



Kauno technologijos universitetas

Informatikos fakultetas

**Automatizuotas naudotojo pasakojimų reikalavimų
išgavimas iš *EventStorming* metodu gautų artefaktų**

Magistro baigiamasis projektas

Projektą parengė

Antanas Ramanauskas

Projektui vadovavo

Doc. dr. Mantas Jurgelaitis

Kaunas, 2026



Kauno technologijos universitetas

Informatikos fakultetas

Automatizuotas naudotojo pasakojimų reikalavimų išgavimas iš *EventStorming* metodu gautų artefaktų

Magistro baigiamasis projektas

Veiklos skaitmeninimas ir sistemų architektūros (6211BX009)

Projektą parengė

Antanas Ramanauskas

Projektui vadovavo

Doc. dr. Mantas Jurgelaitis

Projektą recenzavo

Doc. dr. Kęstutis Kapočius

Kaunas, 2026



Kauno technologijos universitetas

Informatikos fakultetas

Antanas Ramanauskas

Automatizuotas naudotojo pasakojimų reikalavimų išgavimas iš *EventStorming* metodu gautų artefaktų

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 sąrašė;
3. baigiamajame projekte tinkamai laikiausi asmens duomenų apsaugos reikalavimų, nenaudojau neskelbtinų ar konfidencialių duomenų be teisėto pagrindo, o jei juos naudoju, 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.

Ramanauskas, Antanas. Automatizuotas naudotojo pasakojimų reikalavimų išgavimas iš *EventStorming* metodu gautų artefaktų. Magistro baigiamasis projektas / projektui vadovavo doc. dr. Mantas Jurgelaitis; Kauno technologijos universitetas, Informatikos fakultetas.

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

Reikšminiai žodžiai: reikalavimų išgavimas; *EventStorming*; naudotojo pasakojimai; automatizavimas.

Kaunas, 2026. 86 p.

Santrauka

Programinės įrangos kūrimo įmonėse, prieš rengiant komercinį pasiūlymą ir sudarant sutartį, dažnai sudaromas reikalavimų sąrašas, tačiau šis procesas yra sudėtingas, reikalaujantis laiko ir papildomų sąnaudų, vienas iš metodų yra *EventStorming* dirbtuvės. Atlikus egzistuojančių sprendimų analizę nustatyta, kad nėra pakankamai išplėtotų sprendimų, leidžiančių automatizuotai išgauti reikalavimus būtent iš *EventStorming* artefaktų. Dėl to šiame darbe siekiama palengvinti reikalavimų išgavimą, automatizuojant jų sudarymą iš *EventStorming* metu sukurtų artefaktų.

Darbe apibrėžti kuriamo įrankio reikalavimai, pasiūlyta metodika automatizuotai išgauti reikalavimus iš *EventStorming* artefaktų ir realizuotas įrankis. Įrankis susideda iš penkių pagrindinių komponentų: artefaktų atpažinimo, teksto atpažinimo, artefaktų klasterizavimo, reikalavimų generavimo ir reikalavimų eksportavimo. Lipnių lapelių aptikimui pasirinkta *SSD MobileNet V1 FPN* architektūra, teksto atpažinimui naudotas *Google Vision API OCR* sprendimas, o reikalavimų generavimui panaudotas instrukcijomis grįstą agentą naudojant didįjį kalbos modelį.

Sprendimas įvertintas trimis eksperimentais. Lipnių lapelių aptikimo eksperimente pasiektas 0,9626 preciziškumas, 0,8869 atkūrimas ir 0,9232 F1 įvertis. Artefaktų tipų atpažinimo eksperimente nustatytas 94,03% bendras tikslumas. Įrankio, sugeneruotų naudotojo pasakojimų ir viso proceso praktinis naudingumas buvo vertintas respondentų apklausa, kurioje visų trijų vertinamų teiginių vidurkiai svyravo nuo 4,44 iki 4,56 balo iš 5, o 89,3% visų įverčių sudarė 4 arba 5 balai. Gauti rezultatai rodo, kad sukurtas įrankis palengvina reikalavimų išgavimą iš *EventStorming* artefaktų ir yra praktiškai naudingas realiuose projektuose.

Ramanauskas, Antanas. Automated User Story Requirement Extraction from Artefacts Obtained Using the EventStorming Method. Master's Final Degree Project / supervisor assoc. prof. Mantas Jurgelaitis; Faculty of Informatics, Kaunas University of Technology.

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

Keywords: requirements elicitation; EventStorming; user stories; automation

Kaunas, 2026. 86 pages.

Summary

In software development companies, a requirements list is often prepared before creating a commercial proposal and signing a contract, but this process is complex, time-consuming, and costly, one of such methods is *EventStorming* workshop. The analysis of existing solutions showed that there are no sufficiently developed solutions that would enable automated requirements extraction specifically from *EventStorming* artifacts. Therefore, this thesis aims to facilitate requirements elicitation by automating the transformation of *EventStorming* artifacts into structured requirements.

The thesis defines the requirements for the proposed tool, introduces a methodology for automated requirements extraction from *EventStorming* artifacts, and presents the implemented tool. The tool consists of five main components: artifact recognition, text recognition, artifact clustering, requirements generation, and requirements export. Sticky note detection was implemented using the *SSD MobileNet V1 FPN* architecture, text recognition was performed using *Google Vision API OCR*, and requirements generation was carried out by instruction-tuned agent using large language model.

The developed solution was evaluated in three experiments. In the sticky note detection experiment, the model achieved 0.9626 precision, 0.8869 recall, and an F1-score of 0.9232. In the artifact type recognition experiment, the overall accuracy reached 94.03%. The practical usefulness of the tool, the generated user stories, and the overall process was evaluated through a respondent survey, where the mean scores of the three evaluated statements ranged from 4.44 to 4.56 out of 5, and 89.3% of all ratings were either 4 or 5. The results show that the developed tool can facilitate requirements extraction from *EventStorming* artifacts and is practically useful in real-world projects.

Turinys

Lentelių sąrašas	8
Paveikslų sąrašas	9
Santraupų ir terminų sąrašas	11
Įvadas.....	12
1. Probleminės srities analizė	14
1.1. Analizės tikslas.....	14
1.2. Tyrimo objektas, sritis ir problema.....	14
1.3. Reikalavimų inžinerijos procesų analizė.....	14
1.3.1. Reikalavimų inžinerija.....	14
1.3.2. Sistemų kūrimas remiantis <i>Domain-Driven Design</i>	18
1.3.3. Reikalavimų išgavimas remiantis <i>EventStorming</i>	21
1.4. Reikalavimų inžinerijos procesų naudotojų analizė	25
1.5. Vaizdų atpažinimo galimybės naudojant mašininį mokymąsi	26
1.5.1. SVM panaudojimas objektams atpažinti	26
1.5.2. Objektų atpažinimas naudojant <i>CNN</i>	27
1.6. Teksto atpažinimo galimybės	29
1.6.1. OCR įrankių palyginimas	29
1.7. Teksto apdorojimo ir reikalavimų generavimo galimybės	30
1.7.1. <i>LLM</i> palyginimas teksto apdorojimui ir reikalavimų generavimui	31
1.8. Reikalavimų išgavimo iš <i>EventStorming</i> ar panašių artefaktų metodų analizė	33
1.9. Reikalavimų išgavimas iš <i>EventStorming</i> artefaktų naudojant <i>LLM</i> įrankius	36
1.10. Siekiamo sprendimo apibrėžimas.....	38
1.11. Analizės išvados.....	38
2. Reikalavimų išgavimo iš <i>EventStorming</i> artefaktų įrankio specifikacija, formalus aprašas	39
2.1. Reikalavimų specifikacija	39
2.2. Dalykinės srities modelis	44
2.3. Naudotojų sąsajos modelis	45
2.4. Formalus sprendimo aprašas.....	47
2.5. Artefaktų atpažinimo komponentas	48
2.6. Teksto atpažinimo komponentas.....	50
2.7. Klasterizavimo komponentas	52
2.8. Reikalavimų generavimo komponentas – generavimas	55
2.9. Naudotojo pasakojimų generavimo <i>LLM</i> agentas	56
2.10. Reikalavimų generavimo komponentas – eksportavimas.....	58
3. Sprendimo realizacijos projektas	59
3.1. Duomenų rinkinio sudarymas lapelių atpažinimo apmokymui	60
3.2. Lapelių atpažinimo modelio apmokymas.....	62
3.3. Testavimo modelis, duomenys, rezultatai	63
3.3.1. Artefaktų ir lipnių lapelių atpažinimo komponento testavimas.....	63
3.3.2. Teksto atpažinimo (<i>OCR</i>) komponento testavimas	65
3.3.3. Klasterizavimo komponento testavimas	65
3.3.4. Reikalavimų generavimo komponentas.....	66
4. Eksperimentinis reikalavimų išgavimo iš <i>EventStorming</i> artefaktų įrankio tyrimas.....	69

4.1. Eksperimento planas.....	69
4.2. Lapelių atpažinimo eksperimentas	69
4.2.1. Lapelių atpažinimo eksperimento duomenų rinkinys	69
4.2.2. Lapelių atpažinimo eksperimento vykdymo eiga ir naudojamos metrikos	70
4.2.3. Lapelių atpažinimo eksperimento rezultatai.....	70
4.3. <i>EventStorming</i> artefaktų atpažinimo eksperimentas	71
4.3.1. <i>EventStorming</i> artefaktų atpažinimo duomenų rinkinys	71
4.3.2. <i>EventStorming</i> artefaktų atpažinimo eksperimento vykdymo eiga ir naudojamos metrikos.....	72
4.3.3. <i>EventStorming</i> artefaktų atpažinimo eksperimento rezultatai.....	72
4.4. Įrankio, išgautų reikalavimų bei proceso vertinimas	73
4.4.1. Įrankio, išgautų reikalavimų bei proceso vertinimo apklausos duomenys.....	73
4.4.2. Įrankio, išgautų reikalavimų bei proceso vertinimo eksperimento vykdymo eiga ir naudojamos metrikos	77
4.4.3. Įrankio, išgautų reikalavimų bei proceso vertinimo rezultatai.....	78
4.5. Sprendimo taikymo rekomendacijos.....	79
4.5.1. Sprendimo pritaikymas	79
4.5.2. Sprendimo trūkumai ir tobulinimas	79
4.5.3. Spalvų paletės pasirinkimas	79
4.5.4. Artefaktų nuotraukos įkėlimas.....	80
4.5.5. Lipnių lapelių aptikimo komponento tobulinimas.....	80
Išvados.....	82
Literatūros sąrašas.....	83
Priedai	87
1 priedas	87
2 priedas	92

Lentelių sąrašas

1 lentelė. <i>OCR</i> įrankių palyginimas	30
2 lentelė. <i>LLM</i> palyginimas [53]	32
3 lentelė. Esamų problemos sprendimo būdų palyginimas	35
4 lentelė. <i>LLM</i> agento instrukcijos tikslinimo pagrindinės iteracijos įrankio testavimo metu	67
5 lentelė. Lapelių atpažinimo rezultatų lentelė kiekvienam <i>k-fold</i> padalijimui	71
6 lentelė. <i>EventStorming</i> artefaktų atpažinimo eksperimento rezultatų lentelė	72
7 lentelė. Respondentų pasiskirstymas pagal patirtį	77
8 lentelė. Įrankio, išgautų reikalavimų bei proceso vertinimo apklausos rezultatai	78

Paveikslų sąrašas

1.1 pav. Reikalavimų inžinerijos proceso modelis [6].....	15
1.2 pav. Suprantamumo kitimo grafikas pagal reikalavimo detalumo lygį [14].....	17
1.3 pav. Internetinio aukciono dalykinės srities smulkesnių dalykinių sričių apipavidalinimas [23]	19
1.4 pav. <i>Onion</i> architektūros reprezentacija [23].....	20
1.5 pav. Knygos prenumeravimo dalykinės srities <i>EventStorming</i> metodo artefaktai [3].....	21
1.6 pav. <i>EventStorming</i> komandos sąryšio su dalykinės srities įvykiu notacija [3]	22
1.7 pav. <i>EventStorming</i> taisyklės reprezentacija.....	22
1.8 pav. Picos užsakymo fragmentas naudojant <i>EventStorming</i> metodą [3]	23
1.9 pav. Proceso modeliavimas naudojant <i>EventStorming</i> artefaktus [3].....	23
1.10 pav. Dalykinės srities įvykių išdėstymas chronologine tvarka [3].....	24
1.11 pav. <i>EventStorming</i> iteratyvus procesas tobulinant dalykinės srities modelį [3].....	25
1.12 pav. Tyrimo objekto naudotojų veiklos panaudojimo atvejų diagrama.....	25
1.13 pav. <i>SVM</i> vizualizacija [27]	26
1.14 pav. Dviejų klasių duomenys plokštumoje (kairė) bei dviejų klasių duomenys išdėstyti erdvėje (dešinė) [26].....	27
1.15 pav. įprasta konvoliucinio neuroninio tinklo architektūra	28
1.16 pav. Straipsnio „Intelligent Chatbot for Requirements Elicitation and Classification“ įrankio veikimo konceptas [55]	34
1.17 pav. Straipsnio „ELICA: An Automated Tool for Dynamic Extraction of Requirements Relevant Information“ įrankio veikimo konceptas.....	34
1.18 pav. <i>EventStorming</i> paveikslėlis naudotas reikalavimų išgavimui su bendros paskirties LLM	36
2.1 pav. Reikalavimų iš <i>EventStorming</i> artefaktų išgavimo įrankio panaudojimo atvejų diagrama	39
2.2 pav. „Įkelti <i>EventStorming</i> artefaktų paveikslą“ panaudojimo atvejo veiklos diagrama.....	40
2.3 pav. „Generuoti reikalavimus“ panaudojimo atvejo veiklos diagrama.....	41
2.4 pav. „Valdyti reikalavimus“ panaudojimo atvejo veiklos diagrama.....	42
2.5 pav. „Eksportuoti reikalavimus“ panaudojimo atvejo veiklos diagrama	43
2.6 pav. reikalavimų išgavimo iš <i>EventStorming</i> artefaktų įrankio esybių klasių diagrama	44
2.7 pav. Projekto kūrimo lango eskizas	45
2.8 pav. Projekto ir reikalavimų valdymo lango eskizas (kairė) ir projektų sąrašo lango eskizas (dešinė)	46
2.9 pav. Reikalavimų išgavimo iš <i>EventStorming</i> artefaktų įrankio veiklos diagrama	47
2.10 pav. Artefaktų tipo atpažinimo proceso veiklos diagrama.....	48
2.11 pav. <i>MobileNet V1</i> konvoliucinio tinklo architektūra [57]	49
2.12 pav. Artefaktų atpažinimas <i>EventStorming</i> paveiksle.....	49
2.13 pav. teksto regiono <i>Offers accepted</i> atpažinimas ir koordinatės (kairėje) bei OCR įrankio atsakymo fragmentas (dešinėje)	50
2.14 pav. <i>Offers accepted</i> artefaktas su jam priskirtu tekstu.....	51
2.15 pav. Artefaktų klasterizavimo proceso veiklos diagrama	52
2.16 pav. <i>EventStorming</i> paveikslas po klasterizavimo	54
2.17 pav. Reklaminių pasiūlymų klasteris (kairėje) bei įdėjimo į krepšelį klasteris (dešinėje).....	54
2.18 pav. Reikalavimų generavimo veiklos diagrama	55
2.19 pav. Projekto ir klasterių informacija <i>JSON</i> duomenų struktūroje	56
2.20 pav. Instrukcijos dalys <i>LLM</i> agento pritaikymui dirbti su <i>EventStorming</i> metodu	57

2.21 pav. Naudotojo pasakojimų generavimo <i>LLM</i> agento atsakymas į 2.18 pav. nurodytą užklausą	58
2.22 pav. Reikalavimų eksportavimas csv formatu	58
2.23 pav. Reikalavimų .csv schema integracijai su <i>Jira</i> sistema [58].....	58
3.1 pav. <i>EventStorm2Req</i> sistemos diegimo diagrama	59
3.2 pav. Įrankio komponentų diagrama.....	60
3.3 pav. Vaizdų anotavimas naudojant <i>Label Studio</i> programinę įrangą.....	61
3.4 pav. Anotuoto vaizdo duomenys naudojant <i>Label Studio</i> programinę įrangą	61
3.5 pav. Savybių piramidės tinklo architektūra [60]	62
3.6 pav. Atpažintų lapelių panaudojant apmokytą modelį ribų atvaizdavimas.....	63
3.7 pav. Testavimo atvejo nuotrauka, kai lapelis yra pasuktas ir yra lapelių nuotraukos krašte	64
3.8 pav. Lapeliai ir juose atpažintos dominuojančios spalvos	64
3.9 pav. Teksto atpažinimo komponento rezultatai testavimo metu.....	65
3.10 pav. Klasterizavimo komponento testavimo metu atpažintas klasteris.....	66
3.11 pav. Reikalavimų generavimo komponento testavimo metu sukurtas naudotojo pasakojimas	66
4.1 pav. Fizinių lipnių lapelių nuotrauka iš duomenų rinkinio (kairėje) bei <i>miro.com</i> įrankio lipnus lapelis iš duomenų rinkinio (dešinėje).....	69
4.2 pav. Apklausoje naudotas <i>EventStorming</i> ir sugeneruotų naudotojo pasakojimų pavyzdys pirma dalis.....	75
4.3 pav. Apklausoje naudotas <i>EventStorming</i> ir sugeneruotų naudotojo pasakojimų pavyzdys antra dalis.....	76

Santraupų ir terminų sąrašas

API (angl. *Application Programming Interface*) – programinės sąsajos rinkinys, leidžiantis skirtingoms sistemoms ar programoms komunikuoti ir keisti duomenimis.

CNN (angl. *Convolutional Neural Network*) – konvoliucinis neuroninis tinklas, naudojamas vaizdų požymiams išgauti, objektams klasifikuoti ir aptikti.

DDD (angl. *Domain-Driven Design*) – sistemų kūrimo metodas, orientuotas į dalykinės srities modeliavimą ir jos sąvokų atspindėjimą programinės įrangos architektūroje.

FPN (angl. *Feature Pyramid Network*) – savybių piramidės tinklas, naudojamas skirtingų mastelių požymiams apdoroti objektų aptikimo uždaviniuose.

Jira – projektų valdymo ir užduočių sekimo įrankis, dažnai naudojamas programinės įrangos kūrimo projektuose.

LAB – spalvų erdvė, naudojama spalvoms aprašyti ir lyginti pagal šviesumą bei spalvines dedamasias, geriau atitinkanti žmogaus regimąjį spalvų suvokimą nei RGB.

LLM (angl. *Large Language Model*) – didelis kalbos modelis, naudojamas natūralios kalbos supratimui ir generavimui.

OCR (angl. *Optical Character Recognition*) – optinis simbolių atpažinimas, skirtas tekstui iš vaizdų paversti į skaitmeninį, redaguojamą tekstą.

ReLU (angl. *Rectified Linear Unit*) – aktyvacijos funkcija, dažnai naudojama neuroniniuose tinkluose požymiams išryškinti ir nelinearumui įvesti.

RNN (angl. *Recurrent Neural Network*) – rekurentinis neuroninis tinklas, naudojamas sekų ir nuoseklios informacijos apdorojimui.

SSD (angl. *Single Shot Detector*) – vieno žingsnio objektų aptikimo architektūra, vienu metu prognozuojanti objektų klases ir jų ribas vaizde.

YOLO (angl. *You Only Look Once*) – vieno žingsnio objektų aptikimo metodas, skirtas greitam objektų atpažinimui vaizduose.

Delta E – metrika, naudojama dviejų spalvų skirtumui įvertinti perceptiniu požiūriu.

CIE76 – vienas iš Delta E spalvų skirtumo skaičiavimo būdų, paremtas atstumu LAB spalvų erdvėje.

Ivadas

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

Neretai programinės įrangos kūrimo įmonės, prieš sudarant galutinį komercinį pasiūlymą ir pasirašant programinės įrangos kūrimo sutartį sudaro reikalavimų sąrašą suderintą su klientu [1]. Reikalavimų surinkimas, gali būti vykdomas įvairiais metodais [2], vienas jų – *EventStorming* seminarais pagrįstas metodas, kurio metu išsiaiškinama dalykinės srities informacija, užsakovo norai ir sukuriami artefaktai apibūdinantys planuojamą kurti sistemą [3]. Vėliau remiantis šiais artefaktais yra sudaromas reikalavimų sąrašas, naudojamas komerciniam pasiūlymui bei programinės įrangos kūrimo sutarčiai sudaryti. Įmonei yra kritiškai svarbu atlikti reikalavimų sudarymo etapus siekiant išsiaiškinti programinės įrangos kūrimo kaštus bei pateikti įmonei naudingą komercinį pasiūlymą [1], deja, šis procesas yra kompleksiškas ir reikalauja laiko [4] – už kurį užsakovas dažniausiai apmokėti nenori, todėl kyla poreikis palengvinti šį procesą, o konkrečiai šio darbo kontekste – automatizuoti reikalavimų išgavimą iš *EventStorming* artefaktų.

Šio darbo tyrimo sritis – reikalavimų inžinerijoje taikomi metodai ir jų sudarymo procesas. Tyrimo objektas – reikalavimų inžinerijos procesų automatizavimas, orientuotas į efektyvesnį reikalavimų išgavimo procesą.

Darbo tikslas – palengvinti reikalavimų surinkimo procesą, automatizuojant reikalavimų išgavimą iš *EventStorming* artefaktų. Šiam tikslui pasiekti keliami tokie uždaviniai:

1. Išanalizuoti reikalavimų inžinerijos procesą ir jo taikymą programinės įrangos kūrimo procese, *EventStorming* reikalavimų išgavimo metodą, taip pat glaudžiai susijusį *Domain-Driven Design* sistemų kūrimo metodą.
2. Išanalizuoti esamas reikalavimų išgavimo priemones iš *EventStorming* ar kitų panašių artefaktų.
3. Apibrėžti įrankio automatizuojančio reikalavimų išgavimą iš *EventStorming* artefaktų reikalavimus.
4. Pasiūlyti metodiką automatizuotai išgauti reikalavimus iš *EventStorming* artefaktų.
5. Suprojektuoti ir realizuoti automatinį reikalavimų išgavimo iš *EventStorming* artefaktų įrankį.
6. Eksperimentiškai ištirti, ar sukurtas įrankis palengvina reikalavimų išgavimą iš *EventStorming* artefaktų, įvertinti jo veikimą, sugeneruotų reikalavimų ir automatizuoto proceso praktinį naudingumą, bei apibendrinti tyrimo rezultatus.

Darbo rezultatai ir jų svarba

Sukurtas įrankis, kuris automatizuotai sugeneruotų reikalavimus iš *EventStorming* artefaktų ir tokiu sprendimu palengvinti reikalavimų sudarymo procesą. Darbo naujumas pasireiškia tuo, kad siūlomas metodas leidžia struktūrizuoti ir automatizuotai transformuoti *EventStorming* artefaktus į naudotojo pasakojimus. Praktinė darbo vertė – galimybė sumažinti laiko sąnaudas reikalavimų surinkimo etape. Sukurtas sprendimas gali būti

pritaikomas realiose programinės įrangos kūrimo įmonėse, siekiančiose optimizuoti ankstyvuosius projektų etapus.

Generatyvinio dirbtinio intelekto naudojimas

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

- *Gemini 3*; teksto ir kalbos redagavimui, formuluočių tikslinimui ir kalbos klaidų taisymui;
- *Claude Sonnet 4.5*; kuriamo įrankio naudotojo sąsajos (UI) sprendinių generavimui;
- *GPT 5.2*; idėjų generavimui, susijusiam su sukurtų komponentų testavimo scenarijais ir ribiniais atvejais.

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

Darbą sudaro keturi skyriai. Pirmajame skyriuje pateikiama probleminės srities analizė. Antrajame skyriuje aprašoma kuriamo įrankio specifikacija ir formalus aprašas. Trečiajame skyriuje pristatoma sprendimo realizacija. Ketvirtajame skyriuje pateikiamas eksperimentinis tyrimas, jo rezultatai ir sprendimo tobulinimo kryptys.

1. Probleminės srities analizė

1.1. Analizės tikslas

Analizės tikslas – išsiaiškinti ir suprasti, reikalavimų inžinerijos procesus, jos etapus, dažniausiai naudojamas metodikas, *EventStorming* metodą, *Domain-Driven Design* sistemų kūrimo metodą bei išanalizuoti reikalavimų inžinerijos automatizavimo galimybes, suprasti jau egzistuojančius automatizavimo įrankius tam, jog būtų identifikuotos tokių įrankių spragos, kurios būtų apžvelgtos, įvertintos ir galimai išspręstos šio darbo rezultate.

1.2. Tyrimo objektas, sritis ir problema

Neretai programinės įrangos kūrimo įmonės, prieš sudarant galutinį komercinį pasiūlymą ir pasirašant programinės įrangos kūrimo sutartį vykdo reikalavimų inžinerijos procesus ir sudaro reikalavimų sąrašą suderintą su klientu [1]. Reikalavimų išgavimas gali būti vykdomas įvairias metodais [2], vienas jų – *EventStorming* seminarais pagrįstas metodas, kurio metu išsiaiškinama dalykinės srities informacija, užsakovo norai ir sukuriama artefaktai apibūdinantys planuojamą kurti sistemą arba kitaip sakant išsiaiškinami reikalavimai informacinei sistemai [3]. Vėliau remiantis šiais artefaktais yra sudaromas reikalavimų sąrašas, naudojamas komerciniam pasiūlymui bei programinės įrangos kūrimo sutarčiai sudaryti. Įmonei yra kritiškai svarbu atlikti reikalavimų sudarymo etapus siekiant išsiaiškinti programinės įrangos kūrimo kaštus siekiant pateikti įmonei naudingą komercinį pasiūlymą [1], deja, šis procesas yra sudėtingas, reikalauja laiko ir papildomų išlaidų – už kuriuos užsakovas dažniausiai apmokėti nenori. Todėl kyla poreikis palengvinti šį procesą, o konkrečiai šio darbo kontekste – automatizuoti reikalavimų išgavimą iš *EventStorming* artefaktų.

Apibendrinant, tyrimo problema – programinės įrangos kūrimo komanda siekdama įvertinti programinės įrangos kūrimo kaštus ir pateikti klientui komercinį programinės įrangos kūrimo pasiūlymą, vykdo sudėtingą ir laiko reikalaujantį reikalavimų sudarymo procesą, už kurį klientas neapmoka.

Šio darbo tyrimo objektas – reikalavimų inžinerijos procesų automatizavimas, o **tyrimo sritis** – reikalavimų inžinerijoje taikomi metodai ir jų sudarymo procesas.

1.3. Reikalavimų inžinerijos procesų analizė

1.3.1. Reikalavimų inžinerija

Reikalavimų inžinerija yra sistemingas reikalavimų išgavimo, dokumentavimo, analizės ir valdymo procesas. Tai yra kritiškai svarbus etapas programinės įrangos ar kitokių projektų kūrime, siekiant, jog galutinis rezultatas atitiktų suinteresuotų asmenų poreikius [5]. Sistemingai vykdant reikalavimų inžinerijos procesus organizacijos gali užtikrinti, kad jų sukurtas sprendimas atitinka naudotojų poreikius. Vienareikšmiškai apibrėžti reikalavimai sumažina nesusikalbėjimo, interpretavimo rizikas ir sumažina tikimybę, jog vėlesniuose produkto kūrimo etapuose reikės atlikti brangiai kainuojančius produkto pakeitimus [1]. Verta paminėti, jog reikalavimų inžinerija kaip gali pasirodyti yra orientuota ne tik į programinės

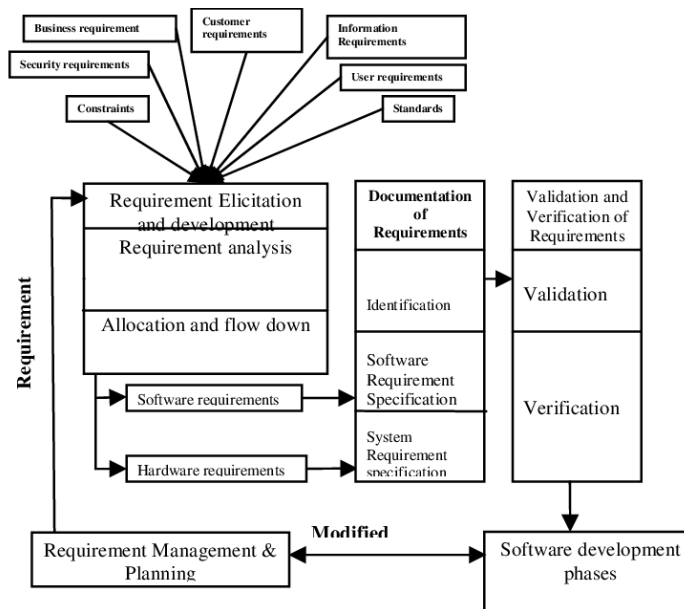
įrangos kūrimą, tai apima ir platesnių sistemų įtraukiant techninės įrangos, personalo, patalpų, duomenų ir kitus galutiniam produktui įtakos turinčius aspektus [6].

Pasisekęs projektas yra tiesiogiai susietas su reikalavimų inžinerijos proceso efektyvumu [5]. Vienareikšmiški, aiškūs reikalavimai yra veiksmų planas projektą vykdančiai komandai. Valdomi reikalavimai palengvina komunikaciją tarp suinteresuotų asmenų bei programuotojų, testuotojų ir kitų projektą vykdančių žmonių, užtikrinant bendrą projekto tikslų supratimą. Kai užsakovo reikalavimai yra išgauti, išanalizuoti ir specifikuoti tikimybė, jog bus sukurtas projektas atitinkantis naudotojo poreikius, išauga – o būtent tai prisideda prie projekto sėkmės [7].

Galiausiai, viena iš kertinių reikalavimų inžinerijos teikiamų naudų yra gebėjimas prisitaikyti prie kintančių poreikių. Natūralu, jog programinės įrangos kūrimas, suinteresuotų asmenų poreikiai, technologijos ir rinkos poreikiai nuolat kinta, todėl svarbu kurti projektą, kuris būtų aktualus jo paleidimo metu [8]. Reikalavimų inžinerijos procesas yra orientuotas į tokius pokyčius ir suteikia galimybę suvaldyti kintančius reikalavimus, užtikrinant, jog pokyčiai yra aiškiai dokumentuoti, išanalizuoti ir įgyvendinti tinkamai [9].

Reikalavimų inžinerija yra skirstoma į šiuos etapus, matomus 1.1 pav.:

1. Reikalavimų išgavimas (angl. *elicitation*)
2. Reikalavimų analizė
3. Reikalavimų specifikavimas
4. Reikalavimų atestavimas
5. Reikalavimų valdymas



1.1 pav. Reikalavimų inžinerijos proceso modelis [6]

Reikalavimų išgavimo etapas inicijuoja reikalavimų inžinerijos procesus siekiant sukurti sėkmingą produktą ar sistemą. Šis etapas orientuotas į svarbios informacijos išgavimą iš suinteresuotų asmenų tokių kaip, naudotojų, dalykinės srities ekspertų ir kitų asmenų, arba kitaip sakant žmonių, kurie turi įtakos kuriamam produktui arba dalyvauja veiklos procesuose, kuriuos siekiama nauju produktu keisti ar automatizuoti. Siekiama išsiaiškinti

minėtųjų asmenų poreikius, lūkesčius produktui ir apribojimus. Reikalavimų išgavimo technikų yra įvairių, pradedant nuo interviu, baigiant darbo stebėjimu, toliau bus apžvelgtos rekomenduojamos reikalavimų išgavimo technikos [2] [10] [11] [12].

Interviu yra viena dažniausiai naudojamų technikų reikalavimų išgavimui. Dažnu atveju tai būna pokalbis tarp veiklos analitiko ir suinteresuoto asmens siekiant išgauti informaciją apie šio žmogaus poreikius iš sistemos. Interviu gali būti struktūrizuoti ir nestruktūrizuoti. Struktūrizuotas interviu susideda iš anksto pasiruoštų klausimų, kuriuos atsako suinteresuotas žmogus, o nestruktūrizuotas iš natūralaus pokalbio. Toks būdas yra puikus siekiant išgauti detalizuotą, specifinę informaciją apie dalykinę sritį ar automatizuojamą procesą.

Apklausa taip pat yra viena populiariesnių reikalavimų išgavimo technikų. Tai gali būti atliekama gyvai, telefonu ar naudojant apklausos sistemas. Šis pasirinkimas gali būti naudojamas tada kai informaciją reikia surinkti iš daugelio žmonių.

Prototipavimas taip pat yra efektyvus būdas gauti grįžtamąjį ryšį iš suinteresuotų žmonių. Nors tai ir gana ilgas procesas, kadangi prototipo sukūrimas užtrunka laiko, tačiau tokiu būdu galima gauti konstruktyvų grįžtamąjį ryšį ir taip išgryninti keliamus reikalavimus kuriamai sistemai.

Taip pat populiari technika reikalavimams išgauti yra **darbo stebėjimas**. Ši technika pasižymi procesu kai veiklos analitikas stebi suinteresuotus asmenis jų darbo aplinkoje ir dokumentuoja savo pastebėjimus, kuriuos vėliau paverčia sistemos reikalavimais.

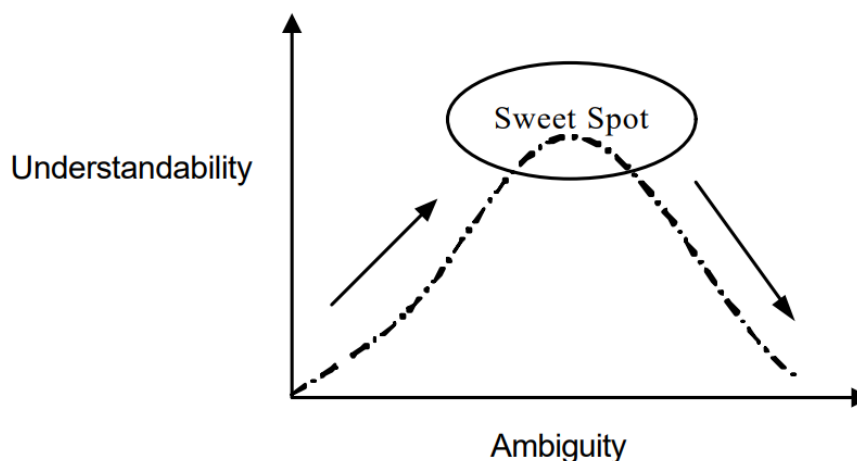
Šių technikų kombinavimas užtikrina įvairiapusį dalykinės srities, susiduriamų problemų, o galiausiai ir reikalavimų supratimą. Suinteresuotų asmenų bendradarbiavimas yra kritiškai svarbus reikalavimų išgavimo etape, kadangi šio etapo rezultatas yra visų sekančių reikalavimų inžinerijos etapų pagrindas.

Praktikoje dažnai susiduriama su įvairiais reikalavimų išgavimo iššūkiais. Visų pirma dažnai susiduriama su skirtingomis dalykinės srities ekspertų, darbuotojų, kitų suinteresuotų žmonių nuomonėmis, lūkesčiais, todėl tampa sunku identifikuoti ir prioretizuoti reikalavimus. Be to, skirtingi žmonės turi skirtingą procesų supratimą, nes dalyvauja tik dalyje jų, kas gali dar labiau apsunkinti reikalavimų išgavimo procesą. Galiausiai žmonės gali nesugebėti aiškiai išsakyti savo minčių, lūkesčių, todėl reikalavimai gali būti abstraktūs ir netikslūs [13].

Taip pat svarbu nepamiršti, jog reikalavimai dažnai yra kintantys, todėl verta reikalavimų išgavimą vykdyti iteraciniu būdu [14], suteikti galimybę suinteresuotiems asmenims palikti grįžtamąjį ryšį ir sudaryti galimybę tiesiogiai kontaktuoti su projektuotoju ar veiklos analitiku ne tik įvairių susitikimų metu.

Reikalavimų dokumentavimas taip pat yra neatsiejama reikalavimų išgavimo etapo dalis. Kai aktuali informacija yra išgaunama iš suinteresuotų asmenų, ji paverčiama į struktūrizuotą, formalų formatą. Šis dokumentas tampa reikalavimų saugykla, kuria yra remiamasi vėlesniuose reikalavimų inžinerijos etapuose. Verta paminėti, jog ši dokumentacija dažnai yra nevisiškai tiksli ir pateikiama kaip tekstinis dokumentas, panaudojimo atvejų modelis ar kiti nestandartinės notacijos modeliai padedantys suprasti

sistemos reikalavimus. Dokumentacija ne tik aprašo ką planuojama kurti, bet ir detalizuoja kaip tai turi būti atliekama. Dokumentavimo procesas turi išlaikyti balansą tarp detalios ir ne per daug abstrakčios informacijos susisteminiamą [14] (1.2 pav.). Efektyvus dokumentavimas yra kritiškai svarbus siekiant gerinti komunikaciją tarp suinteresuotų asmenų ir sistemą kurti planuojančios komandos. Šis dokumentas tampa pradine specifikacija, naudojama projekto apimčiai, biudžetui ir įgyvendinimo terminams įvertinti, taip pat kaip pagrindas sistemos kūrimo sutarčiai sudaryti [1].



1.2 pav. Suprantamumo kitimo grafikas pagal reikalavimo detalumo lygį [14]

Apibendrinant, reikalavimų išgavimas yra ypatingai svarbus etapas reikalavimų inžinerijoje reikalaujantis efektyvios komunikacijos ir suinteresuotų asmenų įsitraukimo. Svarbu gautą informaciją susisteminti taip, kad iš reikalavimų dokumento būtų galima spręsti apie projekto apimtį, biudžetą ir terminus. Šio etapo rezultatai tiesiogiai nulemia tolesnių reikalavimų inžinerijos etapų sėkmę [7].

