Kauno technologijos universitetas

Informatikos fakultetas

Neteisingos kalbos ir lokalizacijos defektų paieškos mobiliosiose programose, analizuojant programos resursus, metodas ir tyrimas

Baigiamasis magistro projektas

Saulius Leškevičius

Projekto autorius

doc. dr. Šarūnas Packevičius

Vadovas

Kaunas, 2021

Kauno technologijos universitetas

Informatikos fakultetas

Neteisingos kalbos ir lokalizacijos defektų paieškos mobiliosiose programose, analizuojant programos resursus, metodas ir tyrimas

Baigiamasis magistro projektas

Programų sistemų inžinerija (6211BX011)

Saulius Leškevičius

Projekto autorius

doc. dr. Šarūnas Packevičius

Vadovas

prof. Eduardas Bareiša

Recenzentas

Kaunas, 2021

Kauno technologijos universitetas

Informatikos fakultetas

Saulius Leškevičius

Neteisingos kalbos ir lokalizacijos defektų paieškos mobiliosiose programose, analizuojant programos resursus, metodas ir tyrimas

Akademinio sąžiningumo deklaracija

Patvirtinu, kad:

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

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

3. įstatymų nenumatytų piniginių sumų už baigiamąjį projektą ar jo dalis niekam nesu mokėjęs;

4. suprantu, kad išaiškėjus nesąžiningumo ar kitų asmenų teisių pažeidimo faktui, man bus taikomos akademinės nuobaudos pagal Universitete galiojančią tvarką ir būsiu pašalintas iš Universiteto, o baigiamasis projektas gali būti pateiktas Akademinės etikos ir procedūrų kontrolieriaus tarnybai nagrinėjant galimą akademinės etikos pažeidimą.

Saulius Leškevičius

Patvirtinta elektroniniu būdu

Leškevičius, Saulius. Neteisingos kalbos ir lokalizacijos defektų paieškos mobiliosiose programose, analizuojant programos resursus, metodas ir tyrimas. Magistro baigiamasis projektas vadovas doc. dr. Šarūnas Packevičius; Kauno technologijos universitetas, informatikos fakultetas.

Studijų kryptis ir sritis: Informatikos mokslai, Programų sistemų inžinerija.

Reikšminiai žodžiai: programinės įrangos testavimas, grafinės vartotojo sąsajos testavimas, tekstiniai defektai, automatinis testavimas, rankinis testavimas.

Kaunas, 2021. 56 p.

Leškevičius, Saulius. Method and Research of Searching for Incorrect Language and Localization Defects in Mobile Apps by Analysing Application's Resources. Master's thesis in Software Engineering / supervisor doc. dr. Šarūnas Packevičius; The Faculty of Informatics, Kaunas University of Technology.

Study field and area: Computing, Software Engineering.

Keywords: Software Testing, Graphical User Interface Testing, Text Defects, Automated Testing, Manual Testing.

Kaunas, 2021. 56 pages.

Summary

Testing the graphical user interface manually is a time-consuming process. As a result, the system is not fully tested, and many human errors occur. To address this problem, tasks have been formulated to automatically detect incorrect language and localization defects in mobile applications by analyzing their resources.

The literature analysis included an overview of the field and solutions for bad spelling, mixed language, complex texts, difficult-to-read texts, offensive messages, synonyms, placeholders, and too-long text defects. After comparing these methods, the following solutions were chosen: dictionary, n-gram model, text readability index, color contrast calculation, text characteristics analysis, lexical database. The design part presents the essential parts of the technical documentation: scope of the work, system functions, functional and non-functional requirements, use cases, classes, activity and design diagrams, system data view. The chosen solutions were detailed in the research part by delving into the technologies and presenting the pseudocodes of their algorithms. For additional solutions to detect synonyms and offensive messages, we have selected logistic regression and neural networks. An experimental part was performed using the developed automated tool, during which the main problems of the methods were observed. We have provided the number of detected defects for each method and evaluated performance. To improve the efficiency of the tool, the text extraction algorithm has been improved, which has made it possible to increase the amount of data extracted from applications several times. Special data filters have been implemented to reduce the number of false-positive results. Validation of false-positive results of the chosen solution was performed, indicating the part of false results and the reasons causing such results. The results obtained were compared with the results of manual testing. The comparison showed the advantages of the automated tool and possible improvements in the future.

Turinys

Lentelių sąrašas

Paveikslų sąrašas

1. Įžanga

1.1. Dokumento paskirtis

Šio dokumento paskirtis – pateikti programų sistemų inžinerijos, magistro studijų metu sukurto automatinio testavimo įrankio, taikant neteisingos kalbos ir lokalizacijos defektų paieškos metodus, tyrimą. Dokumento rengimo metu buvo siekiama išanalizuoti šių defektų sprendimo būdus bei pasirinkti tinkamiausius įrankio kūrimui. Projektinėje dalyje pateikiami esminiai įrankio dokumentacijos aspektai. Tyrimo metu norima įsigilinti į pasirinktų metodų technologijas ir realizaciją. Realizuoto įrankio pagalba buvo siekiama eksperimentiškai įvertinti kiekvieno metodo efektyvumą, lyginat su rankinio testavimo metu pateiktais rezultatais bei pateikti pagrindines problemas su kuriomis buvo susidurta automatinio testavimo metu. Išvadose nurodomi svarbiausi pagrindinių dalių pastebėjimai, kurie bus naudingi tolimesniems tyrimams susijusiems su neteisingos kalbos ir lokalizacijos paieškos metodais.

1.2. Santrauka

Grafinės vartotojos sąsajos testavimas rankiniu būdu yra ilgai užtrunkantis procesas. Dėl šios priežasties sistema nėra ištestuojama pilnai ir atsiranda daug žmogiškųjų klaidų. Šiai problemai spręsti buvo suformuluoti uždaviniai, kurių tikslas yra automatiškai aptikti neteisingos kalbos ir lokalizacijos defektus mobiliosiose programėlės analizuojant jos resursus.

Literatūros analizės metu buvo atlikta srities apžvalga bei išanalizuoti blogos rašybos, mišrios kalbos, sudėtingų tekstų, sunkiai įskaitomų tekstų, įžeidžiančių pranešimų, sinonimų, neužpildytos vietos žymenų ir per ilgų tekstų defektų sprendimo būdai. Atlikus šių metodų palyginimą buvo pasirinkti šie sprendimo būdai: žodyno taikymas, n-gramų modelis, teksto sudėtingumo indeksas, spalvų kontrasto skaičiavimas, teksto charakteristikų analizavimas, leksikos duomenų bazė. Projektinėje dalyje pateiktos esminės techninės dokumentacijos dalys: veiklos kontekstas, sistemos funkcijos, funkciniai ir nefunkciniai reikalavimai, panaudos atvejų, klasių, veiklos ir diegimo diagramos, sistemos duomenų vaizdas. Pasirinkti sprendimo būdai buvo detalizuoti tyrimo dalyje pasigilinant į technologijas bei pateikiant jų algoritmų pseudokodus. Taip pat tyrimo metu buvo išanalizuoti papildomi sprendimo būdai sinonimų ir įžiedžiančių pranešimų defektams aptikti. Jiems aptikti buvo pasirinkta logistinė regresija ir neuroninis tinklas. Naudojantis sukurtu automatiniu įrankiu buvo atliekama eksperimentinė dalis, kurios metu buvo pastebėtos esminės metodų problemos. Pateiktas kiekvieno metodo aptiktų defektų kiekis bei greitis. Norint pagerinti įrankio efektyvumą buvo pagerintas teksto ištraukimo algoritmas, kuris leido kelis kartus padidinti iš programėlių ištrauktą duomenų kiekį. Buvo parašyti specialūs duomenų filtrai siekiant sumažinti klaidingai teigiamų rezultatų kiekį. Atlikta pasirinkto sprendimo klaidingai teigiamų rezultatų validacija, nurodant klaidingų rezultatų dalį, bei priežastis sukeliančias tokius rezultatus. Gauti rezultatai buvo palyginti su rankinio testavimo rezultatais. Atlikus palyginimą buvo pateikti automatinio įrankio privalumai bei galimi patobulinimai ateityje.

2. Analitinė dalis

2.1. Įvadas

Pastaraisiais dešimtmečiais yra pastebimas drastiškai spartėjantis naujų technologijų kūrimas ir taikymas. Šie spartūs technologiniai pokyčiai veikia beveik visas ekonomikos, visuomenės ir kultūros sritis. Todėl natūralu, jog susidarius tokiai milžiniškai konkurencijai, užsakovai reikalauja nuolatinio spartaus naujų funkcijų diegimo, norėdami išlikti paklausūs rinkoje. Tačiau išlaikyti aukštus tempus kuriant programinę įrangą yra nelengva, tai yra sudėtingas uždavinys, juolab norint, kad programinės įrangos kokybė atitiktų aukščiausius standartus. Norint užtikrinti produkto kokybę, programinės įrangos testavimas tampa neatsiejama programinės įrangos kūrimo gyvavimo ciklo dalis.

Programinės įrangos testavimas yra kritinės analizės procesas, kuris padeda identifikuoti ir įvertinti ar kuriamas produktas tenkina iškeltus reikalavimus. Tai yra nenutrūkstamas procesas, įvardijamas kaip programinės įrangos testavimo gyvavimo ciklas.

Testavimas padeda ne tik pristatyti programinę įrangą be klaidų, bet ir pagerinti jau esamą jos funkcionalumą. Dažniausiai testavimo svarba atsiskleidžia stebint vartotojų atsiliepimus. Nuolatos gerindami vartotojo patirtį, naudojantis programine įranga, ne tik keliame vartotojo pasitikėjimą produktu, bet ir optimizuojame verslą, kadangi kokybiško produkto priežiūros išlaidos yra mažesnės.

Vienas iš programinės įrangos testavimo tipų yra grafinės vartotojo sąsajos testavimas. Šio testavimo tikslas yra užtikrinti programinės įrangos darbo funkcionalumą pagal specifikacijas, patikrinant ekranus ir valdiklius, tokius kaip meniu, mygtukai, piktogramos ir kt. Testuojamos programėlės rankiniu būdu dažnai būna neištestuotos iki galo, kadangi rankinis testavimas yra daug resursų reikalaujantis procesas. Taip pat testuojant rankiniu būdu padidėja žmogiškųjų klaidų tikimybė. Taigi norint pagerinti testavimo kokybę bei pagreitinti jo procesą reikia šiuos procesus automatizuoti.

2.2. Tikslas

Mobiliųjų programėlių testavimas rankiniu būdu yra daug resursų reikalaujantis procesas, kuris ne tik užtrunka daug laiko, bet ir reikalauja nemažai patirties šioje srityje. Taip pat testuojant rankiniu būdu yra sunku išvengti žmogiškųjų klaidų.

Darbo tikslas – automatiškai aptikti neteisingos kalbos ir lokalizacijos defektus mobiliosiose programėlėse, analizuojant programos resursus.

Uždaviniai:

- išanalizuoti neteisingos kalbos ir lokalizacijos defektų aptikimų sprendimo būdus;
- išsirinkti geriausius metodus, juos lyginant tarpusavyje;
- sukurti automatinį testavimo įrankį;
- eksperimentiškai įvertinti metodų efektyvumą, lyginant su rankiniu testavimu.

2.3. Srities apžvalga

Programinės įrangos testavimas yra apibrėžiamas kaip veikla, skirta patikrinti ar gauti rezultatai atitinka numatytus ir įsitikinti, kad programinėje įrangoje nėra defektų. Programinės įrangos testavimas taip pat padeda nustatyti klaidas, spragas ar trūkstamus reikalavimus. Testavimas turi

didelę reikšmę įmonėms ir jų verslui užtikrinant kokybę. Visų pirma aukštos kokybės produktas yra svarbus klientui, nes tai suteiks vartotojui geresnę patirtį naudojantis produktu. Už kokybišką produktą klientai tikrai sumokės daugiau. Dar svarbiau yra tai, kad parduodama aukštos kokybės produktą, įmonė susikuria stiprią reputaciją ir gerą prekinio ženklo įvaizdį. Ilgainiui testavimas padės sutaupyti pinigų, kadangi nereikės nuolatos taisyti klaidų [1].

Testavimas yra skirstomas į šiuos tipus [2]:
1. baltos dėžės testavimas;
2. pilkos dėžės testavimas;
3. juodos dėžės testavimas.

Baltos dėžės, kitaip dar žinomas kaip skaidrios dėžės arba struktūrinis testavimas, yra apibrėžiamas kaip programinės įrangos vidinės struktūros, projektavimo ir kodavimo testavimas. Tokio testavimo būdo programinės įrangos kodas yra prieinamas testuotojui. Šiame tipe pirmiausia dėmesys skiriamas programėles įvesties ir išvesties srautų tikrinimui, stiprinant projektavimą, panaudojamumą bei saugumą. Testuotojas pasirenka tokias įvestis, kad sistema veiktų tam tikru algoritmo keliu ir numato, kokių išėjimų jis tikisi [3] (žr. 1 pav.).

1 pav. Baltos dėžės testavimo atvaizdavimas

Privalumai:
- kodo optimizavimas ieškant paslėptų klaidų;
- testavimo atvejai gali būti lengvai automatizuojami;
- testavimas yra nuodugnesnis, nes paprastai visi kodo keliai yra padengiami.

Trūkumai:
- testavimas gali būti gana sudėtingas ir brangus.
- užimantis daug laiko, kuo didesnė programa, tuo daugiau laiko reikės, norint ją visiškai ištestuoti.

Baltos dėžės testavimo technikos [4]:
1. Valdymo srauto testavimas (angl. *Control Flow*): tai yra struktūrinio testavimo strategija, kuri naudoja programos valdymo srautą kaip modelio valdymo srautą ir teikia pirmenybę ištestuoti daugiau paprastų kelių, negu sudėtingesnių ir mažiau kelių.
2. Atšakų testavimas (angl. *Branch Testing*): atšakų testavimo tikslas yra patikrinti kiekvieną variantą kiekviename sąlygos sakinyje.

3. Pagrindinio kelio testavimas (angl. *Basis Path Testing*): tai metodas, skirtas vykdyti programos kodo visus kelius arba pasirinktą vieną kelią. Tai padeda nustatyti visus gedimus, esančius kode.
4. Duomenų srauto testavimas (angl. *Data Flow Testing*): šiuo testavimo būdu valdymo srauto diagrama yra paaiškinta kaip programos kintamieji yra apibrėžti ir naudojami.
5. Ciklų testavimas (angl. *Loop Testing*): šis metodas sutelkia visą dėmesį į ciklų konstrukcijų pagrįstumą.

Pilkos dėžės testavimas yra apibrėžiamas kaip testavimo būdas, kuriuo metu testuotojas turi žinoti vidines duomenų struktūras ir algoritmus, sudarant testinius atvejus. Šis testavimas yra baltos dėžės ir juodos dėžės testavimų būdų derinys. Pilkos dėžės testavimas suteikia galimybę ištestuoti abi programos puses, atvaizdavimo sluoksnį ir kodo dalį [5] (žr. 2 pav.).

2 pav. Pilkos dėžės testavimo atvaizdavimas

Privalumai:
- testuotojai remiasi sąsajos apibrėžimu ir funkciniais reikalavimais, o ne programos kodu;
- testuotojai gali turėti mažiau programavimo patirties lyginant su baltos dėžės testavimu;
- testavimas atliekamas labiau vartotojo požiūriu, nei dizainerio požiūriu.

Trūkumai:
- gali praleisti kritines klaidas programiniame kode;
- aptikus defektą gali būti sunku aptikti tikslų jo šaltinį, lyginant su baltos dėžės testavimu;
- dažniausiai programa yra mažiau padengiama testais, negu atliekant baltos dėžės ir juodos dėžės testavimus atskirai.

Pilkos dėžės testavimo metodai [6]:
1. Stačiakampio masyvo metodas (angl. *Orthogonal Array Testing*): šis testavimo būdas naudoja visų galimų kombinacijų pogrupį.
2. Matricos testavimas (angl. *Matrix Testing*): atliekant matricos testavimą, nurodoma projekto būsenos ataskaita.
3. Regresijos testavimas (angl. *Regression Testing*): jei programinėje įrangoje daromo nauji pakeitimai, regresijos testavimas reiškia testinių atvejų vykdymą.
4. Šabloninis testavimas (angl. *Pattern Testing*): šabloninis testavimas patikrina, ar jo architektūra ir dizainas yra tinkami.

Juodos dėžės testavimas yra apibrėžiamas kaip testavimo būdas, kurio metu išbandomas programos funkcionalumas, neturint žinių apie vidinę kodo struktūrą. Čia visas dėmesys yra sutelkiamas į

programinės įrangos sistemos įvestis ir išvestis ir viskas grindžiama programinės įrangos reikalavimais ir specifikacijomis. Atlikdamas testavimą, testuotojas neturi prieigos prie programos kodo, bet privalo gerai žinoti sistemos architektūrą [7] (žr. 3 pav.).

3 pav. Juodos dėžės testavimo atvaizdavimas

Privalumai:

- efektyvus dideliems kodo segmentams;
- testavimas lengviau suvokiamas testuotojams;
- programuotojai ir testuotojai yra nepriklausomi vienas nuo kito.

Trūkumai:

- testinius atvejus sunku suprojektuoti be aiškių ir glaustų specifikacijų;
- tik pasirinktas skaičius testavimo scenarijų yra iš tikrųjų įvykdomi. To pasekmė yra ribotas kodo padengimas.

