VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Automatinis testavimas programų kūrimo procese
Automation testing in software development

Bakalauro darbas

Atliko: Rugilė Petrukauskaitė

Darbo vadovas: Lekt. dr. Vytautas Valaitis

Darbo recenzentas: Julija Vysockytė-Bilevičienė

Vilnius – 2017

TURINYS

SANTRAUKA LIETUVIŲ KALBA

Šiandieniniame pasaulyje programinė įranga užima vis didesnę reikšmę. Kiekvienas sudėtingesnis produktas kontroliuojamas programinės įrangos, o dauguma komercinių paslaugų remiasi programinėmis sistemomis – internetiniais puslapiais, mobiliomis aplikacijomis ir kita. Pagrindiniai reikalavimai produktams – greitai kuriami, tačiau tuo pačiu metu gebantys prisitaikyti prie kintančių reikalavimų, ir su atitinkama kokybe. Tai galima įgyvendinti gerinant testavimo veiklą. Dėl šios priežasties darbe išsikeltas tikslas – testavimo praktikų taikymo analizė programų sistemų kūrimo gyvavimo cikle.

Darbe, remiantis Krioklio, Agile bei DevOps metodikomis, išanalizuotas rankinis bei automatinis testavimai, atliktas internetinės aplikacijos vartotojo sąsajos automatinis testavimas su skirtingais įrankiais – Selenium, CODED UI, TestComplete. Pagal gautus rezultatus suformuluotos rekomendacijos, padėsiančios išsirinkti automatizavimo įrankį.

Pasinaudojus darbe pateikia testavimo fazės analize galima pagal savo poreikius išsirinkti tinkamiausią darbo metodiką ir išgauti didžiausią naudą testuojant produktą. Atsižvelgiant į rankinio bei automatinio testavimo palyginimą bei praktinę darbo dalį, pagal pateiktas autoriaus rekomendacijas galima išsirinkti testų automatizavimo įrankį.

Raktiniai žodžiai: automatinis testavimas, rankinis testavimas, Krioklio metodika, Agile metodika, DevOps metodika, Selenium automatizavimo įrankis, CODED UI automatizavimo įrankis, TestComplete automatizavimo įrankis.

SANTRAUKA ANGLŲ KALBA (angl. SUMMARY)

Nowadays software has a big impact on people life. Every complex product has it‘s own software, business cannot image their work without web pages, mobile applications and ect. Main requirements for products are fast development, adjustment to changing requirements and good quality. Because of those requirements, main work goal is to analyze testing practice in software development life cycle.

In work, author relied on Waterfall, Agile and DevOps methodologies, analyzed manual and automated testing, wrote web application user interface automated tests with different tools – Selenium, CODED UI and TestComplete. According to result, recommendations for choosing the right automation testing tool are written.

Relying on in the work did analysis of testing phase, reader can chose appropriate methodology and gain the biggest value in testing products. According to manual and automated testing comparison and practical part, also keeping in mind author given recommendations, reader can chose the best automation tool for his testing.

Keywords: automated testing, manual testing, Waterfall methodology, Agile methodology, DevOps methodology, Selenium automation tool. CODED UI automation tool, TestComplete automation tool.

ĮVADAS

Šiais laikais programinė įranga yra visur. Kiekvienas sudėtingesnis produktas kontroliuojamas programinės įrangos, o dauguma komercinių paslaugų remiasi programinėmis sistemomis – internetiniais puslapiais, mobiliomis aplikacijomis ir kita [Lin14]. Tačiau įmonės, kuriančios šiuos produktus, ne visuomet sugeba pilnai įgyvendinti užsakovų reikalavimus. Remiantis 2014m. atliktu tyrimu, 37% iš apklaustų organizacijų įvardino nemokėjimą suvaldyti reikalavimų, kaip pagrindinę priežastį, dėl kurios buvo žlugę jų projektai [Msd15]. Esminiai faktoriai, kurių tikisi kiekvienas programinės įrangos užsakovas yra šie [Lin14]:

1. Reikalavimų atitikimas.
2. Minimalus defektų kiekis.
3. Greitas produkto sukūrimas.
4. Kuo mažesnės išlaidos.

Suprantama, jog tik kokybiškas produktas gali padėti rinkoje išlikti konkurencingiems, tačiau žodis kokybė yra abstrakti sąvoka. Ją sunku apibrėžti, bet nekokybiškas produktas greitai pastebimas. Jau 20a. pradžioje pirmasis kokybės apibrėžimas pagal Shewhart teigė, jog tai yra subjektyvus dalykas [She31]. ISO/IEC 9001 standartas pateikė šiek tiek konkretesnį apibrėžimą – tai programinės įrangos atitikimas numatytiems reikalavimams [Iso99]. Deming nuomonė panaši, tačiau jis dar pridėjo, jog kokybę apibrėžti sunku dėl vienos priežasties – kai atrodo, kad produktas tenkina visus užsakovo poreikius, šie pasikeičia, konkurentai pasistūmėja toliau, atsiranda nauja technologija ar produktas [Dem88]. Dėl to pirmasis punktas „Reikalavimų atitikimas” sukelia nemažai problemų bei apsunkina antrąjį punktą – atsiradus pakeitimams ar naujam funkcionalumui, atsiranda ir naujų defektų. Naujo funkcionalumo įgyvendinimas bei atsiradusių defektų taisymas atitolina produkto pridavimą užsakovams - trečiasis punktas. Atsiradus šioms problemoms, stipriai išauga produkto kaina, kas trukdo įgyvendinti ketvirtąjį punktą. Vėlyvas programinės įrangos integravimas į savo produktą ar tiesiog jos paleidimas į prekybą, mažina šansus išlikti rinkoje konkurencingiems. Šias ir kitas problemas bando išspręsti programų sistemų kūrimo metodikos, taikydamos įvairius metodus programų sistemų kūrimo gyvavimo ciklui.

Viena iš seniausių tradicinių metodikų yra Krioklys (angl. waterfall). Ji skaido projektą į atskiras fazes, kurių atlikimas pažymėtas tam tikra gaire (angl. milestone). Tai linijinis metodas, kiekviena fazė gali būti pradėta tik pabaigus prieš tai esančią. Nemažai tyrimų buvo atlikta bandant išsiaiškinti šios metodikos efektyvumą. 2013 metais atliktame tyrime buvo lyginama keletas metodikų [MS13]. Krioklio metodika buvo giriama už paprastą jos įgyvendinimą, tačiau gavo

nemažai kritikos dėl savo nelankstumo – pasikeitus kliento reikalavimams, jie nebus įgyventi, projekto apimtis negali padidėti ar sumažėti, nes grįžti į jau pabaigtą fazę negalima. 2015 metais atliktame tyrime, ši metodika buvo atmesta kaip netinkama dėl trūkstamo grįžtamojo ryšio tarp kūrimo fazių, vėlyvo testavimo bei trūkstamų tarpinių rezultatų [MKM15]. Dėl šių bei kitų priežasčių tradicinė programinės įrangos kūrimo metodika, paremta ilgu planavimu bei dokumentacija, nesugebėjo pilnai patenkinti šiuolaikinės visuomenės poreikių ir buvo pradėta ieškoti lankstesnių variantų.

Prieš gerą dešimtmetį vis dažniau pradėta taikyti Agile metodiką, kurios pagrindinis bruožas – kūrimas iteracijomis. Šioje metodikoje renkamos daugiafunkcinės komandos, kurios greitai ir lanksčiai prisitaiko prie kintančių reikalavimų. Tikslesnį apibrėžimą galime rasti „Agile Manifesto" dokumente, kurį 2001m. parašė grupė programų sistemų kūrėjų, tikėjusių, kad šis metodas gali ženkliai pagerinti programų sistemų kūrimo procesą. Anot jų, tai iteratyvus (apimantis keletą ciklų) bei laipsniškas (produktą pristato dalimis, ne iš karto visą) metodas kurti programinę įrangą, kurio esmė - artimas bendradarbiavimas tarp daugiafunkcinės komandos narių, kurie turi nemažai laisvės patys rasti geriausią būdą atlikti darbą (neprivalo remtis dokumentacija), ko pasekoje efektyviai kuriami didesnės kokybės produktai, nereikalaujantys daug laiko ir gebantys prisitaikyti prie kintančių užsakovo poreikių. „Agile Manifesto" dokumentas labai greitai tapo visuotinai pripažintas, šis metodas buvo laikomas kaip efektyvus pasirinkimas siekiant išgauti geresnę kokybę per trumpesnį laiką [Agi01]. Agile buvo grindžiama supratimu, jog kuriamų sistemų reikalavimai yra dinamiški ir juos valdo paklausa [Boe02]. Tai buvo lyg atsakas į tradicinės metodikos nesugebėjimą suvaldyti nuolat besikeičiančios aplinkos [Hig12].

Viena populiariausių metodikų išlieka Agile, tačiau vis dažniau pradedamas minėti „DevOps" terminas. Straipsniuose tai apibūdina kaip tam tikras procesas, technologija, mąstymo būdas ar tiesiog nauja metodika. Pirmą kartą šį terminą paminėjo Patricko Debois 2009 metais. Tai dviejų angliškų žodžių – plėtojimas (angl. development) ir veikimas (angl. operations) darinys [HM12]. Jis sujungia šias dvi disciplinas, kad pabrėžtų bendradarbiavimą, kolaboravimą ir ryšį tarp tradiciškai atskirtų programuotojų ir IT procesų valdymo komandų. Remiantis 2016m. švedijos technologijų instituto išleistu straipsniu [Ama16], Agile metodikoje pagrindinis dėmesys buvo skiriamas plėtojimo pusei, tuo pat metu ignoruojant operacijų dalį. Operacijų komanda diegia, tvarko bei palaiko sistemos vykdymą iš kliento pusės. Plėtojimo komanda pristato naujas versijas daug greičiau, o operacijų komanda nespėja, lieka už nugaros. Dėl šios priežasties visas produkto pristatymas gali vėluoti. Plėtojimo ir operacijų dalių atskyrimas, didina galimybę atsirasti

nesusipratimams ir konfliktams. DevOps šį plyšį sugeba užpildyti taikant įvairias praktikas, padedančias programuotojams ir operacijų personalui dirbti kartu. Daugelis žinomų kompanijų – Facebook, Yahoo, Netflix, Flicker and Fotopedia įgyvendina DevOps metodiką [Ama16].

Programų sistemų kūrimo metodikos aprašo aplinką, kuri naudojama organizuoti bei planuoti kūrimo procesą. Nors minėtos metodikos yra ganėtinai skirtingos, tačiau jos visos remiasi į pagrindinius programų sistemų kūrimo etapus – programavimą bei testavimą. Daugumoje literatūros, kurioje rašoma apie šias metodikas, autorius žiūri iš programavimo perspektyvos, pabrėždamas programavimo technikas ar principus [VBP14]. Jei kalbama apie testavimą, dažniausiai minima tik modulinis (angl. unit) testavimas ir atitinkami tam įrankiai. Trumpai tariant, koncentruojamasi į testavimą iš programavimo perspektyvos. Deja, modulinis testavimas nėra pakankamas ir esminis norint pasiekti aukštą kokybę. Tai viena iš priežasčių, kodėl svarbu pažvelgti į testavimą iš testuotojų pusės per įvairių metodikų prizmę [Lin14].