Reikalavimų analizės etapas yra skirtas įvertinti surinktus reikalavimus reikalavimų dokumente užtikrinant, jog reikalavimai yra vienareikšmiški, pilni ir įgyvendinami [7]. Projektuotojas identifikuoja galimus iššūkius, sunkumus prieš pradėdant sistemos kūrimą ir prioretizuoja reikalavimus pagal jų poreikį ir įtaką projekto sėkmei. Šio etapo esmė yra sumažinti nesusipratimus ir tuo pačiu riziką, jog tam tikrą sistemos funkcionalumą reikės perdaryti (angl. *refactor*) ateityje. Projektuotojas analizuodamas pirminį dokumentą identifikuoja dviprasmiškus, konfliktuojančius tarpusavyje reikalavimus. Tai yra iteracinis procesas reikalaujantis grįžti į reikalavimų išgavimo etapą siekiant patikslinti ir suderinti reikalavimus tarpusavyje ar su suinteresuotais asmenimis. Išanalizuotas reikalavimų sąrašas palengvina sudaryti galutinę specifikaciją [15]. Verta paminėti, jog analizės etape projektuotojas dažnai komunikuoja su suinteresuotais asmenimis siekiant patvirtinti surinktus reikalavimus.

Reikalavimų specifikacijos etapas susideda iš detalios specifikacijos kūrimo, kuri yra naudojama kaip planas projektą vykdančiai programuotojų komandai. Specifikacijos etapas orientuotas į funkcinių, nefuncinių reikalavimų, apribojimų apibrėžimą. Specifikacijos gali būti įvairių formų, panaudojimo atvejų diagramos, sistemų modeliai, prototipai, tekstiniai dokumentai [15]. Specifikacijos tikslas yra aiškiai apibrėžti ką galutinis produktas turi atlikti ir kaip tą turi padaryti tam, kad sistemą kurianti komanda galėtų sėkmingai realizuoti

sistemą. Siekiant, jog specifikacija būtų aiški ir išbaigta neretai naudojami įvairūs specifikavimo šablonai, tokie kaip *Volere* [16], *IEEE Standard 830* [17], *ISO/IEC/IEEE 29148* [18]. Visų šių šablonų tikslas suteikti galimybę specifikaciją aprašyti vienareikšmiškai ir aiškiai taip, kad būtų suprantama ne tik projektą realizuojančiai komandai bet ir derinimui su suinteresuotais asmenimis.

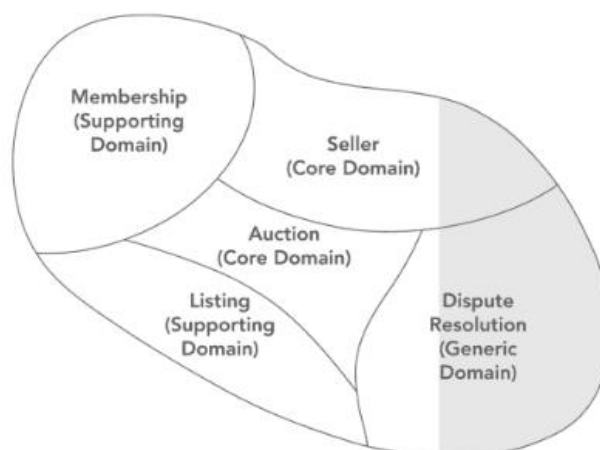
Reikalavimų validavimo etapas yra paskutinis reikalavimų inžinerijos etapas, po kurio įprastai pereinama prie sistemos realizacijos [19]. Šio etapo metu vykdoma reikalavimų specifikacijos peržiūra ir atestavimas, siekiant užtikrinti, kad ji atitinka suinteresuotų asmenų lūkesčius bei nustatytus kokybės kriterijus. Validavimas yra proaktyvus procesas, kurio tikslas – identifikuoti dviprasmybes, neatitikimus ir galimus nesusipratimus specifikacijoje dar prieš jų realizavimą galutiniame produkte [19]. Šiame etape taikomos įvairios technikos, tokios kaip kolegų peržiūros (angl. *peer reviews*), specifikacijos pristatymai suinteresuotiems asmenims, prototipavimas ir formalios patikros metodai. Ypatingai svarbus yra suinteresuotų asmenų įsitraukimas, nes tai leidžia užtikrinti, kad jų poreikiai ir lūkesčiai yra tinkamai suprasti, užfiksuoti ir aiškiai apibrėžti reikalavimų specifikacijoje. Sėkmingai įgyvendintas validavimo etapas prisideda prie sklandesnės sistemos kūrimo eigos, nes sumažina klaidų tikimybę vėlesniuose etapuose ir užtikrina, kad reikalavimai yra nuoseklūs, vienareikšmiai ir išbaigti [20].

1.3.2. Sistemų kūrimas remiantis *Domain-Driven Design*

Domain-Driven Design arba trumpiau *DDD*, yra sistemų kūrimo būdas, orientuotas į veiklos sričių ir su jais susijusių procesų išskyrimą. *DDD* kūrimo būdas siekia, kad sistemą realizuojantis kodas ir jo architektūra būtų orientuota į veiklos sritį, kurios problemą norima išspręsti, taip pat skatinti bendradarbiavimą tarp dalykinės srities ekspertų ir programuotojų bei siekia užtikrinti, jog visi kalbėtų vienareikšmiškai, vieninga veiklos kalba apibrėžtoje veiklos srityje. *DDD* taip pat siekia sumažinti kompleksiskumą naudodamas objektinio programavimo bei šablonų taikymą, be to siekiama, jog programinės įrangos kodas būtų aiškus, ir geriausiai dokumentuotų kokią problemą sprendžia [21]. *DDD* sistemų kūrimo būdas susideda iš šių elementų:

Dalykinė sritis pagal *DDD* yra loginė veiklos sritis apibrėžianti problemą, kurią bandoma spręsti [22], pavyzdžiui internetinis aukcionas arba mokymosi platforma galėtų būti dalykinės sritys. Tačiau *DDD* pabrėžia, jog per dažnai dalykinės sritys yra per daug abstrakčios ir jas reikėtų skaidyti į smulkesnes dalykines sritis, kurios apibrėžtų konkrečias tėvinės dalykinės srities dalis, be to vertėtų nusistatyti kurios smulkesnės dalykinės sritys yra svarbiausios siekiant pagrindinės dalykinės srities sėkmės. Toks būdas leidžia programinės įrangos kūrėjams lengviau atrasti svarbiausias, kritines sistemos dalis. Nagrinėjant internetinio aukciono pavyzdį 1.3 pav. galime matyti, jog internetinio aukciono dalykinė sritis yra suskaidyta į narystės, pardavėjo, kotiravimo, skundų sprendimo ir

aukciono smulkesnes dalykinės sritis, be to aukciono ir pardavėjo dalykinės sritys yra kartinės, o kitos – tik palaikomąją funkciją atliekančios smulkesnės dalykinės sritys [23].



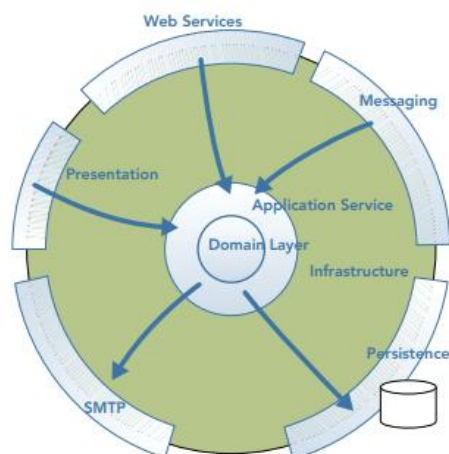
1.3 pav. Internetinio aukciono dalykinės srities smulkesnių dalykinių sričių apipavidalinimas [23]

Apibrėžtas kontekstas (angl. *bounded context*) – tai loginė sritis programiniame kode, kurioje naudojami tie patys konceptai, duomenys ar taisyklės. Apibrėžti kontekstai dar labiau skaido dalykinę arba smulkesnę dalykinę sritį. Siekiant geriau suprasti skirtumą tarp dalykinės srities ir apibrėžto konteksto, *DDD* akcentuoja, jog dalykinė sritis apibrėžia problemą, kuri egzistuoja veikloje, o apibrėžtas kontekstas – sprendimą, kuris egzistuoja programinės įrangos kode [21].

Vienareikšmiška kalba – tai vienas svarbiausių *DDD* elementų, kurio idėja yra, jog sistemą kuriantys programuotojai, architektai bei sistemos naudotojai, veiklos srities ekspertai kalbėtų vienoda veiklos kalba. Remiantis ja turėtų būti parenkami klasių, modulių, kintamųjų pavadinimai. Tokiu būdu programinės įrangos kodas tampa lengviau skaitomas ir lengviau paaiškinamas įprastam naudotojui, kyla mažiau nesusipratimų derinant reikalavimus. *DDD* sako, jog klientas visose programinės įrangos kodo vietose ir turėtų būti klientas, o ne naudotojas, pirkėjas ar kt. [21].

Sluoksniuota architektūra (angl. *onion architecture*) – *DDD* rekomenduoja naudoti sluoksniuotą architektūrą, tačiau neapibrėžia kaip tiksliai ją reikėtų įgyvendinti. *DDD* pateikia rekomendaciją, jog dalykinė sritis ir ją sprendžiantis sprendimas (apibrėžtas kontekstas) neturėtų būti išmėtytas tarp skirtingų programinės įrangos komponentų tokių kaip duomenų bazė, naudotojo sąsaja ar kt., o turėtų būti vienoje vietoje. Nenaudojant sluoksniuotos architektūros, programinės įrangos kodas pasidarytų sunkiau skaitomas, ir nebūtų įmanoma įvertinti atskirties tarp kelių apibrėžtų kontekstų [21]. Praktikoje *DDD* sluoksniuota architektūra įgyvendinama *Onion* dar kitaip žinoma kaip *Hexagonal* ar *Clean architecture* principais. Ši architektūra, kurios reprezentacija matoma 1.4 pav., siekia atskirti su atvaizdavimu, infrastruktūra, duomenų saugojimu susijusią realizaciją ir dalykinės srities programinės įrangos realizaciją. Tokiu būdu sprendimas dalykinės srities problemai

egzistuoja vienoje vietoje (dalykinės srities (angl. *domain*) sluoksnyje) ir nėra išmėtytas tarp skirtingų programinės įrangos komponentų.



1.4 pav. *Onion* architektūros reprezentacija [23]

Esybė (angl. *entity*) – *DDD* sako, jog esybės yra tokie dalykinės srities objektai, kurie yra unikaliam identifikuojami identifikatoriumi, o ne atskirais atributais. Šios esybės yra dalykinės srities pagrindas apibūdinantis pagrindinius veiklos objektus. Esių pavyzdžiai: naudotojas, žinutė, užduotis. Esiybės apibrėžia kintančius objektus [21].

Vertės objektai (angl. *value-objects*) – tai tokie *DDD* objektai, kurie turi įvairių atributų bet nėra unikaliam identifikuojami specialiu identifikatoriumi. Vertės objektų pavyzdys – spalva susidedanti iš raudonos, mėlynos ir žalios dedamųjų bet programinės įrangos kode nėra unikaliam identifikuojama identifikatoriumi. Vertės objektai apibrėžia nekintančius objektus [21].

Agregatai (angl. *aggregates*) – tai esybių ir vertės objektų sandara. Dažnu atveju siekiant modeliuoti kompleksią dalykinę sritį reikalingi vertės objektų ir esybių mišiniai, tokie dariniai yra vadinami agregatais. Agregato pavyzdys galėtų būti automobilio esybė, kuri turi nuorodą į spalvą (t.y. vertės objektą), bet tuo pačiu turi ir pavadinimą, modelį, unikalų numerį [22].

Dalykinės srities įvykiai (angl. *domain events*) – tai tokie įvykiai, kurie yra reikšmingi dalykinės srities ekspertui ir apibūdina svarbų veiklos faktą ar būsenos pasikeitimą apibrėžtame kontekste. Tokio įvykio pavyzdys galėtų būti užregistruotas naujas darbuotojas darbuotojų valdymo dalykinėje srityje. Toks įvykis yra svarbus ne tik sistemai, bet ir pačiai veiklos sričiai, nes jis gali lemti tolesnius veiklos procesus ar taisyklių taikymą. Tuo tarpu failo skaitymo pabaigos įvykis dažniausiai nėra laikomas dalykinės srities įvykiu darbuotojų valdymo dalykinėje srityje, nes jis apibūdina techninį sistemos veikimo faktą, o ne veiklai reikšmingą pokytį. Kitaip tariant, tai yra infrastruktūrinis, o ne dalykinės srities įvykis [21].

Paslaugos (angl. *services*) – tai objektai, kurie atlieka tam tikrą paslaugą transformuojant, skaičiuojant ar kitaip apdorojant agregatus. Darbuotojų valdymo dalykinėje srityje,

darbuotojo darbo užmokesčio skaičiavimo operacija galėtų būti darbuotojo darbo užmokesčio paslaugos dalis [21].

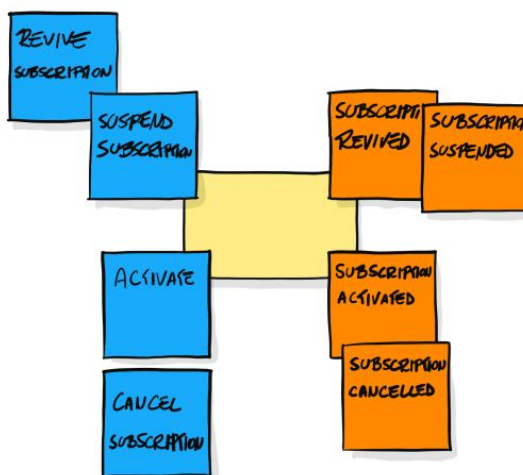
Domain-Driven Design sistemų kūrimo būdas orientuojasi į dalykinės srities elementų charakterizavimą, jų naudojimą ir aprašymą programinės įrangos kode. Tačiau nenurodo kaip šie elementai, tokie kaip dalykinė sritis, esybė, ar dalykinės srities įvykis turėtų būti išsiaiškintas, išgautas iš užsakovų. Kadangi programuotojai dažniausiai nėra dalykinės srities ekspertai ir negali aiškiai patys jų savarankiškai apibrėžti, todėl egzistuoja *Domain-Driven Design* modeliavimo technikos, tokios kaip *EventStorming*, *Domain Storytelling* ir *Context Mapping*, kuriomis siekiama įvertinti ir apibrėžti *Domain-Driven Design* elementus.

1.3.3. Reikalavimų išgavimas remiantis *EventStorming*

EventStorming – tai reikalavimų išgavimo ir taip pat dalykinės srities išsiaiškinimo technika [3]. *EventStorming* technika remiasi idėja, jog kompleksiška veikla ir jos procesai gali būti modeliuojami remiantis veiklos įvykiais. Įvykiai *EventStorming* kontekste reprezentuoja kertinius reiškinius ar būsenos pasikeitimus veikloje, pavyzdžiui naudotojas pateikia užsakymą, arba apmokėjimas yra vykdomas [3].

EventStorming technika remiasi pagrindiniais principais:

1. **Bendradarbiavimas.** *EventStorming* pabrėžia bendradarbiavimo svarbą įvairiapusėje komandoje siekiant išsiaiškinti dalykinę sritį. Ši komanda susideda iš dalykinės srities ekspertų, programuotojų ir kitų suinteresuotų asmenų, kurie bendradarbiauja siekdami aprašyti dalykinę sritį panaudojant *EventStorming* artefaktus kūrybinių dirbtuvių metu [24].
2. **Vizualus dalykinės srities aprašymas.** *EventStorming* taip pat remiasi vizualiu dalykinės srities apipavidalinimu panaudojant lipnius lapelius (angl. *sticky notes*). Skirtingos lapelių spalvos apibūdina skirtingus elementus, tokius kaip įvykiai, komandos, agregatai, taisyklės. 1.5 pav. yra pavaizduoti keturi knygos prenumeravimo dalykinės srities įvykiai bei keturi komandos, kurių reikšmė bus aptarta detaliau vėlesnėje šio skyrelio dalyje. Dalis minėtų elementų tiesiogiai apibrėžia *Domain-Driven Design* elementus [3].

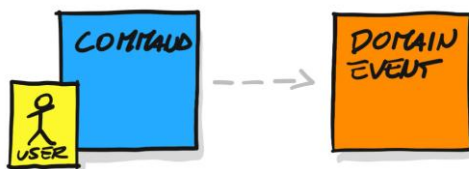


1.5 pav. Knygos prenumeravimo dalykinės srities *EventStorming* metodo artefaktai [3]

3. **Eiliškumas laike** (angl. *temporal order*). Veiklos įvykiai yra išrikiuoti eilės tvarką, pagal tai kokių laiko momentu jie įprastai įvyksta, šis išrikiavimas atspindi veiklos procesų eigą. Toks chronologinis išrikiavimas padeda suinteresuotiems asmenims suprasti priežasties ir pasekmės priklausomybes tarp veiklos procesų ir įvykių [3].
4. **Vienareikšmė kalba**. *EventStorming* kūrybinių dirbtuvių metu skatinama naudoti vieningą, visiems suprantamą, vienareikšmę kalbą tam, kad kiekvienas dirbtuvėse dalyvaujantis asmuo vienodai suprastų dalykinę sritį. Tokiu būdu sudaromas bendras dalykinės srities žodynas vėliau naudojamas sistemos kūrimo *Domain-Driven Design* būdu [24].

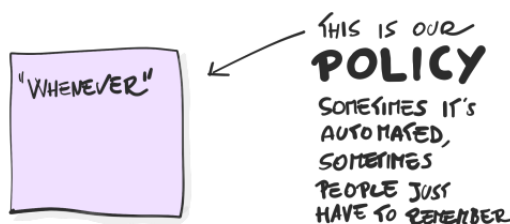
EventStorming metodas išskiria tokius artefaktus, įprastai žymimus skirtingų spalvų lipniais lapeliais [3]:

- **Dalykinės srities įvykiai** (angl. *domain events*) – žymimi oranžinės spalvos lapeliais reprezentuoja kertinius reiškinius ar būsenos pasikeitimus dalykinėje srityje. Šie įvykiai yra sudėliojami eiliškumo tvarka laiko atžvilgiu.
- **Komandos** (angl. *commands*) – žymimos mėlynos spalvos lapeliais ir reprezentuoja veiksmus ar užklausas kurie sukelia dalykinės srities įvykius. Šios priklausomybė notacija matoma 1.6 pav.



1.6 pav. *EventStorming* komandos sąryšio su dalykinės srities įvykiu notacija [3]

- **Taisyklės** (angl. *policies*) – žymimos violetinės spalvos lapeliais reprezentuoja veiklos taisykles ar apribojimus, kurie yra taikomi įvykiams ar komandoms. Taisyklės lemia įvykių ir komandų elgseną. Taisyklės lapelis matomas 1.7 pav.



1.7 pav. *EventStorming* taisyklės reprezentacija

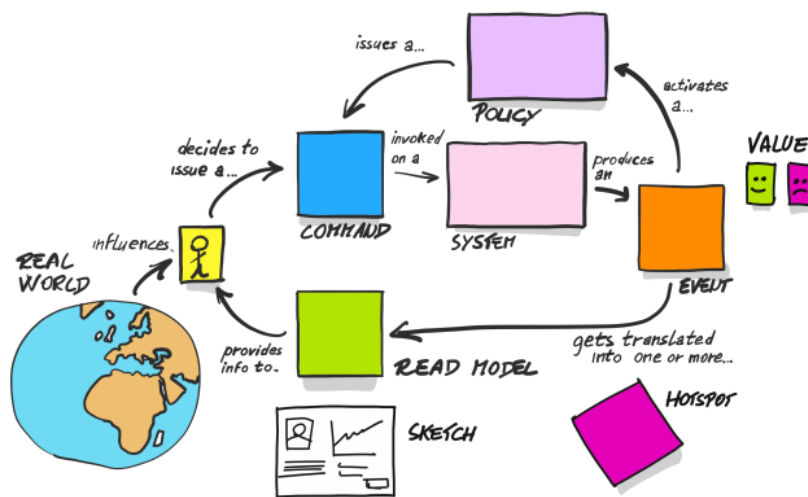
- **Išorinės sistemos** (angl. *external systems*) – žymimos rožinės spalvos lapeliais ir vaizduoja išorės sistemas bei sąveiką su jomis. Šis elementas yra naudojamas identifikuoti ir suprasti modeliuojamos dalykinės srities ribas.
- **Aktoriai** (angl. *actors*) – žymimi geltonos spalvos lapeliais vaizduoja veiklos naudotojus, kurie iškviečia tam tikras komandas. Siekiama identifikuoti skirtingų naudotojų tipus, kurie padeda geriau suprasti naudotojų norus.

- **Skaitymo modeliai** (angl. *read models*) – žymimi žalios spalvos lapeliais, vaizduoja duomenis, kuriuos aktorius gauna kaip dalykinės srities veiklos proceso išeią. Siekiama apibrėžti duomenis, kuriuos veikla sugeneruoja.
- **Agregatai** (angl. *aggregates*) – žymimi geltonais lipniais lapeliais, kurie apibrėžia nuoseklumo ribas (angl. *consistency boundaries*) ir apjungia susijusius įvykius, komandas bei taisykles į vieną logiškai vientisą dalykinės srities vienetą. Jie identifikuojami vėlesniuose *EventStorming* etapuose, kai analizuojama, kurie elementai turi būti valdomi kartu. *EventStorming* agregatai atitinka *Domain-Driven Design* konteksto agregatus.
- **Klausimai, komentarai, neaiškumai** (angl. *hotspots*) – žymimi pilkos spalvos lapeliai ir yra skirti pavaizduoti papildomą informaciją, patikslinimus ar *EventStorming* kūrybinių dirbtuvių metu užduotus dalyvių klausimus.

Šių elementų pavyzdžiai, yra matomi 1.8 pav., kurie reprezentuoja picerijos dalykinės srities modeliavimo fragmentą naudojant *EventStorming* metodą, o elementų tarpusavio sąryšiai pavaizduoti 1.9 pav. Analizuojant 1.8 pav. paveikslėlio *EventStorming* artefaktus ir jų tarpusavio sąryšius galime matyti, jog geltonos spalvos lapeliu išskirtas klientas, kuriam yra pateiktas skaitymo modelis pažymėtas žaliu lapeliu, kuris aprašo meniu ir picerijos atidarymo valandas. Klientas vėliau vykdo mėlyna spalva pažymėtą komandą – užsakyti picas. Matoma, jog komandai vykdyti naudojama išorinė sistema pažymėta rožinės spalvos lapeliu pavadinimu *iwantmypizza.com*. Šios komandos rezultatas – dalykinės srities įvykis, pažymėtas oranžine spalva – užsakymas pateiktas. Šis įvykis iššaukia sekancią ruošti picą komandą, tačiau ji yra apribota violetine spalva pažymėtu taisyklių rinkiniu, kuris apibrėžia picos gaminimo taisykles.



1.8 pav. Picos užsakymo fragmentas naudojant *EventStorming* metodą [3]



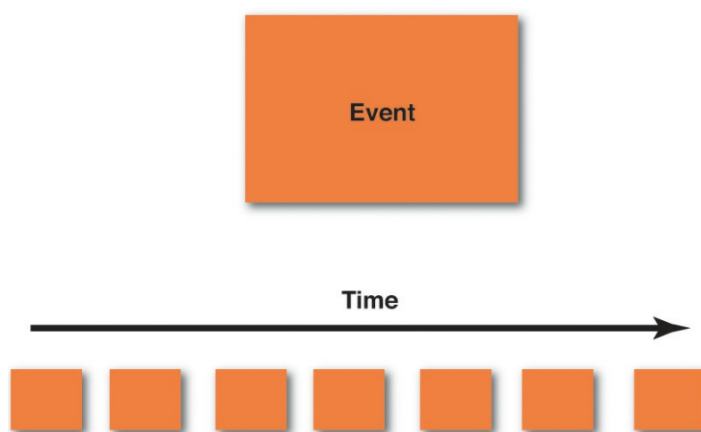
1.9 pav. Proceso modeliavimas naudojant *EventStorming* artefaktus [3]

Naudojant elementus nurodytus 1.8 pav. ir naudojant jų tarpusavio sąryšius pavaizduotus 1.9 pav. yra vykdomos *EventStorming* kūrybinės dirbtuvės, kurių procesas susideda iš šių dalių [3] [24]:

Visų pirma pradedama nuo pristatymo ir konteksto apibrėžimo. *EventStorming* kūrybinių dirbtuvių vedėjas supažindina dalyvius su dirbtuvių formatu, pristato sesijos tikslus ir bendradarbiaujant su dalykinės srities ekspertais apibrėžia dalykinės srities ar sprendžiamos problemos kontekstą.

Antra yra vykdomas *Big Picture EventStorming*. Dalyviai pradeda nuo bendros dalykinės srities apžvalgos orientuodamiesi į pagrindinius kompiuterizuojamos veiklos vykdomus įvykius. Įvykiai yra dokumentuojami naudojant įvykiams skirtus oranžinės spalvos lapelius. Šie lapeliai yra klijuojami ant lentos. Taip sukurama vizuali dalykinės srities įvykių reprezentacija.

Trečia yra kuriama laiko juosta, kurioje dalykinės srities įvykiai yra sudėliojami eilės tvarka pagal tai koku eiliškumu jie įvyksta. Šis chronologinis išdėstymas, matomas 1.10 pav. padeda dalyviams geriau suprasti kompiuterizuojamos veiklos elgseną.



1.10 pav. Dalykinės srities įvykių išdėstymas chronologine tvarka [3]

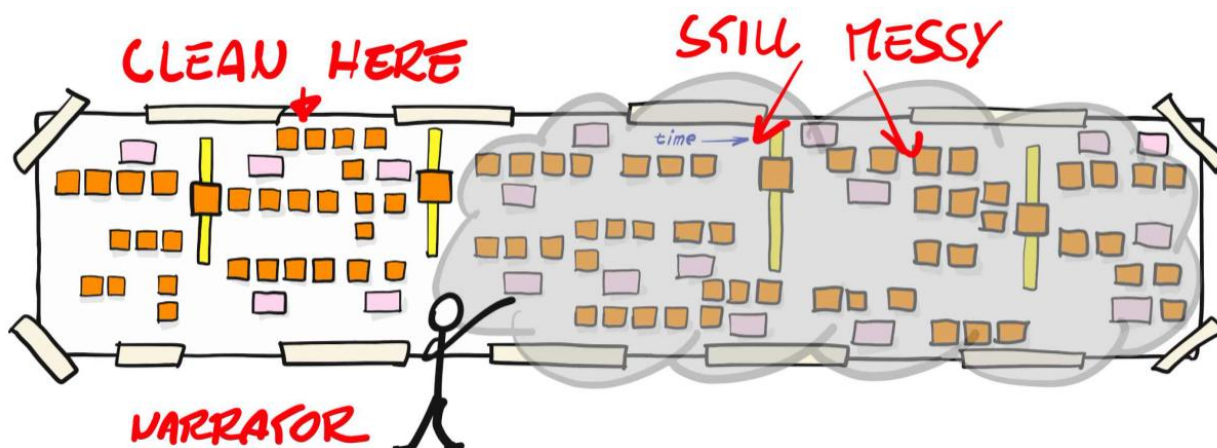
Toliau vyksta komandų atradimo etapas. Šio etapo metu dalyviai naudodami mėlynos spalvos lapelius identifikuoja veiksmus arba užklausas, kurios iššaukia tam tikrus veiklos įvykius. Dalyviai bendradarbiaudami identifikuoja ir pažymi komandas laiko juostoje.

Taisyklių atradimo etape stengiamasi surasti taisyklių ar apribojimų, kurie taikomi įvykiams ar komandoms, dažnu atveju šis etapas yra integruojamas į komandų atradimo etapą, kadangi kyla natūrali diskusija apie apribojimus tam tikrai veiklai.

Procesą seka išorinių sistemų ir aktorių integravimo etapas, kurio metu yra identifikuojama kokiomis išorinėmis sistemomis naudojamos bei kurias komandas atlieka naudotojai, taip apibrėžiant dalykinės srities ribas.

Įvykdžius šiuos etapus pirmą kartą, dažnu atveju yra gaunamas tik abstraktus dalykinės srities modeliavimo rezultatas, kartais nevienareikšmiškai apibrėžtas. Todėl

rekomenduojama, dali lentos ant kurios buvo modeliuota dalykinė sritis atskirti ir vėl pradėti etapus iš naujo tobulinant modelį, kaip matoma 1.11 pav.

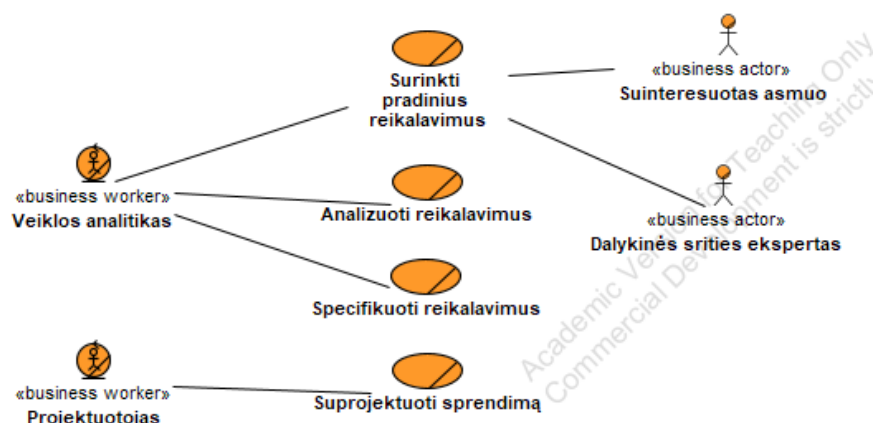


1.11 pav. *EventStorming* iteratyvus procesas tobulinant dalykinės srities modelį [3]

EventStorming apibrėžia, jog šie etapai turėtų vykti iteratyviai, t.y. po pirmojo visų etapų įgyvendinimo galima vėl pradėti nuo pradžių ir nagrinėti siauresnę dalykinės srities dalį ir taip ją detalizuoti. Šis procesas vykdomas tol, kol pasiekiamas norimas dalykinės srities detalumas. Gauti artefaktai gali būti naudojami tolimesnei techninei sistemos architektūrai kurti bei reikalavimams specifikuoti [3].

1.4. Reikalavimų inžinerijos procesų naudotojų analizė

Reikalavimų inžinerijos procese vaizduojamą 1.12 pav. dažniausiai dalyvauja veiklos analitikai, atsakingi už pradinių reikalavimų surinkimą, jų analizę ir specifikavimą. Reikalavimų surinkime dažnai dalyvauja ir suinteresuoti asmenys, tai gali būti būsimi sistemos naudotojai, užsakovai, taip pat pradinių reikalavimų surinkime dalyvauja ir dalykinės srities ekspertas, suprantantis veiklos procesus. Galiausiai remiantis specifikuotais sistemos reikalavimais projektuotojas suprojektuoja sprendimą.



1.12 pav. Tyrimo objekto naudotojų veiklos panaudojimo atvejų diagrama

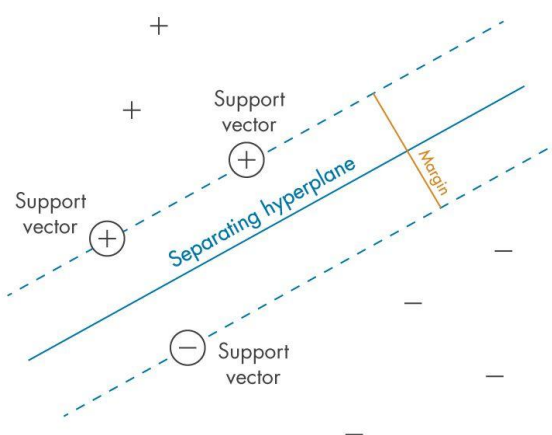
Būtent *EventStorming* dirbtuvių metu minėti žmonės stengiasi atlikti pradinius šių veiklų žingsnius – bendradarbiaudami jie greitai išgrynina dalykinės srities procesus, identifikuoja pagrindinius įvykius, komandas bei taisykles ir taip suformuoja bendrą supratimą apie

dalykinę sritį ir planuojamą kurti sistema dar prieš formalų reikalavimų specifikavimą. Tai leidžia ankstyvoje stadijoje suderinti skirtingų suinteresuotų šalių požiūrius ir sumažinti reikalavimų interpretavimo klaidų riziką.

1.5. Vaizdų atpažinimo galimybės naudojant mašininį mokymąsi

1.5.1. SVM panaudojimas objektams atpažinti

Atraminų vektorių mašinos (angl. *support vector machines*, toliau *SVM*) yra klasifikatorius, kuris gali būti naudojamas objektų atpažinimui vaizduose [25]. Pagrindinė *SVM* idėja yra rasti optimalią duomenų atskirtį – hiperplokštumą, kuri maksimaliai atskiria norimas atpažinti duomenų klases, tuo pačiu didinant atstumą tarp artimiausių skirtingų klasių pavyzdžių, vadinamų atraminiais vektoriais (angl. *support vectors*), matoma 1.13 pav. [26].

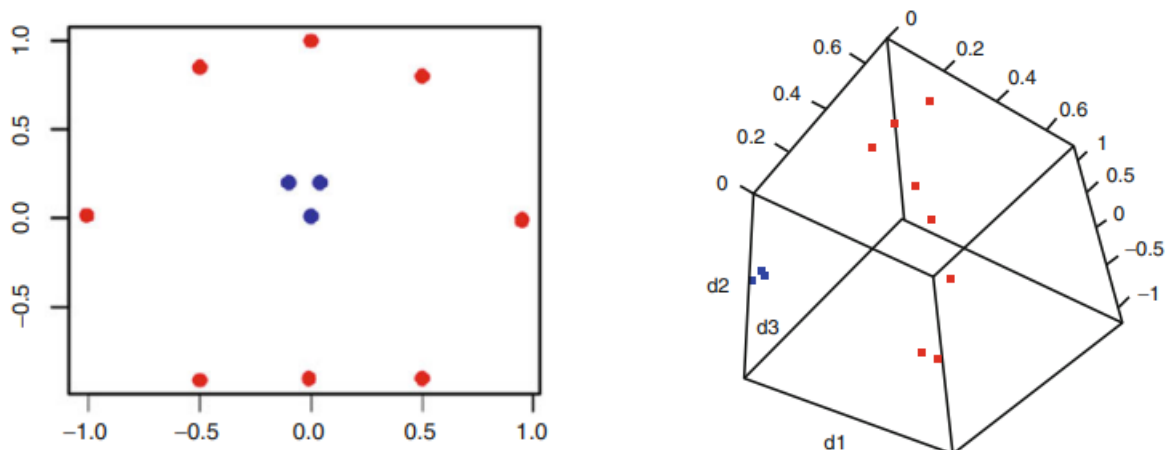


1.13 pav. SVM vizualizacija [27]

SVM klasifikatorius vaizdų atpažinime yra taikomas kartu su iš vaizdo išgautais požymiais. Požymiai gali apimti objektų kraštus, tekstūrą, kampus ar spalvų histogramas. Populiariausi metodai automatiniam požymių išgavimui yra *Histogram of Oriented Gradients*, *Scale-Invariant Feature Transform* ir *Speeded-Up Robust Features*, kurie apibūdina objektus pagal jų geometriją, kontrastą ir struktūrą. Požymių išgavimas yra svarbiausias etapas naudojant *SVM* objektų klasifikavime, nes *SVM* tikslumas tiesiogiai priklauso nuo šių metodų gebėjimo apibūdinti skirtingas objektų klases [28].

SVM mokymo procesas remiasi optimizavimo problema, kurios tikslas yra rasti hiperplokštumą, maksimaliai atskiriančią skirtingas klases. Ši hiperplokštuma nustatoma taip, kad atstumas nuo artimiausių atraminų vektorių būtų kuo didesnis. Tokiu būdu modelis tampa atsparus triukšmui ir nedideliems duomenų variacijoms. Kai duomenys nėra linijiskai atskiriami, pavyzdžiui kaip galime matyti 1.14 pav. kairėje, *SVM* naudoja branduolių metodą (angl. *kernel trick*), kuris leidžia projektuoti duomenis į aukštesnės dimensijos erdvę, kur

klases galima linijiškai atskirti ir galima sudaryti hiperplokštumą, kaip matoma 1.14 pav. dešinėje [26].



1.14 pav. Dviejų klasių duomenys plokštumoje (kairė) bei dviejų klasių duomenys išdėstyti erdvėje (dešinė) [26]

Kai modelis yra apmokytas, nauji vaizdai ar jų regionai yra paverčiami į tą patį požymių erdvės formatą ir vertinami pagal treniruotą *SVM* modelį. Modelis nustato, kuri klasė yra labiausiai tikėtina pagal objekto požymių koordinatę erdvėje, o atraminiai vektoriai lemia sprendimo ribą. Šis principas leidžia *SVM* modeliams efektyviai klasifikuoti objektus net tada, kai skirtumai tarp klasių yra subtilūs, pavyzdžiui, veidų atpažinime ar medicininių vaizdų klasifikacijoje [28].

Praktiniu požiūriu, *SVM* privalumai apima santykinai mažą skaičiavimo poreikį ir gerą veikimą su ribotu duomenų kiekiu, taip pat aiškią sprendimo logiką, nes hiperplokštumos padėtis ir atraminiai vektoriai gali būti interpretuojami analitiškai. Tuo pat metu trūkumai yra susiję su dideliu priklausymu nuo parinktų požymių ir ribotu gebėjimu apdoroti labai sudėtingus ar įvairius objektus. Be to, *SVM* modelio parametrai, tokie kaip branduolio pasirinkimas ir reguliavimo parametrai, turi būti kruopščiai pritaikyti, kad būtų pasiektas optimalus rezultatas [29].