Juodos dėžės testavimo technikos [2]:

1. Ekvivalenčių klasių metodas (angl. *Equivalence Partitioning*): tai gali sumažinti testavimo atvejų kiekį, nes yra padalijama aibė testavimo sąlygų į dalis, kurios gali būti laikomos tomis pačiomis.
2. Ribinės vertės analizė (angl. *Boundary Value Analysis*): daugiau dėmesio skiriama ribų testavimui, arba ten, kur pasirenkamos kraštutinės ribinės vertės. Tai apima minimumą, maksimumą, vidines / išorines ribas, klaidų vertes ir ribines vertes.
3. *Fuzzing*: testavimo rūšis, kai automatiniai arba pusiau automatiniai testavimo būdai yra naudojami programinėje įrangoje, operacinėse sistemose arba tinkluose aptikti kodavimo klaidas ir saugumo spragas.
4. Priežasties ir pasekmės diagrama (angl. *Cause-Effect Graph*): tai testavimo technika, kurioje testavimas pradedamas sudarant grafiką ir nustatant ryšį tarp pasekmės ir jos priežasčių.
5. Stačiakampio masyvo metodas (angl. *Orthogonal Array Testing*): gali būti taikomas problemoms, kuriuose įvesties sritis yra palyginti maža, bet per didelė, kad būtų galima atlikti išsamius bandymus.
6. Porinis testavimas (angl. *All Pair Testing*): čia testavimo atvejai yra projektuojami taip, kad įvykdytų visus įmanomas diskrečias kombinacijas kiekvienai įvestai parametrų porai. Pagrindinis tikslas yra turėti rinkinį testavimo atvejų, kurie apima visas poras.
7. Būsenų perėjimo testavimas (angl. *State Transition Testing*): testavimo technika, kai įvesties sąlygų pokyčiai sukelia būsenos pokyčius bandomojoje programoje.

Juodos dėžės testavimas dar skirstomas į funkcinį ir nefunkcinį testavimą. Funkcinis testavimas yra tai, ką iš tikrųjų daro sistema. Norėdami patikrinti, ar kiekviena programinės įrangos funkcija veikia taip, kaip nurodyta reikalavimų dokumente. Testuojamas visas funkcionalumas, pateikiant tinkamą įvestį, siekiant patikrinti, ar tikrasis išėjimas atitinka numatytą išėjimą.

Nefunkcinis testavimas nusako, kaip gerai sistema veikia. Jis susijęs su įvairiais programinės įrangos aspektais, tokiais kaip našumas, apkrova, stresas, plečiamumas, saugumas, panaudojamumas ir t. t. [8]. Pagrindinis dėmesys skiriamas GUI testavimui. GUI yra kompiuterio programinės įrangos interaktyvių vaizdinių komponentų sistema, rodanti objektus, kurie perteikia informaciją ir nurodo veiksmus, kurių gali imtis vartotojas. GUI testavimas apima ekranų valdiklių tikrinimą, tokių kaip meniu, mygtukai, piktogramos ir visų tipų juostos – įrankių juosta, meniu juosta, dialogo langai [9]. GUI testavime yra keletas defektų klasifikavimų metodų, tokie kaip paminėti [10] [11] [12], jie identifikuoja įprastus defektų tipus, tokius kaip duomenys, sąsaja, logika, funkcija ir dokumentacija. Sąsajos defektų tipai yra suskirstyti į vartotojo sąsajos ir komponentų sąsajos defektų tipus. Pagal Lelli ir kitų autorių pateiktą vartotojo sąsajos defektų klasifikavimo schemą [13], defektai skirstomi į dvi kategorijas: grafinės struktūra ir estetika bei duomenų pateikimas. Remdamiesi defektų klasifikacija [14] [15] [16], mes sutelkiame dėmesį į vartotojo sąsajos defektus atvaizdavimo kategorijoje [17]. Atvaizdavimo kategorijoje buvo nagrinėjami neteisingos kalbos ir lokalizacijos defektai, kurie yra pateikiami žemiau toliau lentelėje (žr. 1 lentelė).

1 lentelė. Analizuojami defektai

Defektas	**Aprašymas**	**Pavyzdys**
Bloga rašyba	Pranešimai turi gramatikos arba rašybos klaidų.	„Labas pasaul“
Mišri kalba	Programėlės langai turi pranešimus su keliomis kalbomis, pranešimai rodomi maišyta klaba.	„Sveikas vartotojau. Continue verification process by pressing button down below“
Sunkiai skaitomas tekstas (sudėtingas)	Pranešimai yra parašyti kalba, kuri nelengvai suprantama jos vartotojams.	„Artificial neural networks (ANNs) were inspired by information processing and distributed communication nodes in biological systems“
Sunkiai skaitomas tekstas (spalvų konfliktavimas)	Teksto spalva konfliktuoja su fono spalva ar paveikslėliu, todėl tekstas tampa sunkiai įskaitomas arba visiškai neįskaitomas.	8.1 Exceapt for the Contract signed for a definite term that can only be terminated in accordance with articles 8.2 and 8.3 here below or due to serious breach of either Party prior to the expiration of the term, the Contract may be terminated by either party at any time, with or without cause, by giving written notice to the other party not less than 30 days before the effective date of termination. Should the Contract be signed for a definite period of time and the Service type is of a repetitive
Įžeidžiantys pranešimai	Programėlės pranešimai turi teksto, kuris yra parašytas netaktiškai arba įžeidžiančia maniera. Tokio tipo pranešimai paprastai pateikiami klaidos pranešimuose.	„You are dumb. Try again!“
Sinonimai	Skirtingi žodžiai yra vartojami apibūdinti tą patį objektą.	„Termination time is very short. Look at expiration date“
Neužpildytos vietos žymenos	Teksto pranešimai turi raktinius žodžius, kurie turi būti pakeisti per programėlės paleidimo laiką, tačiau jie yra palikti neužpildyti ir atvaizduoti raktiniai žodžiai vietoje pranešimų.	„Some dummy text“

Per ilgi tekstai	Pranešimai yra tokie ilgi, kad net vartotojas nesivargina jų skaityti.	„I hope that a study of very long sentences will arm you with strategies that are almost as diverse as the sentences themselves, such as: starting each clause...“

2.3.1. Defektų aptikimo būdai

Testavimo būdai gali būti išskirti į automatinį ir rankinį testavimą. Rankinis testavimas, tai programinės įrangos testavimas, kur testai yra vykdomi rankiniu būdu, kokybės užtikrinimo analitikų. Jis atliekamas programinės įrangos kūrimo metu norint aptikti klaidas. Testuotojas patikrina svarbiausias sistemos funkcijas. Šio proceso metu testavimo atvejai ir ataskaitų sudarymai yra atliekami be jokio automatinio įrankio pagalbos. Automatinio testavimo metu, testuotojai rašo testavimo scenarijus, jog automatizuotų testų vykdymą. Testuotojai naudoja tam tikrus automatinius įrankius, kurie leidžia kurti testavimo scenarijus ir atlikti programinės įrangos validaciją. Automatinis testavimas visiškai priklauso nuo iš anksto parašyto testo, kuris yra paleidžiamas automatiškai, siekiant palyginti gaunamus rezultatus su tikėtinais rezultatais. Tai padeda nustatyti ar programa veikia taip kaip tikimasi. Toks testavimo būdas leidžia atlikti regresijos testus ir vykdyti tą patį uždavinį pakartotinai be rankinio testavimo įsikišimo. Nors visi procesai yra vykdomi automatiškai, tačiau norėdami sukurti pradinius testavimo scenarijus reikia įdėti šiek tiek pastangų, kad jie būtų automatizuoti. Pagrindiniai skirtumai tarp rankinio ir automatinio testavimo yra šie [18]:

- rankinį testavimą atlieka kokybės užtikrinimo analitikas (žmogus), o automatinis testavimas yra atliekamas naudojant scenarijus, programinį kodą ir automatinius įrankius (kompiuterį);
- rankinis testavimas yra netikslus, kadangi padaugėja žmogiškųjų klaidų tikimybė, tuo tarpu automatizavimo procesas yra patikimas, nes jis pagrįstas programiniu kodu ir scenarijais;
- rankinis testavimas yra daug laiko reikalaujantis procesas, o automatinis testavimas yra labai greitas;
- rankinis testavimas įmanomas ir neturint patirties programavimo srityje, tačiau automatiniame testavime patirtis programavimo srityje yra būtina;
- rankinis testavimas leidžia atlikti atsitiktinius bandymus, o automatikos testavimas neleidžia atsitiktinių bandymų.

2.3.1.1. Defekto „Bloga rašyba“ aptikimo metodai

Metodų sąrašas:

1. žodyno taikymas;
2. „Java“ rašybos tikrinimo įrankis „LanguageTool“;
3. „Java“ teksto paieškos variklio biblioteka „Lucene“.

2.3.1.1.1. Žodyno taikymas

Žodyno taikymas yra primityviausias metodas aptinkant blogos rašybos defektus. Kiekvienai kalbai galima rasti žodyną, kurį galima pritaikyti. Taikant žodyną yra naudojami algoritmai, kurie lygina kiekvieną žodį su tūkstančiais teisingai parašytų žodžių. Jei žodžiai yra teisingi, bet neįtraukti į žodyną, tai norėdami, kad metodas veiktų teisingai ir nebūtų rodomos klaidos, reikia naudojamą žodyną patobulinti papildant jį šiais žodžiais. Tokie žodžiai dažniausiai būna vardai, pavadinimai ar skaičiai.

2.3.1.1.2. „Java“ rašybos tikrinimo įrankis „LanguageTool“

„LanguageTool“ yra atviro kodo stiliaus ir gramatikos korektūros programinė įranga, kuri palaiko 31 kalbą. Šį įrankį galima nesunkiai įsidiegti „Maven“ projekte papildant „pom.xml“ failą atitinkama priklausomybe. Jį taip pat galima įsidiegti kaip papildinį daugelyje naršyklių ir elektroninio pašto programėlėse [19]. Naudojant šį įrankį kaip papildinį, vartotojas matys visas gramatikos klaidas realiu laiku ir iškarto gaus pasiūlymą kaip tą klaidą ištaisyti. Skirtingai nuo įprastų rašybos tikrinimo įrankių, šis įrankis geba aptikti tokias klaidas kaip pasikartojantys žodžiai ar nustatyti jei vartotojas sumaišė tam tikrus angliškus žodžius ar artikelius tarpusavyje, pavyzdžiui, „there/their“ ir „a/an/the“ [20].

2.3.1.1.3. „Java“ teksto paieškos variklio biblioteka „Lucene“

Tai atviro kodo biblioteka, kuri leidžia indeksuoti dokumentus ir yra skirta pilno teksto paieškai. Šis įrankis turi ne tik rašybos tikrinimo funkciją, bet ir kitas:

- indeksavimas;
- paieška;
- ženklinimas (angl. *tokenization*).

Ši biblioteka ypatinga tuo, kad naudoja ne klasikinius indeksus, o atvirkštinius. Klasikiniuose indeksuose kiekvienam dokumentui renkame visą dokumente esančių žodžių ar terminų sąrašą. Atvirkštiniuose indeksuose kiekvienam visų dokumentų žodžiui yra saugoma informacija kokiame dokumente ir pozicijoje galima rasti šį žodį ar terminą. Taip pat atvirkštinius indeksus lengviau prižiūrėti [21].

2.3.1.2. Defekto „Mišri kalba“ aptikimo metodai

Metodų sąrašas:

1. „Java“ biblioteka, skirta kalbos nustatymui „Lingua“;
2. „Java“ biblioteka „language-detector“.

2.3.1.2.1. „Java“ biblioteka, skirta kalbos nustatymui „Lingua“

„Lingua“ yra kalbų aptikimo biblioteka, tinkama tiek ilgam, tiek trumpam tekstui. Ji palaiko 74 kalbas. Kaip ir kiekvienas kalbos detektorius naudoja tikimybinį n-gramų modelį. Taip pat ši biblioteka pirmiausia ieško unikalių simbolių vienoje ar keliose kalbose. Taip galima kai kurias kalbas aptikti iškarto. Šią biblioteką galime naudoti „Maven“ projekte, reikia tik papildyti „pom.xml“ failą priklausomybe [22].

2.3.1.2.2. „Java“ biblioteka „Language-detector“

Tai kalbos nustatymo atvirojo kodo biblioteka skirta „Java“ programavimo kalbai. Ji palaiko 71 kalbą. Kaip ir „Lingua“ ši biblioteka iš teksto ištraukia n-gramus. N-gramai buvo sudaryti iš Vikipedijos straipsnių bei socialinio tinklo „Twiter“ pranešimų. „Language- detector“ geriau veikia su ilgesniais tekstais, tad yra netinkama trumpesniems tekstams [23]. Kaip ir „Lingua“ šią biblioteka galime lengvai įsidiegti „Maven“ projekte.

2.3.1.3. Defekto „Sunkiai skaitomas tekstas (Sudėtingas)“ aptikimo metodai

Yra daugybė teksto skaitomumo indeksų, bet analizei buvo pasirinkti patys populiariausi:
1. didėjantis rūko indeksas;
2. smogo indeksas;
3. „java_fathom“ biblioteka.

2.3.1.3.1. Didėjantis rūko indeksas

Didėjantis rūko indeksas (angl. *Gunning fog index*) matuoja skaitomumą pagal sakinio ir žodžio ilgį. Apskaičiavus rūko indeksą pagal formulę (2.1) gaunama nuo 0 iki 20 [24]. Ši vertė nustato formaliojo ugdymo metus, kurių skaitytojui reikia norint suprasti tekstą iš pirmo karto. Jei teksto dalies skaitomumo balas yra 6, tada tai turėtų lengvai perskaityti tie, kurie mokėsi mokyklose iki 6 klasės. Tekstas, kurį skaito plačioji visuomenė, turėtų siekti maždaug 8 balus. Skaitymo lygį reikėtų laikyti iki 9 ar žemesnį, nes visuomenės leidiniai, kuriuos kasdien skaitome, yra parašyti būtent šiuo lygiu (žr. 2 lentelė) [25].

$$0.4\left(\left(\frac{\check{z}odžiai}{sakiniai}\right) + 100\left(\frac{sudėtingi\ \check{z}odžiai}{\check{z}odžiai}\right)\right); \quad (2.1)$$

2 lentelė. Rūko indekso lygių paaiškinimas

Rūko indeksas	Skaitymo lygis klases	Skaitymo lygis pagal leidinius
20	Magistras ir didesnis išsilavinimas	Vyriausybės informacija
17-20	Antrosios studijų pakopos	Akademiniai žurnalai
16	Kolegijos baigiantysis	Standartinės medicininio sutikimo formos (norint juos iššifruoti, jums nereikės medicininio laipsnio!)
13-15	Kolegijos pirmakursis, antrakursis,	Nei vienas populiarus vartotojų leidinys nėra toks sunkus.
11-12	Abiturientai	„Harper’s“, „Time“, „Atlantic Monthly“, „Newsweek“, „The Wall Street Journal“
10	Gimnazistai 10-11kl.	„National Geographic“
9	Gimnazistai 9kl.	Leidinių santrauka
8	8 klasė	Moterų namų žurnalas
7	7 klasė	TV gidas, Biblija, Markas Tvenas
6	6 klasė	„Žmonės“

2.3.1.3.2. Smogo indeksas.

Smogo indeksas įvertina, kiek metų išsilavinimo reikia, norint suprasti tekstą. Jis apskaičiuojamas pagal žemiau pateiktą formulę (2.2) [26]:

$$1.0430\sqrt{sudėtingi\ \check{z}odžiai\ *\ \frac{30}{sakinių\ skaičius}} + 3.1291; \quad (2.2)$$

Smogo indeksas paprastai labiau tinkamas vidurinio amžiaus (nuo 4 klasės iki kolegijos) skaitytojams [27]. Smogo indekso lygių paaiškinimai yra pateikti žemiau esančioje lentelėje (žr. 3 lentelė).

3 lentelė. Smogo indekso lygių paaiškinimas

Bendras daugiaskiemenių žodžių skaičius	**Apytikslis metų skaičius**	**Mokyklos lygis**
1-6	5	Darželis
7-12	6	Pirma klasė
13-20	7	Antra klasė
21-30	8	Trečia klasė
31-42	9	Ketvirta klasė
43-56	10	Penkta klasė
57-72	11	Šešta klasė
73-90	12	Septinta klasė
91-110	13	Aštunta klasė
111-132	14	Devinta klasė
133-156	15	Dešimta klasė
157-182	16	Vienuolikta klasė
183-210	17	Dvylikta klasė
211-240	18-22	Studentas (kolegija)

2.3.1.4. „java_fathom“ biblioteka

Tai „Java“ atvirojo kodo biblioteka. Skirta išmatuoti skaitomumo lygį anglų kalbos tekstuose. Biblioteka gali apskaičiuoti šiuos skaitomumo indeksus:

- Rūko indeksas;
- Flescho skaitymo lengvumo lygis;
- Flescho-Kincaido lygio balas.

2.3.1.5. Defekto „Sunkiai skaitomas tekstas (spalvų konfliktavimas)“ aptikimo metodai

Metodų sąrašas:

1. spalvų kontrastų skaičiavimas rankiniu būdu;
2. spalvų kontrastų skaičiavimas naudojant „Contrast Checker“ įrankį.

2.3.1.5.1. Spalvų kontrastų skaičiavimas rankiniu būdu

Spalvų kontrastas yra šviesos skirtumas tarp šrifto ir jo fono. Skirtumas tarp vienos spalvos ir kitos lemia ar dauguma žmonių galės perskaityti informaciją. Spalvų kontrasto santykis yra skaitinė vertė, nusakanti šviesumo skirtumą tarp dviejų spalvų. Padidinus kontrastą didėja ir skirtumas tarp šviesių ir tamsių spalvų, dėl to tekstas yra lengviau įskaitomas. Norėdami suskaičiuoti spalvų kontrastą pirmiausia reikia apsiskaičiuoti teksto ir fono spalvos skaisčius. **Klaida! Nerastas nuorodos š altinis.**Tada galime apskaičiuoti kontrasto santykį pagal (2.3) formulę [28]:

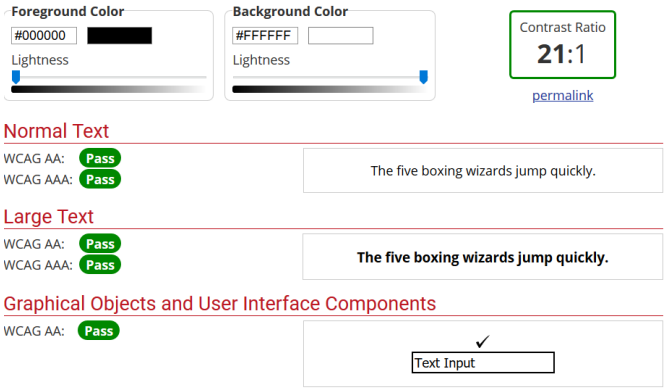
$$Kontrasto\ santykis = (L_1 + 0.05) / (L_2 + 0.05); \quad (2.3)$$

čia L_1 – šviesios spalvos skaistis;

L_2 – tamsios spalvos skaistis.

