KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Julius Purvinis

Žiniatinklio informacinių sistemų regresinio testavimo algoritmo realizavimas ir tyrimas

Magistro darbas

Darbo vadovas
prof. Eduardas Bareiša

Kaunas, 2012

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Julius Purvinis

Žiniatinklio informacinių sistemų regresinio testavimo algoritmo realizavimas ir tyrimas

Magistro darbas

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

Darbo vadovas:
prof. Eduardas Bareiša
2012-05-25

Atliko:
IFM-0/2 gr. stud.
Julius Purvinis
2012-05-25

Kaunas, 2012

Implementation and research of regression testing algorithm in web information systems

SUMMARY

Common usage of web information systems among consumers, business and governmental organizations require higher quality of such systems. Web information systems is such a software, which has high complexity because of the multiple technology usage and tends to rapid changes. Errors may occur because of the changes that have been made through maintenance period. Regression testing purpose is to ensure, that software changes such as new functionality or previously existed functionality modification, did not affect functionality, which was well working in the older version. In these master thesis we analyze test oracle comparators which can compare HTML response. HTML as a text comparison produces too many *false positives*. Meanwhile our test oracle comparator analyses HTML response as is was a tree structure. Thus implementation of the test oracle comparator algorithm enables opportunity to validate or not to validate described HTML semantic features. Therefore, we get fewer *false positives* and *false negatives*. Our implemented test oracle comparator differs from the ones described in literature or the ones proposed in market by capability not to warn tester about changes such as newly added HTML tag, which do not change overall tree structure. In this way the introduction of natural functionality growth is allowed. After improvement our test oracle comparator detects 90 % errors with the 90 % *precision*, whereas in the market existing test oracle comparator of the regression testing tool barely reaches 66,67 % precision with the 60 % *recall*.

Žiniatinklio informacinių sistemų regresinio testavimo algoritmo realizavimas ir tyrimas

SANTRAUKA

Kasdieninis žiniatinklio informacinių sistemų (toliau – IS) naudojimas paprastų vartotojų, verslo ir vyriausybinių organizacijų tarpe, reikalauja vis aukštesnės tokių IS kokybės. Žiniatinklio IS dažniausiai yra tokia programinė įranga, kurios sudėtingumas, dėl keleto apjungiamų technologijų, yra aukštas, o pokyčių greitis yra didelis. Vykdant palaikymo ir priežiūros darbus, gali atsirasti begalė klaidų, kurias iššaukia atlikti pakeitimai. Regresinio testavimo paskirtis yra užtikrinti, jog įdiegus programinės įrangos pakeitimus, tokius kaip naujo funkcionalumo pridėjimas ar jau egzistuojančio taisymas, ankstesnėje IS versijoje veikęs funkcionalumas veiks taip pat gerai ir naujojoje versijoje. Šiame darbe tiriami iš HTML žymų sudarytą IS atsaką galintys palyginti regresinio testavimo algoritmai (toliau – testų orakulai). HTML ãtsako kaip teksto lyginimas pateikia per daug klaidingai teigiamų (angl. *false positives*) rezultatų – testuotojas užverčiamas pranešimais apie netikras klaidas. Mūsų realizuotas testų orakulas HTML žymų atsaką nagrinėja kaip medžio struktūrą. Toks algoritmo veikimas leidžia tikrinti arba netikrinti aprašytas HTML semantines savybes, ko pasekoje yra sumažinamas klaidingai teigiamų ir klaidingai neigiamų rezultatų kiekis. Realizuotas testų orakulas nuo literatūroje aprašomų ir rinkoje siūlomų testų orakulų skiriasi tuo, jog nepraneša apie klaidą, kai HTML atsakas būna papildytas naujomis žymomis, kurios nekeičia pradinės medžio struktūros. Žiniatinklio IS yra leidžiamas natūralus funkcionalumo padidėjimas. Taigi, po patobulinimo mūsų realizuotas testų orakulas aptinka 90 % klaidų su 90 % tikslumu. Tuo tarpu rinkoje esančio regresinio testavimo įrankio testų orakulo tikslumas tesiekia 66,67 %, o klaidų aptikimas lygus 60 %.

TURINYS

Paveikslėlių sąrašas

Lentelių sąrašas

Formulių sąrašas

ĮVADAS

Kasdieninis žiniatinklio IS naudojimas paprastų vartotojų, verslo ir vyriausybinių organizacijų tarpe reikalauja vis aukštesnės tokių informacinių sistemų kokybės [24]. Didėja žiniatinklio IS svarba [1]. Kuriant šių dienų programinę įrangą, dažniausiai yra stengiamasi laikytis esminių projektavimo principų: stipraus *vieningumo,* silpno *susietumo*. Kuriama programinė įranga turėtų būti lengvai papildoma naujomis funkcijomis, kurių pridėjimas neiššauktų jau esamo funkcionalumo modifikacijų (*open-close* principas) [7]. Tačiau žiniatinklio IS dažniausiai yra ta programinė įranga, kurios sudėtingumas, dėl keleto apjungiamų technologijų [3, 13, 17], yra aukštas, o pokyčių greitis – didelis [6]. Dėl žiniatinklio IS polinkio greitai keistis ne visada pavyksta išlaikyti tinkamą programinio kodo architektūrą ir funkcionavimo kokybę. Atliekant priežiūros ir palaikymo darbus vieningumas silpnėja, o susietumas stiprėja. Kai atskirų modulių susietumas stiprėja, taisant klaidas viename modulyje, atsiranda neišvengiami pokyčiai ir kituose moduliuose. Dažniausiai šie pokyčiai lemia naujų klaidų atsiradimą, ko pasekoje krenta žiniatinklio IS našumas ir vartotojų pasitikėjimas sistema [25]. Regresinio testavimo paskirtis yra užtikrinti, jog programinės įrangos pakeitimai, tokie kaip naujo funkcionalumo pridėjimas ar jau egzistuojančio taisymas, neigiamai nepaveikė to funkcionalumo, kuris ir neturėjo keistis. Kitaip tariant, ankstesnėje programos versijoje tinkamai veikęs funkcionalumas naujoje versijoje turi veikti taip pat gerai [10, 37].

Regresinio testavimo metodikoje buvo atlikta nemažai tyrimų, kurie sumažina testavimui reikalingas pastangas testavimo atvejų automatiniame sudaryme ir vykdyme [21, 28], testavimo atvejų kiekio sumažinime, išrinkime ir prioritetų skirstyme [10, 20, 23]. Tačiau egzistuoja mažai tyrimų, o kartu ir mažai metodų, kurie automatizuotai patvirtintų, jog testavimo atvejo aptikta klaida išties egzistuoja. Įvykdyti testai ir gauti rezultatai be jų vertinimo ar analizės regresiniame testavime yra mažai vertingi. Būtent *testų orakulas*, kuriame realizuotas regresiją aptinkantis algoritmas, ir sprendžia šią problemą, t.y. analizuoja testų rezultatus ir nurodo, ar naujoje programos versijoje egzistuoja klaida. Anksčiau testų orakulu būdavo žmogus, kurio tikslumą ir greitį buvo sunku padidinti, todėl šis varginantis ir turintis didžiausią polinkį į netikslumus testavimo atvejų pateiktų rezultatų lyginimas turėtų būti taip pat automatizuojamas [30].

Žiniatinklio IS pagrindinė sąsaja ir funkcionalumas atsispindi HTML atsake, kuris atvaizduojamas naršyklėje, todėl šį atsaką mato vartotojas [30]. Klaidos šiame atsake yra pastebimos dažniausiai ir sukelia daugiausiai sumaišties. Todėl HTML ãtsako regresinis testavimas yra būtinas. Idealiu atveju dviejų rezultatų, pavyzdžiui, dviejų skaičių ar eilučių,

palyginimas yra paprastas. Tačiau rezultatų palyginimas tampa sudėtingu tada, kai reikia palyginti du klasės objektus, o mūsų nagrinėjamu atveju – žiniatinklio IS HTML atsakus [6], kurių struktūra atitinka medžio struktūrą su tarp žymų esančiais duomenimis. Tiesioginis HTML atsako kaip teksto lyginimas, nors ir aptinka visa klaidas, tačiau pateikia per daug klaidingai teigiamų (angl. *false positives*) rezultatų. Reikalingas sumanesnis HTML atsako lyginimas, pavyzdžiui atsaką analizuojant kaip medžio struktūrą.

Šio darbo tikslai ir uždaviniai: realizuoti regresinio testavimo testų orakulo algoritmą, kuris HTML atsaką analizuoja kaip medžio struktūrą. Ištirti, patobulinti ir eksperimentiškai palyginti realizuotą testų orakulo algoritmą su literatūroje ir rinkoje esančių regresinio testavimo įrankių testų orakulų algoritmais. Apskaičiuoti testų orakulų algoritmų tikslumą, aptikimą ir efektyvumą.

Dokumento struktūra: magistrinį darbą sudaro 6 pagrindinės dalys. Pirmojoje dalyje pateikiami darbe naudojami terminai. Antrojoje dalyje atliekama regresinio testavimo metu kylančių problemų analizė, pateikiama žiniatinklio IS testavimo svarba ir išanalizuojami literatūroje pateikiami regresinio testavimo testų orakulų algoritmų sprendimai. Trečiojoje dalyje pateikiami realizuoto žiniatinklio IS testuojančio įrankio svarbiausi architektūriniai sprendimai. Ketvirtojoje dalyje tiriamos bendros visoms žiniatinklio informacinėms sistemoms būdingos semantinės HTML savybės. Pagal šias savybes palyginami testų orakulai. Penktojoje dalyje atliekamas eksperimentas su žiniatinklio IS ir dviem testų orakulais bei šių orakulų skirtingomis versijomis. Įvairiai analizuojant eksperimento metu surinktus duomenis pateikiami rezultatai. Šeštojoje dalyje pateikiamos viso darbo išvados.

1 TERMINŲ IR SANTRUMPŲ ŽODYNAS

1. **AJAX** (*Asynchronous JavaScript and XML*) – asinchroninis komunikavimas tarp serverio ir kliento, kurį įgyvendina JavaScript programinis kodas, sukurdamas *XmlHttpRequest* objektą. Dėl šio komunikavimo, galima atnaujinti tik reikiamą internetinio puslapio dalį, neperkraunant jo viso. Atskiru atveju gali būti ir sinchroninis komunikavimas.
2. **Analizatorius (*parser*)** – teksto eilutę pagal nustatytą gramatiką nagrinėjantis įrankis arba funkcija. Šiame darbe minimas HTML žymų kalbos analizatorius.
3. **Aptikimas (*recall*)** – šis dydis nurodo, kiek testų orakulas aptinka klaidų. Šiam dydžiui didėjant charakteristika gerėja.
4. **Atsakas (*HTML response*)** – žiniatinklio informacinės sistemos sugeneruotas HTML žymų, CSS stilių, JavaScript programinio kodo, paveikslėlių ir kitų duomenų naršyklei pateiktas atsakas į konkrečią užklausą. Tai dinamiška žiniatinklio IS vartotojo sąsaja.
5. **Baltos dėžės testavimas** – tai toks testavimo būdas, kai testavimo atvejai yra sudaromi remiantis testuojamos programinės įrangos vidine struktūra ir veikimo principais.
6. **Deterministinis** – tai toks sistemos veikimo principas, kai įėjimo duomenys daro konkrečią įtaką išėjimo duomenims, t.y. išėjimo duomenys kinta dėsningai. Žiniatinklio IS gautas atsakas laikomas deterministiniu, jeigu įvykdžius tą pačią užklausą keletą kartų gausime tą patį atsaką.
7. **GUI (*Graphical User Interface*)** – grafinė vartotojo sąsaja, kurios pagalba vartotojas gali pasiekti programinės įrangos teikiamas funkcijas. Žiniatinklio IS vartotojo sąsaja yra HTML atsakas.
8. **Informacinė sistema (IS)** – dažniausiai nutolusiame serveryje veikianti tam tikra programinė įranga, teikianti paslaugą jos vartotojams. Tačiau gali būti ir vietiniu mastu veikiančių informacinių sistemų. Jeigu informacinė sistema sąsają suteikia per interneto naršyklę, tokia sistema vadinama žiniatinklio informacine sistema.
9. **Juodos dėžės testavimas** – tai toks testavimo būdas, kai testavimo atvejai yra sudaromi nežinant testuojamos programinės įrangos vidinės struktūros ir veikimo principo.
10. **Klaidingai neigiamas** (*false negative*) – kai testų orakulas **klaidingai** praneša, jog klaidos **nėra**. Toks pranešimas blogina testų orakulo efektyvumą.
11. **Klaidingai teigiamas** (*false positive*) – kai testų orakulas **klaidingai** praneša, jog **yra** klaida. Toks pranešimas blogina testų orakulo efektyvumą.

12. **Mutuota programa (Mutantas)** – programinės įrangos versija su tikslingai įterpta klaida. Mutuota programinė įranga skirta išsiaiškinti, ar testavimo atvejis aptinka įterptą klaidą.
13. **Naujas atsakas** – vykdant tą patį testavimo atvejį iš naujesnės žiniatinklio IS versijos gautas HTML atsakas. Paprastai naujas atsakas yra lyginamas su senu.
14. **RegReload** – magistro studijų metu suprogramuotas žiniatinklio IS apkrovos ir regresinio testavimo įrankis.
15. **Sausainėliai (***cookies***)** – naršyklėje išsaugomi su konkrečiu žiniatinklio puslapiu susiję duomenys. Sausainėliuose naršyklė gali išsaugoti prisijungimo duomenis bei kitus su konkrečiu puslapiu susijusius nustatymus.
16. **Senas atsakas** – vykdant tą patį testavimo atvejį iš senesnės žiniatinklio IS versijos gautas ir išsaugotas HTML atsakas.
17. **Statinė analizė** – procesas, kai programinis kodas yra analizuojamas pagal sintaksę, tačiau jis nėra vykdant.
18. **Susietumas** (*coupling*) – dydis nusakantis objektiškai orientuoto programinio kodo modulių sąryšio laipsnį. Dydžiui mažėjant charakteristikos gerėja.
19. **Teisingai neigiamas** (*true negative*) – kai testų orakulas **teisingai** praneša, jog klaidos **nėra**.
20. **Teisingai teigiamas** (*true positive*) – kai testų orakulas **teisingai** praneša, jog **yra** klaida.
21. **Testas** (**testavimo atvejis**) – mūsų atveju žiniatinklio IS testavimo atvejis yra HTTP užklausa.
22. **Testavimo objektas** – mūsų atveju testavimo objektas yra žiniatinklio IS.
23. **Testų orakulas** – regresinį testavimą vykdančio įrankio modulis, kuriame realizuotas regresiją aptinkantis algoritmas. Šiam moduliui pateikiami testavimo metu gauti rezultatai, kuriuos jis turi palyginti ir pasakyti, ar lyginami duomenys sutampa. Mūsų atveju testų orakulas lygina HTML atsakus.
24. **Tikslumas** (*precision*) – šis dydis parodo, kiek buvo tikrų klaidų iš visų testų orakulo identifikuotų klaidų. Šiam dydžiui didėjant charakteristika gerėja.
25. **Užfiksuok/atkartok** *(capture/replay)* – tai testavimo atvejų sudarymas, kai vartotojo veiksmai yra įrašomi, o vėliau atkartojami kaip testavimo atvejai. Įrašymas gali būti fiksuojamas pelės koordinatėmis, klaviatūros mygtukų paspaudimais. Žiniatinklio IS testai gali būti sudarinėjami fiksuojant iš naršyklės išeinančias HTTP užklausas.
26. **Užklausa (***HTTP request***)** – į serverį siunčiami parametrai ir jų reikšmės, pagal kuriuos serveris atitinkamai vykdo pavestas užduotis ir formuoja atsaką.

27. **Vartotojo sesija** – informacinės sistemos serveryje naršymo metu saugomi vartotojo įvesti duomenys. Vartotojo sesijos duomenis galima transformuoti į HTTP užklausas.
28. **Vieningumas** (*cohesion*) – dydis nusakantis objektiškai orientuoto programinio kodo modulių ar funkcijų vieningą sąsają bei struktūrą. Dydžiui didėjant charakteristikos gerėja
29. **W3C (***World Wide Web Consortium***)** – organizacija plėtojanti ir standartizuojanti su žiniatinkliu susijusius protokolus.

2 ANALITINĖ DALIS

Regresinis testavimas apima automatinį testų sudarymą, jų kiekio mažinimą ir vykdymą, testų orakulo panaudojimą ir kitus susijusius aspektus. Šiame skyriuje analizuosime regresinio testavimo metu kylančias problemas ir jų sprendimo būdus. Taip pat aprašysime regresinio testavimo svarbą, bei kuo skiriasi regresinis žiniatinklio informacinių sistemų testavimas nuo darbastalio tipo programų testavimo.

2.1 Žiniatinklio informacinė sistema

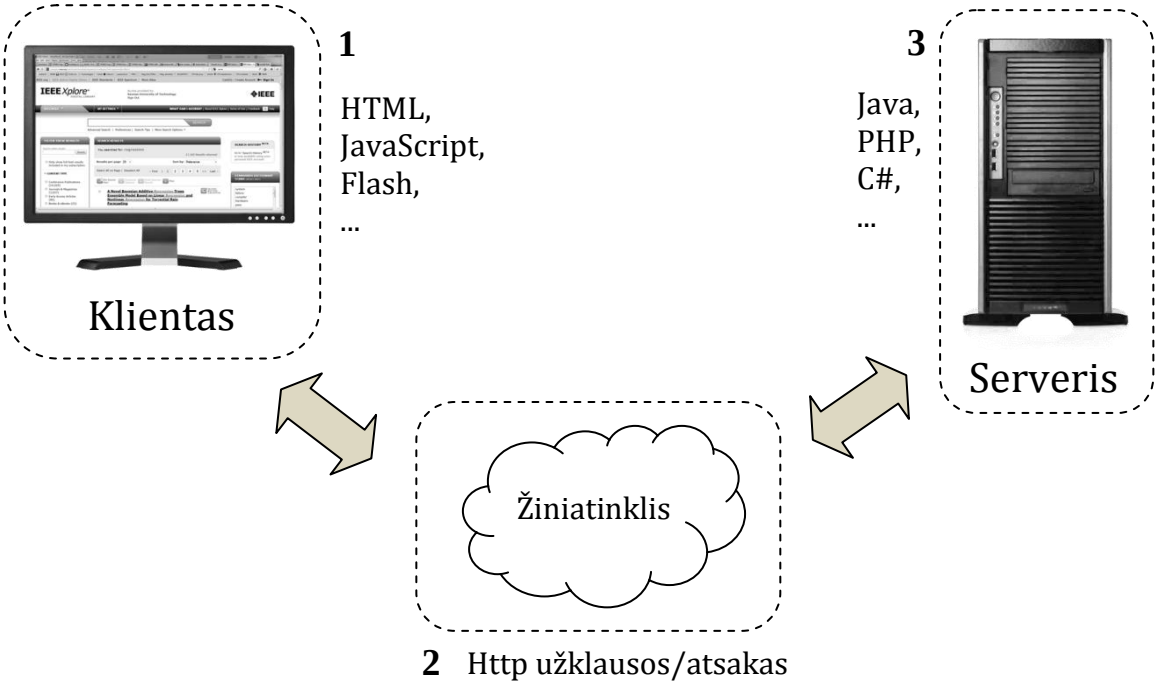
Šiame skyrelyje apibrėšime Žiniatinklio IS sąvoką, funkcionavimą ir jos aplinką. Žiniatinklio IS šiame magistriniame darbe yra svarbi tuo, jog ji yra *testavimo objektas*. **Apibrėžimas**:

> ž*iniatinklio informacinė sistema* (*IS*) yra tokia programinė įranga, kurios veikimas paremtas klientas-serveris komunikavimo principu, t.y. programinis kodas vykdomas nutolusiame serveryje, o šios programinės įrangos funkcijas vartotojas pasiekia naudodamasis klientu (interneto naršykle).

Žiniatinklio IS galime padalinti į tris sritis (1 pav.), kurios yra gana mažai priklausomos viena nuo kitos:

1) **Klientas** – interneto naršyklė, kurioje vartotojui atvaizduojama HTML žymomis paremta žiniatinklio IS sąsaja. Sąsajos interaktyvumą sukuria *JavaScript* ir *Flash* skriptai. Sąsaja yra greitai besikeičianti, interaktyvi ir nepatikima testų sudarymui (žiūrėti 2.4.2.1 Vartotojo sąsaja paremti testavimo atvejai).
2) **Žiniatinklis** – interneto tinklas, kuriame pranešimai tarp serverio ir kliento keliauja *HTTP užklausų* forma. Dažniausiai naudojami užklausų tipai yra GET ir POST.
3) **Serveris** – techninė ir programinė įranga, kuri vykdo žiniatinklio IS programinį kodą. Žiniatinklio IS gali būti sukurta panaudojant Java, PHP, C# ar kitas programavimo kalbas. Serveryje programinio kodo pagalba yra suformuojamas HTML atsakas, kuris ir perduodamas į klientą.

Šio tipo sistemos ir yra unikalios savo architektūra, nes vartotojo sąsaja yra atskirta nuo pačios programinės įrangos. Atskirtas dalis jungia nepriklausoma terpė – žiniatinklis.



1 Paveikslėlis. Klientas-serveris architektūra: trys izoliuotos sritys

Kasdieninis žiniatinklio IS naudojimas paprastų vartotojų, verslo ir vyriausybinių organizacijų tarpe, reikalauja vis aukštesnės tokių informacinių sistemų kokybės [24]. Didėja žiniatinklio IS svarba[1]. *Yoo* ir *Harman* [10] atliko regresinio testavimo metu kylančių problemų ir siūlomų sprendimų analizę, iš kurios galima teigti, jog atsiranda vis didesnis poreikis tobulinti žiniatinklio informacinių sistemų regresijos testavimo algoritmus.

2.2 Regresinio testavimo svarba

Kuriant šių dienų programinę įrangą, dažniausiai yra stengiamasi laikytis esminių projektavimo principų: stipraus *vieningumo*, silpno *susietumo*. Taip pat programinė įranga turėtų būti lengvai papildoma naujomis funkcijomis, kurių pridėjimas neiššauktų jau esamo funkcionalumo modifikacijų (*open-close* principas) [7]. Aktualūs turėtų būti ir šie parametrai: patikimumas ir apsauga[15]. Žiniatinklio IS dažniausiai yra ta programinė įranga, kurios sudėtingumas, dėl keleto apjungiamų technologijų [3, 13, 17], yra aukštas, o pokyčių greitis yra didelis [6]. Dėl žiniatinklio IS polinkio greitai keistis, ne visada pavyksta išlaikyti tinkamą programinio kodo architektūrą ir funkcionavimo kokybę. Vieningumas silpnėja, o susietumas stiprėja. Kai atskirų modulių susietumas stiprėja, taisant klaidas viename modulyje, atsiranda neišvengiami pokyčiai ir kituose moduliuose. Dažniausiai šie pokyčiai lemia naujų klaidų atsiradimą, ko pasekoje krenta žiniatinklio IS našumas ir vartotojų pasitikėjimas sistema [25]. *Regresinio testavimo* paskirtis yra užtikrinti, kad programinės įrangos pakeitimai, tokie kaip naujo funkcionalumo pridėjimas ar jau egzistuojančio taisymas, neigiamai nepaveikė to

funkcionalumo, kuris ir neturėjo keistis. Kitaip tariant, ankstesnėje programos versijoje tinkamai veikęs funkcionalumas, naujojoje versijoje turi veikti taip pat gerai [10, 37].

2.3 Regresinio testavimo metodas

Regresinio testavimo metodas (2 pav.) paprasčiausiai nusakomas panaudojant tokius apibrėžimus. Tarkime, jog turime dabartinę programos versiją *P*, kuri yra ištestuota ir veikianti. Atlikus pakeitimus turime *P'*, kuri yra naujesnė programos *P* versija. Esame paruošę testavimo atvejų rinkinį *T* skirtą testuoti minėtai programai *P*, kur *t* yra konkretus testavimo atvejis. *S* yra programos *P* specifikacija ir *S'* yra programos *P'* specifikacija. *P*(*t*) reiškia programos *P* vykdymą panaudojant testavimo atvejį *t* kaip įėjimo reikšmę [10, 22]. Programos pateikiamų išėjimų rinkinį galime pažymėti *O*. Mažoji *o* raidė reikštų konkretų rezultatą (a formulė).

$$P(t) = o$$

a) formulė programos rezultatai senoje programos versijoje

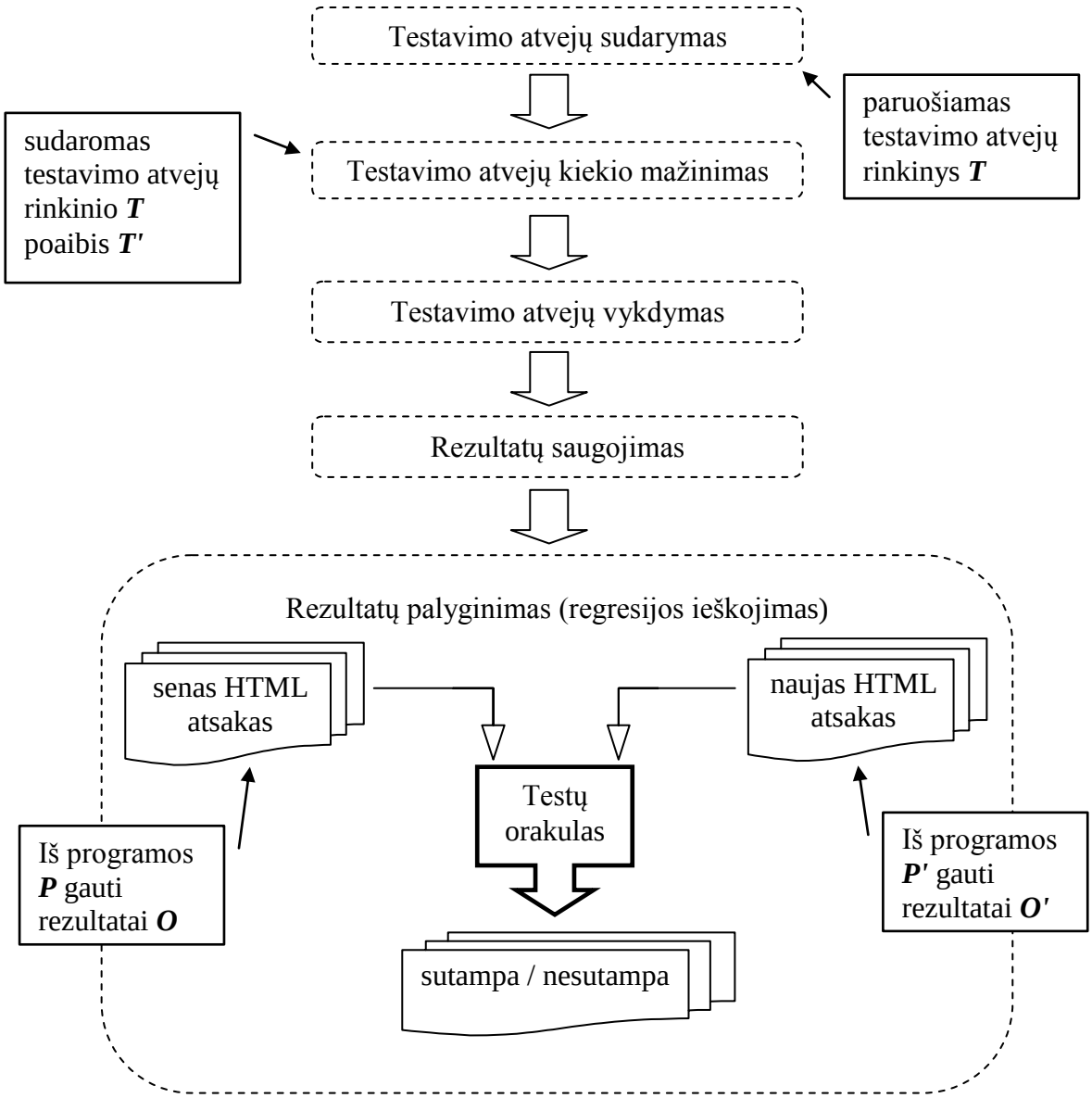
kur, $o \subset O$.

Regresinio testavimo *P*(*t*) metu gauti rezultatai yra išsaugomi. Atsiradus naujesnei programos versijai, testavimas su tais pačiais testais yra įvykdomas pakartotinai (b formulė).

$$P'(t) = o'$$

b) formulė programos rezultatai naujoje programos versijoje

Gauti nauji rezultatai *O'* tikrinami su išsaugotaisiais *O*. Ar rezultatai sutampa, t.y. ar skirtingose programos versijose tas pats funkcionalumas veikia vienodai, nusprendžia *testų orakulas* (2 pav.).



2 Paveikslėlis. Regresinio testavimo metodas: detali testų orakulo schema

2.4 Regresiniame testavime sprendžiamos problemos

Vykdant regresinį testavimą susiduriama su problemomis, kurios lėtina testavimą, daro jį neefektyviu ir labai brangiu. Gali pasirodyti, jog testavimą atlikti neįmanoma arba jo vykdymas neduos jokių apčiuopiamų rezultatų, bus itin brangus ir pareikalaus begalės laiko. Šiame skyriuje apžvelgsime regresinio testavimo metu iškylančias problemas ir sprendimus.