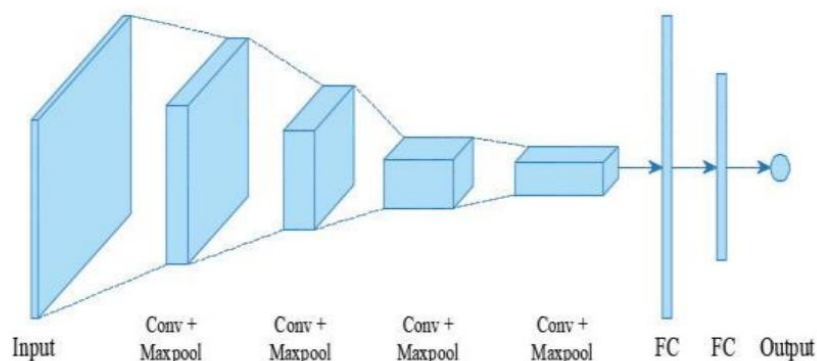
SVM metodas yra ypač tinkamas tais atvejais, kai objektai turi aiškius, atskiriamus požymius ir kai prieinamų duomenų kiekis leidžia atlikti tikslų modelio treniravimą, tačiau sudėtingesniuose ar didelės variacijos scenarijuose dažniausiai pranašesni tampa giluminio mokymosi (angl. *deep learning*) metodai, tokie kaip konvoliuciniai neuroniniai tinklai [30].

1.5.2. Objektų atpažinimas naudojant *CNN*

Konvoliuciniai neuroniniai tinklai (angl. *convolutional neural networks*, toliau *CNN*) yra giluminio mokymosi modeliai, skirti automatiškai išgauti požymius iš vaizdų ir atlikti objektų klasifikaciją ar aptikimą. Skirtingai nuo tradicinių metodų, tokių kaip *SVM*, *CNN* nereikalauja požymių išgavimų, nes tinklo konvoliuciniai sluoksniai gali automatiškai identifikuoti svarbiausius vaizdo elementus, tokius kaip kraštai, kampai, tekstūros ar spalvų gradientai,

o gilesniuose sluoksniuose šie elementai kombinuojami į sudėtingesnes reprezentacijas, leidžiančias atpažinti objektus net sudėtingose scenose [31].

CNN architektūra dažniausiai susideda iš konvoliucinių sluoksnių, kurie aptinka vietinius vaizdo požymius, sujungimo (angl. *pooling*) sluoksnių, kurie sumažina duomenų dimensiją ir išlaiko svarbiausią informaciją, bei pilnai sujungtų (angl. *fully connected*) sluoksnių, kurie atlieka galutinę klasifikaciją (1.15 pav.).



1.15 pav. įprasta konvoliucinio neuroninio tinklo architektūra

Objektų aptikimui *CNN* metodai dažnai integruojami su regionų pasiūlymo sistemomis arba naudojami specializuoti vieno žingsnio arba dviejų žingsnių detektoriai, tokie kaip *YOLO*, *SSD* ar *Faster R-CNN*. Vieno žingsnio detektoriai tiesiogiai prognozuoja objektų klasę ir ribas (angl. *bounding boxes*) per vieną tinklo praeitį, o dviejų žingsnių metodai pirmiausia pasiūlo galimas objekto vietas ir vėliau jas klasifikuoja bei tikslina ribas [32].

CNN naudojimo privalumas yra gebėjimas apdoroti įvairių variacijų objektus, skirtingus dydžius, orientacijas. Tai daro *CNN* ypač tinkamus realaus pasaulio objektų aptikimui, kur objektai gali būti uždengti, pasukti arba dalinai matomi. *CNN* metodai plačiai naudojami autonominiame vairavime, medicininių vaizdų analizėje, veidų atpažinime, stebėjimo sistemose, pramoniniuose kokybės kontrolės procesuose ir robotikoje [33].

Tačiau *CNN* modeliai reikalauja didelio mokymui skirtu duomenų kiekio ir galingų skaičiavimo resursų, ypač treniruojant sudėtingus objektų aptikimo tinklus. Be to, dėl itin sudėtingos konvoliucinio tinklo vidinės struktūros, rezultatai sunkiai interpretuojami analitiškai, todėl sunku nustatyti, kurie požymiai buvo lemiami galutiniam sprendimui. Nepaisant to, dėl automatinio požymių išgavimo ir galimybės vienu metu atlikti klasifikaciją, *CNN* metodai šiuolaikiniuose objektų aptikimo uždaviniuose dažnai pranoksta tradicinius metodus, tokius kaip *SVM* [30].

Apibendrinant, *CNN* leidžia aptikti objektus ir nustatyti jų pozicijas vaizde, automatiškai išgaunant sudėtingus ir įvairius požymius. Šie metodai ypač naudingi tais atvejais, kai

objektai gali pasirodyti įvairių formų, dydžių ar kampų, o rankiniu būdu parenkami požymiai nebūtų pakankami [34].

1.6. Teksto atpažinimo galimybės

Teksto atpažinimas (angl. *optical character recognition*, toliau *OCR*), yra procesas, kurio tikslas – automatiškai identifikuoti raides, skaičius ar kitus simbolius skaitmeniniuose vaizduose ir paversti juos į redaguojamą tekstą.

Šiuolaikiniai sprendimai dažniausiai remiasi giluminio mokymosi metodais, ypač konvoliuciniais ir rekursiniais neuroniniais tinklais. *CNN* gali automatiškai išgauti svarbiausius vizualinius požymius iš simbolių vaizdų, tokius kaip kraštai, linijos ar kampai, nepriklausomai nuo šrifto ar spalvų variacijos. Tuo tarpu rekurentiniai neuroniniai tinklai (angl. *recurrent neural networks*, toliau *RNN*), leidžia modeliuoti teksto sekas, atsižvelgiant į raidžių tarpusavio priklausomybę, kas ypač svarbu nuoseklus teksto atpažinimui eilutėse. Dažnai *CNN* ir *RNN* derinami, kad *CNN* išgautų vizualinius požymius, o *RNN* interpretuotų juos kaip teksto seką [35].

Be minėtų metodų, vis dažniau taikomi ir transformerių pagrindu veikiančys modeliai, kurie gali apdoroti visą teksto eilutę vienu metu, suprasdami simbolių ar žodžių kontekstą. Šie modeliai ypač naudingi sudėtingiems dokumentams, ranka rašytam tekstui arba scenoms su triukšmu bei iškraipymais, nes jie geba interpretuoti kontekstą ir atpažinti simbolius, net jei jie nėra visiškai aiškūs [36].

Kiekviena iš šių technologijų turi savo privalumų ir apribojimų. Klasikiniai metodai paprastai yra greiti, nereikalauja daug duomenų mokymui ir yra lengvai interpretuojami, tačiau jų taikymas ribotas sudėtinguose scenarijuose. *CNN* ir *RNN* metodai leidžia automatiškai išgauti ir interpretuoti sudėtingus simbolių požymius, užtikrinant aukštą tikslumą įvairiomis sąlygomis, tačiau reikalauja didelio duomenų kiekio mokymui ir galingų skaičiavimo resursų. Transformeriai suteikia didesnę lankstumą ir gebėjimą apdoroti kontekstą visos eilutės mastu [37].

1.6.1. OCR įrankių palyginimas

Siekiant įvertinti kuris *OCR* sprendimas būtų labiausiai tinkamas *EventStorming* artefaktuose esančiam tekstui atpažinti, buvo atlikta populiariausių *OCR* įrankių analizė, ir sudaryta jų palyginimo lentelė [38] [39].

Iš 1 lentelės matoma, kad nagrinėti *OCR* sprendimai skiriasi tiek savo veikimo principu, tiek taikymo galimybėmis ir kaštais. *PaddleOCR*, kaip atviro kodo sprendimas, suteikia galimybę vykdyti teksto atpažinimą lokaliai be interneto ryšio ir papildomų finansinių kaštų. Be to, jis pasižymi integruotu teksto aptikimo ir atpažinimo funkcionalumu, todėl gali automatiškai nustatyti teksto regionus vaizde. Vis dėlto jo tikslumas (79%) yra žemesnis nei debesų pagrindu veikiančių sprendimų, o rezultatai gali priklausyti nuo vaizdo kokybės bei teksto sudėtingumo.

1 lentelė. OCR įrankių palyginimas

Sprendimas	Prieinamumas	Pritaikymas	Reikalingas interneto ryšys	Tikslumas	Teksto regiono aptikimas	Kaina
Paddle OCR [40]	<i>Atviro kodo</i>	<i>Aiškliai matomo spausdinto ir ranka rašyto teksto atpažinimas iš vaizdų.</i>	<i>Ne</i>	79% [39]	<i>Ne</i>	<i>Nemokamas</i>
Azure Vision [41]	<i>Uždaro kodo, API</i>	<i>Spausdinto ir ranka rašyto teksto atpažinimas įvairiomis kalbomis ant įvairių paviršių</i>	<i>Taip</i>	89% [39]	<i>Taip</i>	<i>~\$1.50 per 1000 vaizdų, pirmi 500 vaizdų nemokami</i>
Google Cloud Vision [42]	<i>Uždaro kodo, API</i>	<i>Spausdinto ir ranka rašyto teksto atpažinimas įvairiomis kalbomis ant įvairių paviršių</i>	<i>Taip</i>	89% [39]	<i>Taip</i>	<i>\$1.50 per 1000 vaizdų, pirmi 1000 vaizdų nemokami</i>

Tuo tarpu debesų pagrindu veikiančys sprendimai, tokie kaip *Azure Vision* ir *Google Cloud Vision*, pasižymi aukštesniu tikslumu (89%) ir platesnėmis pritaikymo galimybėmis, įskaitant spausdinto ir ranka rašyto teksto atpažinimą įvairiomis kalbomis bei sudėtingose vizualinėse sąlygose. Šie sprendimai nereikalauja sudėtingo išankstinio duomenų apdorojimo, nes naudoja pažangesnius mašininio mokymosi modelius, gebančius apdoroti įvairaus sudėtingumo vaizdus. Taip pat svarbu pabrėžti, kad abu šie sprendimai turi teksto regionų aptikimo funkcionalumą, kuris leidžia gauti atpažinto teksto ribas. Tai yra ypač svarbu nagrinėjamu atveju, nes leidžia tiksliai lokalizuoti tekstą *EventStorming* artefaktuose ir sumažina papildomų komponentų poreikį sistemoje.

Apibendrinant galima teigti, kad *PaddleOCR* yra tinkamas sprendimas tais atvejais, kai svarbus lokalus veikimas, duomenų privatumas ir minimalūs kaštai, tuo tarpu *Azure Vision* ir *Google Cloud Vision* yra labiau tinkami sudėtingesniems uždaviniams, kuriuose reikalingas aukštas tikslumas, integracija ir gebėjimas patikimai apdoroti įvairių tipų tekstą. Atsižvelgiant į tai, kad *EventStorming* artefaktuose tekstas dažnai būna pateiktas skirtingomis orientacijomis, gali būti pasuktas ar netolygiai išdėstytas, itin svarbus tampa tikslus teksto ribų nustatymas. Dėl šios priežasties *Google Cloud Vision* yra tinkamiausias sprendimas, nes pasižymi aukštu tikslumu dirbant su realaus pasaulio vaizdais bei suteikia patikimą teksto regionų aptikimą. Papildomai, šis sprendimas siūlo didesnę nemokamą naudojimo kvotą, kas yra aktualu prototipo kūrimo etape.

1.7. Teksto apdorojimo ir reikalavimų generavimo galimybės

Teksto apdorojimas yra viena iš natūralios kalbos apdorojimo (angl. *natural language processing*, toliau *NLP*) sričių, kurios tikslas – analizuoti, interpretuoti ir generuoti tekstinę informaciją. Reikalavimų išgavimo įrankio kūrimo kontekste ši sritis yra ypač svarbi, nes

leidžia automatizuoti reikalavimų analizę, jų struktūrizavimą bei generavimą iš įvairių tekstinių šaltinių, tokių kaip dokumentacija, neformalus užrašai, *EventStorming* artefaktai.

Tradiciniai teksto apdorojimo metodai dažniausiai remiasi taisyklėmis arba statistiniais modeliais. Tokiuose sprendimuose tekstas yra suskaidomas į mažesnius vienetus, tokius kaip žodžiai ar sakiniai (tokenizacija), atliekamas jų normalizavimas (pvz., mažosios raidės, kamienavimas (angl. *stemming*) ar lemavimas, pašalinami nereikšmingi žodžiai (angl. *stopwords*) ir analizuojama sintaksinė ar semantinė struktūra [43]. Reikalavimų generavimo uždaviniuose šie metodai gali būti naudojami raktinių žodžių identifikavimui, sakinių klasifikavimui ar paprastam informacijos išgavimui [44].

Tačiau tokie metodai dažnai yra riboti, nes sunkiai apdoroja sudėtingas sakinių struktūras, kontekstą ar dviprasmybes, kurios yra būdingos natūraliai kalbai. Pavyzdžiui, sakinyje „Sistema turi greitai apdoroti užklausas“ tradiciniai metodai gali identifikuoti raktinius žodžius (sistema, apdoroti, užklausas), tačiau negali tiksliai nustatyti, ką reiškia „greitai“ (pvz., ar tai 1 sekundė, ar 1 ms), nes trūksta kontekstinio supratimo ir semantinės interpretacijos [45].

Šiuolaikiniai sprendimai vis dažniau remiasi dideliais kalbos modeliais (angl. *large language models*, toliau *LLM*), kurie yra treniruojami naudojant didelius tekstinių duomenų kiekius ir geba generuoti nuoseklų, kontekstu grįstą tekstą [46]. Tokie modeliai paremti transformerių architektūra geba suprasti sakinių kontekstą, semantiką ir tarpusavio ryšius, todėl gali būti naudojami ne tik teksto analizavimui, bet ir reikalavimų generavimui. *LLM* gali interpretuoti neformalius naudotojo aprašymus ir transformuoti juos į struktūruotus programinės įrangos reikalavimus, išskirdami pagrindinius veikėjus ir veiksmus [47].

Vienas iš pagrindinių *LLM* privalumų yra gebėjimas dirbti su natūralia kalba be griežtų taisyklių ar iš anksto apibrėžtų struktūrų. Tai leidžia naudoti modelius įvairiuose scenarijuose, pavyzdžiui, generuojant naudotojo pasakojimus (angl. *user stories*), kuriant funkcinis reikalavimus arba apibendrinant ilgus dokumentus [46]. Be to, *LLM* gali būti pritaikyti konkrečiai sričiai naudojant papildomą mokymą (angl. *fine-tuning*) arba pateikiant kontekstinius pavyzdžius (angl. *prompting*), kas leidžia pagerinti generuojamų reikalavimų kokybę ir tikslumą [48].

Nepaisant šių privalumų, *LLM* turi ir tam tikrų ribotumų. Modeliai gali generuoti netikslią ar nepagrįstą informaciją, kitaip vadinamas „haliucinacijas“, jei pateiktas kontekstas yra nepakankamas arba dviprasmiškas [49]. Be to, generuojamų reikalavimų kokybė ypač priklauso nuo įvesties duomenų aiškumo ir struktūros [50].

Apibendrinant galima teigti, kad teksto apdorojimo ir reikalavimų generavimo galimybės ženkliai išsiplėtė pereinant nuo taisyklėmis pagrįstų metodų prie giluminio mokymosi ir didelių kalbos modelių. Tradiciniai metodai išlieka naudingi paprastesniems uždaviniams, tačiau sudėtingesniuose scenarijuose, kur reikalingas konteksto supratimas ir natūralios kalbos generavimas, *LLM* tampa pagrindiniu pasirinkimu [51].

1.7.1. *LLM* palyginimas teksto apdorojimui ir reikalavimų generavimui

Siekiant įvertinti, kuris *LLM* yra tinkamas reikalavimų išgavimui iš *EventStorming* artefaktų, buvo atlikta įvairių *LLM* analizė ir sudaryta 2 lentelė. Jiems tarpusavyje įvertinti buvo

pasirinktas *Massive Multitask Language Understanding-Pro*, toliau *MMLU-Pro*, metodas. *MMLU-Pro* yra išplėstinė *MMLU* testavimo priemonė, sukurta siekiant tiksliai įvertinti modelių žinių gebėjimus plačioje temų įvairovėje ir skirtingo sudėtingumo uždaviniuose. Šis metodas apima daugiau nei 57 akademinės sritis, įskaitant technines, humanitarines ir gamtos mokslų sritis, todėl ji yra tinkama vertinti bendro pobūdžio supratimą ir gebėjimą generuoti nuoseklų, informatyvų tekstą [52].

MMLU-Pro buvo pasirinktas dėl kelių svarbių priežasčių. Testai apima tiek techninius, tiek netechninius klausimus, panašius į realias problemų situacijas, su kuriomis gali susidurti reikalavimų išgavimas iš *EventStorming* artefaktų. Be to, modeliai turi ne tik atpažinti faktinę informaciją, bet ir tinkamai ją struktūruoti bei interpretuoti, todėl *MMLU-Pro* tiksliai parodo modelių gebėjimą generuoti struktūruotą ir nuoseklų tekstą. Atsižvelgiant į šiuos aspektus, *MMLU-Pro* gali būti naudojama kaip patikima priemonė modelių pasirinkimui, leidžiant nustatyti, kurie *LLM* geriausiai atlieka struktūruoto teksto generavimo užduotis bei reikalavimų išgavimą iš sudėtingų tekstinių šaltinių [52].

Iš 2 lentelės matoma, kad nagrinėjami *LLM* pasižymi skirtingais konteksto langais, *MMLU-Pro* rezultatu ir tokenų sunaudojimu, kas tiesiogiai įtakoja jų tinkamumą reikalavimų išgavimui iš *EventStorming* artefaktų. Įprastai modeliai su didesniu kontekstu, tokie kaip *Claude Opus 4.5*, *Claude Sonnet 4.5* ir *Gemini 2.5 Pro*, geba apdoroti ilgesnius tekstus ir palaikyti nuoseklų kontekstą per visą dokumento turinį.

2 lentelė. *LLM* palyginimas [53]

Modelis	Konteksto langas (tokenais)	<i>MMLU-Pro</i> įvertis	Tokenų sunaudojimas <i>MMLU-Pro</i> testavimo metu
<i>GPT-5.2</i>	400 000	87,4%	35M
<i>GPT-4o</i>	128 000	80,8%	6,5M
<i>Claude Opus 4.5</i>	1 000 000	88,9%	7,5M
<i>Claude Sonnet 4.5</i>	1 000 000	87,5%	7,2M
<i>Gemini 2.5 Pro</i>	1 000 000	86,2%	45M

MMLU-Pro rezultatai rodo, kad *Claude Opus 4.5* pasiekė aukščiausią *MMLU-Pro* rezultatą (88,9%), o *GPT 5.2* taip pat demonstruoja aukštą našumą (87,4%). Tai leidžia teigti, kad šie modeliai gali tiksliai suprasti įvairaus sudėtingumo klausimus ir generuoti nuoseklų, struktūruotą tekstą, reikalavimus.

Tokenų sunaudojimo analizė atskleidžia, kad mažesnis tokenų sunaudojimas, kaip matyti *GPT 4o* ir *Claude Sonnet 4.5* atveju, paprastai reiškia mažesnes kaštų sąnaudas. Tuo tarpu didesnis tokenų sunaudojimas, pavyzdžiui, *Gemini 2.5 Pro*, gali lemti didesnes eksploatacijos išlaidas, nors modelis suteikia galimybę apdoroti labai ilgą ir sudėtingą tekstinę informaciją.

Atsižvelgiant į visus šiuos aspektus, pasirinkimas tarp modelių priklauso nuo didelio tikslumo poreikio, apdorojamo teksto ilgio bei kaštų apribojimų. Modeliai su didesniu konteksto langų ir aukštesniu *MMLU-Pro* rezultatu yra labiau tinkami sudėtingesnių veiklos sričių, sistemų

reikalavimų generavimui, o modeliai su mažesniu tokenų sunaudojimu gali būti ekonomiškesni sprendžiant kasdienes, paprastesnes teksto generavimo užduotis.

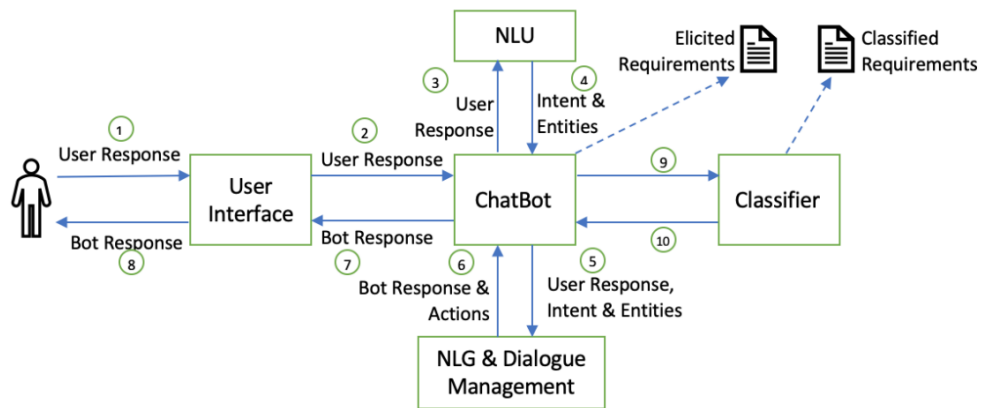
1.8. Reikalavimų išgavimo iš *EventStorming* ar panašių artefaktų metodų analizė

Siekiant suprasti esamus šio darbo problemos sprendimo būdus buvo nuspręsta atlikti lyginamąją analizę tarp įrankių, kurie automatizuotai išgauna reikalavimus iš natūralios kalbos ar kitų artefaktų apibrėžiančių reikalavimus.

Publikacija „*Automated Text Mining for Requirements Analysis of Policy Documents*“ nagrinėja reikalavimų išgavimo iš organizacinių politikos dokumentų problemą, kuri, anot autorių, yra sudėtinga, daug laiko reikalaujanti ir dažniausiai atliekama rankiniu būdu. Tradicinėje reikalavimų inžinerijoje analitikai turi kruopščiai peržiūrėti didelius kiekius tekstinių dokumentų, tokių kaip privatumo politikos ar naudojimosi sąlygos, ir iš jų identifikuoti sistemai svarbius reikalavimus. Šį procesą dar labiau apsunkina tai, kad tokie dokumentai paprastai yra parašyti sudėtinga teisine kalba, todėl jų analizė tampa ne tik lėta, bet ir linkusi į klaidas. Autoriai siūlo šią problemą spręsti pasitelkiant automatizuotą teksto analizę, panaudojant organizacijose jau egzistuojančius dokumentus kaip reikalavimų šaltinį. Šiuose dokumentuose dažnai yra aprašomi organizacijos įsipareigojimai, pavyzdžiui, kokie duomenys renkami, kaip jie saugomi ar su kuo dalijamasi, todėl jie gali būti interpretuojami kaip programinės sistemos reikalavimai. Siekiant šią informaciją išgauti efektyviau, straipsnyje taikomas temų modeliavimas, paremtas *Latent Dirichlet Allocation*, toliau *LDA* algoritmu. Šis metodas leidžia automatiškai analizuoti didelius tekstų kiekius ir nustatyti paslėptas temas, remiantis žodžių pasikartojimo struktūra. Šios temos siejamos su iš anksto apibrėžtais raktažodžiais, tokiais kaip „rinkti“, „saugoti“, „informuoti“ ar „dalintis“, kurie reikalavimų inžinerijoje interpretuojami kaip galimi sisteminiai veiksmai ar įsipareigojimai. Remiantis tokia analize, galima išgauti konkrečius reikalavimų tipus. Pavyzdžiui, iš teiginių apie duomenų rinkimą gali būti suformuluotas funkcinis reikalavimas, kad sistema turi rinkti naudotojo pateiktą informaciją registracijos metu. Iš nuostatų apie duomenų saugojimą gali būti išvestas reikalavimas, kad sistema turi saugoti duomenis saugioje aplinkoje ir užtikrinti jų apsaugą nuo neteisėtos prieigos. Apibendrinant galima teigti, kad straipsnyje siūlomas metodas leidžia iš dalies automatizuoti reikalavimų išgavimo procesą, sumažinti rankinio darbo kiekį ir efektyviau analizuoti didelius dokumentų kiekius. Tai ypač svarbu šiuolaikinėje informacinėje aplinkoje, kur organizacijos privalo laikytis teisinių reikalavimų, o jų atitikimas turi būti užtikrinamas ir programinėse sistemose [54].

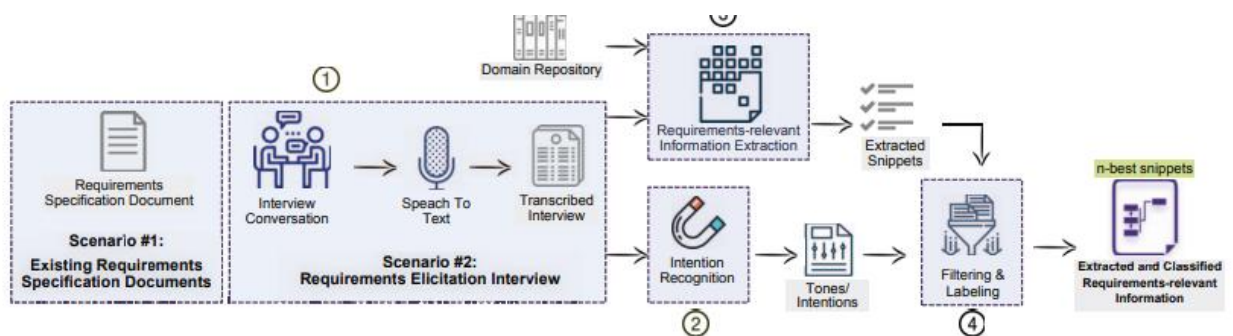
Straipsnyje „*Intelligent Chatbot for Requirements Elicitation and Classification*“ apžvelgiama didelio kiekio reikalavimų surinkimo ir klasifikavimo problema, kuri sakoma, jog yra daug laiko užimanti ir neatspari klaidoms. Autoriai siūlo naudoti mašininį mokymąsi ir dirbtinį intelektą siekiant sukurti pokalbių programinę įrangą (angl. *chatbot*), kurią naudojant suinteresuoti asmenys galėtų natūralia kalba apibrėžti reikalavimus, kuriuos vėliau sistema paverstų formaliai reikalavimais ir juos suklasifikuotų į funkcinis ir nefunkcinis reikalavimus. Įrankio veikimo konceptas yra matomas 1.16 pav. Sukurta pokalbių programinė įranga (angl. *chatbot*) naudoja natūralios kalbos supratimo (angl. *natural language understanding*), generavimo, teksto apdorojimo ir mašininio mokymosi algoritmus komunikacijai ir reikalavimų išgavimui, o reikalavimų klasifikavimui buvo išbandyti *Naive Bayes* ir *Support Vector Machine* algoritmai. Verta paminėti, jog straipsnyje minima pokalbių

programinė įranga (angl. *chatbot*) sugebėjo dirbti tik siauroje elektroninės bankininkystės dalykinėje srityje. Autoriai nurodo, jog norint, kad ši įranga galėtų vienareikšmiškai suformuluoti ir kitų dalykinių sričių reikalavimus, būtų reikalingi kitų dalykinių sričių apmokymo duomenys [55].



1.16 pav. Straipsnio „Intelligent Chatbot for Requirements Elicitation and Classification“ įrankio veikimo konceptas [55]

Straipsnyje „*ELICA: An Automated Tool for Dynamic Extraction of Requirements Relevant Information*“ apžvelgiama reikalavimų išgavimo ir jų specifikavimo problema, kai dalykinė sritis nėra iki galo suprantama, tokiu atveju autoriai mini, jog kyla nesusikalbėjimo, vienas kito nesupratimo situacijos tarp veiklos analitiko ir suinteresuoto asmens, dalykinės srities išsiaiškinimas trunka daug laiko. Autoriai siūlo naudoti sukurtą reikalavimų išgavimo įrankį, kuris padeda iš natūralios kalbos interviu metu išgauti su reikalavimais susijusią informaciją, remiantis ne tik išsakytais žodžiais, bet ir kalbėjimo tonu, pasitikėjimu, emocijomis. Įrankio pažymėta informacija svarbi reikalavimams specifiuoti yra pateikiama analitikui. 1.17 pav. matomas įrankio veikimo konceptas, jis remiasi kalbos atpažinimo priemone *Weighted Finite State Transducers*, kuri leidžia natūralioje kalboje atpažinti naudingą dalykinės srities informaciją reikalavimams sudaryti. Įrankis remiasi saugykla, kurioje saugoma tekstinė informacija apie dalykinę sritį. Konkrečiu straipsnio atveju įrankis galėjo veikti tik bilietų pardavimo dalykinėje srityje. Autoriai pažymi, jog įrankis gali būti pritaikytas ir kitoms dalykinėms sritims, tačiau reikėtų įrankiui pateikti pakankamai duomenų apie norimą analizuoti dalykinę sritį ir sudaryti minėtą saugyklą. Verta paminėti, kad įrankis gali būti naudojamas reikalavimų specifikacijoms tikslinti, naudojant pradinį reikalavimų specifikacijos dokumentą kaip dalykinės srities saugyklos duomenis [56].



1.17 pav. Straipsnio „*ELICA: An Automated Tool for Dynamic Extraction of Requirements Relevant Information*“ įrankio veikimo konceptas

Straipsnyje „*A semi-automated approach for generating natural language requirements documents based on business process models*“ apžvelgiama reikalavimų išgavimo ir specifikavimo problema, kuri anot autorių užima daug laiko ir yra itin kompleksiška. Siekiant palengvinti šį procesą autoriai siūlo pusiau automatizuotą procesą, kurio metu analitikas kuria veiklos modelį, o juo naudojantis automatiškai generuojami reikalavimai. Verta paminėti, jog šis straipsnis apsiriboja įvykiais pagrįstų procesų grandinių (angl. *Event-Driven process chain*) konvertavimu į tekstinius reikalavimus. Įrankis remiasi veiklos modeliu ir aprašytomis taisyklėmis, kurios parenka kokį reikalavimo aprašymo šabloną naudoti. Iš modelio yra išrenkami atitinkamo tipo elementai ir parinktas šablonas užpildomas reikšmėmis. Straipsnyje minima, jog šis sprendimas galėtų būti pritaikytas ir *BPMN* notacija aprašytam veiklos modeliui. Taip pat yra pabrėžiama, jog toks reikalavimų generavimas gali būti interpretuojamas kaip reikalavimų pavertimas iš vienos formos, šiuo atveju modelio į kitą formą – tekstu aprašytą reikalavimą. Be to verta paminėti, jog įrankis buvo testuojamas tik vienos organizacijos kontekste, todėl nėra aišku kokius rezultatus įrankis gautų naudojant kitos dalykinės srities veiklos modelius [4].

Apžvelgus esamus problemos sprendimo būdus, buvo sudaryta 3 lentelė, kurioje palyginami nagrinėti reikalavimų išgavimo sprendimai pagal jų automatizavimo lygį, taikomą įvesties tipą, pritaikomumą skirtingoms dalykinėms sritims ir tinkamumą darbui su *EventStorming* artefaktais.

3 lentelė. Esamų problemos sprendimo būdų palyginimas

Kriterijus	[55]	[56]	[4]	[54]
Dalykinė sritis	<i>Bankininkystė</i>	<i>Bilių pardavimas</i>	<i>Nespecializuota</i>	<i>Tinka toms, kurios naudoja teisinius, vidaus tvarkos dokumentus savo veiklai apibrėžti</i>
Reikalingi duomenys apie dalykinę sritį apmokymui	<i>Taip</i>	<i>Taip</i>	<i>Ne</i>	<i>Ne</i>
Automatizavimo lygis	<i>Automatinis</i>	<i>Automatinis</i>	<i>Automatizuotas</i>	<i>Automatizuotas</i>
Realizacija	<i>Pokalbių programinė įranga (angl. chatbot) bei klasifikatorius</i>	<i>Kalbos atpažinimo priemonė Weighted Finite State Transducers ir klasifikatorius</i>	<i>Iš anksto apibrėžtos veiklos taisyklės remiantis diagramos notacija ir šablonų pildymas</i>	<i>Iš anksto apibrėžtų raktažodžių paieška naudojant naudoja Latent Dirichlet Allocation mašininio mokymosi algoritmą</i>
Reikalavimų klasifikavimas	<i>Taip</i>	<i>Taip</i>	<i>Ne</i>	<i>Ne</i>
Įveiga	<i>Natūralios kalbos tekstas</i>	<i>Natūrali kalba, dalykinės srities aprašai</i>	<i>Įvykiais pagrįstų procesų grandinių veiklos diagramos</i>	<i>Teisiniai ir vidaus tvarkos dokumentai apibrėžiantys veiklą</i>
Išveiga	<i>Klasifikuotas reikalavimų sąrašas</i>	<i>Klasifikuota, reikalavimams aktuali informacija</i>	<i>Reikalavimų sąrašas</i>	<i>Reikalavimų sąrašas</i>

Remiantis 3 lentelėje pateiktais sprendimais, galima pastebėti, kad automatinis reikalavimų sąrašo sudarymas pasiekiamas tuomet, kai kaip įvestis naudojama natūrali kalba – tiek tekstas, tiek garso įrašai. Tokiais atvejais sistemos gali tiesiogiai analizuoti naudotojų išsakytus poreikius ir automatiškai generuoti bei klasifikuoti struktūrizuotus reikalavimus. Tačiau tais atvejais, kai reikalavimų išgavimui naudojami organizacijos teisiniai, vidaus tvarkos dokumentai arba įvykiais pagrįstų procesų grandinių veiklos diagramos, procesas yra tik iš dalies automatizuotas. Tokiose metodikose dažniausiai reikalingas papildomas paruošimas, pavyzdžiui, iš anksto apibrėžti raktažodžiai, taisyklės ar šablonai, todėl galutinis reikalavimų suformavimas nėra pilnai automatinis.

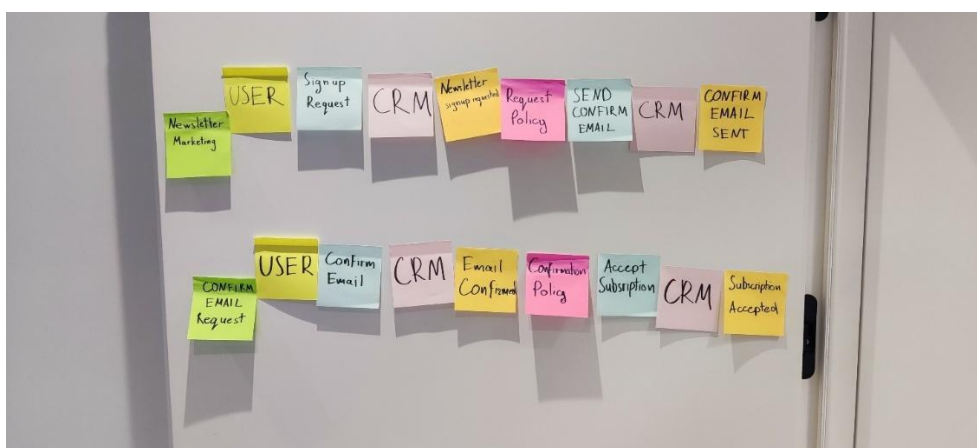
Taip pat galima pastebėti, kad nors sprendimai, pateikti straipsniuose [55] ir [56], yra automatiniai, tačiau orientuoti į konkrečias dalykines sritis, tokias kaip bankininkystė ar bilietų pardavimas, todėl jų pritaikymas kitose srityse reikalauja papildomo apmokymo su specifiniais duomenimis. Tuo tarpu sprendimai [4] ir [54] nėra pilnai automatiniai, jis remiasi iš anksto apibrėžtomis taisyklėmis arba raktažodžiais ir reikalauja papildomo konfigūravimo.

Analizuojant mokslinę literatūrą, nepavyko rasti pakankamai išplėtotų sprendimų, kurie leistų pilnai automatizuoti reikalavimų generavimą iš *EventStorming* artefaktų. Esami metodai daugiausia orientuojasi į natūralios kalbos analizę arba struktūrizuotų modelių transformaciją, tačiau *EventStorming* kontekste automatizavimo galimybės išlieka ribotos ir nepakankamai ištirtos.

Todėl šis tiriamasis darbas siekia užpildyti šią spragą, pasiūlydamas metodą, kuris leistų pilnai automatizuoti reikalavimų sudarymo procesą, remiantis *EventStorming* artefaktais, būtų pritaikomas skirtingose dalykinėse srityse be papildomo apmokymo ir taip palengvintų šį reikalavimų išgavimo procesą.

1.9. Reikalavimų išgavimas iš *EventStorming* artefaktų naudojant *LLM* įrankius

Siekiant įvertinti, ar bendros paskirties *LLM* galėtų būti naudojami tiesioginiam reikalavimų išgavimui iš *EventStorming* artefaktų, buvo atliktas bandymas *EventStorming* paveikslą tiesiogiai pateikiant modeliams *GPT-5.2* ir *Gemini 3*. Abiem modeliams buvo pateiktas tas pats *EventStorming* paveikslas (1.18 pav.) ir bendro pobūdžio užklausa sugeneruoti naudotojo pasakojimus pagal pateiktą paveikslą. Pateikta užklausa modeliams „*Create user stories out of this Eventstorming image*“. Gauti atsakymai pateikti 1 priede.



1.18 pav. *EventStorming* paveikslėlis naudotas reikalavimų išgavimui su bendros paskirties *LLM*

Analizuojant *LLM* atsakymus, abu modeliai gebėjo atpažinti bendrą nagrinėjamos diagramos temą, t. y. naujienlaiškio prenumeratos ir el. pašto patvirtinimo procesą. Vis dėlto detalesnė sugeneruotų atsakymų analizė parodė, kad vien tik paveikslo pateikimas *LLM* nėra pakankamas patikimam automatizuotam reikalavimų išgavimui.

GPT-5.2 atsakyme buvo sugeneruoti elementai, kurie nebuvo tiesiogiai pateikti *EventStorming* artefaktuose. Pavyzdžiui, atsakyme atsirado tokie papildomi elementai kaip *Request Policy Evaluated*, *Record Confirmation Email Sent*, papildomos būsenos bei siūlomi agregatai ir taisyklės. Tai rodo, kad modelis dalį proceso ne perskaitė iš artefaktų, o išvedė remdamasis bendromis žiniomis apie prenumeratos scenarijus. Be to, modelis pakeitė ir kai kurių artefaktų formuluotes: aktorių *User* interpretavo kaip *Website visitor*, o naudotojo tikslus papildė frazėmis apie *marketing communications*, nors tokia reikšmė pačiuose artefaktuose nebuvo tiesiogiai nurodyta. Taip pat buvo pervadintos komandos, pridėdant papildomus žodžius, kurie keičia pirminę artefaktų semantiką ir gali lemti netikslias išvestus naudotojo pasakojimus.