Darbo tikslas - testavimo praktikų bei įrankių taikymo analizė programų sistemų kūrimo gyvavimo cikle.

Darbe analizuojama, kaip projektų kūrimo procese, remiantis minėtomis metodikomis – Krioklys, Agile ir DevOps, įsipina testavimo fazė. Siekiant šio tikslo, darbe išsikelti uždaviniai:

1. Atlikti testavimo fazės analizę skirtingose metodikose.
2. Atlikti automatinio testavimo svarbos analizę.
3. Atlikti automatizuotą internetinės aplikacijos https://connect.bentley.com dalies vartotojo sąsajos testavimą su skirtingais automatizavimo įrankiais.
4. Palyginti naudotus automatizavimo įrankius pagal techninius bei praktinius aspektus.
5. Pateikti rekomendacijas apie įrankio, skirto automatizuoti testavimą, pasirinkimą.

1. TESTAVIMO FAZĖS VIETA SKIRTINGOSE METODIKOSE

Kiekviena metodika turi savą programų sistemų kūrimo gyvavimo ciklą. Tai būdas, pagal kurį sistema turi būti kuriama, padidinantis tikimybę atlikti viską laiku ir su tinkama kokybe. Cikle apibūdinamos fazės, kurias būtina atlikti. Kiekviena fazė kažką perduoda sekančiai. Iš reikalavimų sudaromas projektavimas, pagal tai rašomas kodas, o galiausiai viskas, kas sukurta, yra ištestuojama ir palaikoma. Remiantis 2015m. IJETST[1] atlikta skirtingų metodikų analize, buvo išskirtos šios programų kūrimo gyvavimo ciklo fazės [STC15]:

1. **Reikalavimų surinkimas ir analizė.** Šioje fazėje surenkami užsakovo reikalavimai. Vyksta susitikimai, kurių metų iškeliami klausimai: kas naudosis šia sistema? Kaip ja bus naudojamasi? Kokia bus įvestis? Kokia turėtų būti išvestis? Atsakius į šiuos ir panašius klausimus vyksta reikalavimų analizė. Galiausiai reikalavimai dokumentuojami ir keliauja į kitą etapą.
2. **Projektavimas.** Šioje fazėje pagal surinktus reikalavimus projektuojamas būsimo produkto dizainas ir architektūra.
3. **Įgyvendinimas/Programavimas.** Pagal prieš tai įgyvendintas fazes darbas skaidomas į mažesnius modulius ir pradedamas programavimas. Tai ilgiausia fazė programų sistemų gyvavimo cikle.
4. **Testavimas:** Kai suprogramuota dalis, ar visa sistema, testuojama pagal pradinėje fazėje surinktus reikalavimus.
5. **Įdiegimas:** Po sėkmingo testavimo produktas yra diegiamas į gamybinę aplinką, skirtą naudoti užsakovams.
6. **Palaikymas:** Kai vartotojai pradeda naudotis sistema, dažniausiai atsiranda pagrindinės problemos, kurias kartas nuo karto reikia išspręsti. Būtent tai ir daroma palaikymo fazėje.

Toliau darbe bus analizuojama ketvirtoji fazė – testavimas.

Neįmanoma sukurti kokybiškos programinės įrangos be testavimo. Programinės įrangos testavimas dažniausiai atsieja 30-40% pastangų ir netgi daugiau, jei produktas reikalauja didesnio patikimumo nei įprastai. Tai itin svarbi fazė programų sistemų kūrimo procese. Jei defektai randami vėlesnėse kūrimo stadijose, tai kainuoja žymiai daugiau pinigų, negu radus juos pradžioje. Dėl to, kuo anksčiau pradedama testuoti, tuo didesnė tikimybė, kad projektas bus pabaigtas greičiau ir mažesniais kaštais. Anot 2014m. atlikto tyrimo, sutaisyti problemą, rastą reikalavimuose po

[1] IJETST - International Journal of Emerging Trends in Science and Technology

programos paleidimo fazės, kainuoja apie 10-100 kartų brangiau, negu radus ją reikalavimų peržiūrėjimo metu. 1 lentelėje parodoma defektų taisymo kaina priklausomai nuo to, kurioje fazėje jie buvo rasti [Bas14]:

1 lentelė. Vidutinė defektų taisymo kaina. **R** - Reikalavimai, **A** – Architektūra, **K** – Kūrimas *2014, Sukanta Basak, „Software Testing Process Model from Requirement Analysis to Maintenance*

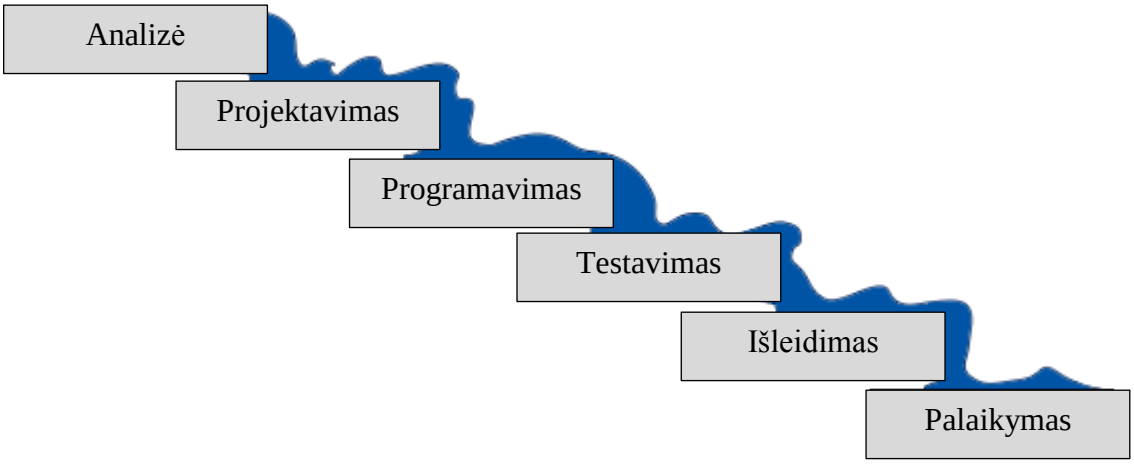
Kaina sutaisyti defektą		**Kuriuo metu rastas defektas**				
		R	**A**	**K**	**Sistemos testavimas**	**Po išleidimo**
Kuriuo metu atsirado	**R**	1x	3x	5-10x	10x	10-100x
	A	-	1x	10x	15x	25-100x
	K	-	-	1x	10x	10-25x

Remiantis šiais duomenimis, rekomenduojama pradėti ieškoti problemų kuo ankstesnėje programų sistemų kūrimo fazėje.

Įvairiose programų sistemų metodikose testavimas įvedamas skirtingu laiku. Pavyzdžiui, iteratyviuose modeliuose testavimas pradedamas daug anksčiau negu Krioklio metodikoje, kurioje pradedama testuoti po viso produkto sukūrimo. Toliau skaitytojas bus supažindintas kaip konkrečiai bei kuriuo metu yra vykdomas testavimas skirtingose metodikose.

1.1. Testavimas Krioklio metodikoje

Kaip jau minėta, tradicinis „Krioklio“ modelis suskaido projektą į atskiras fazes, kuriose turi pasiekti tam tikras žymes (angl. milestones). Jame progresas matomas kaip stabiliai tekantis žemyn (lyg krioklys, 1 pav.) per tradicines fazes.



1 pav. Krioklio metodikos programų kūrimo procesas

Anot šios metodikos, sistemos testavimo veikla turi būti atlikta per testavimo fazę, kai visa sistema sukurta, tačiau defektai gali atsirasi prieš šią fazę ir kuo vėlesnis jų taisymas, tuo didesnė projekto kaina. Pats testavimas pilnai remiasi pirminėse fazėse parašytais reikalavimais. 2014m. išleistoje knygoje „Testig in Scrum“ pabrėžtos šios metodikos silpnybės [Lin14]:

1. Pradinėse fazėse viskas suplanuota gana aiškiai, bet dažniausiai per daug detaliai. Dėl to pirmosios fazės paprastai gauna daugiau išteklių negu joms reikia.
2. Dažniausiai laiką paskirsto vadovai, prieš tai nepasitarę su komanda. Dėl to suplanuoti laiko terminai nėra tikslūs.
3. Prielaida, kad testuotojų komanda ras defektus yra uždengta laiko buferiu skirtu perdarymui. Šitoje vietoje neaišku, ar dvi savaitės pertvarkymui bus per daug ar per mažai. Bet kokiu atveju, perdarymas susijęs su specifiniais tikslinimais, kurie kažkiek paveikia fazės gairę. Tačiau nėra jokio laiko biudžeto skirto perdaryti pradines fazes, kurios taipogi gali būti paveiktos.
4. Daug laiko praeina, kol produktas yra visapusiškai ištestuotas ir pateiktas. Knygoje pateiktame pavyzdyje 20-38 savaitės praeis tarp projektavimo fazės pabaigos ir pirmų integracinių testų, kas reiškia, kad pristatymas bus po 5-8 mėnesių anksčiausiai.

Nors ir turi nemažai minusų, tačiau ši metodika nėra visiškai atmetama. Nemažai straipsnių pataria, kokiais atvejais būtų naudinga naudoti šią metodiką ir būtų galima įvykdyti projektus su atitinkama kokybe [Ist15]:

1. Kai yra aiškūs bei konkretūs reikalavimai, nėra jokių abstraktumų.
2. Kai produktas aiškiai apibrėžtas bei stabilus, nenumatomi jokie pakeitimai.
3. Technologija aiškiai suprasta.

4. Tiksliai apskaičiuotos išlaidos ir resursų yra pakankamai.
5. Projektas yra trumpas.

1.2. Testavimas Agile metodikoje

Anot Agile metodikos, testavimas nėra vien tik atskira fazė. Ji turi būti integruota į visą programų sistemų kūrimo procesą kartu su programavimu. Testavimas ir kodo rašymas yra atliekami palaipsniui, kuriant kiekvieną papildomą funkcionalumą. Šios dvi fazės persipina tol, kol produktą galima talpinti į gamybinę aplinką. Testavimas turi būti nuolatinis, nes, anot šios metodikos, tik taip galima užtikrinti nuolatinį progresą. Taipogi turi būti nuolat teikiamas grįžtamasis ryšys, kuris padeda pasitikrinti, ar nėra nuklydimų nuo užsakovo reikalavimų [Wri14].

Vietoje ilgų dokumentacijų, Agile testuotojai rašo pernaudojamus kontrolinius sąrašus (angl. checklists), fokusuojasi ties esminiu testavimu, vietoj to, kad eikvotų laiką smulkmenoms, kurios atneš mažiau naudos.