2.4.1 Testavimo atvejų sudarymas

Svarbus regresinio testavimo reikalavimas yra patvarūs, atsparūs testuojamos žiniatinklio IS tiek vidiniams, tiek išoriniams (*GUI*) pokyčiams, testavimo atvejai. Žiniatinklio IS veikia klientas-serveris komunikavimo principu. 1 paveikslėlyje (15 psl.)

pavaizduota šios architektūros schema, kurioje išskiriamos trys viena nuo kitos mažai priklausomos sritys:

- klientas,
- žiniatinklis ir
- serveris.

Priklausomai nuo regresinio testavimo tipo, testavimo atvejus galima sudaryti iš kiekvienos iš šių sričių. Kuriuo metu vykdomas testavimo atvejų sudarymas pavaizduota 2 paveikslėlyje (17 psl.).

2.4.1.1 Vartotojo sąsaja paremti testavimo atvejai

Vartotojo sąsaja paremti testavimo atvejai sudaromi kliento dalyje (1 pav., 1 sritis). Toks testų sudarymo būdas vadinamas *užfiksuok/atkartok* (*capture/replay*) [14]. Tokie testavimo atvejai yra nepatvarūs, ypač, jeigu jie sudaryti saugojant pelės paspaudimo koordinates, pelės ratuko pasukimus, ar klaviatūros klavišų paspaudimus, kaip tai daro *TestComplete* įrankis [26]. Tobulesni įrankiai, tokie kaip *SeleniumHQ* [5, 9], geba fiksuoti paspaudimus naršyklėje, t.y. fiksuojami HTML elementai, ant kurių paspaudė vartotojas. Tokie įrankiai kartu fiksuoja ir puslapio interaktyvumą, kurį sukuria JavaScript ir kaip teigiama literatūroje, yra daug artimesni vartotojo veiksmas. Tačiau tai aktualu vartotojo sąsajos testavimui, o mes tiriame informacinės sistemos veikimo pokyčius serveryje. Neigiama tokių įrankių savybė regresiniame testavime: kiekvieno testavimo atvejo metu, įrankis turi "matyti" vartotojo sąsają, kad galėtų paspausti ant įsiminto HTML elemento. O ką daryti, jeigu sąsaja pasikeitė ir nėra ant ko paspausti. Daugelis testų bus veltui sukurti, juos teks atnaujinti.

2.4.1.2 Programiniu kodu paremti testai

Programiniu kodu paremti testai yra sudaromi iš serveryje (1 pav., 3 sritis) saugomo žiniatinklio IS kodo. Tai vienetų testai, kurie remiasi vidine programos struktūra [10, 12, 32, 33]. Tačiau tokių testų bus itin daug, nes bus testuojami smulkiausi sistemos vienetai – funkcijos. Testus reikės dažnai koreguoti, šalinti ar kurti naujus. Testavimo atvejus galima sudaryti ir iš sistemos modelio, kuris sudaromas iš sistemos reikalavimų [31]. Tačiau reikia daug laiko, kad būtų galima sudaryti tikslų sistemos modelį [21]. Reikalavimų specifikacija gali būti nepilna arba neatitikti dabartinės sistemos versijos. Ir svarbiausias faktas yra tas, jog nėra universalaus modelio, kuris atitiktų didžią dalį informacinių sistemų [1].

2.4.1.3 Žiniatinklio užklausomis paremti testai

Populiariausias žiniatinklio IS testavimo atvejų sudarymo būdas regresiniam testavimui yra žiniatinklio užklausų pagrindu paremti testavimo atvejai (1 pav. 2 sritis) [1, 6, 13, 14]. Toks testų sudarymo būdas taip pat priskiriamas prie užfiksuok/atkartok sudarymo būdo, tik fiksuojami ne pelės paspaudimai, bet po paspaudimo naršyklėje sugeneruotos HTTP užklausos. Tokiu būdu sudaryti testavimo atvejai kiek įmanoma atsiriboja:

a) nuo kintančios vartotojo sąsajos, kuri žiniatinklio IS yra HTML žymos (1 pav. 1 sritis)
b) nuo serverio pusėje kintančio informacinės sistemos programinio kodo (1 pav. 3 sritis).

Galima išskirti du būdus, kurie generuoja žiniatinklio pagrindu paremtus testavimo atvejus (užklausas):

I. vartotojo sesijoje kaupiamų duomenų transformavimas į HTTP užklausas [2, 21, 23, 28].
II. kliento pusėje išeinančių HTTP užklausų fiksavimas [1, 13, 16].

Pirmas būdas yra tinkamas tuo, jog gaunamos natūralios vartotojų veiksmų sekos, kurios tiksliausiai atspindi vartotojų veiksmus žiniatinklio IS. Šios technikos efektyvumas didėja kartu su didėjančia vartotojo sesija. Tačiau neigiama pusė ta, kad gaunama daug perteklinių duomenų – reikia spręsti testavimo atvejų kiekio sumažinimo problemą. Šis būdas remiasi baltos dėžės testavimo principu, kai testuojamo objekto struktūra yra žinoma. Sprenkle su kolegėmis [28] siūlo karkasą, kuris automatizuoja veiksmus nuo sesijos surinkimo, testų generavimo, iki jų vykdymo ir rezultatų tikrinimo panaudojant testų orakulą. Tačiau karkasas gali būti integruotas tik į Java/JSP pagrindu veikiančius serverius. Norint testuoti, pavyzdžiui PHP kalba parašytus tinklapius, reikia papildomų plėtinių šiam karkasui.

Antro būdo privalumas yra tas, jog užklausos surenkamos nelendant į žiniatinklio IS ir jos serverį. Užklausos gaunamos iškart po vartotojo veiksmų. Pertekliniai testai atsiranda gana retai. Neigiama savybė būtų ta, jog tokie testavimo atvejai gali neatspindėti tikro vartotojų elgesio, jeigu testavimo atvejus sudaro testuotojas arba programuotojas. Tačiau iš kitos pusės, tai ir pačios žiniatinklio IS logikos problema, jeigu vartotojui yra leidžiama pasiekti tą turinį, kurio jis neturėtų pasiekti arba pasiekti keliais būdais, nors testuotojas ar programuotojas tikisi, jog vartotojas naudosis tik vienu būdu.

2.4.2 Testavimo atvejų kiekio mažinimas

Regresinio testavimo metu, kuris yra ilgalaikis procesas ir tęsiasi visą programinės įrangos gyvavimo ir palaikymo laikotarpį, sukaupiama didžiulė testavimo atvejų aibė.

Didžiulis testavimo atvejų kiekis gali susidaryti ir dėl to, jog testai generuojami iš perteklinių duomenų. Testavimo metu įvykdyti programinę įrangą su visais testavimo atvejais yra neefektyvu, nes gali užtrukti begalę laiko [4]. Šiame skyrelyje aprašomos įvairios su testų kiekio mažinimu susijusios problemos. Kuriuo metu testavimo atvejų mažinimas vykdomas regresiniame testavime pavaizduota 2 paveikslėlyje (14 psl).

2.4.2.1 Testavimo atvejų ilgalaikio sumažinimo problema

Kad programinė įranga būtų ištestuota, reikia tokio testavimo atvejų rinkinio, kuris padengtų visus kelius. Tačiau testavimo atvejų surinkimo metu gali atsirasti testų dubliavimasis. Todėl perteklinius testus reikia pašalinti. Pagal Rothermel [20], ši problema skambėtų taip:

> *turime testavimo atvejų rinkinį T ir testavimo reikalavimų rinkinį {r_1,...,r_n}. Šie reikalavimai turi būti tenkinami, kad programa būtų pakankamai ištestuota. Todėl reikia surasti tokį testų rinkinį T', kurio kiekvienas testavimo atvejis t_j tenkintų keliamą reikalavimą r_i.*

Šis testavimo atvejų kiekio sumažinimas iki minimumo yra ilgalaikis [10]. Autoriai siūlo skaičiuoti sumažinamų testų naudą ir išlaidas, kurios patiriamos dėl praleistų klaidų [20]. Taip galima surasti tuos testavimo atvejus, kurie teikia didžiausią naudą.

2.4.2.2 Testavimo atvejų išrinkimo problema

Nors testavimo atvejų išrinkimo technikos taip pat siekia sumažinti testų kiekį, daugelis išrinkimo technikų yra priklausomos nuo programinėje įrangoje atliktų pakeitimų, t.y. išrinktų testų kiekis yra pritaikytas konkrečiai programos versijai atsižvelgiant į modifikuotas programos dalis [10]. Kadangi reikia žinoti, kurios programos vietos buvo pakeistos, šis regresinis testavimas yra vykdomas kaip *baltos dėžės testavimas* pasitelkiant *statinę analizę*. Visais atžvilgiais, šios problemos sprendimas yra priklausomas nuo sugebėjimo įžvelgti pakeistas programos vietas. Pagal Rothermel ir Harold [18], šios problemos apibrėžimas skambėtų taip:

> *turime programą P. Naujesnė programos P versija yra programa P'. Taip pat turime testavimo atvejų rinkinį T. Reikia surasti T rinkinio poaibį T', su kuriuo būtų galima testuoti konkrečią programos P' versiją.*

Toks testavimo atvejų sumažinimas yra reikalingas todėl, kad kiekvieną kartą atlikus net ir nedidelį pakeitimą, užtektų panaudoti tik nedidelę, specialiai pakeitimui pritaikytą, testavimo atvejų aibę, t.y. identifikuojami konkrečiai programos versijai tinkantys testai [2, 4, 8, 19]. Didžiulės programos gali turėti šimtus ar tūkstančius testavimo atvejų. Vykdant visus testus

būtų gaištamas laikas. Testavimo atvejų išrinkimui taip pat gali būti panaudotas modeliu paremtas metodas [11].

2.4.2.3 Prioritetų priskyrimo testavimo atvejams problema

Prioritetų skirstymas testavimo atvejams suteikia galimybę suskirstyti turimą testų rinkinį tokia tvarka, kad pirmiausiai būtų vykdomi reikalingas savybes turintys testai, pavyzdžiui turintys didžiausią klaidų aptikimo rodiklį arba padengiantys unikalią kodo dalį [23]. Atlikus testavimo atvejų perstatymus pagal svarbą, visi testavimo atvejai yra vykdomi, tačiau testuotojas gali pasirinkti, kada jo nuomone yra pasiektas pakankamas testavimo laipsnis atsižvelgdamas į tai, jog svarbiausi testai buvo įvykdyti testavimo pradžioje [10]. Formalus šios problemos aprašymas atrodytų taip:

> *turime testų rinkinį T, šio testų rinkinio kombinacijų rinkinį PT ir funkciją, kuri pateikia realų kombinacijos PT skaičių.* $f: PT \rightarrow \mathbb{R}$. *Reikia surasti tokį T‘ rinkinį, kur* $T' \in PT$, *taip, kad visi T‘‘, kur* $T'' \in PT$, $T'' \neq T'$, $[f(T') \geq f(T'')]$.

2.4.3 Testų orakulo problema

Įvairios regresinio testavimo technikos stengiasi sumažinti testavimui reikalingas pastangas ir kainą. Testavimo atvejų sumažinimo, išrinkimo ir prioritetų skirstymo technikos stengiasi sutrumpinti testų vykdymo laiką. Kitos technikos stengiasi automatiškai sudaryti testus, juos vykdyti ir išsaugoti gautus rezultatus. Testų orakulo vykdymo kaina glaudžiai siejasi su testavimo kokybe [10]. Anksčiau testų orakulu būdavo žmogus, kurio tikslumą ir greitį buvo sunku padidinti, tuo tarpu testavimo atvejų vykdymo automatizavimą galima nesunkiai įgyvendinti. Su dabartine technine įranga net ir didžiulę testų aibę įmanoma įvykdyti ir tai nepareikalaus žmogaus įsikišimo. Tačiau įvykdyti testai ir gauti rezultatai be jų vertinimo ar analizės regresiniame testavime bus mažai vertingi. Būtent *testų orakulas* ir sprendžia šią problemą, t.y. analizuoja testų rezultatus ir nurodo, ar naujoje programos versijoje egzistuoja klaida. Kuriuo metu testų orakulas atlieka palyginimą pavaizduota 2 paveikslėlyje (14 psl).

Informacinės sistemos funkcijos ir moduliai pateikia vienokio ar kitokio tipo išėjimo duomenis. Jeigu funkcija ar modulis siejasi su vartotojo sąsaja, tuomet dažnai išėjimo duomenys bus HTML žymos. Prie išėjimo duomenų paprastai dar priskiriami generuojami ir išsiunčiami el. laiškai, išvedama vidinė serverio būsena ar bendravimas su servisais. Šiuos papildomus išeinančius duomenis stebėti yra sunkiau. Vartotojui akivaizdžiai jie nėra matomi. Tačiau HTML atsakas yra matomas [30]. Klaidos jame yra pastebimos dažniausiai ir sukelia daugiausiai sumaišties. Todėl HTML atsako regresinis testavimas yra būtinas.

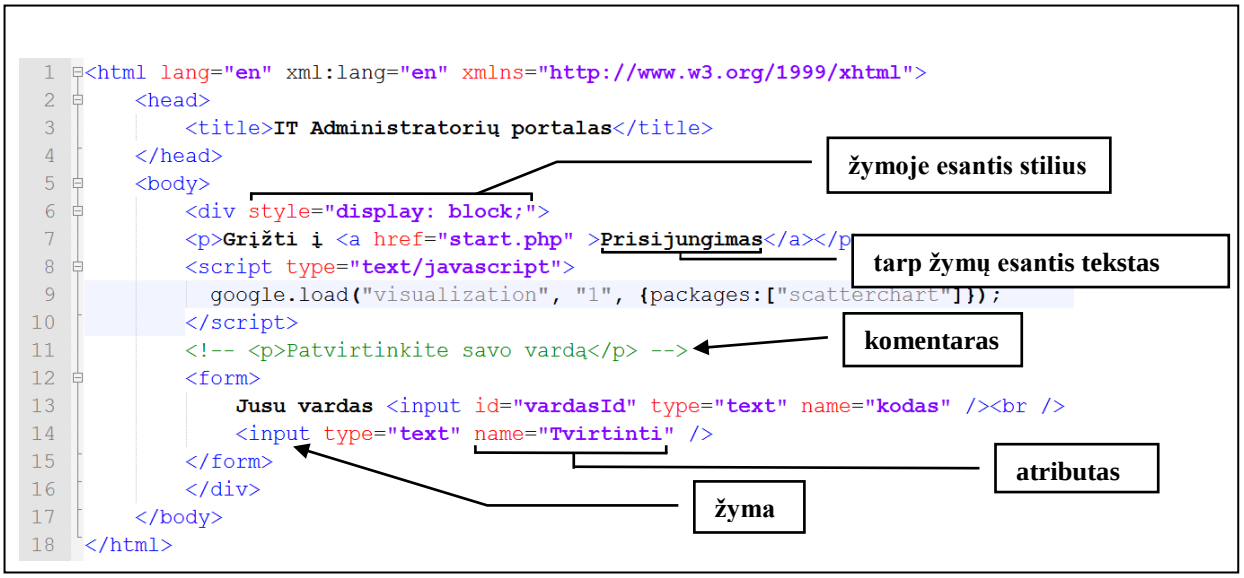
Idealiu atveju dviejų rezultatų palyginimas, pavyzdžiui dviejų skaičių ar eilučių, yra paprastas. Tačiau rezultatų palyginimas tampa sudėtingu tada, kai reikia palyginti du klasės objektus, o mūsų nagrinėjamu atveju, žiniatinklio IS HTML atsakus [6]. Tradicinė regresinio testavimo technika daro prielaidą, jog programa grąžina *deterministinį* rezultatą. Todėl regresinį testavimą sunku pritaikyti programoms, kurių veikimas paremtas lygiagretumu ar nuolat kintančiu turiniu [10]. Tačiau HTML atsako kaip medžio struktūros analizavimas lemia tai, jog į gautų įrašų išsidėstymo tvarką galime nekreipti dėmesio, pakanka, kad įrašai egzistuotų. Regresijos ieškojimas HTML dokumente gali būti paprastas – analizuojama tik siaura sritis ir sudėtingas, kai stengiamasi surasti skirtumus tiek struktūroje, tiek duomenyse. Kai du HTML dokumentai yra beveik vienodi, surasti skirtumus yra sąlyginai paprasta. Tačiau, ką daryti, kai du HTML dokumentai skiriasi iš esmės. Galbūt jų struktūra sutampa tik tam tikrose vietose. Tokiu atveju, vienas iš sprendimų būtų pasinaudoti `DiffX` algoritmu, kuris tarp dviejų dokumentų paskaičiuoja sulygiavimą analizuodamas elementų poras [6]. Tokiu būdu palengvinamas testų orakulo darbas, o svarbiausia – padidinamas *tikslumas*. Iš `DiffX` algoritmo atliktų operacijų skaičiaus ir tipo galima spręsti, ar atlikti pokyčiai tarp dviejų dokumentų yra kritiniai ir lemia ryškius skirtumus.

Testų orakulas, kuris paprasčiausiai aptinka bet kokį pasikeitimą puslapyje (HTML atsakas lyginamas kaip tekstas), gali pranešti apie *klaidingai teigiamą* rezultatą, t.y. testuotojas bus informuojamas, kad egzistuoja klaida, nors jos iš tikrųjų nėra. Tikros klaidos pasislėps tarp aptiktos perteklinės regresijos. Taip gali atsitikti net ir dėl duomenų pasikeitimo: atsinaujino versija, data, laikas. Kad testų orakulas būtų laikomas efektyviu ir automatiniu, turėtų ieškoti semantinių pokyčių HTML atsake. Tačiau jeigu testų orakulas tikrina tik specifines vietas, testuotojas gali būti informuotas apie *klaidingai neigiamą* rezultatą, kai klaida programoje egzistuos, tačiau testų orakulas ją praleis [6, 28, 30]. Dėl tokio testų orakulo veikimo bus praleidžiamos tikros klaidos, žiniatinklio IS kokybė ir našumas kris, vartotojai liks nepatenkinti.

Vieni autoriai bando surasti ir pritaikyti konkretų testų orakulą ar orakulų kombinaciją konkrečiai žiniatinklio IS [29], kiti bando susisteminti ir ištobulinti vieną testų orakulą, kuris tiktų visoms žiniatinklio IS, atsižvelgiant į tai, jog HTML atsakui būdingos klaidos ar netikslumai galioja visoms žiniatinklio IS [6].

Testų orakulo kokybę lemia ir HTML analizatoriaus (*parser*) pasirinkimas. Orakulai įgyvendinti panaudojant skirtingus analizatorius ar programavimo kalbas gali pateikti šiek tiek skirtingus rezultatus. Be to, skirtingos naršyklės analizuoja ir atvaizduoja HTML skirtingai, dėl ko orakulas gali pranešti apie klaidą, tačiau kitose naršyklėse tame pačiame puslapyje funkcionalumas gali būti atvaizduotas pilnai – klaidos nebus[30].

Pagal HTML specifikacijas [34, 35], HTML atsake gali būti saugomi ir duomenys, ir puslapio apipavidalinimas (3 pav.). Taip pat egzistuoja komentarai ir skriptų žymos. Duomenys gali būti ne tik tarp žymų, bet ir atributuose.



3 Paveikslėlis. HTML atsako pavyzdys

2.4.3.1 Testų orakulų metrikos

Testų orakulai dažniausiai yra lyginami pagal du dydžius (c ir d formulės): *tikslumas* ir *aptikimas (recall)*[30].

$$tikslumas = \frac{teisingai\ identifikuotos\ klaidos}{visos\ identifikuotos\ klaidos} \cdot 100\%$$

c) formulė testų orakulo tikslumas

$$aptikimas = \frac{teisingai\ identifikuotos\ klaidos}{tikėtinų\ klaidų\ skaičius} \cdot 100\%$$

d) formulė testų orakulo klaidų aptikimas

Kuo didesnis *tikslumas*, tuo mažiau testų orakulas praneša *klaidingai teigiamų* (*false positives*) rezultatų (c formulė). Kuo didesnis *aptikimas*, tuo mažiau gauname *klaidingai neigiamų* (*false negatives*) rezultatų (d formulė).

Tarkime, jog turime du testų orakulus ir norime įvertinti jų charakteristikas. Testavimo metu didesnis aptikimas buvo pirmojo orakulo. O tikslumas didesnis buvo antrojo orakulo. Didesnis pirmojo orakulo aptikimas rodo, jog testų orakulas HTML dokumente aptiko daugiau pokyčių ir nustatė daugiau klaidų, negu antrasis, ir praleido mažiau klaidingai neigiamų rezultatų. Tačiau to paties testų orakulo tikslumas buvo žemesnis. Tai reiškia, jog pirmasis testų orakulas buvo mažiau tikslus ir aptiko daugiau perteklinės regresijos lyginant su antruoju. Orakulo savybės gerėja didėjant abiems dydžiams.

Šias dvi metrikas apibendrinantis dydis yra *efektyvumas* (e formulė). Kuo efektyvumas didesnis, tuo testų orakulas yra našesnis. Efektyvumas skaičiuojamas pagal harmoninio vidurkio formulę, kurios nariais mūsų atveju tampa tikslumo ir aptikimo dydžiai:

$$efektyvumas = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}} \cdot 100\%$$

e) formulė efektyvumas pagal harmoninį vidurkį

2.4.3.2 Žiniatinklio IS orakulų tipai

Sprenkle su kolegėmis [30] siūlo 22 įvairaus tipo HTML testų orakulus. Testų orakulai skirstomi į grupes pagal tai, kokias HTML atsako vietas analizuoja. Kadangi HTML atsakas savyje turi ir struktūrą ir duomenis (3 pav. 23 psl.), orakulai suskirstyti į tris pagrindines grupes, kur pirmoji grupė apjungia struktūros ir turinio grupes:

I. **Dokumentu paremti** – tai tokie testų orakulai, kurie lygina HTML atsaką lyg tekstą, aptikdami visus pasikeitimus. Tokie orakulai įgyvendinti panaudojant `diff` įrankį.
II. **Turiniu paremti** – tai tokie testų orakulai, kurie specializuoti tikrinti HTML atsake esančius duomenis ir ignoruoja HTML žymas.
III. **Struktūra paremti**– tai tokie testų orakulai, kurie specializuoti tikrinti HTML atsake esančią žymų struktūrą ir ignoruoja tarp žymų esančius duomenis.

Visi 22 testų orakulai turi savo vietą hierarchinėje struktūroje, kurios viršūnėje yra pats paprasčiausias testų orakulas lyginantis du HTML dokumentus kaip tekstus panaudodamas `diff` įrankį. Hierarchijos viršuje esantys testų orakulai turės didesnį aptikimą, t.y. bus pranešama mažiau klaidingai neigiamų rezultatų, ir mažesnį tikslumą, kas reikš, jog bus pranešama daug klaidingai teigiamų rezultatų. Leidžiantis hierarchija žemiau, orakulų tikslumas didėja, nes tikrinama sritis detalizuojama. 1 lentelėje aprašomi minėtieji 22 testų orakulai.

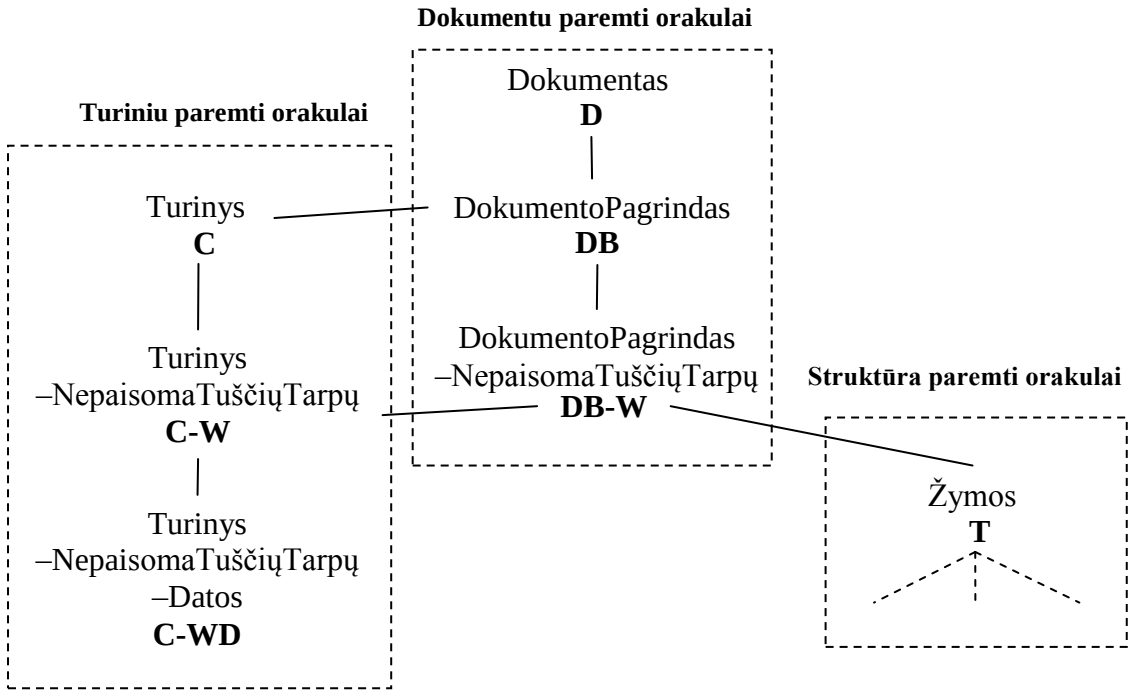
1 Lentelė. HTML testų orakulų detalus aprašas [30]

	Pavadinimas	Žymėjimas[1]	Informacija apie orakulą
Dokumentu paremti orakulai			
1	Dokumentas	**D**	Aptinka bet kokius pasikeitimus tarp dviejų HTML atsako dokumentų, bet praneša ir apie nesamas klaidas, kurios atsiranda dėl nenuspėjamo HTML atsako ar teksto pokyčių [29].
2	DokumentoPagrindas	**DB**	Veikia taip pat kaip **D** orakulas, tik papildomai nekreipiama dėmesio į HTML

[1] Testų orakulų žymėjimas sudarytas iš pirmųjų angliškų žodžių raidžių, pavyzdžiui: **D**ocument**B**ased–**W**hitespace atitinka **DB**–**W** arba Tag**N**ame+**I**mportantAttributes–**S**tyle**L**ayout atitinka **N+I–SL**.