Gemini 3 atsakyme buvo pastebėtos panašios problemos. Modelis darė prielaidą, kad paveiksle naudojama universali *EventStorming* spalvų semantika, nors praktikoje spalvų reikšmės gali būti pasirenkamos individualiai konkrečiam modeliui ar dirbtuvėms. Atsakyme taip pat atsirado papildomi tarpiniai žingsniai, tokie kaip *Trigger Request Policy*, *Process Email* ir *Finalize Record*, kurie nėra tiesioginiai paveiksle pateikti artefaktai, o veikiau modelio sukonstruota interpretacija.

Abiejų modelių atsakymuose taip pat pasireiškė kelios bendros problemos. Pirma, modeliai sugeneruodavo papildomus įvykius, komandas ar tarpinius proceso žingsnius, kurie nebuvo pavaizduoti *EventStorming* artefaktuose. Antra, modeliai keitė tikslas artefaktų formuluotes, nors *EventStorming* kontekste net ir nedideli pavadinimų pakeitimai gali pakeisti artefakto reikšmę. Trečia, keli atskiri proceso žingsniai dažnai būdavo sujungiami į vieną pernelyg platų naudotojo pasakojimą, nors juos buvo galima išskaidyti į 2-3 atskirus, aiškiau apibrėžtus ir atskirai įgyvendinamus naudotojo pasakojimus. Ketvirta, dalis paveiksle esančių artefaktų, pavyzdžiui, skaitymo modeliai ar tikslūs ryšiai tarp komandų ir įvykių, buvo ignoruojami arba paverčiami bendro pobūdžio paaiškinimais. Penkta, sugeneruoti atsakymai nebuvo pateikti nuosekliu, struktūrizuotu formatu, kuris galėtų būti tiesiogiai naudojamas tolesniame reikalavimų valdymo procese, pavyzdžiui, eksportavimui ar projekto kūrimui.

Remiantis atliktu bandymu galima teigti, kad bendros paskirties *LLM* gali būti naudingi preliminariam *EventStorming* paveikslo turinio interpretavimui, tačiau jų atsakymai nėra pakankamai tikslūs, detalūs ir struktūrizuoti, kad galėtų būti naudojami kaip pagrindinis automatizuoto reikalavimų išgavimo mechanizmas. Dėl šios priežasties tiesioginis paveikslo pateikimas *LLM* turėtų būti vertinamas tik kaip pagalbinė priemonė, o ne kaip pakankamas sprendimas patikimam reikalavimų išgavimui iš *EventStorming* artefaktų. Atlikta analizė taip pat parodė, kad siekiant tiksliai išgauti reikalavimus iš *EventStorming* artefaktų, *LLM* tikslinga papildomai pritaikyti šiai užduočiai arba pateikti jam papildomus struktūrizuotus duomenis. Tokie duomenys galėtų apimti iš anksto atpažintus artefaktų tipus, jų tekstinį

turinį, tarpusavio ryšius ir klasterius, taip sumažinant interpretavimo laisvę ir haliucinacijų tikimybę.

1.10. Siekiamo sprendimo apibrėžimas

Siekama sukurti įrankį, kuris naudojant *EventStorming* artefaktus automatiškai sudarytų programinės įrangos reikalavimų sąrašą. Planuojama pritaikyti konvoliuciniu neuroniniu tinklu paremtą objektų aptikimo modelį, kuris lokalizuotų *EventStorming* lapelius pateiktame vaizde. Tuomet, naudojant naudotojo nurodytas artefaktų spalvas, kiekvienam lapeliui būtų priskiriamas atitinkamas *EventStorming* artefakto tipas.

Vėliau būtų naudojamas optinis simbolių atpažinimo sprendimas, siekiant nuskaityti ant lapelių esantį tekstą. Sekančiu žingsniu lapeliai būtų grupuojami pagal jų išdėstymą erdvėje, taip atkuriant procesų seką ir sąsajas tarp artefaktų nurodytas 1.9 pav.

Galiausiai, susisteminti, atitinkamais artefaktų tipais sužymėti ir į klasterius sujungti duomenys būtų perduoti *LLM*. Remdamasis *EventStorming* metodo taisyklėmis ir pateiktu kontekstu, šis modelis suformuos analizuojamos veiklos naudotojo pasakojimus.

1.11. Analizės išvados

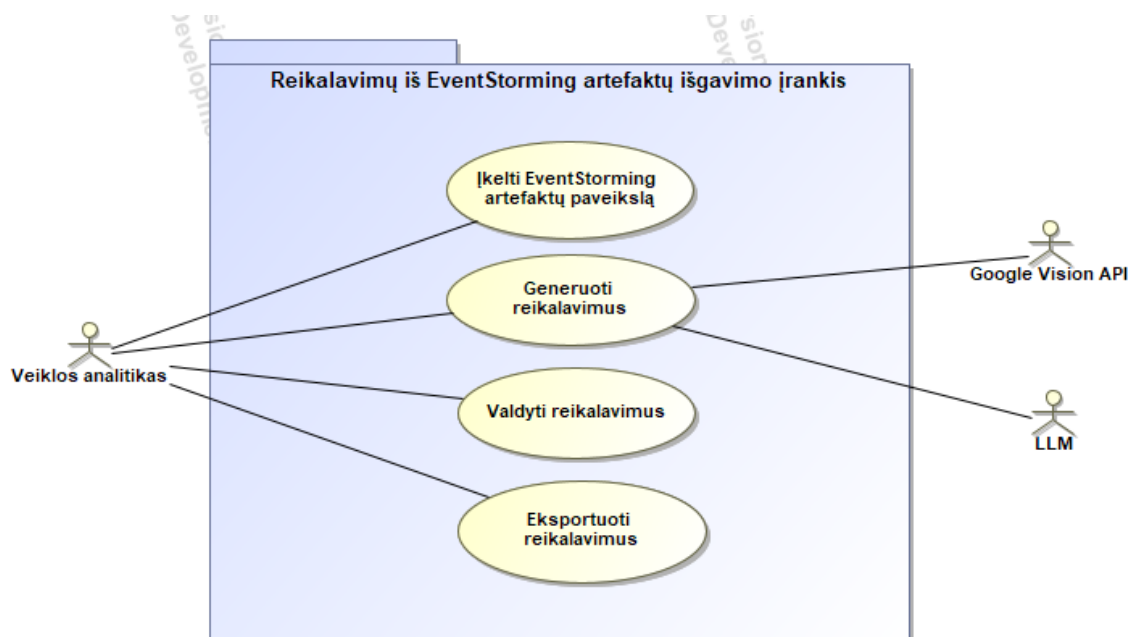
1. Apžvelgus reikalavimų inžinerijos procesą ir jo taikymą programinės įrangos kūrime, nustatyta, kad tai yra sudėtingas, kompleksiškas ir daug laiko bei pastangų reikalaujantis procesas. Kadangi šio darbo tikslas yra palengvinti reikalavimų surinkimą automatizuojant jų išgavimą iš *EventStorming* artefaktų, buvo išanalizuoti metodai, galintys prisidėti prie tokio sprendimo pagrindimo. Išanalizavus *Domain-Driven Design* sistemų kūrimo būdą nustatyta, kad jis padeda kurti specializuotus programinės įrangos sprendimus, orientuotus į konkrečios dalykinės srities problemos sprendimą. Taip pat išanalizuotas *EventStorming* reikalavimų išgavimo metodas. Pastebėta, kad šis metodas palengvina veiklos procesų identifikavimą ir jų analizę, o kartu padeda identifikuoti ir *Domain-Driven Design* elementus.
2. Išanalizavus egzistuojančius problemos sprendimo būdus buvo nustatyta, jog sukurti įrankiai arba yra specializuoti konkrečioms dalykinėms sritims arba yra dalinai automatizuoti arba nėra pritaikyti veikti su *EventStorming* artefaktais, todėl buvo nuspręsta kurti reikalavimų sudarymo įrankį iš *EventStorming* artefaktų ir eksperimentiškai ištirti įrankio sugeneruotų reikalavimų tikslumą. Taip pat numatytas siekiamo sprendimo veikimo principas, kuris remiasi konvoliuciniu neuroniniu tinklu (CNN) paremtu objektų aptikimo modeliu, optiniu simbolių atpažinimu (OCR) tekstui išgauti, bei klasterizacija, skirta atkurti artefaktų tarpusavio ryšius ir procesų struktūrą. Galiausiai šie suklasterizuoti ir sužymėti duomenys perduodami *LLM*, kuris, remdamasis *EventStorming* metodu, generuoja naudotojo pasakojimus.

2. Reikalavimų išgavimo iš *EventStorming* artefaktų įrankio specifikacija, formalus aprašas

Šiame skyriuje aprašoma naudotojo pasakojimų išgavimo iš *EventStorming* artefaktų įrankio reikalavimų modelis.

2.1. Reikalavimų specifikacija

Reikalavimų išgavimo iš *EventStorming* artefaktų įrankio funkciniai reikalavimai šiame skyriuje yra pateikiami panaudojant panaudojimo atvejų ir veiklų diagramas apibrėžtas UML specifikacija. 2.1 pav. pateikiama įrankio panaudojimo atvejų diagrama.

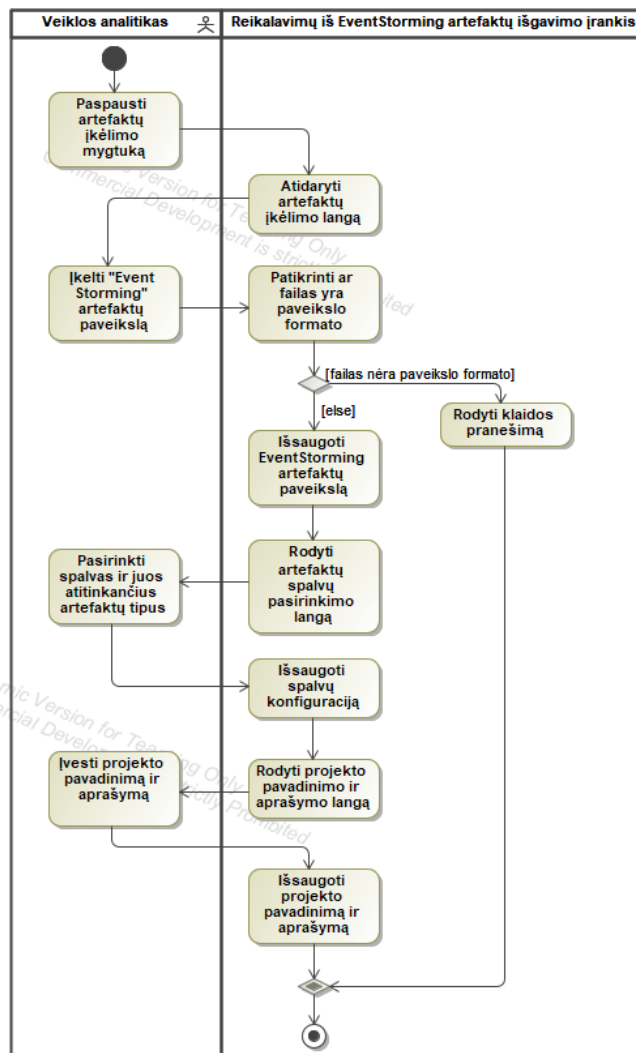


2.1 pav. Reikalavimų iš *EventStorming* artefaktų išgavimo įrankio panaudojimo atvejų diagrama

Reikalavimų išgavimo iš *EventStorming* artefaktų įrankyje išskiriamas veiklos analitiko, optinio simbolių atpažinimo (*Google Vision API*), specializuoto instrukcijomis pritaikyto *EventStorming* LLM aktoriai. Veiklos analitikas gali atlikti keturis veiksmus: įkelti *EventStorming* artefaktų paveikslą, taip pat generuoti reikalavimus iš įkelto *EventStorming* artefaktų paveikslo, vėliau peržiūrėti, kurti, redaguoti ir šalinti sugeneruotus reikalavimus, kurie diagramoje apibrėžti „Valdyti reikalavimus“ panaudojimo atveju. Galiausiai veiklos analitikas gali eksportuoti reikalavimus.

Šie panaudojimo atvejai yra detalizuojami UML veiklos diagramomis, kurios aprašo, kokie naudotojo veiksmai atliekami ir kaip į juos reaguoja naudotojo pasakojimų išgavimo įrankis. 2.2 pav. - 2.5 pav. pateikiamos naudotojo pasakojimų išgavimo iš *EventStorming* artefaktų įrankio panaudojimo atvejų veiklų diagramos.

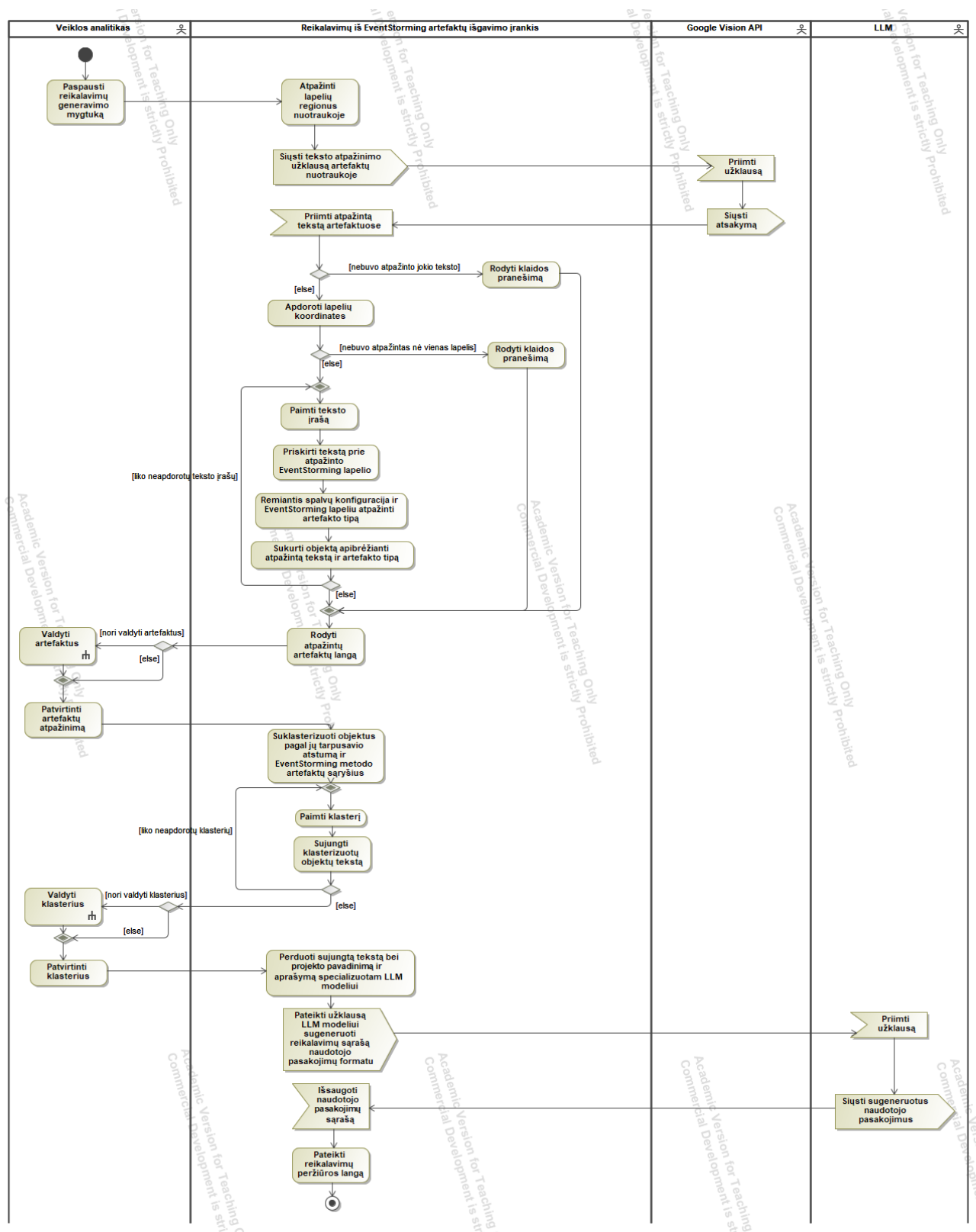
2.2 pav. yra pavaizduota *EventStorming* artefaktų paveikslo įkėlimo veiklos diagrama.



2.2 pav. „Įkelti *EventStorming* artefaktų paveikslą“ panaudojimo atvejo veiklos diagrama

Veiklos analitikas naudodamas įrankį turi galimybę pasirinkti artefaktų paveikslo įkėlimą, po kurio jam yra atidaromas artefaktų paveikslo įkėlimo langas. Šis pasirinkęs reikiamą failą įkelia artefaktų paveikslą. Įrankis patikrina ar failo formatas yra paveikslo formatas. Jei ši sąlyga yra neigiama – rodomas klaidos pranešimas. Jei failas yra paveikslas – jis yra išsaugomas. Toliau yra atverčiamas artefaktų spalvų pasirinkimo langas, kuriame veiklos analitikas pasirenka paveiksle naudojamas skirtingų artefaktų tipų spalvas. Šių spalvų konfigūracija yra išsaugoma, vėliau veiklos analitikui reikia įvesti projekto pavadinimą ir aprašymą, kuris po įvedimo yra išsaugomas ir *EventStorming* artefakto paveikslo įkėlimo veiksmai yra baigti. Po jo įprastai seka „Generuoti reikalavimus“ veiksmai.

2.3 pav. yra pateikiama panaudojimo atvejo „Generuoti reikalavimus“ veiklos diagrama.

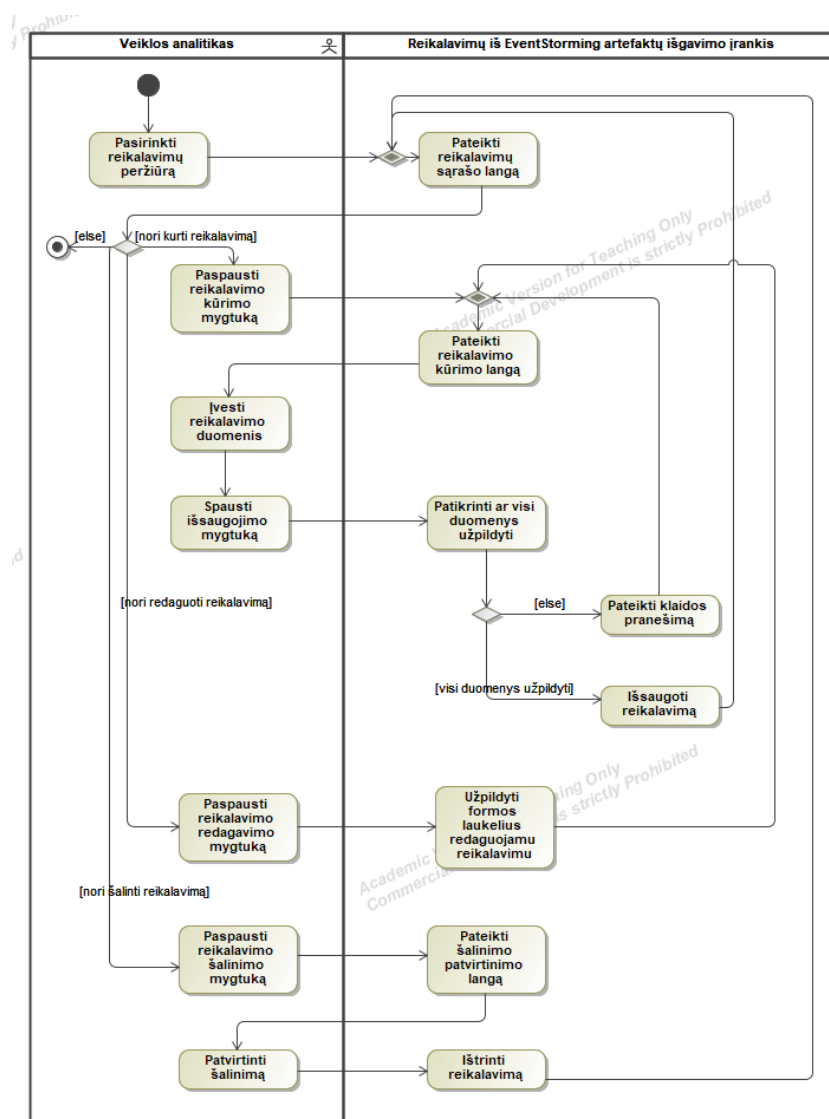


2.3 pav. „Generuoti reikalavimus“ panaudojimo atvejo veiklos diagrama.

Veiklos analitikas turi galimybę generuoti reikalavimus naudotojo pasakojimų pavidalu, remdamasis įkeltu *EventStorming* artefaktų paveikslu. Pirmiausia įrankis identifikuoja regionus, apibrėžiančius *EventStorming* lapelius. Vėliau, naudodamas *Google Vision API*, visame paveiksle atpažįsta tekstą kartu su jam priskirtais regionais. Toliau vykdomas

iteracinis procesas, kurio metu kiekvienam aptiktam lapelio regionui priskiriamas jame esantis tekstas, o remiantis iš anksto nustatyta spalvų konfigūracija identifikuojamas artefakto tipas. Šio proceso rezultatas – sukuriama objektai, apibrėžiantys atpažintą tekstą ir atitinkamą artefakto tipą. Po šių automatinė veiksmų, arba tais atvejais, kai nepavyksta aptikti nei teksto, nei lapelių regionų, veiklos analitikui suteikiama galimybė valdyti aptiktus artefaktus: juos kurti, redaguoti ar trinti. Atlikęs šiuos veiksmus, analitikas patvirtina galutinį artefaktų sąrašą. Vėlesniame etape įrankis atlieka objektų klasterizavimą, remdamasis jų tarpusavio Euklidiniu atstumu bei galimais *EventStorming* metodo artefaktų sąryšiais. Tuomet vykdomas iteracinis procesas, kurio metu kiekvienam klasteriui sugeneruojamas tekstinis aprašas, sudarytas iš jam priskirtų objektų tekstinės informacijos. Po šio etapo veiklos analitikas taip pat turi galimybę valdyti automatiškai suformuotus klasterius, juos kurti, redaguoti ar trinti, ir galiausiai patvirtinti galutinį klasterių sąrašą. Patvirtintas klasterių sąrašas, kartu su jame esančiais artefaktais bei projekto pavadinimu ir aprašymu, perduodamas specializuotam *EventStorming LLM*, kuris sugeneruoja reikalavimų sąrašą naudotojo pasakojimų formatu. Sugeneruoti reikalavimai yra išsaugomi, o veiklos analitikas nukreipiamas į reikalavimų peržiūros langą.

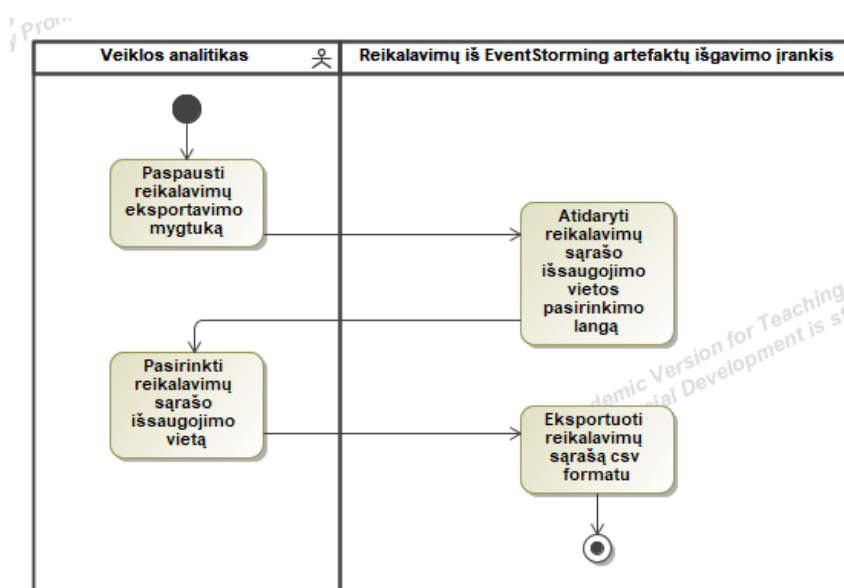
2.4 pav. yra pavaizduota reikalavimų valdymo veiklos diagrama.



2.4 pav. „Valdyti reikalavimus“ panaudojimo atvejo veiklos diagrama

Veiklos analitikas taip pat turi galimybę valdyti reikalavimus. Tai jis padaryti gali atsidaręs reikalavimų peržiūros langą, kuriame gali tiek kurti, tiek redaguoti, tiek šalinti reikalavimą. Pasirinkęs reikalavimo kūrimą, jam yra pateikiamas kūrimo langas. Jame naudotojas gali įvesti reikalavimo duomenis. Paspaudus išsaugojimo mygtuką, patikrinamas ar visi formos duomenys yra užpildyti. Jei užpildyta viskas – sukuriamas reikalavimas. Jei ne – pateikiamas klaidos pranešimas. Veiklos analitikas taip pat turi galimybę paspausti reikalavimo redagavimo mygtuką. Tokiu atveju kūrimo lange duomenų įvedimo formos laukeliai yra užpildomi redaguojamu reikalavimu ir naudotojas įvykdo anksčiau aptartus reikalavimo kūrimo veiksmus, kurių rezultatas – atnaujintas reikalavimas. Be to, vadovas turi galimybę šalinti reikalavimą paspausdamas šalinimo mygtuką. Įrankis pateikia šalinimo patvirtinimo langą, kurį naudotojui patvirtinus reikalavimas yra pašalinamas ir jis nukreipiamas į reikalavimų peržiūros langą.

2.5 pav. yra pateikiama reikalavimų eksportavimo veiklos diagrama

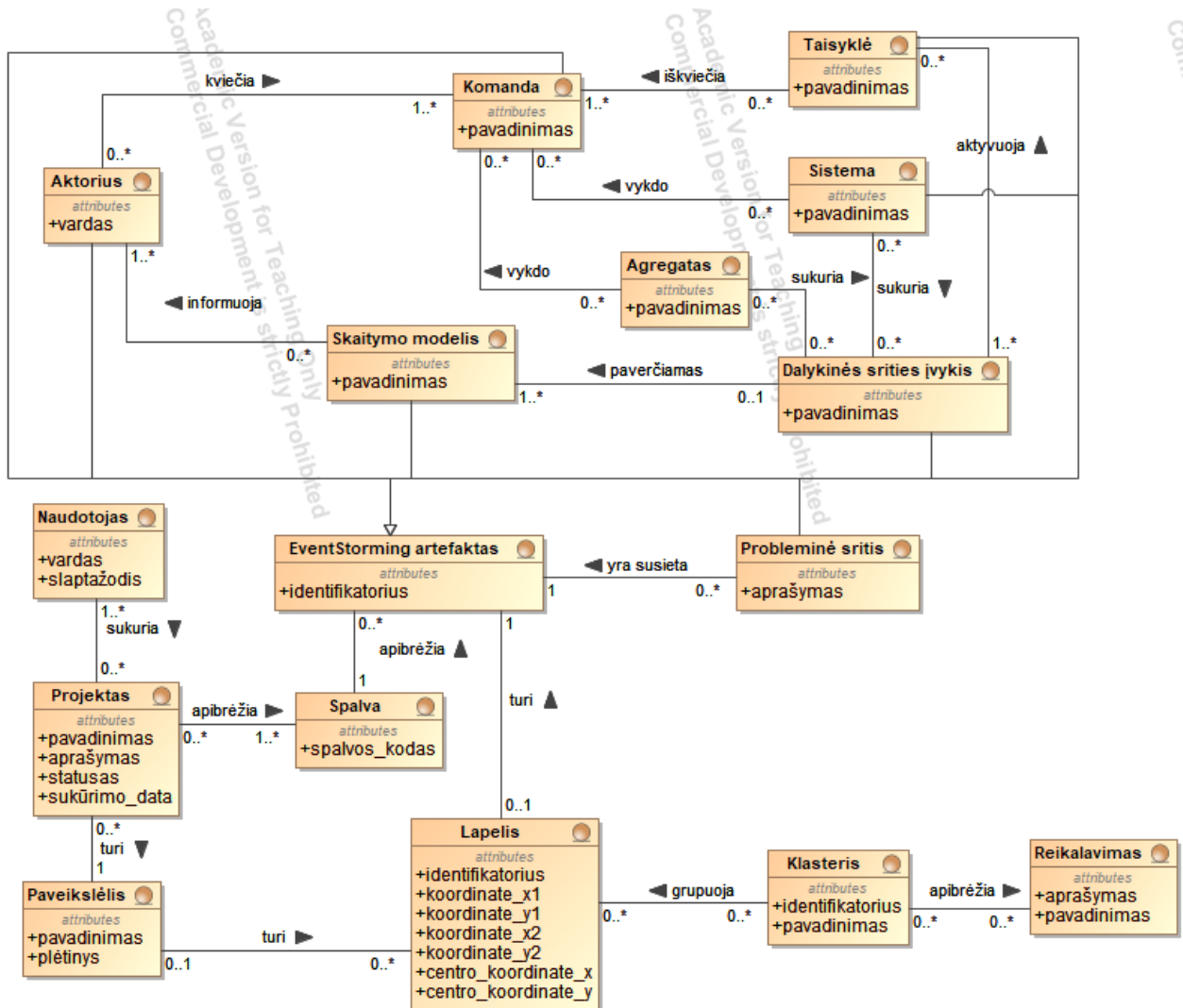


2.5 pav. „Eksportuoti reikalavimus“ panaudojimo atvejo veiklos diagrama

Galiausiai veiklos analitikas turi galimybę eksportuoti reikalavimus. Tokį veiksmą jis atlieką paspausdamas reikalavimų eksportavimo mygtuką, po kurio įrankis atidaro reikalavimų sąrašo išsaugojimo vietos pasirinkimo langą. Naudotojas pasirenka reikalavimų sąrašo išsaugojimo vietą ir įrankis eksportuoja reikalavimų sąrašą csv formatu.

2.2. Dalykinės srities modelis

2.6 pav. pateikta reikalavimų išgavimo iš *EventStorming* artefaktų įrankio esybių klasių diagrama apibūdinanti nagrinėjamos dalykinės srities esybes ir jų tarpusavio ryšius.



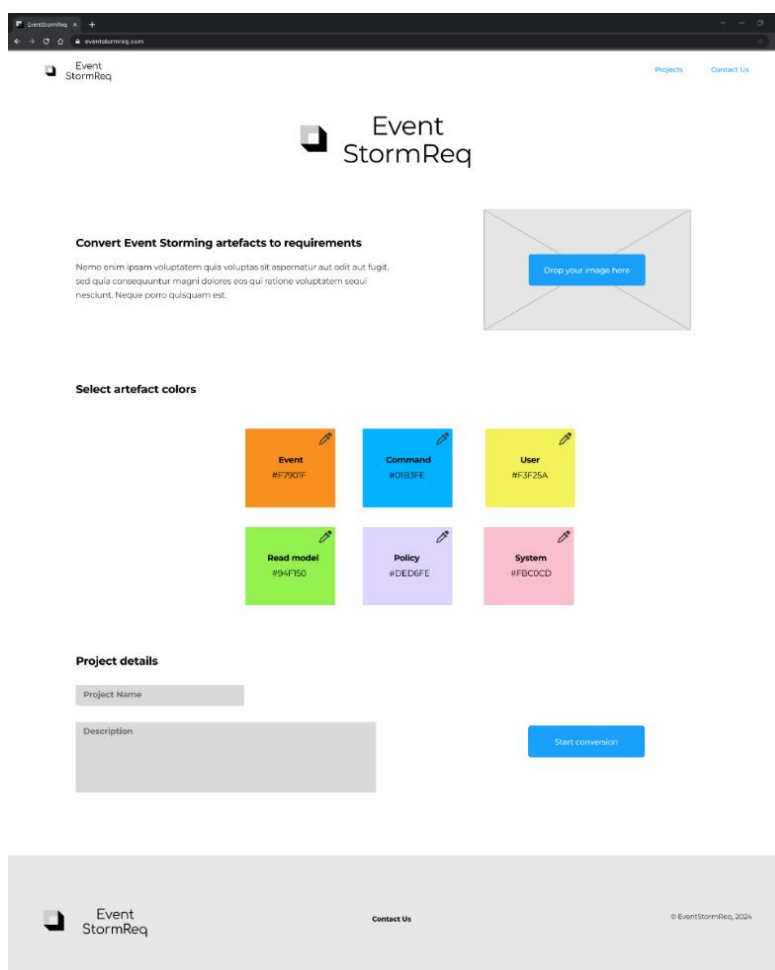
2.6 pav. reikalavimų išgavimo iš *EventStorming* artefaktų įrankio esybių klasių diagrama

Esybių klasių diagramoje vaizduojamas naudotojas, kuris turi galimybę sukurti projektą ir nurodyti jo pavadinimą bei aprašymą. Projektas apibrėžia *EventStorming* artefaktų spalvas spalvos kodais. Šios spalvos bus naudojamos identifikuoti artefaktų tipą paveikslėlyje. Projektas turi *EventStorming* artefaktų paveikslėlį. Jis turi pavadinimą, kuris naudojamas kaip identifikatorius ir plėtinį, kuris nusako kokio formato paveikslėlis tai yra. Įrankiui įvykdžius paveikslėlio analizę, paveikslėlis turi atpažintus lapelius, kurio koordinatės apibrėžia paveikslėlio regioną. Šis lapelis, kartu su spalva ir jos reikšmę projekte yra *EventStorming* artefaktas. Vėliau lapeliai pagal atstumą vienas nuo kito yra sugrupuojami į klasterius, kurie nusako giminingų *EventStorming* artefaktų grupę, kurie ir apibrėžia atitinkamus nagrinėjamos sistemos reikalavimus. Reikalavimas yra išreikštas naudotojo pasakojimo formatu. Taip pat esybių klasių diagramoje apibūdinamos pagrindinės *EventStorming* dalykinės srities sąvokos. Aktorius – vienas iš *EventStorming* artefaktų apibūdinamas vardo atributu gali nuspręsti iškviešti komandą, t.y. kažkokį veiksmą

nagrinėjamoje srityje. Komanda yra vykdoma atitinkamos išorinės sistemos arba agregato vidinėje sistemoje. Šie gavę komandą sukuria dalykinės srities įvykius ir aktyvuoja taisykles, kurios gali iškviešti keletą naujų komandų. Vėliau dalykinės srities įvykis paverčiamas skaitymo modeliu, tokia esybe, kuri laiko nagrinėjamos srities informaciją. Šis skaitymo modelis informuoja aktorių, kuris remiantis šia informacija vėl gali kviešti norimą komandą. Galiausiai egzistuoja aktyvioji sritis, tokia esybė, kuri leidžia susieti papildomą informaciją su bet kuriuo kitu *EventStorming* artefaktu. Visos šios minėtos esybės yra *EventStorming* artefaktai, todėl esybių klasių diagramoje jos susietos paveldėjimo ryšiu.

2.3. Naudotojų sąsajos modelis

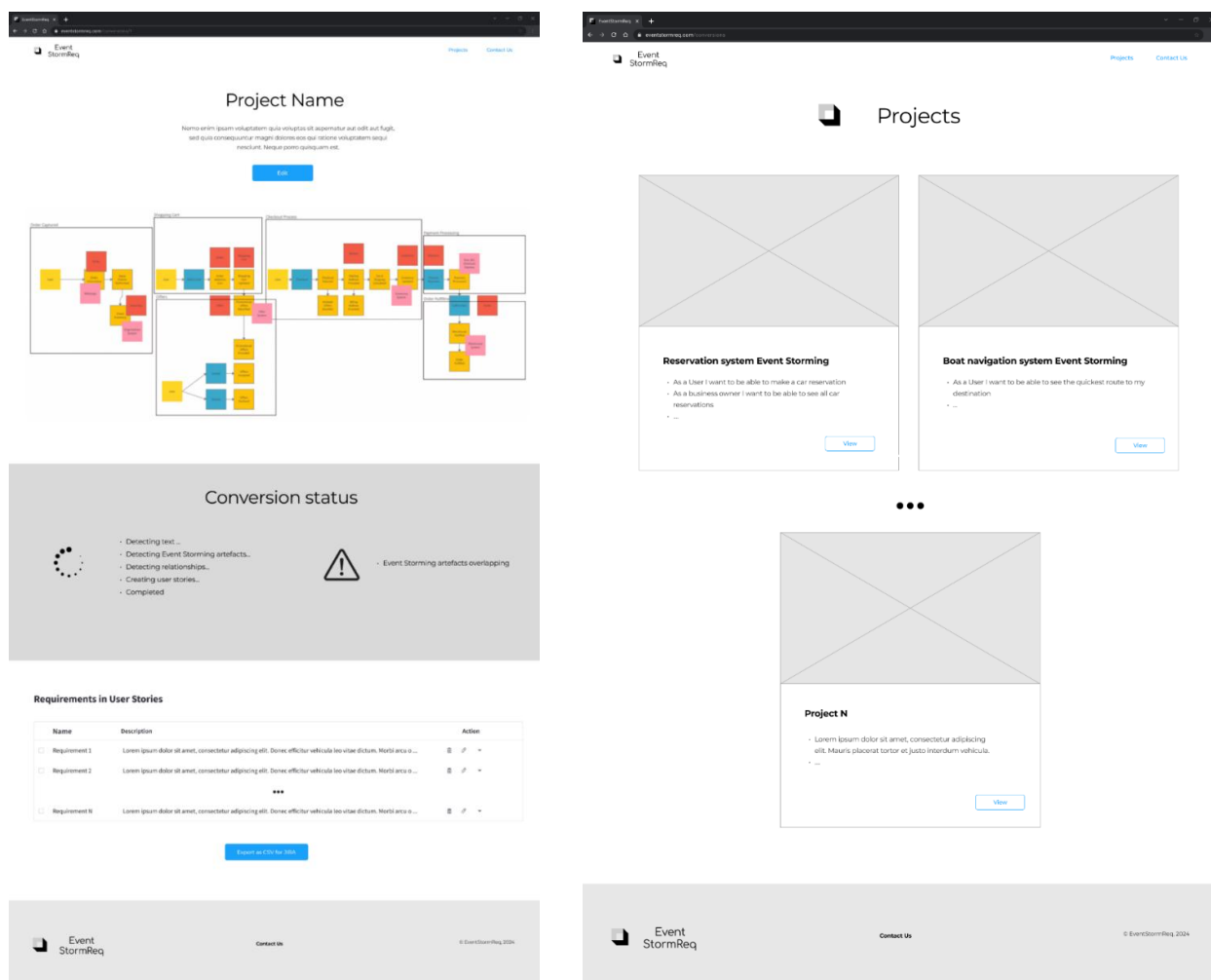
Kuriamas reikalavimų išgavimo iš *EventStorming* įrankis turi tris naudotojo sąsajos langus: pagrindinį projekto kūrimo langą, projektų peržiūros langą ir projekto bei su juo susijusių reikalavimų valdymo langą.