Agile testuotojai nėra tik testuotojai ir defektų rašytojai. Jie dirba kaip svarbi visos komandos dalis ir nemažai bendrauja su produkto savininku (angl. product owner) bei kitais komandos nariais tam, kad išgautų kuo daugiau detalių apie produktą. Tik puikiai išmanantis produktą žmogus jį gali ištestuoti pilnai [Olu14].

Nors ši metodika turi daug pliusų, lyginant su Kriokliu, tačiau literatūroje galima rasti ne tik pagyrų. Minima ir keletas iššūkių, su kuriais susiduria testuotojai [Gur15]:

1. Kadangi dokumentacijai teikiamas žemesnis prioritetas, vis didesnė atsakomybė ir spaudimas tenka testuotojų komandai.
2. Naujas funkcionalumas įvedamas greitai, o tai sumažina laiką, kuris skirtas išanalizuoti, ar šis funkcionalumas atitinka užsakovo bei verslo reikalavimus.
3. Dažnai testuotojams tenka pusiau programuotojų rolė.
4. Testų atlikimo laikas yra stipriai suspaustas.
5. Mažai laiko pasiruošti geriems testavimo planams.

1.3. Testavimas DevOps metodikoje

DevOps perėmęs pagrindines Agile projekto gyvavimo ciklo dalis, tarp jų ir testavimą. Jų pagrindiniai tikslai yra padidinti ROI[2], pakelti klientų pasitenkinimą nuolatos teikiant naujas

[2] ROI (ang. return of investment) – investuoto kapitalo grąža.

galimybes bei kelti paslaugų kokybę. Anot DevOps atstovo Peuraneimi, tai „agile on steroids“[3] [Peu14]. Tikslų siekimas remiasi keturiais pagrindiniais aspektais [Ama16]:

1. **Kultūra** – tai reiškia, kad plėtojimo ir operacijų skyrius privalo turėti nuolatinius susitikimus.
2. **Automatizavimas** – automatizuoti versijas, plėtojimąsi bei testavimą yra būtina, norint gauti greitą grįžtamąjį ryšį (angl. feedback) ir greičiau vykdyti visus procesus.
3. **Matavimai** – svarbu sekti įvairias metrikas – biudžeto pajamas, transakcijas, darbuotojų atliekamą darbą. Matavimai turėtų būti matomi visiems, kurie yra prisidėję prie sistemos kūrimo, kad galėtų matyti, kaip progresuoja.
4. **Dalinimasis** – dalinimasis žiniomis, automatiniais įrankiais bei technikomis tarp abiejų komandų – plėtojimo ir operacijų.

Antrasis aspektas stipriausiai paveikė Agile testavimo fazę. Anot DevOps, testavimas turi būti nuolatinis ir automatizuotas – visi testavimo atvejai turi būti automatizuoti tokiu būdu, kad nebūtų jokio vartotojo įsikišimo ir naujos versijos klaidos būtų aptiktos greičiau [Ama16]. DevOps sekėjas Smeds pabrėžia, jog pritaikyti DevOps metodiką nėra lengva ir kiekviena organizacija turi rasti savo unikalų būdą pritaikyti tai sėkmingai [SNP15]. Virmani teigia, jog nėra praktiška iškart pereiti nuo vienos metodikos prie DevOps – tam reikia laiko [SNP15]. Anot jo, organizacija pirma turi išsianalizuoti procesus (angl. workflows) ir tikslines vietas, kurias reikia optimizuoti.

Kaip ir Krioklys bei Agile metodikos, taip ir DevOps turi minusų. Kadangi DevOps tikslas viską automatizuoti, reikia turėti techninių žinių turinčius testuotojus. Galima apsimokyti esančius darbuotojus arba samdyti naujus techninius žmones, tačiau tai yra papildomos išlaidos. Kitas aspektas, reikia išsirinkti gerą automatizavimo įrankį, bet nemokami/pigūs įrankiai pilni defektų, o geresnės kokybės - brangūs [Ama16].

1.4. Apibendrinimas

Visos aptartos metodikos turi savų pliusų bei minusų. Negalima teigti, jog kažkuri iš jų yra visiškai netinkama. Krioklio metodika tinka mažiems, aiškius bei nekintančius reikalavimus turintiems projektams. Agile metodika tinka dideliems, su dažnai kintančiais reikalavimai, projektams. Jei įmonė turi pakankamai techninių žinių turinčių testuotojų ar resursų, leidžiančių darbuotojus apsimokyti arba pasamdyti naujų, DevOps metodika stipriai pagreitintų testavimo procesą.

[3] „agile on steroid“ – Agile su steroidais.

Anot šio darbo autoriaus, įgyvendinant tam tikrą metodiką, nereikia visų metodų taikyti aklai – kiekvienas projektas yra specifinis, turintis skirtingus reikalavimus, žmogiškuosius resursus. Pagal projekto aplinkybes reikia parinkti labiausiai atsiperkančius metodus ir juos taikyti – esant reikalui, galima remtis viena metodika, tačiau naudoti ir kitos metodikos metodų.

2. AUTOMATINIO TESTAVIMO NAUDA

Automatinis testavimas – tai įvairių įrankių naudojimas, kuris padeda greičiau ištestuoti aplikaciją. Regresinio testavimo metu testuotojas tikrina, ar nesugriuvo senas funkcionalumas - vykdo testavimo plane surašytus žingsnius rankomis, kas atima daug laiko. Dėl to kompanijos vis dažniau egzistuojančius testavimo atvejus verčia automatiniais. Pagal 2016m. Statistiką [Est16], 9% atsakiusiųjų teigė, jog naudoja tik rankinį testavimą, 76% automatizavo 5-50% testų, 10% atsakiusiųjų padengė 50-95% testų ir likusieji 5% teigė, jog automatizavo visus testus (tai pilnai atitinka DevOps keliamą tikslą testavimui). Nors dabar dauguma apklaustųjų pirmenybę teikia rankiniam testavimui, tačiau paklausus apie jų planus ateinantiems penkeriems metams, rezultatai keičiasi: 73% apklaustųjų teigė, kad norėtų automatizuoti 50-75% testų, o 14% teigė, jog apskritai norėtų atsisakyti rankinio testavimo. Prieš gilinantis į automatinį testavimą, iš pradžių bus aptarta rankinio testavimo privalumai bei trūkumai.

2.1.Rankinio testavimo privalumai bei trūkumai

Testavimo būdai gali būti skaidomi į keletą dalių – rankinį bei automatinį. Rankinis testavimas – tai testavimo technika, kai žmogus testavimo atvejus pasiruošia bei vykdo rankomis. Tai kruopšti veikla, reikalaujanti iš testuotojo keleto savybių – kantrybės, pastabumo, kūrybiškumo bei kai kurių techninių įgūdžių. 2016m. parašytame straipsnyje, kuriame ieškoma rankinio testavimo privalumų bei trūkumų, išvardinti šie pliusai [Spa16]:

1. **Išmokstamumas** – testuojant rankiniu būdų reikia mažiau techninių žinių, nei rašant automatizuotus testus, todėl mokymosi laikotarpis trumpesnis.
2. **Aplikacijos padengimas testais** – daugelyje atvejų automatizuoti testai nepadengia viso funkcionalumo, kurį gali padengti rankinis testavimas.
3. **Arčiau tikrų vartotojų –** joks įrankis negali pakeisti žmogaus išmonės bei patirties. Rankiniu būdu testuojant galima pagauti daugiau specifiniu defektų, vykdant kitokius žingsnius, ar kitokia tvarka.
4. **Greičiau atnaujinami testavimo atvejai –** nereikia programuoti naujų testavimo atvejų, užtenka juos trumpai aprašyti.

Rankinis testavimas turi ne tik pliusų, bet ir minusų. Remiantis 2015m. IJETST [4] atlikta rankinio bei automatinio testavimo analize, galima išskirti keletą problemų, susijusių su rankiniu testavimu [Sha14, RM15]:

[4] IJETST - International Journal of Emerging Trends in Science and Technology

1. **Atima daug laiko** – kadangi viskas atliekama rankomis, įvykdyti kiekvieną testavimo atvejį kainuoja daug laiko.
2. **Nuobodus** - tuos pačius testavimo atvejus reikia kartoti keletą kartų, todėl tai gali atsibosti.
3. **Reikalauja daug žmogiškųjų išteklių –** kuo daugiau reikia ištestuoti rankiniu būdu, tuo daugiau testuotojų reikia.
4. **Mažiau patikima –** rankiniame testavime gali pasitaikyti žmogiškųjų klaidų.

Dėl šių rankinio testavimo trūkumų, vis dažniau pradėta naudoti kita testavimo technika – automatizavimas, tačiau jis turėtų padėti testavimo procese, o ne visiškai pakeisti rankinį testavimą. Anot DevOps metodikos, testuotojai turi būti lankstūs – mokėti testuoti ne tik rankiniu, bet ir automatiniu būdu [Gur163].

2.2.Automatinio testavimo privalumai bei trūkumai

Automatinis testavimas atliekamas kompiuteriu su minimaliu žmogaus įsikišimu. Testuoti galima dviem būdais - rašant kodą arba įrašinėjant veiksmus. Automatizavimas leidžia atlikti nurodytus testavimo atvejus, patikrinti tikėtinus bei esamus rezultatus, sugeneruoti detalias testų išvadas. Pagrindiniai automatinio testavimo privalumai yra šie [Sha14, RM15]:

1. **Greitas vykdymas** – testai vykdomi greičiau, negu juos atliekant rankiniu būdu.
2. **Galima vykdyti 24/7** – automatinis testavimas gali būti atliekamas bet kuriuo paros metu.
3. **Mažesnės išlaidos** – kadangi naudojami automatiniai įrankiai, mažiau testuotojų reikia, jie gali užsiimti sudėtingesnėmis užduotimis.
4. **Pernaudojamumas** – tie patys testai gali būti leidžiami ant skirtingų prietaisų/aplinkų.
5. **Patikimumas** – kiekvieną kartą atliekami tie patys žingsniai – išvengiama žmogiškųjų klaidų.
6. **Greiti rezultatai –** atsiradus naujai versijai, automatinis testavimas padeda greičiau sužinoti, ar versija tinkama – pateikia atlikto testavimo išvadas.

Tačiau vien privalumais automatinis testavimas neapsiriboja – pagal 2013m. parašytą straipsnį, kuriame buvo analizuojama automatinio testavimo ribotumas, rasta keletas minusų [Ale13, Kev12]:

1. **Trūksta stabilumo** – testai gali nepraeiti ne vien dėl atsiradusių defektų, bet ir dėl kitų priežasčių. Testuojant vartotojo sąsają, tai gali atsirasti dėl mažo dizaino pakeitimo, dėl neveikiančio (angl. down) serverio, ar dėl interneto sutrikimų. Šios problemos nėra defektai, bet lemia klaidingus testų rezultatus.