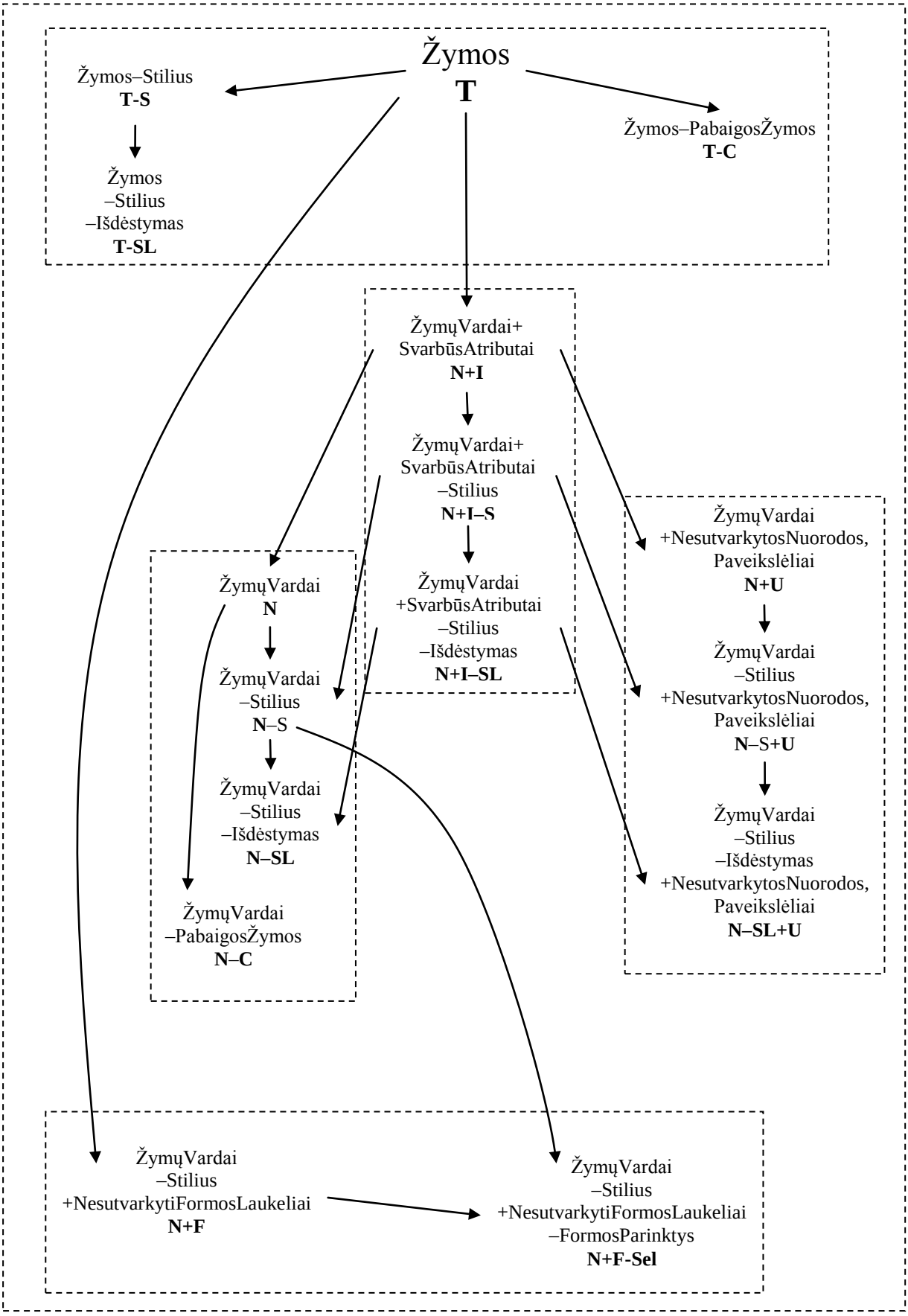
Pavadinimas		Žymėjimas[1]	Informacija apie orakulą
			komentarus, meta-žymas, skriptus ir pasenusias žymas, pvz. `noframe`.
3	DokumentoPagrindas –NepaisomaTuščiųTarpų	**DB-W**	Kaip ir **DB** orakulas, tik papildomai nekreipia į tuščius tarpus.
Turiniu paremti orakulai			
4	Turinys	**C**	Tai orakulas, kuris lygina duomenis ir nekreipia dėmesio į HTML žymų struktūrą.
5	Turinys –NepaisomaTuščiųTarpų	**C-W**	Kaip ir **C** orakulas, tik papildomai nekreipia dėmesio į tuščius tarpus.
6	Turinys –NepaisomaTuščiųTarpų –Datos	**C-WD**	Kaip ir **C-W**, tik papildomai nekreipiama dėmesio į besikeičiančią datą ar laiką.
Struktūra paremti orakulai			
7	Žymos	**T**	Orakulas tikrina žymų struktūrą – išsidėstymą HTML atsake.
8	Žymos–PabaigosŽymos	**T-C**	Kaip ir orakulas **T**, tik papildomai nekreipiama dėmesio į žymos pabaigą.
9	Žymos–Stilius	**T-S**	Kaip ir orakulas **T**, tik papildomai ignoruojamas stilius žymose `style="…"`.
10	Žymos–Stilius –Išdėstymas	**T-SL**	Kaip ir **T-S** orakulas, tik papildomai ignoruojamos išdėstymo žymos.
11	ŽymųVardai	**N**	Kaip ir orakulas **T**, tik papildomai tikrina, ar HTML atsake esanti struktūra sudaryta iš tų pačių žymų.
12	ŽymųVardai–Stilius	**N-S**	Kaip ir **N** orakulas, tik papildomai nekreipiama į stilių žymose.
13	ŽymųVardai–Stilius –Išdėstymas	**N-SL**	Kaip ir **N-S** orakulas, tik papildomai nekreipiama dėmesio į išsidėstymo žymas, pvz: vietoj buvusios `div` žymos naujoje programos versijoje atsirado `span` žyma.
14	ŽymųVardai –PabaigosŽymos	**N-C**	Kaip ir **N** orakulas, tik papildomai nekreipiama dėmesio į žymos pabaigą.
15	ŽymųVardai +SvarbūsAtributai	**N+I**	Kaip ir **N** orakulas, tik papildomai tikrinami svarbūs žymos atributai. Svarbūs atributai buvo išsiaiškinti remiantis HTML specifikacija [35].
16	ŽymųVardai +SvarbūsAtributai –Stilius	**N+I-S**	Taip pat kaip ir **N+I** orakulas, tik papildomai nekreipiama dėmesio į stilių žymose.
17	ŽymųVardai +SvarbūsAtributai –Stilius –Išdėstymas	**N+I–SL**	Kaip ir **N+I-S** orakulas, tik papildomai nekreipiama dėmesio į išdėstymo žymas (`div`, `span`, `table`).

	Pavadinimas	Žymėjimas[1]	Informacija apie orakulą
18	ŽymųVardai +NesutvarkytosNuorodos, Paveikslėliai	**N+U**	Kaip ir **N** orakulas, tik papildomai nekreipiama dėmesio į nuorodų ir paveikslėlių žymų rikiavimą. Taip pat tikrinami šioms žymoms būdingi atributai.
19	ŽymųVardai –Stilius +NesutvarkytosNuorodos, Paveikslėliai	**N–S+U**	Kaip ir **N+U** orakulas, tik papildomai nekreipiama dėmesio į stilių žymose.
20	ŽymųVardai –Stilius –Išdėstymas +NesutvarkytosNuorodos, Paveikslėliai	**N–SL+U**	Kaip ir **N–S+U** orakulas, tik papildomai nekreipiama dėmesio į išdėstymo žymas (`div`, `span`, `table`).
21	ŽymųVardai –Stilius +Nesutvarkyti FormosLaukeliai	**N+F**	Kaip ir **N** orakulas, tik papildomai nekreipiama į formose esančių įvedimo laukų išsidėstymo tvarką.
22	ŽymųVardai –Stilius +Nesutvarkyti FormosLaukeliai –FormosParinktys	**N+F-Sel**	Kaip ir **N+F** orakulas, tik papildomai netikrinamos numatytosios formos parinktys.



4 Paveikslėlis. Testų orakulų hierarchija pagal Sprenkle

HTML testų orakulų hierarchinis skirstymas į tris pagrindines grupes (4 pav.): **dokumentu paremti** orakulai, **turiniu paremti** orakulai, **struktūra paremti** orakulai. Dėl gausos, struktūra paremti orakulai perkelti į 5 paveikslėlį.



5 Paveikslėlis. Testų orakulų hierarchija pagal Sprenkle: struktūra paremti orakulai

Struktūra paremtų orakulų hierarchijoje aukščiausią vietą užima žymas tikrinantis orakulas. Hierarchijoje žemiau esantys orakulai specializuojasi konkrečioje srityje paveldėdami savo pirmtakų savybes (5 pav.).

2.4.3.3 Orakulų rezultatai

Visi aprašyti testų orakulai buvo ištestuoti su keturiomis žiniatinklio IS sistemomis. Dvi iš keturių sistemų vartotojui pateikia nuspėjamą (deterministinį) rezultatą, t.y. kai HTML atsakas kinta tik keičiantis testavimo atvejui. Likusios dvi sistemos vartotojui pateikia kintantį rezultatą, t.y. kai HTML atsakas kinta net ir su tuo pačiu testavimo atveju. Dėl kaskart kintančio HTML atsako, iš šių sistemų gaunamiems rezultatams buvo teikiama pirmenybė, nes su tokiu HTML atsaku testų orakului susidoroti yra sunkiau [30].

Kaip ir tikėtasi, **Dokumentu paremti** orakulai turi aukščiausią aptikimą, nes identifikuoja klaidas ir dokumento struktūroje, ir turinyje, tačiau kartu yra ir mažiausiai tikslūs. Šie orakulai praneša daugiausiai klaidingai teigiamų rezultatų. Didžiausią tikslumą turi **N**, **N+U** ir **N+F-Sel**. Bendrą didžiausią vidutinį efektyvumą turi **N+F-Sel** orakulas (pagal Sprenkle vid. efektyvumas = 0,86). Taip pat didelį efektyvumą turi **N+I** ir **N+I–S** orakulai (efektyvumas = 0,8). Formas tikrinantis **N+F** orakulas praneša nemažai klaidingai teigiamų rezultatų, tuo tarpu **N+F-Sel** orakulas su tam tikromis žiniatinklio IS tokių klaidų visai nepraneša, nes nekreipiama dėmesio į numatytąsias formų parinktis. **Turiniu paremti** orakulai tam tikrais atvejais veikia taip pat prastai, kaip ir **dokumentu paremti** orakulai. Tokius rezultatus lemia atsitiktiniai duomenys žiniatinklio IS puslapiuose.

Pagal atliktus tyrimus Sprenkle su kolegėmis teigia, jog **N+I** orakulas ir **turiniu paremti** orakulai, nors ir nagrinėja skirtingas HTML atsako vietas, yra labai panašūs (99% panašumas).

2.4.3.4 Orakulų kombinacijos ir jų efektyvumas

Orakulai skirstomi pagal analizuojamą HTML atsako sritį: struktūra (žymos) ir turinys (duomenys). Sprenkle su kolegėmis [30] atsižvelgdamos į gautus tyrimų rezultatus pabandė sujungti keletą orakulų į vieną ir patikrinti jų efektyvumą (tikslumą ir aptikimą). Orakulai sujungiami atsižvelgiant į šiuos faktorius:

- Apjungiami papildantys vienas kitą orakulai: pavyzdžiui **N+I** orakulas, kuris nagrinėja struktūrą ir **C-WD**, kuris nagrinėja tarp struktūros esantį turinį nežiūrėdamas į tarpus, datą ir laiką.
- Priešingus rezultatus pranešantys orakulai.
- Panaudota tyrimuose įgauta intuicija.

Sujungus du orakulus yra padidinamas aptikimas (sumažėja klaidingai neigiamų rezultatų). Klaidingai teigiamų rezultatų kiekis tokiu atveju gali mažėti arba didėti. Nagrinėjant orakulų pateikiamų rezultatų sankirtą yra sumažinama klaidingai teigiamų rezultatų tikimybė ir padidinamas tikslumas, tačiau sumažinamas aptikimas, nes abu orakulai turi patvirtinti konkrečią klaidą. Geriausią efektyvumą, tiek su nuspėjamą, tiek su nenuspėjamą HTML atsaką grąžinančioms žiniatinklio IS, pateikia **N+I** ∪ **C-WD** orakulų sąjunga (efektyvumas = 0,91). Nuspėjamą HTML atsaką grąžinančioms žiniatinklio IS testuoti geriausiai tinka **dokumentu paremtos** grupės orakulai. Tačiau antra pagal tinkamumą yra jau minėtoji **C-WD∪N+I** orakulų sąjunga. Tyrimų rezultatai [29] rodo, kad pats efektyviausias testų orakulas yra toks, kuris tikrina ir HTML struktūrą, ir tarp žymų esantį turinį. Kiekvienai žiniatinklio IS bendrais panašumais paremtas testų orakulas [6], kurio pagrindinė analizuojama sritis yra HTML arba XML struktūra, o turinys analizuojamas tik iš dalies (pvz. ieškoma klaidą atspindinčių žodžių), yra nuo 2,5 iki 50 kartų efektyvesnis už `diff` tipo orakulą ir nenusileidžiantis jam aptikimo dydžiu.

2.4.3.5 Tinkamiausio testų orakulo ar orakulo kombinacijos išrinkimas

Ne visada orakulų kombinacijos duoda teigiamų rezultatų. Kartais tik konkrečią HTML atsako sritį tikrinantis orakulas yra efektyvesnis už keleto orakulų kombinaciją, kurie kartu tiria platesnę duomenų aibę. Vėlesniuose tyrimuose Sprenkle su kolegėmis [29] siūlo sprendimų medžiu paremtą geriausio testų orakulo išrinkimą tiriamoms konkrečioms žiniatinklio IS. Orakulų išrinkimo metodika veikia tokiu principu:

1. suformuojamas požymių rinkinys, turintis orakulų pateikiamus rezultatus ir tiriamos žiniatinklio IS tikėtiną veikimą.
2. duomenys pateikiami sprendimų medžiui, kuris atlieka analizę
    a. testavimo metu orakulas testavimo atvejį priėmė ar nepriėmė (*pass / no pass*).
    b. ar tokio paties rezultato ir buvo tikėtasi: testas turėjo būti priimtas arba ne priimtas.

2.4.3.6 Nenuspėjamo HTML atsako įtaka

Jeigu žiniatinklio IS HTML atsakas to paties testo (užklausos) vykdymo metu nekistų, regresiniam testavimui pakaktų **dD** arba **DB-W** testų orakulo [29]. Tačiau šių dienų žiniatinklio IS dažniausiai yra interaktyvios, su nuolatos besikeičiančiu turiniu [6, 17, 27]. Toks informacinės sistemos veikimas lemia nuolat kintantį HTML atsaką, net jeigu vartotojas spaudžia vis ant tos pačios nuorodos, pavyzdžiui šone vartotojui parodomi neseniai skaityti straipsniai arba su skaitomu straipsniu susiję straipsniai. Testų orakulams aptikti tikras klaidas

tokiame HTML atsake yra didžiulis iššūkis. Buvo ištestuotos nenuspėjamą HTML atsaką grąžinančios žiniatinklio IS, kuriose prieš testavimą nebuvo pastebėta ir papildomai pasėta dirbtinių klaidų – programų versijos buvo "švarios". Išanalizavus rezultatus, paaiškėjo, jog daugelis orakulų pateikė didelį kiekį klaidingai teigiamų rezultatų, t.y. HTML pakeitimų, kurie klaidingai buvo identifikuoti kaip klaidos [29]. Kai kurie testų orakulai, įskaitant ankstesniuose tyrimuose [30] turėjusį didžiausią efektyvumą (**N+I** ∪ **C-WD**), kiekvieną HTML atsaką palaikydavo klaidingai teigiamu. Mažiausiai klaidingai teigiamų rezultatų grąžindavo **N-S+U**.

Pats efektyviausias konkrečiai žiniatinklio IS atrinktas orakulas pagal Sprenkle [29] yra **dD** ∪ **N+F-Sel** lenkia net ir **N+I** ∪ **C-WD**. Nepaisant didelio klaidingai teigiamų rezultatų grąžinimo, universaliausias lieka **N+I** ∪ **C-WD** testų orakulas tikrinantis ir struktūrą ir turinį.

2.5 Žiniatinklio IS regresinio testavimo įrankiai

Žiniatinklio informacines sistemas testuojantys įrankiai dažniausiai būna labai universalūs ir geba vykdyti tokius testavimus: apkrovos, regresinį, funkcinį ir grafinės vartotojo sąsajos. Programuotojui ar testuotojui labai patogu turėti vieną įrankį, su kuriuo galima ištestuoti kuriamą informacinę sistemą įvairiais būdais. Tačiau atlikus įrankių analizę galima teigti, kad regresinį testavimą pilnai vykdančių įrankių beveik nėra. Didžioji įrankių dalis geba tik automatiškai sudaryti ir vykdyti testus, tačiau šiuose įrankiuose gauti rezultatai nėra tikrinami – nėra testų orakulo. Nedidelė įrankių dalis leidžia kiekvienam testavimo atvejui pasirašyti tikrinimo kriterijus reguliariųjų išraiškų pagalba, tačiau tai tik labai laikinas automatizavimas. Pavyko surasti tik vieną testavimo įrankį, kuris tenkina minimalius automatinio regresinio testavimo reikalavimus.

TestComplete – darbastalio tipo įrankis, kuris testuoja žiniatinklio ir darbastalio tipo programinę įrangą. Testavimo atvejai sudaromi pačiu nepatvariausiu būdu, kai fiksuojamos pelės koordinatės, pelės ratuko pasukimai ir paspausti klaviatūros klavišai (žiūrėti 2.4.2.1Vartotojo sąsaja paremti testavimo atvejai). TestComplete įrankio regresinių testų orakulas HTML atsaką geba tikrinti tokiais būdais:

- HTML atsakas lyginamas kaip tekstas.
- tikrinamas bet koks HTML žymų ir atributų pasikeitimas.
- galima pažymėti konkretų regioną ar objektą, kurį įrankis tikrins.

2.6 Išvados

Atlikus analizę galime teigti, jog šių dienų žiniatinklio IS regresinio testavimo įrankis turėtų tenkinti šiuos reikalavimus:

1. Testavimas turėtų būti vykdomas automatiškai: testų sudarymas, testų vykdymas, rezultatų išgavimas, rezultatų palyginimas (atlieka testų orakulas). Nors rezultatų palyginimą visiškai automatizuoti yra sudėtinga, tačiau keliami aukšti reikalavimai, kad testuotojo peržiūros reikalaujančių klaidų kiekis būtų kiek įmanoma sumažintas.
2. Testavimo atvejai sudaromi automatiškai, kai testuotojas imituoja vartotojo veiksmus, o įrankis šiuos veiksmus fiksuoja. Pasirinkta testavimo atvejų sudarymo technika neturėtų sukelti perteklinių testavimo atvejų mažinimo problemos.
3. Testavimo atvejai turėtų būti patvarūs ir kiek įmanoma atsparūs sistemos ar grafinės vartotojo sąsajos pokyčiams.

Regresinių testų orakului yra keliami tokie reikalavimai:

1. Testų orakulas turėtų būti universalus, pritaikytas kuo platesnei žiniatinklio IS aibei.
2. Testų orakulas turėtų ieškoti semantinių pokyčių HTML atsake, nes tik taip nagrinėjant HTML atsaką galime sumažinti klaidingai teigiamų ir klaidingai neigiamų rezultatų kiekį.
3. Testų orakulas turėtų nagrinėti ir HTML žymų struktūrą ir tarp žymų esantį turinį, tuomet efektyvumas būna didžiausias. Tačiau turinio nagrinėjimas dažniausiai priklauso nuo konkrečios žiniatinklio IS.
4. Svarbiau yra aptikti visas galimas klaidas (aukštas aptikimas) ir taip pašalinti klaidingai neigiamus orakulo rezultatus, negu dalį klaidų praleisti sumažinant klaidingai teigiamų rezultatų kiekį. Tačiau klaidingai neigiamų rezultatų mažinimas neturėtų stipriai padidinti klaidingai teigiamų rezultatų skaičiaus.

3 PROJEKTINĖ DALIS

3.1 Tikslas

Projekto tikslas: sukurti žiniatinklio informacinių sistemų apkrovos testavimo įrankį, kuris gebėtų suteikti testuotojui žinių apie testuojamos informacinės sistemos ir atskirų jos dalių sudėtingumą. Testuotojas, gavęs ištestuotos informacinės sistemos testavimo rezultatus, galės daryti išvadas apie jos sudėtingumą ir skirti daugiau dėmesio sudėtingesnių modulių programinio kodo optimizavimui.

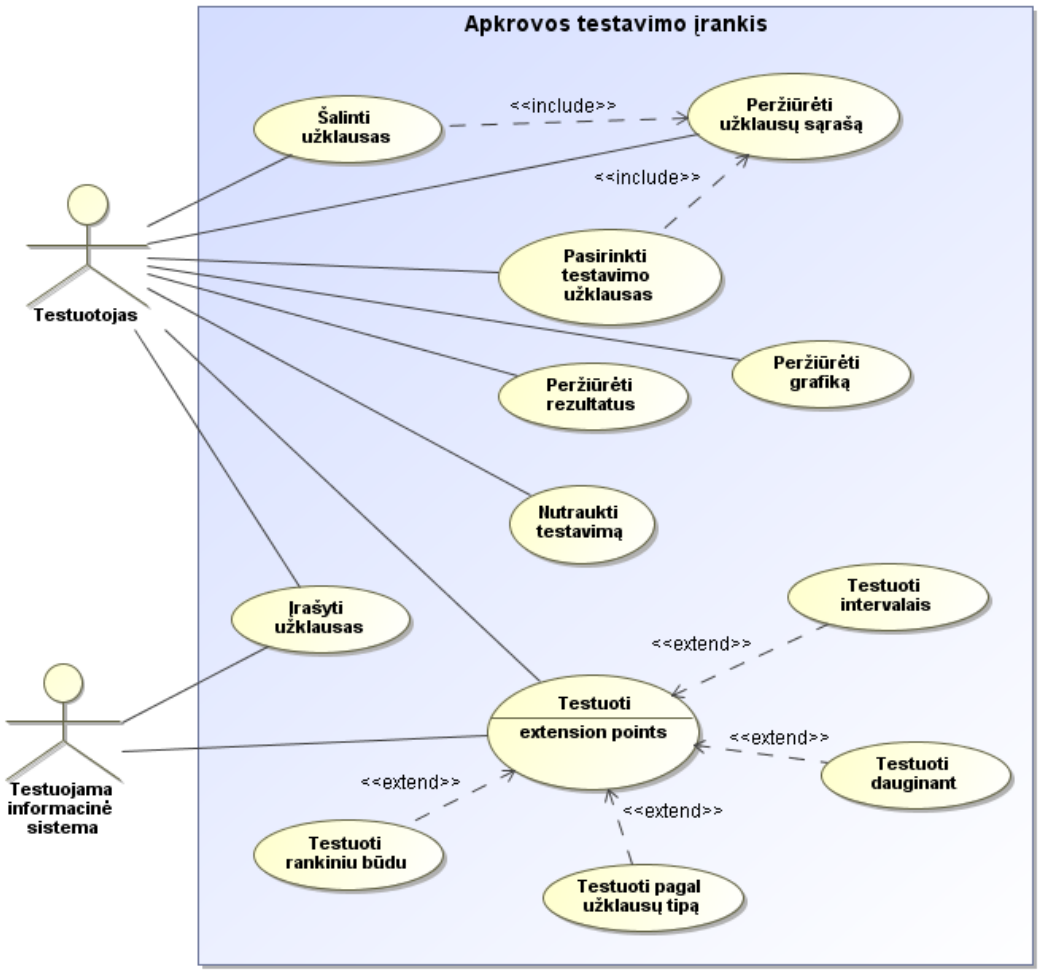
Įrankis taip pat parodys, su kuriomis užklausomis persiunčiamas didžiausias duomenų kiekis, ko pasekoje bus galima daryti išvadą apie serveriui reikalingą linijos pralaidumą. Taip pat, kuriamas įrankis turi būti nepriklausomas nuo testuojamo objekto – turi būti universalus ir turi testuoti visų tipų žiniatinklio informacines sistemas.

3.2 Sistemos kontekstas

Kuriamas testavimo įrankis bus naršyklės *Mozilla FireFox* įskiepis. Todėl galios visi šios naršyklės apribojimai ir privalumai. Naršyklė reikalinga HTTP kanalui sukurti, pasiekti iš jos išeinančias ir į ją ateinančias užklausas, atvaizduoti gautų užklausų duomenis.

3.3 Panaudojimo atvejų vaizdas

Apkrovos testavimo įrankio panaudojimo atvejai pateikti 6 paveikslėlyje. Taip pat pateikti kiekvieno panaudojimo atvejo aprašymai. Detaliai panaudojimo atvejai aprašomi 2 – 13 lentelėse.



6 Paveikslėlis. Panaudojimo atvejų diagrama

2 Lentelė. „Įrašyti užklausas“ PA.

1. Panaudojimo atvejis	**Įrašyti užklausas**
Aktoriai	Testuotojas, Testuojama informacinė sistema
Aprašas	Įrašyti į atmintį testuotojo generuojamas užklausas naršyklėje.
Prieš sąlygos	Įjungta naršyklė, įjungtas testavimo įrankis
Sužadinimo sąlygos	Paspaudžiamas įrašymo mygtukas
Po sąlygos	Užregistruotos testuotojo sugeneruotos užklausos naršant po internetinį puslapį.

3 Lentelė. „Pasirinkti testavimo užklausas“ PA.

2. Panaudojimo atvejis	**Pasirinkti testavimo užklausas**
Aktoriai	Testuotojas
Aprašas	Iš visų įrašytų užklausų parinkti tik tas, kurios bus naudojamos artimiausiam testavimui.
Prieš sąlygos	Pažymėtos visos įrašytos užklausos.
Sužadinimo sąlygos	Pažymimas arba atžymimas žymimasis langelis (*checkbox*).
Po sąlygos	Pasirinktos artimiausiam testavimui reikalingos užklausos.

4 Lentelė. „Peržiūrėti užklausų sąrašą“ PA.

3. Panaudojimo atvejis	**Peržiūrėti užklausų sąrašą**
Aktoriai	Testuotojas
Aprašas	Patikrinti, ar įsirašė visos reikalingos užklausos. Sąraše galima patikrinti užklausos įvykdymo laiką bei matyti konkrečius užklausos duomenis.
Prieš sąlygos	Yra įrašytų užklausų.
Sužadinimo sąlygos	Įjungiamas užklausų sąrašas.
Po sąlygos	Peržiūrėtos ir patikrintos įrašytos užklausos.

5 Lentelė. „Šalinti užklausas“ PA.

4. Panaudojimo atvejis	**Šalinti užklausas**
Aktoriai	Testuotojas
Aprašas	Pašalinti nereikalingas, pasikartojančias, per klaidą įrašytas užklausas po vieną arba visas iškarto.
Prieš sąlygos	Sąraše atvaizduojamos visos užklausos.
Sužadinimo sąlygos	Paspaudžiamas trynimo mygtukas.
Po sąlygos	Iš sąrašo pašalinta užklausa arba visos užklausos.

6 Lentelė. „Peržiūrėti rezultatus“ PA.

5. Panaudojimo atvejis	**Peržiūrėti rezultatus**
Aktoriai	Testuotojas
Aprašas	Pamatyti testavimo metu surinktus serverio atsako laikus esant skirtingiems užklausų kiekiams.
Prieš sąlygos	Atliktas testavimas.
Sužadinimo sąlygos	Įjungiamas testavimo rezultatų sąrašai
Po sąlygos	Peržiūrėti testavimo rezultatai.

7 Lentelė. „Peržiūrėti grafiką“ PA.

6. Panaudojimo atvejis	**Peržiūrėti grafiką**
Aktoriai	Testuotojas
Aprašas	Vizualiai pamatyti gautus testavimo rezultatus. Grafike atvaizduojama serverio atsako laikų ir išsiųstų užklausų kiekio priklausomybė.
Prieš sąlygos	Nėra testavimo rezultatų, nepasirinktas grafiko atvaizdavimas.
Sužadinimo sąlygos	Baigiamas testavimas.
Po sąlygos	Testavimo duomenys atvaizduojami grafike.

8 Lentelė. „Nutraukti testavimą“ PA.

7. Panaudojimo atvejis	**Nutraukti testavimą**
Aktoriai	Testuotojas
Aprašas	Jeigu testavimas yra prasidėjęs, galima jį nutraukti.
Prieš sąlygos	Vyksta testavimas.
Sužadinimo sąlygos	Paspaudžiamas nutraukimo mygtukas.
Po sąlygos	Testavimas nutraukiamas.

9 Lentelė. „Testuoti“ PA.

8. Panaudojimo atvejis	**Testuoti**
Aktoriai	Testuotojas, Testuojama sistema
Aprašas	Vykdo vieną iš keturių testavimo scenarijų.
Prieš sąlygos	Neatliktas testavimas, nėra gautų testavimo rezultatų.
Sužadinimo sąlygos	Paspaudžiamas testavimo mygtukas.
Po sąlygos	Ištestuota informacinė sistema.

10 Lentelė. „Testuoti rankiniu būdu“ PA.

9. Panaudojimo atvejis	**Testuoti rankiniu būdu**
Aktoriai	Testuotojas
Aprašas	Testuojama vieną ar keletą kartų nurodant konkretaus dydžio apkrovą.
Prieš sąlygos	Neatliktas testavimas, nėra gautų testavimo rezultatų.
Sužadinimo sąlygos	Paspaudžiamas testavimo mygtukas.
Po sąlygos	Ištestuota informacinė sistema.

11 Lentelė. „Testuoti intervalais“ PA.

10. Panaudojimo atvejis	**Testuoti intervalais**
Aktoriai	Testuotojas
Aprašas	Testuojama tam tikrą skaičių kartų ir tam tikro apkrovos intervalo ribose, kuriuos nurodo testuotojas.
Prieš sąlygos	Neatliktas testavimas, nėra gautų testavimo rezultatų.
Sužadinimo sąlygos	Paspaudžiamas testavimo mygtukas.
Po sąlygos	Ištestuota informacinė sistema.

12 Lentelė. „Testuoti dauginant“ PA.

11. Panaudojimo atvejis	**Testuoti dauginant**
Aktoriai	Testuotojas
Aprašas	Testuojama dauginant testavimui pasirinktų užklausų kiekį tiek kartų, kiek nurodė testuotojas. Tokiu būdu gaunama didėjanti serverio apkrova.
Prieš sąlygos	Neatliktas testavimas, nėra gautų testavimo rezultatų.
Sužadinimo sąlygos	Paspaudžiamas testavimo mygtukas.
Po sąlygos	Ištestuota informacinė sistema.

13 Lentelė. „Testavimas pagal užklausų tipą“ PA.

12. Panaudojimo atvejis	**Testavimas pagal užklausų tipą**
Aktoriai	Testuotojas
Aprašas	Testuojama su kiekviena pasirinkta skirtinga užklausa atskirai ir nurodyta apkrova. Gaunami rezultatai parodo, kuriom užklausom apdoroti serveris užtrunka ilgiausiai. Matomas atskirų testuojamos sistemos modulių sudėtingumas.
Prieš sąlygos	Neatliktas testavimas, nėra gautų testavimo rezultatų.
Sužadinimo sąlygos	Paspaudžiamas testavimo mygtukas.
Po sąlygos	Ištestuota informacinė sistema.