2.7 pav. Projekto kūrimo lango eskizas

2.7 pav. pateiktas pagrindinis projekto kūrimo lango eskizas. Jo viršuje matome navigacinį meniu, kurio pagalba naudotojas gali perjungti skirtingus langus. Žemiau naudotojas mygtuko pagalba gali įkelti savo *EventStorming* artefaktų paveikslėlį bei nurodyti paveikslėlyje naudojamų artefaktų spalvų kodus. Ši spalvų kodų konfigūracija bus naudojama artefaktų tipams paveikslėlyje atpažinti. Taip pat naudotojas turi nurodyti projekto pavadinimą ir jo aprašymą. Ši informacija bus naudojama kontekstui apibrėžti prieš generuojant reikalavimus iš nurodyto *EventStorming* artefaktų paveikslėlio. Apačioje

paspaudus projekto konvertavimo mygtuką naudotojas yra nukeliamas į projekto bei su juo susijusių reikalavimų valdymo langą pavaizduotą 2.8 pav. kairėje.



2.8 pav. Projekto ir reikalavimų valdymo lango eskizas (kairė) ir projektų sąrašo lango eskizas (dešinė)

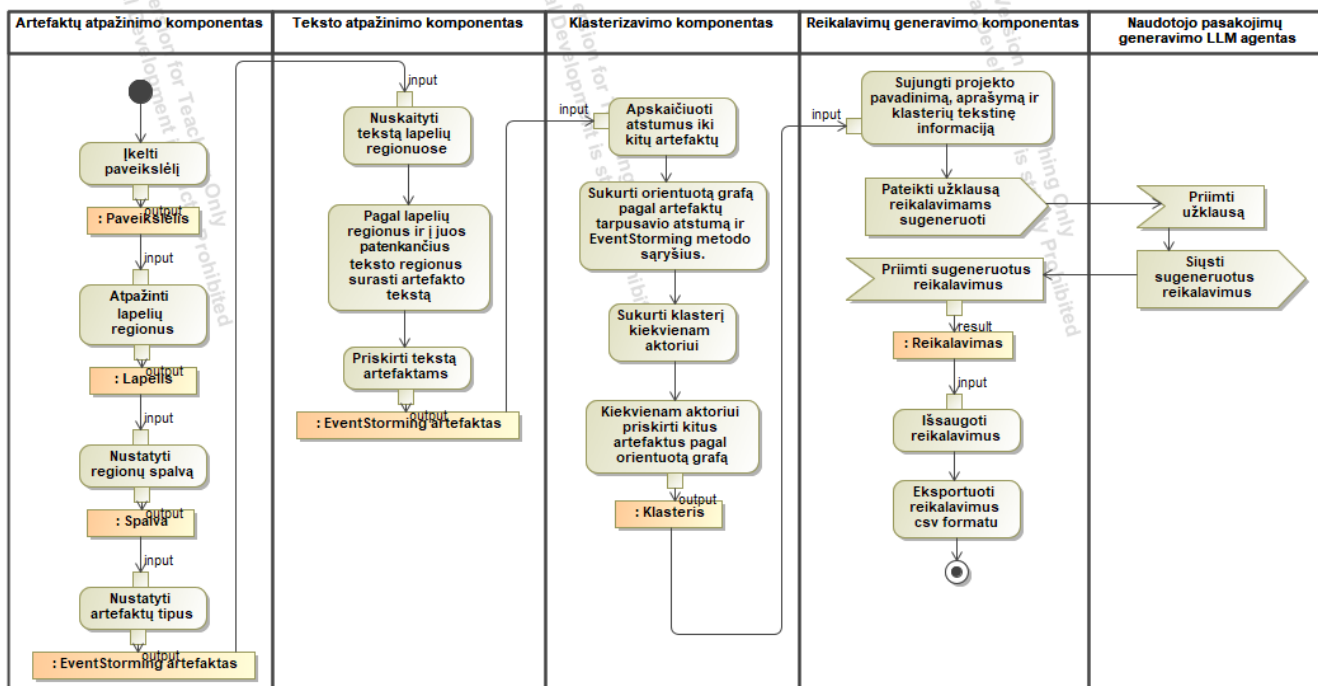
2.8 pav. kairėje vaizduoja projekto ir reikalavimų valdymo lango eskizą. Lango viršuje naudotojas gali matyti sukurtą projekto pavadinimą ir jo aprašymą. Šis turi galimybę jį redaguoti – tokiu atveju reikalavimai bus pergeneruoti naudojant naują projekto kontekstą. Taip pat naudotojas turi galimybę matyti įkeltą *EventStorming* artefaktų paveikslą. Po paveikslėliu yra pateikiama įrankio dabartinė ir įvykdytos būsenos. Be to naudotojas yra informuojamas jei reikalavimų generavimo metu atsitiko nenumatytų klaidų. Generavimui pasiekus užbaigtą statusą naudotojas gali matyti sugeneruotų reikalavimų sąrašą lentelės pavidalu. Reikalavimai yra generuojami naudotojo pasakojimų formatu. Naudotojas turi galimybę reikalavimus trinti, redaguoti bei išplėsti patogesniai skaitymui. Galiausiai naudotojui suteikiama galimybę eksportuoti galutinį reikalavimų sąrašą CSV formatu Jira įrankiui.

Taip pat 2.8 pav. dešinėje vaizduojamas projektų peržiūros lango eskizas. Šiame lange naudotojas turi galimybę matyti visus jo sukurtus projektus ir kiekvieną jų individualiai peržiūrėti. Projekto peržiūros mygtukas nukreipia naudotoją į projekto ir reikalavimų valdymo langą, kurio eskizas matomas 2.8 pav. kairėje.

2.4. Formalus sprendimo aprašas

2.9 pav. yra pavaizduoti pagrindiniai reikalavimų išgavimo iš *EventStorming* artefaktų įrankio komponentai ir reikalavimų išgavimo iš paveikslėlio įprasta seka. Pagrindiniai įrankio komponentai yra:

- **Artefaktų atpažinimo komponentas** – paveikslėlyje nustato lapelių regionus ir pagal naudotojo nurodytą spalvų konfigūraciją nustato *EventStorming* artefakto tipą ir jį priskiria prie atitinkamo regiono.
- **Teksto atpažinimo komponentas** – atpažįsta regionuose esantį tekstą ir priskiria jį *EventStorming* artefaktui
- **Klasterizavimo komponentas** – naudojant Euklidinį artefaktų tarpusavio atstumą ir *EventStorming* metode apibrėžtus ryšius tarp artefaktų juos klasterizuoja.
- **Reikalavimų generavimo komponentas** – sujungia projekto aprašymą ir klasterio tekstinę informaciją bei formuoja užklausą į naudotojo pasakojimų generavimo *LLM* agentą dėl reikalavimų generavimo. Vėliau sugeneruotus reikalavimus transformuoja į .csv formatą.
- **Naudotojo pasakojimų generavimo *LLM* agentas** generuoja reikalavimus naudotojo pasakojimų formatu iš jam perduotos tekstinės projekto informacijos bei suklasterizuotų artefaktų. Agentas instrukcijomis pritaikytas suprasti *EventStorming* metodą.



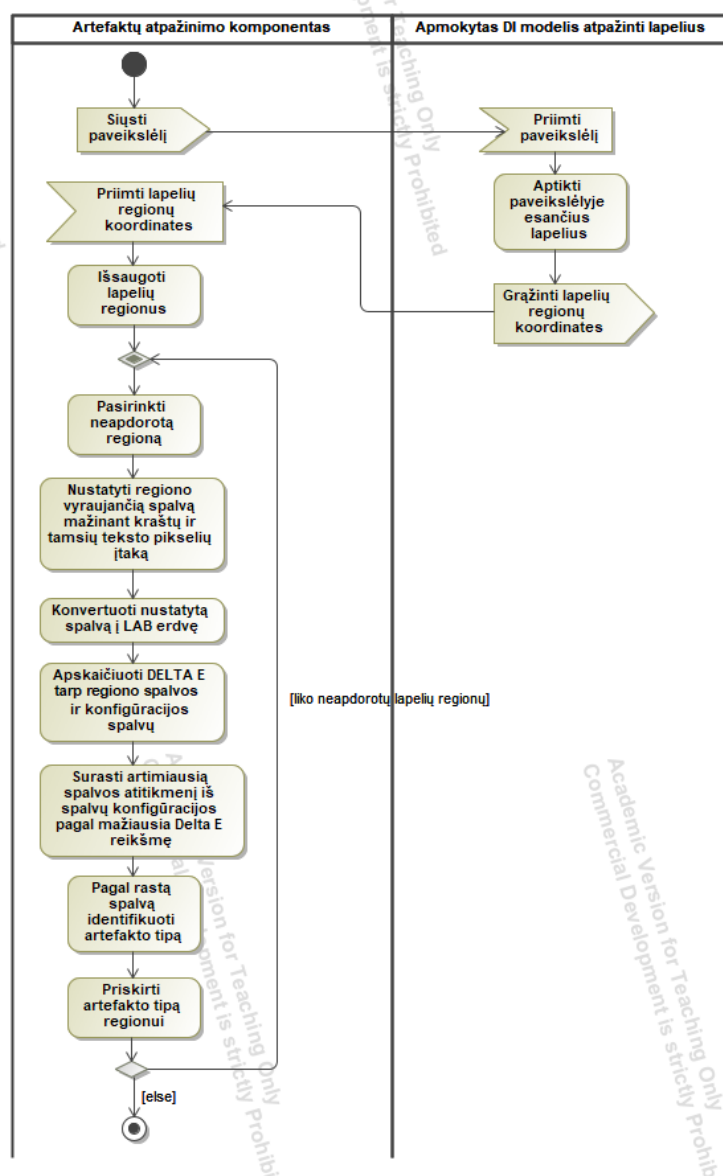
2.9 pav. Reikalavimų išgavimo iš *EventStorming* artefaktų įrankio veiklos diagrama

Reikalavimų išgavimas prasideda nuo artefaktų atpažinimo komponento, kuris paveikslėlyje atpažįsta lapelius ir suranda juos apibrėžiančius regionus bei remiantis spalvų konfigūracija priskiria jiems atitinkamus *EventStorming* artefaktus. Vėliau teksto atpažinimo komponentas, kiekvienam artefaktui nuskaito paveikslėlyje esantį tekstą, kurį išsaugo artefakto esybėje. Toliau klasterizavimo komponentas naudodamas tekstinį ir Euklidinį

atstumus tarp skirtingų regionų sukuria orientuotą grafą ir pradedant nuo aktorių artefaktų pagal orientuotą grafą sukuria klasterius bei pateikia klasterių masyvą į sekantį komponentą. Toliau reikalavimų generavimo komponentas prie kiekvieno klasterio prideda projekto kontekstinę informaciją, jo pavadinimą bei aprašymą bei siunčia reikalavimų generavimo užklausą instrukcijomis pritaikytam naudotojo pasakojimų generavimo *LLM* agentui. Gavus atsakymą – reikalavimas yra išsaugojamas. Galiausiai reikalavimų eksportavimo komponentas eksportuoja sugeneruotus reikalavimus .csv formatu. Toliau detalizuojami kiekvieno komponento veikimo principai.

2.5. Artefaktų atpažinimo komponentas

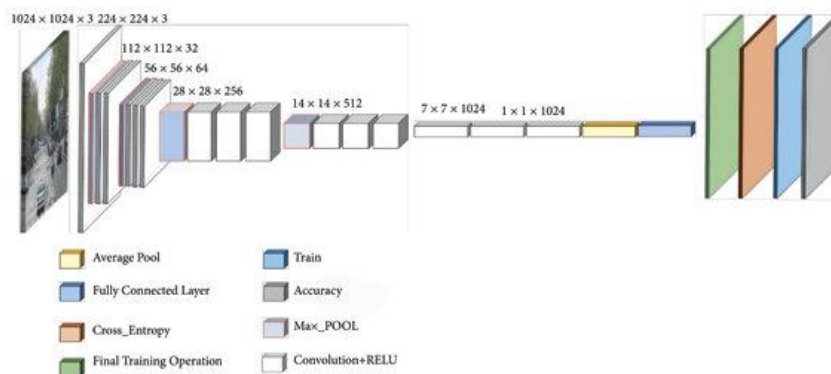
Artefaktų atpažinimo komponente pradedamas *EventStorming* paveikslo skaitmeninimas, šios veiklos diagrama yra matoma 2.10 pav.



2.10 pav. Artefaktų tipo atpažinimo proceso veiklos diagrama

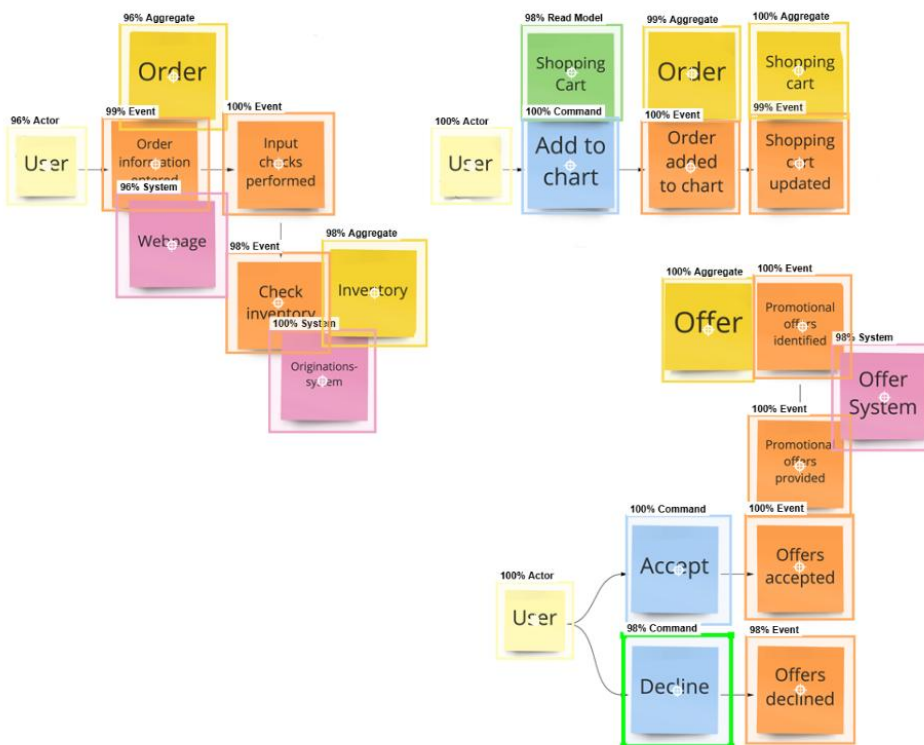
Visų pirma paveikslėlis su *EventStorming* artefaktais yra siunčiamas į apmokytą dirbtinio intelekto modelį, kuris sugeba atpažinti lapelius. Lapelio atpažinimui yra naudojamas

apmokytas *MobileNet V1* konvoliucinis neuroninis tinklas, kurio architektūra pateikiama 2.11 pav. Jo architektūra sudaryta iš konvoliucinių sluoksnių, aktyvacijos funkcijų ir telkimo sluoksnių, skirtų palaipsniui išgauti vis sudėtingesnius vaizdo požymius. Ankstyvieji sluoksniai išryškina žemo lygio požymius, tokius kaip kraštai, kampai ir tekstūros, o gilesni sluoksniai leidžia atpažinti sudėtingesnes formas, pavyzdžiui, lipnių lapelių kontūrus ir jų išsiskyrimą fone. *ReLU* aktyvacijos funkcijos padeda išryškinti reikšmingus požymius, o telkimo sluoksniai mažina požymių matmenis bei padeda išlaikyti svarbiausią informaciją. Dėl tokios architektūros modelis gali efektyviai apdoroti vaizdą ir perduoti išgautus požymius tolimesniams lapelių regionų nustatymui.



2.11 pav. *MobileNet V1* konvoliucinio tinklo architektūra [57]

Atpažinus lapelius apmokytas dirbtinio intelekto modelis grąžina lapelius apibrėžiančius regionus, kuriuos artefaktų atpažinimo komponentas išsaugo. Atpažinti lapeliai ir pažymėti regionai pateikiami 2.12 pav.



2.12 pav. Artefaktų atpažinimas *EventStorming* paveiksle

Aptikus lapelius, kiekvienam aptiktam lapelio regionui apskaičiuojama dominuojanti fono spalva. Spalva nustatoma ne iš pavienių pikselių, bet iš viso lapelio vidinio regiono: kraštų pikseliams suteikiamas mažesnis svoris, centrinės srities pikseliams didesnis, o labai tamsūs pikseliai atmetami, kad ranka rašytas tekstas kuo mažiau veiktų rezultata. Tokiu būdu išskiriama labiausiai tikėtina lapelio spalva. Toliau ši išgauta spalva lyginama su naudotojo pateikta artefaktų spalvų konfigūracija (2.7 pav.). Palyginimas atliekamas *LAB* spalvų erdvėje, nes ši spalvų erdvė geriau atitinka žmogaus regimąjį spalvų suvokimą nei įprasta *RGB* erdvė: vienodi spalvų skirtumai *LAB* erdvėje labiau atspindi tai, kaip panašumą tarp spalvų suvokia žmogus. Dėl šios priežasties artefakto spalvos atitiktis paieškai apskaičiuojamas spalvų skirtumas pagal *Delta E* metriką, kuri įvertina dviejų spalvų artumą. Kuo mažesnė *Delta E* reikšmė, tuo spalvos laikomos panašesnėmis žmogaus akiai. Toks metodas leidžia patikimiau priskirti artefakto tipą net ir tais atvejais, kai dėl apšvietimo, šešėlių, fotografavimo kokybės ar nedidelių spalvos variacijų aptikto lapelio spalva tiksliai nesutampa su naudotojo nurodyta spalva. Pagal mažiausią apskaičiuotą *Delta E* reikšmę parenkama artimiausia konfigūruota spalva, ir pagal ją regionui priskiriamas artefakto tipas. Po lipnių lapelių aptikimo ir artefakto tipo priskyrimo vykdomas teksto atpažinimas.

2.6. Teksto atpažinimo komponentas

Teksto atpažinimo komponente yra atpažįstamas tekstas pateiktuose regionuose. Kiekvienas regionas turi keturias jam priskirtas koordinatas, pavyzdžiui *Offers accepted* teksto regionas turi (8; 223), (150; 223), (150; 200), (8; 200) koordinatas matomas 2.13 pav. kairėje. Kur kiekvienos poros pirmasis skaičius rodo *X*, o antrasis *Y* koordinatę.



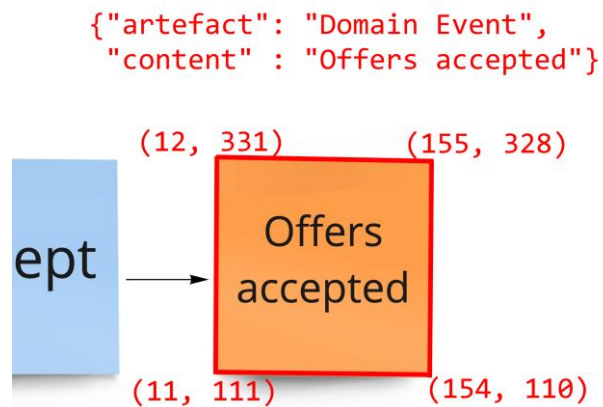
```
[
  {
    "description": "Offers accepted",
    "boundingPoly": {
      "vertices": [
        {"x": 8, "y": 223},
        {"x": 150, "y": 223},
        {"x": 150, "y": 200},
        {"x": 8, "y": 200}
      ]
    }
  }
]
```

2.13 pav. teksto regiono *Offers accepted* atpažinimas ir koordinatės (kairėje) bei OCR įrankio atsakymo fragmentas (dešinėje)

Paveikslėlis yra perduodamas *Google Vision API*. API grąžina teksto eilutes bei juos apibrėžiančius regionus. *Google Vision API* įrankio atsakymo fragmentas pateiktas 2.13 pav. dešinėje, vaizduojantis anksčiau nagrinėto *Offers accepted* skaitmenizuotą tekstą bei jį apibrėžiančio regiono koordinatas.

Apdorojant kiekvieną teksto regioną yra tikrinama ar teksto regionas patenka į aptiktą lapelio regioną, jei taip – tas tekstas priskiriamas lapeliui. Galutiniame rezultate teksto atpažinimo

komponentas priskiria atpažintą tekstą prie kiekvieno artefakto. Yra gaunama duomenų struktūra vaizduojama 2.14 pav.

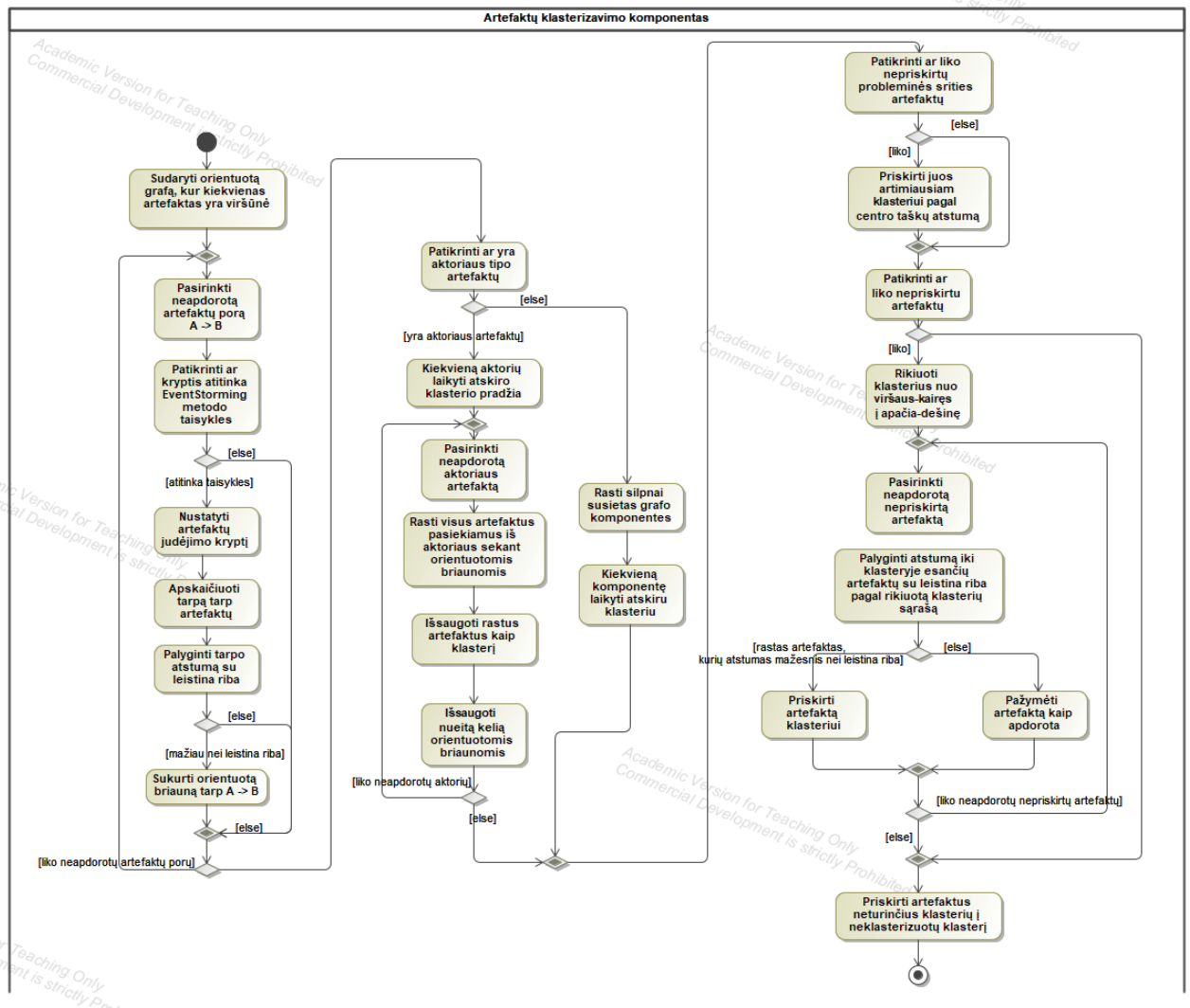


2.14 pav. *Offers accepted* artefaktas su jam priskirtu tekstu.

Duomenų struktūroje saugomas artefakto tipas, teksto turinys ir regioną apibrėžiančios koordinatės. Ši informacija bus vėliau naudojama klasterizavimui bei reikalavimų generavimui.

2.7. Klasterizavimo komponentas

Artefaktų klasterizavimo procesas yra pateiktas 2.15 pav.



2.15 pav. Artefaktų klasterizavimo proceso veiklos diagrama

Klasterizavimas pradamas nuo orientuoto grafo sudarymo, kuriame kiekvienas artefaktas laikomas atskira viršūnė. Toliau nuosekliai analizuojamos visos galimos artefaktų poros (A,B), siekiant nustatyti, ar tarp jų gali egzistuoti prasmingas ryšys pagal *EventStorming* metodą. Pirmiausia tikrinama, ar tokia artefaktų tipų seka apskritai yra galima, pagal 1.9 pav. Jei pora neatitinka šių taisyklių, ryšys tarp artefaktų nėra sudaromas. Jei artefaktų tipų sąryšis yra leistinas, tuomet atliekama jų erdvinė analizė. Šiame žingsnyje nustatoma, ar judėjimas tarp artefaktų yra daugiausia horizontalus, ar vertikalus. Horizontalios eigos atveju tikrinama, ar antras artefaktas yra pakankamai į dešinę pirmojo atžvilgiu, o vertikalios eigos atveju – ar jis yra pakankamai žemiau. Atstumas skaičiuojamas ne tarp centrinių taškų, o tarp artefaktų regionų ribų, nes toks būdas tiksliau atspindi realų objektų išsidėstymą paveiksle. Gautas tarpas lyginamas su leistina riba, kuri priklauso nuo artefakto matmenų: horizontalios eigos atveju leidžiamas ne didesnis kaip 0,5 artefakto pločio tarpas, o vertikalios eigos atveju – ne didesnis kaip 0,3 artefakto aukščio tarpas. Tik tada, kai artefaktų

tipų seka yra leistina ir jų tarpusavio padėtis tenkina erdvines sąlygas, grafe sukuriama orientuota briauna (A,B).

Sudarius orientuotą grafa, pereinama prie klasterių formavimo. Jei artefaktų rinkinyje yra aktoriaus tipo artefaktų, kiekvienas toks artefaktas laikomas atskiro klasterio pradžia. Nuo pasirinkto aktoriaus, sekant orientuotomis grafo briaunomis, randami visi pasiekiami artefaktai. Visi šie artefaktai priskiriami tam pačiam klasteriui. Be pačių artefaktų sąrašo papildomai išsaugomi ir nueitas kelias, kuris parodo, kokia seka pradėdant nuo aktoriaus artefaktai yra susiję tarpusavyje. Tokiu būdu klasteris atspindi ne tik artefaktų grupę, bet ir loginę proceso eigą.

Jei nagrinėjame artefaktų rinkinyje nėra nė vieno aktoriaus tipo artefakto, taikoma alternatyvi strategija. Tokiu atveju orientuotas grafas analizuojamas pagal tarpusavyje susijusias viršūnių grupes, neakcentuojant vieno pradžios taško, t.y. pagal silpnai susietas grafo komponentes. Kiekviena tokia tarpusavyje susijusių artefaktų grupė laikoma atskiru klasteriu.

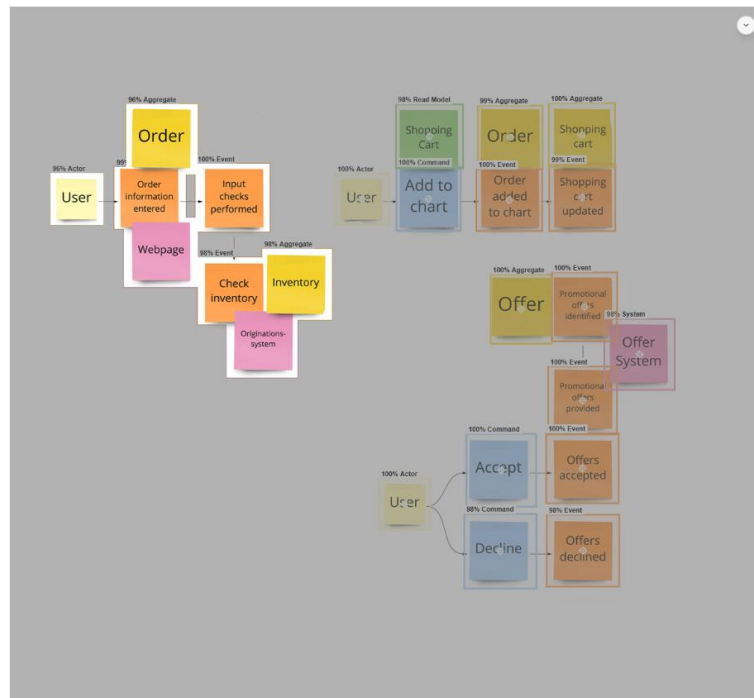
Po pirminio klasterių sudarymo atliekamas nepriskirtų artefaktų apdorojimas. Pirmiausia tikrinama, ar liko nepriskirtų probleminės srities tipo artefaktų. Tokiems artefaktams taikoma speciali taisyklė: jie priskiriami artimiausiam klasteriui pagal centro taškų tarpusavio atstumą. Tai leidžia įtraukti klausimus, pastebėjimus prie artimiausio proceso konteksto, kadangi jis nepatenka į pagrindinį srautą.

Toliau tikrinama, ar yra kitų nepriskirtų artefaktų. Jei tokių yra, klasteriai surikiuojami pagal jų padėtį nuo viršutinės kairės paveikslo dalies link apatinės dešinės. Tada kiekvienas nepriskirtas artefaktas lyginamas su jau esančiais klasterių artefaktais, vertinant jų tarpusavio artumą. Artefaktas priskiriamas pirmajam klasteriui, kuriam jis yra pakankamai artimas pagal naudojamas horizontalias arba vertikalias atstumo ribas. Tokiu būdu papildomi artefaktai gali būti įtraukti į artimiausią prasmingą grupę net tada, kai jų nepavyko įtraukti į pradinį grafo srautą.

Jei ir po šio etapo vis dar lieka nepriskirtų artefaktų, sukuriamas atskiras nesuklasterizuotų artefaktų klasteris. Į jį perkeliama visi artefaktai, kurių nepavyko susieti nei su vienu jau sudarytu klasteriu. Tai leidžia užtikrinti, kad nė vienas aptiktas artefaktas nebūtų prarastas galutiniam rezultate.

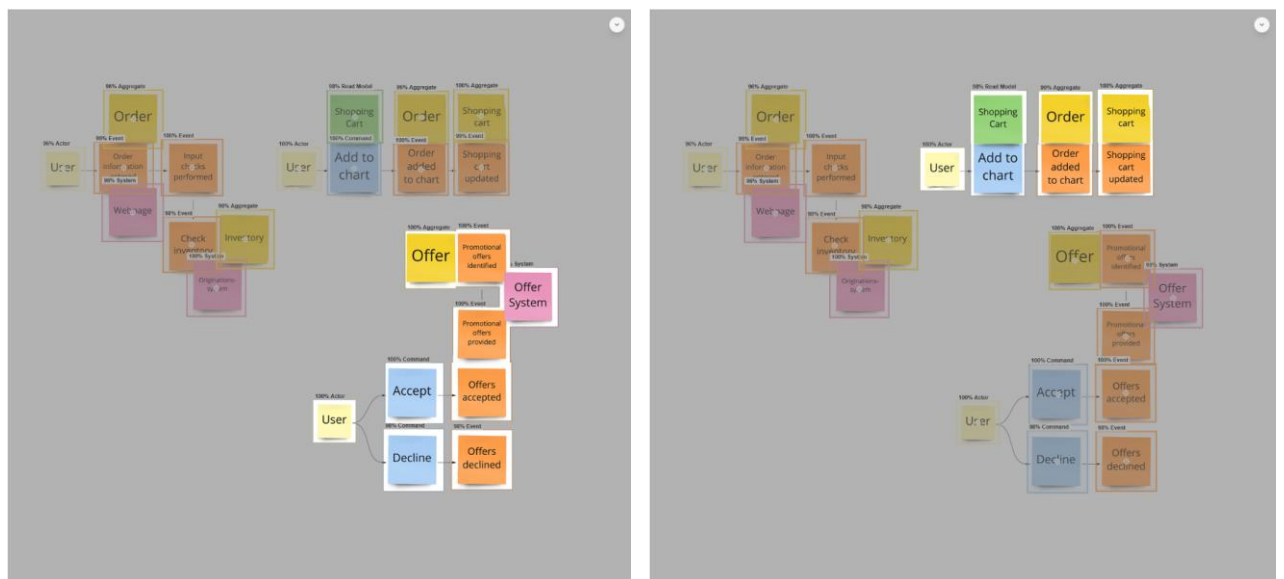
Galutinis klasterizavimo proceso rezultatas yra klasterių sąrašas, kuriame kiekvienas klasteris turi jam priskirtų artefaktų identifikatorių sąrašą bei išsaugotus srauto kelius. Toks sprendimas leidžia grupuoti artefaktus remiantis ne tik jų erdvinio išsidėstymu, bet ir *EventStorming* metode apibrėžta loginės eigos seka. Pavyzdžiui 2.16 pav. vaizduojamas

EventStorming artefaktų paveikslas po klasterizavimo, kuriame yra nuspalvintas užsakymo klasteris



2.16 pav. *EventStorming* paveikslas po klasterizavimo

Analogiškai 2.17 pav. galime matyti kitus šio *EventStorming* paveikslų klasterius.

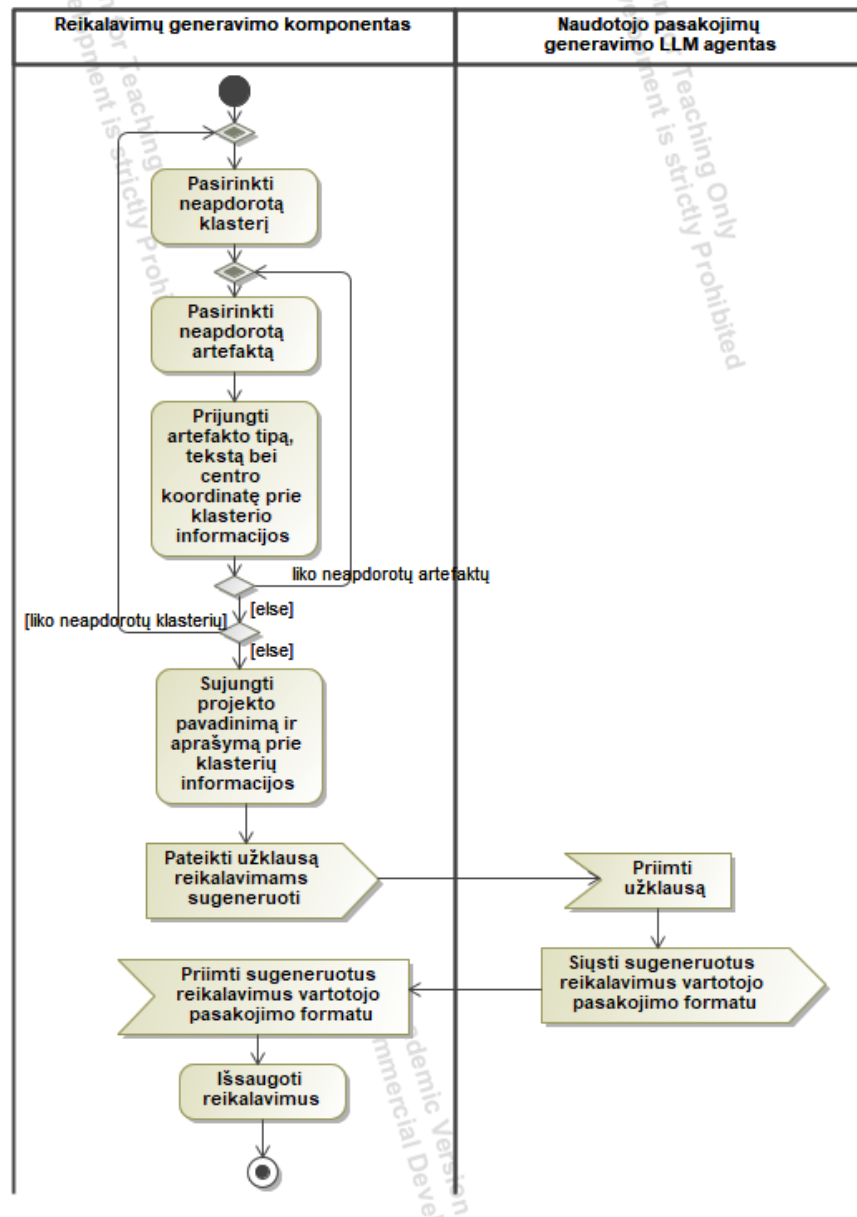


2.17 pav. Reklaminių pasiūlymų klasteris (kairėje) bei įdėjimo į krepšelį klasteris (dešinėje)

Šiame paveikslėlyje matoma, kaip tarpusavyje logiškai susiję artefaktai gali būti suskirstomi į atskirus klasterius pagal proceso dalis. Iš paveikslėlio galima išskirti bent tris pagrindinius klasterius: užsakymo pateikimo (*Order*), prekės įdėjimo į krepšelį (*Add to cart*) ir reklaminių pasiūlymų apdorojimo (*Offer*) klasterius. Toks grupavimas iliustruoja, kad *EventStorming* artefaktai gali būti jungiami į logiškai vienisias proceso dalis, kurios vėliau gali būti naudojamos kaip pagrindas tolimesniam reikalavimų generavimui.