2. **Reikalauja daug techninių žinių** – automatinių testų rašymui reikia techninių žinių apie programavimą. Tam reikia apmokyti esamus darbuotojus, arba įdarbinti naujų specialistų.
3. **Didelės pradinės investicijos** – darbuotojų apmokymai bei nauji įrankiai – tai papildomos investicijos.
4. **Neįmanoma ištestuoti visko** – testų, tikrinančių vartotojo sąsajos draugiškumą (angl. user friendliness) ištestuoti neįmanoma.
5. **Nekokybiški įrankiai** – kaip taisyklė, pradėjus naudoti automatizavimo įrankį, randama daugiau jo defektų, negu testuojamos aplikacijos.

Agile, o ypač DevOps metodikose skatina kuo daugiau automatizuoti, nes tai puikiai įsilieja į jų siekiamus tikslus – greitį bei patikimumą, siekiant išgauti kuo didesnę kokybę. 2012m. buvo vykdoma apklausa tarp Agile metodikos pasekėjų, kurioje uždavė klausimą apie iššūkius, siekiant įgyvendinti Agile metodiką savo darbe. 50% atsakiusių patvirtino, kad didžiausias iššūkis atlikti visą reikiamą testavimą per vieną iteraciją/sprintą [Amb12]. DevOps bandydami užpildyti esamas Agile spragas bei žinodami šią bėdą, kaip vieną iš didžiausių tikslų išsikėlė automatizavimą. Nors Agile metodikoje, testuotojai, gavę naują versiją, atlieka ne tik rankinį, bet ir automatizuotą testavimą (dažniausiai automatizuota regresinis testavimas – patikrina seną funkcionalumą, kuris nesikeičia – tam, kad nereiktų parrašinėti esamų testų, bei atlieka bazinį (angl. smoke) testavimą, kuris vykdomas itin dažnai – po kiekvienos naujos versijos, patikrina, ar pagrindinis funkcionalumas veikia), tačiau DevOps propaguotojo Aniket Deshpande[5] nuomone, automatizuota turi būti viskas [Des17]. Anot šios metodikos, siekiamybė yra 100% kodo padegimas automatizuotais testais, kurie turi pasileisti automatiškai po kiekvienos naujos versijos. Geriausia, jei programuotojai ir testuotojai sėdėtų kartu – testuotojai nuolat būtų informuojami, kokia vieta buvo tvarkyta kiekvienoje versijoje ar kokius naujus testus reikia programuoti [Des17].

2.3.Testavimo technikų apibendrinimas

Rankinio testavimo nepakeis jokie įrankiai, tačiau jie gali palengvinti programų sistemų kūrimo procesą. Nors automatinis testavimas turi nemažai minusų, tokių kaip nekokybiški įrankiai ar stabilumo trūkumas, pliusai ištaiso rankinio testavimo trūkumus ir ilgesniame laikotarpyje atneša didesnę naudą. Agile metodikoje reikalaujama kuo daugiau automatizavimo, kad būtų įgyvendinti laiko apribojimai sprintams. Automatiniam testavimui reikia testavimo įrankių bei testuotojų, pagal

[5] Aniket Deshpande – testuotojų komandos vadovas, dirbantis AFour Technologies daugiau nei 9 metus bei konsultuojantis kitas įmones, įgyvendinančias DevOps metodologiją.

DevOps, puikiai išmanančių automatinį testavimą, tačiau šios investicijos palaipsniui atsiperka. 2016m. atliktoje apklausoje [Est16], 24% teigė, jog automatinio testavimo atsipirkimą mato iš karto, 24% atsakė, jog pamato per 6mėn., 28% per 6-12mėn., po vienerių metų (28%), o 9% teigė, jog automatinis testavimas jiems neatsipirko. Apibendrinus, 81% atsakiusiųjų pamato atsipirkimą per vienerius metus.

2.4. Pasiruošimas testų automatizavimui

Tam, kad būtų pasiekta aukščiau išvardinta automatinio testavimo nauda, reikia kruopščiai pasiruošti testavimui. 2014m. atliktoje literatūros analizėje apie DevOps [Amr14], išvadose buvo pabrėžta, jog norint išgauti didžiausią galimą naudą iš automatizavimo, itin svarbu samdyti kvalifikuotus testuotojus, kurie gerai išmano testų automatizavimą arba surengti kokybiškus mokymus esamiems darbuotojams. Remiantis 2016m. statistika [Est16], 65% automatinių testų parašyti automatinių testų specialistų, 29% - rankinio testavimo specialistų bei iki 6% sudarė programuotojai. Turint tinkamus darbuotojus, reikia pereiti keletą etapų. Kiekviename etape atliekama tam tikra veikla ir gaunamas rezultatas[Zen14]:

1. **Tinkamumo analizė** – reikia išsiaiškinti, ar apskritai pasirinktos aplikacijos testavimas gali būti automatizuotas.
2. **Įrankio išsirinkimas** – kitas, vienas svarbiausių žingsnių, išsirinkti tinkamą įrankį. Tai priklauso nuo to, su kokia technologija aplikacija buvo kurta, įrankio funkcionalumo bei naudojamumo. Šioje stadijoje dažniausiai išsirenkama keletas tinkamiausių įrankių.
3. **Išsiaiškinti reikalavimus** – išsiaiškinti, ar įrankis gali palaikyti didžiąją dalį testuojamos aplikacijos funkcionalumo, apibrėžti norimos ištestuoti dalies apimtį.
4. **Automatizavimo planas** – šioje fazėje apibrėžiamas pasirinktas įrankis, nusprendžiama dėl licencijos (jeigu įrankis mokamas), nustatomi reikalingi resursai, sudaromas testų rašymo grafikas, tikėtini rezultatai.
5. **Testų automatizavimas** – pasiruošiama aplinka ir vyksta testų rašymas.

Sekančioje darbo dalyje aprašyta, kaip praktiškai buvo išbandyta keletas skirtingų automatizavimo įrankių tam pačiam internetiniam puslapiui testuoti, aptarti rasti pliusai, bei minusai.

3. AUTOMATINIO TESTAVIMO ĮRANKIAI

Automatizavimo įrankiai skiriasi pagal testavimo tipus, kurie skaidomi į keletą pagrindinių dalių:

3.1. Automatinių testų tipai

1. **Moduliniai testai** – jie tikrina kodo vienetus atskirai vieną nuo kito. Kiekvienas kodo vienetas (funkcija, klasė, metodas) turi savo vieną ar kelis testus, kuriuos dažniausiai rašo programuotojas.

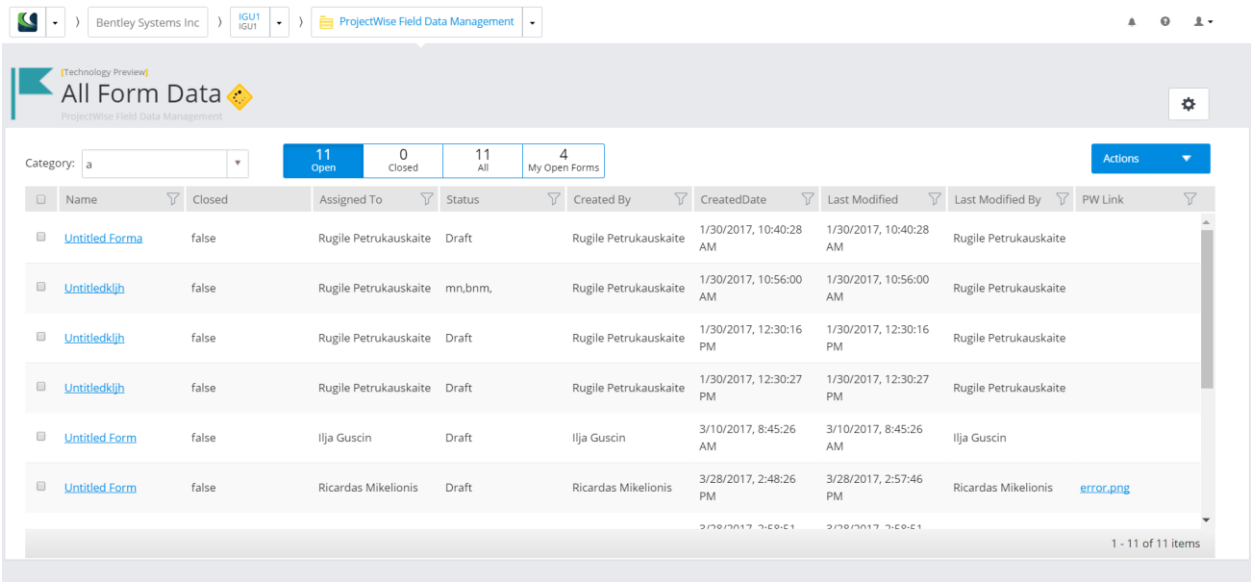
2. **API (angl. Application Programming Interface) testai** – jie tikrina verslo logiką, ar funkcionalumas, skaitomumas, vykdymas (angl. performance) bei saugumas atitinka duotus reikalavimus

3. **GUI (angl. Graphical User Interface)** – tai vartotojo sąsajos testavimas, kuris yra vienas iš sunkiausiai įgyvendinamų dėl dažno sąsajos kitimo, tačiau kartu jis arčiausiai vartotojo daromų veiksmų. Automatizuojama pelės bei klaviatūros paspaudimai, rašymas į laukelius. Tai padeda greitai rasti defektus daugeliu atvejų, pavyzdžiui atliekant regresinį testavimą, ar pildant ilgas formas, kas užima daug laiko.

Praktinėje darbo dalyje pasirinkta rašyti trečiojo tipo - vartotojo sąsajos, automatinius testus daliai internetinio puslapio su skirtingais įrankiais – „Selenium“, „CODED UI“ bei „TestComplete“. Toliau apie juos pateikta pagrindinė informacija, aprašyta, su kokiomis problemomis teko susidurti, kokie privalumai ir trūkumai pastebėti bei kokias išvadas galima daryti.

3.2. Internetinio puslapio testavimo atvejai

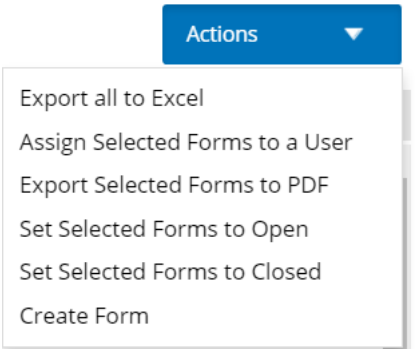
Buvo pasirinkta testuoti https://connect.bentley.com puslapio dalį (2 pav.), kurioje galima daryti įvarius veiksmus su formomis – jas pildyti, peržiūrėti jau sukurtas, eksportuoti į kitus formatus, priskirti vartotoją, uždaryt/atidaryti ir kita.