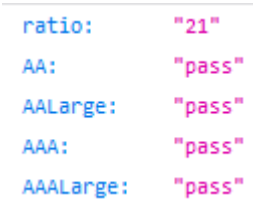
2.3.1.5.2. Spalvų kontrastų skaičiavimas naudojant „Contrast Checker“ įrankį

Spalvų kontrastą taip pat galima suskaičiuoti pasinaudojant „Contrast Checker“ internetiniu įrankiu [29]. Pasirinkę spalvas gausime atsakymus ar spalvos nekonfliktuoja (žr. 4 pav.).



4 pav. „Contrast checker“ įrankio vartotojo sąsaja

Taip pat šis įrankis turi API. Kreipiantis į API reikia nurodyti šrifto ir fono spalvą. Rezultatai būna pateikiami JSON formatu (žr. 5 pav.).



5 pav. Json rezultatas

2.3.1.6. Defekto „Įžeidžiantys pranešimai“ aptikimo metodai

Metodų sąrašas:
1. logistinė regresija;
2. žodyno taikymas;
3. įžeidžiančios kalbos API „plnia“.

2.3.1.6.1. Logistinė regresija

Logistinė regresija yra klasifikavimo algoritmas, naudojamas priskirti stebėjimus atskiram klasių rinkiniui. Keletas klasifikavimo problemų pavyzdžių yra ar elektroninis laiškas yra šlamštas (angl. *spam*) ar ne šlamštas, internetinės operacijos „sukčiavimas ar ne sukčiavimas“, piktybinis navikas arba gerybinis. Logistinė regresija transformuoja savo išvestį naudodama logistinę sigmoidinę funkciją, kad grąžintų tikimybės reikšmę [30].

2.3.1.6.2. Žodyno taikymas

Norėdami aptikti įžeidžiančius pranešimus, kaip ir blogos rašybos aptikime galima taikyti žodyną. Paprasčiausiai reikia turėti įžeidžiančių žodžių sąrašą ir patikrinti ar duotame sakinyje yra įžeidžiančių žodžių.

2.3.1.6.3. Įžeidžiančios kalbos API „plnia“

Tai mokamas API, kuris veikia kaip jautrumo filtras, apsaugantis nuo potencialiai žalingo, įžeidžiančio turinio. Sprendimas paremtas natūralios kalbos apdorojimo ir mašininio mokymosi technologijomis, kurios padeda gana tiksliai nustatyti potencialiai įžeidžiantį turinį. Taip pat „plnia“ suteikia puikią dokumentaciją palengvinančia integracijos procesą [31].

2.3.1.7. Defekto „Sinonimai“ aptikimo metodai

Metodų sąrašas:
1. automatinis nustatymas naudojant žodyną ”WordNet“;
2. neuroninis tinklas „Word2vec“.

2.3.1.7.1. Automatinis nustatymas naudojant žodyną ”WordNet“

„WordNet“ yra didelė leksinė kalbos duomenų bazė [32]. Daiktavardžiai, veiksmažodžiai, būdvardžiai ir prieveiksmiai yra suskirstyti į pažintinių sinonimų rinkinius, kiekvienas išreiškiantis savitą sąvoką. „WordNet“ paviršutiniškai primena žodyną, nes jis sugrupuoja žodžius pagal jų reikšmes. Pagrindinis „WordNet“ žodžių santykis yra sinonimija. Sinonimai – žodžiai, žymintys tą pačią sąvoką ir keičiami daugelyje aplinkybių, yra sugrupuoti į netvarkingus rinkinius. Be to, rinkinyje yra trumpas apibrėžimas ir daugeliu atvejų vienas ar keli trumpi sakiniai, iliustruojantys naudojimą. Žodžių formos su keliomis skirtingomis reikšmėmis pavaizduotos daugelyje skirtingų sintaksių. Taigi kiekviena formos ir reikšmės pora „WordNet“ yra unikali. Tačiau yra keletas svarbių skirtumų tarp žodyno. Pirmiausia, „WordNet“ susieja ne tik žodžių formas ar raidžių eilutes, bet ir specifines žodžių reikšmes. Dėl to žodžiai, kurie randami vienas šalia kito, yra semantiškai išskaidomi. Antra, „WordNet“ pažymi semantinius žodžių ryšius, tuo tarpu žodžių grupavimas žodyne neseka jokio aiškaus modelio, išskyrus prasmės panašumą.

2.3.1.7.2. Neuroninis tinklas „Word2vec“

„Word2vec“ yra dviejų sluoksnių neuroninis tinklas, kuris apdoroja tekstą „vektorizuodamas“ žodžius. Jo įvestis yra teksto rinkinys, o išvestis – vektorių rinkinys: elementų vektoriai, kurie žymi tame rinkinyje esančius žodžius. Nors „Word2vec“ nėra gilusis neuroninis tinklas, jis paverčia tekstą skaitine forma, kurią gilieji neuroniniai tinklai gali suprasti. „Word2Vec“ idėja yra gana paprasta. Ji yra analogiška posakiui „parodyk man savo draugus, ir aš pasakysiu, kas tu esi“. Pvz.: jei turime du žodžius, kurie turi labai panašius kaimynus (reiškia: kontekstas, kuriame jis yra vartojamas beveik tas pats), tada šie žodžiai tikriausiai yra gana panašios prasmės arba bent jau yra susiję. Pavyzdžiui, žodžiai „šokiruoti“, „apstulbę“ ir „nustebinti“ dažniausiai vartojami panašiame kontekste [33].

2.3.1.8. Defekto „Neužpildytos vietos žymenos“ aptikimo metodai

Metodų sąrašas:
1. žodyno taikymas.

2.3.1.8.1. Žodyno taikymas

Norėdami aptikti neužpildytas vietos žymenas taipogi galime naudoti žodyną. Tokį žodyną yra sunku rasti internete, tačiau jį galima susidaryti patiems. Tada tokiu pačiu principu kaip aptinkant blogą rašybą ar įžeidžiančius pranešimus, reikia patikrinti ar duotame sakinyje yra žodžių, kurie identifikuotų, jog tai yra neužpildyta vietos žymena.

2.3.1.9. Defekto „Per ilgi tekstai“ aptikimo metodai

Metodų sąrašas:
1. išanalizuoti teksto charakteristikas.

2.3.1.9.1. Išanalizuoti teksto charakteristikas

Norėdami įvertinti teksto sudėtingumą reikia apskaičiuoti teksto charakteristikas. Atlikus teksto charakteristikų analizę tuo pačiu sužinosime ir žodžių skaičių. Turėdami šį parametrą nesunkiai galime įvertinti ar tekstai yra per ilgi. Daugelyje straipsnių minima, kad maksimalus žodžių skaičius sakinyje neturėtų peržengti 25 ar 30 žodžių ribos.

2.4. Atrinkti sprendimai

Žemiau yra pateikiami kiekvieno defekto aptikimo sprendimai (žr. 4 lentelė).

4 lentelė. Atrinkti sprendimai

Defektas	**Sprendimas**
Bloga rašyba	Žodyno taikymas
Mišri kalba	Biblioteka „Lingua“
Sunkiai skaitomas tekstas (sudėtingas)	Rūko indeksas
Sunkiai skaitomas tekstas (spalvų konfliktavimas)	Spalvų kontrastų santykis
Įžeidžiantys pranešimai	Žodyno taikymas
Sinonimai	Leksikos duomenų bazės „WordNet“ taikymas
Neužpildytos vietos žymenos	Žodyno taikymas
Per ilgi tekstai	Teksto charakteristikų analizavimas

2.4.1. Atrinktų sprendimų pagrindimas

Sprendimai buvo atrinkti remiantis ne tik jų tikslumu, bet greitaveika ir suderinamumu su „Java“ programavimo kalba:

- blogai rašybai, įžeidžiantiems pranešimams bei vietos žymenų aptikimui pasirinkome taikyti žodyną. Pasirenkant gerą duomenų struktūrą žodžio paieška žodyne įvykdoma labai greitai. Be to daugelį žodynų galima atsisiųsti iš interneto;
- mišriai kalbai buvo pasirinkta biblioteka „Lingua“, kadangi ji geriausiai veikia su trumpais pranešimais bei palaiko daugiausiai kalbų;
- sunkiai skaitomam tekstui (sudėtingam) buvo pasirinkta skaičiuoti rūko indeksą, kadangi galima nesunkiai įvardinti skaitymo lygį;
- spalvų kontrastą pasirinkta skaičiuoti rankiniu būdu nenaudojant API, kadangi norima supaprastinti programos veikimą;

- sinonimams aptikti pasirinkta naudoti „WordNet leksinę duomenų bazę, kadangi ji yra semantiškai struktūrizuota ir labai išplėsta;
- per ilgiems tekstams nebuvo alternatyvių metodų. Paprasčiausiai reikia išgauti teksto charakteristikas ir jas kiekybiškai įvertinti.

2.5. Egzistuojančių rinkoje įrankių defektų aptikimų palyginimas

Žemiau pateiktoje lentelėje yra palyginami rinkoje egzistuojantys įrankiai, pagal tai kokius defektų tipus gali aptikti (žr. 5 lentelė).

5 lentelė. Įrankių palyginimas pagal išsprendžiamus defektus

Defektas/Įrankis	**plnia API**	**java_fathom**	**Lucene**	**language-detector**	**WordNet**	**LanguageTool**	**Lingua**	**Contrast Checker**
Bloga rašyba			+			+		
Mišri kalba				+			+	
Sunkiai skaitomas tekstas (Sudėtingas)		+						
Sunkiai skaitomas tekstas (spalvų konfliktavimas)								+
Įžeidžiantys pranešimai	+							
Sinonimai					+			
Neužpildytos vietos žymenos								
Per ilgi tekstai								

Atlikus palyginimą pagal įrankio aptinkamų defektų tipus aiškiai matosi, kad nei vienas įrankis neaptinka daugiau nei vieno defekto tipo. Taip yra todėl, kad visi įrankiai specializuoti vienos problemos sprendimui. Tuo tarpu kuriamas automatinis įrankis gebės aptikti visus minėtus defektų tipus.

2.6. Įgyvendinimo problemos

Tekstinių defektų aptikimo efektyvumas priklauso ne tik nuo pačio metodo tikslumo, bet ir nuo įrankio gebėjimo ištraukti visus reikiamus tekstus. Daugeliui defektų aptikti galima naudoti mašininio

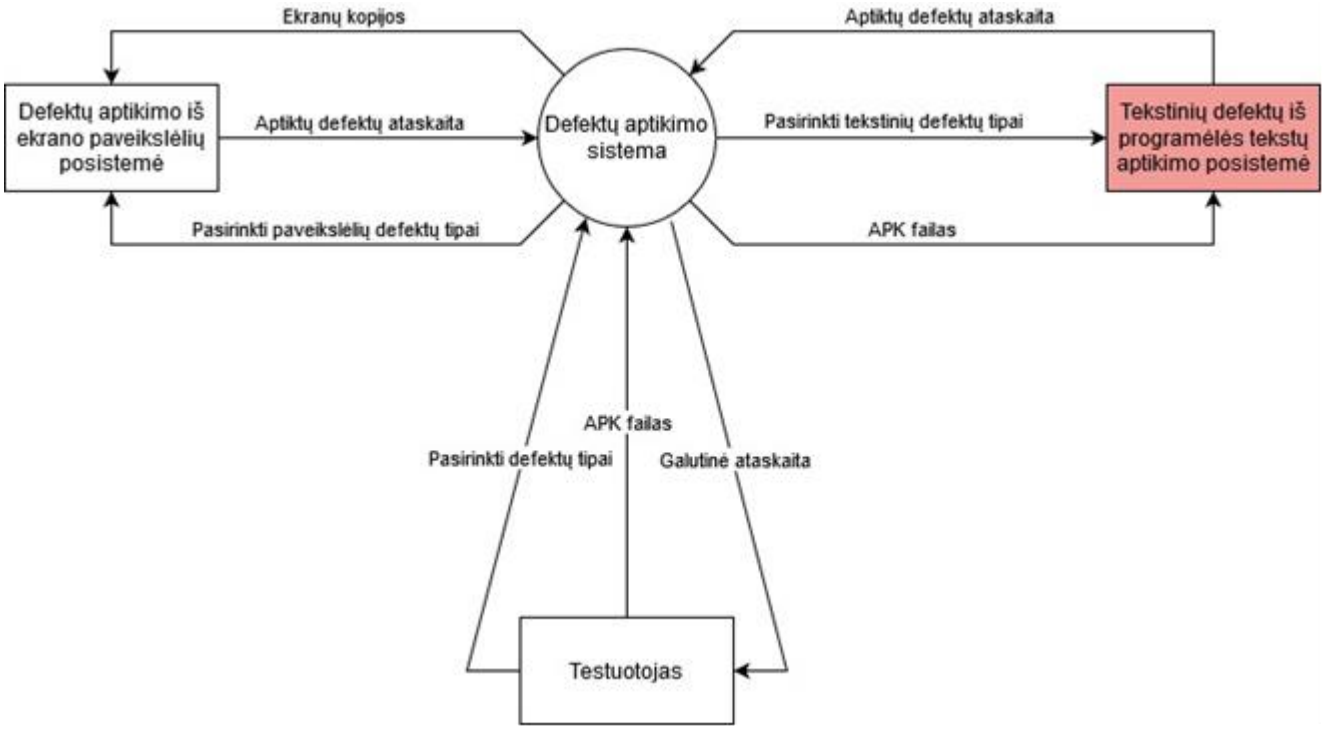
mokymo modelius, tačiau jų apmokymas gali užtrukti daug laiko. Kai kurių defektų aptikimas yra labai sudėtingas, todėl yra tikimybė, kad tam tikrus defektus aptikti yra neįmanoma arba bus pasiektas tik minimalus tikslumas. Yra nemažai gerų įrankių, kurie aptinka tam tikrus defektus, tačiau jie yra mokami. Gali iškilti problemų pritaikant visus metodus vienoje aplinkoje. Daugelis metodų veikia tik dirbant su anglų kalbos tekstais.

3. Projektinė dalis

3.1. Veiklos sudėtis

3.1.1. Veiklos kontekstas

Pagrindinė idėja yra sukurti bendrą žiniatinklio taikomąją programą, kurioje vartotojas galėtų atlikti automatinį programėlės testavimą įkeliant APK tipo failą. Sistemą sudaro 2 posistemės. Viena posistemė atsakinga už tekstinių defektų aptikimą analizuojant programėlės ekrano kopijas, o kita posistemė atsakinga už tekstinių defektų aptikimą analizuojant programėlės resursus. Kuriamas įrankis apima tekstinių defektų aptikimą iš programėlės resursų posistemę, kuri diagramoje pažymėta raudona spalva (žr. 6 pav.).



6 pav. Veiklos konteksto diagrama

3.1.2. Veiklos padalinimas

Žemiau esančioje lentelėje yra detalizuojami duomenų srautai (žr. 6 lentelė).

6 lentelė. Veiklos padalinimas

Eil. Nr.	Įvykio pavadinimas	Įeinantys/Išeinantys informacijos srautai
1	APK failas	Perduodamas APK failas reikalingas defektų analizei
2	Aptiktų defektų ataskaita	Programėlėje rastų defektų sąrašas ir jų aprašymas
3	Pasirinkti tekstinių defektų tipai	Testuotojo pasirinkti defektai, kurie yra tekstinio tipo

4	Pasirinkti paveikslėlių defektų tipai	Testuotojo pasirinkti defektai, kurie yra grafinio tipo
5	Pasirinkti defektų tipai	Visi testuotojo pasirinkti defektų tipai
6	Galutinė ataskaita	Sistemos gražinta aptiktų defektų ataskaita
7	Ekranų kopijos	Ekranų kopijos reikalingos grafinio tipų defektams aptikti

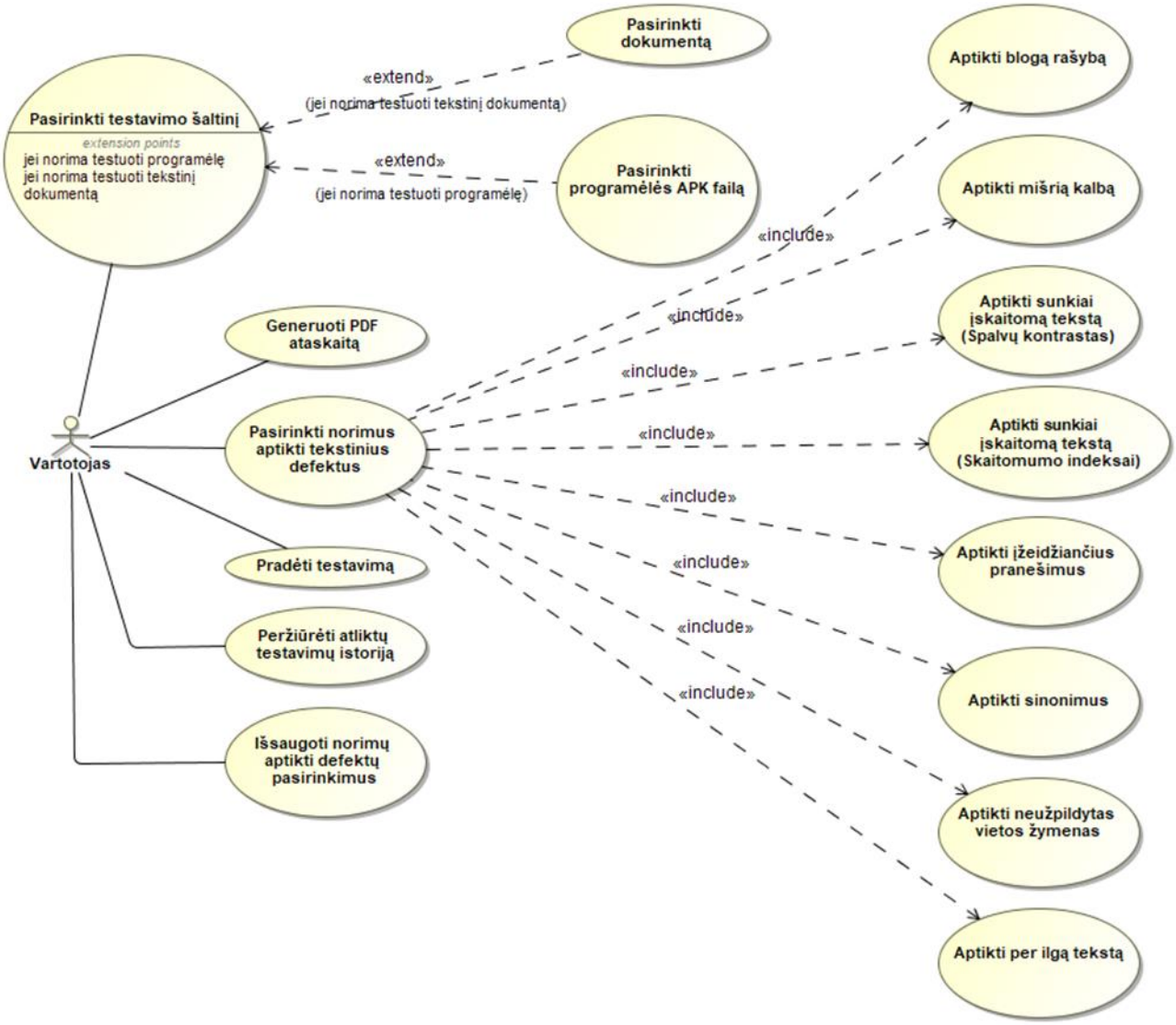
3.1.3. Sistemos funkcinis aprašymas

Sukurtas įrankis yra skirtas neteisingos kalbos ir lokalizacijos defektų paieškai mobiliosiose programose analizuojant programos resursus. Aptikdamas defektus automatiškai, įrankis sumažins testuotojų rankinį darbą, bei žmogiškųjų klaidų tikimybę.

