VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
KOMPIUTERIJOS KATEDRA

Baigiamasis magistro darbas

Žiniatinklio aplikacijų automatinis testavimas

Automated testing of web applications

Atliko: 2M kurso, 9 grupės studentas

Marius Žilėnas (parašas)

Darbo vadovas:

Dr. Valdas Rapševičius (parašas)

Recenzavo:

Dr. Linas Bukauskas (parašas)

Vilnius
2012

Turinys

Santrauka

Testavimas yra svarbus ir imlus darbui aplikacijos kūrimo ir vystymo procesas. Norint suvaldyti augančias testavimo sąnaudas, siekiama dalį žiniatinklio aplikacijos testavimo proceso automatizuoti. Žiniatinklio aplikacijų atnaujinimo ciklas yra trumpas – žiniatinklio aplikacijos atnaujinamos dažnai, todėl žiniatinklio aplikacijos testavimas tampa ypač aktualus.

Automatinis testavimas yra vertinamas nevienareikšmiškai, tai yra taip pat imlus darbui uždavinys, nes reikalauja apibrėžti testavimo procesą ir sukurti modelį automatiniam testų vykdymui. Vis dėlto, parengus automatinį testavimą, žiniatinklio aplikacijos testavimas palengvėja – sumažinamas rankinio testavimo kiekis.

Darbo tikslas – parinkus geriausią žiniatinklio aplikacijų automatinio testavimo įrankį, pasiūlyti metodą žiniatinklio aplikacijų automatiniam testavimui.

Uždaviniai – įvertinti žiniatinklio aplikacijų automatinio testavimo įrankius, parinkti geriausią įrankį, sukurti automatinio testavimo modelį.

Rezultatai – magistro baigiamajame darbe nustatytas žiniatinklio aplikacijos automatiniam testavimui geriausiai tinkantis įrankis, kuriuo remiantis sukurtas ir įvertintas įrankis automatiniam žiniatinklio aplikacijų testavimui; apibrėžtas žiniatinklio aplikacijos automatinio testavimo modelis (sukurtas įrankis pritaikomas modelyje).

Raktiniai žodžiai: testavimas, žiniatinklio aplikacija, automatinis testavimas.

Summary

Testing is important and labor intensive part of application development. In order to manage growing testing efforts, we try to automate part of the web application testing process. Web application testing is important because the timespan between web application updates is short and web applications are updated often.

Automated testing is a two-edged effort, it is also labour intensive task, because one has to define testing process and create a model for automated test execution. However after automated testing succeeds – testing takes less efforts, amount of manual testing is reduced.

The objective of this work is after selecting the best automated web application testing tool to introduce an approach to automated web application testing.

Goals: to evaluate web application automated testing tools, identify best tool, create a model of automated testing.

Results: we identified the best tool fitting for automated testing of web application. Created and evaluated automated web application testing tool, defined and implemented a model of automated testing, implemented tool adapted in the model.

Keywords: testing, web application, automated testing.

Įvadas

Žiniatinklio aplikacijos užima didelę dalį šiandien naujai sukuriamų aplikacijų, jos plinta ir keičia darbastalio (desktop) aplikacijas dėl didelio prieinamumo ir patogumo. Žiniatinklio aplikacija – tai aplikacija, kuri iškviečiama naršyklės pagalba per žiniatinklį. Žiniatinklio aplikacijų atnaujinimo ciklas yra labai trumpas (ypatingai dėl judriojo programavimo metodikų taikymo), tad žiniatinklio aplikacijos yra dažnai atnaujinamos, dėl to jų testavimas yra labai aktualus.

Testavimas – vienas iš aplikacijos kūrimo procesų. Tai veikla, atliekama siekiant įvertinti produkto kokybę ir produktą gerinti, nustatant defektus (trūkumus) bei problemas. Visos aplikacijos prieš jas pristatant ir naudojant turi būti testuojamos. Testavimas, kaip aplikacijos kūrimo procesas, yra senai vystomas ir sukurta daug testavimo metodikų.

Dėl dažno žiniatinklio aplikacijos atnaujinimo, jos testavimas reikalauja daug išteklių. Todėl testavimo procesas ar jo dalis dažnai yra atliekami automatiškai. Automatinio testavimo metu gali būti įvertinamas aplikacijos korektiškumas, funkcionalumo pilnumas, duomenų integralumas, saugumas ir kokybė. Automatinio testavimo proceso paskirtis – atskleisti kokybinę informaciją apie aplikaciją, pasinaudojant klaidų suradimu automatinio testavimo metu. Automatinis testavimas atliekamas panaudojant automatinio testavimo įrankius. Automatinis testavimas vykdomas be testuotojo ir gali būti vykdomas nuolatos.

Norint atlikti pilną žiniatinklio aplikacijos testavimą, reikia sukurti testavimo atvejus ir pagal juos atlikti testavimą. Tuo tarpu norint atlikti automatinį testavimą, reikia papildomai sukurti testavimo skriptus bei jų vykdymui parengti automatinio testavimo įrankius. Testuojant rankiniu būdu pakanka pagal testavimo atvejį panaudoti aplikaciją ir užrašyti rezultatus. Testuojant automatiniu būdu naudotojui nebereikia vykdyti testavimo atvejyje nurodytų veiksmų – juos įvykdo testavimo skriptas, o testavimo atvejo veiksmų rezultatas įvertinamas automatiškai. Testuotojams sumažėja darbų – atsiranda laiko, kurį užuot testuodami sistemą, testuotojai gali panaudoti kitam uždaviniui – testų kūrimui ir plėtojimui.

Žiniatinklio aplikacijų automatiniam testavimui yra sukurta daug įrankių: tiek atvirojo kodo, tiek komercinių. Pagrindiniai automatinio testavimo įrankiai, kurie yra kitų testavimo įrankių pagrindas, yra Selenium RC (pirma Selenium versija) ir Selenium WebDriver (antra Selenium versija), Sahi, Watir ir kt. Kaip vienas iš būdų, kuriuo siekiama palengvinti testavimą: automatinių testų kūrimą ir automatinį vykdymą, magistro baigiamajame darbe, remiantis raktažodinio testavimo koncepcija, sukuriamas ir įvertinamas žiniatinklio aplikacijos testavimo proceso modelis, leidžiantis pritaikyti geriausią žiniatinklio aplikacijos automatinio testavimo įrankį, siekiant pagerinti žiniatinklio aplikacijos testavimo procesą.

Nors magistro baigiamajame darbe nagrinėjama automatinio testavimo tema iš esmės nėra nauja, o pastaruoju metu dar ir plačiai diskutuojama, tačiau žiniatinklio aplikacijų automatinis testavimas yra ypač aktualus dėl žiniatinklio aplikacijų plataus panaudojimo. Automatinio testavimo tikroji vertė atsiskleidžia ne naujų klaidų radime, o kuomet reikia atsakyti į klausimą, ar aplikacijoje pasikartoja jau anksčiau rastos klaidos.

Magistro baigiamojo darbo tikslas – parinkus geriausią žiniatinklio aplikacijų automatinio testavimo įrankį, pasiūlyti metodą žiniatinklio aplikacijų automatiniam testavimui.

Uždaviniai:

1) įvertinti žiniatinklio aplikacijų automatinio testavimo įrankius;
2) parinkti geriausią įrankį automatiniam žiniatinklio aplikacijų testavimui;
3) sukurti žiniatinklio aplikacijų automatinio testavimo modelį.

Rezultatai – magistro baigiamajame darbe nustatytas žiniatinklio aplikacijos automatiniam testavimui geriausiai tinkantis įrankis, kuriuo remiantis sukurtas ir įvertintas įrankis automatiniam žiniatinklio aplikacijų testavimui; apibrėžtas žiniatinklio aplikacijos automatinio testavimo modelis (sukurtas įrankis pritaikomas modelyje).

Baigiamojo darbo struktūra. Darbą sudaro įvadas, trys pagrindinės dalys, išvados ir rekomendacijos. Pirmoje darbo dalyje pateikiama testavimo samprata, išskiriami pagrindiniai testavimo metodai ir pateikiamas jų skirstymas į black-box, white-box metodus. Antroje dalyje pateikiama žiniatinklio aplikacijos samprata, išskiriami argumentai už ir prieš automatinį žiniatinklio aplikacijos testavimą, pateikiama automatinio testavimo kritika ir pasiūlymai, nagrinėjamas raktažodinis testavimas. Trečioje dalyje pateikiamas automatinio testavimo proceso modelis, įvertinama jo realizacija raktažodiniam testavimui, panaudojant atrinktą geriausią automatinio testavimo įrankį.

1. Testavimas

Testavimas – vienas iš aplikacijos kūrimo procesų. Pagal apibrėžimą „*Testavimas* – tai veikla, atliekama siekiant įvertinti produkto kokybę ir produktą gerinti nustatant defektus (trūkumus) ir problemas.“[1]

Automatinis testavimas – testavimas, panaudojant automatinio testavimo įrankius, automatiniame testavime aplikaciją testuoja ne testuotojas, o automatinio testavimo įrankis, vykdantis testavimo skriptą. Automatinio testavimo metu gali būti nustatomas aplikacijos korektiškumas, funkcionalumo pilnumas, duomenų integralumas, saugumas ir kokybė.

1. 1. Testavimo metodų skirstymas į white-box ir black-box

Testavimo metodai skirstomi į black-box ir white-box metodus[2], atitinkančius dvi perspektyvas, iš kurių testuotojas žiūri į testuojamą aplikaciją, kai kuria testus.

White-box testavimo atveju testuotojas turi priėjimą prie aplikacijos programinio kodo ir turi žinių apie testuojamos aplikacijos architektūrą, vidinę sandarą. Be to, testuotojas turi tiesioginį priėjimą prie testuojamos aplikacijos duomenų, duomenų struktūrų ir jas įgyvendinančių algoritmų programinio kodo. White-box testavimas yra žinomas kaip „struktūrinis testavimas“.[3] Toks testavimas yra naudojamas trijuose testavimo tipuose: unit testavimas, integravimo testavimas bei regresinis testavimas.

Kitas testavimo būdas yra black-box testavimas – tai testavimas, neatsižvelgiantis į testuojamos sistemos ar komponento vidinius mechanizmus ir susitelkiantis tik į sugeneruotą rezultatą atlikus duomenų įvedimą.[4] Black-box testuoja, ar aplikacija veikia taip, kaip tikimasi remiantis funkciniais reikalavimais.

1. 2. Testavimo metodai

Yra keli testavimo metodai, kurių kiekvienas turi savo vietą testavimo procese (žr. Lentelė 1). Pavyzdžiui, unit testavimas yra naudojamas pratestuoti kiekvieną mažiausią aplikacijos dalį, paprastai tokia dalis yra klasė. Apkrovos testavimas atliekamas, kai reikia sužinoti, kaip žiniatinklio aplikacija veikia esant didelei apkrovai. Testuojant konkrečią žiniatinklio aplikacijos vartotojo sąsajos dalį, naudojamas funkcinis arba grafinės vartotojo sąsajos testavimas.

---

[1] Bourque P., Dupuis R. "Guide to the Software Engineering Body of Knowledge 2004 Version", Guide to the Software Engineering Body of Knowledge, 2004. SWEBOK, 2004. ISBN: 0-7695-2330-7.

[2] Myers G. J. „The Art of Software Testing“. John Wiley & Sons, 2004. ISBN: 0471469122.

[3] Bourque P., Dupuis R. "Guide to the Software Engineering Body of Knowledge 2004 Version", Guide to the Software Engineering Body of Knowledge, 2004. SWEBOK, 2004. ISBN: 0-7695-2330-7.

[4] IEEE. IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, 1990.

Lentelė 1. Testavimo metodai ir jų skirstymas į black-box, white-box

Testavimo metodas	Testavimo būdas	Specifikacija testavimui	Kas atlieka
Unit testavimas	White-box ir black-box	Programinis kodas, žemo lygmens	Programuotojas, parašęs kodą
Integravimo testavimas	White-box ir black-box	Programinis kodas, aukšto lygmens	Programuotojas, parašęs kodą
Funkcinis testavimas	Black-box	Aukšto lygmens, reikalavimų specifikacija	Testuotojas
Sisteminis testavimas	Black-box	Reikalavimų specifikacija	Testuotojas
Priėmimo testavimas	Black-box	Reikalavimų specifikacija	Naudotojas
Regresinis testavimas	White-box ir black-box	Bet kuris modifikaciją aprašantis dokumentas, aukšto lygmens	Programuotojas arba testuotojas

Kiekvienas testavimo metodas turi savo ypatumus ir nevienodai tinkamas automatizuoti.