3.4 Funkciniai reikalavimai ir reikalavimai duomenims

Apkrovos testavimo įrankiui projektuoti surinkti svarbiausi reikalavimai:

1. Testavimo įrankis perima įrankio naudotojo iššauktas užklausas.
2. Testavimo įrankis išsaugo perimtas užklausas.
3. Įrankio naudotojas pasirenka artimiausiam testavimui reikalingas užklausas.
4. Įrankio naudotojas peržiūri testavimo įrankyje išsaugotas užklausas.
5. Įrankio naudotojas gali pašalinti nereikalingas užklausas po vieną.
6. Įrankio naudotojas gali pašalinti visas užklausas.
7. Įrankio vartotojas peržiūri testavimo metu surinktus rezultatus.
8. Įrankio naudotojas peržiūri testavimo metu sukauptų duomenų grafiką.
9. Įrankio naudotojas nutraukia testavimo procesą.
10. Įrankio naudotojas atlieka testavimą pasirinkdamas testavimo scenarijų.
11. Įrankio vartotojas pats nustato generuojamos apkrovos dydį ir testavimo etapų skaičių.
12. Įrankio naudotojas pasirenka kuriamos apkrovos intervalą ir nurodo tame intervale atliekamų testavimo etapų skaičių.
13. Įrankio naudotojas nurodo daugiklį ir testavimo etapų skaičių.
14. Įrankio naudotojas gali testuoti informacinę sistemą pagal surinktų užklausų tipą.
15. Kad būtų gauti testavimo rezultatai, apkrovos testavimo įrankis testavimo metu turi išmatuoti užklausų vykdymo laikus.
16. Nustatytas apkrovos dydis siunčiamas į nurodytą IS serverį.
17. Įrankio naudotojo pasirinktos užklausos prieš testavimą nukopijuojamos į atskirą duomenų struktūrą.

18. Atliekant testavimą pagal užklausų tipą parenkamos užklausos.
19. Vidutinių testavimo laikų apskaičiavimas.
20. Gauto atsako šaltinio tikrinimas.
21. Įrankio nustatymų ir parinkčių išsaugojimas.

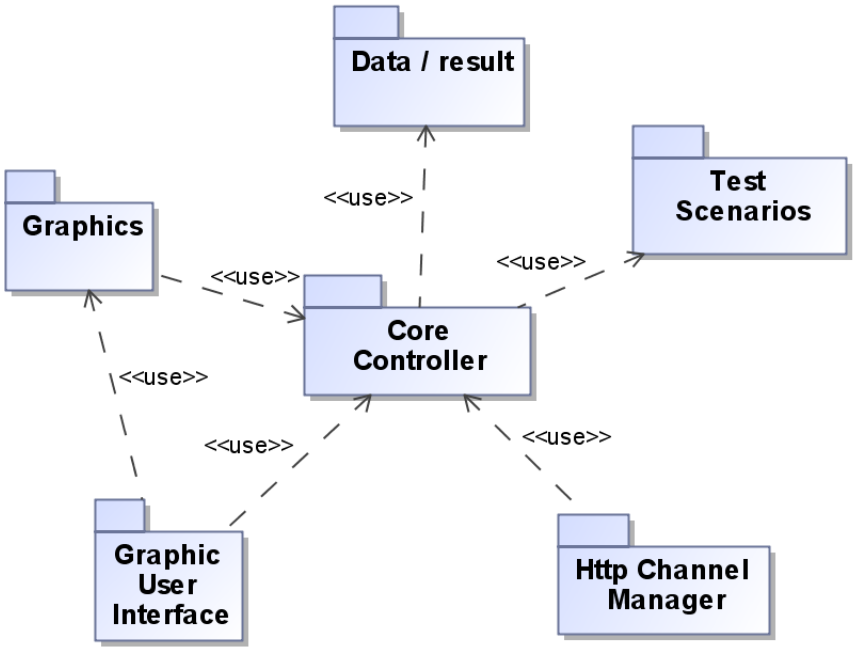
3.5 Nefunkciniai reikalavimai

Nefunkciniai reikalavimai reikalingi suprojektuoti apkrovos testavimo įrankį:

1. Paprasta, greitai perprantama, intuityvi ir netrukdanti vartotojo sąsaja.
2. Įrankio naudotojams turi būti lengva naudotis testavimo įrankiu.
3. Paaiškinimai, pavadinimai ir terminai anglų kalba.
4. Matuojamas užklausų įvykdymas turėtų būti kuo tikslesnis.
5. Efektyvus kompiuterio resursų panaudojimas.
6. Kiek įmanoma, aukštesnis patikimumas.
7. Įrankis turi būti nesunkiai palaikomas ir jo galimybės praplečiamos.
8. Įrankio komponentai turėtų būti perkami / naudojami tik iš patikimų įmonių ar organizacijų.
9. Kūrimo proceso metu turi būti laikomasi ISO 9000 serijos standartų.

3.6 Sistemos statinis vaizdas

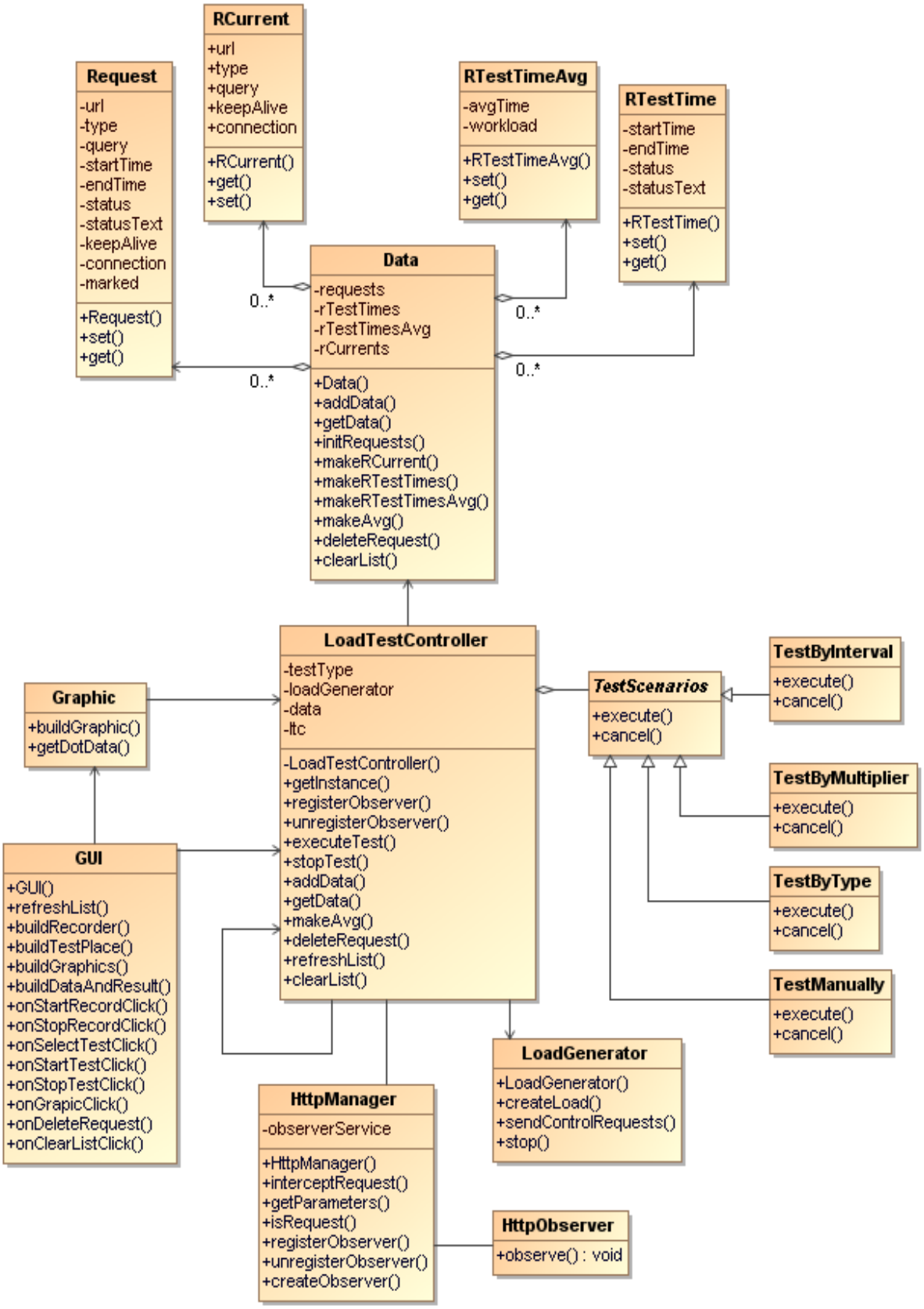
Apkrovos testavimo įrankis suskaidytas į 6 paketus (7 pav.).



7 Paveikslėlis. Testavimo įrankio paketų diagrama

Žemiau pateikta bendra, nesuskirstyta į paketus, klasių diagrama (8 pav.). Į *Data* klasę kreipiasi 4 klasės: *Request, RCurrent, RTestTime, RTestTimeAvg*, kurių ryšiai yra *aggregation*

tipo. Į *TestScenarios* klasę kreipiasi taip pat 4 klasės: *TestByInterval*, *TestByMultiplier*, *TestByType*, *TestManually*, kurių ryšiai yra *generalization* tipo. Likę kitokio tipo ryšiai yra *directedassociation* arba *association*.



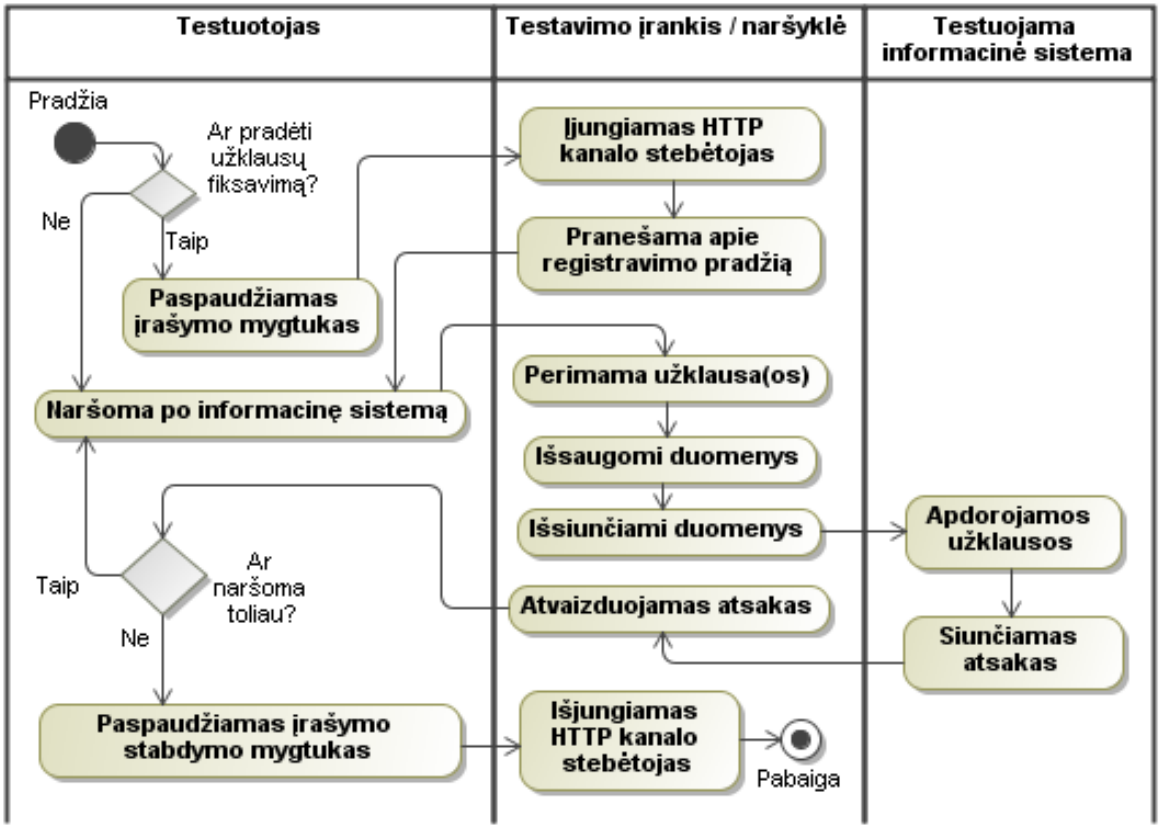
8 Paveikslėlis. Bendra visų testavimo įrankio klasių diagrama

3.7 Sistemos dinaminis vaizdas

Šiame skyriuje pateikiamas apkrovos testavimo įrankio veikimo principas ir esminės elgsenos ypatybės.

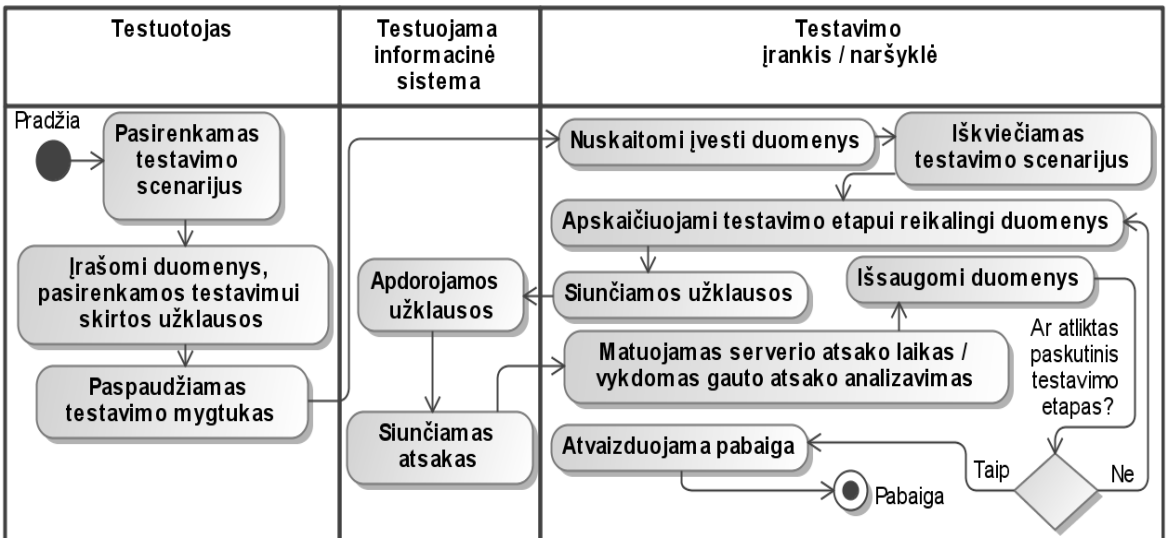
3.7.1 Testavimo įrankio veiklos diagramos

Kadangi apkrovos testavimo įrankis yra Mozilla Firefox naršyklės dalis ir su ja yra glaudžiai susijęs, veiklos diagramose pateikiamas kaip bendra sistema su naršykle. Kad būtų galima įrašyti užklausas, testuotojas turi įjungti HTTP kanalo stebėtoją. Tuomet visos naršyklėje testuotojo generuojamos užklausos bus fiksuojamos. Jeigu reikia fiksuoti tik konkrečią užklausą, HTTP stebėtoją galima sustabdyti, puslapyje naršyti iki ten, kur reikia ir HTTP stebėtoją įjungti tada, kai to reikia (9 pav.).



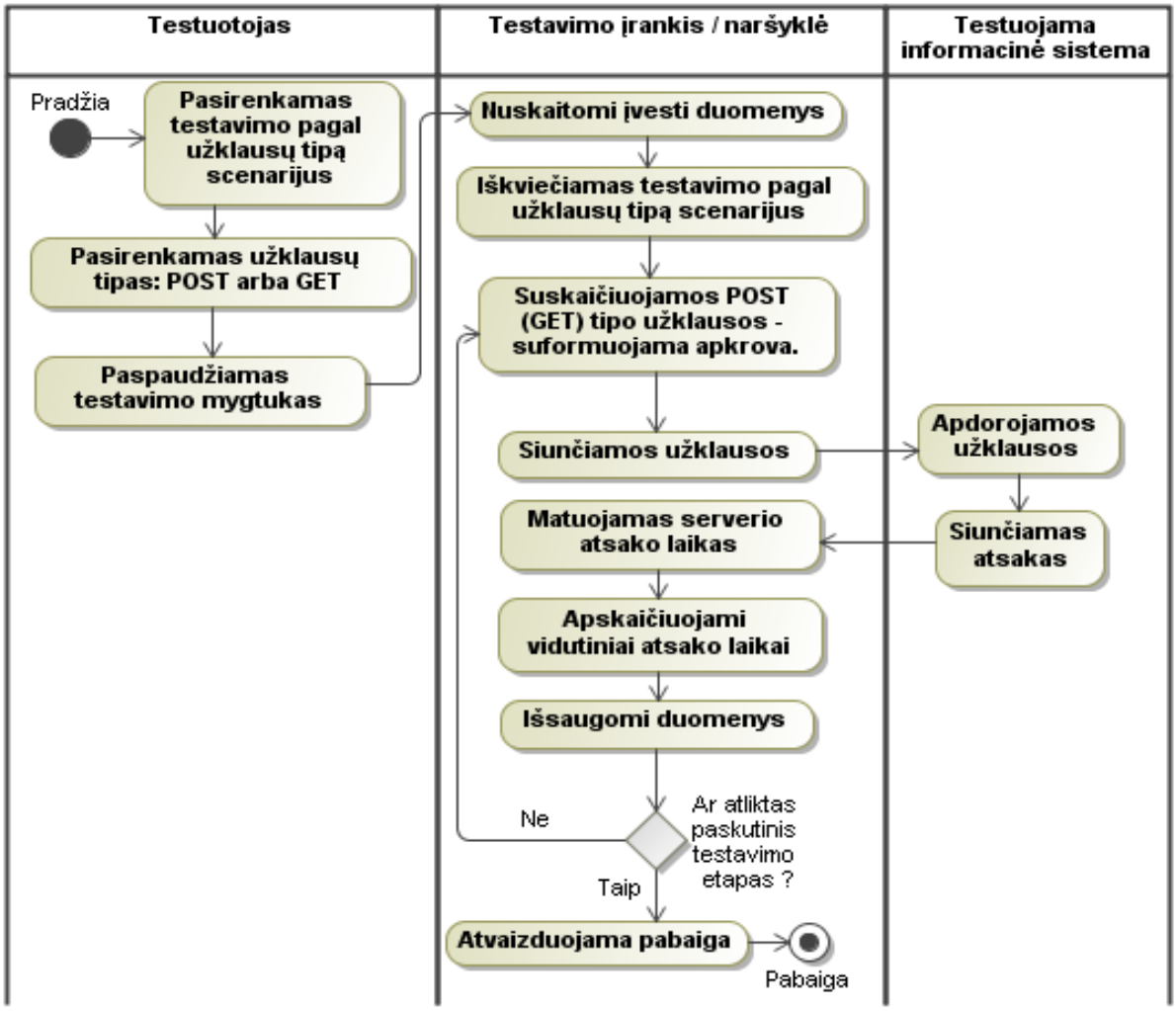
9 Paveikslėlis. „Įrašyti užklausas“ veiklos diagrama

Žemiau pateikta diagrama (10 pav.) atspindi bendrą testavimo scenarijų veikimo principą. Kai testuotojas pasirenka testavimo scenarijų ir suveda duomenis, iškviečiamas pasirinktas scenarijus, kuriame yra apskaičiuojami testavimo etapai ir reikalingos išsiųsti apkrovos, jei tai apkrovos testavimo scenarijus. Regresinio testavimo metu etapų kiekis yra tolygus iškviestam regresijos aptikimo testui. Tokių etapų kiekis priklauso nuo to, koks informacinės sistemos modulis yra testuojamas, ir kiek yra numatytų priklausomybių su kitais testais.



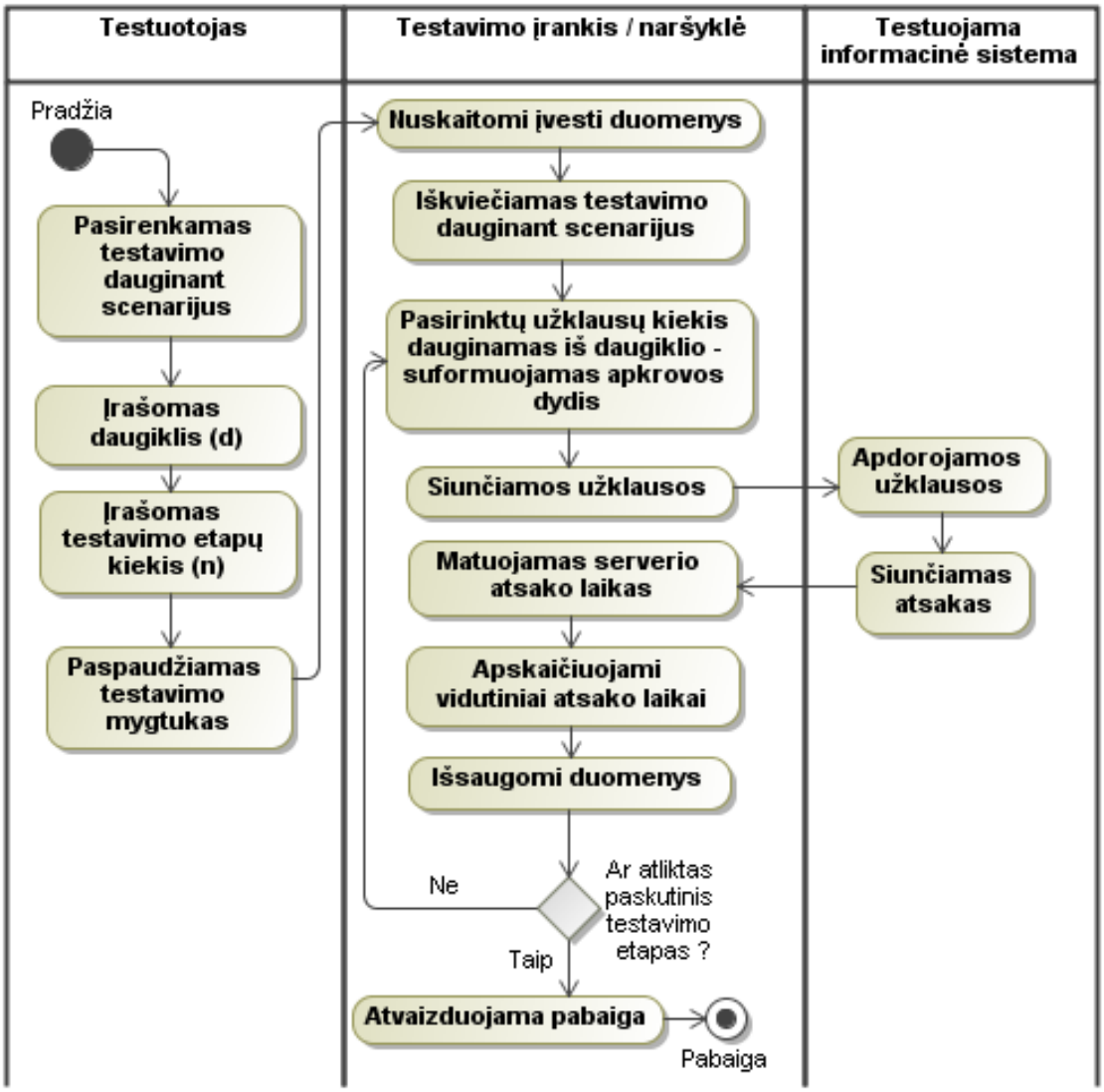
10 Paveikslėlis. Regresinio ir apkrovos testavimo veiklos diagrama

Žemiau esančiose diagramose pavaizduoti konkretūs testavimo scenarijai (11 – 14 pav.). Jie tarpusavyje skiriasi įvedamų duomenų kiekiu, tipu ir atitinkamai suformuojamais skirtingais testavimo etapų kiekiais.



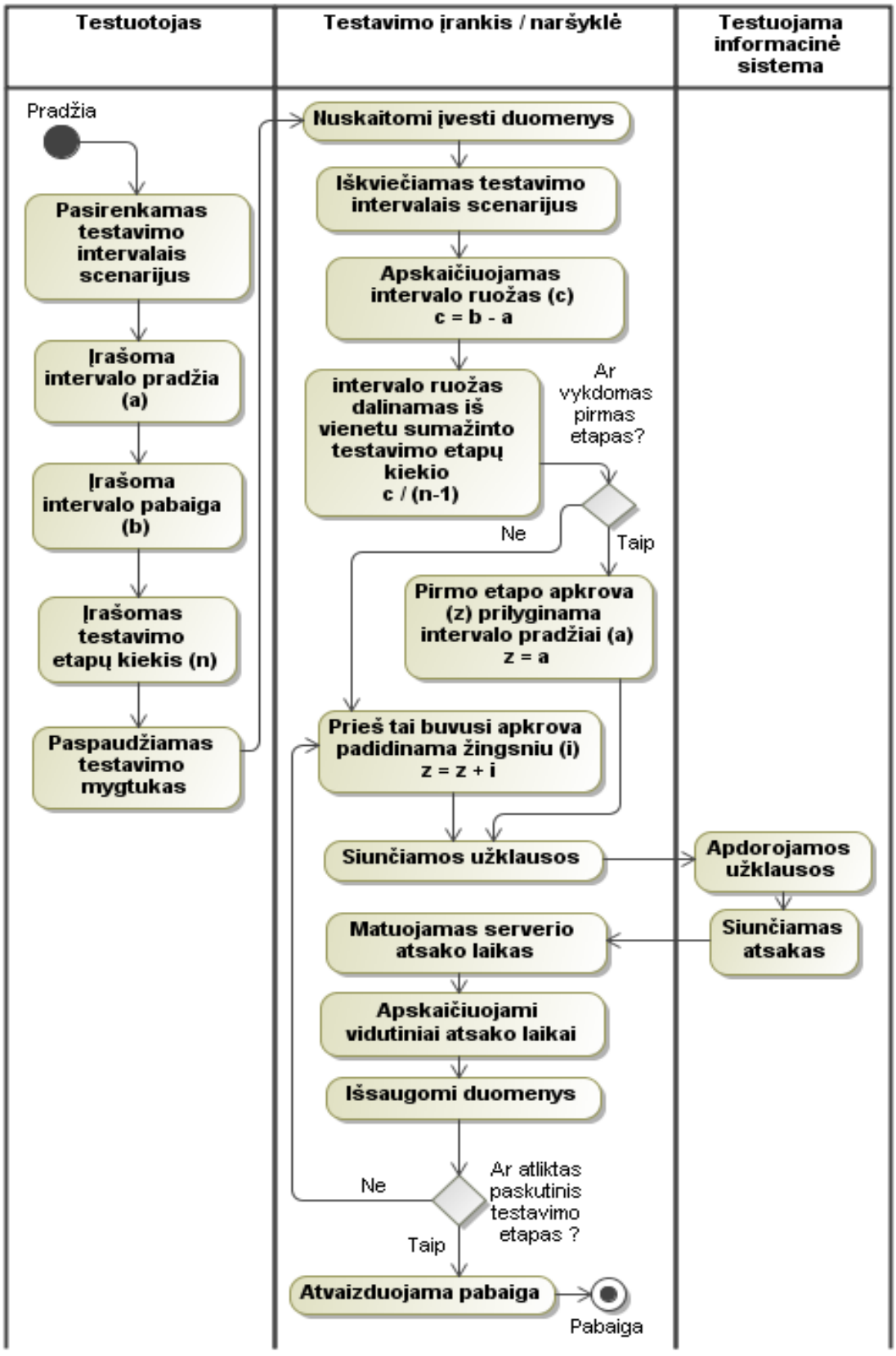
11 Paveikslėlis. „Testavimas pagal užklausų tipą“ veiklos diagrama

Kaip pavaizduota diagramoje (11 pav.), šio testavimo scenarijaus vykdymo metu kaip apkrova siunčiamos tik konkretaus tipo užklausos: GET arba POST.



