 2026-03-25].
8. FAROOQ, M. S. ir TAHREEM, T. Requirement-Based Automated Test Case Generation: Systematic Literature Review. *VFAST Transactions on Software Engineering*, 2022, t. 10, nr. 2, p. 133–142.
9. ALAGARSAMY, S.; TANTITHAMTHAVORN, C. ir ALETI, A. A3Test: Assertion-Augmented Automated Test Case Generation. *Information and Software Technology*, 2024, t. 176, p. 107565.
10. DÍAZ, I.; LOSAVIO, F.; MATTEO, A. ir PASTOR, O. A Specification Pattern for Use Cases. *Information & Management*, 2004, t. 41, nr. 8, p. 961–975. Prieiga per internetą: https://www.sciencedirect.com/science/article/pii/S0378720603001502 [žiūrėta 2026-03-25].
11. GUO, X.; OKAMURA, H. ir DOHI, T. Optimal Test Case Generation for Boundary Value Analysis. *Software Quality Journal*, 2024, t. 32, p. 1–24.
12. SHOLEH, M.; GISFAS, I.; CAHIMAN ir FAUZI, M. A. Black Box Testing on ukmbantul.com Page with Boundary Value Analysis and Equivalence Partitioning Methods. *Journal of Physics: Conference Series*, 2021, t. 1823, nr. 1, p. 012029. Prieiga

per internetą: https://dx.doi.org/10.1088/1742-6596/1823/1/012029 [žiūrėta 2026-03-29].

13. PRAYUDHA, I.; HARTANI, R. ir DIVAYANA, Y. Boundary Value Analysis Testing Techniques on Learning Management System Applications. *International Journal of Engineering and Emerging Technology*, 2020, t. 4, p. 31.
14. SIANTURI, E. Boundary Value Analysis and Decision Table Testing Methods in Software Testing. *International Journal of Information Technology and Education*, 2022, t. 1, p. 124–129.
15. ALI SHAH, S. A.; SHAHZAD, R. K.; BUKHARI, S. S. A. ir HUMAYUN, M. Automated Test Case Generation Using UML Class & Sequence Diagram. *Current Journal of Applied Science and Technology*, 2016, t. 15, nr. 3, p. 1–12. Prieiga per internetą: https://journalcjast.com/index.php/CJAST/article/view/1345 [žiūrėta 2026-03-25].
16. DOBING, B. Understanding the Role of Use Cases in UML: A Review and Research Agenda. Prieiga per internetą: https://www.researchgate.net/publication/220373634_Understanding_the_Role_of_Use_Cases_in_UML_A_Review_and_Research_Agenda [žiūrėta 2026-03-25].
17. OBJECT MANAGEMENT GROUP (OMG). *Unified Modeling Language (UML) Specification Version 2.5.1*. 2017. Prieiga per internetą: https://www.omg.org/spec/UML/2.5.1/PDF [žiūrėta 2026-03-25].
18. TRINH, T.; NGUYEN, T.; LE, N. ir HA, N. V. FSTG: Few-Shot Test Case Generation from Use Case Specifications Using Large Language Models. Iš: *Computational Intelligence in Engineering*. Cham: Springer Nature Switzerland, 2026, p. 302–316 [žiūrėta 2026-03-08].
19. FOWLER, M. ir SCOTT, K. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 2003.
20. ROBERTSON, J. ir ROBERTSON, S. *Volere Requirements Specification Template*. 16 leid. 2012. Prieiga per internetą: https://www.cs.uic.edu/~i440/VolereMaterials/templateArchive16/c%20Volere%20template16.pdf [žiūrėta 2025-03-04].
21. HIBATULLAH, M. H. ir WIDYANI, Y. SRS for Software with Machine Learning Features. Iš: *2024 IEEE International Conference on Data and Software Engineering (ICoDSE)*. IEEE, 2024, p. 211–216 [žiūrėta 2026-03-21].
22. PRIMADIATI, R.; PRIYADI, Y. ir RICHASDY, D. Software Development for Object Formation for Sequence Diagrams Based on Use Case Description Extraction: Case Study SRS Scenery. Iš: *2023 10th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*. IEEE, 2023, p. 642–648 [žiūrėta 2026-04-03].
23. JACOBSON, I. ir COCKBURN, A. Use Cases Are Essential. *ACM Queue*, 2023, t. 21, nr. 5.
24. VRANIĆ, V.; LANG, J.; NORES, M. L.; ARIAS, J. J. P.; SOLANO, J. ir kt. Use Case Modeling in a Research Setting of Developing an Innovative Pilgrimage Support System. *Universal Access in the Information Society*, 2024, t. 23, nr. 4, p. 1543–1560 [žiūrėta 2026-02-27].