Unit testavimas – testavimo metu atliekami testai, kuriais patikrinama, ar programinio kodo dalys (unit'ai) suprogramuoti korektiškai. Unit testavimo metu surandama apie 65% visų klaidų.[5] Unit testavimo paskirtis – užtikrinti, kad aplikacijos kodo dalis veiktų korektiškai. Testuotojas, naudodamas white-box metodus, patikrina, ar programinis kodas žemajame struktūriniame lygmenyje veikia taip, kaip reikalaujama. Pavyzdžiui, unit testavimui testuotojas parašo testavimo kodą, kuris iškviečia testuojamos klasės objekto metodą ir patikrina, ar grąžinama reikšmė atitinka tikėtiną. Vienas unit testas paprastai apsiriboja vienos klasės ar komponento patikrinimu. Unit testavimą paprastai atlieka programinį kodą parašęs programuotojas, kuris sukuria kiekvienai programos klasei bent po vieną testavimo atvejį.

Integravimo testavimas – testavimo metu patikrinama sąveika tarp programinės įrangos komponentų – testavimo atvejai tikrina sąsajas tarp kelių unit'ų. Naudodamas black-box ir white-box testavimo metodus testuotojas (gali būti ir programuotojas) patikrina, kad integravus mažesnes programos dalis (unit'us), jos veikia kaip reikalaujama.

Funkcinis testavimas – paprastai atliekamas testavimas per aplikacijos grafinę vartotojo sąsają, siekiant išsiaiškinti/patikrinti, kaip aplikacija ir naudotojas sąveikauja, ar aplikacija veikia teisingai. Paprastai tokio testavimo metu tikrinama, kaip aplikacija apdoroja įvedamus duomenis, kaip veikia aplikacijoje naudojamos formos, vaizduojami duomenų įvedimo langai, žiniatinklio aplikacijoje tikrinama, kokie tekstai atvaizduojami, meniu elementai, paveiksliukai, kaip aplikacija reaguoja į elementų suaktyvinimą. Paprastai funkcinis testavimas atliekamas žmogaus, rankiniu būdu, tačiau greičiau ir patikimiau funkcinis testavimas atliekamas automatiniu būdu.

[5] Beizer B. Software Testing Techniques. Boston, International Thompson Computer Press, 1990.

Funkcinis testavimas dar yra skirstomas į:

- apkrovos testavimą – tikrina, ar testuojama aplikacija, ar jos komponentas veikia pagal reikalavimus esant ribiniam apkrovimui;
- našumo testavimą – tikrina, ar sistema atitinka specifikacijoje nurodytus greitaveikos reikalavimus;
- panaudojamumo testavimą – tikrina, kaip naudotojas gali: naudoti, įvesti duomenis, gauti rezultatus, dirbti su aplikacija ar komponentu. Tai testavimas, kuris negali būti automatizuotas.

Priėmimo testavimas – formalus testavimas, atliekamas siekiant patikrinti, ar aplikacija atitinka priėmimo kriterijus, kuriuos formuluoja galutinis aplikacijos naudotojas. Priėmimo testavimas vykdomas po funkcinio ir sisteminio testavimo, kai aplikacija tinkama pirminiam naudojimui. Priėmimo testavimas vykdomas galutinio aplikacijos naudotojo, kuris aplikaciją mato kaip „juodą dėžę“ (black-box).

Regresinis testavimas – „pasirinktinis sistemos ar jos komponento pertestavimas siekiant patikrinti, ar modifikacijos nesukėlė nepageidaujamo efekto ir, kad sistema ar jos komponentas vis dar atitinka specifikacijoje nurodytus reikalavimus.“[6]

Kadangi regresiniame testavime pakartotinai panaudojami jau įvykdyti testavimo atvejai, tai paprastai white-box unit testai ir white-box integravimo testai pakartojami regresinio testavimo metu. Regresinis testavimas yra toks aplikacijos pertestavimas, kai tikrinama, ar modifikacijos nepadarė nepageidaujamo efekto aplikacijai ar jos komponentui ir, ar aplikacija atitinka specifikacijoje nurodytus reikalavimus. Regresinio testavimo metu atliekami black-box tipo testai integravimo, funkcinių, sisteminių ir priėmimo testų lygmenyse.

Regresinio testavimo metu reikalaujama pertestuoti aplikaciją po kiekvieno pakeitimo. Tai dažnai atitinka visų turimų testų pakartotinį atlikimą siekiant ištestuoti aplikaciją, kad būtų patikrintas aplikacijos naujasis funkcionalumas ir senasis funkcionalumas. Regresinių testų automatizavimas padeda suvaldyti didėjantį testavimo atvejų ir testavimo rinkinių kiekį, palengvina testavimą. Todėl pirmiausiai automatizuojamas testavimas – regresinis testavimas.

Remiantis tyrimų išvadomis,[7] regresinis testavimas yra svarbi testavimo proceso dalis, kuriai išleidžiama ne mažiau nei 80% testavimo sąnaudų ir apie 50% programinės įrangos palaikymo sąnaudų. Nors regresinis testavimas apskritai yra pravartus aplikacijų kūrime ir vystyme, jis ypatingai naudingas tais atvejais, kai aplikacijos kūrimas ir vystymas tęsiasi ilgais periodais (ilgesniais nei keletas metų) – regresinis testavimas panaudojamas bendram aplikacijos veikimo

[6] IEEE. IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, 1990.

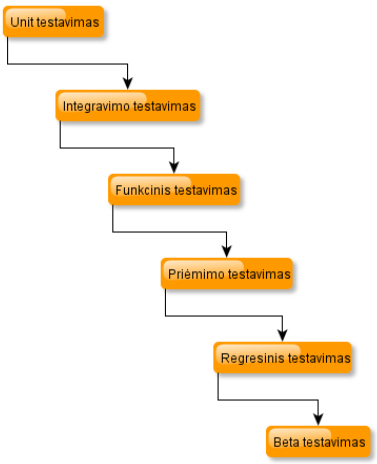
[7] Harrold M. J. "Reduce, reuse, recycle, recover: Techniques for improved regression testing", Software Maintenance, 2009. ICSM 2009. IEEE International Conference, p. 5, 20-26 Sept. 2009. ISSN: 1063-6773.

įvertinimui; vystant aplikacijas, turinčias funkcionuoti be sutrikimų – regresiniai testai panaudojami testuoti ir pertestuoti aplikacijas nuolatos.[8]

Beta testavimas – kai didžioji dalis aplikacijos funkcionalumo arba visa aplikacija jau gali būti panaudojama, atliekamas beta testavimas, kurį atlieka naudotojai, naudosiantys sukurtą aplikaciją.

Apibendrinant teigtina, jog nors žiniatinklio aplikacijos testavimo metodų yra išskiriama keli, testavimui paprastai būdingas visų testavimo metodų taikymas ir toks testavimo metodų taikymo eiliškumas:

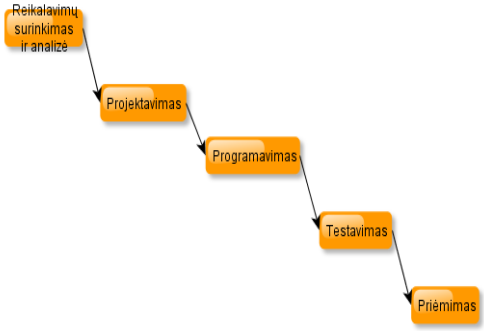
Paveikslas 1. Žiniatinklio aplikacijos testavimo vykdymo eiliškumas



1. 3. Testavimas programinės įrangos kūrimo procese

Nepriklausomai nuo to, kokiu modeliu remiantis kuriama programinė įranga, testavimas yra kiekviename modelyje numatytas svarbus etapas. Krioklio modelyje testavimas yra vienas iš atskirų etapų: analizė, reikalavimų surinkimas, projektavimas, programavimas, testavimas, priėmimas. Krioklio modelis numato tik vieną testavimo etapą, kuris vyksta po aplikacijos sukūrimo, dėl to krioklio modelyje testavimas nėra labai efektyvus, nes nenumatyti grįžimai prie projektavimo.

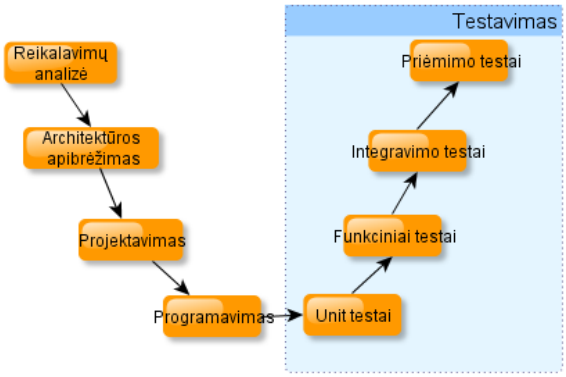
Paveikslas 2. Krioklio modelis.



[8] Onoma A. K., Tsai W. T., Poonawala M., Suganoma H. 1998. Regression testing in an industrial environment. Commun. ACM 41, 5 (May 1998), p. 81-86.

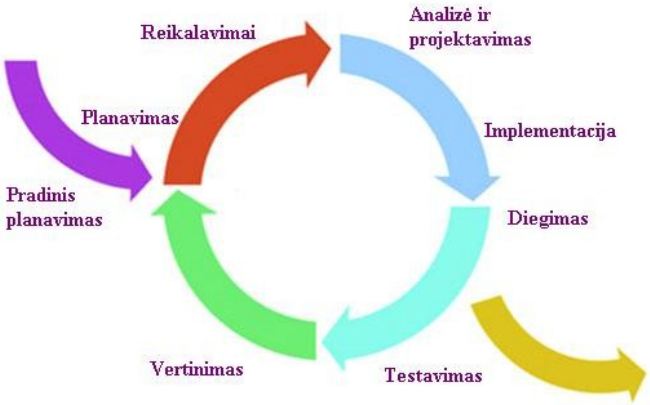
Kaip patobulintas krioklio modelis naudojamas V-modelis, kuris išskaido testavimą į kelis etapus ir susieja testavimą su visu aplikacijos kūrimo procesu.

Paveikslas 3. V-modelis.



Iteratyvusis modelis yra ciklinis kūrimo procesas, sukurtas atsižvelgiant į Krioklio modelio trūkumus. Modelis turi tokias pačius etapus (reikalavimai, analizė, projektavimas, implementacija), kaip krioklio modelis, tačiau jie yra įtraukti į iteracijas. Pirmosiose iteracijose svarbesni yra reikalavimai, analizė ir projektavimas, o vėlesnėse iteracijose daugiau dėmesio skiriama implementacijai ir testavimui.[9]

Paveikslas 4. Iteratyvus modelis.



[9] Pastaba. Svarbiausių SDLC modelių apžvalgą pateikia Ruparelia N. B. „Software development lifecycle models“. 2010.

2. Žiniatinklio aplikacijų testavimas

2. 1. Žiniatinklio aplikacija

Žiniatinklio aplikacija – aplikacija, kuri iškviečiama naršyklės pagalba per žiniatinklį.[10] Žiniatinklio aplikacija suteikia daugiau galimybių sąveikai su naudotoju, tačiau yra sudėtingesnė ir reikalauja daugiau išteklių aplikacijos vystymui.[11]

Kuo žiniatinklio aplikacija yra kitokia nei įprastinė aplikacija. Žiniatinklio aplikacija – tai programa, leidžianti naudotojui priimti ir siųsti informaciją iš nutolusios duomenų bazės per žiniatinklį, pasinaudojant naršykle duomenims atvaizduoti ir atlikti manipuliacijas su jais. Gaunami duomenys generuojami dinamiškai ir atvaizduojami naršyklėje po to, kai jie apdorojami serveryje. Žiniatinklio aplikacijos siunčia užklausas į serverį ir dinamiškai generuoja turinį, kuris pateikiamas aplikacijos naudotojui. Turinys yra sugeneruojamas standartiniu formatu (HTML), palaikomu visų naršyklių. Dažniausiai dėl dinamiškumo poreikio žiniatinklio aplikacijos plačiai naudoja Javascript'ą atvaizduojamų elementų modifikavimui (pavyzdžiui, formų turinio patikrinimas, tekstinių elementų parodymas, paslėpimas ir kt. veiksmus). Pagrindinė platforma, kurioje veikia žiniatinklio aplikacija – naudotojo interneto naršyklė, kurioje vykdomi skriptai, atliekamos užklausos į serverį ir atvaizduojami gauti duomenys, duomenų įvestis-išvestis vykdoma dirbant su naršykle.