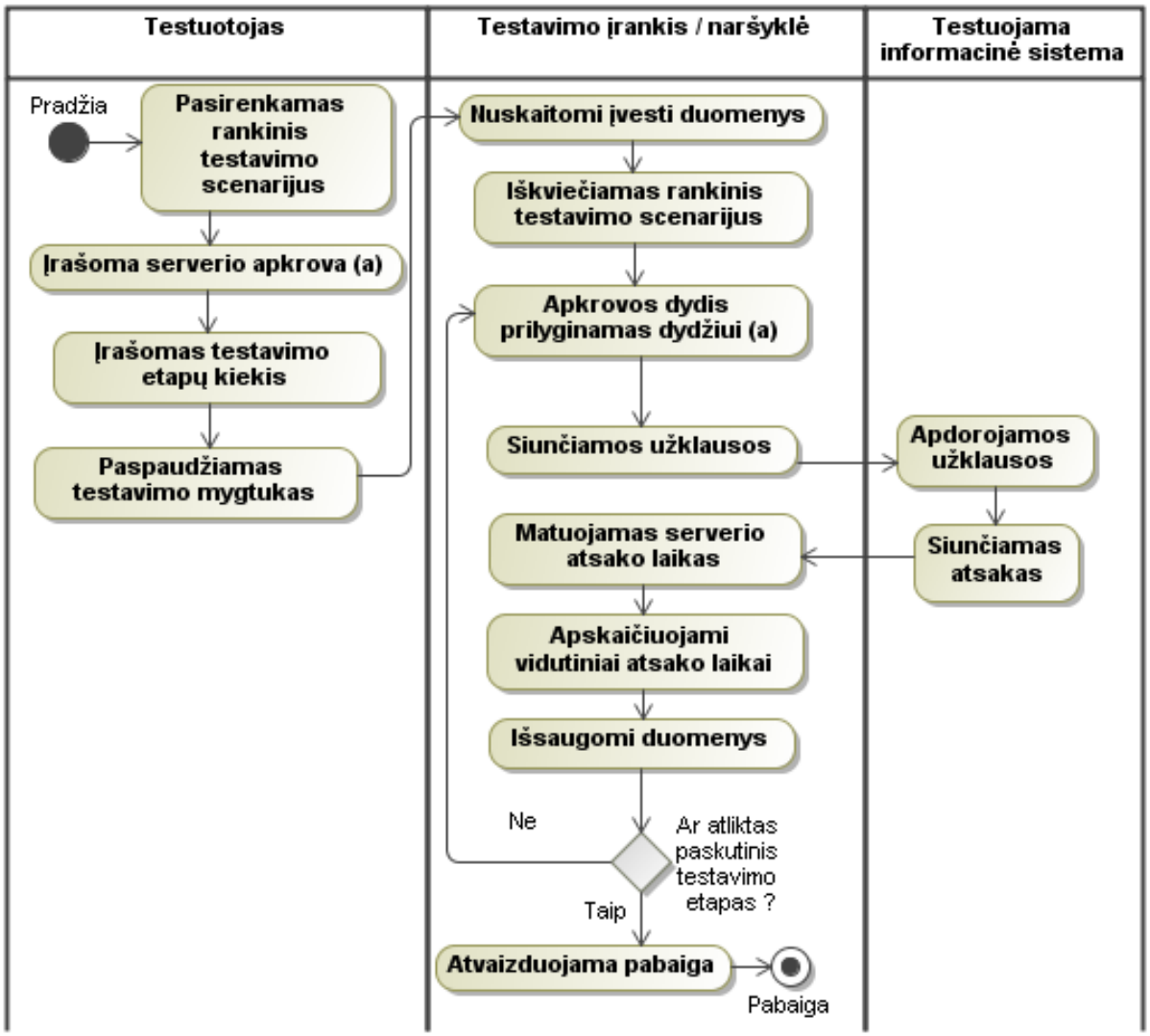
12 Paveikslėlis. „Testuoti dauginant“ veiklos diagrama

Šio testavimo scenarijaus metu (12 pav.) galima suformuoti eksponentiškai kintančią apkrovą, kuri parodys, kaip informacinė sistema susidoroja su greitai padidėjusia apkrova.



13 Paveikslėlis. „Testuoti intervalais“ veiklos diagrama

Vykdydami testavimą intervalais (13 pav.) galime formuoti apkrovos dydžius pasirinktame intervale.

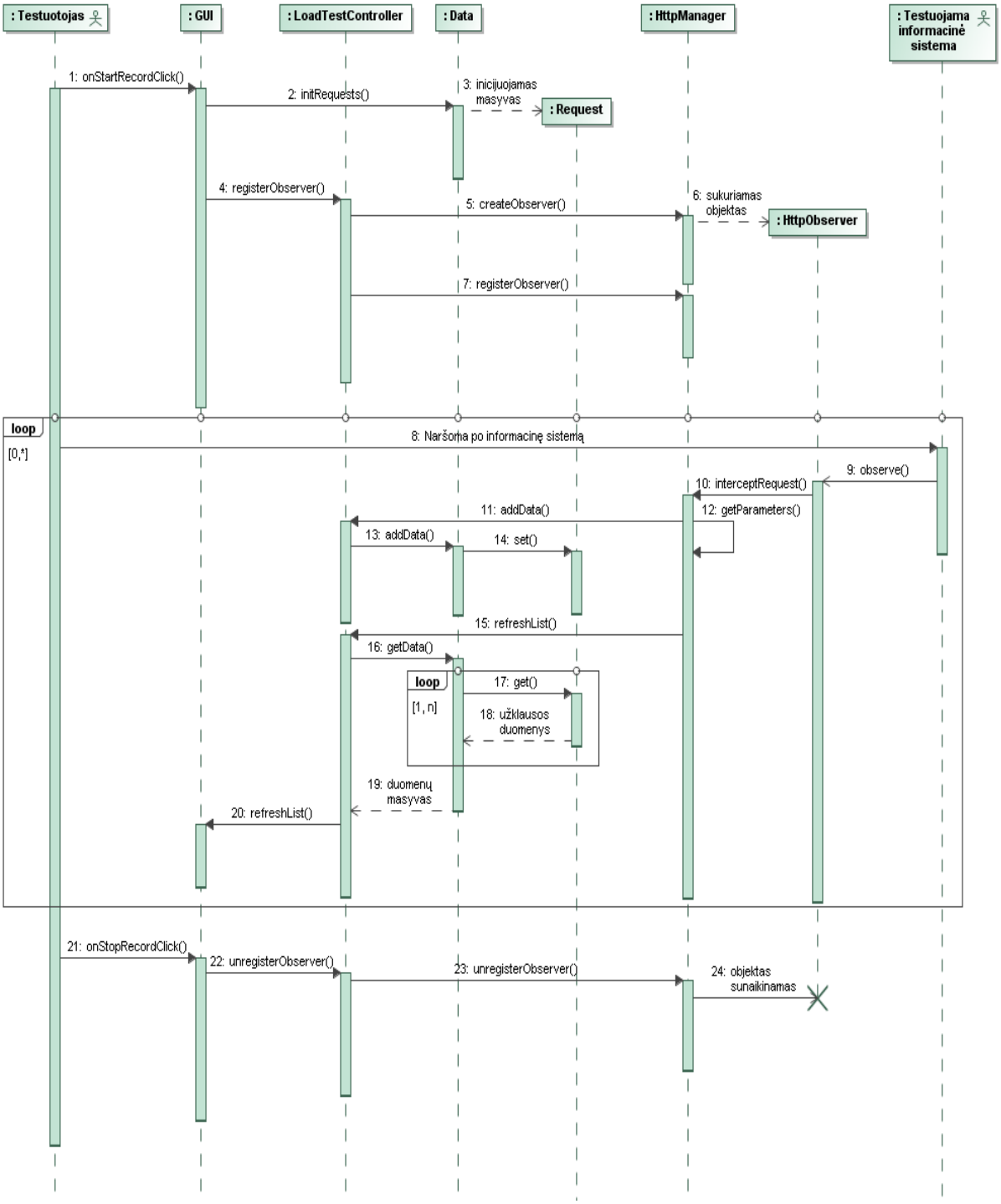


14 Paveikslėlis. „Testuoti rankiniu būdu“ veiklos diagrama

Testavimas kai užklausų kiekį galima nurodyti nepriklausomai nuo testavimo etapų kiekio pavaizduotas 14 paveikslėlyje.

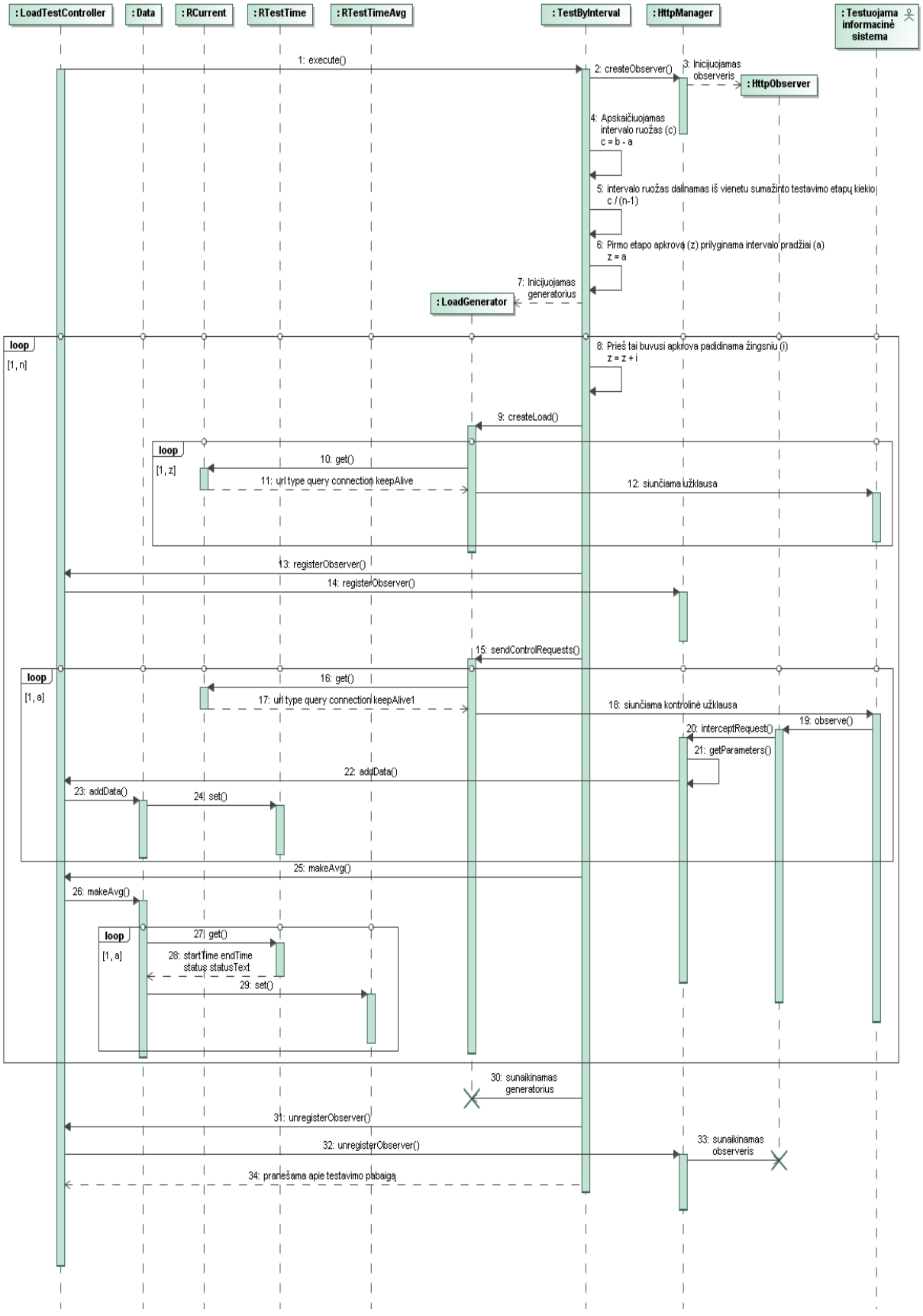
3.7.2 Testavimo įrankio sekų diagramos

15 paveikslėlyje pavaizduota užklausų įrašymo metu vykdoma seka. Iš pradžių yra užregistruojamas HTTP stebėtojas, tuomet vartotojo veiksmai naršyklėje iššaukia užklausų fiksavimą.



15 Paveikslėlis. „Įrašyti užklausas“ sekų diagrama

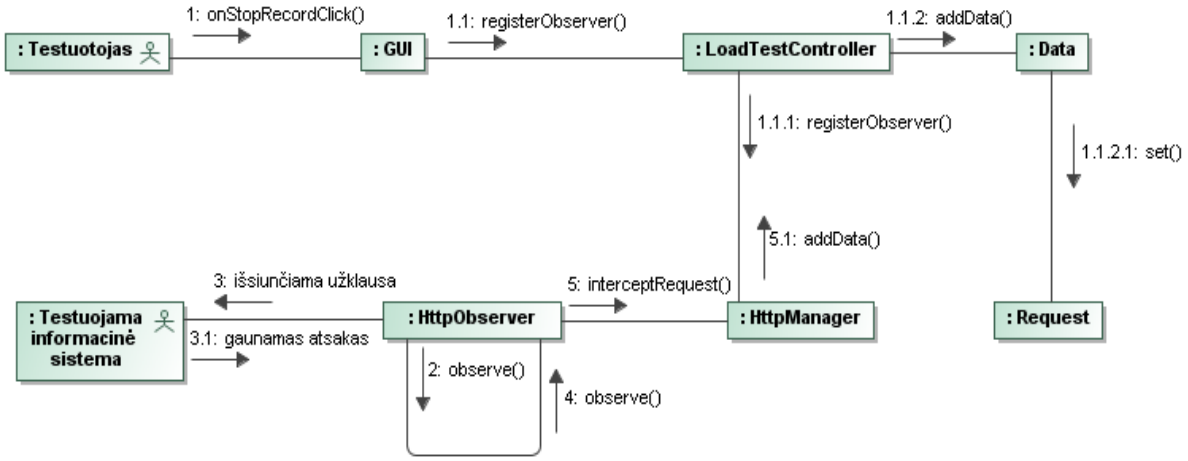
Žemiau pateikta diagrama (16 pav.) atspindi testavimo scenarijų veiklą. **Pastaba**: diagramoje nepavaizduota testuotojo sąveika, kadangi turima galvoje, kad testuotojas jau paspaudė testavimo mygtuką.



16 Paveikslėlis. „Testuoti intervalais“ sekų diagrama

3.7.3 Testavimo įrankio bendradarbiavimo diagramos

17 pav. pavaizduota užklausų įrašymo bendradarbiavimo diagrama. Nors „Testuojama informacinė“ sistema tiesiogiai nebendrauja su *HttpObserver* objektu, tačiau netiesiogiai šis objektas reaguoja į kiekvieną užklausą / atsaką siunčiamą ar parsiunčiamą į naršyklę.



17 Paveikslėlis. Užklausų įrašymo metu žinučių perdavimai. Bendradarbiavimo diagrama

3.8 Išdėstymo vaizdas

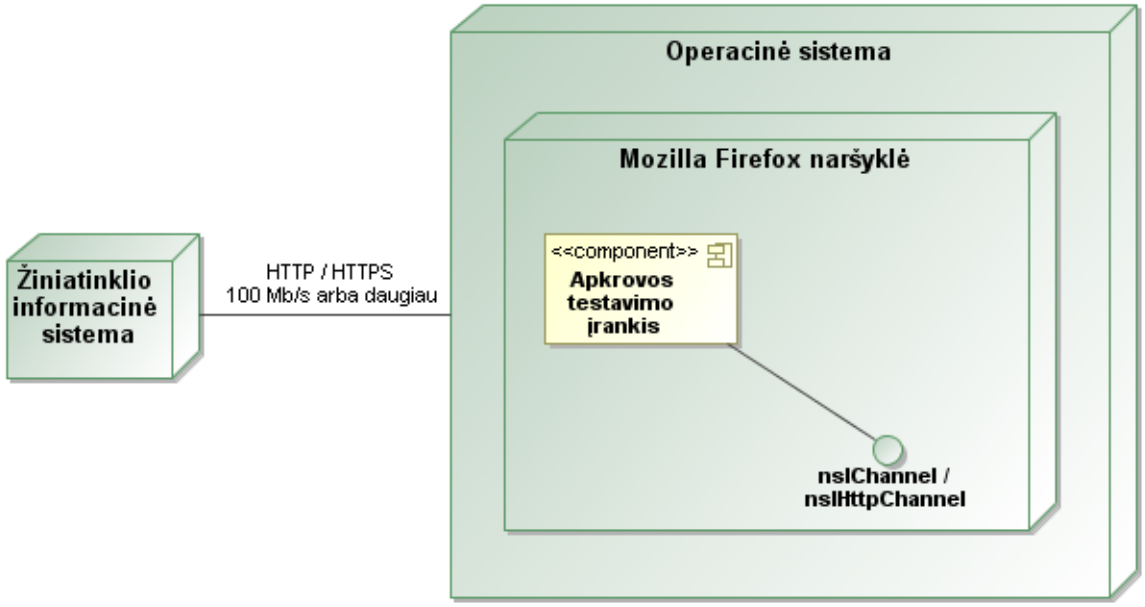
Apkrovos testavimo įrankio išdėstymo vaizdas parodytas 18 pav. Kadangi testavimo įrankis yra *Mozilla Firefox* įskiepis, visi reikalingi failai yra toje pačioje įskiepio vietoje. Nėra naudojamasi nutolusiais resursais. Naršyklė komunikuoja su testuojama informacine sistema HTTP / HTTPS protokolo pagalba. Programinė įranga reikalinga apkrovos testavimo įrankio veikimui:

- Windows Xp / Vista / 7 / Linux / Mac OS (ir kitos OS, kuriose veikia *Mozilla Firefox*)
- Mozilla Firefox v3.6.* , v4.* arba naujesnė.

Minimalūs techniniai reikalavimai:

- Bent 1 Ghz procesorius.
- 1 Gb RAM atminties.
- 52 MB kieto disko atmintis.
- 100 Mb/s interneto pralaidumas.

Nors Windows Xp operacinei sistemai ar *Mozilla Firefox* naršyklei keliami reikalavimai yra bent 5 kartus mažesni ir apkrovos testavimo įrankis veiktų pavyzdžiui su Pentium II 233 MHz procesoriumi, tačiau toks procesorius nesukurtų pakankamos apkrovos, kuria būtų galima nors kiek apkrauti informacinės sistemos serverį. Todėl minimalūs reikalavimai remiasi *Mozilla Firefox* naršykle ir Windows 7 operacine sistema, kuriai pačiai reikalavimai yra daug aukštesni lyginant su kitomis operacinėmis sistemomis.



18 Paveikslėlis. Apkrovos testavimo įrankio išdėstymo vaizdas

Pasirinkus vieną iš 4 apkrovos testavimo scenarijų, reikia nurodyti apkrovos dydi, kuris nusakomas siunčiamų užklausų kiekiu ir testavimo etapų kiekį, kuris leis aiškiai matyti grafike, kokiai apkrovai esant, serveris sugaišta vienokį ar kitokį laiko tarpą, kol apdoroja užklausas.

Pasirinkus regresijos testavimo scenarijų, užklausos yra siunčiamos po vieną ir sulaukiamas atsakas iš serverio. Naujai gautas atsakas yra palyginamas su užklausos įrašymo metu išsaugotuoju atsaku. Esant tam tikriems neatitikimams, įrankis parodys, kad yra klaida. Iš serverio gautas atsakas nebūtinai turi būti identiškas anksčiau išsaugotajam. Informacinės sistemos puslapyje gali atsirasti naujų mygtukų, laukų, formų ir t.t. Regresijos testavimo metu įrankis tai aptiks ir į naujus pakeitimus nekreips dėmesio (papildomas IS funkcionalumas).

3.9 Regresinio testavimo metu kylančių problemų sprendimas

3.9.1 Testavimo atvejų sudarymas

Mūsų siūlomas regresinio testavimo įrankis įgyvendintas panaudojant antrą (II) žiniatinklio pagrindu kuriamų testavimo atvejų būdą (žiūrėti 2.4.2.1 Vartotojo sąsaja paremti testavimo atvejai).

3.9.2 Testavimo atvejų kiekio mažinimas

Mūsų siūlomame įrankyje testavimo atvejų kiekis mažinamas nustatant testavimo atvejų priklausomybes [16, 36]. Pavyzdžiui, programuotojas žino, jog naują žiniatinklio IS versiją reikėtų testuoti panaudojant konkrečius 5 testavimo atvejus, nes jie tiesiogiai siejasi su programuotojo atliktais pakeitimais. Tačiau šie 5 testavimo atvejai turi papildomų

priklausomybių, kurių kiekvieną kartą nustatinėti nereikia. Susiejimas tarp testavimo atvejų atliekamas sukuriant testą. Tai nėra automatinis būdas. Jį turi atlikti testuotojas arba programuotojas, pažymėdamas, jog vykdant vieną testų grupę, reikėtų įvykdyti ir kitą. Galimas ir atvirkščias variantas: pažymima, kurie testavimo atvejai neturėtų būti vykdomi, t.y. nesusiję su atnaujintos žiniatinklio IS versijos pakeitimais. Regresinių testų sumažinimo problema mūsų siūlomam įrankiui nėra itin svarbi, nes testavimo atvejus sudaro ar juos papildo bei atnaujina testuotojas. Priešingai nei vartotojo sesijos kaupimu paremtame metode, šio testų sudarymo metu perteklinių testavimo atvejų ženkliai nedaugėja.

3.9.3 Testų orakulas

Šiek tiek kitaip nei literatūroje [6, 30] ar praktikoje [26] siūlomi sprendimai, kurie remiasi `diff` principu, mūsų regresijos aptikimas remiasi iš serverio surinkto atsako kaip HTML žymų medžio struktūros analizavimu. Tikrinamas žymų korektiškumas, daromos išimtys dėl papildomų atributų, nekreipiama dėmesio į žymų susikeitimą vietomis tam tikrais atvejais ir į tarp žymų esantį tekstą. Kurdami darėme šias prielaidas:

I. HTML atsakas yra pilnai struktūrizuotas ir atitinka minimalius W3C reikalavimus. Toks analizavimo metodas reikalauja tikslesnių duomenų. Nors minimaliai struktūrizuotas HTML atsakas pasitaiko palyginus retai, tačiau toks atsakas turėtų būti visų informacinių sistemų programuotojų siekiamybė.
II. Grąžinamas HTML atsakas yra sudarytas iš vieno medžio: visas žymas apjungia viena šakninė (tėvinė) žyma. Jeigu atsakas susideda iš kelių nepriklausomų medžių (medžiai nėra apjungiami vienintele šaknimi), algoritmas žemiau esančių medžių netikrins.

3.10 Įrankio įdiegimas

Žiniatinklio IS apkrovos ir regresijos testavimo įrankis įdiegtas VĮ Registrų Centro Informacinių sistemų konstravimo departamento Kauno skyriaus programuotojų dirbančių su paslaugas administruojančia sistema kompiuteriuose (žiūrėti priedus „Įdiegimo aktas“). Įrankis buvo naudojamas testuoti informacinės sistemos apkrovos galimybes.

3.11 Išvados

1. Suprojektuotas, realizuotas ir ištestuotas žiniatinklio informacinių sistemų apkrovos ir regresijos testavimo įrankis.
2. Įrankis įdiegtas įmonėje VĮ Registrų Centras.
3. Sąvoka “IS našumo testavimas” gali būti išskaidyta į 5 tipus: atsako laiko(1), pralaidumo(2), apkrovos(3), streso (4) ir smaigo(5).
4. Tikrą vartotojų naršymą atspindintis apkrovos generavimas yra sudėtingesnis procesas, nei gali atrodyti.
5. Įgyvendinus HTML regresijos aptikimo algoritmą reikalingas tolesnis jo tobulinimas atsižvelgiant į semantines HTML savybes.

4 TYRIMO DALIS

Šiame skyriuje bus tiriami žiniatinklio IS regresinio testavimo orakulai ir semantinės HTML savybės, į kurias atsižvelgiant galima tobulinti testų orakulą. Pradžioje apibrėžiami tyrimo tikslai, nustatomi tyrimo uždaviniai. Detaliai aprašomos semantinės HTML savybės. Sukuriamas hipotetinis testų orakulas, kuris teisingai tikrina visas aprašytas savybes. Palyginami literatūroje aprašyti testų orakulai. Iškeliamos hipotezės ir pateikiamos tyrimo išvados.

4.1 Tyrimo tikslai ir uždaviniai

- Išnagrinėti HTML semantines savybes.
- Palyginti testų orakulus.
- Patobulinti ir įvertinti regresijos testų orakulą remiantis semantinėmis HTML savybėmis.
- Išmatuoti kiekvieno testų orakulo algoritmo tikslumą, aptikimą ir efektyvumą (2.4.3.1 Testų orakulų metrikos).

4.2 Tyrimo metodika

Paprastai lygindami testų orakulo pateiktus rezultatus negalėtume suprasti, kuris orakulas yra tikslesnis ir teisingai praneša apie atsiradusias klaidas. Todėl yra reikalinga įvesti vertinimo kriterijus. Kriterijai turėtų sietis su testų orakulo nagrinėjamais duomenimis. Šio tyrimo metu pagrindinis dėmesys bus skiriamas semantinėms HTML savybėms, kurios ir bus mūsų vertinimo kriterijais. Šios savybės leidžia sukonkretinti nagrinėjamus HTML dokumento aspektus ir objektyviau palyginti testų orakulus.

4.3 Mūsų testų orakulo algoritmo aprašas

Mūsų siūlomame žiniatinklio IS regresinio testavimo įrankyje RegReload regresijos aptikimo algoritmas veikia taip:

1 žingsnis. HTML atsako teksto eilutė transformuojama į DOM objektą panaudojant *Mozilla Firefox* XML/HTML analizatorių.

2 žingsnis. Iš DOM objekto sudaroma medžio struktūra, kurioje kiekvienas elementas turi tris pagrindinius parametrus:

a. HTML žymos pavadinimas
b. Žymos maksimalus gylis (kiek surastume vaikų kartų nagrinėdami šią žymą)
c. Vaikų kiekis

3 žingsnis. Naujai gautame HTML atsake surandami ieškomo atsako galimi tėviniai (šakniniai) elementai.

4 žingsnis. Rekursijos pagalba nagrinėjami medžiai ir lyginami požymiai (pavadinimas, gylis, vaikų kiekis).

1-me žingsnyje analizatorius patikrina HTML atsako struktūros taisyklingumą. Jei struktūra bloga, pranešama apie analizavimo klaidą ir algoritmas baigia darbą.

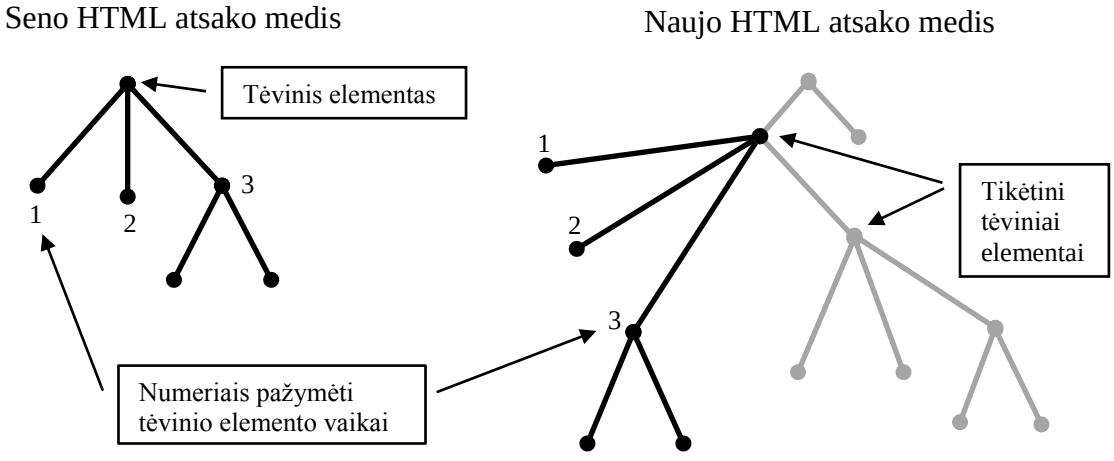
2-me žingsnyje iš DOM objekto į sudaromą naują medžio struktūrą nėra įtraukiami komentarai, skriptų ir kitos specifinės žymos, kurios neperteikia informacinės sistemos funkcionalumo.

3-me žingsnyje yra ieškoma *seno atsako* tėvinio elemento *naujame* HTML *atsake*. Toks veiksmas reikalingas dėl to, jog turi būti žinoma, nuo kurios naujo atsako vietos pradėti lyginti du HTML atsakus. Šiame žingsnyje gali būti surastas daugiau nei vienas tikėtinas tėvinis elementas. Jei nesurandamas nei vienas tėvinis elementas – daroma prielaida, jog naujas HTML atsakas pasikeitė neatpažįstamai, pranešama apie regresiją.

4-me žingsnyje senas atsakas lyginamas su nauju atsaku pradedant nuo surasto tikėtino tėvinio elemento. Žyma yra nagrinėjama taip:

- Tikrinamas vardas, gylis ir vaikų kiekis.
- Iš seno atsako imamas pirmas žymos vaikas ir pagal vardą ieškomas naujame atsake (19 pav.). Jeigu surandama žyma tokiu pat vardu, toliau rekursijos pagalba nagrinėjamos šios dvi žymos. Taip nagrinėjant žymos vaikus naujame atsake atsiradę vaikai yra praleidžiami ir lieka netikrinami.

Jeigu medžio elementų parametrai kurioje nors medžio šakoje nesutampa, imamas naujas tikėtinas tėvinis elementas. Veiksmai tęsiami tol, kol baigiasi tikėtinų tėvinių elementų sąrašas arba aptinkamas medžių sutapimas.



19 Paveikslėlis. Naujo ir seno HTML atsako medžiai

4.4 Semantinės HTML savybės

HTML dažnai nebūna gerai struktūrizuotas. Taip pat, kiekviena naršyklė gali atpažinti ir atvaizduoti žymas šiek tiek skirtingai. Dėl šių priežasčių, literatūroje [6, 30] yra siūlomos HTML (daugelis tinka ir XML) savybės, kurias testų orakulas turėtų tikrinti arba ignoruoti. Šiame skyriuje panagrinėsime semantines HTML savybes, kuriomis remiantis galima tobulinti testų orakulą: didinti jo *tikslumą* ir *aptikimą*. Didžioji dalis savybių yra aprašytos literatūroje, tačiau S15 ir S16 yra naujos, mūsų įvestos savybės. Savybių nagrinėjimas remiasi literatūroje aprašytais testų orakulų pasiekimais, HTML atsako dokumentų ir specifikacijų [34, 35] nagrinėjimu bei intuicija ir patirtimi.