Dabartinė sistemos versija atlieka šias funkcijas:

- leidžia analizuoti „txt“, „pdf“, „doc“, „docx“ ir „apk“ plėtinius turinčius failus;
- pasirinkti aptikti šiuos defektų tipus: bloga rašyba, sudėtingas tekstas (sudėtingumo indeksai), sunkiai įskaitomas tekstas (blogas kontrastas tarp spalvų), neužpildytos vietos žymenos, per ilgas tekstas, įžeidžiantys pranešimai, sinonimai, mišri kalba;
- išsaugoti norimų aptikti defektų pasirinkimą;
- peržiūrėti analizuotų failų istoriją, bei ją ištrinti;
- peržiūrėti aptiktų defektų ataskaitą;
- generuoti PDF formato ataskaitą.

Toliau pateikiama panaudojimo atvejų diagrama (žr. 7 pav.).



7 pav. Panaudojimo atvejų diagrama

3.1.4. Funkciniai reikalavimai

Žemiau yra pateikiami įrankio funkciniai reikalavimai:

- vartotojui turi būti galimybė panaudoti įrankį testuojant ir APK failą ir paprastą tekstinį dokumentą;
- pasibaigus testavimui turi būti iš karto parodomi testavimo rezultatai. Su aptiktais defektais bei jų vietomis;
- vartotojui turi būti suteikiamas funkcionalumas leidžiantis išsaugoti norimus testavimo rezultatus istorijoje automatiškai sugeneruojant PDF ataskaitą;
- turėtų būti galimybė išsaugoti pasirinktus testuoti defektus.

3.1.5. Sistemos apribojimai

Žemiau yra pateikiami įrankio nefunkciniai reikalavimai. Reikalavimai sistemos išvaizdai:

- vartotojo sąsaja turi būti neįkyri;
- aptikus klaidas: bloga rašyba, neužpildytos vietos žymenos, per ilgi tekstai, įžeidžianti kalba, jas paryškinti tekste nuspalvinant spalvomis.

Reikalavimai panaudojamumui:

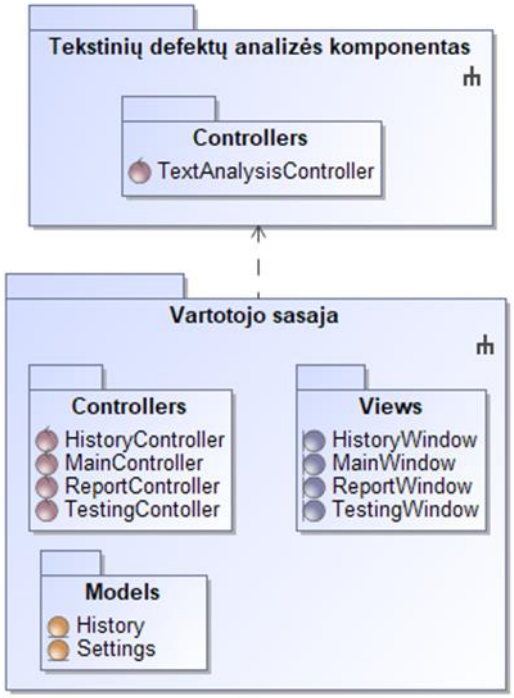
– sistema turi būti Anglų kalba;
– sisteminiai pranešimai turėtų būti aiškūs ir netik informatikams.

Reikalavimai vykdymo charakteristikoms:

– programėlė negali užtrukti ilgiau nei 10 min jei programėlės apimtis labai didelė.
– sistema turėtų apdoroti txt, pdf, doc ir docx dokumentų tipus;
– įrankis turėtų neleisti pasirinkti šaltinio, kurio nepalaiko;
– sistema turėtų būti realizuota taip, kad būtų galima lengvai pridėti daugiau defektų aptikimo metodų.

3.1.6. Sistemos statinis vaizdas

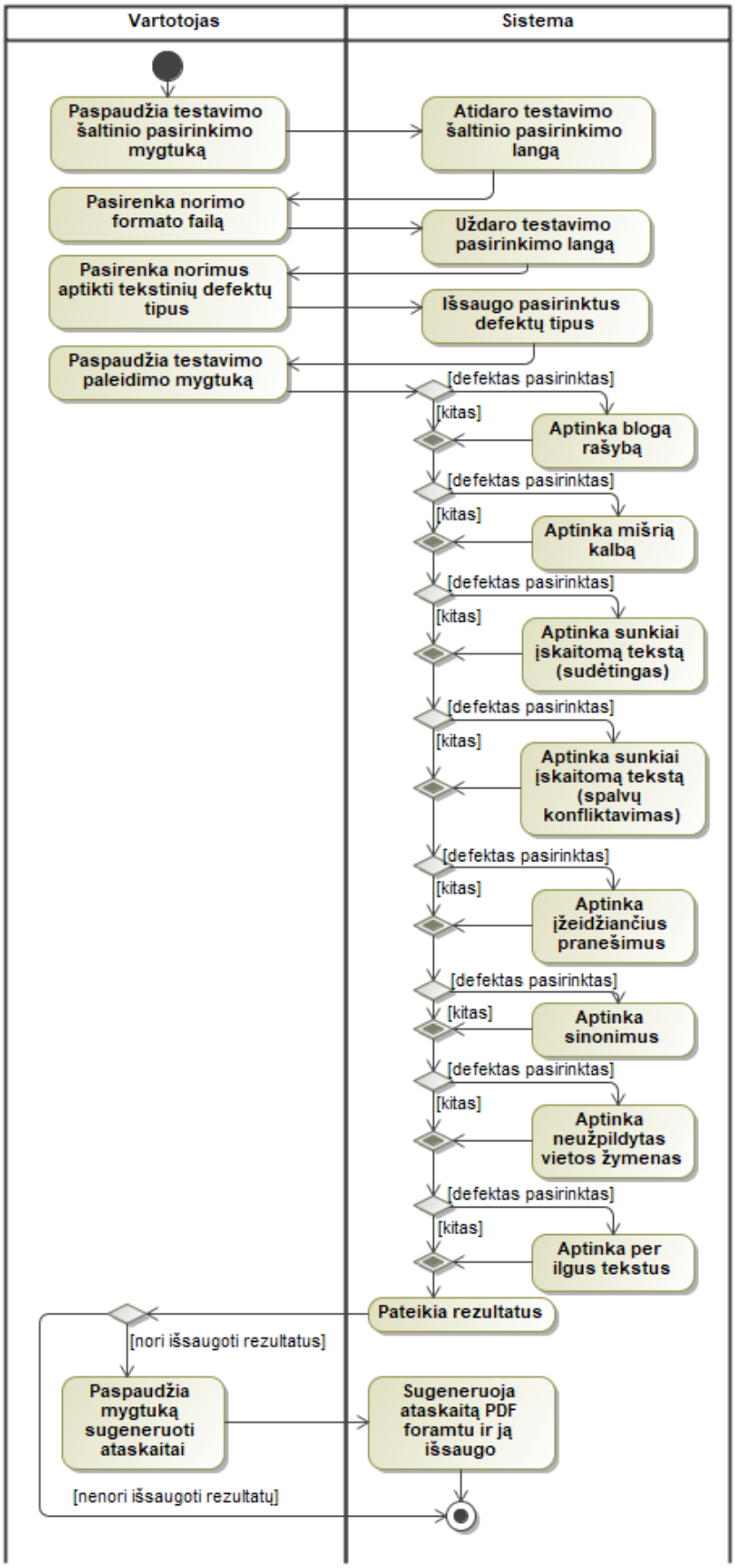
Žemiau pateiktoje diagramoje yra pavaizduota paketų diagrama (žr. 8 pav.). Sistemą sudaro dvi pagrindinės dalys: vartotojo sąsaja ir tekstinių defektų analizės komponentas. Vartotojo sąsajos paketas atsakingas už programos langų atvaizdavimą ir jų tarpusavio bendravimą. Šis paketas taip pat skirtas testavimų rezultatų bei nustatymų išsaugojimui. Tekstinių defektų analizės komponentas yra svarbiausia sistemos dalis, kuri nuskaito tekstą iš programėlės resursų ar tekstinio failo. Šio paketo pagalba, teksto analizės metu, bandoma aptikti tekstinius defektus bei gražinti rezultatus vartotojo sąsajos komponentui.



8 pav. Klasių paketų diagrama

3.1.7. Sistemos dinaminis vaizdas

Dinaminis sistemos vaizdas pavaizduotas veiklos diagrama (žr. 9 pav.). Diagramoje atsispindi pagrindinė veiksmų seka norint atlikti pasirinktos programėlės testavimą.



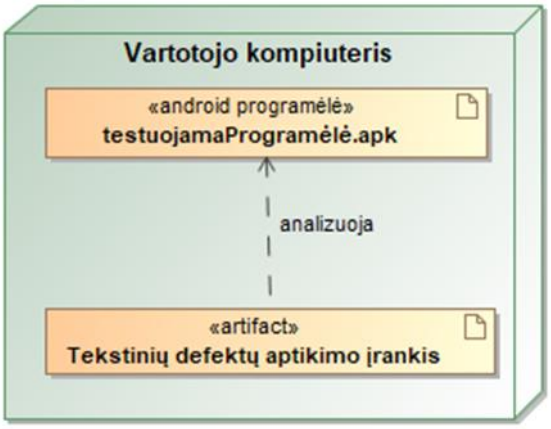
9 pav. Bendra sistemos veikimo veiklos diagrama

3.1.8. Išdėstymo vaizdas

Sistema bus diegiama asmeniniame vartotojo kompiuteryje. Diegimo diagrama yra pateikiama žemiau (žr. 10 pav.).

Sistemos rekomenduojami reikalavimai:

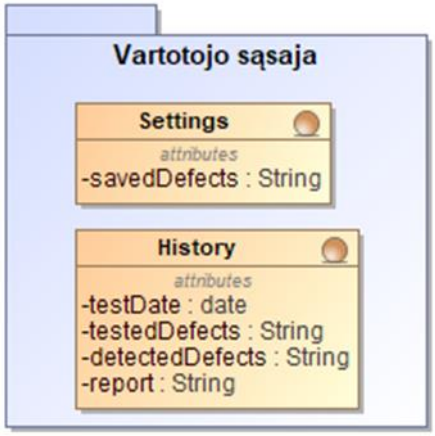
- *Intel Core i5* 4 branduolių 2.60GHz procesorius;
- 8GB darbinės atminties;
- *Windows 10* operacinė sistema;
- JDK 11 arba naujesnė.



10 pav. Sistemos diegimo diagrama

3.1.9. Duomenų vaizdas

Sistema duomenų bazės nenaudos. Visi reikiami duomenys bus išsaugoti failuose. Žemiau pavaizduota duomenų modelio schema (žr. 11 pav.).



11 pav. Duomenų modelio schema

4. Tyrimo dalis

Šiame skyriuje pateikiamas kiekvieno algoritmo, kuris naudojamas eksperimentinei daliai atlikti, aprašymas.

4.1. Teksto skaitymas iš APK failo

Norėdami aptikti tekstinį defektą, pirmiausia reikia iš programėlės APK failo ištraukti tekstą. Norėdami tai padaryti reikia APK failą konvertuoti į ZIP formato failą, jog duomenys būtų suprantami automatinio testavimo įrankiui. Tai atlikti galime pasinaudoję „Java“ programavimo kalbos „java.util.zip“ paketu. Turėdami programėlės ZIP failą galime nuskaityti resursų katalogą, kuriame yra pateikti programėlės išdėstymo (angl. *layouts*) langai. Išdėstymo langai yra XML formato, kuris yra patogus atliekant analizę. Android programėlės išdėstymo lango XML failo pavyzdys yra pateikiamas žemiau esančiame paveikslėlyje (žr. 12 pav.).

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

12 pav. Išdėstymo lango pavyzdys

Kaip matome išdėstymo langą sudaro 2 elementai: teksto rodinys ir mygtukas. Abu elementai turi parametrą „android:text“, kuris yra skirtas tekstui nustatyti. Kadangi žinome kaip yra nurodomi programėlės tekstai, belieka parašyti algoritmą, kuris nuskaitytų šį tekstą. Teksto nuskaitymo algoritmo pseudokodas aprašomas žemiau pateiktame paveikslėlyje (žr. 13 pav. Teksto nuskaitymo pseudokodas).

```
Function readAPKFile(app)
    resourceFiles ← extractResourceFiles(app)
    texts ← ∅

    For each xmlFile in resourceFiles
        texts ← extractText(xmlFile)
    Endfor

    Return texts
Endfunction
```

13 pav. Teksto nuskaitymo pseudokodas

Kaip matome funkcijos įvestyje yra nurodomas programėlės failas, o išvestyje yra gražinami nuskaityti tekstai.

4.2. Blogos rašybos aptikimas

Blogai rašybai aptikti naudojamas anglų kalbos žodynas. Algoritmo esmė yra ištraukti iš APK resursų failų tekstą ir patikrinti ar tame tekste esantys žodžiai patenka į anglų kalbos žodyną. Anglų kalbos žodynai yra prieinami internete įvairiais formatais. Žodynas buvo parsisiųstas TXT formatu, kurį yra paprasta nuskaityti. Beliko nuskaitytą tekstą išskaidyti į atskirus žodžius ir patikrinti ar šis tekstas patenka į anglų kalbos žodyną. Blogos rašybos algoritmo pseudokodas pateikiamas žemiau (žr. 14 pav.).

```
1  Function checkBadSpelling(texts)
2      words ← extractWords(texts)
3      incorrectWords ← ∅
4
5      For each word in words
6          If isWordCorrect(word) == False
7              incorrectWords ← word
8          EndIf
9      Endfor
10
11     Return incorrectWords
12 Endfunction
```

14 pav. Blogos rašybos tikrinimo pseudokodas

Funkcijos įvestyje nurodome iš programėlės nuskaitytus tekstus, o išvestyje gauname žodžių sąrašą, kurie buvo parašyti su klaida.

4.3. Mišrios kalbos aptikimas

Mišrios kalbos aptikimui buvo naudojama „Java“ biblioteka „Lingua“. Ši biblioteka grindžiama n-gramų tikimybiniu modeliu. Pagrindinis n-gramų tikslas yra tas, kad jie užfiksuoja kalbos struktūrą statistiniu požiūriu, pavyzdžiui, kokia raidė ar žodis turi didžiausią tikimybę eiti po kito. N-gramai yra tiesiog visi gretimų žodžių ar raidžių n ilgio deriniai, kuriuos galite rasti tekste [34]. Pavyzdžiui, žodžio „šuo“, bigramai yra „šu“ ir „uo“. Daugumoje bibliotekų naudojami tik trigramai, tai yra patogu aptinkant ilgesnių, iš kelių sakinių sudarytų teksto fragmentų, kalbą. Trumpoms frazėms ar pavieniams žodžiams vis dėlto nepakanka trigramų. Kuo trumpesnis įvesties tekstas, tuo mažiau n-gramų yra. Tikimybė, apskaičiuota pagal kelis n-gramus, nėra patikima. Štai kodėl „Lingua“ naudoja n-gramus nuo 1 iki 5 dydžio, o tai lemia daug tikslesnį teisingos kalbos prognozavimą. Žemiau pateikiamas algoritmas skirtas mišrios kalbos aptikimui (žr. 15 pav.).

```
Function detectLanguages(texts)
    sentences ← extractSentences(texts)
    detectedLanguages ← ∅

    For each sentence in sentences
        detectedLanguages ← sentence, predictLanguage(sentence)
    Endfor

    Return detectedLanguages
Endfunction
```

15 pav. Mišrios kalbos aptikimo pseudokodas

Algoritmo esmė funkcijai paduoti tekstą ir jį išskaidyti sakiniais. Po to kiekvienam sakiniui iškviesti funkciją „predictLanguage()“, kuri gražintų kalbą turinčią didžiausią tikimybę, jog paduotas sakinys yra būtent ta kalba. Išvestyje gauname sąrašą sakinių ir kalbą, kuria šis sakinys parašytas. Turėdami šiuos duomenis galime juos įvairiai apdoroti. Vienas iš apdorojimo būdų yra paduoti funkcijai kalbą, kuria mes tikimės, jog tekstas yra parašytas, tada ši funkcija išfiltruotų tuos sakinius, kurie yra ta kalba. Galiausiai liktų tik tie sakiniai, kurie yra kita kalba, tokiu būdų sužinotume, kurie tekstai yra neišversti reikiama kalba.