Vienas iš svarbiausių žiniatinklio aplikacijų privalumų yra tas, kad jos funkcionuoja nepriklausomai nuo naudotojo operacinės sistemos ir naršyklės – žiniatinklio aplikacijos kodas programinis kodas saugomas ir valdomas nutolusiame serveryje, o naudotojui pateikiamas tik sugeneruotas dinamiškai rezultatas. Žiniatinklio aplikacijas lengva platinti (jos nereikalauja instaliavimo), kadangi vienintelis dalykas, kurio reikia, norint pradėti naudotis žiniatinklio aplikacija, yra interneto naršyklė. Tačiau tai yra ir didžiausia problema, kadangi naršyklės skirtingai gali apdoroti gaunamą kodą ir pateikti nevienodus rezultatus su ta pačia aplikacija.

Vienas iš didžiausių iššūkių, su kuriuo susiduriama testuojant žiniatinklio aplikacijas, kyla iš to, kad žiniatinklio aplikacijų turinys ir struktūra yra kintantys – dinaminiai. Žiniatinklio aplikacijų testavimo metu reikia atsižvelgti į kintančią aplikacijos formų struktūrą, kliento skriptų ir serverio programinio kodo veikimo rezultatus, kas daro poveikį testavimui.

Sukurti greitus, paprastus regresinius testus dinaminei žiniatinklio aplikacijai gali būti nemažų pastangų reikalaujantis iššūkis. Naudojant daugelį automatinio testavimo įrankių (įskaitant komercinius) testų kūrimas apima skriptų rašymą kuria nors programavimo kalba (arba

[10] Jazayeri M. Some Trends in Web Application Development. IEEE Computer Society, Washington, DC, USA, 199-213. ISBN: 0-7695-2829-5.

[11] Ten pat.

dažnai nuosavybine kalba) ir didelio kiekio klaidingų (angl. false-positives) pranešimų sugeneravimą.

Žiniatinklio aplikacijos testavimas yra sudėtingesnis už įprastinės aplikacijos testavimą. Įprastinės aplikacijos struktūra nekinta, tuo tarpu žiniatinklio aplikacijoje kinta struktūra ir duomenys. Žiniatinklio aplikacijos paprastai veikia lėčiau už įprastines aplikacijas dėl to, kad žiniatinklio aplikacijoje vykdomas duomenų perdavimas/gavimas tarp aplikacijos kliento ir serverinės dalių.

2. 2. Argumentai prieš ir už automatinį žiniatinklio aplikacijų testavimą

Aplikacijos testavimą galima atlikti rankiniu būdu arba automatiniu būdu. M. Fewster teigia, *kad remiantis Bach (1997) 85% defektų randami rankinio testavimo metu ir 60-80% defektų randami kuriant testus (Kaner, 1997). Net ir turint automatinius testus, didžioji dalis defektų randama, kai testai yra kuriami ir įvykdomi pirmą kartą (rankiniu būdu), užuot tuomet, kai jie yra automatizuojami. Nepaisant to, tai nereiškia, kad automatinis testavimas yra nenaudingas. Automatinio testavimo tikroji vertė atsiskleidžia ne naujų klaidų radime, o kuomet reikia atsakyti į klausimą, ar aplikacijoje pasikartoja jau anksčiau rastos klaidos.*[12]

Rankinis testavimas – atliekamas testuotojo, tiesiogiai naudojančio aplikaciją. Testuotojas išmėgina įvairius aplikacijos panaudojimo atvejus, patikrindamas gaunamus rezultatus, po testavimo metu patikrinto aplikacijos veikimo, užrašo testavimo rezultatus. Testuotojas rankinį testavimą atlieka daug kartų aplikacijos kūrimo metu, po programinio kodo pakeitimų ir kitose situacijose: kai aplikacijos konfigūracija ar veikimo aplinkos yra pakeičiamos. Žiniatinklio aplikacijos kūrimas reikalauja palaikymo automatinio testavimo priemonėmis. Rankinis testavimas yra imlus darbui[13]. Automatinio testavimo įrankio įdiegimas gali pakeisti pasikartojančias rankinio testavimo veiklas, atlikti jas automatiniu būdu.

Automatinio testavimo įrankis gali patobulinti rankinį testavimą, kadangi automatiškai atlieka tuos pačius rankiniam testavimui iš anksto užrašytus veiksmus, patikrina gaunamus rezultatus su tikėtinais ir užrašo rezultatą apie sėkmingą arba nesėkmingą testavimo atvejį ataskaitos formatu.

Kai tik žiniatinklio aplikacijai testuoti pritaikomas automatinio testavimo įrankis, testus galima kartoti neribotą kartų kiekį, o automatinius testus išvystyti taip, kad jie atliktų tai, kas neįmanoma rankiniame testavime (pavyzdžiui, panaudotų dinamiškai generuojamus duomenis testavimui ar įvairias duomenų kombinacijas, vykdytų testavimą nuolatos ir pan.). Todėl automatinis testavimas yra pagrindinis žiniatinklio aplikacijos testavimo proceso komponentas.

[12] Fewster M. „Software test automation“. Addison-Wesley Professional, 1999, p. 23, 206. ISBN: 0201331403.

[13] Dustin E., Rashka J., Paul J. „Automated Software Testing – Introduction, Management and Performance“. Addison-Wesley Professional, 1999, p. 17. ISBN: 0201432870.

Vis dėlto, pažymėtina, jog automatinis testavimas turi ne tik minėtus privalumus, bet ir trūkumus.

Galima išskirti šiuos automatinio testavimo privalumus:

1) automatiniai testai, kaip ir bet kuri programa, gali veikti nuolatos, be žmogaus įsikišimo, o jų vykdymas yra greitesnis[14] nei rankiniu metodu, todėl aplikaciją galima testuoti nuolatos ir kaskart gauti greitesnį atsakymą, ar aplikacija veikia korektiškai;
2) automatinius testus galima naudoti aplikacijos našumo testavimui – tuos pačius testus vykdant paraleliai iš didelio kiekio kompiuterių, simuliuojant didelę aplikacijos apkrovą;
3) automatinio testavimo metu galima rinkti objektyvius statistinius duomenis apie aplikacijos greitaveiką.

Automatinio testavimo trūkumai:[15,16]

- automatiniai testai, tai yra tos pačios testavimo programos, instrukcijos, kurioms (jei testai yra parašyti programavimo kalba) keisti ar atnaujinti reikalingas programavimas;
- kaip programinis kodas, automatiniai testai taip pat gali turėti klaidų;
- automatiniai testai dažniausiai suranda tik tas klaidas, kokias surasti jie yra suprogramuoti.

Galimybės automatizuoti viso testavimo, kuris atliekamas su žiniatinklio aplikacija, nėra,[17] todėl svarbu nustatyti, kuriuos testavimo atvejus reikia automatizuoti pradžioje. Didelė nauda iš automatinio testavimo gaunama, kai automatizuojami tokie testai:

- testai, pasikartojantys tarp skirtingų aplikacijos versijų;
- testai, kuriuose dažnai pasitaiko žmogiškosios klaidos;
- testai, kuriems reikia daug duomenų rinkinių;
- testai, tikrinantys dažniausiai naudojamą aplikacijos funkcionalumą;
- testai, kurių neįmanoma atlikti rankiniu būdu;
- testai, kurie atliekami keliose naršyklėse, keliose aplikacijos panaudojimo platformose, su keliomis konfigūracijomis;
- testai, kurių atlikimas rankiniu būdu reikalauja didelių pastangų ir laiko sąnaudų.

Argumentai už automatinį testavimą

Nors funkcinis testavimas gali būti atliktas visiškai rankiniu metodu, tačiau automatizavimas duoda testavimo procesui naudos: sumažina žmogaus klaidų, kurios pasitaiko atliekant testų

[14] Dustin E., Rashka J., Paul J. „Automated Software Testing – Introduction, Management and Performance“. Addison-Wesley Professional, 1999, p. 50. ISBN: 0201432870.

[15] Pastaba. Automatinio testavimo privalumus ir trūkumus atskleidžia Fewster M. „Software test automation“. Addison-Wesley Professional, 1999, p. 9-10. ISBN: 0201331403.

[16] Bach J. „Test automation snake oil“, 14th International conference on Testing Computer Software, 1997.

[17] Fewster M. „Software test automation“. Addison-Wesley Professional, 1999, p. 22. ISBN: 0201331403.

žingsnius ir užfiksuojant testų rezultatus (kiekvienas testuotojas padaro klaidų, atlikdamas tuos pačius pasikartojančius testavimo žingsnius, tuo tarpu automatiniai testai kiekvieną kartą pakartoja testavimo žingsnius vienodai ir užfiksuoja rezultatus vienodai). Automatizavimas pagerina testavimo skriptų pakartotinį panaudojimą su data-driven testavimo atvejais.

Apibendrinant galima teigti, kad automatinis testavimas turi konkrečius pranašumus gerinant testavimo procesą. Automatinis testavimas suteikia:

- nuolatinį regresinį testavimą;
- greitą grįžtamąjį ryšį iš testuotojų programuotojams;
- iš esmės neribotas testavimo atvejų vykdymo iteracijas;
- dera su judraus (agile) ir ribinio (extreme) aplikacijų vystymo metodologijomis;
- pateikia sistemingą testavimo atvejų dokumentaciją;
- pateikia ataskaitas apie defektus;
- atskleidžia defektus, nepastebėtus testuojant rankiniu būdu.

Vienas iš judriojo (agile) aplikacijų vystymo modelio privalumų yra trumpas laikas iki naudotinos (be klaidų) aplikacijos paruošimo. Kuo anksčiau paruošti minimaliai veikiančią aplikacijos versiją per trumpą laiką galima tik atidžiai vykdant testavimą, eliminuojantį klaidas. Testavimo vykdymas automatiniu būdu leidžia sumažinti[18] aplikacijos testavimo laiką, per kurį surandamos naujos klaidos.

Automatinio testavimo skriptų pakartotinis panaudojimas yra vertingas būdas atlikti regresinius testavimus aplikacijoje, užtikrinant, kad naujos savybės arba klaidų pataisymai nesugadina esamo funkcionalumo.

Automatinis testavimas paprastai papildo rankinį testavimą, tiriamąjį testavimą (angl. exploratory), kurie yra vertingi ir efektyvūs testuojant naują aplikacijos funkcionalumą ir įdiegtas naujas galimybes. Žiniatinklio aplikacijai pilnai ištestuoti reikia atlikti tuos pačius automatinius testus keliose platformose (Windows, Linux, Macos), keliose naršyklėse (Internet Explorer, Firefox, Opera), keliose aplikacijos versijose (realioji, bandomoji), keliuose įrenginiuose (asmeninis kompiuteris, telefonas), o automatiniai testai tokiu atveju gali būti vykdomi paraleliai su įvairiomis konfigūracijomis.

Automatinio testavimo kritika

Automatinis testavimas neišsprendžia visų testavimo problemų, nepatenkina visų testavime iškylančių poreikių. Iš tikrųjų automatinis testavimas iškelia ir keletą naujų problemų. Tuo metu, kai daugelis naujų problemų gali būti išspręstos laikantis disciplinuoto automatinio testavimo, vis dėlto yra keletas dalykų, į kuriuos reikia atkreipti dėmesį ir išnagrinėti:

[18] Dustin E., Rashka J., Paul J. „Automated Software Testing – Introduction, Management and Performance“. Addison-Wesley Professional, 1999, p. 5. ISBN: 0201432870.