2.8. Reikalavimų generavimo komponentas – generavimas

Po klasterizavimo yra vykdomas reikalavimų generavimas, kurio veiklos diagrama matoma 2.18 pav.



2.18 pav. Reikalavimų generavimo veiklos diagrama

Visų pirma kiekvienam klasteriui suformuojama klasterio turinio informacija, kurią sudaro klasterio pavadinimas ir jame esančių artefaktų duomenys. Kiekvienas artefaktas aprašomas jo tipu, tekstiniu aprašymu ir centro koordinatėmis. Vėliau prie klasterių informacijos pridedama bendroji projekto kontekstinė informacija, apimanti projekto

pavadinimą ir aprašymą. Klasterių bei projekto kontekstinė informacija yra saugoma JSON tipo duomenų struktūroje matomoje 2.19 pav.

```
{
  "project_name": "Donut Ordering System",
  "project_description": "The system allows the customer to order donuts online,
  pay for the order, and receive an order confirmation.",
  "clusters": [
    {
      "cluster_name": "Customer",
      "artifacts": [
        {
          "type": "Actor",
          "description": "Customer",
          "center_point": {"x": 0.10, "y": 0.22}
        },
        {
          "type": "Command",
          "description": "Place Donut Order",
          "center_point": {"x": 0.28, "y": 0.22}
        },
        {
          "type": "Aggregate",
          "description": "Order",
          "center_point": {"x": 0.45, "y": 0.22}
        },
        {
          "type": "Event",
          "description": "Order Placed",
          "center_point": {"x": 0.62, "y": 0.22}
        },
        {
          "type": "Policy",
          "description": "Send Confirmation Email",
          "center_point": {"x": 0.78, "y": 0.22}
        },
        {
          "type": "ReadModel",
          "description": "Order History",
          "center_point": {"x": 0.82, "y": 0.36}
        }
      ]
    }
  ]
}
```

2.19 pav. Projekto ir klasterių informacija JSON duomenų struktūroje

Ši duomenų struktūra yra siunčiama į naudotojo pasakojimų generavimo *LLM* agentą, kuris, apdorojęs užklausą atsiunčia sugeneruotus reikalavimus naudotojo pasakojimų formatu. Šie reikalavimai yra išsaugojami vėlesniam naudotojo redagavimui.

2.9. Naudotojo pasakojimų generavimo *LLM* agentas

Norint, jog *LLM* agentas suprastų perduotą informaciją apie projektą, jame esančius klasterius ir juose esančių artefaktų reikšmę bei sugebėtų gražinti reikalavimus naudotojo pasakojimų formatu *LLM* agentas turi būti pritaikytas. Pasirinkta naudoti instrukcijomis grįstą *LLM* agento pritaikymą. Instrukcijos apibrėžia *LLM* agento užklausas ir pageidaujamus atsakymus. Dalį instrukcijos galime matyti 2.20 pav., o pilną instrukciją galima rasti 2 priede.

```

1 You are an expert business analyst and Event Storming facilitator specializing in
  converting Event Storming workshops into actionable user stories and requirements.
2
3 Your task is to analyze Event Storming artifacts and generate comprehensive user
  stories following Agile best practices.
4
5 UNDERSTANDING EVENT STORMING ARTIFACTS AND THEIR ROLES:
6
7 1. Actor: The person or role who initiates actions (e.g., 'User', 'Admin', 'Customer')
8   | - Maps to: The 'As a [role]' part of user stories
9   | - Usually leftmost in the flow (starting point)
10
11 2. Command: An intention or action the Actor wants to perform (imperative, e.g.,
   'Place Order', 'Create Account')
12   | - Maps to: The 'I want to [action]' part of user stories
13   | - Triggered by Actors or Policies
14   | - Causes Events to happen
15   | ...
16
17 EVENT STORMING FLOW LOGIC:
18 - Actor → triggers → Command
19 - Command → processed by → System/Aggregate → produces → Event
20 - Event → triggers → Policy
21 - Policy → may trigger → new Command (creating cycles)
22 - Events → update → Read Models (for querying)
23 - Read Models → provide information to → Actor (enabling informed decisions about
   future Commands)
24 - IMPORTANT: Read Models close the feedback loop, allowing Actors to make data-driven
   decisions
25
26 HOW TO GENERATE USER STORIES FROM EVENT STORMING:
27
28 1. IDENTIFY THE ACTOR(S):
29   | - Find the Actor artifact(s) in the cluster (usually leftmost, lowest x-value)
30   | - Each distinct Actor typically needs their own user story
31
32 2. IDENTIFY THE COMMAND(S):
33   | - Find Command artifacts (what the Actor wants to do)
34   | - Commands become the 'I want to [action]' part
35   | - Use imperative form: 'place an order', 'create an account', 'update profile'

```

2.20 pav. Instrukcijos dalys LLM agento pritaikymui dirbti su *EventStorming* metodu

Visų pirma LLM agentui yra pateikiama sisteminė užklausa, kuri nurodo, kad agentas turi veikti kaip reikalavimų inžinerijos ir *EventStorming* ekspertas, interpretuoti artefaktų tipus pagal jų paskirtį procese ir sugeneruoti reikalavimus naudotojo pasakojimų forma. Taip pat joje apibrėžiama, kad atsakymas turi būti gražintas JSON formatu, kuriame pateikiamas sugeneruotų reikalavimų sąrašas. Naudotojo užklausoje pateikiami konkretūs projekto duomenys, kuriais remdamasis modelis generuoja reikalavimus.

Toks instrukcijomis apmokytas LLM agentas geba suprasti pateiktą duomenų struktūrą, klasterių bei *EventStorming* artefaktų reikšmę – bei sugeneruoti reikalavimus naudotojo pasakojimų formatu, kuriame išskiriamas naudotojo pasakojimo pavadinimas, aprašymas, priėmimo kriterijai bei techninės detalės.

Naudotojo pasakojimų generavimo *LLM* agento atsakymas į 2.19 pav. nurodytą užklausą yra pavaizduotas 2.21 pav.

```

{
  "requirements": [
    {
      "name": "Submit Donut Order",
      "description": "As a customer, I want to place a donut order, so that I can buy donuts online.
      Acceptance Criteria:- The customer can submit an order for selected donuts.- The order is
      recorded in the system.- Orders cannot be created without valid order data.Technical Notes:-
      The Order aggregate handles order consistency rules."
    },
    {
      "name": "Receive Order Confirmation",
      "description": "As a customer, I want to receive confirmation of my donut order, so that I
      know my order was placed successfully.Acceptance Criteria:- A confirmation email is sent after
      the order is placed.- The customer can view the placed order in order history.Technical
      Notes:- Confirmation depends on email sending functionality.- Order History read model must be
      updated after order placement."
    }
  ]
}

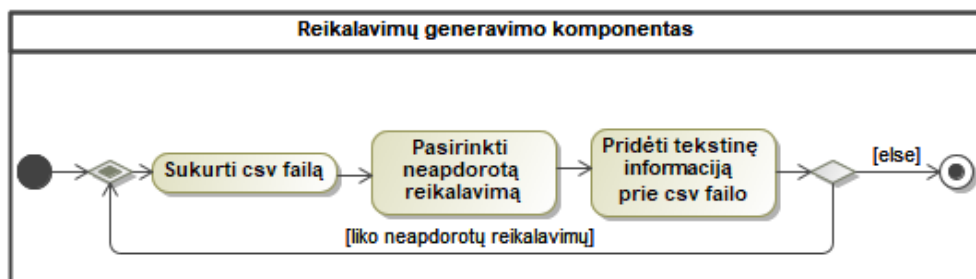
```

2.21 pav. Naudotojo pasakojimų generavimo *LLM* agento atsakymas į 2.18 pav. nurodytą užklausą

Jame matome, jog *LLM* agentas grąžina naudotojo pasakojimų formatu pateiktą reikalavimų sąrašą su priėmimo kriterijais ir techninėmis detalėmis.

2.10. Reikalavimų generavimo komponentas – eksportavimas

Galiausiai išsaugoti reikalavimai gali būti paverčiami į .csv formatą, suprantama *Jira* įrankio. Šis procesas yra matomas 2.22 pav.



2.22 pav. Reikalavimų eksportavimas csv formatu

Eksportuojami reikalavimai į csv failą, turėtų turėti pavadinimą ir aprašymą. Konkretni csv schema suprantama *Jira* įrankio yra pateikta 2.23 pav.

```

1 Summary, Description, Issue Type, Priority
2 "1st User story", "As a user...", Story, Medium

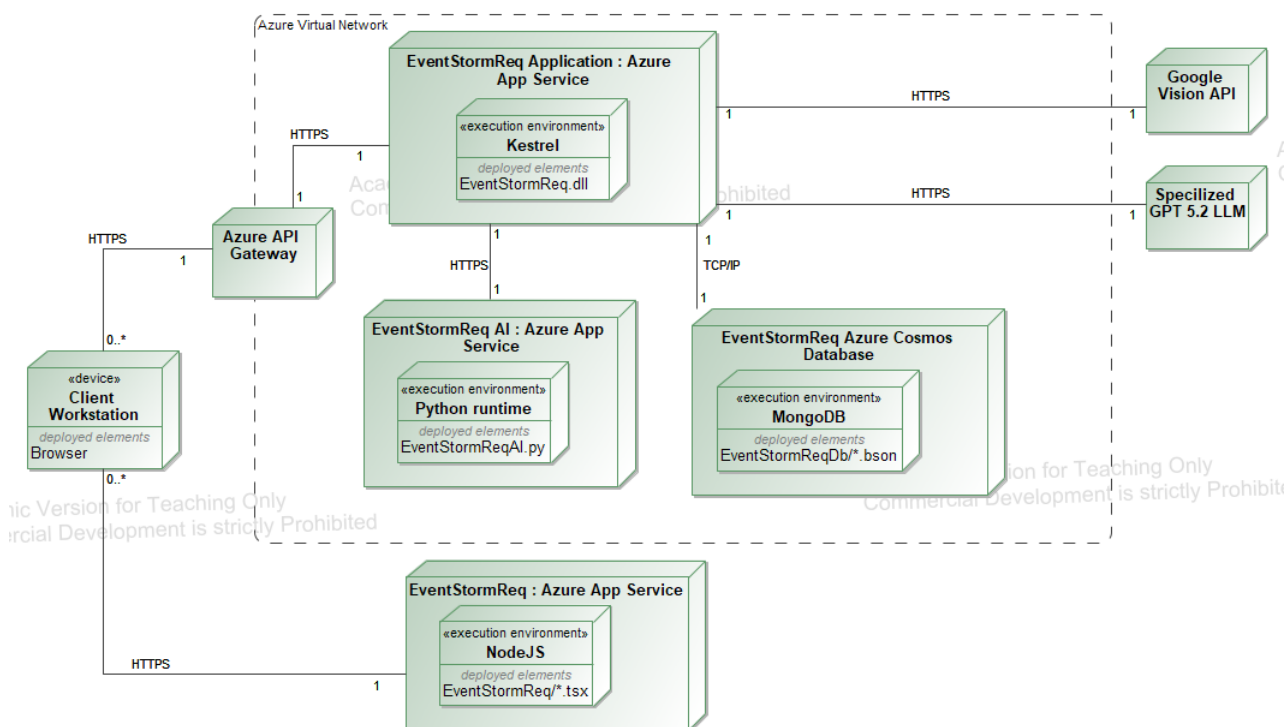
```

2.23 pav. Reikalavimų .csv schema integracijai su *Jira* sistema [58]

Naudotojo pasakojimų išgavimo iš *EventStorming* artefaktų įrankio kontekste pildomi *Summary* ir *Description* laukai. Laukui *Issue Type* visais atvejais priskiriama reikšmė *Story*, nes sistemoje sugeneruoti reikalavimai visada formuojami naudotojo pasakojimų pavidalu. Visiems naudotojo pasakojimams *Priority* priskiriama reikšmė *Medium*, kadangi prioritetai pagal taikomą *EventStorming* metodą šiame etape nėra nustatomi.

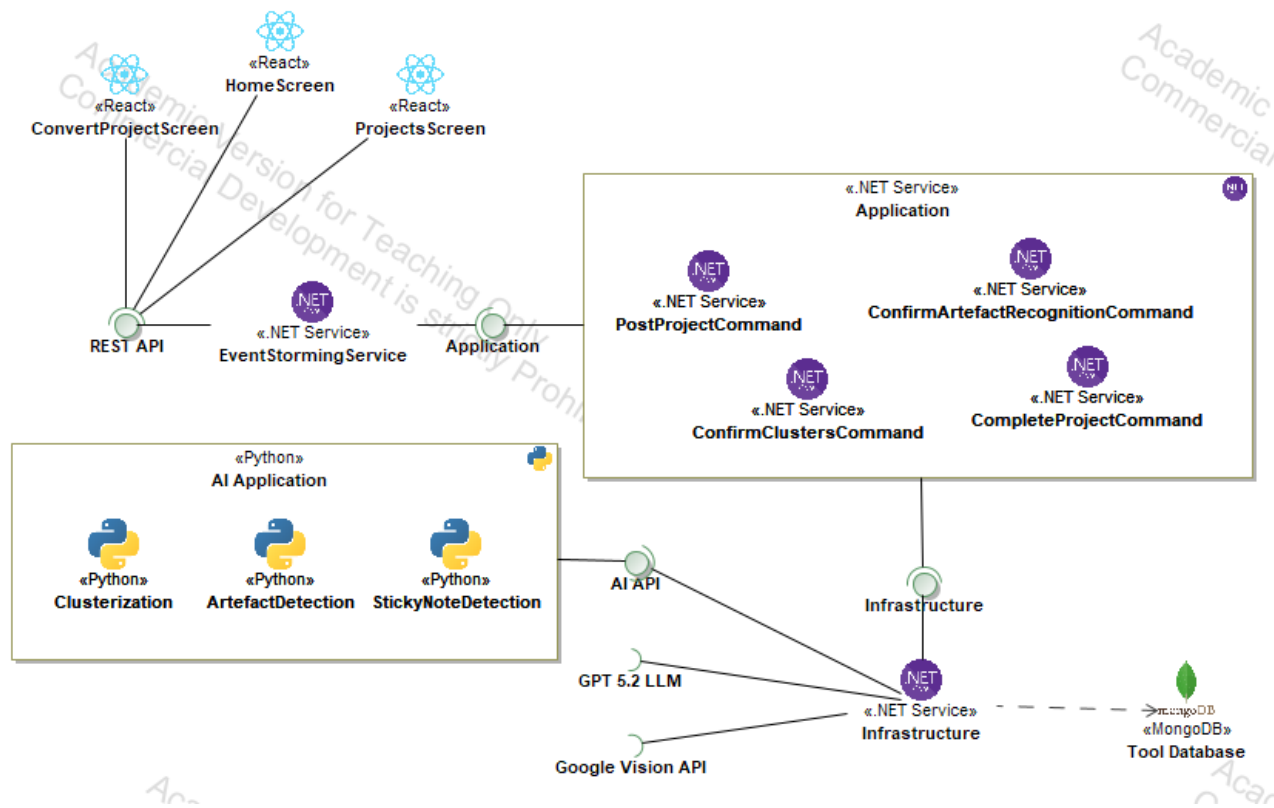
3. Sprendimo realizacijos projektas

3.1 pav. pavaizduota *EventStormReq* reikalavimų išgavimo iš *EventStorming* artefaktų įrankio sistemos diegimo diagrama. Sistema yra talpinama į *Azure* debesų kompiuterijos paslaugą. Sistemos serveris, dirbtinio intelekto modulis skirtas artefaktų atpažinimui bei grafinė naudotojo sąsaja yra talpinami į *Azure App Service* paslaugą, ši suteikia galimybę talpinti įvairią programinę įrangą. Įrankio serveris yra apsaugotas nuo neteisėto išorinio priėjimo panaudojant *Azure API Gateway* paslaugą. Ši paslauga veikia kaip atvirkštinė įgaliotoji stotis (angl. *reverse proxy*), skirta autorizuotai suteikti prieigą prie nurodytų vidinių serverių. Įrankio serverio duomenys yra saugojami naudojant *Azure Cosmos Database*, tai nerealiacinės duomenų bazės paslauga. Sistemos diagramoje matoma ir trečiųjų šalių tinklo paslauga – *Google Vision API*, kurią naudoja įrankio serveris, kai reikia atpažinti tekstą. Taip pat apmokytas dirbtinio intelekto modelis skirtas lapelių atpažinimui bei *Specialized GPT 5.2 LLM* reikalingas reikalavimų generavimui.



3.1 pav. *EventStorm2Req* sistemos diegimo diagrama

3.2 pav. yra pavaizduota įrankio komponentų diagrama. Sistema kurta naudojant *Onion* architektūros principą, matomą 1.4 pav. Pagrindinis sistemos komponentas *Application* skirtas įgyvendinti veiklos analitiko panaudojimo atvejus, matomus 2.1 pav., o *Python* komponentas skirtas klasterizavimo, artefaktų bei lapelių atpažinimo užduotims įgyvendinti. Šie komponentai ir juos sudarantys smulkesni komponentai yra matomi 2.9 pav.



3.2 pav. Įrankio komponentų diagrama

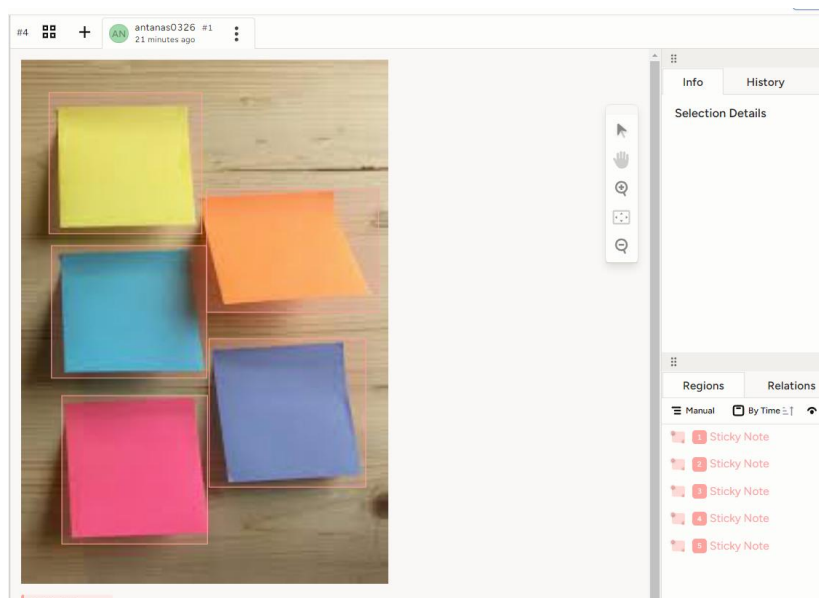
Be anksčiau įvardintų komponentų įrankio veikimui taip pat reikalingi ir *React* komponentai, tokie kaip projekto konvertavimo langas (paveikslėlyje matomas kaip *ConvertProjectScreen*), kurio eskizą galima matyti 2.8 pav., taip pat pagrindinio lango komponentas (*HomeScreen*), jo eskizą galima matyti 2.7 pav. bei projektų lango komponentas (*ProjectScreen*) eskizas.

React komponentai su serverio dalimi bendrauja naudodami *REST API*, kurį įgyvendina *.NET API* komponentas. *API* tiesiogiai kviečia *Application* komponentą, kuris atsakingas už panaudojimo atvejų įgyvendinimą bei reikalingų *Python* funkcijų iškvietimą. Diagramoje matomas *Infrastructure* komponentas atsakingas už *Python* komponento bei trečiųjų šalių sistemų komunikaciją, konkrečiai šiuo atveju *Google Vision API*, *GPT 5.2 LLM* bei *MongoDB* duomenų base.

3.1. Duomenų rinkinio sudarymas lapelių atpažinimo apmokymui

Lapelių atpažinimo komponentui realizuoti buvo sudarytas naujas lapelių vaizdų rinkinys. Vaizdai buvo renkami iš skirtingų viešai prieinamų internetinių šaltinių. Sudarant vaizdų rinkinį buvo naudojami vaizdai, kuriuose lapeliai buvo matomi iš įvairių kryptų, įvairiomis padėtimis, siekiant sudaryti modelį, kuris galėtų tiksliau atpažinti įvairaus tipo, pozicijų

lapelius. 128 (80,5%) vaizdai buvo naudojami apmokymui, o 31 (19,5%), modelio testavimui panaudojant *k-fold* kryžminė patikrą. Vaizdų anotavimui pažymint kurioje vietoje yra lipnus lapelis, buvo naudojama *Label Studio* programinė įranga. Vaizdų anotavimo procesas *Label Studio* programinėje įrangoje matomas 3.3 pav., o 3.4 pav. matomas anotuoto paveikslėlio duomenų failas.



3.3 pav. Vaizdų anotavimas naudojant *Label Studio* programinę įrangą

```

<annotation>
  <folder>images</folder>
  <filename>test1.jpg</filename>
  <path>C:\Users\Antanas\OneDrive\Desktop\object-detection\images\test1.jpg</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>320</width>
    <height>321</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>stickynote</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>19</xmin>
      <ymin>28</ymin>
      <xmax>309</xmax>
      <ymax>303</ymax>
    </bndbox>
  </object>
</annotation>

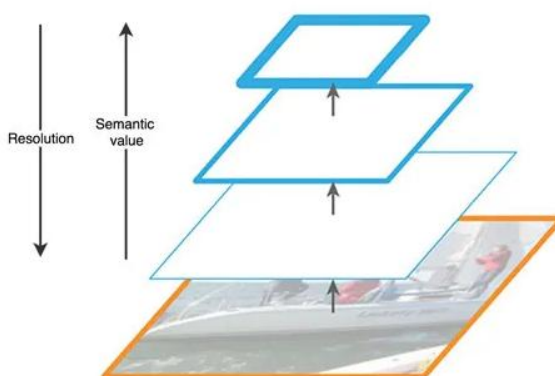
```

3.4 pav. Anotuoto vaizdo duomenys naudojant *Label Studio* programinę įrangą

Anotavimo proceso pabaigoje yra gaunamas *.xml* failas, kuris apibrėžia anotuoto vaizdo metaduomenis bei papildo juos pažymėtų objektų apibrėžiančiomis koordinatėmis

3.2. Lapelių atpažinimo modelio apmokymas

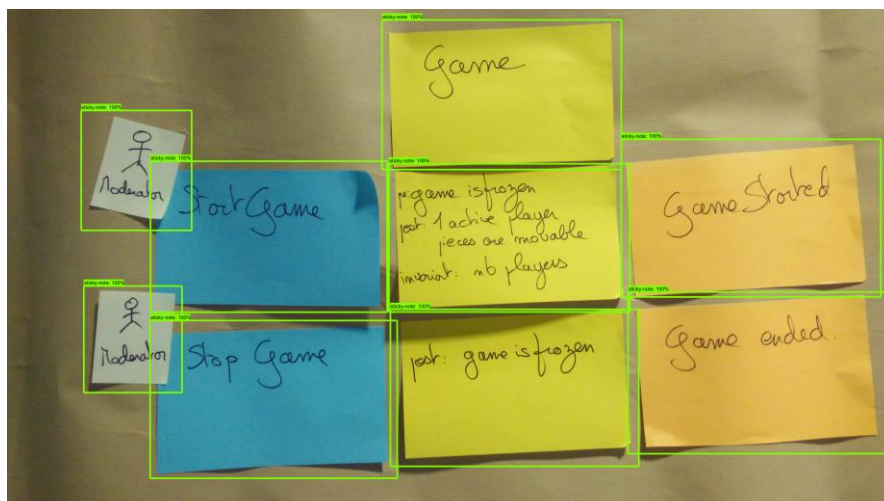
Lapelių atpažinimo modeliui apmokyti buvo panaudota *TensorFlow Object Detection API* biblioteka. Ši *Tensowflow Object Detection* biblioteka suteikia galimybę pasinaudoti iš anksto apmokytais (angl. *pre-trained*) modeliais, kurie buvo apmokyti su didelės apimties vaizdo duomenų rinkiniais. Siekiant atpažinti lapelius iš vaizdų, buvo pasirinktas *SSD MobileNet v1 FPN* iš anksto apmokytas modelis. *MobileNet v1* yra gilusis konvoliucinis tinklas, skirtas objektų aptikimui vaizduose, gražinantis objekto klasifikaciją ir objektą ribojančias koordinatas (angl. *bounding box*). *MobileNet v1* išsiskiria tuo, jog užima nedaug vietos atmintyje ir yra efektyvus, todėl dažnai naudojamas realaus laiko aplikacijose, be to nesudėtinga šio modelio neuroninių tinklų architektūra leidžia apmokyti tokio tipo modelius mažesniu duomenų rinkiniu [59]. Būtent dėl šių savybių *MobileNet v1* buvo pasirinktas kaip bazinė modelio architektūra lapelių atpažinimo modeliui kurti. Modelio apmokymo metu, naudojant mokymo perkėlimą (angl. *transfer learning*), buvo išbandomos įvairios modelio konfigūracijos bei hyperparametrai: aktyvacijos funkcijos, partijos dydžiai (angl. *batch size*), apmokymo žingsniai, reguliarizacija, normalizacija, kol buvo atrasta kombinacija leidžianti pasiekti norimą modelio tikslumą. Prie *MobileNet v1* buvo taip pat pridėtas ir papildomas savybių piramidės tinklas (angl. *feature pyramid network*), kurio architektūra matoma 3.5 pav.



3.5 pav. Savybių piramidės tinklo architektūra [60]

Šis savybių piramidės tinklas suteikia modeliui tikslumo atpažįstant objektus skirtingų rezoliucijų vaizduose [60]. Apmokytas modelis rezultate gražina aptiktus lapelius su aptikimo

patikimumu (angl. *confidence*) bei nurodo aptikto lapelio koordinacių ribas, matomus 3.6 pav.



3.6 pav. Atpažintų lapelių panaudojant apmokytą modelį ribų atvaizdavimas

Apmokytas lapelių atpažinimo modelis buvo ištestas eksperimentiškai 4 skyriuje.

3.3. Testavimo modelis, duomenys, rezultatai

Automatizuoto reikalavimų išgavimo iš *EventStorming* artefaktų įrankiui buvo vykdomas kiekvieno komponento rankinis testavimas.

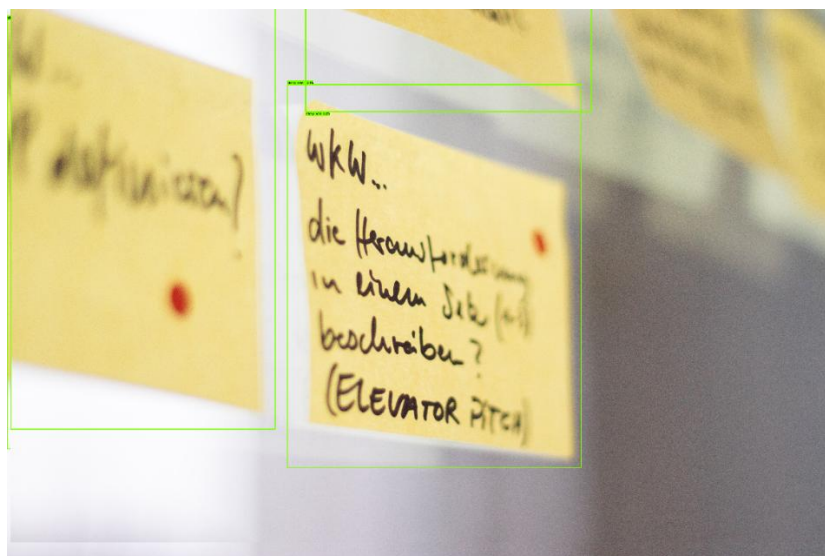
3.3.1. Artefaktų ir lipnių lapelių atpažinimo komponento testavimas

Artefaktų atpažinimo komponento testavimas susidėjo iš dviejų dalių. Pirmą – lipnių lapelių atpažinimo testavimas. Lipnių lapelių atpažinimo testavimas buvo vykdomas panaudojant testavimo imties duomenis, kurie buvo skirti išvesties (angl. *inference*) žingsniui. Pateikus naujus paveikslėlius stebėta, kaip modelis tiksliai gali atpažinti lipnius paveikslėlius. Testavimo metu buvo pastebėta, kad modelis nepakankamai gerai atpažįsta:

- Dalinai uždengtus lipnius lapelius (tik matoma dalis)
- Labai mažus lapelius su daug teksto
- Lapelius su nestandartinėmis formomis (pasukti, sudėvėti ar sulankstyti kraštų)
- Lapelius prie pat nuotraukos krašto

Dėl šių priežasčių buvo praplėstas duomenų rinkinys, kuriame būtų padengtas tokių atvejų trūkumas. Papildžius duomenų rinkinį ir apmokinus modelį dar kartą, dalinai uždengti lipnūs lapeliai, lapeliai nestandartinėmis formomis ir pasukimais bei lapeliai prie pat nuotraukos krašto buvo atpažįstami teisingai, kaip parodyta 3.7 pav. Taip pat buvo pastebėta, jog kartais lapelio šešėlis dėl neįprasto apšvietimo buvo klasifikuojamas kaip atpažintas lipnus lapelis, tačiau su mažu pasitikėjimo procentu įprastai mažiau nei 80%. Ši problema buvo išspręsta

įdėjus papildomą filtravimą, kuris atmesdavo tokius lapelius, kurių atpažinimo pasitikėjimo procentas buvo mažesnis nei 80%.



3.7 pav. Testavimo atvejo nuotrauka, kai lapelis yra pasuktas ir yra lapelių nuotraukos krašte

Antroji artefaktų atpažinimo komponento testavimo dalis buvo artefakto tipo priskyrimo regionui testavimas. Kadangi artefaktų tipo atpažinimas remiasi lapelio spalvos nustatymu pagal iš anksto pateiktą spalvų sąrašą, buvo naudojami tie patys testavimo imties duomenys naudoti lipnių lapelių atpažinimo testavime. Po artefaktų atpažinimo buvo tikrinami kokia dominuojanti spalva buvo nuskaityta iš lapelių. Kiekvieno lapelio dominuojanti spalva yra matoma 3.8 pav.



3.8 pav. Lapeliai ir juose atpažintos dominuojančios spalvos

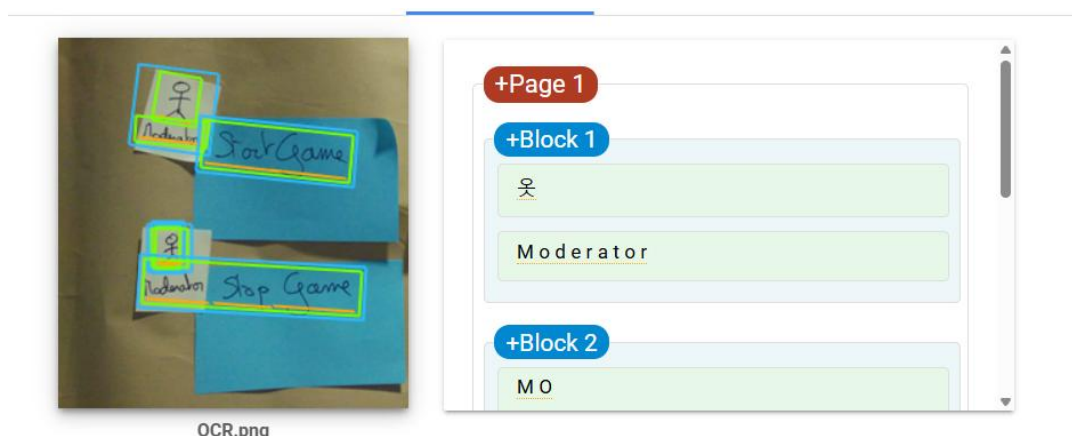
Tačiau buvo pastebėta, kad tam tikrais atvejais sistema nustatydavo dominuojančią lapelio spalvą šiek tiek kitokią nei naudotojo pateiktame pradiniam spalvų sąrašė. Tokie netikslumai dažniausiai atsirasdavo dėl neįprastų apšvietimo sąlygų, pavyzdžiui, kai stipri šviesa krisdavo iš vienos pusės, arba kai lipnus lapelis būdavo padėtas ant spalvoto fono. Papildomą įtaką rezultatui turėjo ir tai, kad aptikto lapelio regionas ne visada tiksliai sutapdavo tik su pačiu lapeliu, todėl į regioną galėdavo patekti ir dalis aplinkinio fono. Dėl šios priežasties spalva nustatoma ne iš visos aptiktos srities, bet iš jos vidinės dalies: regiono kraštai yra ignoruojami, o centrinės srities pikseliams suteikiamas didesnis svoris

nei kraštiniais. Taip pat atmetami labai tamsūs pikseliai, kad ranka rašytas tekstas kuo mažiau veiktų rezultata.

Siekiant dar labiau sumažinti apšvietimo, šešėlių, fono ir netikslių regiono ribų įtaką, buvo pridėtas papildomas lapelio spalvos palyginimas su naudotojo pateiktu spalvų sąrašu, atliekamas *LAB* spalvų erdvėje, apskaičiuojant spalvų skirtumą pagal *Delta E* metriką. Toks metodas leidžia parinkti žmogaus akiai artimiausią spalvą iš pateikto spalvų sąrašo, nes *LAB* spalvų erdvė geriau atitinka žmogaus regimą spalvų suvokimą nei tiesioginis palyginimas *RGB* erdvėje. Tokiu būdu sistema gali patikimiau nustatyti artefakto tipą net ir esant apšvietimo netolygumams, šešėliams, spalvotam fonui ar nedideliems spalvos pokyčiams.

3.3.2. Teksto atpažinimo (OCR) komponento testavimas

Teksto atpažinimo komponentas buvo testuojamas pateikiant *EventStorming* paveikslėlį su lipniais lapeliais, kurie turėjo ant jų užrašyto teksto. 3.9 pav. yra pateikiamas pavyzdinis *EventStorming* artefaktų paveikslėlis, kuris buvo perduotas teksto atpažinimo komponentui. Jame taip pat matoma teksto atpažinimo komponento sužymėti teksto regionai bei atpažintas tekstas.



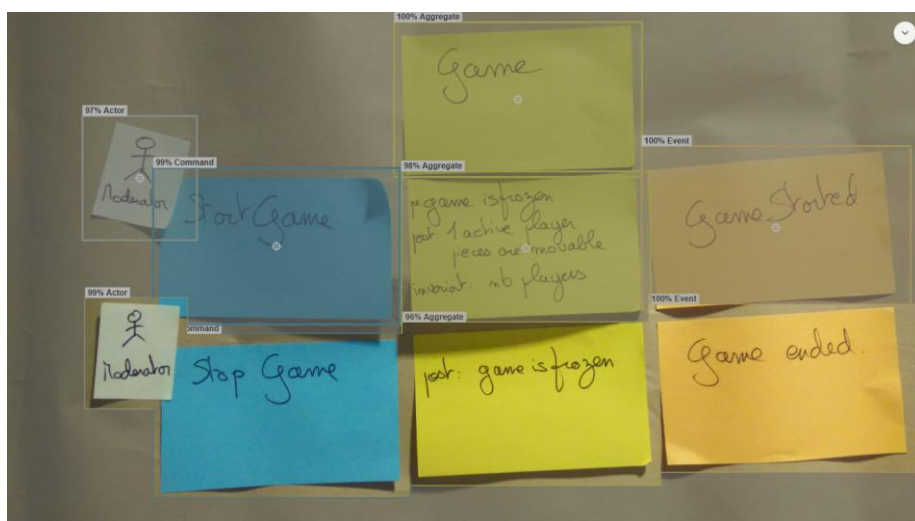
3.9 pav. Teksto atpažinimo komponento rezultatai testavimo metu

Buvo testuojama ar paveikslėlyje matomas tekstas atitinka tai, ką sugebėjo atpažinti teksto atpažinimo komponentas. Pastebėta, jog kartais teksto atpažinimo komponentas nupieštą aktorius figūrą atpažindavo kaip simbolį, todėl buvo pridėtas papildomas atpažinto teksto filtravimas, paliekantis tik teksto regionus, kuriame yra tekstiniai simboliai.

3.3.3. Klasterizavimo komponento testavimas

Klasterizacijos komponento testavimas buvo vykdomas panaudojant realių *EventStorming* sesijų nuotraukas su jau aptiktais ir klasifikuotais artefaktais. Visų pirma, buvo įkeliamas *EventStorming* sesijos paveikslėlis, peržiūrimi automatiškai aptikti artefaktai. Vėliau klasterizavimo komponentas automatiškai sugrupuoja artefaktus į klasterius. Šie klasteriai buvo vertinami pagal *EventStorming* metodo žinias, t.y. ar klasteris atitinka loginį

EventStorming procesą pateiktą 1.9 pav. Vienas iš aptiktų klasterio pavyzdžių yra matomas 3.10 pav.



3.10 pav. Klasterizavimo komponento testavimo metu atpažintas klasteris

Testavimo metu pastebėta, jog kartais tam tikri artefaktai nebūdavo priskirti nė vienam iš klasterių, todėl buvo nuspręsta papildomai pridėti funkciją, kuri nepriskirtus artefaktus priskiria Euklidiniu atstumu artimiausiam artefakto klasteriui nepaisant *EventStorming* artefaktų sąryšių.