4.4. Sunkiai skaitomo teksto (sudėtingo) aptikimas

Teksto sudėtingumo lygiui nustatyti buvo pasirinkta skaičiuoti rūko indeksą. Rūko indekso apskaičiavimo pseudokodas pateikiamas žemiau (žr. 16 pav.).

```
Function getGunningFog(allText)
    words ← extractWords(allText)
    complexWords ← extractComplexWords(allText)
    sentences ← extractSentences(allText)
    fogIndex ← ∅

    fogIndex ← 0.4 * (words /sentences + 100 * complexWords / words)

    Return fogIndex
Endfunction
```

16 pav. Rūko indekso aptikimo pseudokodas

Norėdami apskaičiuoti rūko indeksą pirmiausia reikia suskaičiuoti paduoto teksto charakteristikas tokias kaip: žodžių kiekis, sudėtingų žodžių kiekis (žodžiai, kurie turi bent 3 skiemenis) ir sakinių skaičius. Turėdami šias teksto charakteristikas galime nesunkiai apskaičiuoti rūko indeksą.

4.5. Sunkiai skaitomo teksto (konfliktuojančių spalvų) aptikimas

Norint nustatyti ar teksto ir fono spalvos tarpusavyje dera ir tekstas yra aiškiai matomas, reikia suskaičiuoti šių spalvų kontrastų santykį [35]. Tarkime šios spalvos aprašytos pagal RGB spalvų modelį, kuriame kiekviena raidė atitinka sveikąjį skaičių tarp 0 ir 255. Tada šias reikšmes reikia

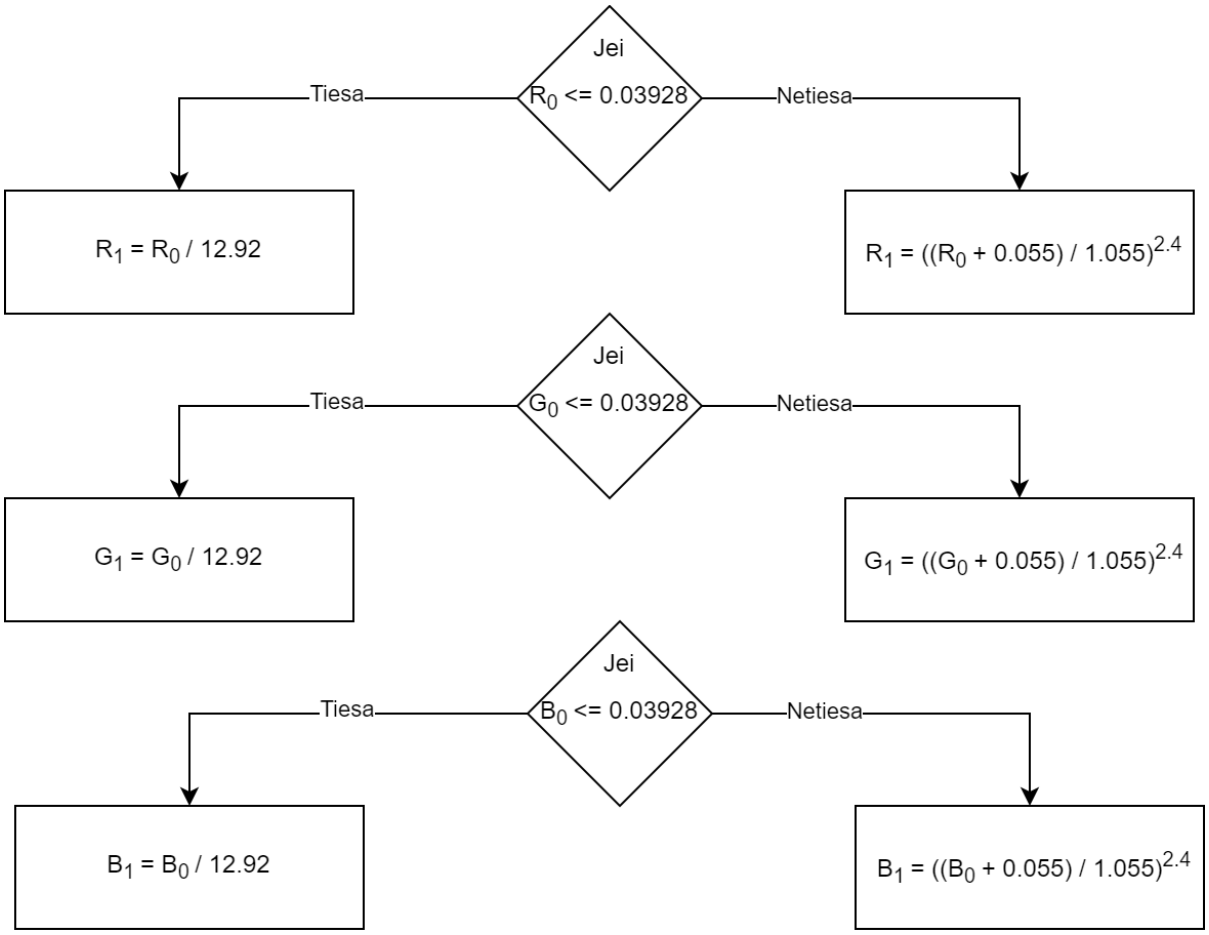
perskaičiuoti, jog jos būtų rėžiuose nuo 0 iki 1, tai padaryti galima naudojant žemiau pateiktas formules (4.1) (4.2) (4.3).

$$R_0 = R / 255; \quad (4.1)$$

$$G_0 = G / 255; \quad (4.2)$$

$$B_0 = B / 255; \quad (4.3)$$

Gavę RGB reikšmes naujuose rėžiuose galime suskaičiuoti jų tiesines reikšmes. Tiesinės reikšmės apskaičiuojamos priklausomai nuo to, kokias reikšmes gavome perskaičiavę rėžius tarp 0 ir 1. Kaip apskaičiuoti tiesines reikšmes galime pamatyti apačioje pateiktame paveikslėlyje (žr. 17 pav.).



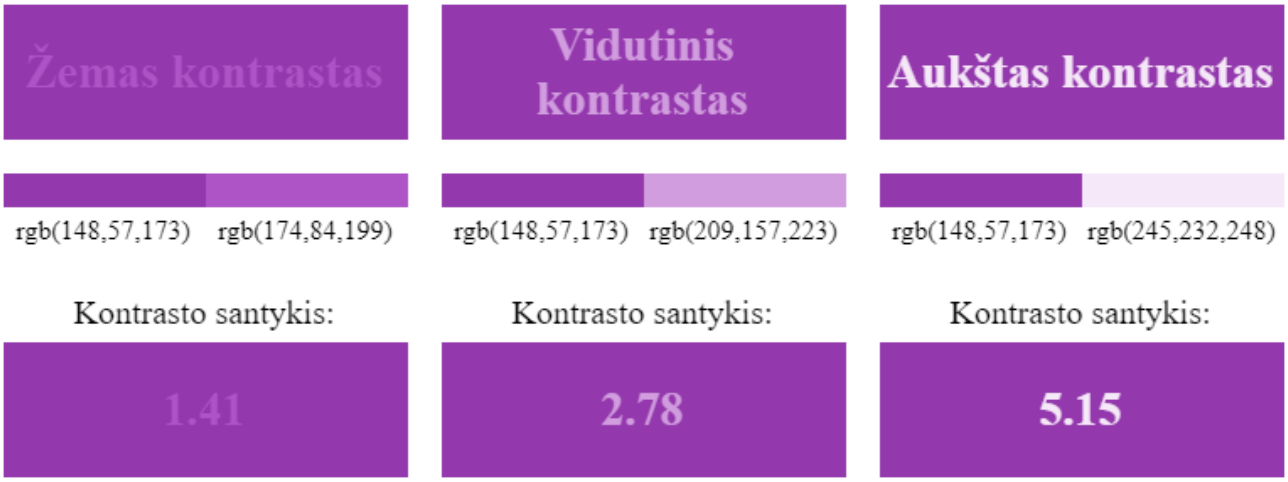
17 pav. RGB tiesinių reikšmių apskaičiavimas

Galiausiai galime apskaičiuoti spalvos skaistį pagal (4.4) formulę:.

$$L = (0.2126 * R_1) + (0.7152 * G_1) + (0.0722 * B_1); \quad (4.4)$$

Žinodami kaip apskaičiuoti spalvų skaistį, galime apskaičiuoti teksto ir fono spalvos kontrasto santykį. Norėdami tai padaryti pirmiausia turime apskaičiuoti abiejų spalvų skaistį ir nustatyti, kuri spalva yra šviesiausia, o kuri spalva yra tamsiausia. Žinodami šias reikšmes jau galime apskaičiuoti kontrasto santykį tarp norimų spalvų pagal (2.3) formulę.

Jeigu norime, kad vartotojui tekstas būtų lengvai įskaitomas, spalvų kontrastas neturėtų būti mažesnis nei 4. Žemiau pateikiami kontrastų santykių, tarp fono ir teksto spalvų, pavyzdžiai (žr. 18 pav.).



18 pav. Spalvų kontrastų santykių pavyzdžiai

Kaip jau minėjome 4.1 skyriuje, iš programėlės resursų galime ištraukti tekstus, tačiau resursų failuose būna nurodyti ne tik tekstai, bet ir teksto bei fono spalvos. Teksto spalva nusakoma parametru „android:textColor“, o fono spalva parametru „android:background“ (žr. 19 pav.). Šiuo atveju spalvos nurodytos HEX formatu, tačiau pasinaudoję Java programavimo kalbos bibliotekomis, galime šias spalvas konvertuoti į RGB spalvų modelį.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical" >

    <TextView
            android:id="@+id/text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="#CAB8B8"
            android:text="Hello, I am a TextView"
            android:textColor="#510E0E" />
</LinearLayout>
```

19 pav. Išdėstymo lango pavyzdys su spalvų parametrais

Turėdami teksto ir fono spalvas, galime apskaičiuoti jų kontrastų santykius ir aptikti kurios spalvos tarpusavyje konfliktuoja. Šio algoritmo pseudokodas yra pateiktas žemiau esančiame paveikslėlyje (žr. 20 pav.).

```
Function detectBadContrast(colorPairs)
    badContrasts ← ∅

    For each colorPair in colorPairs
        if getContrast(colorPair.textColor, colorPair.backGroundColor) < 4
            badContrasts ← colorPair
        Endif
    Endfor

    Return badContrasts
Endfunction
```

20 pav. Blogo kontrasto aptikimo pseudokodas

Algoritmo esmė yra patikrinti visas teksto ir fono spalvas tarpusavyje ir pažiūrėti ar jų kontrastų santykis nėra mažesnis už 4.

4.6. Įžeidžiančių pranešimų aptikimas

Įžeidžiantiems pranešimams aptikti buvo taikomas žodynas, bei logistinė regresija.

4.6.1. Žodyno taikymas

Žodyno taikymas įžeidžiantiems pranešimas aptikti yra toks pat kaip ir blogos rašybos aptikimui. Kaip ir anglų kalbos žodynai, įžeidžiančių žodžių sąrašai taip pat yra prieinami internete TXT formatu. Turėdami įžeidžiančių žodžių sąrašą, galime iš nuskaityto teksto patikrinti ar jame esantys žodžiai patenka į tą sąrašą. Įžeidžiančių pranešimų aptikimo pseudokodas pateikiamas žemiau esančiame paveikslėlyje (žr. 21 pav.).

```
Function checkOffensiveLanguage(texts)
    words ← extractWords(texts)
    offensiveWords ← ∅

    For each word in words
        If isWordOffensive(word) == true
            offensiveWords ← word
        EndIf
    Endfor

    Return offensiveWords
Endfunction
```

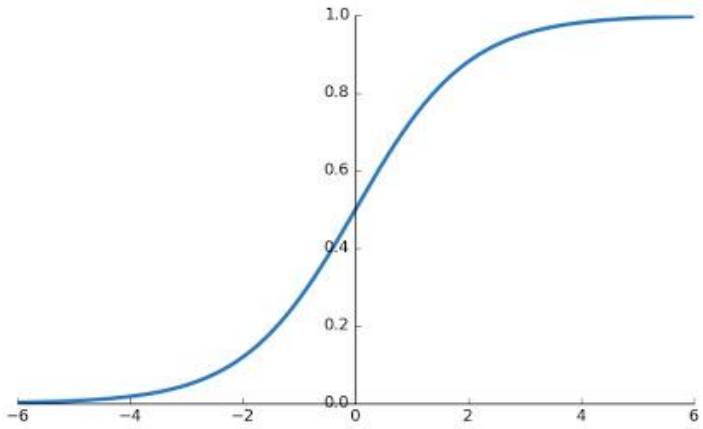
21 pav. Įžeidžiančių pranešimų aptikimo, taikant žodyną, pseudokodas

4.6.2. Logistinės regresijos taikymas

Logistinė regresija yra vienas iš paprasčiausių ir dažniausiai naudojamų mašininio mokymo algoritmų atliekant klasifikavimą 2 klasėms. Šiuo atveju viena klasė būtų, jog tekstas yra

neįžeidžiantis, o kita, kad įžeidžiantis. Logistinė regresija apibūdina ir įvertina ryšį tarp vieno priklausomo dvejetainio kintamojo ir kitų nepriklausomų kintamųjų.

Logistinės regresijos pagrindas yra logistinė funkcija, dar vadinama sigmoidine funkcija, kuri ima bet kokį realųjį skaičių ir susieja jį su reikšme tarp 0 iki 1 [36]. Žemiau pateikiamas šios funkcijos grafikas ir formulė (žr. 22 pav.) (4.5).



22 pav. Sigmoidinės funkcijos grafikas

$$y = 1/(1 + e^{-x}); \quad (4.5)$$

Reikšmės esančios didesnės už 0.5 bus laikomos kai įžeidžiantys pranešimai. Norint pritaikyti logistinę regresiją ir apmokyti mašininio mokymo modelį, galima naudoti „Python“ mašininio mokymo biblioteka „scikit-learn“.
Turėdami apmokytą modelį, galime nesunkiai realizuoti algoritmą, kuris aptiktų įžeidžiančius pranešimus. Algoritmo pseudokodas pateikiamas žemiau esančiame paveikslėlyje (žr. 23 pav.).

```
1  Function checkOffensiveLanguage(texts)
2      sentences ← extractSentences(texts)
3      offensiveSentences ← ∅
4
5      For each sentence in sentences
6          If isSentenceOffensive(sentence) == true
7              offensiveSentences ← sentence
8          EndIf
9      Endfor
10
11     Return offensiveSentences
12 Endfunction
```

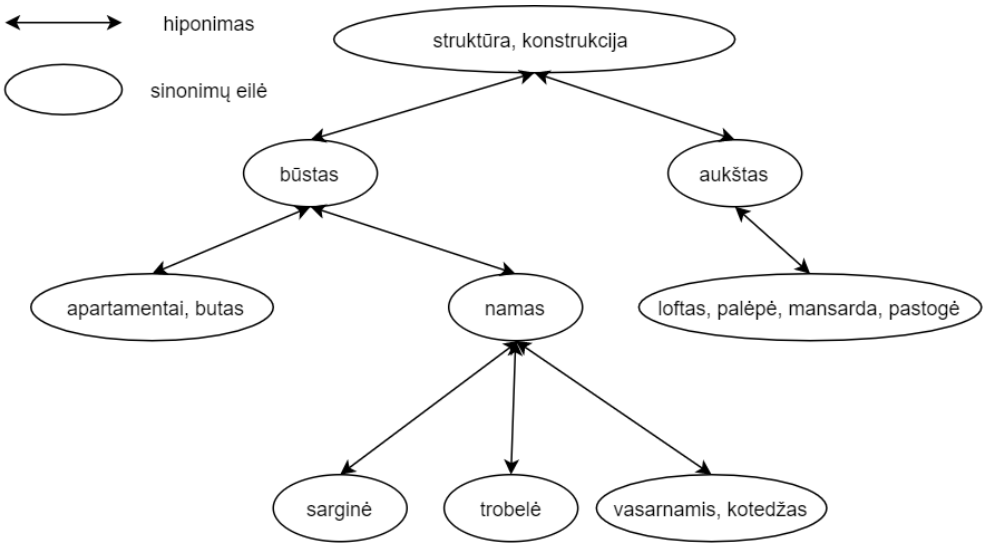
23 pav. Įžeidžiančių pranešimų aptikimo, taikant apmokytą modelį, pseudokodas

4.7. Sinonimų aptikimas

Sinonimams aptikti buvo naudojama leksinė duomenų bazė „WordNet“ ir neuroninis tinklas „Word2vec“.

4.7.1. Sinonimų aptikimas naudojant „WordNet“

„WordNet“ duomenų bazėje žodžio paieška grindžiama psicholingvistinėmis teorijomis. WordNet informacija yra saugoma pagal sintaksines kategorijas ir yra sujungta įvairiais santykių tipais. Yra ir sinonimų, ir poliseminių žodžių. Sinonimai yra skirstomi į grupes, vadinamas sinonimų eilėmis (angl. *synsets*). Kreipiantis į šias sinonimų eiles buvo atrinkti visi aktualūs sinonimai. Žemiau pateikiamas „WordNet“ semantinio tinklo fragmentas (žr. 24 pav.).



24 pav. „WordNet“ semantinio tinklo fragmentas

Toliau pateikiamas algoritmo, aptinkančio sinonimus, pseudokodas (žr. 25 pav.).