2 pav. Pasirinkto puslapio *https://connect.bentley.com* testuota dalis

Testavimui buvo pasirinkti šie testavimo atvejai:

1. Patikrinti vartotojo sąsajos elementus:
   1.1. Antraštę bei paraštę
   1.2. Pavadinimą bei paantraštę
1. Prisijungimas – tikrinama, ar po prisijungimo atidarytas teisingas puslapis.
2. Formų eksportavimas į Excel formatą per „Actions“ mygtuką (3 pav.) – tikrinama, ar failas atsisiųstas.



3 pav. „Actions” mygtuko testuojami veiksmai,

https://connect.bentley.com

3. Formos priskyrimas vartotojui per „Actions“ mygtuką – tikrinama, ar forma priskirta.
4. Formų eksportavimas į PDF formatą per „Actions“ mygtuką – tikrinama, ar dialogas, pranešantis apie failo atsisiuntimą, atsirado.
5. Formos atidarymas – tikrinama, ar po formos statuso pakeitimo pasikeitė atidarytų bei uždarytų formų skaičius.

6. Formos uždarymas – tikrinama, ar po formos statuso pakeitimo pasikeitė atidarytų bei uždarytų formų skaičius.
7. Formos pildymas – patikrinama, ar po formos užpildymo ir išsaugojimo laukuose teisingos reikšmės.

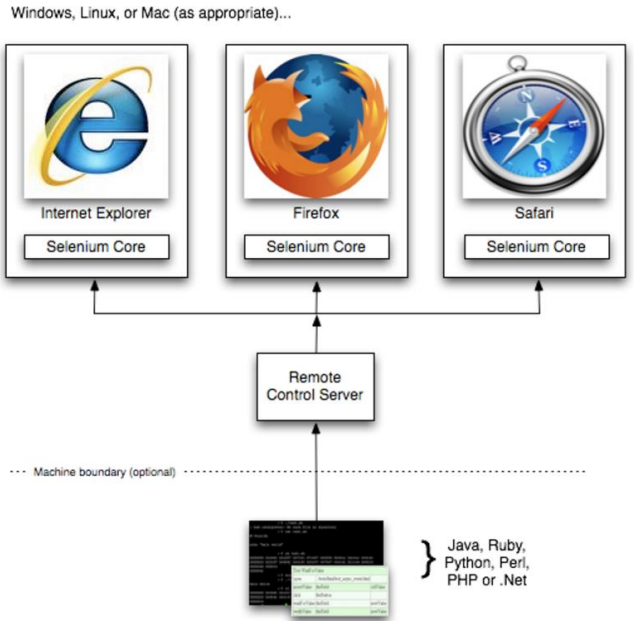
3.3. Selenium

Selenium yra vienas iš populiariausių laisvai prieinamo kodo (angl. open source) internetinių puslapių automatinio testavimo įrankių. Jis plačiai naudojamas testuoti grafinę vartotojo sąsają bei įvairių industrijų (elektroninės komercijos, biotechnikos, farmacijos ir kitų) internetinių puslapių funkcionalumą [SMS+14].

Šis įrankis palaiko visas svarbiausias platformas – Windows, MacOS ir Linux. Rašyti kodą galima įvairiomis programavimo kalbomis – Java, Ruby, Perl, C# ir Python. Darbe buvo naudojama C# „Visual Studio Enterprise 2015“ aplinkoje, testavimas buvo atliktas naudojant Chrome naršyklę. Selenium taipogi palaiko ir kitas populiarias naršykles - Firefox, IE, Safari, Opera, Safari.

Selenium yra kelių skirtingų įrankių kolekcija:

1. **„Selenium Integrated Development Environment (IDE)“** – tai Firefox naršyklės įskiepis, kuris leidžia testuotojams įrašyti veiksmus ir taip formuoti testus. Įrašai paverčiami į pasirinktos programavimo kalbos kodą ir gali būti leidžiami automatiškai. Kadangi įrašinėjimas galimas tik ant FireFox naršyklės, darbe jis nebuvo naudojamas, nes testuotas puslapis palaiko tik dvi naršykles – Chrome ir Internet Explorer.
2. **„Selenium Remote control (RC)“** – leidžia rašyti automatinius UI testus bet kokia programavimo kalba, bet kuriam HTTP/HTTPS puslapiui. Jį sudaro dvi dalys – serveris, kuris atidaro ir uždaro naršykles, bei klientinės bibliotekos, skirtos bet kuriai programavimo kalbai. Šios architektūros reprezentacija pavaizduota 4 paveiksle [SMS+14]:



4 pav. Selenium Remote Control dalys, *Selenium dokumentacija*

3. **„Selenium WebDriver“** – Selenium RC tęsėjas, kuris siunčia komandas naršyklei ir gauna rezultatus. Šis įrankis bendrauja tiesiogiai su naršykle, kas vyksta greitai, priešingai negu Selenium RC, kuris yra priklausomas nuo serverio. Būtent šis įrankis buvo naudojamas testuojant aplikaciją. Šios architektūros reprezentacija vaizduojama 5 pav.:

5 pav. Selenium WebDriver dalys, *Selenium dokumentacija*

4. **„Selenium Grid“** – įrankis, skirtas paraleliai testuoti ant skirtingų kompiuterių ir skirtingų naršyklių, kas padeda sutaupyti laiko.

Nors produktas yra laisvai prieinamas ir neturi oficialaus palaikymo, tačiau internete gausu dokumentacijos apie patį įrankį. Taipogi yra nemažai diskusijų, kuriose kalbama apie esamas problemas ir jų sprendimo būdus.

Testavimo aplinkos pasiruošimas paprastas - reikia įsirašyti IDE (darbe buvo naudojama „Visual Studio Enterprise 2015“), ir atsisiųsti įrankį pagal poreikį – dažniausiai naudojamas Selenium WebDriver. Kadangi, įrašinėjimas galimas tik ant Firefox naršyklės, kurios darbe testuota aplikacija nepalaiko, todėl buvo apsiribota tik Selenium WebDriver. Taipogi reikėjo atsisiųsti papildinius atskirom naršyklėms paleisti.

Tam, kad kodas būtų struktūrizuotas ir pernaudojamas, jį reikia organizuoti. Remiantis 2013m. metais parašytu straipsniu, patogiausias būdas struktūrizuoti kodą rašant automatinius testus su Selenium - naudoti „Page Object“ šabloną [LCR+13]. Šis šablonas padeda sumažinti klasių ir testų prisirišimą, atsiranda didesnė abstrakcija, o tai reiškia, kad atsiradę pokyčiai aplikacijoje nesugriaus visų testų. Šio šablono naudojimui būtina gerai išmanyti pagrindinius objektinio programavimo principus.

Rašant automatinius testus prireikė nemažai techninių žinių. Html objektai buvo ieškomi naudojant unikalius identifikatorius kaip Id, ClassName, LinkText, tačiau ne visi objektai turėjo nors viena iš šių. Šiai problemai spręsti testuose naudota CssSelector bei Xpath (6 pav.):

```
//Find Category text elements
categoryElement = driver.FindElement(By.XPath("//span[contains(., 'Category:')]"));
categoryText = categoryElement.Text;
categoryExpectedText = "Category:";

//Find Categories List element
helper.WaitForLoadFinish(6000);
categoriesListElement = driver.FindElement(By.CssSelector("span.k-dropdown-wrap.k-state-default"));
```

6 pav. CssSelector bei XPath naudojimo pavyzdžiai ieškant objektų,

paimta iš praktinės darbo dalies

Teste, kuriame reikėjo tikrinti užpildytos formos laukų reikšmes (7 testas), prireikė žinių apie JavaScript bei Jquery, nes laukų reikšmės buvo saugomos ne Html, o JavaScript kode (7 pav.):

```
//Find form fields values
datePickerFieldEnteredValue = (string)jsDriver
    .ExecuteScript("return document.getElementsByClassName('k-picker-wrap k-state-default')[0].children[0].value;");
numberFieldEnteredValue = (string)jsDriver
    .ExecuteScript("return document.getElementsByClassName('flow-layout ng-pristine ng-valid')[1].children[0].children[0].value;");
```

7 pav. Jquery naudojimo pavyzdys ieškant objektų,

paimta iš praktinės darbo dalies

Atliekant darbą rasti šie privalumai:

1. Internete lengva rasti atsakymus į iškilusius klausimus bei problemų sprendimus.
2. Palyginus su CODED UI ar TestComplete automatinio testavimo įrankiais, visi testai įvykdavo greičiausiai (2 priedas):
    a. Selenium ~7min.
    b. CODED UI ~ 9min.
    c. TestComplete ~8min.
3. Techniniam žmogui lengvai perprantama sintaksė, paprastos bei naudingos funkcijos.
4. Tereikia atsisiųsti papildinį, kad būtų galima leisti testus ant skirtingų naršyklių.

Taipogi rasta ir keletas minusų:

1. Testuotoje aplikacijoje reikėjo patikrinti formų eksportavimą į .xlsx bei .pdf formatus, tačiau „Selenium“ apima tik internetinius puslapius ir eksportavimo/talpinimo nepalaiko.
2. Įrašinėti testus galima naudojant tik vieną naršyklę, kas yra didelis apribojimas netechniniams testuotojams, jeigu jų aplikacija tos naršyklės nepalaiko.
3. Leidžiant tą patį testą kelis kartus, retkarčiais jis nepavykdavo dėl nerasto objekto aplikacijoje.
4. Kadangi nėra palaikymo, testuojant rasta įrankio defektų, kurie yra užrašyti prieš keletą metų, tačiau neaišku, ar bus ištaisyti.
5. Nors testų vykdymas vyksta gana greitai, pats kodo rašymas užtruko ilgiausiai (įskaitant mokymąsi bei dokumentacijos skaitymą, 2 priedas):
    a. Selenium ~ 7dienos
    b. CODED UI ~ 5dienos
    c. TestComplete ~3dienos

3.4. CODED UI

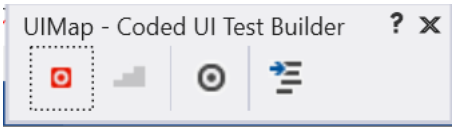
Tai gana naujas testavimo įrankis programinės įrangos prekyboje. Jis atsirado kaip dalis “Visual Studio 2010” ir palaiko C# bei Visual Basic .NET. Anot Microsoft, kaip ir Selenium, CODED UI įrankis leidžia patikrinti visą aplikaciją, įskaitant vartotojo sąsają ir funkcionalumą [Msd15].

Šis įrankis leidžia testuoti ne tik internetinius puslapius, bet ir „Windows Forms“, „Windows Presetation Foundation“ bei Windows aplikacijas, parašytas su XAML. Būtinas reikalavimas – „Visual Studio Ultimate“, „Visual Studio Enterprise“, „Visual Studio Premium“, kas yra mokama.

Palaikomos naršyklės – Chrome, Firefox, IE, Opera ir Edge. Darbe naudota „Visual Studio 2012 Premium“, testuota naudojant naujausią IE versiją.