25. CASTILLO, N.; GÜEMEZ, A. ir AGUILAR VERA, R. Team Composition in Software Engineering Development Phases: A Systematic Literature Review. *International Journal of Combinatorial Optimization Problems and Informatics*, 2024, t. 15, p. 237–253.
26. RAKIĆ, M.; ZIZOVIC, M.; BOZA, M.; NJEGUŠ, A.; ZIZOVIC, M. ir kt. Multi-Criteria Selection of Standards for System Analyst Activities in Organizations. *Facta Universitatis Series Mechanical Engineering*, 2023.
27. ENOIU, E.; GAY, G.; ESBER, J. ir FELDT, R. Understanding Problem Solving in Software Testing: An Exploration of Tester Routines and Behavior. 2023. Prieiga per internetą: https://greg4cr.github.io/pdf/23problemmodel.pdf [žiūrėta 2026-03-25].
28. LIM, J. W.; CHIEW, T. K.; SU, M. T.; ONG, S.; SUBRAMANIAM, H. ir kt. Test Case Information Extraction from Requirements Specifications Using NLP-Based Unified Boilerplate Approach. *Journal of Systems and Software*, 2024, t. 211, p. 112005. Prieiga per internetą: https://www.sciencedirect.com/science/article/pii/S0164121224000487 [žiūrėta 2026-03-25].
29. VISHAL, V.; KOVACIOGLU, M.; KHERAZI, R. ir MOUSAVI, M. Integrating Model-Based and Constraint-Based Testing Using SpecExplorer. 2012. Prieiga per internetą: https://ieeexplore.ieee.org/document/6405445 [žiūrėta 2026-03-25].
30. VEANES, M.; CAMPBELL, C.; GRIESKAMP, W.; SCHULTE, W.; TILLMANN, N. ir kt. Model-Based Testing of Object-Oriented Reactive Systems with Spec Explorer. Iš: *Formal Methods and Testing: An Outcome of the FORTEST Network, Revised Selected Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, p. 39–76. Prieiga per internetą: https://doi.org/10.1007/978-3-540-78917-8_2 [žiūrėta 2025-01-19].
31. GRIESKAMP, W.; TILLMANN, N. ir VEANES, M. Instrumenting Scenarios in a Model-Driven Development Environment. *Information and Software Technology*, 2004, t. 46, p. 1027–1036.
32. KHERRAZI, R. Using Tabular Notation to Support Model Based Testing: A Practical Experience Using STTSpec and Spec Explorer. Iš: *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2020, p. 18–23 [žiūrėta 2026-04-11].
33. MICROSOFT. *Spec Explorer*. 2013. Prieiga per internetą: https://learn.microsoft.com/en-us/previous-versions/visualstudio/spec-explorer/ee620411(v=specexplorer.10) [žiūrėta 2025-01-19].
34. SVING, R. ir ÖMAN, P. *Pilot Project for Model Based Testing Using Conformiq Qtronic*. 2010. Prieiga per internetą: https://www.diva-portal.org/smash/get/diva2:343741/FULLTEXT01.pdf [žiūrėta 2026-03-25].
35. KHAN, M. B. A. ir SHANG, S. *Evaluation of Model Based Testing and Conformiq Qtronic*. 2009. Prieiga per internetą: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A249286&dswid=891 [žiūrėta 2026-03-25].
36. SULI, S. *An Evaluation of Model-Based Testing for an Industrial Train Control Software*. 2018. Prieiga per internetą: https://mdh.diva-portal.org/smash/get/diva2:1220927/FULLTEXT01.pdf [žiūrėta 2026-03-25].
37. CONFORMIQ INC. *Conformiq Creator 2 Tutorial*. 2014. Prieiga per internetą: https://people.scs.carleton.ca/~jeanpier/4004F18/T7%20MBT%20tools/2-%20Conformiq/10-%20Conformiq-Creator-2-Tutorial.pdf [žiūrėta 2025-01-19].