```
Function findSynonyms(texts)
    words ← extractWords(texts)
    detectedSynonyms ← Ø

    For each word in words
    synonymsOfCurrentWord ← getSynonyms(word)
        For each synonym in synonymsOfCurrentWord
            For each word in words
                If word == synonym
                   detectedSynonyms ← word
                EndIf
            Endfor
        Endfor
    Endfor

    Return detectedSynonyms
Endfunction
```

25 pav. Sinonimų aptikimo, taikant „WordNet“ tinklą, algoritmo pseudokodas

Algoritmo esmė yra gauti kiekvieno žodžio sinonimus kreipiantis į „WordNet“ tinklą, tada patikrinti ar gauti sinonimai sutampa su kitais likusiais žodžiais.

4.7.2. Sinonimų aptikimas taikant neuroninį tinklą „Word2vec“

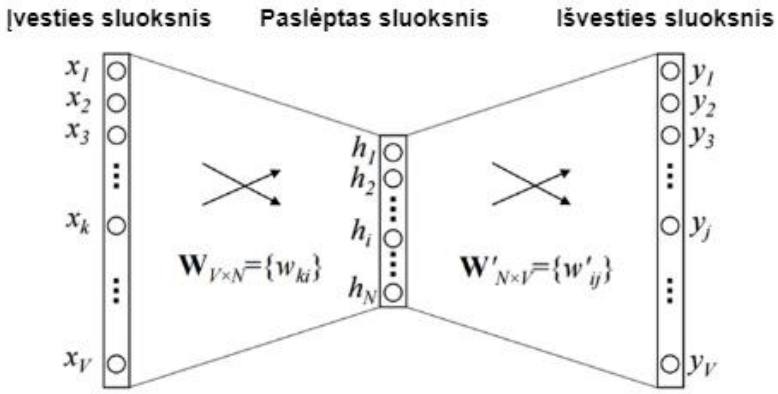
„Word2Vec“ realizavimui galima naudoti „Python“ biblioteka „Gensim“, skirta temų modeliavimui, dokumentų indeksavimui ir panašumų paieškai su dideliais tekstų rinkiniais bei CBOW modelis [37].

CBOW modelis paduoda kiekvieno žodžio kontekstą kaip įvestį ir bando nuspėti kontekstą atitinkantį žodį. Pvz.: palinkėjimas „Labas rytas, geros dienos!“.

Tegul įvestis į neuroninį tinklą būna žodis „geros“. Atkreipkite dėmesį, kad čia mes bandome nuspėti tikslinį (angl. *target*) žodį „dienos“ naudodami konteksto įvesties žodį „geros“. Įvesties žodžiui naudojame vieno vektoriaus kodavimą (angl. *one hot encoding*) ir išmatuojame išvesties klaidą, lygindami su tikslinio žodžio „dienos“ vieno vektoriaus kodavimu. Prognozuodami tikslinį žodį „dienos“ išmokstame tikslinio žodžio vektoriaus atvaizdavimą.

Pvz vieno vektoriaus kodavimas: Labas = [1,0,0,0]; rytas = [0,1,0,0]; geros = [0,0,1,0]; dienos = [0,0,0,1].

Įvesties arba konteksto žodis yra vieno vektoriaus būdu užkoduotas V dydžio vektorius. Paslėptame sluoksnyje yra N neuronų, o išvestis vėl yra V ilgio vektorius, kurio elementai yra minkštojo maksimumo (angl. *softmax*) funkcijos vertės arba kitaip tikimybės (žr. 26 pav.).

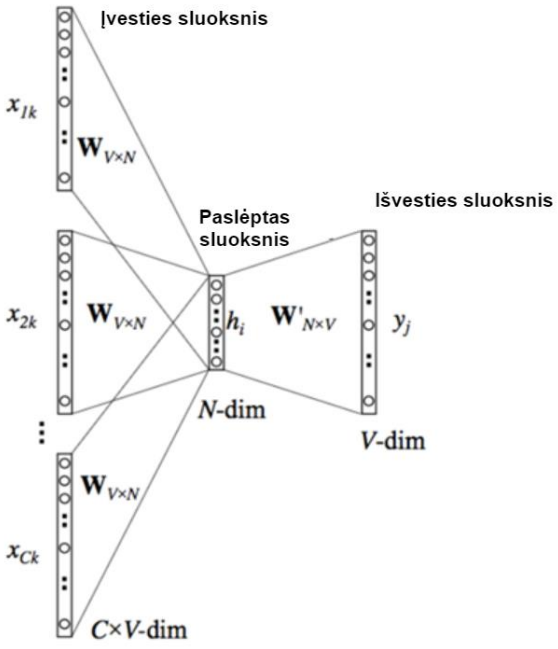


26 pav. Neuroninis tinklas

- $W_{V\times N}$ yra svorių matrica, nubrėžianti (angl. *maps*) įvestį x į paslėptą sluoksnį ($V \times N$ matmenų matrica);
- $W'_{N\times V}$ yra svorių matrica, nubrėžianti paslėpto sluoksnio išėjimus į galutinį išvesties sluoksnį ($N \times V$ matmenų matrica).

Paslėptojo sluoksnio neuronai tiesiog nukopijuoja svertinę įėjimų sumą į kitą sluoksnį. Nėra tokių aktyvacijos funkcijų kaip „sigmoid“, „tanh“ ar „ReLU“. Vienintelis netiesiškumas yra minkštojo maksimumo skaičiavimai išvestiniame sluoksnyje.

Prieš tai minėtas modelis naudojo vieną konteksto žodį, spėjimui. Mes galime panaudoti kelis konteksto žodžius. Šis modelis paima C konteksto žodžių. Kai $W_{V\times N}$ yra naudojama apskaičiuoti paslėpto sluoksnio įvestis mes imame visų šių C konteksto žodžių įvesties vidurkį (žr. 27 pav.).



27 pav. Neuroninis tinklas naudojant kelis žodžius

Algoritmo pseudokodas būtų identiškas kaip ir aptinkant sinonimus naudojant „WordNet“ semantinį tinklą (žr. 25 pav.). Tiesiog galimų sinonimų sąrašas būtų gražintas ne iš „WordNet“, o iš apmokyto neuroninio tinklo.

4.8. Neužpildytų vietos žymenų aptikimas

Neužpildytos vietos žymenas, kaip ir aptinkant blogos rašybos ar įžeidžiančių pranešimų defektus, galime taikyti žodyną. Tačiau neužpildytos vietos žymenų žodyną reikia susidaryti patiems. Neužpildytos vietos žymenų žodyną galima susidaryti ištraukiant numatytąsias reikšmes iš Android „attrs.xml“ failo bei papildant rankiniu būdu. Turint šį žodyną, reikia patikrinti ar paduotame tekste esantys žodžiai patenką į neužpildytų vietos žymenų žodyną. Žemiau pateikiamas šios algoritmo pseudokodas (žr. 28 pav.).

```
1  Function findPlaceholders(texts)
2      words ← extractWords(texts)
3      placeholders ← ∅
4
5      For each word in words
6          If isWordPlaceholder(word) == true
7              placeholders ← word
8          EndIf
9      Endfor
10
11     Return placeholders
12 Endfunction
```

28 pav. Neužpildytų vietos žymenų aptikimo pseudokodas

4.9. Per ilgų tekstų aptikimas

Kadangi jau prieš tai nagrinėtiems defektų aptikimo algoritmams reikėjo naudoti teksto charakteristikas, tai belieka šias charakteristikas panaudoti aptinkant per ilgus tekstus. Kaip jau minėjome analitinėje dalyje, daugelyje straipsnių rekomenduojama, kad maksimalus žodžių skaičius sakinyje neturėtų peržengti 25 ar 30 žodžių ribos. Šio algoritmo pseudokodas pateikiamas žemiau esančiame paveikslėlyje (žr. 29 pav.).