1. Automatinio testavimo skriptai tampa neaktualūs, kadangi testuojama aplikacija nuolatos tobulinama – kai automatiniai testai rašomi nenaudojant objektinio (arba objektiškai-orientuoto) programavimo principų, tikėtina, kad automatiniai skriptai turės pasikartojantį programinį kodą. Įprastas testavimo procesas apima kelių dešimčių ar net šimtų automatinių testavimo skriptų įvykdymą. Pavyzdžiui, paimkime dokumentų valdymo sistemą, kurios automatinio testavimo skriptų kiekis gali viršyti kelias dešimtis. Tikėtina, kad duotuose automatinio testavimo skriptuose bus programinis kodas, naudojantis kelias aplikacijos savybes, vien jau tam, kad privestų aplikaciją iki testavimui reikalingos būsenos (užpildytų reikiamais duomenimis, atvertų testuojamus langus). Pavyzdžiui, dokumentų valdymo sistemos savybė „dokumento redagavimas“. Tam, kad išbandytume duotą savybę turime atverti aplikacijos langą, prisijungti kaip aplikacijos naudotojas, surasti dokumentą redagavimui, dokumentą atverti, kiekvieną laukelį, kurio duomenis keisime, turime suaktyvinti, įrašyti naujus duomenis, galiausiai aplikacijoje turime paspausti mygtuką „Išsaugoti“. Vien duotas testas atlieka septynis veiksmus su aplikacija, neskaitant kiekvieno atlikto veiksmo sėkmingumo patikrinimo, turėsime ganėtinai daug mažų žingsnių vieno testo atlikimui. Bet koks aplikacijos patobulinimas, pakeitimas duotose savybėse sukelia poreikį modifikuoti duoto testo skriptą, o tuo pačiu ir visus kitus skriptus, naudojančius aprašytas savybes.

Automatiniai testai taip pat yra programinis kodas, reikalaujantis atnaujinimų, pataisymų keičiantis testuojamai aplikacijai, dėl to paaiškėja vienas automatinių testų trūkumas – juos reikia valdyti kaip programinį kodą.[19]

Automatiniame žiniatinklio aplikacijų testavime tinkamas sprendimas galėtų būti: apjungti testavimo atvejį ir testavimo skriptą, kad liktų tik vienas. Kaip tai padaryti? Automatinio testavimo įrankyje Cucumber tai realizuota tokiu būdu: kiekvienas testas yra parašomas tekstu, kuris testavimo metu automatiškai interpretuojamas ir vykdomas, tačiau kiekvienam testavimo atvejui reikia parašyti programinį kodą, kuris vykdytų testavimo atvejyje aprašytus veiksmus.

Pavyzdžiui, Cucumber automatinio testavimo įrankio testavimo atvejis:

```
Savybė:
'Kai atidarau Google Search tinklapi ir atlieku paieska, tureciau matyti rezultatus.'
  Scenarijus:
    Duota kad as atidariau Google tinklapi
    Kai as irasau "mif.vu.lt" i paieskos laukeli
    Ir paspaudziu Paieskos mygtuka
    Tada tureciau rasti tinklapi "Matematikos ir Informatikos fakultetas"
```

[19] Kaner C. „Improving the Maintainability of Automated Test Suites“. Software QA, 4(4), 1997, p. 2.

Turės programinį kodą, vykdantį nurodytus veiksmus:

```
Given /^kad as atidariau Google tinklapi$/ do
  @browser = Selenium::WebDriver.for :ie
  @browser.navigate.to 'http://www.google.com'
end
When /^as irasau "(.*)" i paieskos laukeli$/ do |item|
  @browser.find_element(:name, 'q').send_keys(item)
end
And /^paspaudziu Paieskos mygtuka$/ do
  @browser.find_element(:name, 'btnG').click
end
Then /^tureciau rasti tinklapi "(.*)"$/ do |item|
  wait = Selenium::WebDriver::Wait.new(:timeout => 5000)
  wait.until { @browser.find_element(:id => "ires").displayed? }
  @browser.find_element(:partial_link_text => item).click
  @browser.title.should include(item)
  @browser.quit
End
```

2. Lankstumas ir galimybės dažnai paaukojamos dėl skriptų rašymo supaprastinimo. Daugelis prieinamų automatinio testavimo įrankių kaip pagrindinę savybę pristato tai, kad jie padaro automatinių testų rašymą greitą ir paprastą.[20] Dėl to komerciniai automatinio testavimo įrankiai dažniausiai remiasi supaprastintomis skriptų programavimo kalbomis (sukurtomis įrankių gamintojų), kurios ne visuomet sukuriamos atsižvelgiant į poreikį panaudoti išorinius išteklius arba viešai vystomas ir palaikomas atvirojo kodo bibliotekas. Komercinių produktų siūlomos skriptų kalbos dažnai apriboja abstrakcijos ir pakartotinio panaudojimo galimybes arba mažų mažiausiai neskatina geros programavimo tvarkos siekiant skriptų aptarnavimo sąnaudų sumažinimo ilguoju periodu (aplikacijos gyvavimo cikle). Siekiant padaryti skriptų rašymą paprastu, kai kurie automatinio testavimo įrankiai praranda pilnos programavimo kalbos suteikiamus privalumus, todėl dažnai skriptai tinkamai gali ištestuoti tik paprastus atvejus, neišnaudojant automatinio testavimo suteikiamų privalumų. Dėl lankstumo trūkumo dažnai aplikacijoms testuoti naudojami keli įrankiai: vienas įrankis aplikacijos funkcionalumo testavimui, kitas įrankis apkrovos testavimui ir t.t. Be to, kiekvienas testuotojas su nauju testavimo įrankiu turi mokytis naujas testavimo skriptų rašymo kalbas, kas taip pat nepalengvina aplikacijos testavimo proceso.

3. Poreikis naudoti nuosavybines (proprietary) technologijas. Daugelis testavimo įrankių reikalauja išmokti nuosavybinę kalbą (proprietary language) arba technologiją. Tai apsunkina galimybes rasti tinkamų testuotojų, patyrusių specialistų, mokančių duotą kalbą ir žinančių, kaip

[20] Dustin E., Rashka J., Paul J. „Automated Software Testing – Introduction, Management and Performance“. Addison-Wesley Professional, 1999, p. 35. ISBN: 0201432870.

dirbti su automatinio testavimo įrankiu. Tai apriboja galimybes pasirinkti testavimo įrankį (aplikacijos testavimui pasirenkamas toks įrankis, kad būtų prieinami specialistai, mokantys jį naudoti). Siauras testavimo įrankio pritaikomumas atbaido esamus specialistus nuo gilesnių žinių apie naudojamą įrankį įgijimo.

4. Automatiniai skriptai yra sunkiai skaitomi, o išorinė dokumentacija ilgainiui nebesutampa su jų turiniu. Tiek automatinio testavimo skriptams, tiek rankiniam testavimui naudojami testavimo atvejai, kurie dokumentuoja žingsnius, atliekamus, kad būtų įvykdyti reikalavimai. Testavimo atvejai yra kiekvieno testavimo plano pagrindas. Dažniausiai testavimo skriptai tvarkomi atskirai nuo testavimo atvejų. Automatiniai skriptai ir testavimo atvejai ilgainiui nebesutampa ir nebelieka žinančių, kas atitinka realią situaciją, apibrėžia teisingesnius reikalavimus aplikacijai – testavimo atvejis ar testavimo skriptas.

5. Automatinius skriptus pritaikyti kelioms testavimo aplinkoms gali būti sudėtinga. Daugelis testavimo įrankių neatsižvelgia į aplikacijos kūrimo ciklą. Kol galutinė aplikacijos versija pasiekia naudotoją, kuriama aplikacija pereina per kelis testavimo etapus: alpha versijos testavimas, beta versijos testavimas, našumo testavimas. Kadangi labai sudėtinga redaguoti didelį kiekį testavimo skriptų, tai daugeliu atvejų automatinio testavimo skriptai paleidžiami tik vienoje aplinkoje, testuoja tik vieną aplikacijos versiją, tokiu būdu sumažėja galimybė atlikti regresinį testavimą vystant aplikaciją.[21]

Pasiūlymai

- Testavimo atvejus ir testavimo skriptus reikėtų sutapatinti. Naudoti tik testavimo skriptus.
- Nesaugoti skriptuose informacijos, susijusios su testuojamos aplikacijos aplinka. Vykdyti testavimo skriptus įvairiose aplikacijos aplinkose be skriptų modifikavimo.
- Remtis data-driven testais – įgalinti testavimo skriptus pasiimti duomenis iš bet kokio išorinio šaltinio (duomenų lentelės XLS arba CSV failuose, duomenys iš duomenų bazės). Tai leistų ištestuoti vieną aplikacijos savybę vienu skriptu, tačiau su daugeliu testavimo duomenų reikšmių, paduodamų į tą patį testavimo skriptą.

Kada automatizuoti testavimą?

Norint atsakyti į klausimą „Kada testavimo atvejo automatizavimas duoda naudos, viršijančios sąnaudas, testavimo atvejo automatizavimui?“, galima remtis stebėjimų rezultatais[22], tvirtinančiais, kad jei testavimo atvejis bus panaudotas dešimt ar daugiau kartų, tai toks

[21] Pastaba. Automatinio testavimo privalumus ir trūkumus atskleidžia Fewster M. „Software test automation“. Addison-Wesley Professional, 1999, p. 9-10. ISBN: 0201331403.

[22] Berner S., Weber R., Keller R. K. Observations and lessons learned from automated testing. ACM, New York, NY, USA, 2005, p. 571-579. ISBN: 1-58113-963-2.

testavimo atvejis turėtų būti automatizuotas, o kiekvienas sukurtas testavimo atvejis per savo gyvavimą yra panaudojamas mažiausiai 5-10 kartų, o mažiausiai 25% testavimo atvejų yra panaudojami daugiau nei 20 kartų. Todėl reikėtų atsižvelgti į tai, kad planuojant panaudoti testavimo atvejį daugiau nei kelis kartus – prasminga jį automatizuoti.

Atliktas tyrimas[23] patvirtina teiginius[24], kad automatinis testavimas negali visiškai pakeisti rankinio testavimo. Tuo remiantis[25], galima panaudoti keturis testų automatizavimo žingsnius: 1) numatyti darbus nuolatiniam testavimo atvejų tobulinimui ir priežiūrai; 2) numatyti testų automatizavimo tikslą – trumpesnis PĮ kūrimo ciklas, geresnė PĮ kokybė; 3) apjungti kelis metodus (sisteminių testų automatizavimas, funkcinių testų automatizavimas, integravimo testų automatizavimas, unit testų automatizavimas); 4) tikrinti, ar testų automatizavimo tikslai pasiekti, ir nuolatos tobulinti automatinius testus.

Ką automatizuoti?

Tyrimų[26,27] išvadose kritikuojamas testavimo automatizavimas, kuomet testuojama panaudojant grafinę naudotojo sąsają, kadangi grafinė naudotojo sąsaja linkusi dažnai keistis ir testavimo skriptai turi būti atitinkamai modifikuojami, pritaikant juos prie tų pokyčių. Negalima patikrinti sistemos būsenos ir sistemos atsakymo (angl. system response) – dažnai sistemos būsena yra nematoma, grafinės vartotojo sąsajos lygmenyje dirbantys automatiniai testai randa tik tas klaidas, kurias testo autorius numatė surasti. Tačiau negalime nepaisyti aplinkybės, kad žiniatinklio aplikacijos yra prieinamos tik per naudotojo sąsają ir vienintelis tiesioginis būdas patikrinti jų funkcionalumą yra testavimas per vartotojo sąsają.

Automatinis testavimas dažniausiai remiasi tokiomis testavimo skriptų metodikomis[28]:

Skriptas	Apibūdinimas	Esminis trūkumas	Esminis privalumas
Linijinis	Tai rankinio testavimo veiksmų tiesioginio užrašymo rezultatas.	Naudoja įkoduotas reikšmes, kūrimas ir atnaujinimas imlus darbui.	Pats paprasčiausias, nereikalauja išankstinio parengimo, galima naudoti su „capture/replay“ įrankiais.

[23] Martin D., Rooksby J., Rouncefield M., Sommerville I. 'Good' Organisational Reasons for 'Bad' Software Testing. IEEE Computer Society, Washington, DC, USA, 2007, p. 602-611. ISBN: 0-7695-2828-7.

[24] Berner S., Weber R., Keller R. K. Observations and lessons learned from automated testing. ACM, New York, NY, USA, 2005, p. 571-579. ISBN: 1-58113-963-2.

[25] Ten pat.

[26] Ten pat.

[27] Memon A. M. 2008. Automatically repairing event sequence-based GUI test suites for regression testing. ACM Trans. Softw. Eng. Methodol. 18, 2, Article 4 (November 2008).