Savybė 1. *Pabaigos žyma*

Literatūroje teigiama, kad naršyklei nėra svarbu, ar žymos turi pabaigą ir pateikiami patys paprasčiausi pavyzdžiai su `<p>` žyma. Tačiau kaip rodo patirtis, žymų pabaiga yra svarbi. Naršyklė nenuspręs, kur turi baigtis žyma, jei tai bus sudėtingas, su struktūra susijęs sprendimas, pvz. `<div>` ar `<span>` pabaigos žymų nebuvimas. Jeigu pabaigos žyma pradingo, tai reiškia, kad žiniatinklio IS tikrai egzistuoja klaida ir ji anksčiau ar vėliau išlįs. W3C griežtai rekomenduoja užbaigti žymas. Įvairūs HTML analizatoriai netgi neveikia, jeigu HTML yra blogai struktūrizuotas.

Savybė 2. *Išdėstymo žymų korektiškumas ir atributai*

Literatūroje teigiama, jog galime nekreipti dėmesio į tai, jog pavyzdžiui vietoj `<table>` žymos naujame HTML atsake atsiras `<span>` ar `<div>`, nes tai yra išsidėstymo žymos ir tokie pokyčiai funkcionalumo nekeičia. Tačiau iš kur galime žinoti, ar `<table>` žyma tikrai galėjo keistis į `<div>`. Ne visais atvejais tai gali būti leistinas atnaujinimas. Tokios žymos sudaro visą HTML atsako struktūrą todėl nekreipti į jas dėmesio negalime. Dažnai tokios žymos lemia ne tik išdėstymą, bet yra kaip ir konteineris, į kurį dinamiškai gali patekti nauja informacija (naudojant AJAX). Tačiau tokie žymų atributai kaip *align* ar *width* yra senos HTML versijos palikimas. Šiuo metu išdėstymo žymų konkretų išdėstymą lemia CSS.

Savybė 3. *stilius žymose ir stiliaus žymos*

Tokios žymos kaip `<b>`, `<i>` ar `<u>` keičia tik teksto išvaizdą, todėl jų tikrinti nereikia. Stiliaus atributai *style* ir *class* taip pat turėtų būti ignoruojami.

Savybė 4. *skriptų nuorodos atributuose ir žymose*

JavaScript, *Flash* ir kiti skriptai papildo kliento pusės interaktyvumą, tačiau tai daugiau siejasi su HTML dinamika ir vartotojui maloniu HTML atvaizdavimu realiu laiku, negu su duomenų ar HTML atsako korektiškumu. Mes nesiekiame testuoti vartotojo sąsają. Todėl žyma `<script>` su visu turiniu turėtų būti visiškai ignoruojama. Galėtų būti tikrinamas tik `<object>` žymos egzistavimas neatsižvelgiant į joje esančius atributus.

Savybė 5. *Paveiksliukų ir nuorodų rikiavimo tvarka*

Paveiksliukų ar nuorodų išsidėstymas HTML žymose dažnai nėra svarbus, ypač tada, kai puslapis formuojamas dinamiškai, pavyzdžiui susijusių atsitiktinių straipsnių nuorodos su paveikslėliais. Svarbiau yra nuorodų ir paveiksliukų skaičius, pvz. ar visos nuorodos buvo suformuotos ir pateiktos vartotojui. Taip pat, pasikeitusi paveiksliukų ar nuorodų tvarka gali lemti pataisytą funkcionalumą. Tik išimtiniais atvejais šių žymų tvarka yra svarbi.

Savybė 6. *Formos įvedimo laukų išsidėstymo tvarka*

Kaip ir S5, formos įvedimo laukų tvarka nėra svarbi. Svarbiau yra tai, ar visi laukai yra atvaizduojami. Tačiau formos laukai dažnai būna apipinti įvairiomis struktūros žymomis. Todėl nekreipimas į formos įvedimo laukų tvarką gali būti keblus.

Savybė 7. *Numatytosios parinktys formose*

Literatūroje teigiama, kad tai nėra aktualu. Pavyzdžiui atributo *value="..."* reikšmė nėra itin svarbi, nes tai labiau siejas su duomenimis ar tekstu, kuris gali kisti priklausomai nuo vartotojo profilio ar kitų faktorių. Tačiau vyriausybinėse, bankų sistemose būtų kritinė klaida, jeigu vartotojui būtų leidžiama įvesti į tam tikrą lauką duomenis ir juos išsiųsti į serverį. Vartotojas būtų klaidinamas arba iškreipiama neutrali nuomonė (pavyzdžiui apklausose).

Savybė 8. *Žymų inversija*

Nors literatūroje teigiama, kad tai nebus klaida, pvz.`<b><u>Žyma</u></b>` ir `<u><b>Žyma</b></u>`, tačiau atsižvelgiant į tai, jog pavyzdyje minimos žymos ir taip yra ignoruojamos (S3), visais likusiais atvejais žymų inversija gali lemti žymią regresiją pvz.: `<h1>` žymos ir `<li>` žymų inversija gali lemti blogą numeravimą.

Savybė 9. *Teksto tarp žymų pasikeitimas*

Vienose IS teksto tarp žymų pasikeitimas yra svarbus, kitose ne. Tačiau literatūroje teigiama, kad tekstas tarp žymų bent minimaliai turėtų būti tikrinamas (iš dalies tai atlieka S12). Jeigu tekstas tarp žymų yra konkretus, pvz.: ieškomas konkretus IS vartotojas ir tikimasi

gauti deterministinį rezultatą priklausomą nuo konkretaus testavimo atvejo. Jeigu grąžinamas blogas rezultatas, sistemoje turime klaidą. Toks pokytis turėtų būti tikrinamas. Tačiau sunkiausia yra nustatyti, kurių žymų tekstą reikėtų tikrinti. Tai priklauso ir nuo konkrečios IS, jos versijos ir nuo konkretaus testavimo atvejo. Reikalingas testuotojo įsikišimas.

Savybė 10. *Staigus teksto tarp žymų pagausėjimas*

Staigus teksto pagausėjimas gali lemti tai, kad galbūt serveris detaliai pranešė apie klaidą ar sistemos lūžį. Tačiau gali būti ir priešingai: IS paieškoje grąžinamas įrašas su detalesniu aprašymu. Nors tai įdomi savybė, tačiau ją pilnai padengia S12 savybė.

Savybė 11. *Vaikinių žymų rikiavimo tvarka*

Paprastai vaikinių žymų išsidėstymo tvarka nekeičia semantikos ir puslapio atvaizdavimo, todėl neturėtų būti tikrinama. Tai iš dalies siejasi su S5 ir S6 savybėmis. Skirtumas tas, jog S5 ir S6 savybėse paveikslėliai, nuorodos ar formos įvedimo laukai dažnai būna apipinti papildomomis struktūros žymomis. Tokių apipintų vaikinių elementų tarpusavio ryšį nustatyti yra sunkiau.

Savybė 12. *Raktažodžiai "exception", "error" ir kt.*

Naujoje versijoje tarp žymų atsiradę raktažodžiai *exception, error* ar kiti programavimo kalbose dažnai išspausdinami klaidų žodžiai gali lemti tai, kad įvyko klaida. Tai svarbi ir aiški savybė.

Savybė 13. *Atributų reikšmių pokyčiai*

Iš dalies tai nėra svarbu, nes atributo reikšmės pasikeitimas gali lemti atnaujintą arba naują funkcionalumą. Tačiau tokių atributų kaip *id, name, type, disabled* ar *colspans, rowspan* reikšmės yra svarbios ir gali lemti netinkamą informacinės sistemos veikimą.

Savybė 14. *Naujo atributo atsiradimas*

Toks naujo funkcionalumo ignoravimas sumažina klaidingai teigiamų rezultatų kiekį. Tai labai svarbi savybė, dėl kurios naujų atributų atsiradimas neturėtų būti tikrinamas.

Savybė 15. *Naujos žymos atsiradimas*

Jeigu nauja žyma nekeičia HTML medžio struktūros, o tik jį papildo, tuomet tai yra naujas funkcionalumas. Dauguma naujai atsirandančių žymų turėtų būti ignoruojamos.

Savybė 16. *HTML atsakas su keliais medžiais*

Grąžinamas HTML atsakas su keliomis nepriklausomomis žymomis: nėra vienos bendros šakninės žymos. Tai nėra dažnas reiškinys, bet įmanomas ir leistinas, ypač kai naudojama AJAX technologija. Testų orakulas turėtų nagrinėti visus atsake esančius iš žymų sudarytus medžius.

Savybė 17. *Minimalūs W3C reikalavimai*

HTML atsakas turi atitikti minimalius W3C keliamus XHTML reikalavimus, į kuriuos įeina žymų pradžios ir pabaigos vardų sutapimas, pabaigos žymų egzistavimas ir t.t. Tačiau rekomendacinio pobūdžio reikalavimai nėra būtini, pvz.: *id* atributo reikšmė turi prasidėti raide.

Ne visos savybės yra vienodai svarbios. Vienos atsikartoja rečiau (pvz.: S5, S6, S8, S11), kitos dažniau (S13, S14, S15). Tačiau kaip svarbiausias ir daugiausiai klaidingai teigiamų rezultatų kiekį sumažinančias savybes išskirčiau S14 ir S15. Kai kurios savybės yra trivialios (S5, S6 ir ypač S9), todėl turėtų būti tiriamos detaliau atsižvelgiant į konkrečias žiniatinklio IS. Tačiau mūsų tyrimas apsiriboja bendrinėmis, daugumai žiniatinklio IS būdingoms savybėms. Konkrečios atskiroms žiniatinklio IS būdingos klaidos nėra tiriamos. Susumavus rezultatus, 14 lentelėje matome, ar testų orakulas turėtų tikrinti konkrečios HTML savybės korektiškumą. Dėl aiškumo, tokį orakulą pavadinsime Hipotetiniu.

14 Lentelė. Hipotetinis orakulas: tikrinamos ir netikrinamos semantinės HTML savybės

Nr.	Pavadinimas	Ar Hipotetinis orakulas turėtų tikrinti šios savybės korektiškumą?
S1.	Pabaigos žyma	**Taip**
S2.	Išdėstymo žymų korektiškumas ir atributai	**Taip**, iš dalies
S3.	Stilius žymose ir stiliaus žymos	Ne
S4.	Skriptų nuorodos atributuose ir žymose	Ne
S5.	Paveiksliukų ir nuorodų rikiavimo tvarka	Ne
S6.	Formos įvedimo laukų išsidėstymo tvarka	Ne
S7.	Numatytosios parinktys formose	**Taip**, iš dalies
S8.	Žymų inversija	**Taip**
S9.	Teksto tarp žymų pasikeitimas	**Taip**, iš dalies
S10.	Staigus teksto tarp žymų pagausėjimas	Ne
S11.	Vaikinių žymų rikiavimo tvarka	Ne
S12.	Raktažodžiai "exception", "error" ir kt.	**Taip**
S13.	Atributų reikšmių pokyčiai	**Taip**, iš dalies
S14.	Naujo atributo atsiradimas	Ne
S15.	Naujos žymos atsiradimas	Ne
S16.	HTML atsakas su keliais medžiais	Ne
S17.	Minimalūs W3C reikalavimai	**Taip**

4.5 Testų orakulų tyrimas ir tobulinimas

Sprenkle su kolegėmis detaliai aprašė ir ištestavo 22 HTML testų orakulus bei jų kombinacijas [29, 30] ir nustatė, jog efektyviausias ir universaliausias yra **N+I** ∪ **C-WD** testų orakulų sąjunga (žiūrėti 2.4.3.2 Žiniatinklio IS orakulų tipai), kuris tikrina HTML žymų struktūrą, jų vardus, taip pat atsižvelgia į tarp žymų esantį turinį nepaisydamas tuščios erdvės ir datos ar laiko žymų. Dobolyi su kolega [6] įgyvendintas XML/HTML testų orakulas analizuoja panašias sritis, kaip ir Sprenkle siūlomas orakulas, tik dėl savybės sulyginti XML failus, yra bendresnis. Dalis autorių orakulų yra įgyvendinti atsižvelgiant į semantines HTML savybes. Tačiau Sprenkle siūlo ir specializuotus orakulus, kurie pritaikyti konkrečiai žiniatinklio IS.

Mūsų siūlomame įrankyje testų orakulas įgyvendintas remiantis šių autorių pasiekimais ir HTML semantika. Pagal Sprenkle pasiūlytą testų orakulų hierarchiją, mūsų siūlomas testų orakulas atitiktų tokį trumpą apibrėžimą: **N-S+F+U**. Mūsų orakule yra lyginama žymų struktūra ir jų pavadinimai nekreipiant dėmesio į stilių. Papildomai orakulas nekreipia dėmesio į HTML žymų ir žymų atributų atsiradimą naujoje žiniatinklio IS versijoje – leidžiamas nemenkas senos HTML žymų struktūros apaugimas naujomis žymomis. Tai stipriai pranoksta **N-S+F+U** orakulą.

Mūsų testų orakulas apima šias savybes, kurias tikrindamas elgiasi taip, kaip nurodyta 14 lentelėje (56 psl.): S1-S6, S8, S10-S11, S15, S17. Savybės S7, S9, S12, S13, S14 lieka ignoruojamos, todėl atsiranda galimybė papildyti mūsų testų orakulo algoritmą šių savybių tikrinimu.

Atsižvelgdami į semantines HTML savybes, patobulinome RegReload įrankio testų orakulą. Kai kurių savybių įgyvendinimas pablogina testų orakulo veikimą, pavyzdžiui pridėjus atributų tikrinimą yra tikrinami visi atributai (S13). Todėl buvo reikalinga įtraukti išimtis, kad būtų tenkinamos likusios savybės (S3). Atlikti pakeitimai yra tokie:

- Pridėtas žymų atributų tikrinimas (S7, S13 ir S14).
- Pridėtos atributų išimtys: *style*, *class*, *align*, *width*, *height* (S3).
- Pridėtas svarbių atributų reikšmių tikrinimas: *id*, *name*, *colspan*, *span*, *colspans*, *type*, *href*, *form*, *action*, *rel*, *label*, *disabled*, *required*, *readonly*, *checked*.
- Pridėta klaidų raktažodžių paieška tekste. Ieškoma šių žodžių ir jų junginių: *error*, *exception*, *undefined method*, *no such file or directory*.
- Papildomose dvejose versijose pridėtas teksto tarp žymų tiesioginis lyginimas ir teksto tekste ieškojimas (S9).

S5, S6 ir S11 savybių įgyvendinimas iš programinės pusės prieštarauja S13 ir S14 savybių įgyvendinimui. Nes vienu metu ir tikrinti žymų atributus bei jų reikšmes, ir kartu atsiriboti nuo žymų tvarkos (atsiribojimui reikia netikrinti atributų ir jų reikšmių) yra sudėtinga. Atsižvelgiant į tai, jog S13 ir S14 savybės yra svarbesnės, dažniau pasitaikančios realiose situacijose, tobulinant testų orakulą šioms savybėms buvo suteikta pirmenybė. Tačiau savybių suderinamumo problema nėra neišsprendžiama.

4.5.1 Tyrimo rezultatai

15 lentelėje pateikiami literatūroje aprašyti orakulai ir mūsų siūlomas orakulas prieš patobulinimus ir po patobulinimų. Ženklas "–" reiškia, kad orakulas ignoruoja savybę ir jos netikrina. Ženklas "✚" reiškia, kad orakulas tikrina savybę iš dalies arba taip pat, kaip ir hipotetinis orakulas.

15 Lentelė. Testų orakulų palyginimas atsižvelgiant į HTML savybes

HTML savybė	**Hipotetinis orakulas**	**Sprenkle geriausias orakulas**	**Dobolyi orakulas**	**RegReload orakulas**			
				prieš patobulinimus	**patobulinimų kartos**		
					1-a	**2-a[2]**	**3-a[3]**
S1.	**Taip**	–	–	✚	✚	✚	✚
S2.	**Taip**, iš dalies	✚	✚	✚	✚	✚	✚
S3.	Ne	✚	–	✚	✚	✚	✚
S4.	Ne	✚	–	✚	✚	✚	✚
S5.	Ne	✚	–	✚	–	–	–
S6.	Ne	✚	–	✚	–	–	–
S7.	**Taip**, iš dalies	✚	–	–	✚	✚	✚
S8.	**Taip**	✚	–	✚	✚	✚	✚
S9.	**Taip**, iš dalies	✚	–	–	–	✚	✚
S10.	Ne	–	–	✚	✚	✚	✚
S11.	Ne	–	–	✚	–	–	–
S12.	**Taip**	–	✚	–	✚	✚	✚
S13.	**Taip**, iš dalies	–	–	–	✚	✚	✚
S14.	Ne	–	✚	–	✚	✚	✚
S15.	Ne	–	–	✚	✚	✚	✚
S16.	Ne	✚	✚	–	–	–	–
S17.	**Taip**	–	–	✚	✚	✚	✚
Teisingai tikrinamos savybės:		9	4	11	12	13	13

Dobolyi orakulas teisingai tikrina mažiau savybių, tačiau jis yra kuriamas ne tik HTML, bet ir XML lyginimui. Todėl specifinės HTML savybės nėra analizuojamos.

[2] S9 savybė tenkinama tiesiogiai lyginant tekstą tarp žymų.
[3] S9 savybė tenkinama ieškant seno HTML atsako teksto naujame HTML atsako tekste. Taip įgyvendintas tikrinimas leidžia naujo teksto atsiradimą – mažina klaidingai teigiamus orakulo rezultatus.

4.5.2 Hipotezės

Iš atlikto testų orakulų tyrimo ir palyginimo rezultatų (15 lentelė), galime teigti, jog Sprenkle ir Dobolyi siūlomi testų orakulai tinkamai tikrina mažiau semantinių HTML savybių – atitinkamai 9 ir 4, todėl šių orakulų tikslumas ir aptikimas turėtų būti mažesnis lyginant su mūsų testų orakulu. RegReload prieš ir po patobulinimo teisingai tikrina beveik po vienodą HTML savybių skaičių – 11 prieš ir 12-13 po patobulinimų. Gali atrodyti, jog naujasis orakulas beveik niekuo nepatobulėjo. Tačiau ne visos savybės yra lygiai atsikartojančios. Apibendrinant galime iškelti tokią hipotezę:

Hipotezė 1: Patobulintas pirmos kartos RegReload testų orakulas turi didesnį tikslumą ir didesnį klaidų aptikimą negu prieš patobulinimus.

Hipotezė 2: Patobulintas pirmos kartos testų orakulas turi didžiausią efektyvumą.

Hipotezė 3: Antros kartos testų orakulas turi mažesnį tikslumą, tačiau didesnį aptikimą lyginant su pirmuoju patobulinimu.

Hipotezė 4: Trečios kartos testų orakulas turi didesnį tikslumą nei antros kartos testų orakulas.

Objektyviai palyginti Sprenkle ir Dobolyi testų orakulų su mūsų siūlomu orakulu nėra galimybių, nes šie orakulai įgyvendinti remiantis DIFFX algoritmu (Dobolyi) ir Perl kalbos HTML analizatoriumi. Net jeigu pakeistume savo orakulą atsižvelgiant į HTML savybes, vis tiek nepasiektume tokio paties sprendimo, kokį siūlo autoriai. Todėl eksperimento metu RegReload testų orakulas bus lyginamas su TestComplete (2.5 Žiniatinklio IS regresinio testavimo įrankiai) įrankio testų orakulu.

4.6 Išvados

Šio tyrimo metu:

1. Išgrynintos ir apibrėžtos semantinės HTML savybės. Nors tai nėra galutinis sąrašas, jis galėtų būti tobulinamas, tačiau pagrindinės ir būdingos visoms žiniatinklio IS HTML semantinės savybės buvo aptartos.
2. Pagal semantines HTML savybes palyginti literatūroje aprašomi testų orakulai su mūsų siūlomomis testų orakulo modifikacijomis.
3. Atsižvelgiant į patirtį ir dažniausiai atsirandančius leistinus pakitimus HTML atsake, kaip svarbiausias išskirčiau S14 ir S15 HTML semantines savybes.
4. Retai pasitaikančios ir mažai reikšmingos būtų S5, S6, S8 ir S11 savybės, nes paveiksliukų ar formos laukų rikiavimo tvarka keičiasi retai. Greičiau žyma gali pradingti ar atsirasti nauja, nei keisis jos vieta tame pačiame hierarchijos lygyje.
5. S9 savybė negali būti tikrinama visada, nes tai priklauso nuo konkrečių žymų ir konkrečios žiniatinklio IS. Kad ši savybė būtų tikrinama efektyviai, testavimo atvejų sudarymo metu reikėtų nurodyti, tarp kurių žymų ir kaip reikėtų lyginti tekstą.
6. Iškeltos hipotezės, kurias patvirtinsime arba paneigsime atlikę eksperimentą.

5 EKSPERIMENTINĖ DALIS

Šiame skyrelyje eksperimentiškai palyginsime du testų orakulus ir skirtingas jų versijas, pateiksime eksperimento metu surinktų duomenų analizę. Patvirtinsime arba paneigsime tyrimo dalyje iškeltas hipotezes.

5.1 Uždaviniai

- Atsižvelgiant į semantines HTML savybes, testuojamoje IS sukurti mutuotus HTML dokumentus, kurie atspindėtų naujoje žiniatinklio IS versijoje atsirandančius pokyčius.
- Įvykdyti testus su kiekvienu įrankiu ir jo versijomis.
- Gautus rezultatus išanalizuoti ir patvirtinti arba paneigti iškeltas hipotezes.

5.2 Testavimo objektas

Testavimo objektu buvo pasirinkta žiniatinklio informacinė sistema „IT Administratorių portalas“, kuri buvo sukurta 4-tame kurse kaip kursinis darbas. Šioje IS vartotojai registruoja kompiuterinius gedimus, o sistema šiuos gedimus paskirsto registruotiems IT specialistams. Kaupiama ištaisytų klaidų statistika. Ši žiniatinklio IS įgyvendinta AJAX technologijos pagrindu.

5.3 Mutuotų HTML dokumentų kūrimas

Iš testuojamos žiniatinklio IS buvo išsaugoti registracijos, vartotojo profilio ir statistikos HTML dokumentai. Sistema buvo kurta palyginti seniai ir nesilaikant griežtų HTML struktūros reikalavimų, todėl kiekvienas (išskyrus vieną) HTML atsakas buvo pataisytas, kad atitiktų minimalius W3C reikalavimus. Kitu atveju dėl RegReload įrankyje naudojamo griežto HTML analizatoriaus nebūtų įmanoma analizuoti HTML dokumentų. Pataisytuose HTML dokumentuose buvo sukurti pakitimai. Viename dokumente buvo sukurta tik po vieną pakitimą, kuris atitiko vieną iš apibrėžtų HTML semantinių savybių. Buvo sukurta ir tokių HTML dokumentų, kurie atitinka tą pačią savybę, tik iš skirtingų pozicijų, pavyzdžiui: S14 teigia apie naujo atributo atsiradimą. Buvo sukurti du HTML dokumentai. Viename buvo pridėta naujų atributų, kitame pašalintas atributas. Testuodami tokias žiniatinklio IS versijas galime iš tiesų įsitikinti, ar orakulas tikrina atributų pokyčius. Pakeisti HTML dokumentai buvo įkelti į IS.

5.4 Eksperimento eiga

Eksperimento metu buvo išbandyti du TestComplete regresijos aptikimo būdai ir keturios RegReload versijos. Pirmas TestComplete būdas tikrina HTML žymų struktūrą ir atributus, antras būdas tikrina visą puslapio HTML atsaką kaip tekstą. TestComplete tinkamai

veikė tik su *Internet Explorer* naršykle. Taip pat užtrukdavo gerokai ilgiau, nes visas testavimo atvejo vykdymas siejosi su vartotojo sąsaja naršyklėje. Taip pat šis įrankis užtrukdavo gerokai ilgiau tikrindamas klaidingą HTML dokumentą, nei teisingą. Galbūt ilgą tikrinimą lėmė tai, jog papildomai kelis kartus būdavo kreipiamasi į žiniatinklio IS serverį pasitikslinti HTML atsako. RegReload testavimo atvejai būdavo įvykdomi akimirksniu. Eksperimento metu visi testavimo atvejai buvo įvykdyti sėkmingai tiek su TestComplete, tiek su RegReload.

5.5 Rezultatai ir jų analizė

16 lentelėje pateikiami eksperimento metu surinkti duomenys. Stulpelyje „Laukiamas rezultatas“ ženklu „✓“ pažymėta teisingą rezultatą grąžinanti žiniatinklio IS versija. Ženklu „×“ pažymėta klaidingą rezultatą grąžinanti IS versija. Jeigu visi įrankio rezultatai sutampa su tikėtinais, galime teigti, kad įrankis buvo 100% tikslus ir aptiko visas klaidas. Tačiau tokių rezultatų nebuvo. Įdomus pastebėjimas tas, jog RegReload po 1-os kartos patobulinimo, nors ir netikrina S11 savybės, tačiau grąžino teisingą pranešimą, t.y. žiniatinklio IS pakitimas buvo identifikuotas kaip leistinas. Tokį testų orakulo klaidų aptikimo algoritmo veikimą galima paaiškinti tuoj, jog HTML atsakas yra nagrinėjamas kaip medžio struktūra. S11 teigia apie vaikinių žymų rikiavimo tvarkos nepaisymą. Kaip minėta 4.4 Mūsų testų orakulo algoritmo aprašas skyrelyje, algoritmas nagrinėdamas vaikines žymas nepaiso jų išsidėstymo tvarkos, todėl ir testo rezultatas buvo teisingas. Tačiau sudėtingesniu atveju (su daugiau atributų), algoritmas vis tiek aptiktų klaidą, nors ji būtų leistina. Šioje lentelėje kai kurie pakitimai yra suskaidyti į 2 ar daugiau dalių. Buvo sukurta ir eksperimento metu panaudota daugiau mutuotų HTML dokumentų, kad būtų įsitikinta, jog testų orakulas išties atsižvelgia į konkrečią savybę.

16 Lentelė. Eksperimento metu praleistos ir identifikuotos klaidos

Pakitimas[4]		**Laukiamas rezultatas**	**TestComplete**		**RegReload**			
			Žymų struktūros tikrinimas	Viso puslapio turinio tikrinimas	Prieš patobulinimus	patobulinimų kartos		
						1-a	2-a	3-a
1.		×	×	×	×	×	×	×
2.		×	×	×	×	×	×	×
3.		✓	✓	×	✓	✓	✓	✓
4.		✓	✓	×	✓	✓	✓	✓
5.		✓	✓	×	✓	✓	×	×
6.	1	✓	×	×	✓	✓	✓	✓
	2	✓	✓	×	✓	×	×	×
7.	1	✓	✓	×	✓	✓	✓	✓
	2	×	✓	×	✓	×	×	×
8.		×	×	×	×	×	×	×
9.	1	✓	✓	×	✓	✓	×	✓
	2	×	✓	×	✓	✓	×	×
10.		✓	✓	×	✓	✓	×	×
11.		✓	✓	×	✓	✓	×	×
12.		×	✓	×	✓	×	×	×
13.	1	✓	✓	×	✓	✓	✓	✓
	2	×	✓	×	✓	×	×	×
14.	1	✓	✓	×	✓	✓	✓	✓
	2	×	×	×	×	×	×	×
15.	1	✓	×	×	✓	✓	✓	✓
	2	✓	×	×	✓	✓	✓	✓
	3	×	×	×	×	×	×	×
	4	✓	×	×	×	×	×	×
16.		×	×	×	✓	✓	✓	✓
17.		×	✓	×	×	×	×	×

17 lentelėje iš eksperimento rezultatų išrinkti ir apskaičiuoti duomenys. Pirmose 5 duomenų eilutėse yra pateikiamas klaidų kiekis vienetais. Paskutinėse eilutėse tikslumas, aptikimas ir efektyvumas pateikiamas procentiniu dydžiu.

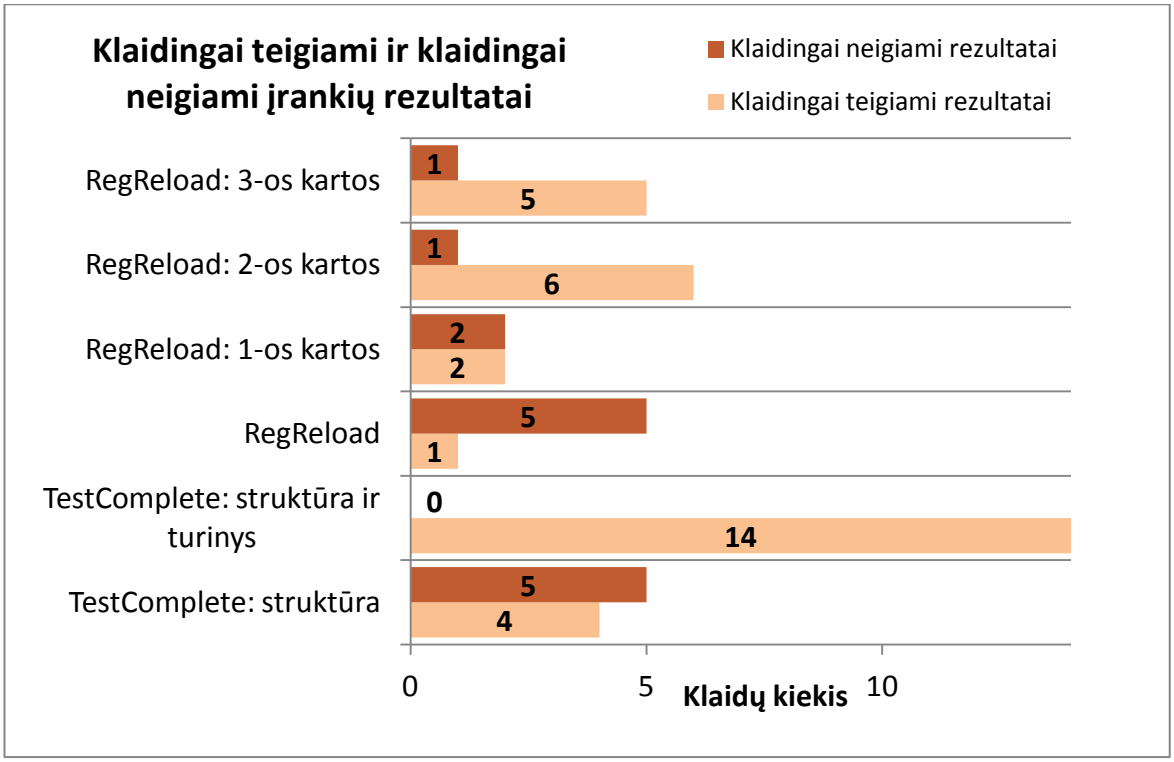
[4] Žiniatinklio IS pakitimas atitinkantis konkrečią HTML savybę.