```
1  Function findTooLongSentences(texts)
2      sentences ← extractSentences(texts)
3      tooLongSentences ← ∅
4
5      For each sentence in sentences
6          If getWordsNumber(sentence) > 25
7              tooLongSentences ← sentence
8          EndIf
9      Endfor
10
11     Return tooLongSentences
12 Endfunction
```

29 pav. Per ilgų tekstų aptikimo pseudokodas

5. Eksperimentinė dalis

Buvo atliktas tyrimas, kurio metu rankiniu būdu buvo ištestuotos 778 mobiliosios programėlės. Šios programėlės buvo pasirinktos iš Google Play parduotuvės geriausių kategorijų, kadangi labiausiai tikėtina, jog šios programėlės yra kokybiškesnės ir kūrėjai yra įdėję daugiau pastangų užtikrindami jų kokybę. Tyrimo metu iš viso buvo aptikti 22209 defektai, kuriuos galima sugrupuoti į 59 defektų tipus. Šiuos tipus galima suskirstyti į 7 klases [38]:

- skaitomumo defektai (angl. *Readability defects*);
- suprantamumo defektai (angl. *Understandability defects*);
- išdėstymo defektai (angl. *Layout defects*);
- matomumo defektai (angl. *Visibility defects*);
- navigacijos defektai (angl. *Navigation defects*);
- prieinamumo defektai (angl. *Accessibility defects*);
- vientisumo defektai (angl. *Consistency defects*).

Norėdami eksperimentiškai palyginti rankinio testavimo rezultatus su automatinio įrankio testavimo rezultatais iš 59 defektų tipų, tyrimui buvo pasirinkti 8 defektų tipai, kurie buvo ištestuoti automatiniu būdu.:

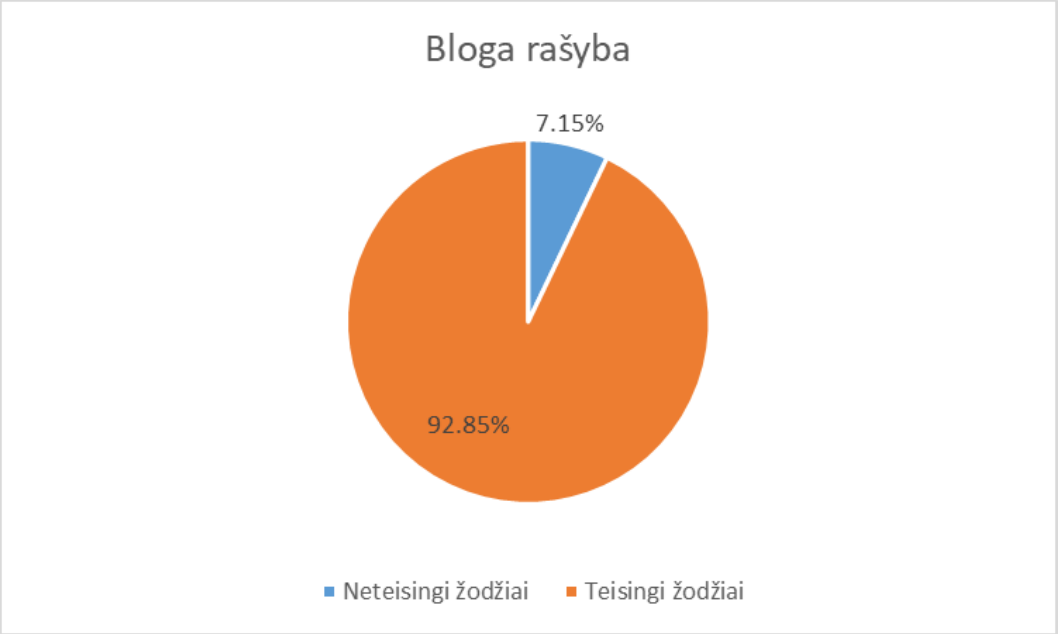
1. bloga rašyba;
2. mišri kalba;
3. sunkiai skaitomas tekstas (sudėtingas);
4. sunkiai skaitomas tekstas (spalvų konfliktavimas);
5. neužpildytos vietos žymenos;
6. per ilgi tekstai;
7. įžeidžiantys pranešimai;
8. sinonimai.

5.1. Teksto skaitymo iš APK failo rezultatai

Teksto ištraukimas buvo viena iš esminių sukurto automatinio testavimo įrankio dalių. Pirmojoje programos versijoje iš visų programėlių, naudojant 4.1 aprašytą metodą, pavyko ištraukti 583447 žodžių. Kadangi programėlės buvo atrinktos iš geriausių kategorijų tikėtina, jog programuotojai taikė gerąją programavimo praktiką. Tokiu atveju programuotojai norėdami lokalizuoti programėlę, jog ji būtų prieinama ir suprantama daugelyje šalių, naudoja resursų failus, kuriuose laiko tik tam tikros kalbos tekstus. Pavyzdžiui anglų kalbos tekstai yra pagal nutylėjimą, todėl juos būtų galima laikyti „res/values/strings.xml“ XML formato faile, o lietuvių kalbos tekstus „res/values-lt/strings.xml“ faile [40]. Atkreipdami dėmesį į tai atlikome eksperimentą ir pabandėme dekompiliuoti „resources.arsc“ failą, kuriame šie resursai yra laikomi. Dekompiliavus failą beliko parašyti atitinkamus filtrus ir ištraukti reikiamus tekstus. Tai atlikus iš programėlių pavyko ištraukti 1588998 žodžių, o tai yra 2.7 karto daugiau žodžių negu pirmuoju atveju. Kadangi testavimui reikia daug duomenų, tai palikome antrą teksto skaitymo iš APK failo metodą. Be to daugelis algoritmų buvo sukurti, jog veiktų tik su anglų kalbos tekstais, tai teksto ištraukimo metu buvo nuskaitomos tik tuos vietos, kuriuose pagal nutylėjimą yra laikomi anglų kalbos tekstai.

5.2. Blogos rašybos defekto testavimo rezultatai

Atlikus automatinį testavimą iš 1588998 žodžių buvo aptikti 113567 blogos rašybos defektai. Blogos rašybos defektai sudarė apie 7.15% analizuotų žodžių (žr. 30 pav.).



30 pav. Aptiktų blogos rašybos defektų dalis programėlėse

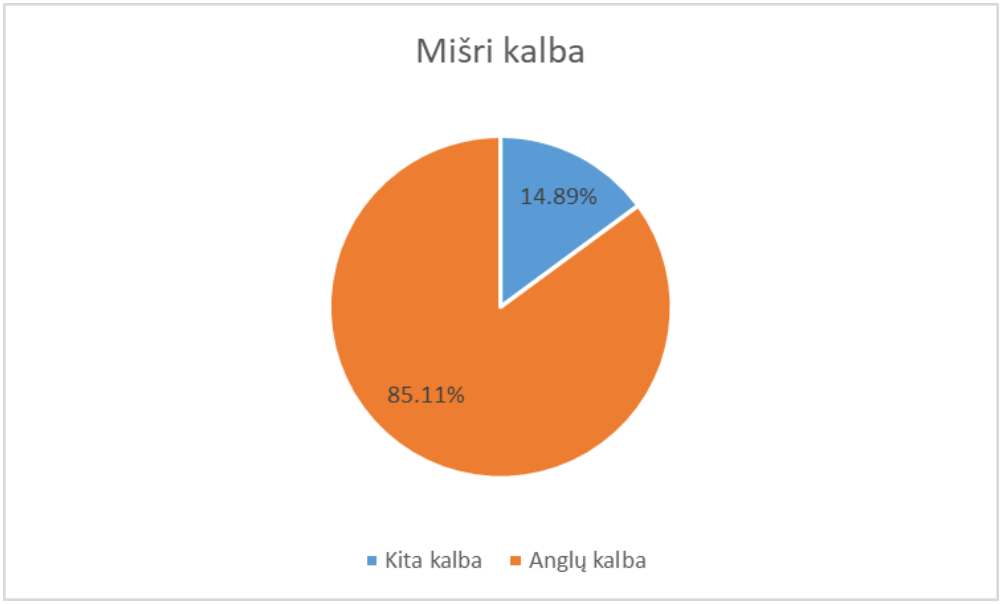
Automatiniu testavimo metu buvo iškilusios šios pagrindinės problemos, kurios lėmė didesnį defektų kiekį:

1. taikomas žodynas buvo anglų kalbos, todėl esant mišrios kalbos defektams, kitos kalbos žodžiai buvo skaičiuojami, kaip klaida;
2. tekstuose buvo naudojami akronimai, kurie į žodyną nepatenka;
3. dinamiškai atvaizduojami kintamieji pvz.: „%1$s“, „%1$d“, kurie suprantami kaip klaidos;
4. naudojamos HTML žymos pvz.: „ paragrafas “, „<h1>tai yra antraštė</h1>“, kurios taip pat yra neįtrauktos į žodyną.

Dalį minėtų problemų pavyko išspręsti, įgyvendinus papildomus teksto filtrus. Vienos programėlės blogos rašybos defektų analizė vidutiniškai užtruko 3s.

5.3. Mišrios kalbos defekto testavimo rezultatai

Atlikus automatinį testavimą iš 306455 sakinių buvo aptikti 45644, kurie buvo parašyti kita kalba. Tai sudarė apie 14.89% visų sakinių (žr. 31 pav.).



31 pav. Aptiktų mišrios kalbos defektų dalis programėlėse

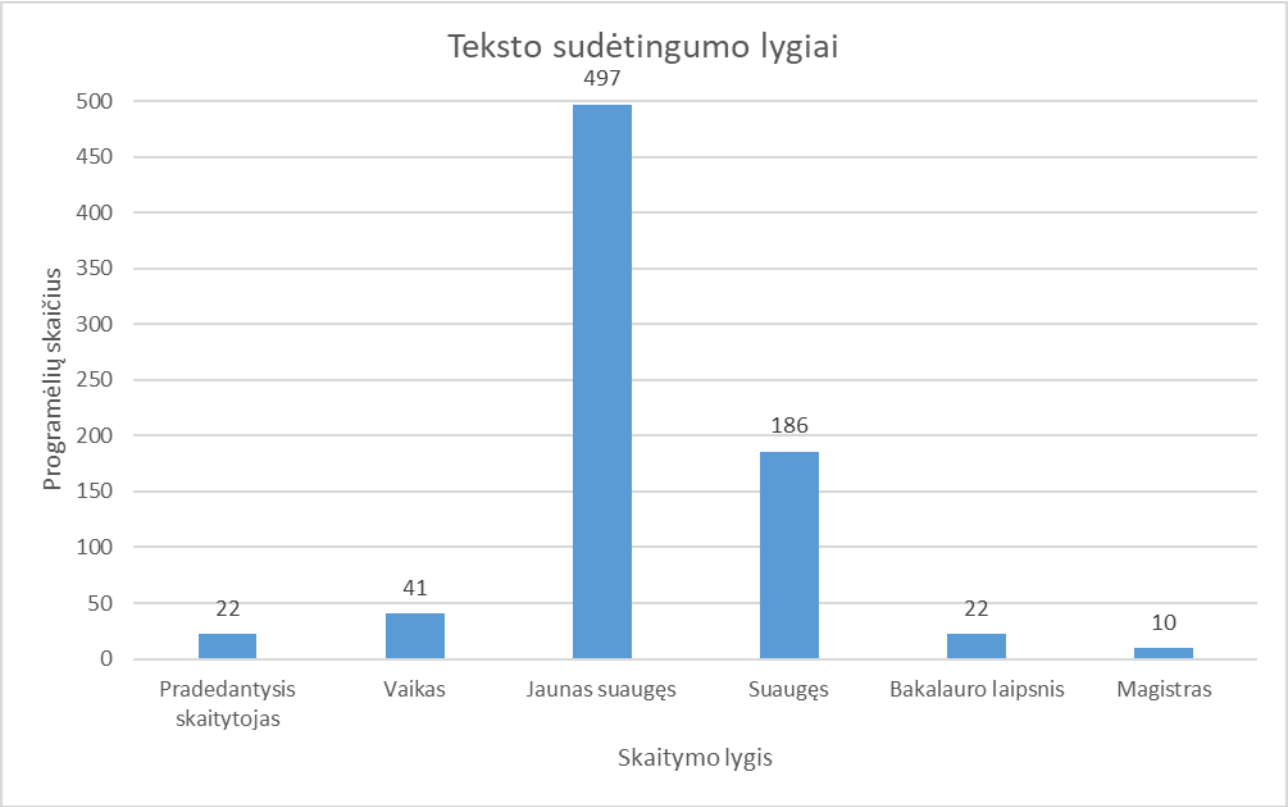
Pagrindiniai aspektai, kurie apsunkino algoritmo efektyvumą yra šie:

1. pasitaikė keletas programėlių, kurios pagal nutylėjimą buvo ne anglų kalba, todėl visos programėlės sakiniai buvo traktuojami kaip defektai;
2. dažnai programėlėse pateikti tekstai yra labai trumpi iš 2 ar 3 žodžių, todėl algoritmo tikslumas suprastėja.

Vieną programėlę išanalizuoti vidutiniškai užtruko 0.3s.

5.4. Sunkiai skaitomo teksto (sudėtingo) testavimo rezultatai

Atlikus automatinį testavimą iš 778 mobiliųjų programėlių buvo aptiktos 33, kurių tekstai buvo labai sudėtingi. Žemiau pateikiamas grafikas, kuriame atvaizduojamas teksto sudėtingumų lygių pasiskirstymas visuose programėlėse (žr. 32 pav.). Šios kategorijos buvo išskirtos atsižvelgiant į (žr. 2 lentelė) pateiktą lentelę.



32 pav. Teksto sudėtingumo lygių pasiskirstymas programėlėse

Skaitomumo indeksui nustatyti pakako ištraukti teksto charakteristikas nusakančias žodžių, sudėtingų žodžių ir sakinių skaičių, todėl algoritmo įgyvendinimui didelių problemų neiškilo. Vienos programėlės sudėtingumo lygio nustatymas vidutiniškai užtruko apie 0.27s.

5.5. Sunkiai skaitomo teksto (konfliktuojančių spalvų) testavimo rezultatai

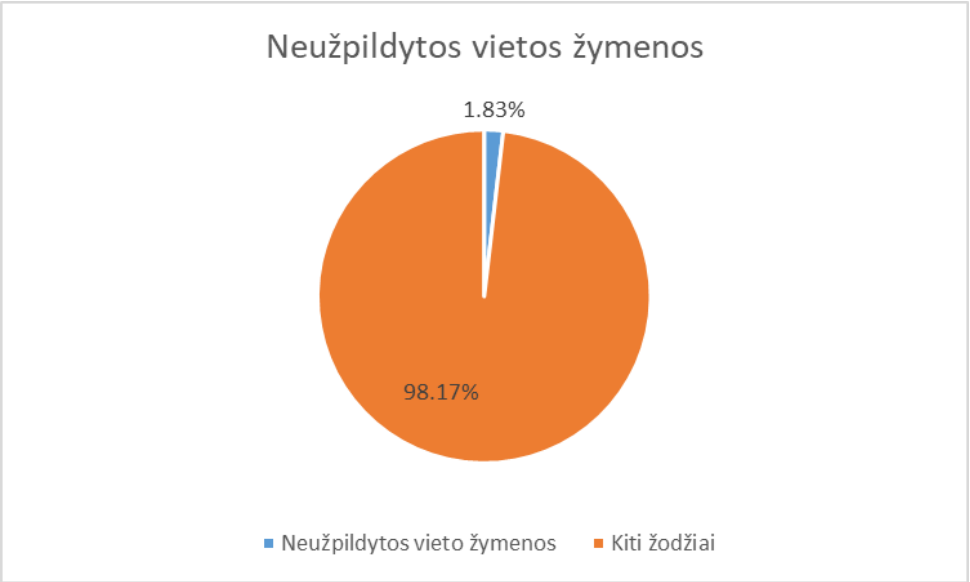
Atlikus automatinį testavimą 778 programėlėse buvo aptiktos 76715 konfliktuojančios spalvų poros, kurių kontrastų santykis buvo mažesnis nei 4. Algoritmo įgyvendinimo metu buvo iškilusios šios pagrindinės problemos:

1. spalvos buvo pateikiamos įvairiais formatais pvz.: RGB, HEX, HSL;
2. reikėjo įsitikinti, jog spalvos priklauso tam pačiam elementui ir yra susijusios.

Problemoms išspręsti buvo įgyvendintas spalvų formato konverteris bei parašyti specialūs filtrai spalvų susiejimo nustatymui. Konfliktuojančių spalvų analizė vienai programėlei vidutiniškai užtruko 0.31s.

5.6. Neužpildytų vietos žymenų aptikimo testavimo rezultatai

Atlikus automatinį testavimą iš 1588998 žodžių buvo aptikti 29136, kurie buvo laikomi kaip neužpildytos vietos žymenos. Neužpildytos vietos žymenos sudarė apie 1.83% analizuotų žodžių (žr. 33 pav.).

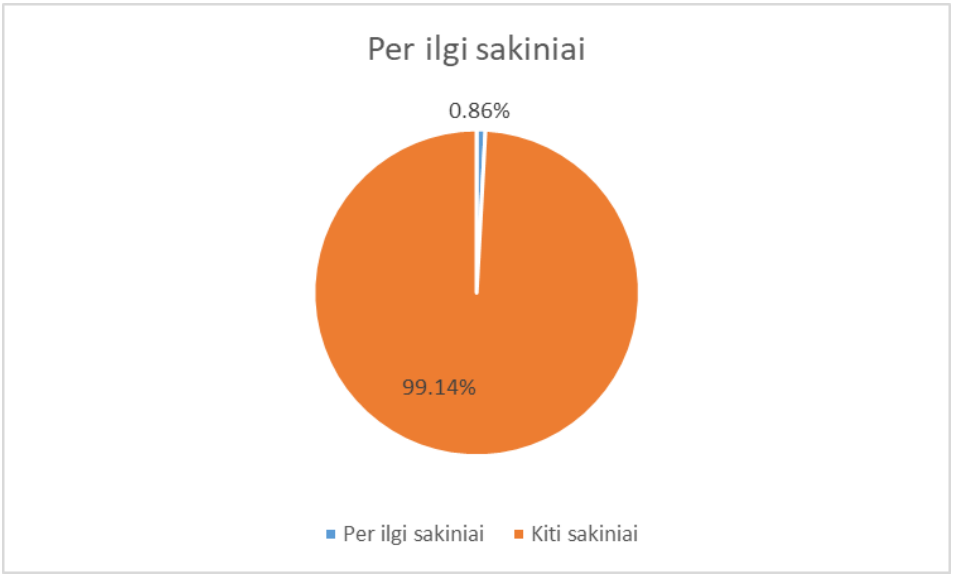


33 pav. Neužpildytos vietos žymenų kiekio dalis programėlėse

Kaip ir blogos rašybos aptikimui, neužpildytoms vietos žymenoms aptikti buvo taikomas žodynas. Tačiau iškilusios problemos blogos rašybos aptikimui neturėjo įtakos neužpildytos vietos žymenų aptikimui. Taip yra todėl, kad neužpildyta vietos žymena yra laikoma kaip defektas tik tada, kada žodis patenka į žodyną, priešingai nei blogos rašybos aptikime, kada klaida traktuojama iškarto kai žodis nepatenka į žodyną. Vienos programėlės analizė vidutiniškai užtruko 0.0002s. Algoritmas buvo daug kartu greitesnis už blogos rašybos aptikimo, kadangi jis yra tiesiogiai susijęs su žodyno dydžiu, šiuo atveju neužpildytos vietos žymenų žodynas buvo labai mažas palyginus su visos anglų kalbos žodynu.

5.7. Per ilgų tekstų aptikimo testavimo rezultatai

Atlikus automatinį testavimą iš 306455 sakinių buvo aptikti 2640, kurie buvo sudaryti iš daugiau kaip 30 žodžių. Tai sudarė apie 0.86% analizuotų sakinių (žr. 34 pav.).



34 pav. Per ilgų sakinių dalis programėlėse

Kadangi reikėjo tik suskaičiuoti ištraukto teksto charakteristikas, tai algoritmo įgyvendinimo metu, didelių problemų neiškilo. Algoritmas viena programėlę vidutiniškai išanalizuodavo per 0.00025s.

5.8. Sinonimų paieškos testavimo rezultatai

Sinonimams aptikti buvo naudojami keli sprendimo būdai:
1. leksinė duomenų bazė „WordNet“;
2. neuroninis tinklas „Word2vec“.

5.8.1. Sinonimų paieškos testavimo rezultatai taikant leksinę duomenų bazę „WordNet“

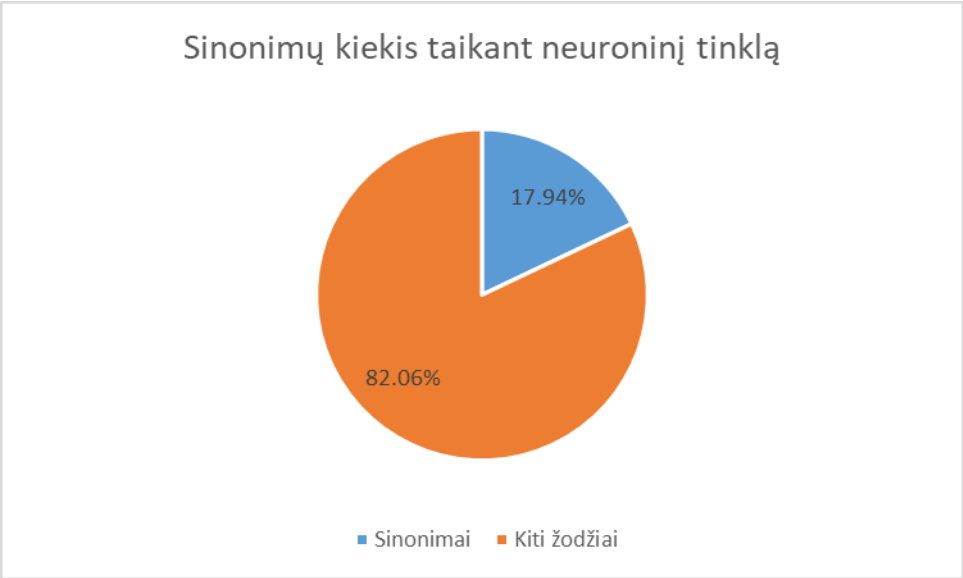
Atlikus automatinį testavimą iš 1588998 žodžių buvo aptikti 195095 sinonimai. Tai sudarė apie 12.28% analizuotų žodžių (žr. 35 pav.).

35 pav. Sinonimų kiekio dalis programėlėse taikant „WordNet“

Algoritmo įgyvendinimo metu, didelių problemų neiškilo, kadangi „WordNet“ yra gerai išplėtota leksinė duomenų bazė, kuri gražindavo gana patikimus sinonimų sąrašus. Algoritmas viena programėlę vidutiniškai išanalizuodavo per 0.32s.

5.8.2. Sinonimų paieškos testavimo rezultatai taikant neuroninį tinklą „Word2vec“

Atlikus automatinį testavimą iš 1588998 žodžių buvo aptikti 285142 sinonimai. Tai sudarė apie 17.94% analizuotų žodžių (žr. 36 pav.).



36 pav. Sinonimų kiekio dalis programėlėse taikant neuroninį tinklą

Automatiniu testavimo metu buvo iškilusios šios pagrindinės problemos:

1. modelio metodai buvo labai lėti ir netikslūs;
2. apmokytas modelis sinonimus suprasdavo pagal tai, kokiam kontekste tas žodis būdavo. Pavyzdžiui žodis „šaltas“ ir „karštas“ dažnai pasitaikys tame pačiame kontekste, bet tai nereiškia, jog jie yra sinonimai;
3. norint, kad modelis būtų tikslesnis reikia labai daug duomenų. Modelis buvo apmokytas naudojant Vikipedijos duomenų kopiją, tačiau to vis tiek neužteko geriems rezultatams gauti.

Vienos programėlės blogos rašybos defektų analizė vidutiniškai užtruko 140s.

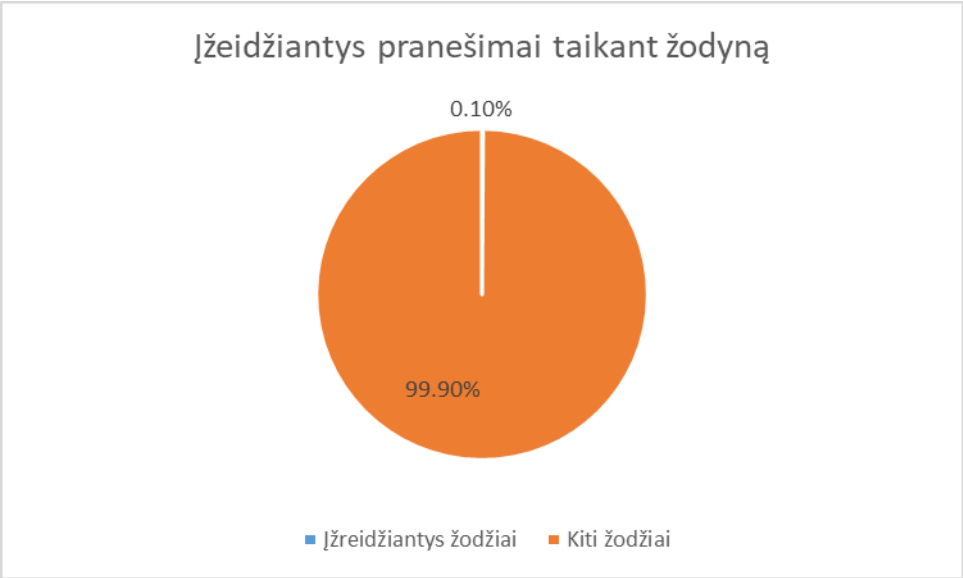
5.9. Įžeidžiančių pranešimų paieškos testavimo rezultatai

Įžeidžiantiems pranešimams aptikti buvo naudojami keli sprendimo būdai:

1. žodyno taikymas;
2. logistinė regresija.

5.9.1. Įžeidžiančių pranešimų testavimo rezultatai taikant žodyną

Atlikus automatinį testavimą iš 1588998 žodžių buvo aptikti 1525 įžeidžiantys. Tai sudarė apie 0.10% visų analizuotų žodžių (žr. 37 pav.).



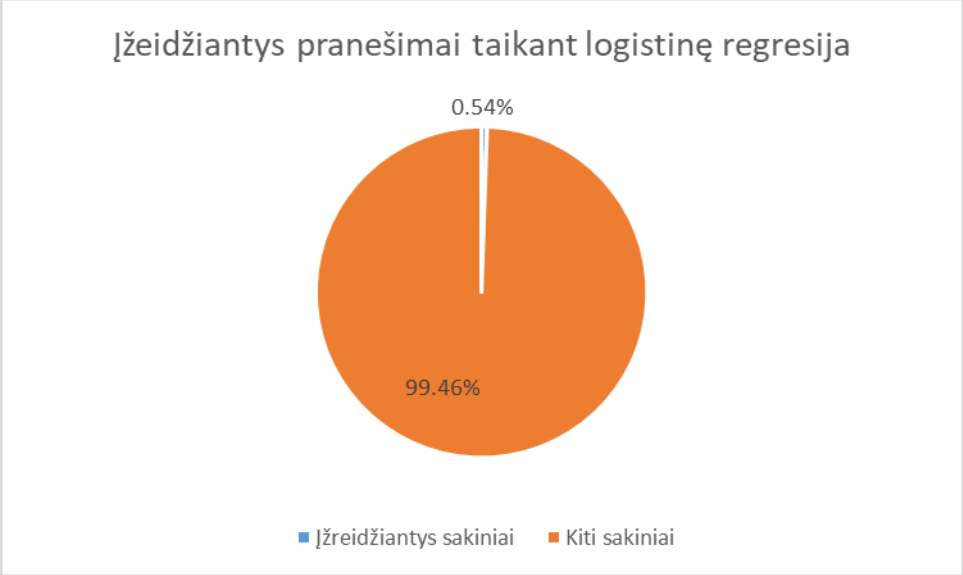
37 pav. Įžeidžiančių pranešimų kiekio dalis programėlėse taikant žodyną

Automatiniu testavimo metu buvo iškilusi problema, jog žodžiai buvo laikomi įžeidžiantys, nežinant kokiame kontekste tie žodžiai vartojami. Pavyzdžiui, angliškas žodis „fat“ gali būti naudojamas nusakyti žmogaus riebalinę masę, tačiau naudojamas programėlėje, kuri yra skirta sveikai mitybai ir gyvensenai, toks žodis nėra įžeidžiantis, o tiesiog pasakantis tam tikrą informaciją, pagal kurią asmuo gali atitinkamai keisti savo mitybą ar sportą.

Algoritmas viena programėlę vidutiniškai išanalizuodavo per 0.025s.

5.9.2. Įžeidžiančių pranešimų testavimo rezultatai taikant logistinę regresiją

Atlikus automatinį testavimą iš 306455 sakinių buvo aptikti 1641 įžeidžiantys. Tai sudarė apie 0.54% analizuotų sakinių (žr. 38 pav.).



38 pav. Įžeidžiančių pranešimų kiekio dalis programėlėse taikant logistinę regresiją

Automatiniu testavimo metu buvo iškilusi problema, jog apmokytas modelis negeba atskirti kultūriniu išsireiškimų, pavyzdžiui, angliškas išsireiškimas „you're a bad ass!“, kuris nėra įžeidžiantis, bet yra traktuojamas kaip įžeidžiantis.

Vienos programėlės blogos rašybos defektų analizė vidutiniškai užtruko 2.5s.

5.10. Klaidingai teigiamų įžeidžiančių pranešimų rezultatų validacija

Norėdami patikrinti įžeidžiančių pranešimų algoritmo tikslumą taikant žodyną, rankiniu būdu buvo patikrinti sakiniai, kuriuose buvo rasti galimai įžeidžiantys žodžiai. Keletas galimai įžeidžiančių sakinių yra pateikiami žemiau esančioje lentelėje (žr. 7 lentelė). Rankinių būdu buvo peržiūrėtas kiekvienas sakinys ir nuspręsta ar jis tikrai yra įžeidžiantis.

7 lentelė. Aptiktų įžeidžiančių pranešimų validacijos pavyzdžiai

Analizuotas sakinys, kuris laikomas įžeidžiančiu	**Ar įžeidžiantis?**
Burn body fat with high intensity interval workouts.	ne
Reduce body fat.	ne
Homosexual.	taip
Niger.	ne
Japanese.	ne
Reduce body fat.	ne
What's your sexual orientation?	taip
How to tell whether your baby is getting enough breast milk.	ne
Get exclusive access to the most popular girls.	taip

Iš pateiktų pavyzdžių matosi, kad nemažai sakinių vis dėl to nėra įžeidžiantys. Įvertinus testavimo rezultatus rankiniu būdu, buvo nustatyta, kad apie 45% rezultatų yra klaidingi. Kaip ir minėjome 5.9.1 skyriuje, pagrindinė problema testavimo metu buvo tokia, jog algoritmas neatsižvelgdavo į tai kokiame kontekste tas žodis yra vartojamas. Dėl šios problemos šalių ir kalbos pavadinimai buvo suprantami kaip įžeidžiantys, nors šie žodžiai buvo vartojami po vieną. Kadangi jie vartojami po vieną, tai pasako, jog šie žodžiai buvo naudojami tiesiog pasirenkant šalį ar kalbą, o ne įžeidžiant kitą asmenį. Nemažai žodžių pasitaikė, kurie yra naudojami sporto ar medicinos srityje ir tame kontekste esantys žodžiai dažniausiai nėra įžeidžiantys. Kai kuriuos rezultatus buvo sunku įvertinti ir rankiniu būdu, kadangi iš pateikto sakinio, kai kurie atvejai vis tiek nebuvo akivaizdūs. Pavyzdžiui užduotas klausimas „Kokia yra jūsų seksualinė orientacija?“ medicinos tikslais nėra įžeidžiantis, tačiau jeigu jis bus užduotas užpildant informaciją apie save programėlėje, kurioje yra talpinami darbo skelbimai, vartotojas gali įsižeisti ir jaustis diskriminuojamas.

5.11. Visų defektų testavimo rezultatų apibendrinimas