[28] Fewster M. „Software test automation“. Addison-Wesley Professional, 1999, p. 82-117. ISBN: 0201331403.

Struktūrinis	Tai linijinis testavimo skriptas, kuris yra praturtintas vykdymo kontrolės instrukcijomis.	Naudoja įkoduotas reikšmes.	Gali panaudoti vykdymo kontrolę (if, else), ciklus dėl to universalesnis už linijinius skriptus.
Bendras	Testavimo skriptas, kurį panaudoja keli testavimo atvejai.	Tinkamas mažoms aplikacijoms testuoti, kur testų nedaug. Kai bendrų skriptų yra daug, sudėtingėja jų valdymas.	Eliminuoja pasikartojančius veiksmus tarp skriptų.
Data-driven	Testavimo skriptas, kuriame panaudojami kintantys duomenys saugomi atskirai (duomenų bazėje, CSV failuose ar kt.)	Mažoms aplikacijoms testuoti, kai testavimo atvejų nėra daug, parengti data-driven testavimą gali būti didesnis uždavinys už patį testavimą.	Tas pats skriptas gali būti panaudotas vykdant skirtingus testus.
Raktažodinis	Skriptas apjungiantis data-driven techniką ir testo veiksmų užrašymą raktažodžiais (instrukcijomis), kurie nurodo atliekamus veiksmus.	Raktažodinio testavimo galimybės parengimas reikalauja programavimo.	**Raktažodinis skriptas gali būti vystomas ir vykdomas nepriklausomai nuo pasirinkto automatinio testavimo įrankio ir testavimo platformos.**

Raktažodinis testavimas (angl. keyword testing) – testai sukuriami nepriklausomai nuo pasirenkamo automatinio testavimo įrankio. Dėl to, kai turima sukūrus daug testavimo skriptų, juos galima pritaikyti kitiems įrankiams. Tokia savybė aktuali žiniatinklio aplikacijų automatiniam testavimui, kadangi turime ne vieną tinkamą automatiniam testavimui įrankį (Selenium, Watir, Sahi ir kt.), ir jų sukuriama daugiau. Raktažodinis testavimas taip pat labiau orientuotas į testuotojus, nes raktažodinio testavimo atveju testavimo įrankis pritaikomas prie testavimo proceso (parašomas raktažodžių vykdymą realizuojantis programinis kodas), o ne atvirkščiai.

Be to, raktažodinių testų kūrimas nereikalauja programavimo kalbų žinių; raktažodinio testavimo atvejis – tai yra paprastų raktažodžių, su nurodytu veiksmu, rinkinys. Raktažodinio testavimo pagalba galima imituoti klavišo paspaudimus, mygtukų spustelėjimus, pasirinkti meniu elementą, iškviesti objektų metodus ir tikrinti objektų savybes. Raktažodinis testavimas yra dažna alternatyva programavimo kalba parašytiems automatiniams testavimo skriptams. Raktažodinių testų aprašymus, priešingai programavimo kalba (dažniausiai Java, C#, Ruby, Python) parašytiems skriptams, gali lengvai skaityti, suprasti ir redaguoti bet kurie naudotojai, kuriantys automatinius testus. Pavyzdžiui, programavimo kalba užrašytos komandos:

```
el_value = @driver.find_element(:id, 'Control6232')[:value]
assert(!el_value.empty?)
```

gali būti užrašomos kaip instrukcija, kuriai naudojama XPATH:

```
<instruction>
        <action>
                assert
        </action>
        <target>
                count(*[@id='Control6232' and text()!=""])>0
        </target>
</instruction>
```

Su raktažodinio testo koncepcija supažindina sukurtas ir magistro darbo prieduose pateikiamas testavimo įrankis „Instructable TestMaker“, panaudojantis instrukcijas: action (nurodo atliekamą testavimo veiksmą) – gali turėti tokias reikšmes „Navigate“, „Settext“, „Click“, „Assert“; target – identifikuoja testuojamą žiniatinklio aplikacijos aktyvaus lango elementą; parameters – tekstinė reikšmė, kuri panaudojama atliekant action instrukcijoje nurodytą veiksmą.

Raktažodinis testas, tai yra raktažodžių seka, kur kiekvienas raktažodis atitinka vieną funkcinio testavimo veiksmą (teksto įvedimą iš klaviatūros, pelės paspaudimą, kt.) Kiekvienas funkcinio testavimo veiksmas gali būti išreiškiamas raktažodinio testo instrukcija. Funkcinio testo sukūrimas atitinka reikiamų testavimo veiksmų išvardinimą raktažodžiais ir parametrų tiems raktažodžiams uždavimą. Kadangi raktažodinis testavimas yra lanksčiausias[29] būdas implementuoti automatinį funkcinį testavimą, tai raktažodiniai testai naudojami funkciniam testavimui atlikti.

Raktažodiniai testai susideda iš operacijų, kurios atlieka įvairius veiksmus, tokius kaip: naudotojo veiksmų modeliavimas testuojamuose objektuose, įvairių patikrinimo veiksmų atlikimas, testuojamų objektų metodų iškvietimas ar savybių tikrinimas. Raktažodiniai testai gali būti lengvai kuriami vizualinėmis priemonėmis, taip pat užrašyti su veiksmų naršyklėje įrašymo savybę („capture/record“) turinčiu testavimo įrankiu (Selenium-IDE įskiepis, Push To Test testavimo įrankis, Test Complete Automated Testing ir kt.)

Raktažodžiais paremtas testavimas yra dažniausiai įgyvendinama testavimo programos savybė, kuri dėl testavimo paprastumo yra gera alternatyva testavimo skriptų užrašymui programavimo kalba. Naudojant raktažodinį testavimą testuotojas gali lengvai kurti funkcinius testus ir automatizuoti testavimą.

[29] Fewster M. „Software test automation“. Addison-Wesley Professional, 1999, p. 89. ISBN: 0201331403.

Data-driven testavimas

Data driven testai – tokie funkciniai testai, kuriuose atliekama ta pati testavimo veiksmų seka panaudojant kintamus duomenis. Tokios priemonės kaip IBM Rational Functional Tester automatiškai užfiksuoja testų užrašymo metu įvedamus duomenis ir paruošia testus naudoti su kintamais duomenimis. Naudojant lentelės pavidalo duomenų šablonus, testuotojas gali surašyti įvairius duomenų rinkinius testui. Gauti testavimo duomenų rinkiniai vėliau panaudojami vykdant testą pakartotinai. Tokiu būdu testavimo skriptas gali būti panaudotas kelis kartus be pačio skripto perrašymo. Apibendrinant teigtina, jog data-driven testavimo atvejai sumažina pasikartojimus testavimo plane. Vieno testavimo skripto parašymas grupei testavimo atvejų padaro paprastesniu testavimo skriptų atnaujinimą. Data driven testavimo skriptai yra pakartotinai panaudojami, jie reikalauja tik naujų duomenų pridėjimo. Iš testuotojo perspektyvos data driven testavimas leidžia išsaugoti duomenų kombinacijas sąraše ir vykdyti nekintantį testavimo skriptą su kintančiais duomenimis. Data driven testavimo skriptų privalumas yra tai, kad jie yra šablonai testavimo atvejams.

Kuo ypatingas žiniatinklio aplikacijos testavimas?

Žiniatinklio aplikacija, tai kitaip nei tipinės aplikacijos veikianti programa, paprastai žiniatinklio aplikacijos loginė dalis įvykdoma serveryje, o naudotojas gauna tik įvykdymo rezultatą, kuris atvaizduojamas naršyklėje. Tuo tarpu įprastinės aplikacijos vykdomos tame pačiame kompiuteryje, kuriuo dirba naudotojas. Žiniatinklio aplikacijose yra tam tikri standartiniai elementai, kurie panaudojami visose aplikacijose, todėl žiniatinklio aplikacijų testavimas gali būti standartizuotas. Dažniausiai žiniatinklio aplikacijos testavimas suvedamas į nuorodų (ar nuorodos veikia) ir struktūrinių elementų patikrinimą, duomenų įvedimo formų (kokius duomenis priima, įvairios duomenų kombinacijos) veikimo patikrinimą.

Siekiama, kad žiniatinklio aplikacija veiktų vienodai keliose naršyklėse, todėl testavimo metu žiniatinklio aplikacija testuojama su keliomis naršyklėmis. Tai padidina testavimo apimtis.

Žiniatinklio aplikacijos nereikia instaliuoti, todėl jos atnaujinimas yra labai paprastas – naudotojas kiekvieną kartą atidaręs aplikacijos langą dirba su naujausia jos versija. Dėl atnaujinimo paprastumo, žiniatinklio aplikacijos yra atnaujinamos dažnai, todėl jų atnaujinimo ciklas yra trumpas. Dėl dažno atnaujinimo žiniatinklio aplikacijos turi būti testuojamos kaip ir visos aplikacijos po kiekvieno atnaujinimo (regresinis testavimas). Kadangi regresinį testavimą atlikti po kiekvieno aplikacijos atnaujinimo rankiniu būdu gali užimti daug laiko, tai žiniatinklio aplikacijoms testuoti naudojami automatinio testavimo įrankiai, kurie dalį testavimo atlieka automatiniu būdu.

Magistro baigiamajame darbe buvo įvertinti ir palyginti žiniatinklio aplikacijų automatinio testavimo įrankiai.

3. Automatinio testavimo įrankiai

Testavimo įrankiai

Esminės automatinio testavimo įrankio savybės:[30]

- Įvairių testavimo tipų palaikymas. Įrankiai pateikia galimybes atlikti funkcinį testavimą, našumo testavimą, apkrovos testavimą, unit testavimą. Dažnai galima tą patį funkcinį testą atlikti ne tik keletą kartų, tačiau ir vienu metu atlikti iš kelių kompiuterių ar iš vieno kompiuterio vienu metu paleisti keletą testų. Tokiu būdu simuliuojant žiniatinklio aplikacijos didesnį apkrovimą galima testuoti, kaip aplikacija veikia, esant didesniam vienu metu su ja dirbančių naudotojų kiekiui.
- Įvairių naršyklių palaikymas (angl. cross-browser). Žiniatinklio aplikacijų testavimui viena svarbiausių užduočių yra patikrinti, ar aplikacija vienodai veikia visose pagrindinėse naršyklėse (Internet Explorer, Firefox, Chrome, Opera, Safari).
- Automatinių testų kūrimo galimybė neatliekant programavimo darbų. Viena iš svarbiausių testavimo įrankių užduočių – supaprastinti automatinio testavimo skriptų kūrimą. Todėl pasitelkiamas raktažodinis testavimas, kuris turi paprastą struktūrą ir lengvai sukuriamas. Raktažodinis testas susideda iš operacijų, kuriomis simuliuojamas naudotojo darbas su testuojama aplikacija, pavyzdžiui, pelės mygtukų paspaudimai, tekstų įvedimai klaviatūra ar kitų automatinio testavimo veiksmų atlikimas: testuojamos aplikacijos paleidimas, pranešimų į testavimo žurnalą įrašymas, kitų testų paleidimas (pavyzdžiui, žemesnio lygmens – unit testų įvykdymas). Testuotojai neprivalo turėti programavimo įgūdžių, kad sukurtų funkcinį testą, naudojant raktažodžius. Taip pat galima lengvai įrašyti raktažodžiais paremtą testą tuo metu, kai testuotojas naudoja aplikaciją. Raktažodiniai testai yra būdas greitai sukurti funkcinius testus.
- Ataskaitų eksportavimas – automatinio testavimo įrankiai turėtų naudotojams pateikti galimybę išsaugoti testavimo ataskaitą. Pavyzdžiui, tai naudinga, kai automatinis testavimas be testuotojo priežiūros arba kai testavimo ataskaitos yra peržiūrimos kompiuteryje, neturinčiame suinstaliuoto testavimo įrankio.
- Testavimo eigos žurnalas – automatinio testavimo įrankis turėtų užrašyti visus testavimo veiksmus ir pateikti detalią ataskaitą apie veiksmų įvykdymo rezultatus. Paprastai testavimo žurnale būna tokio tipo duomenys: klaidų pranešimai, įspėjimai ir informaciniai pranešimai, pasirodę testavimo metu. Naudojant testavimo žurnalą, galima lengvai nustatyti, kas testavimo metu nesuveikė ir nustatyti klaidos priežastį.