38. CAVALCANTI, A.; MIYAZAWA, A.; SCHULZE, U. ir TIMMIS, J. Bringing RoboStar and RT-Tester Together. Iš: *Applicable Formal Methods for Safe Industrial Products: Essays Dedicated to Jan Peleska on the Occasion of His 65th Birthday*. Springer, 2023, p. 16–33.
39. MIYAZAWA, A.; SCHULZE, U. ir TIMMIS, J. Bringing RoboStar and RT-Tester Together. *Applicable Formal Methods for Safe Industrial Products: Essays Dedicated to Jan Peleska on the Occasion of His 65th Birthday*, 2023, t. 14165, p. 16.
40. PELESKA, J.; VOROBEV, E.; LAPSCHIES, F. ir ZAHLTEN, C. *Automated Model-Based Testing with RT-Tester Technical Report 2011-05-25*. 2011. Prieiga per internetą: https://www.informatik.uni-bremen.de/agbs/testingbenchmarks/turn_indicator/tool/rtt-mbt.pdf [žiūrėta 2026-03-25].
41. MEYER, M. D. O. ir PELESKA, J. Automated Testing with RT-Tester – Theoretical Issues Driven by Practical Needs. Prieiga per internetą: https://www.researchgate.net/publication/2540960_Automated_Testing_with_RT-Tester_-_Theoretical_Issues_Driven_by_Practical_Needs [žiūrėta 2026-03-25].
42. BRAUER, J.; MÖLLER, O.; POP, A. ir MÖLLER, O. *Distributed Testing and Simulation Network*. 2015. Prieiga per internetą: https://into-cps.org/fileadmin/into-cps.org/Filer/D5.1b_Testing.pdf [žiūrėta 2025-01-19].
43. SHAFIQUE, M. ir LABICHE, Y. A Systematic Review of State-Based Test Tools. *International Journal on Software Tools for Technology Transfer*, 2015, t. 17, nr. 1, p. 59–76. Prieiga per internetą: https://doi.org/10.1007/s10009-013-0291-0 [žiūrėta 2025-01-19].
44. THANKI, M. ir SHINDE, P. Test Case Generation and Minimization Using UML Activity Diagram in Model Driven Environment. *International Journal of Computer & Organization Trends*, 2014, t. 9, p. 41–44.
45. *TestOptimal V7*. Prieiga per internetą: https://testoptimal.com/v7/wiki/doku.php?id=start [žiūrėta 2025-01-19].
46. ALYABROODI, Z.; ABUASAL, S.; ALAMAREEN, A. B.; AL-MASHAGBEH, M. ir ANGAWI, M. A Comparison Study for Test Case Management Tools. Iš: *Artificial Intelligence and Economic Sustainability in the Era of Industrial Revolution 5.0*. Cham: Springer Nature Switzerland, 2024, p. 563–570. Prieiga per internetą: https://doi.org/10.1007/978-3-031-56586-1_41 [žiūrėta 2025-01-19].
47. TESTCASELAB. *Features*. 2026. Prieiga per internetą: https://www.testcaselab.com/features [žiūrėta 2025-01-19].
48. TAO, C.; GAO, J. ir WANG, T. Testing and Quality Validation for AI Software: Perspectives, Issues, and Practices. *IEEE Access*, 2019, t. 7, p. 120164–120175.
49. BAQAR, M. ir KHANDA, R. The Future of Software Testing: AI-Powered Test Case Generation and Validation. *arXiv preprint arXiv:2409.05808*, 2024.
50. LIU, K.; CHEN, Z.; LIU, Y.; ZHANG, J. M.; HARMAN, M. ir kt. LLM-Powered Test Case Generation for Detecting Bugs in Plausible Programs. 2025, p. 430–440. Prieiga per internetą: https://aclanthology.org/2025.acl-long.20/ [žiūrėta 2026-03-25].