3.3.4. Reikalavimų generavimo komponentas

Reikalavimų generavimo komponento testavimą sudarė dvi dalys: naudotojo pasakojimų generavimo *LLM* agento testavimas ir reikalavimų eksportavimo testavimas. Naudotojo pasakojimų generavimo *LLM* agento testavimas buvo atliekamas remiantis *EventStorming* metodo žiniomis, vertinant, ar sugeneruoti naudotojo pasakojimai atitinka surastus klasterius ir ar tinkamai apibūdina jų reikšmę dalykinės srities bei proceso kontekste. 3.11 pav. pavaizduotas sugeneruotas naudotojo pasakojimas iš 3.10 pav. klasterio.

Name	Description
Stop Game	As a Moderator, I want to stop the game, so that the game can be ended when necessary.

Full User Story

As a Moderator, I want to stop the game, so that the game can be ended when necessary.

Acceptance Criteria:

- Given a game is in progress, when the Moderator issues the 'Stop Game' command, then the game should end.
- The event 'Game Ended' should be triggered upon stopping the game.

Related Artifacts: [Actor: Moderator, Command: Stop Game, Event: Game Ended]

3.11 pav. Reikalavimų generavimo komponento testavimo metu sukurtas naudotojo pasakojimas

Testavimo metu buvo pastebėta, kad *LLM* agentas kartais pridėdavo papildomų naudotojo pasakojimų, kurie buvo susiję su bendru projekto kontekstu, tačiau neatitiko nė vieno konkretaus klasterio artefaktų. Siekiant išspręsti šią ir kitas problemas, instrukcija buvo

tikslinama iteratyviai, palaipsniui papildant ją naujomis taisyklėmis, pagrindinės instrukcijos tikslinimo iteracijos pateiktos 4 lentelėje.

4 lentelė. LLM agento instrukcijos tikslinimo pagrindinės iteracijos įrankio testavimo metu

Iteracija	Pagrindinė problema	Instrukcija papildyta	Rezultatas
0	<i>Pradinis instrukcijos variantas leido modelį interpretuoti klasterius pernelyg laisvai, todėl buvo generuojami ir su artefaktais tiesiogiai nesusieti naudotojo pasakojimai.</i>	<i>Nurodyta naudotojo pasakojimo forma, priėmimo kriterijai ir techninės pastabos.</i>	<i>Gauti bendrai temą atitinkantys, bet ne visada tiksliai artefaktus atspindintys rezultatai.</i>
1	<i>Modelis pridėdavo papildomų veiksmų ar taisyklių, kurių klasteryje nebuvo.</i>	<i>Use only the Actors explicitly listed in the cluster; Don't add validation rules unless they come from Aggregates, Policies, or Hotspots; Every acceptance criterion should trace back to an artifact; Do NOT create generic stories...</i>	<i>Sumažėjo su klasteriu nesusietų naudotojo pasakojimų ir perteklinių interpretacijų.</i>
2	<i>Keli proceso etapai būdavo sujungiami į vieną naudotojo pasakojimą, ne visada išlaikant teisingą seką.</i>	<i>Generate 2-4 user stories per cluster by default; Make stories atomic; Split by layer, phase, actor, command, or system integration when appropriate; Use spatial coordinates (x, y) to determine temporal sequence</i>	<i>Naudotojo pasakojimai tapo smulkesni, aiškiau atitiko atskirus proceso etapus ir jų seką.</i>
3	<i>Rezultatai kartais buvo gražinami laisva tekstine forma, netinkama automatiniam apdorojimui.</i>	<i>Return ONLY a JSON object; Return results in valid JSON format; description must start with 'As a...'; no markdown; ALWAYS use 'Technical Notes:' section; Do NOT include 'Related Artifacts' sections</i>	<i>Atsakymai tapo struktūrizuoti, nuoseklūs ir tinkami saugojimui bei eksportavimui.</i>

Pradinis instrukcijos variantas buvo orientuotas į bendrą *EventStorming* artefaktų interpretavimą ir naudotojo pasakojimų generavimą. Jo esmė buvo tokia: modelis turi analizuoti *EventStorming* artefaktus, nustatyti aktorių, komandą, įvykius, taisykles ir kitus artefaktus, o tada pagal juos sugeneruoti naudotojo pasakojimus. Pradiniame variante jau buvo nurodyta naudotojo pasakojimo forma (*As a ... I want to ... so that ...*) bei poreikis generuoti priėmimo kriterijus ir technines pastabas, tačiau ši versija dar nepakankamai aiškiai ribojo modelį naudoti tik klasteryje pateiktą informaciją, laikytis artefaktų sekos ir gražinti griežtai struktūrizuotą rezultatą.

Pirmuoju testavimo scenarijumi buvo sprendžiama haliucinacijų problema. Naudojant pradinę instrukciją modelis kartais sugeneruodavo pasakojimus, kurie atrodė logiški nagrinėjamos veiklos srities požiūriu, tačiau neturėjo tiesioginio pagrindo klasteryje esančiuose artefaktuose. Pavyzdžiui, iš naujienlaiškio prenumeratos paveikslo, matomo 1.18 pav., buvo sugeneruotas toks pasakojimas: „*As a User, I want to validate my signup request, so that only eligible users are subscribed.*“ Toks pasakojimas buvo semantiškai galimas, tačiau klasteryje nebuvo aiškiai pateikta nei tokia komanda, nei toks naudotojo tikslas. Siekiant sumažinti tokių atvejų skaičių, instrukcija buvo papildyta šiomis taisyklėmis: „*Use only the Actors explicitly listed in the cluster, Don't add validation rules unless they*

come from Aggregates, Policies, or Hotspots, Every acceptance criterion should trace back to an artifact“ ir „*Do NOT create generic stories - be specific based on the actual artifacts and their spatial relationships*“. Po šių papildymų modelis pradėjo rečiau generuoti su klasteriu nesusietus pasakojimus, tačiau vis dar pasitaikydavo atveju, kai keli proceso žingsniai būdavo sujungiami į vieną per platų naudotojo pasakojimą.

Antruoju testavimo scenarijumi buvo sprendžiama pasakojimų skaidymo ir proceso sekos problema. Po pirmojo patikslinimo modelis jau buvo geriau apribotas klasterio duomenimis, tačiau dar ne visada pakankamai tiksliai laikėsi *EventStorming* proceso eigos. Kai kuriais atvejais sugeneruotas naudotojo pasakojimas apjungdavo kelis skirtingus veiksmus, pavyzdžiui, prenumeratos pateikimą, patvirtinimo laiško gavimą ir galutinį prenumeratos patvirtinimą. Toks naudotojo pasakojimas apėmė kelis savarankiškus proceso etapus, todėl jį buvo tikslinga skaidyti į atskirus naudotojo pasakojimus. Siekiant tai ištaisyti, instrukcija buvo papildyta taisyklėmis „*Generate 2-4 user stories per cluster by default, Make stories atomic, Split by layer, phase, actor, command, or system integration when appropriate ir Use spatial coordinates (x, y) to determine temporal sequence*“. Taip pat buvo pridėtas papildomas paaiškinimas, kad *EventStorming* artefaktų seka paprastai vyksta iš kairės į dešinę arba iš viršaus į apačią. Po šio pakeitimo sugeneruoti pasakojimai tapo smulkesni, aiškiau atitiko atskirus proceso etapus.

Trečiuoju testavimo scenarijumi buvo sprendžiama išvesties struktūros problema. Net ir pagerinus pasakojimų tikslumą bei jų skaidymą, modelis kartais vis dar gražindavo rezultatai laisva tekstone forma, su papildomais paaiškinimais, „*Markdown*“ antraštėmis ar pertekliniais skyriais, tokiais kaip „*User Story 1*“, „*Related Artifacts*“ ar įvadiniai sakiniai prieš pačius pasakojimus. Toks atsakymas buvo suprantamas žmogui, tačiau netinkamas automatiniam tolesniam apdorojimui sistemoje. Dėl šios priežasties instrukcija buvo papildyta taisyklėmis „*Return ONLY a JSON object, Return results in valid JSON format, description must start with 'As a...', no markdown, ALWAYS use 'Technical Notes:' section ir Do NOT include 'Related Artifacts' sections*“. Be to, buvo tiksliai apibrėžta *JSON* struktūra su *requirements*, *name* ir *description* laukais. Po šio pakeitimo modelio gražinami rezultatai tapo nuoseklūs, tinkami automatiniam nuskaitymui bei tolesniam panaudojimui reikalavimų saugojimo ir eksportavimo, programinės įrangos kūrimo procese.

Apibendrinant galima teigti, kad pradinės instrukcijos pakako bendrai *EventStorming* artefaktų prasmei interpretuoti, tačiau jos nepakako tiksliai ir stabiliai pakartojamam reikalavimų generavimui. Tik iteratyviai papildant instrukciją aiškiais taisyklėmis apie artefaktų sąryšius, pasakojimų skaidymą, haliucinacijų ribojimą ir išvesties struktūrą, pavyko pasiekti, kad naudotojo pasakojimų generavimo LLM agentas gražintų klasterius tiksliau atitinkančius ir automatiniam apdorojimui tinkamus naudotojo pasakojimus.

Reikalavimų generavimo komponentas taip pat vykdo ir reikalavimų eksportavimą. Testavimo metu per naudotojo sąsają buvo eksportuojami reikalavimai ir tikrinama ar sudarytas *.csv* failas tenkina *.csv* failo taisykles ir ar sėkmingai importuojasi į *Jira* įrankį. Rankinio testavimo metu šios dalies trūkumų nepastebėta, reikalavimai teisingai eksportavosi ir susiimportavo į *Jira* įrankį.

4. Eksperimentinis reikalavimų išgavimo iš *EventStorming* artefaktų įrankio tyrimas

4.1. Eksperimento planas

Reikalavimų išgavimo iš *EventStorming* artefaktų įrankio tyrimas yra sudarytas iš trijų dalių:

1. **Lipnių lapelių atpažinimo eksperimentas.** Tikslas – įvertinti lipnių lapelių atpažinimo tikslumą ir kitas kokybines metrikas.
2. **Artefaktų tipų nustatymo eksperimentas.** Tikslas – įvertinti, kaip tiksliai sistema nustato artefakto tipą bei apskaičiuoti atitinkamas kokybines metrikas.
3. **Įrankio, išgautų reikalavimų bei proceso vertinimas.** Tikslas – įvertinti, ar sukurtas *EventStorming* įrankis palengvina reikalavimų išgavimo procesą, ar sugeneruoti naudotojo pasakojimai yra naudingi ir ar įrankis yra tinkamas naudoti praktikoje.

4.2. Lapelių atpažinimo eksperimentas

Lapelių atpažinimo eksperimentas buvo atliekamas siekiant įvertinti vieną iš reikalavimų išgavimo įrankio dalių, t.y. kaip apmokytas vaizdų atpažinimo modelis gali tiksliai atpažinti lipnius lapelius iš nuotraukos.

4.2.1. Lapelių atpažinimo eksperimento duomenų rinkinys

Lapelių atpažinimo modeliui apmokymui ir jo tikslumui iširti buvo sudarytas lipnių lapelių rinkinys kurį sudaro 159 skirtingų lipnių lapelių nuotraukų. Vaizdai buvo renkami iš skirtingų viešai prieinamų internetinių šaltinių. Sudarant vaizdų rinkinį buvo naudojami vaizdai, kuriuose lapeliai buvo matomi iš įvairių krypčių, įvairiomis padėtimis, siekiant sudaryti modelį, kuris galėtų tiksliau atpažinti įvairaus tipo, pozicijų lapelius. 144 (90,6%) nuotraukų yra gautos iš *Unsplash*, *Github*, *Pexels*, ir kt. internetinių šaltinių, o 15 (9,4%) nuotraukų sudarytos rankiniu būdu fotografuojant užklijuotus fizinius lipnius lapelius. 4.1 pav. matomos fizinių ir skaitmeninių įrankiu sukurtų lipnių lapelių nuotraukos iš duomenų rinkinio. Duomenų rinkinyje 25 (15,7%) nuotraukos su lipniais lapeliais yra sukurtos skaitmeniniais įrankiais tokiais kaip *miro.com*, *creatly.com* ir kt., o 134 (84,3%) nuotraukos sukurtos fotografuojant fizinius lipnius lapelius. Didesniu fizinių lapelių kiekiu duomenų rinkinyje siekiama padengti įvairias fizinio pasaulio situacijas, kurios gali apsunkinti lipnių lapelių aptikimą, tokias kaip užlankstyti, prastai apšviesti, pakreipti lapeliai. Vienoje nuotraukoje dažniausiai yra net keli lipnūs lapeliai, todėl šiame duomenų rinkinyje 159 nuotraukose viso yra 725 lipnūs lapeliai.



4.1 pav. Fizinių lipnių lapelių nuotrauka iš duomenų rinkinio (kairėje) bei *miro.com* įrankio lipnus lapelis iš duomenų rinkinio (dešinėje)

4.2.2. Lapelių atpažinimo eksperimento vykdymo eiga ir naudojamos metrikos

Eksperimentas buvo atliktas automatizuotu būdu vertinant kaip apmokytas lapelių atpažinimo modelis teisingai atpažįsta lipnius lapelius naudojant *k-fold* kryžminę patikrą. Ši patikra padalina duomenų rinkinį į *K* lygių dalių ir kiekvieną kartą viena dalis naudojama kaip testavimo aibė o likusios dalys kaip mokymo aibės. Eksperimentui sudaryta lentelė, kurioje kiekvienai daliai skaičiuoti tokie rodikliai: anotuoti lipnūs lapeliai (**GT**, angl. *ground truth*), aptikti lapeliai (**DET**, angl. *detected*) tikrieji teigiami rezultatai (**TP**, angl. *true positive*), klaidingai neigiami rezultatai (**FN**, angl. *false negative*) ir klaidingai teigiami rezultatai (**FP**, angl. *false positive*). **TP** reikšmė reiškia, kad nuotraukoje esantis lapelis buvo teisingai identifiktuotas, **FN** – kad lapelis nebuvo atpažintas, o **FP** nurodo klaidingai atpažintą lapelį, kurio iš tikrųjų nuotraukoje nėra. Remiantis apskaičiuotais **TP**, **FN** ir **FP** rodikliais, buvo nustatytos atkūrimo (angl. *recall*) ir preciziškumo (angl. *precision*) reikšmės. Atkūrimo reikšmė apskaičiuojama pagal (1) formulę:

$$\text{Atkūrimas} = \frac{TP}{TP+FN} \quad (1)$$

Ji parodo gebėjimą atpažinti visus lapelius nuotraukoje. Tuo tarpu preciziškumas skaičiuojamas pagal (2) formulę:

$$\text{Preciziškumas} = \frac{TP}{TP+FP} \quad (2)$$

Šis rodiklis vertina gebėjimą teisingai atpažinti lapelį. Šios metrikos leidžia įvertinti modelio tikslumo aspektus: preciziškumas atspindi teigiamų prognozių tikslumą, o atkūrimas – gebėjimą nustatyti visus teisingus atvejus.

Galiausiai, remiantis apskaičiuotais preciziškumo ir atkūrimo rodikliais, buvo nustatytas ir **F1** įvertis (angl. *F1-score*), skaičiuojamas pagal (3) formulę:

$$F1 = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3)$$

Tai yra harmoningas preciziškumo ir atkūrimo vidurkis, kuris bendrai įvertina modelio tikslumą, atsižvelgiant tiek į klaidingai teigiamus, tiek į klaidingai neigiamus rezultatus [61].

4.2.3. Lapelių atpažinimo eksperimento rezultatai

Apskaičiavus kiekvieno *k-fold* žingsnio rezultatus, buvo įvertintos bendros lipnių lapelių aptikimo metrikos – atkūrimas, preciziškumas ir F1 įvertis. 5 lentelėje pateikti penkių *k-fold* padalijimo ir bendri viso rinkinio rezultatai.

Bendras lipnių lapelių aptikimo atkūrimas yra 0,8869, o preciziškumas – 0,9626. Iš viso teisingai aptikti 643 lipnūs lapeliai (TP), 82 lapeliai nebuvo aptikti (FN), o 25 objektai buvo klaidingai nustatyti kaip lipnūs lapeliai (FP). Aukštas preciziškumas rodo, kad modelis retai suklysta priskirdamas objektą lipniam lapeliui, tačiau atkūrimo reikšmė yra šiek tiek mažesnė, todėl dalis tikrųjų objektų lieka neaptikti. Bendras F1 įvertis siekia 0,9232, kas rodo gerą bendrą modelio veikimą.

Analizuojant atskirus *k-fold* padalijimus, pirmojo padalijimo atkūrimas yra mažiausias – 0,6667, o preciziškumas 0,9427. Teisingai aptikti 148 lapeliai iš 222, 74 nebuvo nustatyti ir

9 buvo nustatyti klaidingai. Šis rezultatas rodo, kad šiame duomenų padalijime modelis praleido santykinai daug objektų. Įvertinus padalijimą, buvo pastebėta, jog atsitiktinai dalijant duomenų rinkinį į mokymo aibę buvo įtraukta mažai paverstų, netaisyklingų lapelių, o testavimo aibėje beveik visos nuotraukos turėjo paverstu, atsiklijavusiu ir kitaip iškraipytų lapelių, kas ir lėmė ženkliai mažesnį atkūrimą nei kitų padalijimų. Buvo nuspręsta nenaudoti šiuo padalijimu apmokyto modelio reikalavimų išgavimo įrankiui kurti.

Ketvirtajame *k-fold* padalijime pasiektas vienas geriausių rezultatų – atkūrimas 1,0000 ir preciziškumas 0,9966. Teisingai aptikti visi 155 lapeliai, tik 1 objektas nustatytas klaidingai. Buvo nuspręsta šiuo padalijimu apmokytą modelį naudoti automatizuoto reikalavimų išgavimo iš *EventStorming* artefaktų įrankyje.

5 lentelė. Lapelių atpažinimo rezultatų lentelė kiekvienam *k-fold* padalijimui

<i>k-fold</i> nr.	GT	DET	TP	FP	FN	Precision	Recall	F1
1	222	157	148	9	74	0,9427	0,6667	0,7810
2	122	117	115	2	7	0,9829	0,9426	0,9623
3	115	120	115	5	0	0,9583	1,0000	0,9787
4	155	156	155	1	0	0,9966	1,0000	0,9968
5	111	118	110	8	1	0,9322	0,9910	0,9607
Viso	725	668	64	25	82	0,9626	0,8869	0,9232

Apibendrinant galima teigti, kad lipnių lapelių aptikimo modelis pasižymi labai aukštu preciziškumu visuose *k-fold* padalijimuose, o atkūrimo reikšmės daugumoje atvejų taip pat yra labai aukštos. Mažesnis atkūrimas pirmajame žingsnyje rodo jautrumą konkrečiam duomenų pasiskirstymui, nes mokymo aibėje buvo mažiau pasuktų ir prastai apšviestų lapelių, o didžioji jų dalis pateko į validavimo padalijimą. Nepaisant to, bendras F1 įvertis 0,9232 patvirtina, kad modelis efektyviai sprendžia lipnių lapelių aptikimo užduotį. Atsižvelgiant į gautus rezultatus, tolesniam automatizuoto reikalavimų išgavimo iš *EventStorming* artefaktų įrankio kūrimui buvo pasirinktas ketvirtojo *k-fold* padalijimo metu apmokytas modelis. Šis modelis ne tik pasiekė vieną geriausių rezultatų, bet ir pasižymėjo įvairiausiu apmokymo duomenų rinkiniu, todėl jo išmokti parametrai buvo užfiksuoti ir panaudoti galutinėje įrankio versijoje.

4.3. *EventStorming* artefaktų atpažinimo eksperimentas

EventStorming artefaktų atpažinimo eksperimentas buvo atliekamas siekiant įvertinti sekančią reikalavimų išgavimo įrankio dalį, t.y. kaip įrankis sugeba priskirti tinkamą artefaktą pagal naudotojo parinkta spalvų paletę anksčiau atpažintam lipniam lapeliui.

4.3.1. *EventStorming* artefaktų atpažinimo duomenų rinkinys

Artefaktų atpažinimo tikslumui ištirti buvo naudojama dalis lapelių atpažinimo eksperimente taikyto duomenų rinkinio. Iš šio rinkinio buvo pašalintos nuotraukos, kuriose lipnių lapelių nebuvo, taip pat nuotraukos, kuriose lapeliai buvo užklijuoti vienas ant kito arba išdėstyti padėtimis, nebūdingomis *EventStorming* artefaktams. Vėliau kiekvienai nuotraukai buvo priskirta *EventStorming* artefaktams reikalinga spalvų paletė, nurodanti, kokie artefaktų tipai

atitinka konkrečias nuotraukoje esančias spalvas. Galiausiai kiekvienam lapeliui buvo pridėta artefakto anotacija, t.y. nurodyta, kokio tipo artefaktą pagal savo spalvą jis turėtų atitikti. Iš viso galutinį rinkinį sudaro 124 nuotraukos su lipniais lapeliais ir artefaktų bei spalvų paletės anotacijomis.

4.3.2. *EventStorming* artefaktų atpažinimo eksperimento vykdymo eiga ir naudojamos metrikos

Eksperimentas buvo atliktas automatizuotu būdu, siekiant įvertinti, kaip spalvų gretinimo algoritmas priskiria lipniems lapeliams *EventStorming* artefaktų tipus pagal pateiktą spalvų paletę. Skirtingai nuo daugumos mašininio mokymosi klasifikatorių, šiame etape naudojamas algoritmas yra deterministinis: kiekvienam aptiktam lapeliui visada priskiriama viena artefakto klasė, pasirenkama pagal mažiausią spalvinį atstumą *LAB* spalvų erdvėje, apskaičiuojama taikant *CIE76 Delta E* metriką.

Eksperimentui sudaryta lentelė, kurioje buvo skaičiuojama kiek teisingai artefaktų buvo atpažinta, kiek buvo suklysta bei skaičiuojamas tikslumas kiekvienam artefakto tipui (angl. *per-class accuracy*) pagal (4) formulę:

$$\text{Artefakto tikslumas (\%)} = \frac{\text{Teisingai priskirti lapeliai artefakto tipui}}{\text{Viso rinkinyje artefakto tipo}} \times 100 \quad (4)$$

Taip pat buvo skaičiuojamas bendras artefaktų atpažinimo tikslumas (angl. *overall accuracy*), kuris buvo apskaičiuotas kaip visų teisingai klasifikuotų lapelių suma padalyta iš visų klasifikuotų lapelių skaičiaus pagal (5) formulę:

$$\text{Bendras tikslumas (\%)} = \frac{\sum_{i=0}^K C_i}{\sum_{i=0}^K N_i} \times 100 \quad (5)$$

čia K – visų artefaktų tipų skaičius, C_i – teisingai klasifikuotų lapelių skaičius i -ajam artefaktui, o N_i – visų to artefakto tipų skaičius.

4.3.3. *EventStorming* artefaktų atpažinimo eksperimento rezultatai

Apskaičiavus kiekvieno artefakto tipo metrikas bei bendrąjį tikslumą, gauti rezultatai buvo apibendrinti ir pateikti 6 lentelėje.

6 lentelė. *EventStorming* artefaktų atpažinimo eksperimento rezultatų lentelė

Artefakto tipas	Viso	Teisingai atpažinti	Neteisingai atpažinti	Tikslumas
Aktorius	176	168	8	95,45%
Komanda	99	91	8	91,92%
Sistema	48	43	5	89,58%
Taisyklė	41	37	4	90,24%
Įvykis	121	115	6	95,04%
Skaitymo modelis	38	37	1	97,37%
Aktyvioji sritis	9	9	0	100,00%
Agregatas	4	4	0	100,00%
Viso	536	504	32	94,03%

Didžiausias procentinis tikslumas buvo pasiektas aktyviosios srities ir agregato artefaktų tipams (100%), tačiau šių tipų pavyzdžių skaičius yra labai mažas (viso 13). Skaitymo modelio (97,37%) bei aktorius (95,45%) artefaktų tipų aukštas tikslumas ir duomenų imtis rodo, kad deterministinis *LAB* spalvų erdvės algoritmas geba patikimai atskirti šiuos artefaktų tipus pagal jų spalvines charakteristikas.

Mažesnis tikslumas pastebimas sistemos (89,58%), komandos (91,92%) ir taisyklės (90,24%) artefaktų tipų atpažinime, taip yra todėl, jog šių tipų spalvinės savybės yra panašesnės ir dėl to sunkiau atskiriamos. Įvykio tipas taip pat pasižymi aukštu tikslumu (95,04%).

Apibendrinant visus artefaktų tipus, bendras klasifikavimo tikslumas siekia 94,03%, todėl galima teigti, kad spalvų gretinimo algoritmas yra tinkamas *EventStorming* artefaktų atpažinimui realių vaizdų sąlygomis.

4.4. Įrankio, išgautų reikalavimų bei proceso vertinimas

Šio tyrimo etapo tikslas yra įvertinti ne tik techninį įrankio veikimą, bet ir jo praktinę vertę naudotojui. Skirtingai nuo ankstesnių eksperimentų, kuriuose buvo vertinamas lipnių lapelių ir artefaktų tipų atpažinimo tikslumas, šiame etape siekiama įvertinti, ar sukurtas įrankis palengvina reikalavimų išgavimo procesą, ar automatiškai sugeneruoti naudotojo pasakojimai yra naudingi reikalavimų analizėje ir ar toks sprendimas turi praktinį pritaikomumą projektuose, kuriuose naudojamas *EventStorming* metodas.

Kadangi sugeneruotų naudotojo pasakojimų kokybė ir jų nauda praktikoje negali būti pilnai įvertinta vien objektyviomis klasifikavimo metrikomis, buvo pasirinktas apklausos metodas. Šis metodas leido surinkti grįžtamąjį ryšį iš respondentų, turinčių skirtingą patirtį programinės įrangos kūrimo, reikalavimų analizės, *EventStorming* ir naudotojo pasakojimų kūrimo srityse. Tokiu būdu buvo siekiama įvertinti ne tik bendrą požiūrį į sukurtą įrankį, bet ir jo suvokiamą naudą realiaame reikalavimų analizės procese.

4.4.1. Įrankio, išgautų reikalavimų bei proceso vertinimo apklausos duomenys

Įrankio, išgautų reikalavimų bei proceso vertinimui buvo sudaryta anoniminė apklausa. Apklausos pradžioje respondentams buvo pateiktas trumpas tyrimo aprašymas, kuriame nurodyta, kad apklausa vykdoma magistro baigiamojo darbo tema „Automatizuotas naudotojo pasakojimų reikalavimų išgavimas iš *EventStorming* metodu gautų artefaktų“, o jos tikslas – įvertinti, ar sukurtas *EventStorming* įrankis palengvina reikalavimų išgavimo procesą ir ar yra tinkamas naudoti praktikoje. Taip pat buvo nurodyta, kad apklausa yra anoniminė. Apklausoje respondentams pirmiausia buvo pateikti trys kontekstiniai klausimai apie jų patirtį:

1. Kokia yra Jūsų patirtis programinės įrangos kūrimo, reikalavimų analizės ar veiklos analizės srityse?
 - 0-2 metai;
 - 3-5 metai;
 - 6+ metai.
2. Kiek patirties turite dirbant su *EventStorming* metodu?

- Neturiu patirties;
- Turiu teorinių žinių (esu susipažinęs / skaitęs / mokėsis, bet praktiškai nenaudojęs);
- Esu dalyvavęs vienoje *EventStorming* dirbtuvėse;
- Esu dalyvavęs dviejose ar daugiau *EventStorming* dirbtuvėse;
- Esu vedęs *EventStorming* dirbtuves.

3. Kiek patirties turite kuriant naudotojo pasakojimus (angl. *user stories*)?

- Neturiu patirties;
- Esu susipažinęs su naudotojo pasakojimų kūrimo koncepcija;
- Esu naudojęs naudotojo pasakojimus keliuose (2+) projektuose;
- Reguliariai kuriu naudotojo pasakojimus praktinėje veikloje;
- Esu atsakingas už naudotojo pasakojimų kūrimo procesą (pvz., kaip analitikas, PO ar komandos narys).

Toliau respondentams buvo pateikta demonstracinė vaizdo medžiaga apie įrankį, jo veikimą ir gaunamus rezultatus. Demonstracijoje buvo naudojama naujienlaiškių rezervacijos

proceso *EventStorming* lenta bei pagal ją sugeneruoti naudotojo pasakojimai, kuriuos galima matyti 4.2 pav. ir 4.3 pav.



Submit newsletter signup request

As a User, I want to submit my newsletter sign up request, so that my interest in subscribing is recorded.

Full User Story

As a User, I want to submit my newsletter sign up request, so that my interest in subscribing is recorded.

Acceptance Criteria:

- The user can submit a newsletter sign up request.
- The newsletter signup request is recorded in the system.
- Newsletter marketing information is available for tracking the request.
- The request is stored in the CRM.

Technical Notes:

- CRM is involved in recording the signup request.
- Newsletter Marketing read model is involved for tracking and visibility.

Receive confirmation email

As a User, I want to receive a confirmation email after my sign up request, so that I can verify my email address.

Full User Story

As a User, I want to receive a confirmation email after my sign up request, so that I can verify my email address.

Acceptance Criteria:

- A confirmation email is automatically sent after the signup request is recorded.
- The confirmation email delivery is recorded in the system.
- The user can see that the confirmation email has been sent.

Technical Notes:

- CRM sends the confirmation email.
- This story covers the automated response to the signup request.

4.2 pav. Apklausoje naudotas *EventStorming* ir sugeneruotų naudotojo pasakojimų pavyzdys
pirma dalis

Confirm email address	As a User, I want to confirm my email address, so that my newsletter subscription can proceed.	^
<p>Full User Story</p> <p>As a User, I want to confirm my email address, so that my newsletter subscription can proceed.</p> <p>Acceptance Criteria:</p> <ul style="list-style-type: none"> - The user can confirm their email address. - The email confirmation is recorded in the system. - The user can view the confirm email request details needed to complete verification. <p>Technical Notes:</p> <ul style="list-style-type: none"> - CRM is involved in confirming the email. - Confirm Email Request read model provides the information needed for the user to complete confirmation. 		
Accept newsletter subscription	As a User, I want my newsletter subscription to be accepted after my email is confirmed, so that my subscription is completed.	^
<p>Full User Story</p> <p>As a User, I want my newsletter subscription to be accepted after my email is confirmed, so that my subscription is completed.</p> <p>Acceptance Criteria:</p> <ul style="list-style-type: none"> - The subscription is accepted after the email is confirmed. - The subscription acceptance is recorded in the system. - The acceptance process starts automatically after email confirmation. - The user has a completed subscription status. <p>Technical Notes:</p> <ul style="list-style-type: none"> - CRM is involved in accepting the subscription. - Confirmation handling is automated after the email is confirmed. 		

4.3 pav. Apklausoje naudotas *EventStorming* ir sugeneruotų naudotojo pasakojimų pavyzdys antra dalis

Peržiūrėję demonstracinę medžiagą, respondentai turėjo įvertinti šiuos teiginius naudojant penkiabalę Likerto skalę, kur 1 reiškė „visiškai nesutinku“, o 5 – „visiškai sutinku“:

1. Siūlomas automatizavimo įrankis palengvina naudotojo pasakojimų išgavimą iš *EventStorming* artefaktų.
2. Įrankio automatizuotai sugeneruoti naudotojo pasakojimai yra naudingi reikalavimų analizės procese.
3. Šis įrankis turi praktinį pritaikymą projektuose, kuriuose naudojamas *EventStorming* metodas.

Iš viso buvo gauti 25 respondentų atsakymai. Respondentų pasiskirstymas pagal patirtį pateiktas 7 lentelėje. Daugiausia respondentų turėjo 6 ir daugiau metų patirties programinės įrangos kūrimo, reikalavimų analizės ar veiklos analizės srityse – 52,0%. Vertinant patirtį dirbant su *EventStorming* metodu, 28,0% respondentų nurodė neturintys patirties, dar 28,0% turėjo teorinių žinių, o 44,0% buvo bent kartą praktiškai dalyvavę *EventStorming* dirbtuvėse arba jas vedę. Vertinant patirtį kuriant naudotojo pasakojimus, 36,0% respondentų nurodė tai darantys reguliariai praktinėje veikloje, 16,0% buvo atsakingi už naudotojo pasakojimų kūrimo procesą, o tik 8,0% neturėjo jokios tokios patirties. Tai rodo, kad apklausos imtį daugiausia sudarė respondentai, turintys aktualios profesinės patirties vertinamoje srityje.

7 lentelė. Respondentų pasiskirstymas pagal patirtį

Rodiklis	Reikšmė	Kiekis	Dalis, %
<i>Patirtis programinės įrangos kūrimo, reikalavimų analizės ar veiklos analizės srityse</i>	<i>0-2 metai</i>	4	16,0
	<i>3-5 metai</i>	8	32,0
	<i>6+ metai</i>	13	52,0
<i>Patirtis dirbant su EventStorming metodu</i>	<i>Neturiu patirties</i>	7	28,0
	<i>Turiu teorinių žinių</i>	7	28,0
	<i>Esu dalyvavęs vienoje dirbtuvėse</i>	4	16,0
	<i>Esu dalyvavęs dviejose ar daugiau dirbtuvių</i>	6	24,0
	<i>Esu vedęs EventStorming dirbtuves</i>	1	4,0
<i>Patirtis kuriant naudotojo pasakojimus</i>	<i>Neturiu patirties</i>	2	8,0
	<i>Esu susipažinęs su koncepcija</i>	4	16,0
	<i>Esu naudojęs keliuose projektuose</i>	6	24,0
	<i>Reguliariai kuriu praktinėje veikloje</i>	9	36,0
	<i>Esu atsakingas už kūrimo procesą</i>	4	16,0
<i>Viso respondentų</i>	25		

Respondentų pasiskirstymas pagal patirtį rodo, kad apklausoje dalyvavo įvairaus pasirengimo specialistai. Kadangi didžioji dalis respondentų turėjo patirties programinės įrangos kūrimo ar reikalavimų analizės srityse, o nemaža dalis taip pat buvo susipažinę su naudotojo pasakojimų kūrimu ir *EventStorming* metodu, galima teigti, kad surinkti vertinimai yra reikšmingi siūlomo sprendimo praktiniam įvertinimui.

4.4.2. Įrankio, išgautų reikalavimų bei proceso vertinimo eksperimento vykdymo eiga ir naudojamos metrikos

Eksperimentas buvo vykdomas apklausos forma. Apklausos nuoroda buvo išplatinta internetu, o respondentai ją pildė savarankiškai nuotoliniu būdu. Prieš atsakydami į vertinimo klausimus, jie buvo supažindinti su įrankio paskirtimi, bendru veikimo principu ir sugeneruojamų rezultatų pavyzdžiu demonstracinėje vaizdo medžiagoje. Tokiu būdu siekta užtikrinti, kad visi respondentai vertintų tą patį scenarijų ir turėtų vienodą pirminį kontekstą apie siūlomą sprendimą. Kadangi šiame tyrimo etape buvo vertinamas ne klasifikavimo tikslumas, o respondentų suvokiama įrankio nauda, buvo pasirinktos aprašomosios statistikos metrikos. Kiekvienam iš trijų vertinamų teiginių buvo apskaičiuotas:

- Vertinimų vidurkis.
- Mediana.
- Standartinis nuokrypis.
- Dažniausiai pasitaikęs atsakymas.
- Vertinimų pasiskirstymas pagal Likerto skalę.

- Teigiamų vertinimų dalis, laikant teigiamais 4 ir 5 balus.

4.4.3. Įrankio, išgautų reikalavimų bei proceso vertinimo rezultatai

Apklauskos rezultatų suvestinė pateikta 8 lentelėje. Vertinant teiginį, kad siūlomas automatizavimo įrankis palengvina naudotojo pasakojimų išgavimą iš *EventStorming* artefaktų, buvo gautas 4,48 vidurkis, 5 medianos reikšmė ir 0,77 standartinis nuokrypis. Dažniausiai pasirinktas įvertinimas buvo 5 balai, o 84,0% respondentų šį teiginį įvertino 4 arba 5 balais. Tai rodo, kad dauguma respondentų sutiko, jog įrankis padeda palengvinti reikalavimų išgavimo procesą.

Vertinant teiginį, kad įrankio automatiškai sugeneruoti naudotojo pasakojimai yra naudingi reikalavimų analizės procese, gautas 4,44 vidurkis, 5 medianos reikšmė ir 0,65 standartinis nuokrypis. 92,0% respondentų šį teiginį įvertino 4 arba 5 balais. Tai leidžia teigti, kad respondentai sugeneruotus naudotojo pasakojimus vertino kaip naudingus tolesniame reikalavimų analizės procese.

Aukščiausias vertinimų vidurkis buvo gautas teiginiui, kad šis įrankis turi praktinį pritaikymą projektuose, kuriuose naudojamas *EventStorming* metodas. Šio teiginio vidurkis siekė 4,56, mediana buvo 5, standartinis nuokrypis – 0,65, o 92,0% respondentų pasirinko 4 arba 5 balus. Tai rodo, kad dauguma respondentų siūlomą sprendimą vertina kaip potencialiai naudingą praktinėje veikloje.

Apibendrinant visų trijų teiginių rezultatus, bendras visų vertinimų vidurkis siekia 4,49, o 89,3% visų pateiktų įverčių sudaro 4 arba 5 balai. Taip pat svarbu pažymėti, kad nė vienas respondentas nepasirinko 1 ar 2 balų nei vienam teiginiui. Tai rodo nuosekliai teigiamą respondentų požiūrį į siūlomą įrankį, iš jo gaunamus reikalavimus ir patį automatizavimo procesą.

8 lentelė. Įrankio, išgautų reikalavimų bei proceso vertinimo apklauskos rezultatai

Vertinamas teiginys	Vidurkis	Mediana	Standartinis nuokrypis	Moda	4-5 balų dalis, %
Įrankis palengvina naudotojo pasakojimų išgavimą iš <i>EventStorming</i> artefaktų	4,48	5	0,77	5	84,0
Sugeneruoti naudotojo pasakojimai yra naudingi reikalavimų analizės procese	4,44	5	0,65	5	92,0
Įrankis turi praktinį pritaikymą projektuose, kuriuose naudojamas <i>EventStorming</i> metodas	4,56	5	0,65	5	92,0
Bendra visų vertinimų reikšmė	4,49	5	0,69	5	89,3

Papildomai buvo pastebėta, kad respondentai, turintys praktinės *EventStorming* patirties, siūlomą sprendimą vertino šiek tiek palankiau nei tie, kurie nurodė tik teorines žinias arba patirties neturėjo. Tačiau dėl nedidelės imties šie skirtumai negali būti interpretuojami.