Reikia atkreipti dėmesį į tai, kad rankiniu būdu buvo testuojamos programėlių ekranų kopijos. Tai reiškia, kad viename paveikslėlyje galėjo būti keletas defektų. Automatiniu testavimo metu rezultatai buvo pateikiami žodžių, sakinių ar programėlių skaičiumi. Taip pat dalis defektų tipų rankinio testavimo metu nebuvo testuojami. Žemiau pateikiami rankinio ir automatinio testavimo rezultatai (žr. 8 lentelė).

8 lentelė. Aptiktų defektų rezultatai

Defekto tipas	**Taikytas sprendimas**	**Aptiktų defektų kiekis rankinio testavimo metu (ekrano kopijų skaičius)**	**Aptiktų defektų kiekis po automatinio testavimo**
Bloga rašyba	Žodyno taikymas	11	113567 žodžių
Mišri kalba	Biblioteka „Lingua“	1251	45644 sakinių
Sunkiai skaitomas tekstas (sudėtingas)	Rūko indeksas	0	32 programėlių
Sunkiai skaitomas tekstas (spalvų konfliktavimas)	Spalvų kontrastų santykis	420	76715 spalvų porų
Neužpildytos vietos žymenos	Žodyno taikymas	30	29136 žodžių
Per ilgi tekstai	Teksto charakteristikų analizavimas	0	2640 sakinių
Įžeidžiantys pranešimai	Žodyno taikymas	0	1525 žodžių
Įžeidžiantys pranešimai	Mašininis mokymas	0	1641 sakinių
Sinonimai	Leksikos duomenų bazės „WordNet“ taikymas	0	195095 žodžių
Sinonimai	Mašininis mokymas	0	285142 žodžių

Pagal rezultatus matome, kad automatiniu būdu buvo aptikta daug kartų daugiau defektų. Jeigu laikysime, kad viename programėlės lange buvo rasti vidutiniškai 3 defektai, tai iš eksperimentui pasirinktų defektų, rankiniu būdu išviso buvo aptikti 4002 defektai. Tuo tarpu automatinio testavimo metu buvo rasti 270900 defektai (neskaičiuojant papildomų eksperimentų taikant mašininį mokymą). Taip pat rankiniu būdu visų defektų testavimas užtruko iki kelių mėnesių, o automatiniu būdu, visų minėtų defektų tipų, testavimas užtruko apie 2 valandas. Reikėtų atkreipti dėmesį ir į tai, kad testuojant automatiniu būdu tekstai buvo ištraukti iš programėlės resursų, kurie galbūt galėjo būti net nenaudojami ir matomi programėlės naudotojui.

5.12. Ateities darbai

Automatiniu būdu ištestavus 778 programėles išryškėjo pagrindinės problemos, kurios atsiranda aptinkant tam tikrus defektų tipus. Buvo pastebėta, kad didelė dalis efektyvumo priklauso ne tik nuo pačio algoritmo tikslumo, bet ir nuo įrankio gebėjimo išfiltruoti reikiamus tekstus, jog algoritmams būtų pateiktas kuo mažiau klaidinantis tekstas. Ateityje būtų galima sukurti atskirą komponentą, kuris būtų atsakingas už teksto skaitymą. Komponente galėtų būti kiekvienam defekto tipui atskiras taisyklių failas, kuris būtų atsakingas už tai, kad algoritmus pasiektų išfiltruoti ir aktualūs tekstai. Taip pat automatinio testavimo metu gauti rezultatai galės būti naudojami kaip etalonas tolimesniam algoritmų tobulinimui.

6. Išvados

1. Atlikus srities apžvalgą buvo rasta daug įrankių, kurie gali aptikti tam tikrus neteisingos kalbos ar lokalizacijos defektus, tačiau nei vienas įrankis nesugebėjo aptikti daugiau nei vieno defekto tipo.
2. Atsižvelgus į metodų pliusus ir minusus, problemos sprendimui buvo atrinkti geriausiai tinkantys metodai.
3. Atlikus analizę buvo pastebėtos įgyvendinimo problemomis su kuriomis gali tekti susidurti automatinio įrankio kūrimo metu: mašininio mokymo modelių apmokymas reikalauja daug resursų, įrankio efektyvumas priklauso ne tik nuo metodo efektyvumo aptinkant defektus, bet ir nuo įrankio efektyvumo ištraukiant programėlės tekstus, gali būti sunku pritaikyti visus defektų aptikimo metodus vienoje aplinkoje.
4. Atlikus tyrimą buvo plačiau išanalizuoti metodų veikimo principai, bei pateikti galimi sprendimų pseudokodai. Norėdami atlikti daugiau eksperimentų, sinonimų ir įžeidžiančios kalbos aptikimui, papildomai buvo atrinkti mašininio mokymo sprendimai.
5. Eksperimentiškai įvertinus algoritmus paaiškėjo, kad automatinis testavimas yra ženkliai efektyvesnis vertinant greitaveikos ir aptiktų defektų kiekio atžvilgiu. Rankiniu būdu išviso buvo aptikti 4002 defektai, o automatinio testavimo metu buvo rasti 270900 defektai. Testavimas visų tekstinių defektų rankiniu būdu užtruko iki kelių mėnesių, o pasirinktų defektų testavimas automatu užtruko iki 2 valandų.
6. Kadangi buvo testuojamas labai didelis duomenų kiekis, yra labai sunku įvertinti pateiktų rezultatų validumą, nes testuojamos realios programėlės ir nėra žinoma kiek iš tikrųjų yra neteisingos kalbos ir lokalizacijos defektų.
7. Nors dabartinė įrankio versija pateikia ir klaidingai teigiamų rezultatų, tačiau tai vis tiek ženkliai palengvina testuotojo darbą, kadangi nereikia peržiūrėti visos programėlės rankiniu būdu. Testuotojas gali susitelkti tik į tas programėlės dalis, kuriuose įrankis aptinka defektus.
8. Buvo pastebėta su kokiais sunkumais susiduriama nuskaitant programėlės resursus, bei kokios problemos dažniausiai kyla aptinkant tekstinius defektus. Šie pastebėjimai palengvins tolimesniam netaisyklingos kalbos ir lokalizacijos defektų aptikimų metodų tyrinėjimui.

7. Literatūros sąrašas

1. Ana Lozančić, „Benefits of Software Testing“ [Tinkle]. Prieiga per internetą: https://gauss-development.com/benefits-software-testing. [kreiptasi 2019-11-02]
2. KHAN, Mohd Ehmer, et al. A comparative study of white box, black box and grey box testing techniques. Int. J. Adv. Comput. Sci. Appl, 2012, 3.6.
3. „What is WHITE Box Testing? Techniques, Example, Types & Tools“ [Tinkle]. Prieiga per internetą: https://www.guru99.com/white-box-testing.html. [kreiptasi 2019-11-02]
4. KHAN, Mohd Ehmer, et al. Different approaches to white box testing technique for finding errors. International Journal of Software Engineering and Its Applications, 2011, 5.3: 1-14.
5. „What is Grey Box Testing? Techniques, Example“ [Tinkle]. Prieiga per internetą: https://www.guru99.com/grey-box-testing.html. [kreiptasi 2019-11-02]
6. HUSSAIN, Taraq; SINGH, Satyaveer. A Comparative Study of Software Testing Techniques Viz. White Box Testing Black Box Testing and Grey Box Testing. IJAPRR), ISSN, 2015, 2350-1294.
7. „What is BLACK Box Testing? Techniques, Example“ & Types [Tinkle]. Prieiga per internetą: https://www.guru99.com/black-box-testing.html. [kreiptasi 2019-11-03]
8. „What is Non Functional Testing? Types with Example“ [Tinkle]. Prieiga per internetą: https://www.guru99.com/non-functional-testing.html. [kreiptasi 2019-11-03]
9. JANSEN, Bernard J. The graphical user interface. ACM SIGCHI Bulletin, 1998, 30.2: 22-26.
10. CHILLAREGE, Ram, et al. Orthogonal defect classification-a concept for in-process measurements. IEEE Transactions on software Engineering, 1992, 18.11: 943-956.
11. IEEE Standard Classification for Software Anomalies. IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993), 2010: p. 1-23
12. Beizer, B., Software testing techniques. 2nd ed. Van Nostrand Reinhold electrical/computer science and engineering series. 1990, New York: Van Nostrand Reinhold. xviii, 290 p.
13. LELLI, Valéria; BLOUIN, Arnaud; BAUDRY, Benoit. Classifying and qualifying GUI defects. In: 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST). IEEE, 2015. p. 1-10.
14. PACKEVIČIUS, Šarūnas, et al. The testing method based on image analysis for automated detection of UI defects intended for mobile applications. In: International Conference on Information and Software Technologies. Springer, Cham, 2015. p. 560-576.
15. MORAN, Kevin, et al. Automated reporting of GUI design violations for mobile apps. In: Proceedings of the 40th International Conference on Software Engineering. 2018. p. 165-175.
16. JOHNSON, Jeff. GUI bloopers 2.0: common user interface design don'ts and dos. Elsevier, 2007.
17. PACKEVIČIUS, Šarūnas, et al. Text Semantics and Layout Defects Detection in Android Apps Using Dynamic Execution and Screenshot Analysis. In: International Conference on Information and Software Technologies. Springer, Cham, 2018. p. 279-292.
18. „Automation Testing Vs. Manual Testing: What’s the Difference?“ [Tinkle]. Prieiga per internetą: https://www.guru99.com/difference-automated-vs-manual-testing.html [kreiptasi 2021-05-17]
19. „LanguageTool“ [Tinkle]. Prieiga per internetą: https://en.wikipedia.org/wiki/LanguageTool [kreiptasi 2021-05-17]

20. „Grammar and Spell Checker — LanguageTool“ [Tinkle]. Prieiga per internetą: https://chrome.google.com/webstore/detail/grammar-and-spell-checker/oldceeleldhonbafppcapldpdifcinji [kreiptasi 2021-05-17]
21. Shubham Aggarwal, „Introduction to Lucene“ [Tinkle]. Prieiga per internetą: https://linuxhint.com/introduction-to-lucene [kreiptasi 2021-05-17]
22. Peter M. Stahl, „Lingua“ [Tinkle]. Prieiga per internetą: https://github.com/pemistahl/lingua [kreiptasi 2021-05-17]
23. „Language Detection Library for Java“ [Tinkle]. Prieiga per internetą: https://github.com/optimaize/language-detector [kreiptasi 2021-05-17]
24. GUNNING, Robert. The fog index after twenty years. Journal of Business Communication, 1969, 6.2: 3-13.
25. Karmakar, Saurav, and Ying Zhu. "Visualizing multiple text readability indexes." 2010 International Conference on Education and Management Technology. IEEE, 2010.
26. „SMOG“ [Tinkle]. Prieiga per internetą: https://en.wikipedia.org/wiki/SMOG. [kreiptasi 2021-05-17]
27. „SMOG Index“ [Tinkle]. Prieiga per internetą: https://www.webfx.com/tools/read-able/smog-index.html. [kreiptasi 2021-05-17]
28. Justin Baker, „The Science of Color Contrast — An Expert Designer’s Guide“ [Tinkle]. Prieiga per internetą: https://medium.muz.li/the-science-of-color-contrast-an-expert-designers-guide-33e84c41d156. [kreiptasi 2021-05-17]
29. „Contrast Checker“ [Tinkle]. Prieiga per internetą: https://webaim.org/resources/contrastchecker/. [kreiptasi 2021-05-17]
30. Saishruthi Swaminathan, „Logistic Regression — Detailed Overview“ [Tinkle]. Prieiga per internetą: https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc. [kreiptasi 2021-05-17]
31. „Identify the Language of any Text“ [Tinkle]. Prieiga per internetą: https://www.plnia.com/products/language-detection-api/. [kreiptasi 2021-05-17]
32. MILLER, George A. WordNet: a lexical database for English. Communications of the ACM, 1995, 38.11: 39-41.
33. RONG, Xin. word2vec parameter learning explained. arXiv preprint arXiv:1411.2738, 2014.
34. CAVNAR, William B., et al. N-gram-based text categorization. In: Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval. 1994.
35. „Colour Luminance and Contrast Ratio“ [Tinkle]. Prieiga per internetą: https://www.101computing.net/colour-luminance-and-contrast-ratio/. [kreiptasi 2021-05-17]
36. Ayush Pant, „Introduction to Logistic Regression“ [Tinkle]. Prieiga per internetą: https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148. [kreiptasi 2021-05-17]
37. Dhruvil Karani, „Introduction to Word Embedding and Word2Vec“ [Tinkle]. Prieiga per internetą: https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa. [kreiptasi 2021-05-17]
38. PACKEVIČIUS, Šarūnas; RUDŽIONIENĖ, Greta; BAREIŠA, Eduardas. Automated Visual Testing of Application User Interfaces Using Static Analysis of Screenshots. International Journal of Software Engineering and Knowledge Engineering, 2021, 31.02: 167-191.

39. grafinės vartotojo sąsajos (angl. *graphical user interface*)
40. „Localize your app“ [Tinkle]. Prieiga per internetą: https://developer.android.com/guide/topics/resources/localization. [kreiptasi 2021-05-17]

8. Terminų ir santrumpų žodynas

Santrumpos:

GUI (angl. *Graphical user interface*) – grafikos priemonėmis pagrįsta sąsaja tarp žmogaus ir kompiuterio;

API (angl. *Application programming interface*) – tai sąsaja, kurią suteikia kompiuterinė sistema, biblioteka ar programa tam, kad programuotojas per kitą programą galėtų pasiekti jos funkcionalumą ar apsikeistų su ja duomenimis;

APK (angl. *Android application package*) – failas su APK failo plėtiniu yra Android paketų failas, kuris naudojamas platinti programas Android operacinėje sistemoje;

XML (angl. *Extensible Markup Language*) – tai paprasti tekstiniai failai , kurie nieko nedaro savyje, išskyrus duomenų apie transportavimą, struktūrą ir saugojimą apibūdinimą;

TXT (angl. *TeXT*) – teksto failo plėtinys, kurį naudoja įvairūs teksto redaktoriai;

RGB (angl. *Red Green Blue*) – spalvų modelis, kuriame raudona, žalia ir mėlyna šviesa įvairiais būdais pridedama, jog būtų sukurtos kitos spalvos;

HEX – spalvos, kurios yra atvaizduojamos iš įvairių spalvų modelių per šešioliktaines reikšmes;

HSL (angl. *Hue Saturation Lightness*) **–** spalva, kurią galima nurodyti naudojant atspalvį, sodrumą ir lengvumą.

Terminai:

Testavimo atvejis – testavimo atvejis yra veiksmų, atliktų tam tikrai jūsų programinės įrangos ypatybei ar funkcionalumui patikrinti, rinkinys.

Atviro kodo – Tai kompiuterinė programa, kuri platinama pagal atvirojo kodo licenciją. Tokia programinė įranga dažniausiai pasižymi šiomis savybėmis: nemokama, laisvai prieinami išeities kodai, galima laisvai platinti ir modifikuoti, nekeičiant licencijos.

Vietos žymena (angl. *Placeholder*) – vietos žymena yra simbolis, žodis ar simbolių eilutė, laikinai užimanti galutinius duomenis.

Psicholingvistika – kalbotyros šaka, tirianti kalbą kaip psichikos reiškinį.