CODED UI įrankis leidžia rašyti testus dviem būdais [Gur162]:

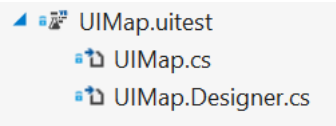
1. **„Test Builder“** – šis įrankis skirtas įrašinėti žingsnius (8 pav.):



8 pav. „Test Builder“ įrankis,

paimta iš Visual Studio 2012 Premium

Jis leidžia įrašyti žingsnius, juos peržiūrėti/trinti, išsaugoti objektą, patikrinti jo atributus bei generuoja kodą. Įrašinėti galima tik ant IE naršyklės. Baigus įrašinėjimą, „Test Builder“ sukuria žemėlapį (ang. map), kuris sukuria failus (9pav.):



9 pav. CodedUITest1.cs**,**

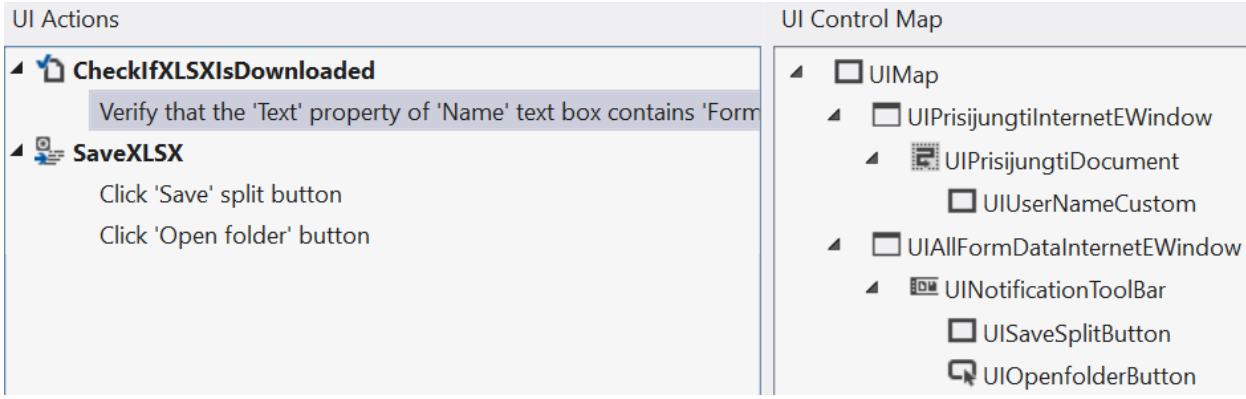
paimta iš Visual Studio 2012 Premium

1. 1. **UIMap.Designer.cs –** čia saugomas sugeneruotas kodas iš testuotojo atliktų žingsnių – įvairūs metodai, tikrinimai, objektų deklaracijos. Šio failo redaguoti negalima, nes po kiekvieno naujai įrašyto žingsnio, šis failas pergeneruojamas.

1. 2. **UIMap.cs -** šis failas būna tuščias. Jame testuotojas gali rašyti sau reikiamus metodus.

1. 3. **CodedUITest1.cs** – šis failas skirtas formuoti testus. Jame vykdomi visi tikrinimai (angl. assertions) ir metodai.

1. 4. **UIMap.uitest –** jame yra XML reprezentacija UIMap klasės. Jame yra saugomi langai, kontroleriai, metodai, veiksmai, tikrinimai, 10 pav.:



10 pav. UIMap.uitest,

paimta iš Visual Studio 2012 Premium, praktinės dalies

Darbe įrašinėjimas buvo naudojamas tik kai kurių objektų aptikimui bei testuojant .xlsc/.pdf dokumentų eksportavimą.

2. **Rašyti kodą rankomis** – kaip ir su Selenium, viską rašyti rankomis – paleisti naršyklę, rasti objektus, daryti veiksmus (pelė paspaudimai, teksto įvedimas į laukelius ir kita).

Internete galima rasti nemažai Microsoft dokumentacijos apie šį įrankį – kaip įsirašyti, kaip pasileisti bei kaip naudotis, tačiau mokymų (ang. tutorial) nepakankamai. Diskusijų apie esamas problemas ir jų sprendimo būdus taipogi trūksta.

Testavimo aplinkos pasiruošimas paprastas - reikia įsirašyti IDE (darbe buvo naudojama laikina (angl. trial) versija „Visual Studio Premium 2012“) ir susikurti CODED UI projektą. Buvo bandoma testuoti ir ant „Visual Studio Enterprise 2017“, tačiau UIMap.uitest įrankis turėjo defektų (buvo neįmanoma redaguoti šio failo), todėl IDE teko pasikeisti į senesnę.

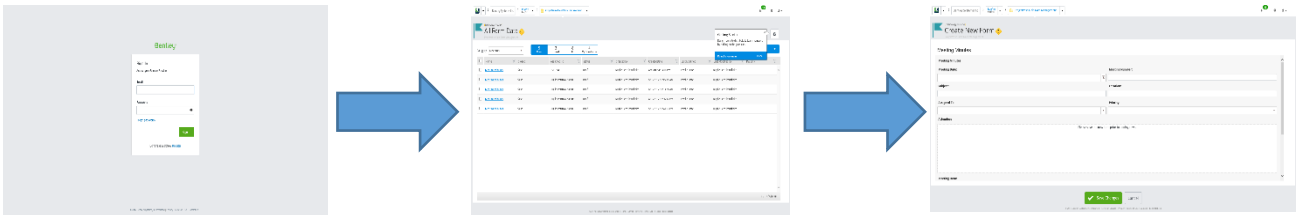
Visų pirma, buvo bandoma testuoti naudojant tik Test Builder, tačiau šiuo būdu kodas nėra struktūrizuojamas, visi metodai, tikrinimai bei objektai sudedami į vieną failą. Dėl šios priežasties buvo nuspręsta atsisakyti naudoti tik Test Builder. Anot Marcel de Vries (IT specialistas, rašantis automatinius testus virš 7metų, bei „Pluralsight” puslapio mokymų autorius [Vri15]) automatiniai testai, kaip ir programos kodas, turi atitikti tam tikrus reikalavimus - DAMP (angl. Descriptive And Meaningful Phrases) principus testų scenarijams. Štai vienas DAMP testo pavyzdys iš atlikto darbo (11 pav.):

```
[TestMethod]
public void AssignUserThroughActionsButtonTest()
{

    var browserWindow = BrowserWindow.Launch(new Uri("https://qa-
    FDMPage fdmPage = new FDMPage(browserWindow);

    Assert.IsTrue(fdmPage.CloseGetStartedNotification()
        .ClickOnTheFormInGrid()
        .ExpandActionsButton()
        .ClickToAssignUserOnActionsButton()
        .EnterUserName()
        .ClickUpdareFormsToAssignUser()
        .IsUserAssigned()
        , "Failed to assign user through Actions button");

}
```

11 pav. DAMP principus atitinkantis testas,

paimta iš darbo praktinės dalies

Tam, kad pasiektume DAMP principus, reikia abstrakcijos šablono, kuris būtų tinkamas UI automatiniam testavimui. „Page Object“ šablonas būtent toks. Darbe šablonas buvo pritaikytas tokiu būdu (12pav.):



Abstrakcija: LogInPage

Abstrakcija: FDMPage

Abstrakcija: MeetingMinutesPage

```
public FDMPage LoginWithSendKeys(string name, string psw)
{
    return new FDMPage(_bw);
}
```

```
public MeetingMinutesPage ClickToCreateForm()
{
    return new MeetingMinutesPage(_bw);
}
```

```
public bool AreFormValuesCorrect()
```

12 pav. *„Page Object“*šablono įgyvendinimas

Kiekvienam puslapiui sukurta klasė – abstraktus bendravimas su puslapiu. Kiekvienam testui sukurtas metodas, kuris atlieka tam tikrus žingsnius ir grąžina teisingą/klaidingą (ang. true/false) reikšmes. Marcel de Vries savo mokymuose teigia, kad tai vienas geriausių pasirinkimų, nes

kiekvienas puslapio objektas turi tik vienos funkcinės dalies informaciją bei pasikeitimai puslapyje nesugriaus visų testų [MV14].

Atlikus automatinį testavimą CODED UI įrankiu, rasti šie pliusai:

1. Netechniniam žmogui gali būti gera pradžia įrašinėjimas su „TestBuilder“.
2. Rašant kodą, objektų ieškojimas lengvesnis– ieškoma pagal objekto požymius (angl. attribute) HTMLE‘e (13 pav.). Lyginant su Selenium, CODED UI lengviau išmokstamas ir užtenka mažiau techninių žinių, nereikia rašyti dažnai ilgų CssSelector ar Xpath išraiškų, 13 pav. pateiktas CODED UI objekto radimas:

```
public HtmlControl GetFDMTitle()
{
    HtmlControl fdmTitle = new HtmlControl(_bw);
    fdmTitle.SearchProperties.Add(HtmlDiv.PropertyNames.InnerText, "All Form Data");

    return fdmTitle;
}
```

13 pav. objektų radimas su CODED UI įrankiu,

paimta iš praktinės dalies

3. Priešingai negu Selenium, jis neapsiriboja vien tik internetiniais puslapiais, todėl buvo įmanoma ištestuoti formų eksportavimą.
4. Jei rašant kodą sunku surasti objektą, galima naudoti „TestBuilder“ objektų aptikimui.
5. Testai yra stabilūs – jei viena kartą įvyko be klaidų, juose ir neatsiras klaidų toje pačioje versijoje. Tai didelis pliusas, lyginant su Selenium, kuriame atsirasdavo netikėtų klaidų (angl. random) ir testai sugriūdavo dėl neaiškios priežasties.
6. Jei nors vienas testas nepavyko, automatiškai sukuriamas aplankas pavadinimu „TestResults“, kuriame talpinama informacija apie nepavykusius testus su darbalaukio nuotraukomis.