Gauti rezultatai leidžia teigti, kad sukurtas įrankis respondentų yra vertinamas teigiamai: jis suvokiamas kaip palengvinantis naudotojo pasakojimų išgavimą iš *EventStorming* artefaktų, generuojantis praktiškai naudingus rezultatus ir turintis pritaikomumą realiuose projektuose.

Vis dėlto svarbu pažymėti, kad ši apklausa vertina respondentų suvokiamą naudą ir tinkamumą praktikoje, o ne tiesioginį sugeneruotų reikalavimų semantinį tikslumą.

4.5. Sprendimo taikymo rekomendacijos

4.5.1. Sprendimo pritaikymas

Įvairiose programinės įrangos kūrimo organizacijose, kurios naudoja *EventStorming* metodą reikalavimų išgavimui, dažnai *EventStorming* dirbtuves vykdo naudojant fizines priemones, tokias kaip lipnūs lapeliai ant lentos, arba laisvos formos skaitmeninius įrankius, tokie kaip *Miro*. Šiuo būdu sukurtas turinys nėra struktūruotas ir vėliau turi būti interpretuojamas veiklos analitiko, siekiant sudaryti reikalavimų sąrašą. Šis procesas yra neefektyvus, reikalaujantis daug laiko. Tiriamojo darbo metu sukurtas įrankis gali būti naudojamas siekiant palengvinti šį procesą automatizuojant reikalavimų išgavimą iš *EventStorming* artefaktų.

Su įrankiu galima sėkmingai išgauti reikalavimus iš *EventStorming* artefaktų, kurie yra:

- atitinka nurodytą spalvų paletę;
- aiškiai matomi ir be pašalinių elementų, nesusijusių su nagrinėjamu procesu;
- be stipriai persidengiančių elementų;
- su įskaitomu tekstu;
- logiškai išdėstyti (aiškiai atskiri įvykiai, komandos ar kiti elementai pagal *EventStorming* metodą);
- pakankamos kokybės (aukštos rezoliucijos).

Svarbu pažymėti, kad kiekviename proceso etape veiklos analitikas gali atlikti papildomus patikslinimus. Esant poreikiui, galima pridėti arba pašalinti artefaktus, jei jie nebuvo automatiškai aptikti, taip pat koreguoti jų tipą, jei jis buvo nustatytas neteisingai. Be to, galima keisti aptiktų artefaktų grupavimą, pavyzdžiui, priskiriant netinkamai identifikuotą elementą kitam loginiam klasteriui. Galiausiai, sugeneruoti reikalavimai gali būti redaguojami, siekiant ištaisyti netikslumus ar pritaikyti juos konkrečiam kontekstui.

Sugeneravus reikalavimus, jie pateikiami *Jira* struktūruotu formatu ir gali būti toliau naudojami projektavimo, programinės įrangos kūrimo etapuose. Rekomenduojama peržiūrėti sugeneruotus rezultatus ir prireikus juos pakoreguoti, siekiant užtikrinti atitikimą pradinei artefaktų informacijai.

4.5.2. Sprendimo trūkumai ir tobulinimas

4.5.3. Spalvų paletės pasirinkimas

Šiuo metu sistemoje naudojama spalvų paletė yra paremta dažniausiai *EventStorming* metodikoje taikomomis spalvomis, kurios atitinka skirtingus artefaktų tipus. Tačiau tiek metodikoje, tiek praktikoje nėra griežtai apibrėžta, kokios spalvos turi būti naudojamos – svarbiausia, kad pasirinktos spalvos būtų nuosekliai taikomos visų dirbtuvių kontekste. Dėl šios priežasties skirtinguose projektuose tie patys artefaktų tipai gali būti žymimi skirtingomis

spalvomis, tad prieš atliekant analizę spalvų paletė turi būti apibrėžiama individualiai kiekvienam projektui.

Šiuo metu spalvų paletė nustatoma rankiniu būdu, pasirenkant spalvas iš įkeltos *EventStorming* artefaktų nuotraukos naudojant spalvų parinkimo (angl. *color picker*) įrankį. Toks sprendimas leidžia pritaikyti sistemą konkrečiam atvejui ir sumažina konfigūravimo sudėtingumą veiklos analitikui, tačiau vis tiek reikalauja papildomo rankinio darbo ir yra jautrus naudotojo klaidoms, kadangi pasirinkus netinkamą spalvą – artefaktai praranda, pakeičia savo reikšmę ir tai lemia netikslius sugeneruotus reikalavimus.

Ateityje šis procesas galėtų būti tobulinamas automatizuojant spalvų paletės nustatymą. Remiantis artefaktuose esančia tekstine informacija ir jų tarpusavio ryšiais, sistema galėtų automatiškai identifikuoti galimus artefaktų tipus ir atitinkamai pasiūlyti spalvų priskyrimą. Taip pat būtų galima taikyti vaizdų analizės metodus, leidžiančius aptikti dominuojančias spalvas ir jas susieti su dažniausiai pasitaikančiais artefaktų tipais. Toks sprendimas sumažintų rankinio darbo poreikį, padidintų sistemos naudojimo patogumą bei sumažintų neteisingo spalvų priskyrimo tikimybę.

4.5.4. Artefaktų nuotraukos įkėlimas

Šiuo metu sistema leidžia įkelti tik vieną *EventStorming* artefaktų nuotrauką, kuri naudojama tolesniam apdorojimui. Tačiau praktikoje, analizuojant sudėtingesnes dalykines sritis ar didelės apimties sistemas, visa artefaktų lenta gali netilpti vienoje nuotraukoje. Dėl šios priežasties dalis informacijos gali būti prarasta arba analizuojama fragmentuotai, o tai apsunkina reikalavimų išgavimą.

Dabartinė sprendimo versija buvo stabiliai išbandyta su iki 45 lipnių lapelių viename *EventStorming* paveiksle. Esant didesniai artefaktų kiekiui, tikslinga lentą skaidyti į atskirus, logiškai susijusius procesus ir juos analizuoti atskirai.

Siekiant padidinti sprendimo pritaikomumą realiose situacijose, būtų tikslinga suteikti galimybę įkelti kelias artefaktų nuotraukas. Tokiu atveju veiklos analitikas galėtų jas sujungti į vieną bendrą vaizdą įrankio aplinkoje. Toks sprendimas leistų išlaikyti loginius ryšius tarp artefaktų, kurie *EventStorming* metu yra išdėstomi erdvėje, o jų tarpusavio atstumai turi semantinę reikšmę.

4.5.5. Lipnių lapelių aptikimo komponento tobulinimas

Nuo lipnių lapelių aptikimo komponento tiesiogiai priklauso automatinė tolimesnių analizės etapų kokybė. Nors bendri aptikimo modelio rezultatai F1 0,9232 (92,32%) rodo gerą veikimą, bendras atkūrimo (angl. *recall*) įvertis siekia 0,8869 (88,69%), todėl dalis artefaktų vis dar lieka neaptikti. Tai gali turėti neigiamą įtaką galutiniam reikalavimų išgavimui, nes prarandama dalis svarbios informacijos. Tuo tarpu preciziškumas yra aukštas – 0,9626 (96,26%), kas rodo, kad modelis retai klaidingai identifikuoja objektus kaip lipnius lapelius.

Viena pagrindinių priežasčių, lemiančių mažesnę atkūrimą, yra duomenų įvairovės trūkumas mokymo metu. Praktikoje *EventStorming* artefaktai dažnai būna įvairių formų: lipnūs lapeliai gali būti pasukti skirtingais kampais, dalinai atsiklijavę, susiglamžę ar nevienodai apšviesti.

Jei tokios variacijos nėra pakankamai atspindėtos mokymo duomenyse, modelis tampa mažiau jautrus netipiniams atvejams ir gali jų neaptikti.

Siekiant pagerinti lipnių lapelių aptikimo atkūrimą, ateityje tikslinga plėsti mokymo duomenų rinkinį įtraukiant didesnę artefaktų įvairovę. Ypač svarbu įtraukti pavyzdžius su skirtingomis lapelių orientacijomis, deformacijomis, daliniu uždengimu bei įvairiomis apšvietimo sąlygomis. Taip pat galima taikyti duomenų augmentacijos metodus, tokius kaip vaizdų pasukimas, mastelio keitimas ar triukšmo pridėjimas, siekiant dirbtinai padidinti duomenų įvairovę.

Papildomai galima tobulinti modelio parametrus siekiant padidinti atkūrimą, tačiau svarbu išlaikyti balansą su preciziškumu, kad nebūtų ženkliai padidintas klaidingai aptiktų objektų skaičius.

Apibendrinant galima teigti, kad nors esamas modelis pasižymi aukštu preciziškumu (96,26%) ir geru bendru F1 įverčiu (0,9232), atkūrimo reikšmė (88,69%) rodo, jog dalis artefaktų vis dar praleidžiama, todėl tolimesnis modelio tobulinimas turėtų būti orientuotas į šio rodiklio gerinimą.

Išvados

1. Apžvelgus reikalavimų inžinerijos procesą, jo taikymą PĮ kūrime, pastebėta, jog tai sudėtingas procesas reikalaujantis laiko ir pastangų. Išanalizuotas *Domain-Driven Design* sistemų kūrimo būdas, toks būdas padeda sukurti specializuotus programinės įrangos sprendimus, specifinei dalykinės srities problemai spręsti. Taip pat išanalizuotas ir *EventStorming* reikalavimų išgavimo metodas ir pastebėta, jog šis palengvina veiklos procesų identifikavimą, jų analizę, o tuo pačiu padeda identifikuoti *Domain-Driven Design* elementus.
2. Išanalizavus egzistuojančius problemos sprendimo būdus buvo nustatyta, jog sukurti įrankiai arba yra specializuoti konkrečioms dalykinėms sritims arba yra dalinai automatizuoti ir nėra pritaikyti veikti su *EventStorming* artefaktais, dėl to nuspręsta kurti automatizuotą reikalavimų išgavimo iš *EventStorming* artefaktų įrankį.
3. Remiantis atlikta analize ir egzistuojančiais sprendimo būdais buvo apibrėžti įrankio automatizuojančio reikalavimų išgavimą iš *EventStorming* artefaktų pagrindiniai reikalavimai. Numatyti galimi automatizavimui veiklos analitiko veiksmai susiję su *EventStorming* artefaktų poveikslo įkėlimu, reikalavimų generavimu, valdymu ir eksportavimu. Taip pat apibrėžti *EventStorming* metodo artefaktai, jų tarpusavio sąsajos.
4. Apibrėžta metodika automatizuotai išgauti reikalavimus iš *EventStorming* artefaktų. Nustatyta, kad šiai metodikai įgyvendinti reikalingas lipnių lapelių aptikimo dirbtinio intelekto modelis, artefaktų tipų nustatymo komponentas, teksto atpažinimo OCR komponentas, artefaktų klasterizavimo komponentas, naudotojo pasakojimų generavimo LLM agentas bei reikalavimų eksportavimo komponentas.
5. Automatizuoto reikalavimų išgavimo įrankis iš *EventStorming* artefaktų buvo sėkmingai suprojektuotas ir įgyvendintas panaudojant artefaktų atpažinimo, teksto atpažinimo, klasterizavimo, reikalavimų generavimo ir eksportavimo komponentus. Artefaktų atpažinimas realizuotas dviem etapais: lipnių lapelių aptikimui buvo pritaikytas mokymosi perkėlimas ir esamas *SSD MobileNet V1 FPN* modelis papildomai apmokytas 159 lipnių lapelių rinkiniu, o artefaktų tipų nustatymui taikytas spalvų gretinimas *LAB* spalvų erdvėje, naudojant *Delta E* metriką. Teksto atpažinimas įgyvendintas panaudojant *Google Vision API OCR* sprendimą. Klasterizavimas – taisyklėmis grįstu grafų analizės algoritmu. Reikalavimų generavimui panaudotas instrukcijomis pritaikytas *gpt-5.2* didelio masto kalbos modelis.
6. Atliktas eksperimentinis tyrimas parodė, kad sukurtas įrankis gali palengvinti reikalavimų išgavimą iš *EventStorming* artefaktų. Lipnių lapelių aptikimo eksperimente pasiektas aukštas bendras preciziškumas 0,9626, atkūrimas 0,8869 ir F1 įvertis 0,9232, todėl galima teigti, kad įrankis geba pakankamai tiksliai atpažinti lipnius lapelius *EventStorming* paveiksluose. Artefaktų tipų atpažinimo eksperimente nustatytas 94,03% bendras tikslumas, rodantis, kad *LAB* spalvų erdvėje ir *Delta E* metrika pagrįstas spalvų gretinimo algoritmas yra tinkamas *EventStorming* artefaktų tipams nustatyti. Įrankio, sugeneruotų reikalavimų ir viso proceso praktinis naudingumas buvo įvertintas respondentų apklausa, kurios rezultatai parodė teigiamą siūlomo sprendimo vertinimą: atskirų teiginių vidurkiai svyravo nuo 4,44 iki 4,56 balo iš 5, o 89,3% visų pateiktų įverčių sudarė 4 arba 5 balai. Tai leidžia teigti, kad sukurtas sprendimas ne tik techniškai veikia pakankamai tiksliai, bet ir yra vertinamas kaip praktiškai naudingas bei padedantis palengvinti reikalavimų išgavimo procesą iš *EventStorming* artefaktų.

Literatūros sąrašas

1. MELLIS, W.; LOEBBECKE, C. ir BASKERVILLE, R. Requirements uncertainty in contract software development projects. *Journal of Computer Information Systems*, t. 53 (2013), nr. 3, p. 97–108.
2. ZACHARIAH, B. ir NONYELUM, O. F. A comparative analysis of requirement gathering techniques. *IUP Journal of Computer Sciences*, t. 14 (2020), nr. 2, p. 7–32.
3. BRANDOLINI, A. Introducing EventStorming: An act of deliberate collective learning. *Leanpub: Victoria, BC, Canada* (2018).
4. AYSOLMAZ, B.; LEOPOLD, H.; REIJERS, H. A. ir DEMIRÖRS, O. A semi-automated approach for generating natural language requirements documents based on business process models. *Information and Software Technology*, t. 93 (2018), p. 14–29.
5. UR REHMAN, T.; KHAN, M. N. A. ir RIAZ, N. Analysis of requirement engineering processes, tools/techniques and methodologies. *International Journal of Information Technology and Computer Science (IJITCS)*, t. 5 (2013), nr. 3, p. 40.
6. PANDEY, D.; SUMAN, U. ir RAMANI, A. K. Performance measurement of different requirements engineering process models: a case study. *International Journal of Computer Engineering & Technology (IJCET)*, t. 1 (2010), nr. 2, p. 1–15.
7. MACAULAY, L. A. *Requirements engineering*. Springer Science & Business Media, 2012..
8. INAYAT, I.; SALIM, S. S.; MARCZAK, S.; DANEVA, M. ir SHAMSHIRBAND, S. A systematic literature review on agile requirements engineering practices and challenges. *Computers in Human Behavior*, t. 51 (2015), p. 915–929.
9. KASAULI, R.; KNAUSS, E.; HORKOFF, J.; LIEBEL, G. ir DE OLIVEIRA NETO, F. G. Requirements engineering challenges and practices in large-scale agile system development. *Journal of Systems and Software*, t. 172 (2021), p. 110851.
10. LANE, S.; O'RAGHALLAIGH, P. ir SAMMON, D. Requirements gathering: the journey. *Journal of Decision systems*, t. 25 (2016), nr. sup1, p. 302–312.
11. SINGH, V. ir PANDEY, D. Problems associated with requirement elicitation process: An overview. *International Journal of Advanced Research in Computer Science*, t. 8 (2017), nr. 5, p. 2637–2639.
12. SHARMILA, P. ir UMARANI, R. A walkthrough of requirement elicitation techniques. *International Journal of Engineering Research and Applications*, t. 1 (2011), nr. 4, p. 1583–1586.
13. KUŠTELEGA, M. ir MEKOVEC, R. A Systematic Analysis of Requirements Elicitation Problems and Challenges. *Iš: Proceedings of the Central European Conference on Information and Intelligent Systems, Varazdin, Croatia*, pp. 20–22.2023..
14. GEOGY, M. ir DHARANI, A. A scrutiny of the software requirement engineering process. *Procedia Technology*, t. 25 (2016), p. 405–410.
15. DO PRADO LEITE, J. C. S. ir DOORN, J. H. *Perspectives on Software Requirements*. Springer Science & Business Media, 2012..
16. *Volere Requirements Specification Template*. Prieiga per internetą: <https://www.volere.org/templates/volere-requirements-specification-template/> [žiūrėta 2025 m. kovo 29 d.].

17. IEEE Computer Society. Software Engineering Standards Committee ir IEEE-SA Standards Board. *IEEE recommended practice for software requirements specifications*. IEEE, 1998..
18. *Iso/iec/ieee 29148:2018*. Prieiga per internetą: <https://www.iso.org/standard/72089.html> [žiūrėta 2025 m. kovo 29 d.].
19. ATOUM, I.; BAKLIZI, M. K.; ALSMADI, I.; OTOOM, A. A.; ALHERSH, T. ir kt. Challenges of software requirements quality assurance and validation: A systematic literature review. *IEEE Access*, t. 9 (2021), p. 137613–137634.
20. POPOOLA, O. A.; ADAMA, H. E.; OKEKE, C. D. ir AKINOSO, A. E. Advancements and innovations in requirements elicitation: Developing a comprehensive conceptual model. *World Journal of Advanced Research and Reviews*, t. 22 (2024), nr. 01, p. 1209–1220.
21. EVANS, E. Domain-Driven Design: Tackling Complexity in the Heart of Software: Amazon. de: Eric J. Evans: Fremdsprachige Bücher. *vol*, t. 7873 (2003), p. 529.
22. VERNON, V. *Implementing domain-driven design*. Addison-Wesley, 2013..
23. MILLETT, S. ir TUNE, N. *Patterns, principles, and practices of domain-driven design*. John Wiley & Sons, 2015..
24. VERNON, V. *Domain-driven design distilled*. Addison-Wesley Professional, 2016..
25. KYRKOU, C. ir THEOCHARIDES, T. A parallel hardware architecture for real-time object detection with support vector machines. *IEEE Transactions on Computers*, t. 61 (2011), nr. 6, p. 831–842.
26. SUTHAHARAN, S. Modeling and algorithms. Iš: *Machine Learning Models and Algorithms for Big Data Classification: Thinking with Examples for Effective Learning*, p. 123–143. Springer, 2016..
27. *What is a Support Vector Machine?* . Prieiga per internetą: <https://se.mathworks.com/discovery/support-vector-machine.html> [žiūrėta 2026 m. gegužės 29 d.].
28. KORTLI, Y.; JRIDI, M.; AL FALOU, A. ir ATRI, M. A comparative study of CFs, LBP, HOG, SIFT, SURF, and BRIEF for security and face recognition. Iš: *Advanced Secure Optical Image Processing for Communications*, p. 13–22. IOP Publishing Bristol, UK, 2018..
29. PISNER, D. A. ir SCHNYER, D. M. Support vector machine. Iš: *Machine learning*, p. 101–121. Elsevier, 2020..
30. CHANDRA, M. A. ir BEDI, S. S. Survey on SVM and their application in image classification. *International Journal of Information Technology*, t. 13 (2021), nr. 5, p. 1–11.
31. LI, Z.; LIU, F.; YANG, W.; PENG, S. ir ZHOU, J. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, t. 33 (2021), nr. 12, p. 6999–7019.
32. ABOYOMI, D. D. ir DANIEL, C. A comparative analysis of modern object detection algorithms: YOLO vs. SSD vs. faster R-CNN. *ITEJ (Information Technology Engineering Journals)*, t. 8 (2023), nr. 2, p. 96–106.
33. CONG, S. ir ZHOU, Y. A review of convolutional neural network architectures and their optimizations. *Artificial Intelligence Review*, t. 56 (2023), nr. 3, p. 1905.

34. ZHAO, Z.; ZHENG, P.; XU, S. ir WU, X. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, t. 30 (2019), nr. 11, p. 3212–3232.
35. CHAUDHURI, A.; MANDAVIYA, K.; BADELIA, P. ir GHOSH, S. K. Optical character recognition systems. Iš: *Optical character recognition systems for different languages with soft computing*, p. 9–41. Springer, 2016..
36. LI, M.; LV, T.; CHEN, J.; CUI, L.; LU, Y. ir kt. Trocr: Transformer-Based Optical Character Recognition with Pre-Trained Models. Iš: *Proceedings of the AAAI conference on artificial intelligence*, pp. 13094–13102.2023..
37. NAZEEM, M.; ANITHA, R. ir NAVANEETH, S. Open-Source OCR Libraries: A Comprehensive Study for Low Resource Language. Iš: *Proceedings of the 21st International Conference on Natural Language Processing (ICON)*, pp. 416–421.2024..
38. *OCR Ranking 2025 – Comparison of the Best Text Recognition and Document Structure Software – Pragmile*. Prieiga per internetą: <https://pragmile.com/ocr-ranking-2025-comparison-of-the-best-text-recognition-and-document-structure-software/> [žiūrėta 2025 m. sausio 29 d.].
39. *OCR Benchmark: Text Extraction / Capture Accuracy*. Prieiga per internetą: <https://aimultiple.com/ocr-accuracy> [žiūrėta 2025 m. sausio 29 d.].
40. CUI, C.; SUN, T.; LIN, M.; GAO, T.; ZHANG, Y. ir kt. Paddleocr 3.0 technical report. *arXiv preprint arXiv:2507.05595* (2025).
41. *Azure Vision in Foundry Tools | Microsoft Azure*. Prieiga per internetą: <https://azure.microsoft.com/en-us/products/ai-foundry/tools/vision> [žiūrėta 2025 m. vasario 1 d.].
42. PRAKISYA, N. P. T.; KUSMANTO, B. T. ir HATTA, P. Comparative analysis of Google Vision OCR with Tesseract on newspaper text recognition. *Media of Computer Science*, t. 1 (2024), nr. 1, p. 31–46.
43. ALIERO, A. A.; BASHIR, S. A.; ALIYU, H. O.; TAFIDA, A. G.; BASHAR, U. K. ir kt. Systematic review on text normalization techniques and its approach to non-standard words (2023).
44. IZHAR, R.; BHATTI, S. N. ir ALHARTHI, S. A. Bridging precision and complexity: A novel machine learning approach for ambiguity detection in software requirements. *IEEE Access* (2025).
45. CHAI, C. P. Comparison of text preprocessing methods. *Natural language engineering*, t. 29 (2023), nr. 3, p. 509–553.
46. MINAEE, S.; MIKOLOV, T.; NIKZAD, N.; CHENAGHLU, M.; SOCHER, R. ir kt. Large language models: A survey. *arXiv preprint arXiv:2402.06196* (2024).
47. XU, K.; FENG, Y.; LI, Q.; DONG, Z. ir WEI, J. Survey on terminology extraction from texts. *Journal of Big Data*, t. 12 (2025), nr. 1, p. 29.
48. MARVIN, G.; HELLEN, N.; JJINGO, D. ir NAKATUMBA-NABENDE, J. Prompt Engineering in Large Language Models. Iš: *International conference on data intelligence and cognitive informatics*, pp. 387–402. Springer, 2023..
49. HUANG, L.; YU, W.; MA, W.; ZHONG, W.; FENG, Z. ir kt. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, t. 43 (2025), nr. 2, p. 1–55.

50. ABDULLAHI, S.; DANYARO, K. U. ir CHIROMA, H. The rise of hallucination in large language models: systematic reviews, performance analysis and challenges. *Cluster Computing*, t. 29 (2026), nr. 2, p. 124.
51. HAN, S.; WANG, M.; ZHANG, J.; LI, D. ir DUAN, J. A review of large language models: Fundamental architectures, key technological evolutions, interdisciplinary technologies integration, optimization and compression techniques, applications, and challenges. *Electronics*, t. 13 (2024), nr. 24, p. 5040.
52. WANG, Y.; MA, X.; ZHANG, G.; NI, Y.; CHANDRA, A. ir kt. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, t. 37 (2024), p. 95266–95290.
53. *MMLU-Pro Benchmark Leaderboard*. Prieiga per internetą: <https://artificialanalysis.ai/evaluations/mmlu-pro> [žiūrėta 2025 m. kovo 25 d.].
54. MASSEY, A. K.; EISENSTEIN, J.; ANTÓN, A. I. ir SWIRE, P. P. Automated Text Mining for Requirements Analysis of Policy Documents. *Iš: 2013 21st IEEE International Requirements Engineering Conference (RE)*, pp. 4–13. IEEE, 2013..
55. SURANA, C. S. R. K.; GUPTA, D. B. ir SHANKAR, S. P. Intelligent Chatbot for Requirements Elicitation and Classification. *Iš: 2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, pp. 866–870. IEEE, 2019..
56. ABAD, Z. S. H.; GERVASI, V.; ZOWGHI, D. ir BARKER, K. Elica: An automated tool for dynamic extraction of requirements relevant information. *arXiv preprint arXiv:1808.05857* (2018).
57. KADAM, K. D.; AHIRRAO, S. ir KOTECHA, K. [Retracted] Efficient Approach towards Detection and Identification of Copy Move and Image Splicing Forgeries Using Mask R-CNN with MobileNet V1. *Computational Intelligence and Neuroscience*, t. 2022 (2022), nr. 1, p. 6845326.
58. *Import Data from a CSV File*. Prieiga per internetą: <https://support.atlassian.com/jira-cloud-administration/docs/import-data-from-a-csv-file/> [žiūrėta 2025 m. kovo 26 d.].
59. KADAM, K.; AHIRRAO, S.; KOTECHA, K. ir SAHU, S. Detection and localization of multiple image splicing using MobileNet V1. *IEEE Access*, t. 9 (2021), p. 162499–162519.
60. LIN, T.; DOLLÁR, P.; GIRSHICK, R.; HE, K.; HARIHARAN, B. ir kt. Feature Pyramid Networks for Object Detection. *Iš: Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125. 2017..
61. SUJON, K. M.; HASSAN, R.; CHOI, K. ir SAMAD, M. A. Accuracy, precision, recall, f1-score, or MCC? empirical evidence from advanced statistics, ML, and XAI for evaluating business predictive models. *Journal of Big Data*, t. 12 (2025), nr. 1, p. 268.

Bendro pobūdžio LLM atsakymas į *EventStorming* naudotojo pasakojimų generavimo užklausą.

GPT-5.2 atsakymas

Based on the eventstorming board, this appears to describe a **newsletter subscription double opt-in flow** between a User, CRM, and Newsletter Marketing system.

Here are the user stories extracted from the workflow.

Epic: Newsletter Subscription Management

User Story 1 — Request Newsletter Signup

As a website visitor

I want to request to sign up for the newsletter

So that I can receive marketing communications.

Flow

1. User submits signup request.
2. CRM receives newsletter signup request.
3. Request policy validates the request.
4. System sends confirmation email.
5. CRM records that confirmation email was sent.

Acceptance Criteria

- User can submit an email address for signup.
- Signup request is stored in CRM.
- Validation policy checks request eligibility.
- Confirmation email is automatically sent.
- CRM logs the email delivery event.
- User does not become subscribed until confirmation is completed.

Events Identified

- Newsletter Signup Requested
- Request Policy Evaluated
- Confirmation Email Sent

Commands Identified

- Submit Signup Request
- Validate Signup Request
- Send Confirmation Email

- Record Confirmation Email Sent
-

User Story 2 — Confirm Newsletter Subscription

As a newsletter subscriber

I want to confirm my email address

So that my newsletter subscription becomes active.

Flow

1. User clicks confirmation link from email.
2. CRM receives email confirmation.
3. Confirmation policy validates confirmation.
4. System accepts subscription.
5. CRM records subscription acceptance.

Acceptance Criteria

- User can confirm subscription via email link.
- CRM receives and validates confirmation token.
- Confirmation policy verifies request validity.
- Subscription status changes to “Accepted”.
- CRM stores accepted subscription state.
- User is eligible to receive newsletters after acceptance.

Events Identified

- Confirmation Email Requested
- Email Confirmed
- Confirmation Policy Passed
- Subscription Accepted

Commands Identified

- Confirm Email
 - Validate Confirmation
 - Accept Subscription
 - Record Accepted Subscription
-

Domain Events (from the board)

1. Newsletter Signup Requested
 2. Confirmation Email Sent
 3. Email Confirmed
 4. Subscription Accepted
-

Suggested Aggregates

Subscription Aggregate

Responsible for:

- Signup requests
- Confirmation state
- Subscription activation

Possible States

- Pending Confirmation
 - Confirmed
 - Accepted
 - Rejected
-

Suggested Policies

Request Policy

Checks:

- Email validity
- Duplicate subscriptions
- Consent requirements

Confirmation Policy

Checks:

- Confirmation token validity
- Token expiration
- Already confirmed subscriptions

Gemini 3 atsakymas

Based on the EventStorming session captured in your image, here are the user stories organized by the two distinct workflows shown.

The color-coding in your image follows standard EventStorming conventions (Yellow for actors/events, Blue for commands, Pink for systems/aggregates, and Purple/Dark Pink for policies).

Workflow 1: Newsletter Signup Request

This workflow covers the initial interaction where a user expresses interest in the newsletter.

User Story: Initial Signup Request

As a prospective subscriber,

I want to submit a sign-up request for the newsletter,

so that I can begin the process of receiving marketing updates.

- **Acceptance Criteria:**

- The system (CRM) captures the "Newsletter signup requested" event.
 - The "Request Policy" is triggered immediately after the signup request is logged.
 - A "Send Confirm Email" command is issued to the CRM.
 - **Success Metric:** The user receives a confirmation email (Event: CONFIRM EMAIL SENT).
-

Workflow 2: Email Confirmation & Activation

This workflow covers the verification step required to finalize the subscription.

User Story: Confirming Email Address

As a user who has requested a subscription,

I want to confirm my email address,

so that my identity is verified and my subscription becomes active.

- **Acceptance Criteria:**

- The user interacts with the "Confirm Email" action (usually via a link).
 - The CRM records the "Email Confirmed" event.
 - The "Confirmation Policy" validates the event and triggers the "Accept Subscription" command.
 - **Success Metric:** The system status updates to "Subscription Accepted."
-

Technical Summary of the Flows

If you are handing these off to a development team, here is the functional logic derived from your sticky notes:

<i>Step</i>	<i>Actor / System</i>	<i>Action (Command)</i>	<i>Result (Event)</i>
1	User	Sign up Request	Newsletter signup requested
2	Policy	Trigger Request Policy	Send Confirm Email
3	CRM	Process Email	Confirm Email Sent
4	User	Confirm Email	Email Confirmed
5	Policy	Confirmation Policy	Accept Subscription
6	CRM	Finalize Record	Subscription Accepted

Naudotojo pasakojimų generavimo LLM agento instrukcija

Sisteminė (angl. *system*) instrukcija

You are an expert business analyst and EventStorming facilitator specializing in converting EventStorming workshops into actionable user stories and requirements.

Your task is to analyze EventStorming artifacts and generate comprehensive user stories following Agile best practices.

UNDERSTANDING EVENTSTORMING ARTIFACTS AND THEIR ROLES:

1. Actor: The person or role who initiates actions (e.g., 'User', 'Admin', 'Customer')
 - Maps to: The 'As a [role]' part of user stories
 - Usually leftmost in the flow (starting point)
2. Command: An intention or action the Actor wants to perform (imperative, e.g., 'Place Order', 'Create Account')
 - Maps to: The 'I want to [action]' part of user stories
 - Triggered by Actors or Policies
 - Causes Events to happen
3. System: External systems or services involved in the workflow
 - May execute Commands or produce Events
 - Important for technical acceptance criteria and integration points
4. Aggregate: Domain objects that enforce business rules and maintain consistency
 - Receives Commands and decides if they can succeed
 - Protects business invariants (e.g., 'Order', 'User Account', 'Shopping Cart')
 - Important for understanding business constraints in acceptance criteria
5. Event: Something that HAPPENED as a result (always past tense, e.g., 'Order Placed', 'Account Created', 'Payment Processed')
 - Maps to: Acceptance criteria (verify the outcome occurred)
 - Facts that cannot be changed
 - Trigger Policies and inform Read Models
6. Policy: Business rules or automated reactions (e.g., 'When order placed, send confirmation email', 'When payment fails, notify customer')
 - Maps to: Additional acceptance criteria (verify automated reactions happen)
 - Can trigger new Commands (creating workflow cycles)
 - Represents business logic and automation
7. Read Model: Data views, queries, or information displays (e.g., 'Order History', 'User Dashboard', 'Product Catalog')
 - Maps to: Information requirements that enable Actor decision-making
 - CRITICAL: Read Models provide the information Actors need to decide whether/how to execute Commands
 - Not just "what is shown" - but "what information enables the Actor to make informed decisions"
 - Example: 'Product Inventory' Read Model helps Customer decide whether to place an order
 - Example: 'Order Status' Read Model helps Customer decide whether to contact support or wait
 - Important for UI/UX requirements and preconditions in acceptance criteria

8. Hotspot: Issues, questions, risks, or areas needing discussion

- Maps to: Risks, open questions, or notes in user story description
- May indicate need for further refinement or technical spikes

EVENTSTORMING FLOW LOGIC:

- Actor → triggers → Command
- Command → processed by → System/Aggregate → produces → Event
- Event → triggers → Policy
- Policy → may trigger → new Command (creating cycles)
- Events → update → Read Models (for querying)
- Read Models → provide information to → Actor (enabling informed decisions about future Commands)
- IMPORTANT: Read Models close the feedback loop, allowing Actors to make data-driven decisions

UNDERSTANDING SPATIAL COORDINATES:

- Each artifact has a 'center_point' with normalized coordinates: $x \in [0, 1]$, $y \in [0, 1]$
- Coordinate system: $x = 0$ (left) → $x = 1$ (right), $y = 0$ (top) → $y = 1$ (bottom)
- EventStorming convention: workflows typically flow LEFT-TO-RIGHT (increasing x) and/or TOP-TO-BOTTOM (increasing y)
- Typical flow pattern: Actor (leftmost) → Command → System/Aggregate → Event → Policy → Read Model (rightmost)
- Artifacts with similar x -values but different y -values may represent parallel activities or alternative paths
- Artifacts very close in both x and y are tightly coupled and likely part of the same workflow step

HOW TO GENERATE USER STORIES FROM EVENTSTORMING:

1. IDENTIFY THE ACTOR(S):

- Find the Actor artifact(s) in the cluster (usually leftmost, lowest x -value)
- Each distinct Actor typically needs their own user story

2. IDENTIFY THE COMMAND(S):

- Find Command artifacts (what the Actor wants to do)
- Commands become the 'I want to [action]' part
- Use imperative form: 'place an order', 'create an account', 'update profile'

3. IDENTIFY THE BENEFIT:

- Look at Events (outcomes) and Read Models (data needed)
- Determine WHY the Actor wants to perform this action
- The benefit is the value delivered by the Events that result

4. CREATE ACCEPTANCE CRITERIA:

- Use natural business language to describe what needs to happen
- DO NOT force Given/When/Then format
- Events: Translate to natural business outcomes
- Policies: Express as natural automated behaviors
- Read Models: Describe what information becomes available to inform decisions
- Aggregates: State business rules naturally
- Focus on WHAT needs to happen, not HOW it's implemented

5. HANDLE COMPLEXITY AND STORY SPLITTING:

- DEFAULT: Prefer splitting into 2-4 smaller stories per cluster
- Split by layer, phase, actor, command, or system integration when appropriate
- Keep as a single story only when the flow is genuinely simple
- When in doubt, split into smaller focused stories

USER STORY FORMAT:

As a [Actor/Role], I want to [Command/Action], so that [Benefit based on Events/Outcomes].

Acceptance Criteria:

- [Natural language describing what needs to happen]
- [What outcomes occur]
- [What automated behaviors happen]
- [What information becomes available]
- [What business rules are enforced]

Technical Notes:

- [Systems involved]
- [Aggregates for business rules]
- [Hotspots as risks/questions]

CRITICAL RULES:

1. Generate 2-4 user stories per cluster by default.
2. Use spatial coordinates (x, y) to determine temporal sequence.
3. Each story must map to specific artifacts: Actor → Command → Events → Policies.
4. Make stories atomic.
5. Use past tense for Events in acceptance criteria.
6. Use imperative form for Commands in 'I want to'.
7. Include business value in the 'so that' clause.
8. Write acceptance criteria in natural business language.
9. Do NOT use Given/When/Then format.
10. ALWAYS use 'Technical Notes:' section.
11. Do NOT include 'Related Artifacts' sections.
12. Return results in valid JSON format.

AVOIDING HALLUCINATION - STAY GROUNDED IN THE DATA:

13. Use only the Actors explicitly listed in the cluster.
14. If the story's Actor uses a Read Model, write naturally; otherwise use passive voice.
15. Don't add validation rules unless they come from Aggregates, Policies, or Hotspots.
16. Every acceptance criterion should trace back to an artifact.
17. Do NOT create generic stories - be specific based on the actual artifacts and their spatial relationships.

Return ONLY a JSON object with this structure:

```
{
  "requirements": [
    {
      "name": "Short user story title",
      "description": "Complete user story in PLAIN TEXT format starting with 'As a...' (no 'User Story:' header, no markdown). Use section headers: 'Acceptance Criteria:' and 'Technical Notes:'"
    }
  ]
}
```

Naudotojo (angl. *user*) instrukcija

Below is an EventStorming session for the project "[PROJECT_NAME]".

CRITICAL CONTEXT:

- Artifacts are organized into clusters based on EventStorming flow rules and spatial proximity.
- Within each cluster, artifacts are definitively related by the clustering algorithm.
- Each cluster represents a cohesive business capability or user journey.

PROJECT CONTEXT:

[PROJECT_DESCRIPTION]

EVENTSTORMING DATA:

[JSON_INPUT]

YOUR TASK:

For each cluster, generate 2-4 user stories unless the cluster is genuinely too small to split.

Use the provided artifacts, their descriptions, and their spatial order to infer the flow.

Return only the JSON object requested by the system instructions.