[30] Pastaba. Automatinio testavimo įrankių savybes aprašo, analizuoja ir detalizuoja Fewster M. „Software test automation“. Addison-Wesley Professional, 1999. ISBN: 0201331403.

- Kelių duomenų šaltinių tipų palaikymas – atliekant funkcinį testavimą dažnai simuliuojamas naudotojo duomenų įvedimas į testuojamos žiniatinklio aplikacijos formas. Išsamus testavimas turi patikrinti įvairių duomenų įvedimą į tekstinius laukus ir kad testuojama aplikacija įvairius duomenis apdoroja teisingai. Todėl automatizavus funkcinį testą, dažnai atliekamas kelių duomenų rinkinių pritaikymas tam testui. Pavyzdžiui, testavimo duomenys būna imami iš duomenų bazės ar XML failų. Tokiu būdu užtikrinamas testavimo veiksmų ir testavimo duomenų atsiejimas, kas leidžia tą patį testą atlikus su skirtingais duomenimis, patikrinti aplikaciją išsamiau.

Funkcinis testavimas yra plačiausiai naudojamas žiniatinklio aplikacijų metodas. Egzistuojantys įrankiai (pvz., Push To Test Maker, Ranorex, Selenium-IDE) paremti veiksmų užrašymo (capture/replay)[31] savybėmis: įrankiai įrašo naudotojo veiksmus naršyklėje ir vėliau juos pakartoja atliekant regresinį testavimą. Automatinio žinatinklio aplikacijų testavimo įrankio savybės „Capture/replay“ panaudojimas turi būti minimalus, ši savybė daugiau skirta tiriamajam (angl. exploratory) testavimui. Šis metodas turi tokių trūkumų[32,33]:

- Įkoduotos reikšmės (angl. hard-coded). „Capture/replay“ panaudojimas sugeneruoja testavimo skriptą remiantis naudotojo sąveika su žiniatinklio aplikacija (įrašomi visi duomenys įvesti ar gauti per naudotojo sąsają). Įkoduotos duomenų reikšmės gali padaryti testavimo skriptą nepanaudojamu, kai žiniatinklio aplikacijos testuojama dalis pasikeičia, pvz., tekstiniai laukai ar kiti elementai pervardinami ar pakeičiami.
- Pasikartojantys keliuose skriptuose veiksmai. Naudotojas aplikacijoje atlieka įvairius veiksmus, kurie pasikartoja. Pavyzdžiui, testuojant aplikaciją, kur reikia užpildyti formą, reikia atlikti tą patį užpildymą kelis kartus, bet su skirtingais duomenimis. Gauname, kad tą patį testą reikės įrašinėti kelis kartus, nors iš esmės atliekami tie patys veiksmai.

3. 1. Įrankių palyginimas

Automatinio testavimo įrankius įvertinome pagal tokias savybes: testų įrašymas (capture/replay), kelių naršyklių palaikymas, objektiškai-orientuotų skriptų palaikymas, išorinių bibliotekų palaikymas, ataskaitų generavimas, aplikacijos elementų identifikavimo galimybės (naudojant XPATH, CSS arba pagal atributus ID, NAME), testavimo būdas (testuoja kaip naudotojas – per atidarytą naršyklės langą, imituoja naršyklę), pop-up ir kt. langų palaikymas, Frame palaikymas, testų panaudojimas našumo testavimui. Sudėjus visus vertinimus, gauti rezultatai parodė, kad trys (pagal Pareto principą imame 20% geriausių įrankių iš 14 gavusių

[31] Fewster M. „Software test automation“. Addison-Wesley Professional, 1999. ISBN: 0201331403.

[32] Dustin E. „Effective software testing: 50 specific ways to improve your testing“. Addison-Wesley Professional, 2002, p. 189. ISBN: 0201794292.

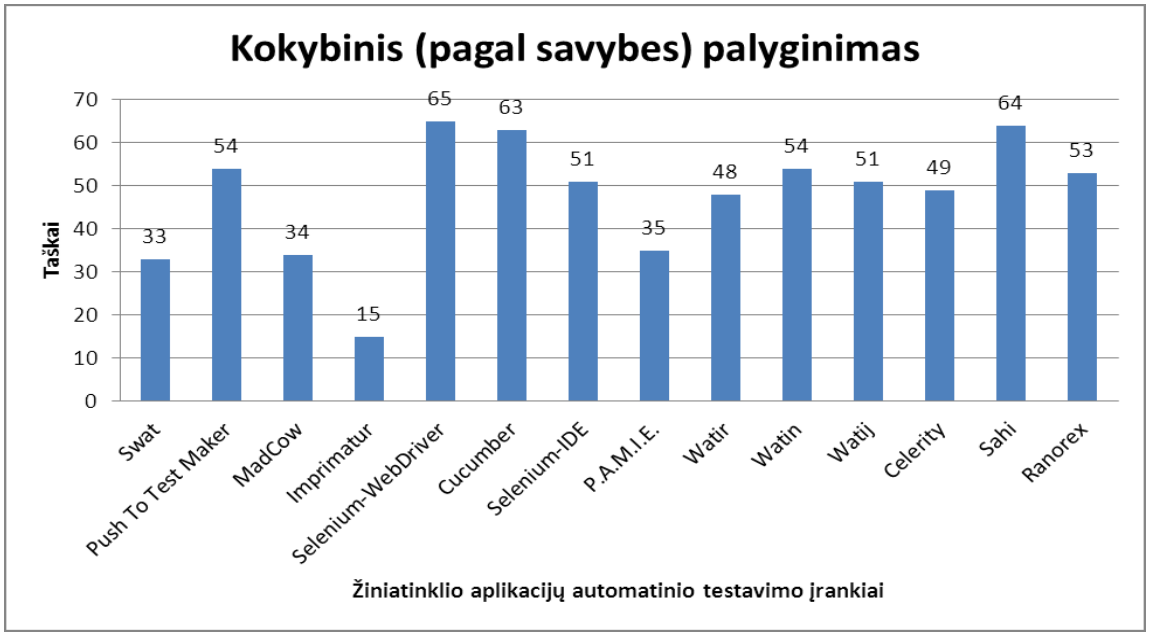
[33] Hayes L. G. „Automated Testing Handbook“. Software Testing Institute, 2004, p. 111. ISBN: 0970746504.

įvertinimus) geriausi įrankiai yra Selenium-WebDriver, Cucumber, Sahi. Geriausiai įvertintų įrankių negalime vertinti vienareikšmiškai dėl to, kad Selenium-WebDriver ir Sahi yra ne tos pačios klasės testavimo įrankiai kaip Cucumber, kuris yra testų užrašymo kalbą pateikiantis įrankis. Automatinio testavimo įrankiui Cucumber galime panaudoti Selenium-WebDriver, Watir ar kitą naršyklės valdymo įrankį, kad vykdytų testus.

Kiti įrankiai: „Push To Test Maker“ yra komercinis automatinio testavimo įrankis; Selenium-IDE yra testų kūrimo įrankis, nepritaikytas dažnam testų paleidimui; Watin – įrankis neturi cross-browser savybės (veikia tik su „Internet Explorer“ naršykle). Realizuojant raktažodinio testavimo modelį paaiškėjo, kad Sahi nepalaiko XPATH (įgalinus XPATH palaikymą pasinaudojant Javascript plėtiniu „Javascript-XPATH“, paaiškėjo, kad įrankis Sahi veikia nenašiai – testo atlikimas sulėtėjo iki keleto minučių, kai su Selenium-WebDriver tas pats testas veikė keletą sekundžių). Dėl to, nors Sahi ir surinko tiek taškų, vertinant pagal savybes yra antroje vietoje, tačiau Sahi netinkamas panaudoti siūlomame modelyje – nepalaiko XPATH.

Realizuotame raktažodinio testavimo modelyje panaudotas toks naršyklės valdymo įrankis – Selenium-WebDriver. Dėl modelio abstrakcijos modelyje galima panaudoti ir kitą naršyklės valdymo įrankį, todėl įtrauktas ir Watir. Be to, esant poreikiui į modelį galima įtraukti kitą automatinio testavimo įrankį.

Paveikslas 5. Įvertinus ir palyginus automatinio testavimo įrankius gauti tokie rezultatai:



3. 2. Įrankis automatinio testavimo modeliui

Modelį realizuojančioje demonstracinėje programoje panaudojome Selenium-WebDriver žiniatinklio aplikacijų automatinio testavimo įrankį. Selenium-WebDriver yra du automatinio testavimo įrankius apjungiantis API, kuriame yra Selenium ir WebDriver. Selenium – parašytas Javascript‘u, dėl ko yra apribotas naršyklės saugumo modeliu, kadangi naršyklės vykdomam

Javascript‘ui naudojamas griežtas saugumo modelis siekiant apsaugoti naudotoją nuo kenksmingų skriptų, tai ir Selenium įrankis turi su tuo susijusius apribojimus.

WebDriver – testavimui daryti naudoja kitokį metodą nei Selenium. Užuot veikęs kaip Javascript aplikacija naršyklėje, WebDriver‘is naudoja kiekvienai naršyklei unikalų mechanizmą, kad valdytų naršyklę. Naršyklei Firefox, Webdrive‘is veikia kaip plėtinys, naršyklei Internet Explore WebDriver‘is panaudoja Internet Explorer Automation priemones, kas leidžia valdyti naršyklę tuo būdu, kuris yra priimtiniausias. WebDriver yra greitas paprastas įrankis automatiniam žiniatinklio aplikacijų testavimui.

Magistrinio darbo eigoje pagal savybes buvo išbandyti ir įvertinti[34, 35] komerciniai ir atvirojo kodo automatinio testavimo įrankiai, sukurtas ir realizuotas automatinio testavimo programos modelis, įgyvendinantis raktažodinį testavimą funkciniam žiniatinklio aplikacijos testavimui.

Testavimo įrankis aplikacijos langus (gautą HTML kodą) traktuoja kaip elementų rinkinį, sudarytą iš elementų, kuriems neatsižvelgiant į tipą (<input>, <a>, , ir kt.), galima taikyti tokius veiksmus:

- Patikrinti elemento atributus;
- Suaktyvinti elementą („click“ veiksmas);
- Pakeisti elemento atributo „value“ reikšmę, jei atributą „value“ elementas turi.

Modelį galima plėtoti ir pritaikyti: kurti naujus raktažodžius; pajungti kitą naršyklės driver‘į.

Remiantis klaidų klasifikacija,[36] galima identifikuoti šias klaidas:

Lentelė 2. Identifikuojamos klaidos.

Klaidos klasė	Apibūdinimas	Ar tikrinama	Kaip įrankis testuoja tokias klaidas
Duomenų saugojimo klaidos	Klaidos, liečiančios tai, kaip aplikacija išsaugo duomenis.	Ne	Tai white-box priėjimo reikalaujanti klaida. Kadangi remiamasi black-box metodu, tokios klaidos neidentifikuojamos.
Loginės klaidos	Aplikacijos kodo loginės klaidos, liečiančios duomenų ir srauto valdymą (flow control).	Taip	Galima remtis nuorodų tikrinimu ir apjungti kelis patikrinimo veiksmus į seką – patikrinti, ar aplikacijos veiksmų seka teisinga.

[34] Pastaba. Su automatinio testavimo įrankio vertinimo ir parinkimo praktika susipažinti galima Dustin E., Rashka J., Paul J. „Automated Software Testing – Introduction, Management, and Performance“. Addison-Wesley Professional, 1999, p. 67. ISBN: 0201432870.

[35] Michael J. B., Bossuyt B. J., Snyder B. B. “Metrics for measuring the effectiveness of software-testing tools", 2002, p. 117-128.

[36] Sampath S., Sprenkle S., Gibson E., Pollock L., Greenwald A. S. "Applying Concept Analysis to User-Session-Based Testing of Web Applications", Software Engineering, IEEE Transactions, vol. 33, no. 10, p. 643-658, Oct. 2007. ISSN: 0098-5589.

Formos klaidos	Klaidos, liečiančios veiksmus su formomis.	Taip	Galima tikrinti formų turinį, formų veikimą.
Atvaizdavimo klaidos	Klaidos, įtakojančios tai, kaip aplikacijos duomenys pavaizduojami naudotojui.	Taip	Elementų atvaizdavimą ir struktūrinius aplikacijos turinio pokyčius galima tikrinti naudojant XPATH.
Nuorodų klaidos	Klaidos, pakeičiančios nuorodų adresus.	Taip	Nuorodų tikrinimas galimas kaip ir kiekvieno elemento.