17 Lentelė. Iš eksperimento rezultatų apskaičiuoti duomenys

	TestComplete		**RegReload**			
	Žymų struktūros tikrinimas	Struktūra ir turinys	Prieš patobulinimus	patobulinimų kartos		
				1-a	2-a[5]	3-a[6]
Klaidingai teigiami rezultatai	4	14	1	2	6	5
Klaidingai neigiami rezultatai	5	0	5	2	1	1
Teisingai identifikuotos klaidos	6	11	6	8	10	10
Visos identifikuotos klaidos	10	25	7	10	16	15
Tikėtinos klaidos	11	11	11	11	11	11
Tikslumas	60,00%	44,00%	85,71%	80,00%	62,50%	66,67%
Aptikimas	54,55%	100,00%	54,55%	72,73%	90,91%	90,91%
Efektyvumas	57,14%	61,11%	66,67%	76,19%	74,07%	76,92%

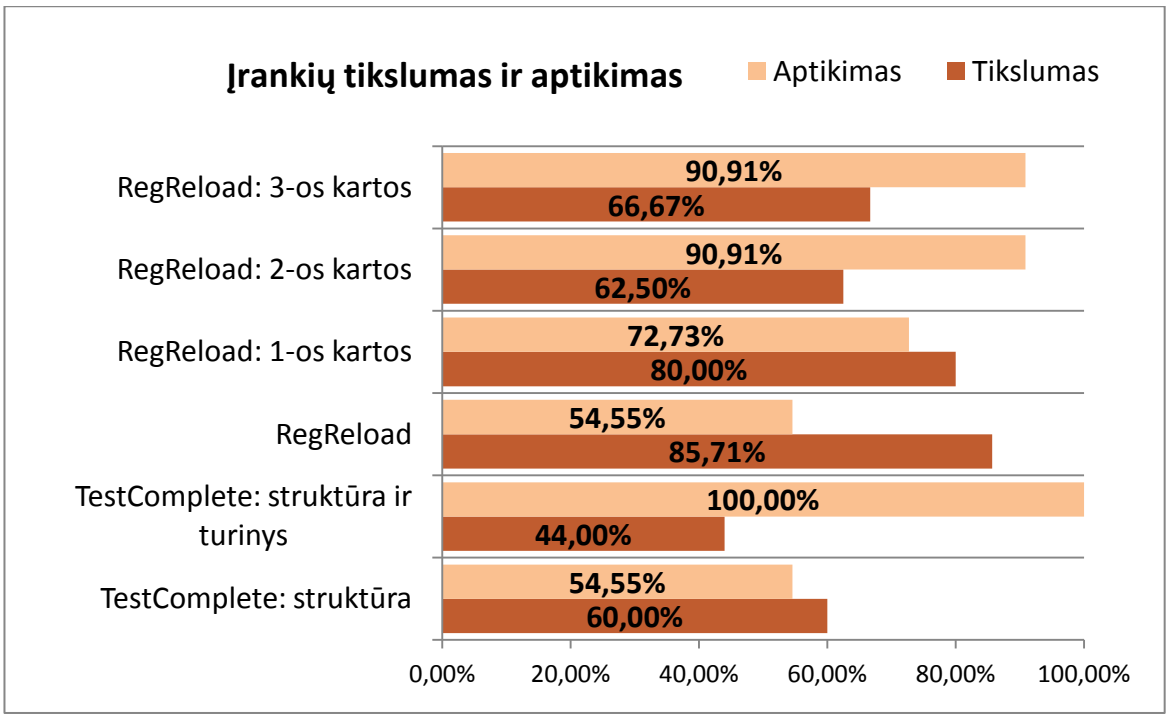
20 paveikslėlyje atvaizduoti kiekvieno įrankio pranešti klaidingai teigiami ir neigiami rezultatai. Kaip matyti iš diagramos, daugiausiai klaidingai teigiamų rezultatų pateikė TestComplete įrankio viso puslapio nagrinėjimas (14). Bet koks pakitimas HTML atsake buvo identifikuotas kaip klaida. RegReload po 1-mos kartos patobulinimo klaidingai neigiamų rezultatų kiekį sumažino daugiau nei dvigubai, iki 2. Dvigubai išaugo klaidingai teigiamų rezultatų kiekis, iki 2. Iš testuotojo ir programuotojo pusės tai yra tinkamesnis testų orakulas, nes aptinkama daugiau klaidų, o klaidingai teigiamų rezultatų kiekis išauga nežymiai. 2-os ir 3-os kartos patobulinimai RegReload testų orakule dar sumažino klaidingai neigiamų rezultatų kiekį, iki 1, tačiau stipriai išaugo klaidingai teigiami pranešimai (atitinkamai 6 ir 5). TestComplete struktūros tikrinimas praleido tiek pat klaidų, kiek ir RegReload be patobulinimų, tačiau pranešė daugiau klaidingai teigiamų rezultatų (4 lyginant su 1).

[5] Kaip ir 1-mos kartos, tik papildomai tiesiogiai lyginamas tekstas tarp žymų.
[6] Kaip ir 1-mos kartos, tik sename atsake esančio teksto tarp žymų yra ieškoma naujame atsake.



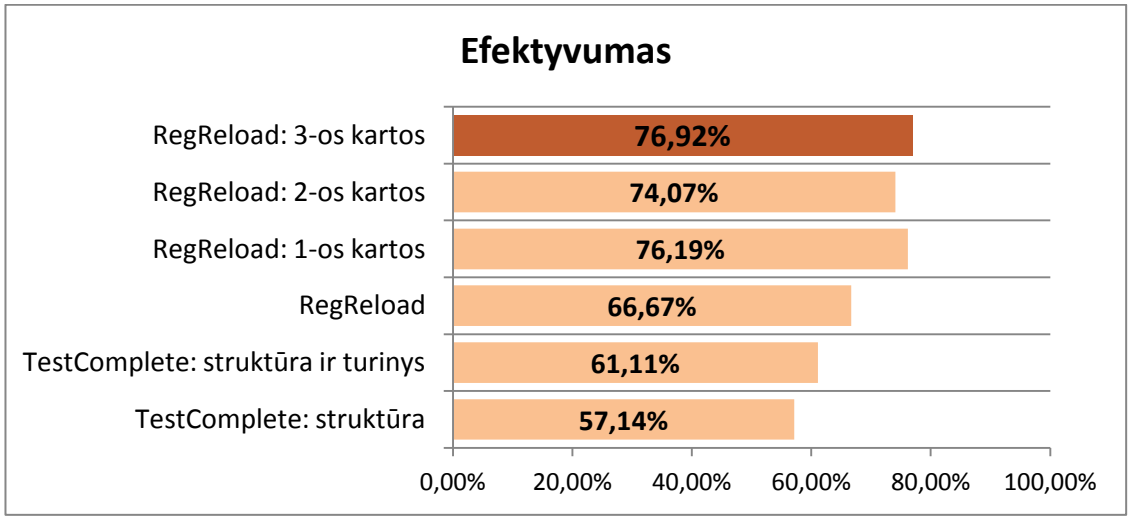
20 Paveikslėlis. Kiekvieno įrankio klaidingai teigiami ir neigiami rezultatai

21 paveikslėlyje pateikiamas įrankių tikslumas ir klaidų aptikimas. Kaip matyti iš rezultatų, po RegReload 1-mos kartos patobulinimo nors tikslumas ir sumažėjo per 5.71% (iki 80,00%), tačiau klaidų aptikimas išaugo 1.33 karto. 2-os ir 3-os kartos RegReload patobulinimai stipriai padidino klaidų aptikimą, iki 91.91%. Tačiau dar labiau sumažėjo tikslumas, kuris artėjo prie TestComplete. TestComplete viso puslapio turinio tikrinimas atskleisdavo 100% klaidų, tačiau 56% aptiktų klaidų buvo netikros. TestComplete žymų struktūros tikrinimas parodė neblogus rezultatus. Kaip ir RegReload, klaidų aptikimas buvo 54.55%. Tačiau tikslumas buvo mažesnis (85,71% lyginant su 60.00%), kas lėmė klaidingai teigiamų rezultatų padidėjimą lyginant su RegReload. ***Hipotezė 1*** pasitvirtino tik iš dalies, nes tikslumas ne padidėjo, bet sumažėjo. ***Hipotezė 3*** pilnai patvirtinta. Antros kartos RegReload turi didesnį aptikimą ir mažesnį tikslumą lyginant su pirma karta. ***Hipotezė 4*** pasitvirtino, trečios kartos RegReload turi didesnį tikslumą.



21 Paveikslėlis. Kiekvieno įrankio tikslumas ir klaidų aptikimas

22 paveikslėlyje pavaizduotas visų eksperimente ištirtų įrankių testų orakulų efektyvumas. Efektyvumas buvo skaičiuojamas pagal harmoninio vidurkio formulę. Didžiausią efektyvumą turėjo RegReload po 3-os kartos patobulinimų (76.92%). Nuo šio rezultato tik per kelias dešimtąsias atsiliko RegReload po 1-os kartos patobulinimų (76.19%). ***Hipotezės 2*** šiuo atveju negalime patvirtinti, nes RegReload po 1-os kartos patobulinimų efektyvumas nežymiai nusileidžia 3-os kartos patobulinimams.



22 Paveikslėlis. Įrankių ir jų versijų testų orakulų efektyvumas.

Tarp žymų esančio teksto lyginimas padidino testų orakulo aptikimą ir efektyvumą. Tačiau ar teksto pokyčiai yra svarbūs, priklauso nuo konkrečios žiniatinklio IS ir netgi nuo

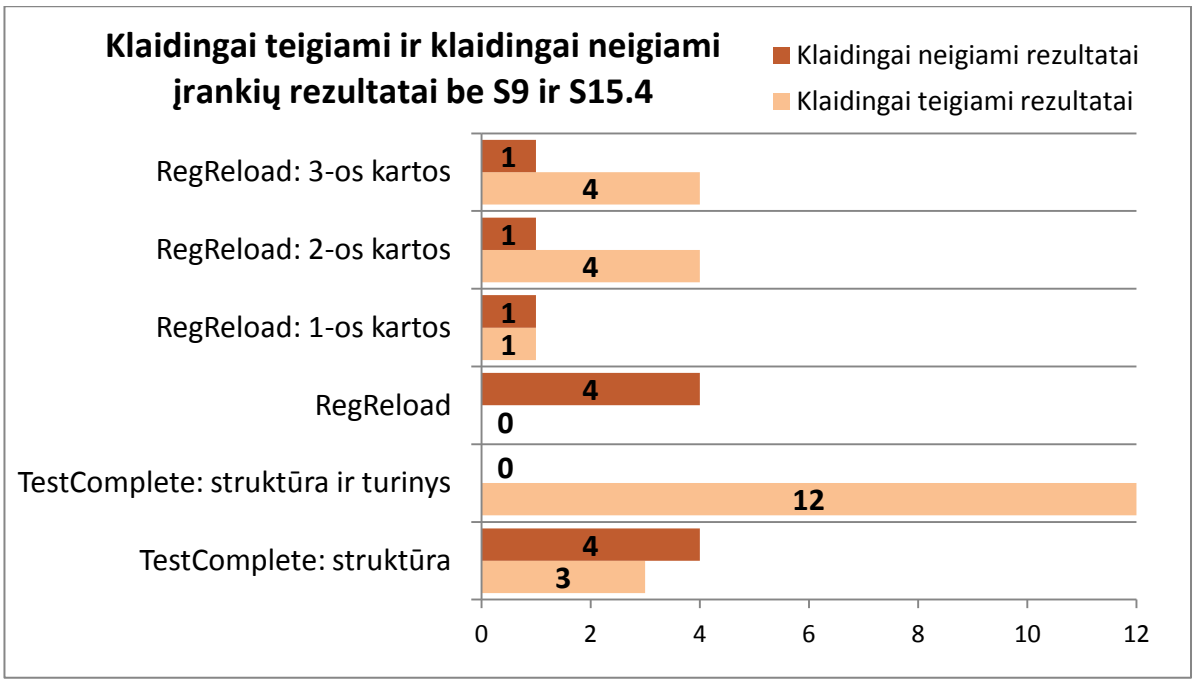
konkretaus testavimo atvejo. Dažniausiai tik tarp konkrečių žymų esančių tikslių duomenų, tokių kaip gautas ataskaitų kiekis, asmens pavardė ir vardas arba asmens kodas, pokyčiai galėtų būti interpretuojami kaip klaida. Panagrinėję atidžiau 16 lentelę (63 psl.) su eksperimento metu surinktais duomenimis, nusprendėme nekreipti dėmesio, ar S9 HTML savybė yra tikrinama. Šių eksperimento duomenų atmetimas leidžia nagrinėti testų orakulus universalumo prasme, nes S9 savybė turėtų būti tikrinama tik tam tikrose žiniatinklio IS. Taip pat atmetėme S15 savybės ketvirtą rezultatą. Šioje žiniatinklio IS versijoje HTML struktūra buvo pakeista taip (žiūrėti priedus, 26 pav.), jog visi testų orakulai pranešė apie klaidą. Todėl šių duomenų atmetimas neiškraipo testų orakulų parametrų. 18 lentelėje susumuoti eksperimento rezultatai neįskaičiuojant S9 ir S5.14 savybių.

18 Lentelė. Iš eksperimento rezultatų apskaičiuoti duomenys neįskaičiuojant S9 ir S15.4

	TestComplete		**RegReload**			
	Žymų struktūros tikrinimas	Struktūra ir turinys	Prieš patobulinimus	patobulinimų kartos		
				1-a	2-a[7]	3-a[8]
Klaidingai teigiami rezultatai	3	12	0	1	4	4
Klaidingai neigiami rezultatai	4	0	4	1	1	1
Teisingai identifikuotos klaidos	6	10	6	9	9	9
Visos identifikuotos klaidos	9	22	6	10	13	13
Tikėtinos klaidos	10	10	10	10	10	10
Tikslumas	66,67%	45,45%	100,00%	90,00%	69,23%	69,23%
Aptikimas	60,00%	100,00%	60,00%	90,00%	90,00%	90,00%
Efektyvumas	63,16%	62,50%	75,00%	90,00%	78,26%	78,26%

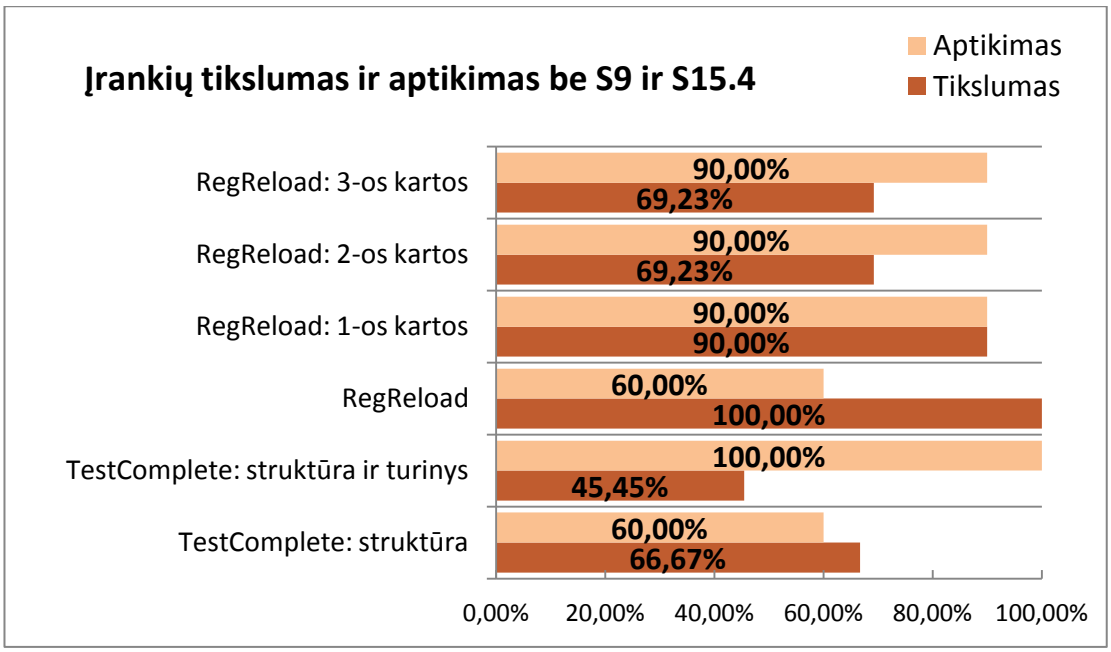
23 paveikslėlyje pavaizduoti klaidingai teigiami ir klaidingai neigiami įrankių rezultatai po minėtų eksperimento duomenų atmetimo. Klaidingai neigiami rezultatai sumažėjo TestComplete ir RegReload prieš patobulinimus ir po pirmos kartos patobulinimų. Visų testų orakulų klaidingai teigiamų rezultatų kiekis sumažėjo vienetu arba dviem.

[7] Kaip ir 1-os kartos, tik papildomai tiesiogiai lyginamas tekstas tarp žymų.
[8] Kaip ir 1-os kartos, tik sename atsake esančio teksto tarp žymų yra ieškoma naujame atsake.



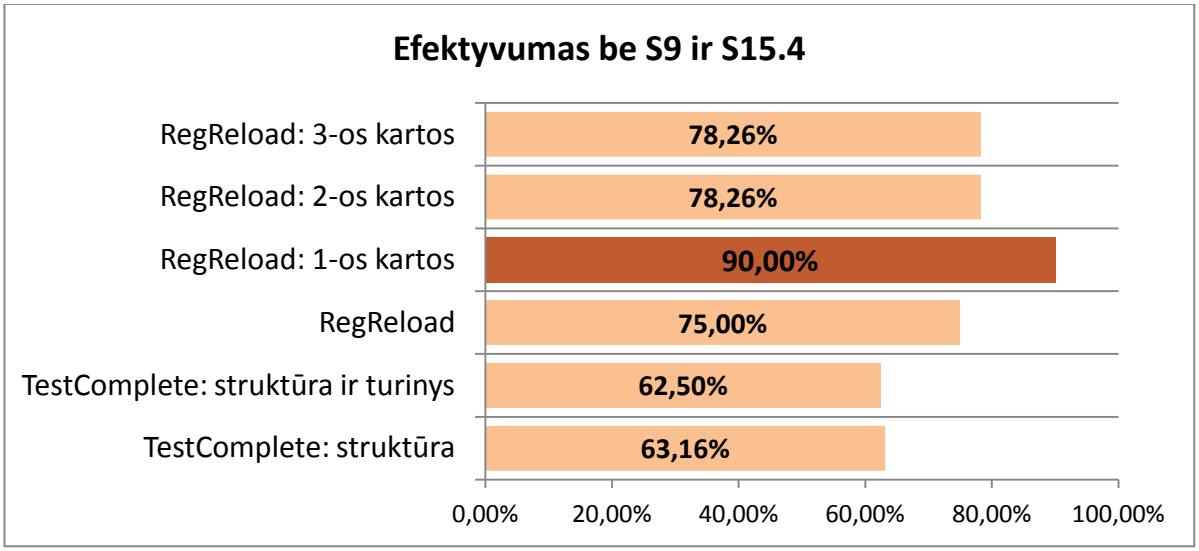
23 Paveikslėlis. Įrankių klaidingai teigiami ir neigiami rezultatai atmetus S9 ir S15.4 savybių tikrinimą.

24 paveikslėlyje pavaizduoti įrankiu testų orakulų tikslumas ir aptikimas. Įdomu tai, jog atmetus minėtus eksperimento duomenis, RegReload testų orakulas prieš patobulinimus turi net 100% tikslumą. Tai reiškia, jog šio testų orakulo pranešta klaida yra tikra. RegReload po 1-mos kartos patobulinimų šiuo atveju rodo puikų rezultatą. Tiek aptikimas, tiek tikslumas yra lygūs 90.00%. 2-os ir 3-os kartos RegReload gerokai nusileidžia tikslumu. TestComplete visais kriterijais nusileidžia mūsų RegReload testų orakului. ***Hipotezė 1*** kaip ir anksčiau pasitvirtino tik iš dalies, nes tikslumas sumažėjo. ***Hipotezė 3*** pasitvirtina tik iš dalies. Antros kartos RegReload turi mažesnį tikslumą lyginant su pirma karta, tačiau aptikimas sutampa. ***Hipotezė 4*** šiuo atveju netenka prasmės, nes antros ir trečios kartos testų orakulai specializuoti tikrinti tekstą, todėl patvirtinti negalime.



24 Paveikslėlis. Įrankių tikslumas ir klaidų aptikimas be S9 ir S15.4 savybių tikrinimo.

25 paveikslėlyje pavaizduotas įrankių testų orakulų efektyvumas. Didžiausias efektyvumas, kaip ir buvo galima tikėtis, buvo 1-os kartos RegReload testų orakulo, kuris padidėjo per 13.81%. ***Hipotezė 2*** pilnai patvirtinta.



25 Paveikslėlis. Įrankių ir jų versijų testų orakulų efektyvumas.

5.6 Išvados

Šio eksperimento metu:

1. Ištestuotos skirtingos žiniatinklio IS versijos, surinkti TestComplete ir RegReload testų orakulų rezultatai.
2. Apskaičiuotas testų orakulų tikslumas, aptikimas ir efektyvumas. Geriausi testų orakulai ir jų efektyvumai:
    a. Nagrinėjant visus eksperimento duomenis efektyviausias buvo 3-os kartos RegReload testų orakulas 76.92%. Tačiau nuo jo tik keliomis dešimtosiomis atsiliko 1-os kartos RegReload testų orakulas (76.19%).
    b. Atmetus S9 (teksto tarp žymų tikrinimas) ir S15 (stipriai pakitusi struktūra) eksperimento metu surinktus duomenis, efektyviausias buvo 1-os kartos RegReload testų orakulas (90%) ir nuo kitų orakulų pirmavo 13.81% ir daugiau.
3. Tyrime iškeltų hipotezių patvirtinimai ir paneigimai
    a. Nagrinėjant visus eksperimento duomenis:
    Hipotezė 1: Patvirtinta iš dalies.
    Hipotezė 2: Nepatvirtinta.
    Hipotezė 3: Pilnai patvirtinta.
    Hipotezė 4: Pilnai patvirtinta.
    b. Atmetus S9 (teksto tarp žymų tikrinimas) ir S15.4 (stipriai pakitusi struktūra) eksperimento metu surinktus duomenis:
    Hipotezė 1: Patvirtinta iš dalies.
    Hipotezė 2: Pilnai patvirtinta.
    Hipotezė 3: Patvirtinta iš dalies.
    Hipotezė 4: Nepatvirtinta.
4. Testų orakulo efektyvumas labai priklauso nuo tikrinamų HTML semantinių savybių interpretavimo. Jau nekalbant apie skirtingas žiniatinklio informacines sistemas, netgi toje pačioje testuojamoje žiniatinklio IS su tuo pačiu testavimo atveju vienoje versijoje vaikinių žymų rikiavimo tvarka (S11 savybė) gali būti svarbi, kitoje ne. Tas pats yra ir su teksto lyginimu (S9 savybe) bei paveiksliukų, nuorodų, formos įvedimo laukų rikiavimo tvarką (S5 ir S6 savybės).

6 IŠVADOS

Apibendrindami darbą galime teigti:

1. Testų orakulas turėtų būti universalus, pritaikytas kuo platesnei žiniatinklio IS aibei.
2. Testų orakulas turėtų ieškoti semantinių pokyčių HTML atsake, nes tik taip nagrinėjant HTML atsaką galime sumažinti *klaidingai teigiamų* ir *klaidingai neigiamų* rezultatų kiekį.
3. Atlikus tyrimą išgrynintos ir apibrėžtos būdingos visoms žiniatinklio IS semantinės HTML savybės.
4. Atliktas eksperimentas parodė, jog testų orakulas, nagrinėjantis ir HTML žymų struktūrą, ir tarp žymų esantį turinį, turi didžiausią efektyvumą. Tačiau tai svarbu tik toms žiniatinklio IS, kuriose teksto tarp žymų pakitimai yra laikomi klaida. Dažnu atveju būna priešingai – tekstas tarp žymų būna beveik arba visai nesvarbus.
5. Testų orakului yra svarbiau aptikti visas klaidas (*klaidingai neigiamų* pranešimų sumažinimas iki 0) su nežymiu *klaidingai teigiamų* pranešimų padidėjimu, negu praleisti ir neinformuoti testuotojo apie tikras klaidas, tačiau su mažesniu klaidingai teigiamų pranešimų kiekiu.
6. Eksperimento metu išsiaiškinti geriausi testų orakulai ir jų efektyvumai:
    a. Nagrinėjant visus eksperimento duomenis efektyviausias buvo 3-os kartos RegReload testų orakulas 76.92%, kuris papildomai ieško teksto tekste.
    b. Atmetus S9 (teksto tarp žymų tikrinimas) ir S15.4 (stipriai pakitusi struktūra) eksperimento metu surinktus duomenis, efektyviausias buvo 1-os kartos RegReload testų orakulas (90%).
7. Testų orakulo efektyvumas labai priklauso nuo tikrinamų HTML semantinių savybių interpretavimo. Jau nekalbant apie skirtingas žiniatinklio informacines sistemas, netgi toje pačioje testuojamoje žiniatinklio IS su tuo pačiu testavimo atveju, vienoje versijoje vaikinių žymų rikiavimo tvarka (S11 savybė) gali būti svarbi, kitoje ne. Prie šio pastebėjimo galime priskirti ir teksto lyginimo (S9 savybė), paveiksliukų, nuorodų, formos įvedimo laukų rikiavimo tvarką (S5 ir S6 savybės).

7 LITERATŪRA

[1] **Al Shaar, H., Haraty, R.** Modeling and automated blackbox regression testing of web application, *Journal of Theoretical and Applied Information Technology,* vol. 4, no. 12, p. 1182-1198, 2008.

[2] **Alshahwan, N., Harman, M.** Automated Session Data Repair for Web Application Regression Testing, *in Software Testing, Verification, and Validation, 2008 1st International Conference*, Lillehammer, p. 298 - 307, 2008.

[3] **Anttonen, N., Salminen, A. *et al.*** Transforming the web into a real application platform: new technologies, emerging trends and missing pieces, *in Symposium on Applied Computing* Taichung, Taiwan, 2011.

[4] **Biswas, S., Mall, R. *et al.*** Regression Test Selection Techniques: A Survey, *Informatica,* vol. 35, p. 34, 2010.

[5] **Bruns, A., Kornstadt, A. *et al.*** Web Application Tests with Selenium, *Software, IEEE,* vol. 26, no. 5, p. 4, 2009.

[6] **Dobolyi, K., Weimer, W.** Harnessing Web-Based Application Similarities to Aid in Regression Testing, *in Software Reliability Engineering (ISSRE) International Symposium,* p. 71, 2009.

[7] **Esposito, D., A;, S.** Architecting Microsoft® .NET Solutions for the Enterprise, p. 66. *Microsoft Press,* 2009.

[8] **Graves, T. L. H., M.J. ; Kim, J. ; Porters, A. ; Rothermel, G. ;** . An empirical study of regression test selection techniques, *Software Engineering*, 1998.

[9] **Holmes, A., Kellogg, M.** Automating Functional Tests Using Selenium, *in Agile Conference*, Minneapolis, 2006.

[10] **Yoo, S., Harman, M.** Regression Testing Minimisation, Selection and Prioritisation : A Survey, *Software Testing, Verification and Reliability,* vol. Volume 22, no. Issue 2, , p. pages 67–120,, 2007.

[11] **Khan, T. A., Heckel, R.** A Methodology for Model-Based Regression Testing of Web Services, *in Testing: Academic and Industrial Conference - Practice and Research Techniques*, p. 123 - 124, 2009

[12] **Korel, B., Al-Yami, A. M.** Automated Regression Test Generation, *in Software Testing and Analysis*, New York, 1998.

[13] **Marchetto, A., Ricca, F. *et al.*** A case study-based comparison of web testing techniques applied to AJAX web applications, *International Journal on Software Tools for Technology Transfer,* vol. 10, no. 6, p. 16, 2008.

[14] **Memon, A. M., Xie, Q.** Studying the fault-detection effectiveness of GUI test cases for rapidly evolving software, *in Software Engineering IEEE Transactions*, 2005.