Šio įrankio minusai:

1. Su „Visual Studio Enterprise 2017“ testuoti sunkiau, nes dėl defektų negalima redaguoti UIMap.uitest failo.
2. Nors produktas ir yra palaikomas, atliekant darbą buvo rasta ganėtinai rimtų klaidų, kurios žinomos daugiau nei 2metus, tačiau dar vis neištaisytos. Pvz: atidarant naršyklę, puslapis „pakibdavo“ ir jokie kiti testai negalėjo būti pakeisti (išeitis buvo rasta – susikurti app.config failą ir pakeisti keletą nustatymų). Kitas defektas – dažnai atsirasdavo klaidų dėl mygtuko,

uždengto objektu, kuris iš tikrųjų visai neuždengtas. Apėjimas – spausti ant mygtuko vidurio, kaip ant tam tikro taško puslapyje (14 pav.):

```
public FDMPage ClickExportToExcellOnActionsButton()
{
    HtmlControl btn = new HtmlControl(_bw);
    btn.SearchProperties.Add(HtmlControl.PropertyNames.Class, "ng-scope openleft");
    UITestControlCollection collection = btn.FindMatchingControls();
    collection[0].DrawHighlight();

    Point location = collection[0].BoundingRectangle.Location;
    location.Offset(collection[0].BoundingRectangle.Width / 2,
                    collection[0].BoundingRectangle.Height / 2);

    Mouse.Click(location);
    return this;

}
```

14 pav. objektų paieška su CODED UI įrankiu,

paimta iš praktinės dalies

3. Testų atlikimo greitis lėtesnis, nei su Selenium (2 priedas).
4. Bandant įrašinėti testus su „Test Builder“, pastebėta nemažai trūkumų:
    a. Atsiranda didelis pagrindinių programavimo principų neatitikimas, nes viskas – objektai, tikrinimai, metodai – sudėta į vieną failą.
    b. Dirbti su komanda viską įrašinėjant į vieną failą ganėtinai sunkus uždavinys.
    c. Kiekvieną kartą pridėjus naują įrašą visas failas yra perrašomas, kas lėtina darbą.
    d. Atsiras daug dublikatų, nes kelios veiksmų sekos gali naudotis tuo pačiu objektu.
    e. Pasikeitus daliai funkcionalumo, gali sugriūti visi testai.

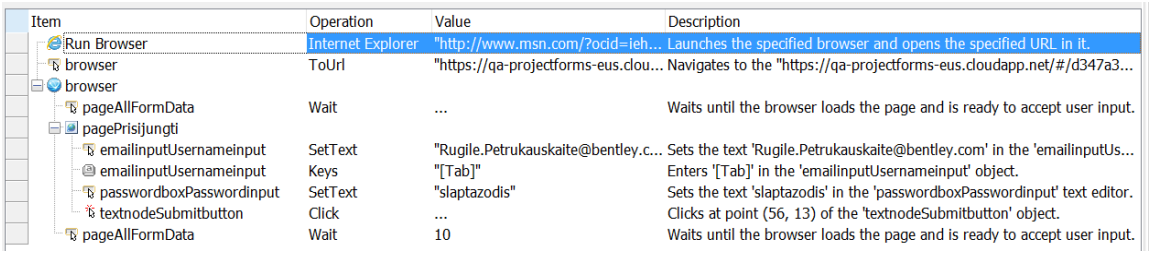
3.5. TestComplete

TestComplete sukūrė „SmartBear“ kompanija, 1999m. Juo galima testuoti ne tik internetinius puslapius, bet darbalaukiu bei mobilias aplikacijas. Galimos programavimo kalbos – VBScript, Jscript, DelphiScript, C#Script, Python, VB. Galima testuoti naudojant šias naršykles – IE, Firefox, Chrome ir Edge.

Šis įrankis ganėtinai brangus, pradinė kaina 1000e (priklauso nuo licencijos tipo, norimų paketų kiekio). Iš jau aptartų įrankių, šis yra brangiausias, tačiau jis turi savo IDE ir daug, palengvinančių darbą savybių (angl. features) [Sma17]. Darbe naudota laikina versija, leidžianti naudotis produktu 30 dienų. Pasirinkta naršyklė – IE.

Šiuo įrankiu galima testus įrašinėti arba rašyti kodą.

1. **Įrašinėjimas** - pats įrašinėjimas vyksta kaip su „Test Builder“, tačiau skiriasi sugeneruota išeitis. Įrašius testą, yra sugeneruojamas ‚keyword test‘ (žemiau pateiktam pavyzdyje testuojamas prisijungimas, 15 pav.):



Item	Operation	Value	Description
Run Browser	Internet Explorer	"http://www.msn.com/?ocid=ieh...	Launches the specified browser and opens the specified URL in it.
browser	ToUrl	"https://qa-projectforms-eus.clou...	Navigates to the "https://qa-projectforms-eus.cloudapp.net/#/d347a3...
browser			
pageAllFormData	Wait	...	Waits until the browser loads the page and is ready to accept user input.
pagePrisijungti			
emailinputUsernameinput	SetText	"Rugile.Petrukauskaite@bentley.c...	Sets the text 'Rugile.Petrukauskaite@bentley.com' in the 'emailinputUs...
emailinputUsernameinput	Keys	"[Tab]"	Enters '[Tab]' in the 'emailinputUsernameinput' object.
passwordboxPasswordinput	SetText	"slaptazodis"	Sets the text 'slaptazodis' in the 'passwordboxPasswordinput' text editor.
textnodeSubmitbutton	Click	...	Clicks at point (56, 13) of the 'textnodeSubmitbutton' object.
pageAllFormData	Wait	10	Waits until the browser loads the page and is ready to accept user input.

15 pav. Testavimas TestComplete įrankiu,
paimta iš praktinės dalies

Galima lengvai pakeisti naršyklę, perrikiuoti žingsnius, įterpti naujus, ištrinti nereikalingus, keisti reikšmes, uždėti laiko buferius, daryti ciklus ir daug kitų naudingų funkcijų. Būtent šis testų kūrimo įrankis buvo naudojamas darbe.

2. **Rankinis kodo rašymas** – Įrašytus testus galima eksportuoti į pasirinktą programavimo kalbą. Priešingai, negu su CODED UI įrankiu, kiekvienas testas turi savo atskirą failą. Pats kodo rašymas mažai tesiskiria nuo ankščiau minėtų įrankių.

„SmartBear“ puslapyje yra labai daug dokumentacijos įvairiausiais klausimas. Gausu ne tik parašytų, bet ir video mokymų, kuriuose žingsnis po žingsnio aprašyta eiga, įvairių problemų sprendimo būdai. Taipogi galima įsigyti keleto savaičių apmokymų kursus.

Testavimo aplinkos paruošimas neužėmė daug laiko – reikėjo įsirašyti IDE (kiekvienas žingsnis smulkiai aprašytas dokumentacijoje) bei aktyvuoti laikiną licenciją.

IDE suteikia daug naudingo funkcionalumo – gali pasirinkti, kad būtų daroma kiekvieno įrašyto/leidžiamo žingsnio nuotrauka. Po testų įvykdymo pateikiama išsami analizė su diagrama – kiek testų pavyko, kiek nepavyko, kiek laiko buvo leidžiama. Nepavykusiems testams padaromos darbalaukio nuotraukos su apibrėžtu objektu, dėl kurio įvyko klaida.

Atliekant darbą, rasti šie privalumai:

1. Lengva organizuoti testus – juos galima keisti vietomis, pernaudoti, grupuoti. Galima nustatyti, kad nepavykus kai kuriems testam būtų nenutraukiamas testų leidimas ir atvirkščiai. Galima uždėti įvairias sąlygas – testą kartoti keletą kartų, po testo atlikimo uždėti laiko buferį.

2. Galima parametrizuoti duomenis, kad nebūtų „hard coded" reikšmių. Galima užkrauti duomenis iš įvairių failų.
3. Įrašinėjant testus reikia mažai techninių programavimo žinių.

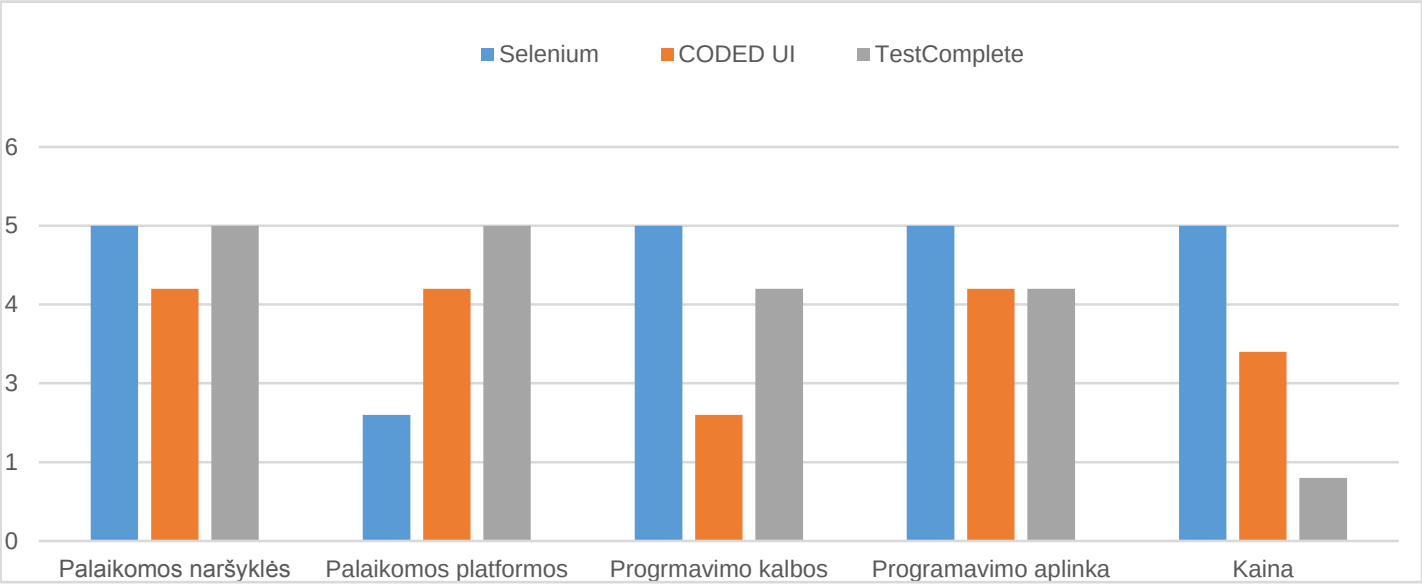
Taipogi aptikta ir keletas minusų:

1. Nors įrankis ir palaikomas, tačiau dažnai leidžiant testus atsikartoja problemos: naršyklė ne visada paleidžiama iš pirmo karto bei atsirandančios klaidos dėl nerasto objekto.
2. IDE veikia lėtai, dažnai stringa, „pakimba". Kuo daugiau funkcionalumo naudojama, pavyzdžiui kiekvieno žingsnio darbalaukio fotografavimas, tuo lėtesnis darbas.
3. Testai vykdomi lėčiau negu su Selenium (2 priedas).