3. 3. Sukurto įrankio įvertinimas

Lentelė 3. Kiekybinis įrankių vertinimas – užrašius ir atlikus testavimą su automatinio testavimo įrankiu.

		Įrankis											
		Instructable TestMaker	Selenium-WebDriver	Cucumber (su Selenium-WebDriver)	Selenium-IDE	Sahi	Watir	Watij	Watin	P.A.M.I.E.	Celerity	Simple Web Automation Toolkit	Imprimatur
Kiek eilučių skripte	X	70	37	31	54	4	20	32	29	16	18	13	16
Kiek klasių skripte	Y	0	1	0	1	0	1	1	1	0	1	0	0
Kiek skriptų (failų) teste	Z	1	1	2	1	1	1	1	1	1	1	1	1
Kiek papildomų įrankių naudojama	W	1	2	1	1	2	1	1	1	0	0	1	0
Kiek bibliotekų panaudota	Q	2	4	1	3	0	2	2	2	1	1	1	0
Esminės problemos	P	1	1	1	1	1	1	1	1	1	1	1	1

Kadangi testavimo veikimo trukmei turi įtakos naršyklės automatinio naršymo įrankio greitaveika, tai bendras testo vykdymo laikas priklauso daugiausia nuo testavimo įrankio. Todėl testų vykdymo laikas neįtrauktas į vertinimą.

Pastaba. Visi automatinio testavimo įrankiai turi vieną problemą apsunkinančią teisingą testų vykdymą – vykdant automatinį testavimą aplikacijos duomenys nespėja pasikrauti į naršyklę.

Problema yra susijusi su žiniatinklio aplikacijos architektūra. Žiniatinklio aplikacija veikia per tinklą, vykdomas duomenų apsikeitimas tarp serverinės ir klientinės žiniatinklio aplikacijos dalių. Vykdant dinamines užklausas (asinchronines), žiniatinklio aplikacijai dar nepateikus atsakymo testavimo įrankis mėgina atlikti sekantį testavimo žingsnį arba patikrinti rezultatus. Negavus tikrinamo rezultato, automatinio testavimo įrankis užfiksuoja klaidą (angl. „false positive“ tipo klaidą), nors realios klaidos nėra.

Pavyzdys. Turime žiniatinklio aplikaciją, kurioje pateikiama galimybė atlikti paiešką įvedus į tekstinį laukelį paieškos frazę. Jei žiniatinklio aplikaciją testuosime automatiniu būdu, tai turėsime numatyti, kad įvedus paieškos frazę atsakymas iš žiniatinklio aplikacijos vėluos – atsakymą ir rezultatą galėsime patikrinti tik praėjus kuriam laikui, o ne iš karto po tekstinio paieškos laukelio užpildymo paieškos fraze.

Dėl to, kuriant žiniatiniklio aplikacijų testavimo atvejus, reikia numatyti užlėtinimą, po veiksmo vykdyti komandą „wait“ („wait for page to load“, „wait for text“ ir pan.) Automatinio testavimo įrankis Sahi automatiškai vykdo „wait“, kuriant testus kitiems įrankiams Selenium (1,2 versijos), Watir, Watin reikia naudoti „wait“.

3. 4. Įrankis – Instructable TestMaker

Įrankis, naudotas testavimo modelyje automatiniam testų vykdymui – „Instructable TestMaker“ – atlieka testo žingsnius pagal instrukcijas, aprašytas XML faile.

Elementų aplikacijos lange ieško pagal XPATH, patikrinimus (teiginius – angl. assertion, tikrina) atlieka pagal XPATH, tuo paremtas įrankis. Instructable TestMaker yra funkcinį testavimą atliekantis įrankis, kuris remiasi raktažodinėmis instrukcijomis.

Įrankio savybės:

- Įrankis vykdo tris pagrindinius veiksmus:
  1) Rasti elementą – ar elementas egzistuoja duotame lange.
  2) Suaktyvinti elementą – atlieka „click“ ant elemento.
  3) Įvesti tekstą į elementą – jei elementas turi lauką „value“ (t.y. tekstą priimantis elementas – input type=“text“), tai pakeičia jo reikšmę į duotą.
- Įrankis vykdo funkcinio testavimo instrukcijas užrašytas XML formatu. XML‘o, kurį įrankis supranta, struktūrą apibrėžia toks DTD:

```
<?xml version="1.0" ?>
<!DOCTYPE test [
  <!ELEMENT test (instruction)*>
  <!ELEMENT instruction (action,target,parameters?)>
  <!ELEMENT action (#PCDATA)>
  <!ELEMENT target (#PCDATA)>
  <!ELEMENT parameters (#PCDATA)>
]>
```

Action – veiksmas, kurį atlikti. Galimos reikšmės:

- Navigate – atidaryti langą adresu, kuris nurodytas instrukcijos elemente „target“.
- Settext – įvesti tekstą, nurodytą elemente „parameters“ į elementą, nurodytą instrukcijos elemente „target“.
- Click – suaktyvinti žiniatinklio aplikacijos aktyvaus lango elementą, nurodytą parametre „target“.
- Assert – atlikti elemente „target“ nurodytą veiksmą (XPATH) ir patikrinti, ar veiksmo rezultatas yra TRUE.

Target – rodo, kokį XPATH atlikti, tai gali būti bet koks XPATH. Elemento identifikavimui: a) id('gbqfq'); b) /html/body/table/descendant::span[contains(.,'mif.vu.lt') and position()=1]; rezultatui patikrinti: count(/html/body/table//span[contains(.,'tvarkarastis')])=1.

Parameters – saugo reikšmę, kuri panaudojama atliekant veiksmą „action“.

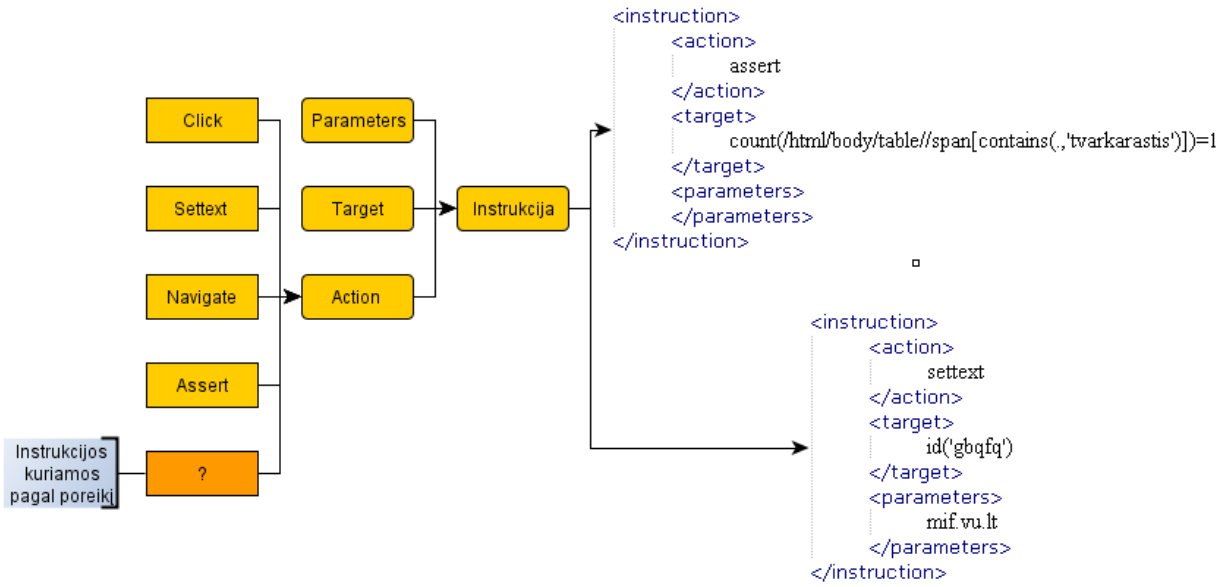
Lentelė 4. Elementų reikšmės.

	Target	Parameters
Action	**Reikšmė privaloma**	
Navigate	Taip	Ne
Settext	Taip	Ne
Click	Taip	Ne
Assert	Taip	Ne

Lentelė 5. Instrukcijos pavyzdys.

Action	Target	Parameters
	Reikšmė	
Navigate		http://www.google.com
Settext	id('gbqfq')	mif.vu.lt
Click	/html/body/table/descendant::span[contains(.,'mif.vu.lt') and position()=1]	
Assert	count(.//*[@id='rso']//a[contains(.,"Matematikos ir Informatikos fakultetas")])=1	

Paveikslas 6. Instrukcijų struktūra.



1) Įrankis sugeneruoja ataskaitą HTML formatu, kurioje parodo pavykusių ir nepavykusių testų pavadinimus kartu su klaidos pranešimu.

Paveikslas 7. Ataskaita.

- PASSED Test name: tests/GoogleSearch1.steps.xml
- Start time: 2012-05-14 20:24:33 +0300
- Time elapsed: 54

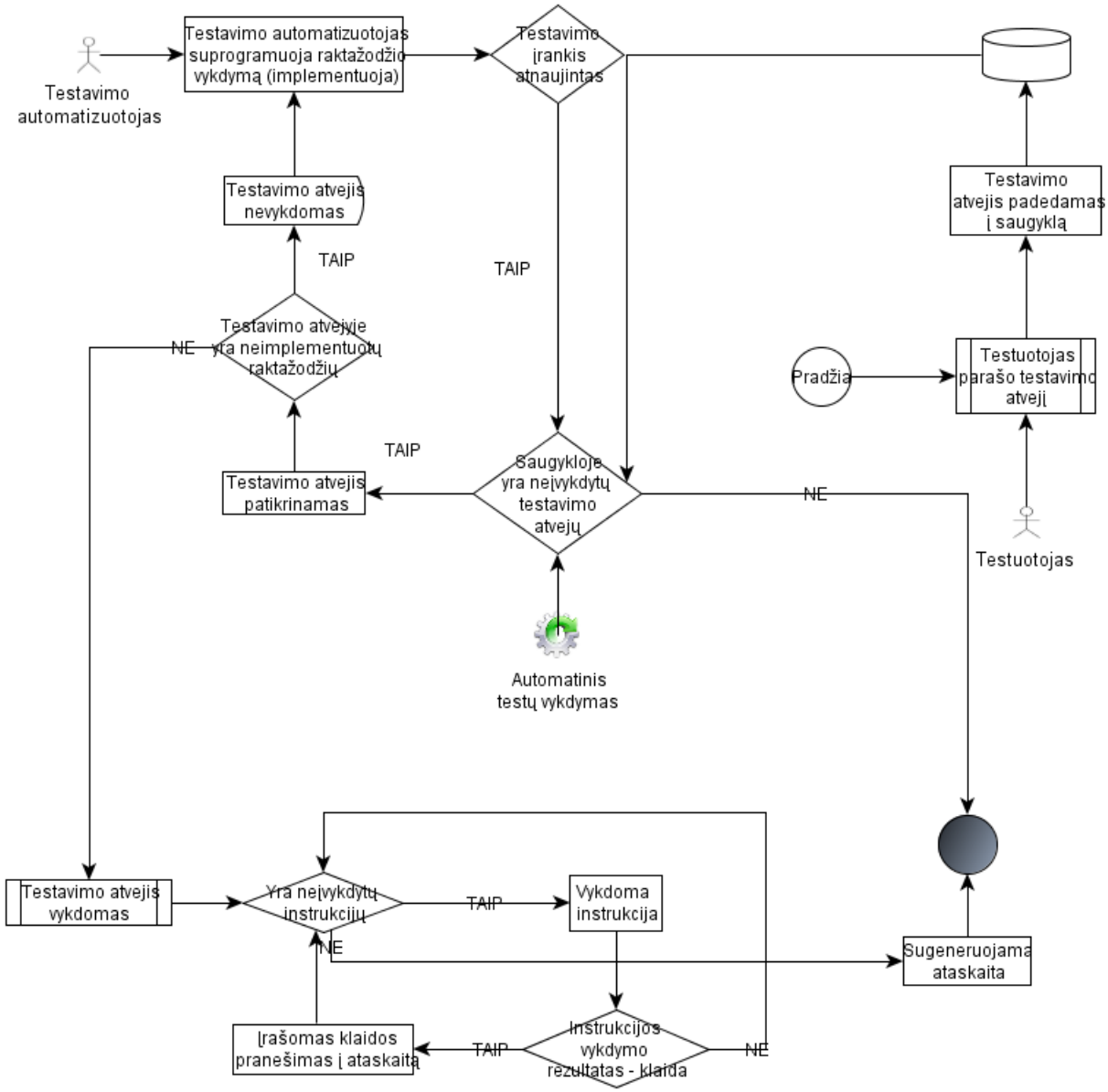
- FAILED Test name: tests/GoogleSearch1_with_failure.steps.xml
- Start time: 2012-05-14 20:25:29 +0300
- Time elapsed: 25
- Error msg: Assertion failed: count(/html/body/table//span[contains(.,'tvarkarasciai')])=1;Assertion failed: count(/html/l

Įrankis remiasi XPATH. Tam, kad galėtume pasiekti elementą lange, turime nurodyti kelią iki jo, adresą dokumente (HTML). Tai atliekame naudodami XPATH. Kadangi XPATH palaiko funkcijas, pavyzdžiui, count(), tai mums leidžia atlikti daugiau patikrinimų, pvz., tikrinti, ar elementas yra lange (atliekame elemento paiešką ir suskaičiuojame, pvz., lange atliekame XPATH‘ą „count(/html/body/table//span[contains(.,'tvarkarastis')])=1); naudojant kitas funkcijas, pvz., text() galime tikrinti elementų turinį, kombinuoti funkcijas, pvz., count(/html/head/title[contains(text(),"Mieli studentai,")])=1.