[15] **Offutt, J.** Quality attributes of web software applications, *Software, IEEE,* vol. 19, no. 2, p. 8, 2002.

[16] **Purvinis, J., Prelgauskas, J.** Žiniatinklio informacinių sistemų testavimo įrankis, *in XVI tarpuniversitetinė magistrantų ir doktorantų konferencija*, Kaunas, p. 31 - 34, 2011.

[17] **Roest, D., Mesbah, A. *et al.*** Regression Testing Ajax Applications: Coping with Dynamism, *in Software Testing, Verification and Validation (ICST), 2010 Third International Conference,* Paris, p. 127, 2010.

[18] **Rothermel, G., Harrold, M. J.** Analyzing regression test selection techniques, *IEEE Transactions on Software Engineering,* vol. 22, no. 8, 1996

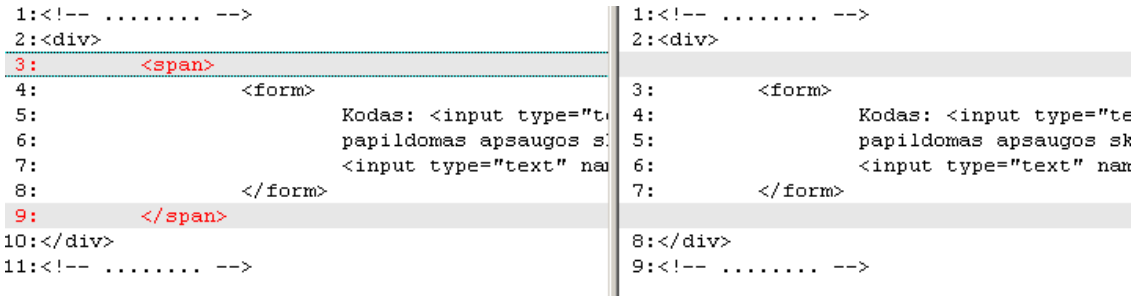
[19] **Rothermel, G., Harrold, M. J.** Selecting regression tests for object-oriented software, *in Software Maintenance International Conference,* Victoria, p. 14 - 25, 1994.

[20] **Rothermel, G., Harrold, M. J. *et al.*** Empirical studies of test-suite reduction, *Software Testing, Verification and Reliability,* vol. 12, no. 4, p. 219 - 249, 2002.

[21] **Rothermel, G., Karre, S. *et al.*** Leveraging user-session data to support Web application testing, *Software Engineering*, vol. 31, no. 3, p. 187 - 202 2005.
[22] **Ruth, M., Tu, S.** A Safe Regression Test Selection Technique for Web Services, *International Conference on Internet and Web Applications and Services*, 2007.
[23] **Sampath, S., Bryce, R. C. *et al.*** Prioritizing User-Session-Based Test Cases for Web Applications Testing, *in Software Testing, Verification, and Validation International Conference*, 2008.
[24] **Sampath, S., Sprenkle, S. *et al.*** Applying Concept Analysis to User-Session-Based Testing of Web Applications, *Software Engineering, IEEE Transactions*, vol. 10, no. 33, p. 16, 2007.
[25] **Silva, L. M.** Comparing Error Detection Techniques for Web Applications: An Experimental Study, *in Network Computing and Applications*, p. 144 - 151, 2008.
[26] **SmartBear**. "TestComplete from SmartBear," 2012-04-30; http://smartbear.com/products/qa-tools/automated-testing-tools.
[27] **Sneed, H. M.** Testing a Web application *in Web Site Evolution*, p. 3 - 10, 2004.
[28] **Sprenkle, S., Gibson, E. *et al.*** Automated Replay and Failure Detection for Web Applications, 2005.
[29] **Sprenkle, S., Hill, E. *et al.*** Learning Effective Oracle Comparator Combinations for Web Applications, *in Quality Software*, Portland, p. 8, 2007.
[30] **Sprenkle, S., Pollock, L. *et al.*** Automated Oracle Comparators for Testing Web Applications, *in Software Reliability*, Trollhattan p. 10, 2007.
[31] **Tahat, L. H., Vaysburg, B. *et al.*** Requirement-based automated black-box test generation, *in Computer Software and Applications Conference*, Chicago, p. 489 - 495, 2001.
[32] **Taneja, K., Xie, T.** DiffGen: Automated Regression Unit-Test Generation, *in Automated Software Engineering (ASE) International Conference*, 2008
[33] **Visser, W., Păsăreanu, C. S. *et al.*** Test input generation with java PathFinder, *in Software testing and analysis* New York, 2004.
[34] **W3C**. "HTML5 differences from HTML4," 2012-04-28; http://www.w3.org/TR/html5-diff/.
[35] **W3C**. "HTML 4.01 Specification," 2012-04-28; http://www.w3.org/TR/html4/.
[36] **Wei-Tek, T., Xiaoying, B. *et al.*** Scenario-based functional regression testing, *in Computer Software and Applications Annual International Conference*, Chicago, p. 496, 2001.
[37] **Wong, W. E., Horgan, J. R. *et al.*** A study of effective regression testing in practice, *PROCEEDINGS The Eighth International Symposium On Software Reliability Engineering*, p. 264 1997

8 PRIEDAI

8.1 Žiniatinklio IS S15 savybę atitinkančios ketvirtos mutacijos fragmentas



26 Paveikslėlis. Žiniatinklio IS S15 savybę atitinkančios ketvirtos mutacijos fragmentas

8.2 Įdiegimo aktas

REGISTRŲ CENTRAS

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

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

PROGRAMŲ SISTEMOS PERDAVIMO IR APROBAVIMO AKTAS

2011 m. Gruodžio 19 d.

Programų sistemos pavadinimas „Žiniatinklio informacinių sistemų testavimo įrankis“

Kūrinio tipas Programinė įranga

Programų sistemos sukūrimo data 2011 m. Gruodžio 16 d.

Kūrinio įteikimo UŽSAKOVUI data 2011 m. Gruodžio 19 d.

Užsakovo arba trečiojo asmens Kūrinio aprobavimo rezultatas:

Sistema įdiegta sėkmingai ir atitinka iškeltus reikalavimus.

Kūrinio aprobavimo data 2011 m. Gruodžio 19 d.

Kūrinio originalo saugotojas Julius Purvinis

AUTORIUS		UŽSAKOVAS	
Julius Purvinis		Algirdas Remeikis	
(vardas, pavardė)	*(parašas)*	*(vardas, pavardė)*	*(parašas)*

Valstybės įmonės Registrų centro
Informacinių technologijų centro
IS konstravimo departamento viršininkas — Antanas Krikščiūnas

Asmens pareigų pavadinimas — *(Parašas)* — *(Vardas ir pavardė)*

Parengė
Algirdas Remeikis
2011 12 19

8.3 ŽINIATINKLIO INFORMACINIŲ SISTEMŲ TESTAVIMO ĮRANKIS

Julius Purvinis[1], **Justinas Prelgauskas**[2]

Kauno Technologijos Universitetas, Programų inžinerijos katedra, Studentų gatvė 50–406, Kaunas, Lietuva

julius.purvinis@stud.ktu.lt [1] *justinas.prelgauskas@ktu.lt* [2]

Santrauka (abstract). Tobulėjant technologijoms, kuriant vis sudėtingesnes žiniatinklio informacines sistemas, taip pat tokias, kurios aptarnauja daug klientų, tampa vis sunkiau užtikrinti stabilų ir efektyvų informacinių sistemų darbą. Šioje publikacijoje pristatome mūsų sukurtą testavimo įrankį, skirtą tokių sistemų regresiniam ir apkrovos testavimui. Taip pat pateikiame panašių sistemų analizę ir palyginimą su mūsų sukurtu įrankiu.

Raktiniai žodžiai: žiniatinklis, informacinė sistema, regresinis, apkrovos, vartotojo sąsajos, juodos dėžės, testavimas.

1 Įžanga

Tobulėjant technologijoms, kuriant vis sudėtingesnes žiniatinklio informacines sistemas (IS), taip pat tokias, kurios aptarnauja daug klientų, tampa sunku užtikrinti stabilų ir efektyvų IS darbą. Kiekviena IS, kuri aptarnauja didesnį vartotojų kiekį, anksčiau ar vėliau susiduria su apkrova, dėl kurios prastėja teikiamų paslaugų kokybė, vartotojai būna nepatenkinti aptarnavimu. Todėl į šių dienų IS kūrimo procesą vis dažniau įtraukiami apkrovos, streso būsenų testavimai, programinio kodo optimizavimas [9]. Tačiau atsiranda dar didesnė problema, kai sukurtą didžiulę IS reikia koreguoti. Tai yra dažniausiai vykdoma veikla eksploatavimo bei palaikymo fazėje [14] ir dažniausiai yra brangi [5]. Kad būtų galima efektyviai sekti, ar pataisymas ne tik pašalino klaidą, bet ir nepakenkė jau esamoms funkcijoms, yra naudojamas regresinis testavimas. Regresinis testavimas pagrįstas veiksmų atkartojimu žiniatinklio informacinėje sistemoje ir gautų rezultatų tikrinimu po atlikto pataisymo. Atlikus iš IS gauto atsako analizę ir palyginus su testuose išsaugotu atsaku galima identifikuoti sistemos klaidas.

Regresinis testavimas gali būti vykdomas keliais būdais: programinio kodo lyginimu [11], vienetų testų vykdymas regresijai surasti [10], programinio kodo analizavimu [3, 6, 14], bei grafine vartotojo sąsaja paremtas būdas, kai programos elgesio pakitimų ieškoma imituojant vartotojo veiksmus ir lyginant gautus programos išėjimo duomenis [4]. Kadangi žiniatinklio IS regresiniame testavime programinio kodo analizė nėra efektyvi, nes neatskleidžia pilno įėjimo ir išėjimo srautų sąryšio, mūsų plėtojamas įrankis remiasi veiksmų atkartojimu grafinėje vartotojo sąsajoje [1]. Veiksmai atkartojami užfiksuotų užklausų siuntimu, kadangi visą žiniatinklio IS funkcionalumą galime pasiekti naršyklės pagalba. Šiame dokumente apžvelgiama regresinio ir apkrovos testavimo įrankio architektūra bei veikimo principas. Atliekama rinkoje esančių alternatyvų analizė ir palyginimas.

2 Testavimo įrankis

Kad būtų galima atlikti žiniatinklio informacinių sistemų regresijos ir apkrovos testavimą, įrankį turi sudaryti 3 nesunkiai įžvelgiamos sritys [1]: naršyklė, užklausas / atsaką fiksuojanti sritis, bei analizatorius. Naršyklė reikalinga atsako atvaizdavimui, kurio pagalba yra sudaromi ir atnaujinami testai. Užklausų / atsako fiksavimas reikalingas automatiniam interaktyvumo sukūrimui, testų sudarymui. Analizatorius atlieka turimo ir naujai gauto atsako analizę, parodo tikėtinas klaidas. Naršyklė ir užklausų / atsako fiksatorius turi egzistuoti kartu.

Atsako atvaizdavimui pasirinkta *Mozilla Firefox* naršyklė dėl galimybės panaudoti jos teikiamas HTTP kanalo sąsajas užklausų ir atsako fiksavimui. Atsako analizavimo darbus atlieka *Regresinių testų orakulo* klasė. *Mozilla Firefox* naršyklei yra sukurtas ne vienas įskiepis, padedantis žiniatinklio IS programuotojui ar testuotojui. Kaip pavyzdį galima paminėti programuotojų tarpe gerai žinomą *FireBug* – padeda visapusiškai nagrinėti DOM[9] objektą. Taip pat kiti *Mozilla Firefox* pagalbiniai įrankiai būtų: *httpfox* ir *Tamper Data* skirti HTTP srauto nagrinėjimui. Mūsų plėtojamas regresinio ir apkrovos testavimo įrankis papildo šių pagalbinių priemonių aibę naršyklės aplinkoje.

2.1 Veikimo principas

Įrankio veikimas pagrįstas iš *Mozilla Firefox* naršyklės (kliento) išeinančių užklausų ir iš serverio grįžtančio atsako fiksavimu bei užklausų atkūrimu (siuntimu) [4]. Iš pradžių vartotojas naršo po informacinės

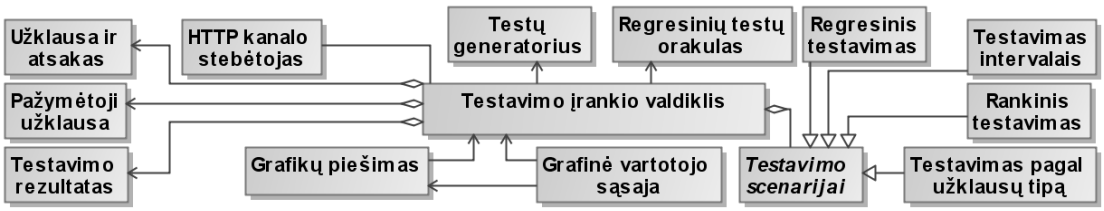
[8] angl. *Model-based testing.*

sistemos internetinį puslapį. Naršymas generuoja HTTP protokolu siunčiamas *GET* ir *POST* tipo užklausas. Testavimo įrankis išsaugo generuojamas užklausas ir gautą atsaką, o testavimo metu užklausas atkuria, pilnai imituodamas vartotojo interaktyvumą, taip pat fiksuoja iš serverio gautą atsaką, bei užklausų apdorojimo laiką.

Regresiniame testavime dažnai sprendžiama problema – reikalingų vykdyti testų atrinkimas iš sukauptų testų aibės. Tam yra naudojami įvairaus patikimumo testų atrinkimo metodai [3, 7, 8, 10, 14], kurie svyruoja nuo konservatyvaus, kai po kiekvienos programos modifikacijos yra vykdomi visi testai, iki atsitiktinių testų parinkimo, kuris dažniausiai nėra efektyvus ir pilnai nepadengia galimų klaidų atsiradimo sričių. Mūsų kuriamas įrankis remiasi reikalingų vykdyti testų atrinkimu pagal priklausomybes [12], kurias nustato testuotojas arba programuotojas. Šios priklausomybės bus priskiriamos sukauptų užklausų grupių sąraše, pažymint, jog testuojant vienos grupės užklausas, bus testuojamos ir kitos susijusios grupės. Sudėtingiausia žiniatinklio IS regresinio testavimo dalis – parodyti, ar gautas HTML žymų atsakas neturi neatitikimų [2]. Dažniausiai pasitaiko tai, kad HTML žymų neatitikimai yra leistini: jie nekeičia puslapio išvaizdos arba atspindi naują funkcionalumą [13]. Tiesiogiai lyginant du HTML dokumentus yra galimas perteklinis regresijos aptikimas. Todėl regresijos aptikimui surinktas serverio atsakas yra analizuojamas kaip HTML žymų medžio struktūra. Tikrinamas žymų korektiškumas, daromos išimtys dėl papildomų atributų, nekreipiama dėmesio į žymų susikeitimą vietomis tam tikrais atvejais ir į tarp žymų esantį tekstą [13]. Toks analizavimo metodas nėra visiškai tikslus, tačiau jis yra nuolat tobulinamas ir daug efektyvesnis už tiesioginį HTML lyginimą.

Apkrovos testavimo metu, iš išsaugotų užklausų priklausomai nuo scenarijaus, įrankis sukuria apkrovą, kuri lemia vienokį ar kitokį serverio užimtumą. Apkrovos metu matuojamas serverio atsako laikas: kiek serveris užtrunka, kol apdoroja konkrečią užklausą, ar užklausų grupę, kai jis yra veikiamas tam tikro dydžio apkrova.

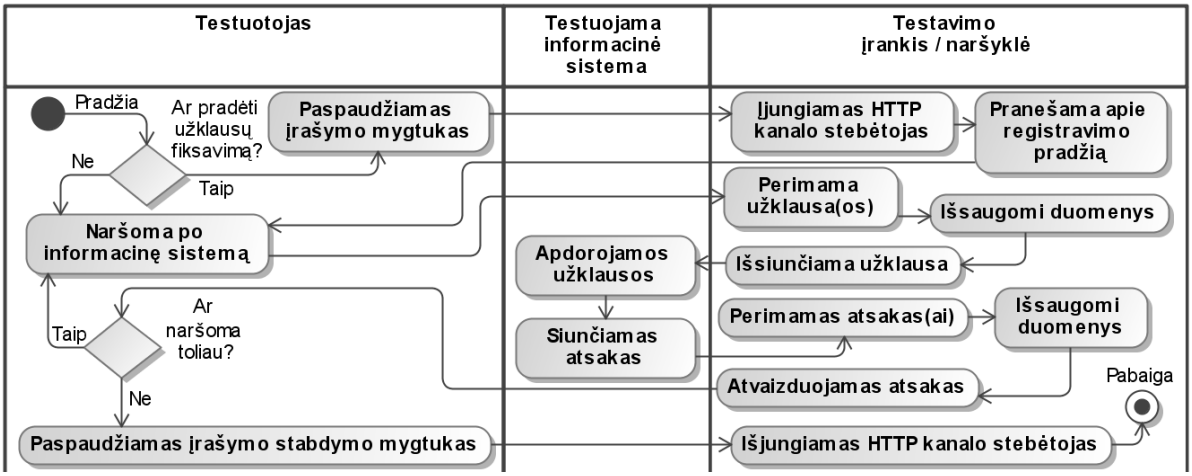
2.2 Architektūra



1 pav. Testavimo įrankio klasių diagrama

Klasių diagrama pateikta 1 pav. Įrašų klasės, iš kurių sudaromi masyvai su duomenimis: *Užklausa ir atsakas*, *Pažymėtoji užklausa*, *Testavimo rezultatas*. Testavimo scenarijai: *Testavimas intervalais*, *Testavimas pagal užklausų tipą*, *Rankinis testavimas*, *Regresinis testavimas*. Pirmi trys iš išvardintųjų scenarijų yra skirti apkrovos testavimui. *HTTP kanalo stebėtojas* registruoja visas HTTP sraute atsirandančias užklausas. *Testų generatorius* pagal testavimo scenarijus atlieka užklausų siuntimo darbą. *Regresinių testų orakulas* analizuoja, sulygina iš serverio gautus atsakus ir aptinka neatitikimus. *Grafikų piešimo* klasė atvaizduoja duomenis grafikais. *Testavimo įrankio valdiklis* reguliuoja visą įrankio darbą ir teikia sąsajas. *Grafinė vartotojo sąsaja* atvaizduoja įrankio išvaizdą ir funkcionalumą.

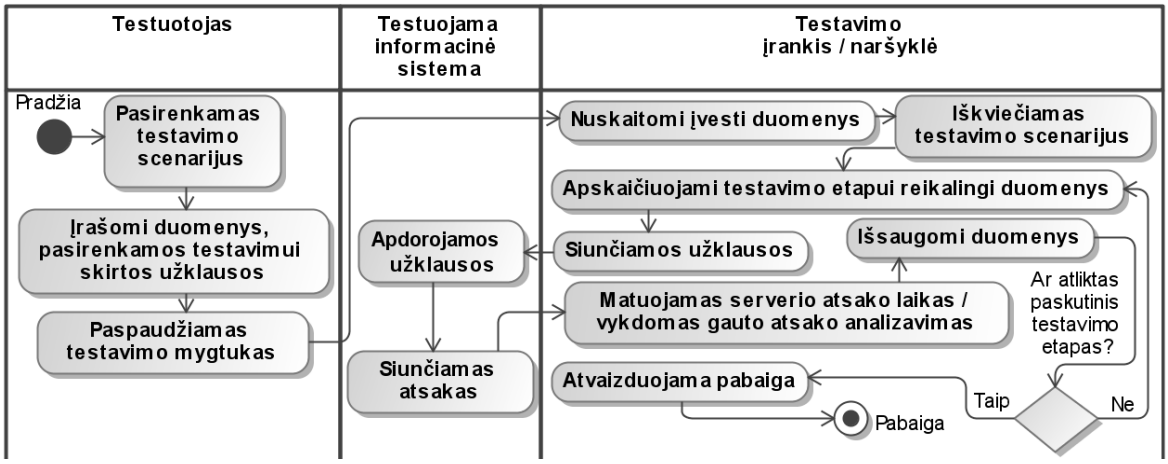
Testavimo įrankio užklausų ir atsako įrašymas bei testavimo veiksmai pateikti 2 ir 3 pav. veiklos diagramomis. Kadangi įrankis glaudžiai susijęs su naršykle, diagramose pateikiamas kaip viena dalis.



1 pav. Užklausų fiksavimo veiklos diagrama

Kaip matyti 2 pav. testuotojas norėdamas įrašyti užklausas ir atsakus, visų pirma, turi paspausti įrašinėjimo mygtuką (įjungti HTTP kanalo stebėtoją). Tuomet galima naršyti po puslapį. Užklausos ir atsakas bus fiksuojami. Jeigu reikia fiksuoti konkrečią užklausą su atsaku, kurią galima pasiekti tik gilesniuose

informacinės sistemos puslapiuose, įrašinėjimą galima sustabdyti ir vėl įjungti prieš tai, kai paspaudus puslapyje bus vykdoma jau norima užklausa.



2 pav. Regresinio ir apkrovos testavimo veiklos diagrama

Tai yra bendra – testavimo scenarijų nenagrinėjanti, tik testavimo procesą aprašanti diagrama. Kai testuotojas pasirenka testavimo scenarijų ir suveda duomenis, iškviečiamas pasirinktas scenarijus, kuriame yra apskaičiuojami testavimo etapai ir reikalingos išsiųsti apkrovos, jei tai apkrovos testavimo scenarijus. Regresinio testavimo metu etapas yra tolygus iškviestam regresijos aptikimo testui. Tokių etapų kiekis priklauso nuo to, koks informacinės sistemos modulis yra testuojamas, ir kiek yra numatytų priklausomybių su kitais testais.

3 Alternatyvūs įrankiai

Rinkoje egzistuoja nemažai testavimo įrankių tiek regresiniam, tiek apkrovos testavimui. Šiame skyriuje palyginama tik nedidelė rinkoje egzistuojančių testavimo įrankių dalis. Visų veikimo principai skiriasi. Įrankių galima rasti kur kas daugiau, specifinių, pritaikytų konkrečiai sričiai.

3.1 SeleniumHQ

Įrankis skirtas automatizuoti žiniatinklio informacinių sistemų testavimą daugelyje platformų. Veikia nepriklausomai nuo naršyklės. SeleniumHQ veikimas pagrįstas vartotojo veiksmų naršyklėje stebėjimu [4]. Kiekvienas pelės paspaudimas ant tam tikro puslapio elemento ir klaviatūros mygtukų paspaudimai yra fiksuojami. Visi užfiksuoti vartotojo veiksmai gali būti sukonvertuoti į *Ruby*, *Java*, *Perl*, *PHP*, *Python*, *C#* programavimo kalbas ir vykdomi kaip vienetų testai. Įrankis turi galimybę nuotoliniu būdu paleisti keletą skirtingų testų nutolusiuose serveriuose arba net GRID serverių tinkle. Tai padeda sutrumpinti testavimo laiką, kai žiniatinklio IS iškart testuojama su keletu skirtingų naršyklių ir keletu operacinių sistemų.

3.2 TestOptimal

Šis įrankis yra *modeliu paremtus*[10] [3] testavimo įrankius vienijantis rinkinys, skirtas žiniatinklio informacinių sistemų funkciniam, regresiniam ir apkrovos, streso testavimui. Pats įrankis yra lokaliai veikianti žiniatinklio informacinė sistema, todėl veikia su daugeliu naršyklių. Tai pažangus, sudėtingas ir daug funkcionalumo turintis įrankis. Duomenis geba išsaugoti savo duomenų bazėje ir atvaizduoti grafikais.

3.3 WebLoad

Didžiulė darbastalio tipo programa skirta žiniatinklio informacinių sistemų apkrovos testavimui. Susidedanti iš trijų pagrindinių dalių: testavimo scriptų sudarymo, apkrovos sudarymo ir testavimo, testavimo rezultatų analizavimo. Apkrovos generatorius gali būti ir atskiras kompiuteris (serveris). Veikimo principas yra toks: iš testuotojo veiksmų naršyklėje sudaromas testavimo scriptas, išsiunčiamas į testų generatorių (kitą kompiuterį, arba vykdomas tame pačiame). Testų generatorius išsiunčia apkrovą į testuojamą IS, išmatuoja atsako laikus ir atsiunčia rezultatus. Tada su specialiu programos įrankiu, gauti testavimo rezultatai analizuojami įvairių grafikų pagalba.

4 Testavimo įrankių palyginimas

Apžvelgtų įrankių palyginimas pateiktas lentelėje 1.

[8] angl. *Model-based testing.*

Lentelė Nr.1 Įrankių palyginimas

Įrankio pavadinimas	Tipas	Paskirtis	Privalumai	Trūkumai
SeleniumHQ	Naršyklės įskiepis	Vienetų testų sudarymas	Veikia daugelyje naršyklių. Testavimo scriptai gali būti verčiami į keletą kalbų.	Tikslui, kuriam sukurtas trūkumų neturi.
TestOptimal	Lokaliai veikianti žiniatinklio IS	Regresinių, apkrovos testų sudarymas, vykdymas.	Duomenis saugo savo duomenų bazėje. Turi daug testavimo būdų.	Sudėtingas valdymas, sunku perprasti.
WebLoad	Darbastalio programa	Įvairių tipų apkrovos testavimas	Apkrova gali būti generuojama iš atskiro kompiuterio.	Sudėtingas valdymas, sunku perprasti. Sudėtingas generatoriaus konfigūravimas
Mūsų įrankis	*Firefox* plėtinys	Regresinių, apkrovos testų sudarymas, vykdymas.	Paprasta naudotis. HTTP lygio užklausų fiksavimas.	Veikia tik su *Firefox*.

5 Išvados

Kadangi tiek regresinis, tiek apkrovos testavimas yra būtini, tačiau brangūs procesai programinės įrangos gyvavimo metu, tokių testavimo įrankių paklausa auga. Daugelis vartotojų apsilankę žiniatinklio informacinėje sistemoje ir pastebėję netinkamą ar lėtą jos veikimą dažniausiai renkasi kitą informacinę sistemą. Todėl yra svarbu nuolat gerinti sukurtos žiniatinklio informacinės sistemos veikimo kokybę ir našumo charakteristikas. Nors rinkoje testavimo įrankių yra pakankamai daug, tačiau surasti efektyvų ir nesunkiai panaudojamą, vis dar yra sunku.

Literatūros sąrašas

[1] **Al Shaar, H., Haraty, R.** Modeling and automated blackbox regression testing of web application, *Journal of Theoretical and Applied Information Technology*, vol. 4, no. 12, p. 1182-1198, 2008.

[2] **Dobolyi, K., Weimer, W.** Harnessing Web-Based Application Similarities to Aid in Regression Testing, *in Software Reliability Engineering (ISSRE) International Symposium*, p. 71, 2009.

[3] **Khan, T. A., Heckel, R.** A Methodology for Model-Based Regression Testing of Web Services, *in Testing: Academic and Industrial Conference - Practice and Research Techniques*, p. 123 - 124, 2009

[4] **Memon, A. M., Xie, Q.** Studying the fault-detection effectiveness of GUI test cases for rapidly evolving software, *in Software Engineering IEEE Transactions*, 2005.

[5] **Rothermel, G., Harrold, M. J.** Analyzing regression test selection techniques, *IEEE Transactions on Software Engineering*, vol. 22, no. 8, 1996

[6] **Rothermel, G., Harrold, M. J.** Selecting regression tests for object-oriented software, *in Software Maintenance International Conference*, Victoria, p. 14 - 25, 1994.

[7] **Ruth, M., Tu, S.** A Safe Regression Test Selection Technique for Web Services, *International Conference on Internet and Web Applications and Services*, 2007.

[8] **Sampath, S., Bryce, R. C. *et al.*** Prioritizing User-Session-Based Test Cases for Web Applications Testing, *in Software Testing, Verification, and Validation International Conference*, 2008.

[9] **Subraya, B. M.**, Integrated Approach to Web Performance Testing: A Practitioner's Guide. *Web-Based Systems and Performance Testing*. M. Potter, K. Roth *et al.*, eds., p. 1-7, 9-14, 15-21, 22-27, *IRM Press*, 2006.

[10] **Taneja, K., Xie, T.** DiffGen: Automated Regression Unit-Test Generation, *in Automated Software Engineering (ASE) International Conference*, 2008

[11] **Vokolos, F. I., Frankl, P. G.** Empirical evaluation of the textual differencing regression testing technique, *International Conference on Software Maintenance*, p. 44 - 53, 1998.

[12] **Wei-Tek, T., Xiaoying, B. *et al.*** Scenario-based functional regression testing, *in Computer Software and Applications Annual International Conference*, Chicago, p. 496, 2001.

[13] **Wen, R. B.** URL-driven automated testing, *in Quality Software Asia-Pacific Conference*, p. 268, 2001.

[14] **Zhang, Z., Huang, J. *et al.*** Regression Test Generation Approach Based on Tree-Structured Analysis, *in Computational Science and Its Applications (ICCSA) International Conference*, p. 244 - 249, 2010.

Testing tool of web-based information system

As software developers are developing technologies of the web, implementing more complex web-based information systems, also those, which serves many clients, it becomes more and more complex to maintain stable and effective work of information systems. In this paper we introduce our developed test tool designed to execute regression and load tests. Also we provide similar tools analysis and comparison with our created tool.