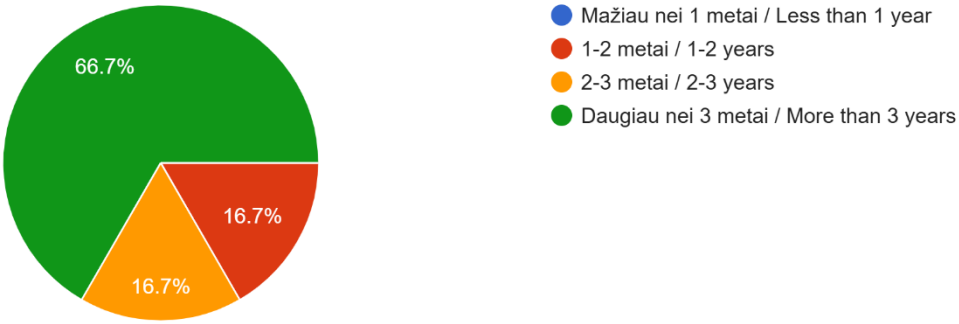
Priedai

1 priedas.

Testavimo ekspertų apklausos rezultatai pagal klausimą

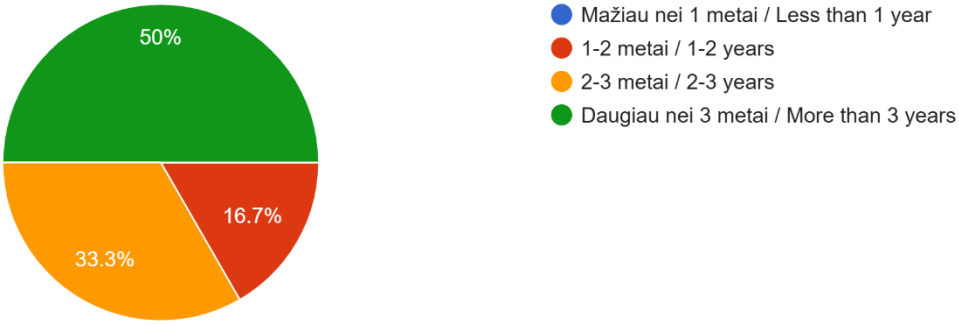
Kiek metų patirties turite programinės įrangos testavime? How many years of experience do you have in software testing?

6 responses



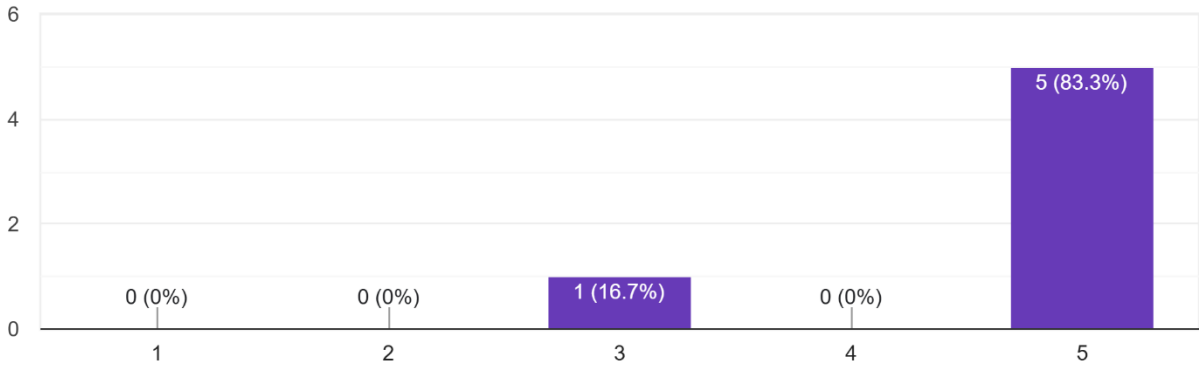
Kiek patirties turite rankinių testavimo atvejų kūrime? How much experience do you have in creating manual test cases?