Naršyklės valdymo įrankis, kurį įrankis naudoja, yra Selenium-WebDriver, Watir. Atsiradus poreikiui, galima pridėti kito naršyklės valdymo įrankio palaikymą, tačiau iš palaikančių XPATH įrankių yra Selenium-WebDriver, Watir, kurie ir panaudoti. Kiti įrankiai, pvz., Sahi nepalaiko XPATH (įgalinus XPATH palaikymą pasinaudojant Javascript plėtiniu Javascript-Xpath, bandymo metu pastebėta, kad įrankis Sahi veikia nenašiai – testo atlikimas sulėtėjo iki keleto minučių, dėl ilgos elementų paieškos, kai Selenium-WebDriver tas pats testas veikė keletą sekundžių).

3. 5. Automatinio testavimo modelis

Paveikslas 8. Automatinio testavimo proceso modelis naudojant raktažodinį testavimą realizuojantį testų vykdytoją.



Automatinio žiniatinklio aplikacijų testavimo modelio privalumai:

1) vienodos instrukcijos nepriklausomai nuo testavimui naudojamų įrankių;
2) testui užrašyti reikalingas – tik XPATH ir aplikacijos elementų (HTML) sandaros žinios;
3) numatytas naujų instrukcijų kūrimas;
4) testų vykdymas yra atskirtas nuo testų kūrimo.

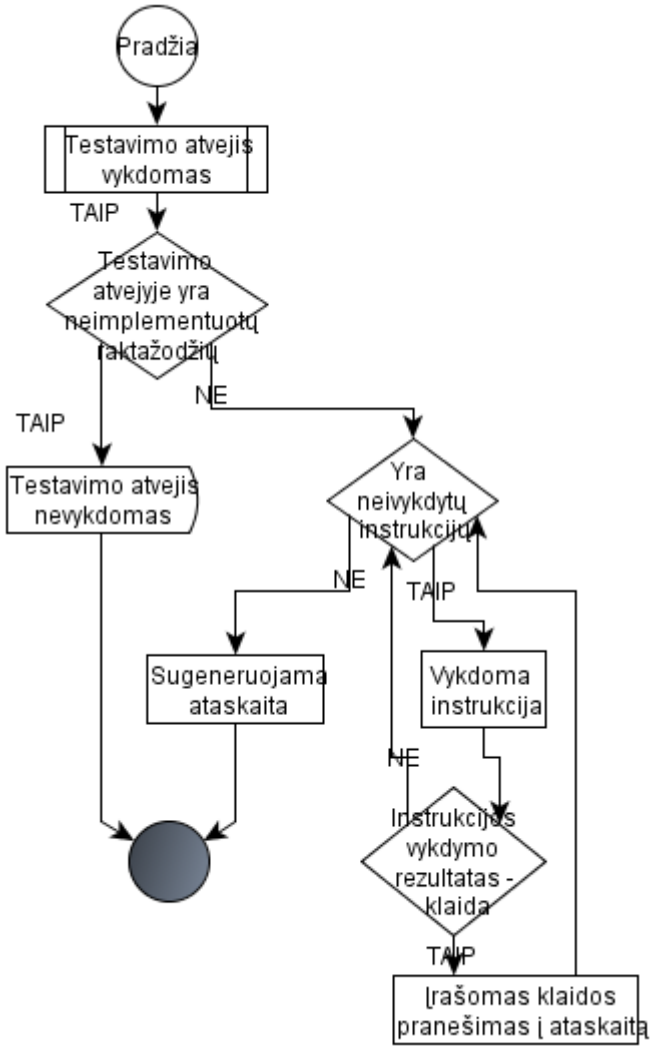
Modelio trūkumai: naudojamos įkoduotosios reikšmės testuose. Tokį trūkumą galima spręsti modelį praplečiant Data-driven technika.

Modelyje atskleidžiamos tokios savybės:

1. elementų identifikavimas pagal kelis atributus;
2. nepriklausomas nuo driver‘io modelis (galima naudoti Selenium-WebDriver, Watir, kaip pavyzdyje, arba pridėti naują);
3. sugeneruojama ataskaita;
4. automatiniams testams užrašyti nereikia naudoti programavimo,

Sudėtinė automatinio testavimo modelio dalis yra automatinių testų vykdymo įrankis.

Paveikslas 9. Testų vykdymo įrankio veikimo schema



Išvados

Testavimas – svarbi ir imli darbui aplikacijos kūrimo ir vystymo proceso dalis, kuri yra įtraukta į visas programų kūrimo metodikas. Kaip ir aplikacijos kūrimas, testavimas reikalauja nemažai sąnaudų. Todėl norint suvaldyti augančias testavimo sąnaudas, galima dalį aplikacijos testavimo automatizuoti.

Rengiant magistro baigiamąjį darbą išryškėjo tai, kad automatinis testavimas yra vertinamas nevienareikšmiškai: automatinis testavimas taip pat yra imlus darbui uždavinys – būtina sukurti testų vykdymo modelį, prieš įgyvendinant automatinį testavimą būtina apibrėžti testavimo procesą. Visgi, parengus automatinio testavimo įrankius, testavimo procesas palengvėja, nes testų nebereikia vykdyti rankiniu būdu.

Tiriamojo darbo metu susipažinus su žiniatinklio automatinio testavimo įrankiais: įrankių savybėmis, galimybėmis, trūkumais, ir juos įvertinus, parinktas geriausias automatinio testavimo įrankis. Sukurtas žiniatinklio aplikacijos raktažodinio testavimo įrankis, kurio pagalba vykdomi automatiniai žiniatinklio aplikacijos testai, ir testavimo modelis. Sukurtas testavimo modelis gali būti taikomas žiniatinklio aplikacijos automatinių testų kūrimui ir vykdymui.

Magistro baigiamajame darbe išanalizavus ir palyginus automatinio testavimo įrakius, nustatytas geriausiai žiniatinklio aplikacijos automatiniam testavimui tinkantis įrankis – Selenium-WebDriver, kurio pagrindu sukurtas įrankis, žiniatinklio aplikacijos elementų identifikavimui naudojantis XPATH, o naršyklės valdymui – nepriklausomus nuo testavimo įrankio driver‘ius.

Tolimesniam testavimo modelio vystymui reikėtų kurti naujas instrukcijas, pridėti automatinį testavimo rezultatų įvertinimą (ištirti galimybę panaudoti testavimo orakulus). Svarbiausias patobulinimas modeliui būtų – apjungimas su Data-driven testavimu, kuris įgalintų platesnį testų panaudojamumą.

Automatinis testavimas nėra panacėja, turi apribojimus ir, kaip matome, nepakeičia rankinio testavimo. Nepaisant to, teisingai įgyvendintas automatinio testavimo modelis palengvina testavimą ir užtikrina savalaikį žiniatinklio aplikacijos klaidų radimą.

Literatūros sąrašas

1) Araújo B. C., Rocha A. C., Xavier A., Muniz A. I., Garcia F. P. Web-based tool for automatic acceptance test execution and scripting for programmers and customers. ACM, New York, NY, USA, 2007. Article 56. ISBN: 978-1-59593-598-4.
2) Bach J. „Test automation snake oil“, 14th International conference on Testing Computer Software, 1997. [žiūrėta: 2012-04-20]. Prieiga per internetą: < http://www.satisfice.com/articles/test_automation_snake_oil.pdf >
3) Beizer B. Software Testing Techniques. Boston, International Thompson Computer Press, 1990.
4) Kam B., Dean T. R. “Lessons Learned from a Survey of Web Applications Testing”. IEEE Computer Society, Washington, DC, USA, 125-130. ISBN: 978-0-7695-3596-8.
5) Berner S., Weber R., Keller R. K. Observations and lessons learned from automated testing. ACM, New York, NY, USA, 2005. ISBN: 1-58113-963-2.
6) Bourque P., Dupuis R. "Guide to the Software Engineering Body of Knowledge 2004 Version", Guide to the Software Engineering Body of Knowledge, 2004. SWEBOK, 2004. ISBN 0-7695-2330-7.
7) Choudhary S. R., Versee H., Orso A. “A cross-browser web application testing tool”. 2010. ISBN: 978-1-4244-8630-4.
8) Dustin E. „Effective Software Testing: 50 Specific Ways to Improve Your Testing“. Addison-Wesley Professional, 2002. ISBN: 0201794292.
9) Dustin E., Rashka J., Paul J. „Automated Software Testing – Introduction, Management, and Performance“. Addison-Wesley Professional, 1999. ISBN 0201432870.
10) Fewster M. „Software test automation“. Addison-Wesley Professional, 1999. ISBN: 0201331403.
11) Harrold M. J. "Reduce, reuse, recycle, recover: Techniques for improved regression testing", Software Maintenance, 2009. ICSM 2009. p. 5, 20-26 Sept. 2009. ISSN: 1063-6773.
12) Hayes L. G. „Automated Testing Handbook“. Software Testing Institute, 2004. ISBN: 0970746504.
13) IEEE. IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, 1990.
14) Kaner C. „Improving the Maintainability of Automated Test Suites“. Software QA, 4(4), 1997.

15) Martin D., Rooksby J., Rouncefield M., Sommerville I. 'Good' Organisational Reasons for 'Bad' Software Testing. IEEE Computer Society, Washington, DC, USA, 2007. ISBN: 0-7695-2828-7.
16) Memon A. M. 2008. “Automatically repairing event sequence-based GUI test suites for regression testing.” ACM Trans. Softw. Eng. Methodol. 18, 2, Article 4 (November 2008).
17) Michael J. B., Bossuyt B. J., Snyder B. B. “Metrics for measuring the effectiveness of software-testing tools", 2002.
18) Myers G. J. „The Art of Software Testing“. John Wiley & Sons, 2004. ISBN: 0471469122.
19) Onoma A. K., Tsai W. T., Poonawala M., Suganoma H. 1998. Regression testing in an industrial environment. Commun. ACM 41, 5 (May 1998).
20) Ricca F., Tonella P. “Analysis and testing of Web applications.” 2001. ISBN: 0-7695-1050-7.
21) Ruparelia N. B. „Software development lifecycle models“. 2010.
22) Sampath S., Sprenkle S., Gibson E., Pollock L., Greenwald A. S. "Applying Concept Analysis to User-Session-Based Testing of Web Applications", Software Engineering, IEEE Transactions, vol. 33, no. 10, 2007. ISSN: 0098-5589.
23) Wandan Z., Ningkang J., Xubo Z. “Design and Implementation of a Web Application Automation Testing Framework.” IEEE Computer Society, Washington. 2009.

Priedai

Priedas Nr. 1 Automatinio testavimo įrankis „Instructable TestMaker“ (žiūr. CD laikmeną).