6 responses



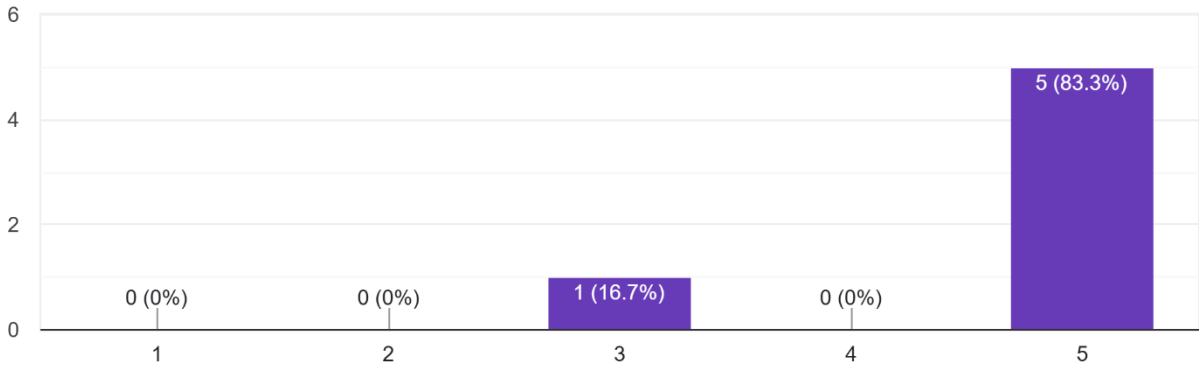
Rankinių testavimo atvejų kūrimas yra daug resursų reikalaujantis procesas, todėl jo automatizavimas būtų naudingas. Manual test case...ss; therefore, its automation would be beneficial.

6 responses



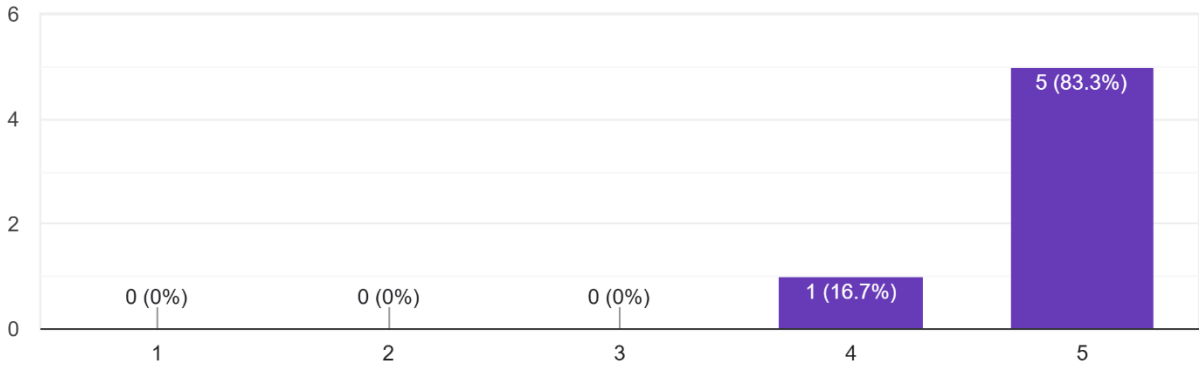
Automatizuoto rankinių testavimo atvejų generavimo įrankio veikimo principai man buvo aiškūs ir suprantami. The operating principles of the autom...neration tool were clear and understandable to me.

6 responses



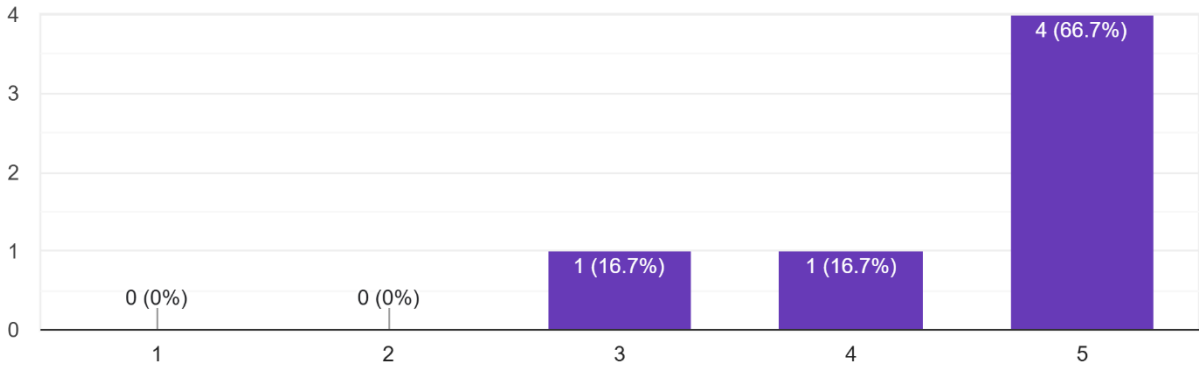
Pateiktas automatizuoto rankinių testavimo atvejų generavimo įrankis gali palengvinti rankinių testavimo atvejų kūrimo procesą. The presented au...ilitate the process of creating manual test cases.

6 responses



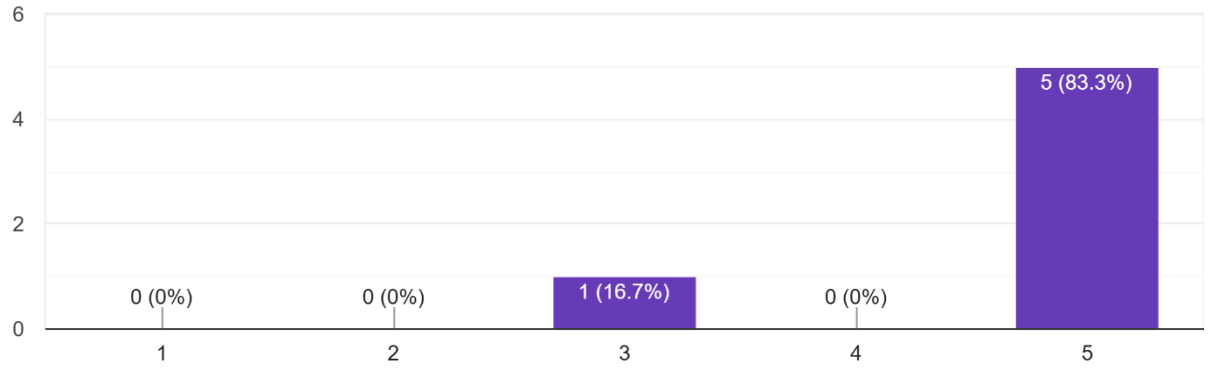
Jeigu man reikėtų kurti rankinius testavimo atvejus pagal UML panaudojimo atvejų specifikacijas, naudočiau šį įrankį. If I had to create manual test ... UML use case specifications, I would use this tool.

6 responses



Rekomenduočiau šį įrankį kitiems specialistams, kuriems tenka kurti rankinius testavimo atvejus. I would recommend this tool to other professionals who are responsible for creating manual test cases.

6 responses



2 Priedas.

Straipsnis IVUS 2026 konferencijoje

The Use of UML Use Case Diagrams and Their Specifications for Test Case Generation

Ieva Valatkevičiūtė [1], Lina Čeponienė [1] and Tomas Valatkevičius [1]

[1] *Faculty of Informatics, Kaunas University of Technology, Studentų St. 50, Kaunas, 51368, Lithuania.*

Abstract

Manual test cases remain an important part of software testing, but their creation is often time-consuming and requires significant effort. In this paper, we propose a method for generating manual test cases based on UML use case diagrams and their specifications. The approach uses structured information from use case specification tables, which are created based on an adapted Volere template, and allows the user to participate in the generation process by selecting relevant test scenarios. A test case generation tool implementing the proposed method was developed using Python. The tool performs an initial analysis of the uploaded specification, identifies use cases, and generates test cases according to predefined rules. Evaluation results demonstrate that the proposed approach can successfully generate structured test cases from UML use case specifications and thus support the software testing process.

Keywords

UML, use case specification, manual testing, test cases.