3.6. Testavimo įrankių apibendrinimas

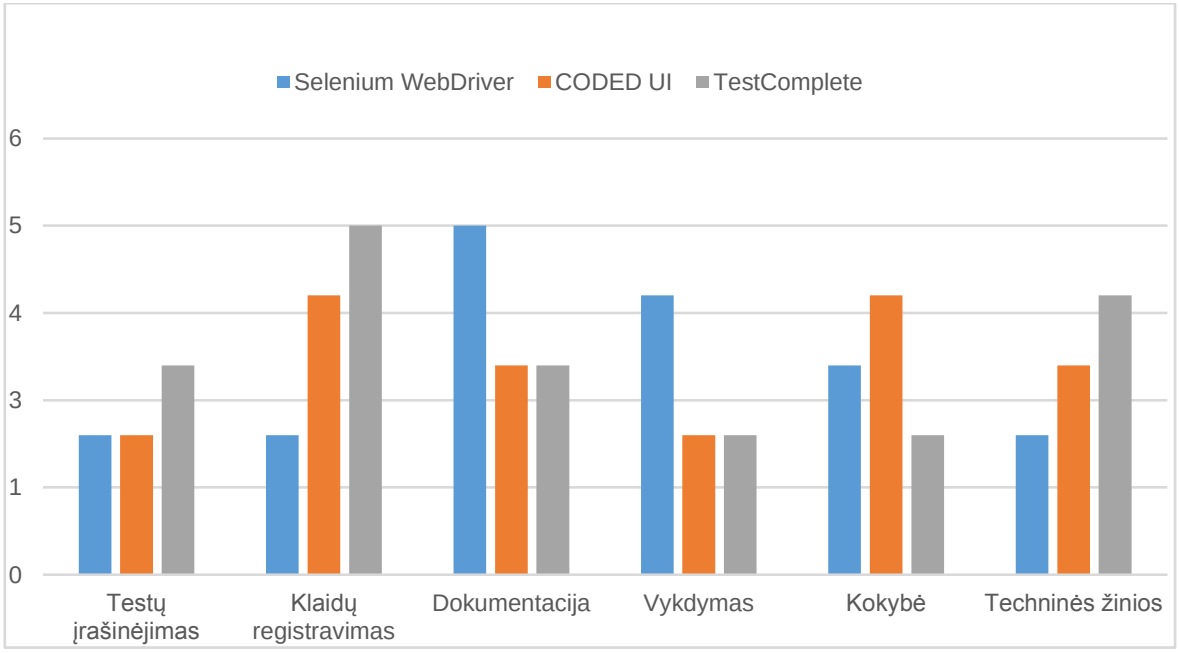
Internete galima rasti daugybę siūlomų automatinio testavimo įrankių. Pagrindinės jų funkcijos yra panašios, jie skirias techninėmis savybėmis, naudojamumu [KG15].

Techninių savybių palyginimo diagramoje (16 pav.) pagal techninių savybių palyginimo lentelę (1 priedas) galima pastebėti, jog Selenium (22 balai) įrankis pirmauja pagal beveik visus aspektus – išskyrus palaikomas platformas, šis įrankis apsiriboja tik internetinių puslapių testavimu. Antrąją vietą užima TestComplete (19 balų) , tačiau kaina stipriai sumažina jo pranašumą. Paskutinis, bet ne prasčiausias, likęs CODED UI (17 balų) įrankis, kurio didžiausias apribojimas – mažai palaikomų programavimo kalbų.



16 pav. Techninių savybių palyginimas,
remiantis per praktinę dalį sudaryta techniių savybių lentele (1 priedas)

Vertinant vartotojo patirtį, naudojant šiuos įrankius (17 pav.), galima teigti, jog visi įrankiai ganėtinai panašūs (2priedas) . Pirmoje vietoje TestComplete (19 balų), jo silpnosios vietos vykdymo greitis bei kokybė. Antrąją vietą dalinasi Selenium su CODED UI (18 balų). Selenium silpnosios vietos yra testų įrašinėjimas, klaidų registravimas bei reikalingos nemažos techninės žinios. CODED UI minusai - lėtas testų vykdymas bei prastas testų įrašinėjimas.



17 pav. Praktinės dalies palyginimas,

remiantis per praktinę dalį sudaryta praktinės dalies palyginimo lentele (2 priedas)

Negalima teigti, jog kažkuris iš šių įrankių blogesnis, o kažkuris grasesnis. Vienas funkcionalumas reikalingas viename projekte, visiškai kitas naudingas kitame.

4. REKOMENDACIJOS RENKANTIS AUTOMATIZAVIMO ĮRANKĮ

Prieš pradedant ieškoti tinkamo automatizavimo įrankio, svarbu susikurti sąrašą reikalavimų, pagal kuriuos bus vertinamas įrankis. Neturėjimas tokių reikalavimų gali lemti laiko švaistymą atsisiunčiant, įsirašant bei vertinant įrankius, kurie tenkina tik kelis reikalavimus, o gal ir nei vieno iš jų [Mun14]. Toliau darbe pateikta keletas klausimų, į kuriuos reiktų atsižvelgti renkantis automatizavimo įrankį:

1. **Ar aplikacija apskritai tinkama testavimo automatizavimui?**

   Testuojant UI/funkcijas svarbu, kad kiekvienas objektas turėtų identifikatorių – ID, kuris būtų unikalus ir niekada nekintantis. Jeigu projektas tik pradeda savo gyvavimo ciklą, to galima pasiekti programuotojų pagalba ganėtinai nedideliais kaštais. Jei ID yra dinaminis, testai gali greitai dėl to sugriūti. Atliekant praktinę dalį, ne visi objektai turėjo unikalų identifikatorių. Tai sukėlė papildomų rūpesčių ir sugaišino nemažai laiko – su Selenium gavosi sudėtingos Xpath išraiškos, o keletą kartų buvo naudota CssSelector (tai nėra gerai, nes pasikeitus stiliui, testai gali nustoti veikti). Su CODED UI ši problema buvo išspręsta paprasčiau, objektų buvo ieškoma pagal daugiau ypatybių arba pasitelkiant įrašinėjimą. TestComplete tai įtakos neturėjo.

2. **Kokias platformas/naršykles palaiko**?

   Tai sekantis klausimas, į kurį reikėtų atsakyti – po to dauguma įrankių atkrenta kaip netinkami dėl nepalaikomų naršyklių. Tai priklauso nuo norimos ištestuoti aplikacijos – kokios naršyklės yra palaikomos, ar yra/planuojama integracija su kitomis platformomis (pvz. Windows langais – failų atsisiuntimas, duomenų talpinimas) ir kita. Jei taip, reikėtų rinktis įrankį, kuri palaiko daugiau platformų. Šiuo atžvilgiu Selenium įrankis atkristų, nes jis palaiko tik internetines aplikacijas. CODED UI bei TestComplete būtų tinkamesni, nes palaiko ne tik internetines, bet ir Windows aplikacijas.

3. **Kuriuos testus norima automatizuoti?**

   Agile skatina kruopščiai atsirinkti, ką testuoti. Pagal Agile testuotoją JoEllen West[6], jei testas atliekamas vieną ar du kartus, jis neturėtų būti automatizuojamas. Tokius testus pigiau atlikti rankiniu būdu. Anot jos, geriausias pasirinkimas yra automatizuoti bazinius testus, kurie dažniausiai vykdomi po kiekvienos naujos versijos, kurios,

[6] JoEllen West – Agile testuotoja is www.VersionOne.com

dirbant pagal Agile metodika, atsiranda kas 1-2 dienas bei sugaišina nemažai laiko testuojant rankiniu būdu. Taip pat patartina automatizuoti regresinius testus, kurie patikrina pagrindinį funkcionalumą ir yra retai kintantys. Tačiau DevOps metodikoje siekiama, jog automatiniai testai kodą padengtų 100% ir vietoj rankinio testavimo, testuotojai rašytų kodą, testuoti naujam funkcionalumui [Isa15, Des17].

4. **Ar yra kvalifikuotų darbuotojų?**

   Svarbu išsiaiškinti, kokių įgūdžių turi esami testuotojai, ar jie bus pajėgūs automatizuoti naudojant techninių žinių reikalaujančius įrankius. JoEllen West pabrėžia, jog iš Krioklio metodologijos į Agile pereinančios komandos susiduria su didele problema dėl trūkstamų žinių esamiems testuotojams. 2014m. atliktoje literatūros analizėje apie DevOps [Amr14], išvadose pabrėžta, jog norint išgauti didžiausią naudą iš automatizavimo, itin svarbu samdyti kvalifikuotus testuotojus, kurie gerai išmano testų automatizavimą arba surengti kokybiškus mokymus esamiems darbuotojams.

5. **Kiek resursų skiriama?**

   Automatizavimo įrankiai yra dviejų rūšių – laisvai prieinami (nemokami) bei komerciniai (kaina priklauso nuo licencijos tipo). Internetiniame seminare apie automatinį testavimą Agile metodikoje [Bla17], buvo aptariama keletas automatinio testavimo įrankių. Seminaro vedantysis išvadose paminėjo, jog Selenium įrankis yra ganėtinai geras ir mokėti didelius pinigus už komercinę įrangą turi mažai prasmės. Agile metodikoje, kurioje automatizavimo mažiau, šis įrankis tiktų, tačiau DevOps metodikoje, kurioje siekiama automatizuoti visą testavimą, reikėtų pasirinkti įrankį, kuriuo testus būtų galima rašyti greitai bei patogiai ir kuris būtų patikimesnis (neatsirastų neaiškių klaidų, kad kaip naudojantis Selenium). Iš darbe aprašytų įrankių, labiausiai tiktų CODED UI – greičiau randami objektai bei mažesnė tikimybė, kad testai sugrius atsiradus didesniems pakitimams, kadangi šis įrankis, priešingai negu Selenium, nenaudoja sudėtingų xPath išraiškų bei CssSelector – pasikeitus stiliui, sugriūtų visi testai; TestComplete įrašinėjimas netiktų dėl sunkiai palaikomo, pernaudojamo bei struktūrizuojamo kodo.

6. **Kokias platformas naudojant, kuriama aplikacija?**

   Patogu, jei įrankis sukurtas tai pačiai platformai, kaip ir kuriama pati aplikacija – tai palengvina bendravimą tarp testuotojų ir programuotojų, kurie, pagal DevOps metodiką, turėtų glaudžiai bendradarbiauti[Des17]. Be to, naudojant tą pačią

programavimo kalbą, testuotojai gali sulaukti pagalbos iš programuotojų ir taip sutrumpinti mokymosi laikotarpį. Jei aplikacija parašyta C# .NET aplinkoje ir naudojama Microsoft TFS (Team Foundation Server), kaip testavimo įrankį, patartina naudoti CODED UI. Jei buvo naudojamos kalbos kaip Java, Perl, PHP, Python, Ruby ar Scala, patogiau būtų naudotis Selenium įrankiu [Des17]. TestComplete įrašinėjimui tai reikšmės neturi, tačiau rašant kodą, šio įrankio irgi nereikėtų atmesti, jei naudojama BScript, JScript, DelphiScript, C++Script ar C#Script.

7. **Kaip pateikiamas testų rezultatas?**

   Paleidus testus, jie praeina, arba randama klaidų. Jei testai praėjo, daug informacijos nereikia, pakanka jų vykdymo trukmės. Jei atsirado klaidų, naudinga gauti informaciją apie šią klaidą, bent jau kokiame teste ji atsirado, to momento nuotrauka. Taip pat naudinga, jei šią informaciją būtų galima eksportuoti į kitus formatus ir pasidalinti su suinteresuotais žmonėmis. Patogiausiai pateikiami rezultatai TestComplete įrankyje [Sma172]. Po testų atlikimo, rezultatus galima matyti iš karto sistemoje, arba eksportuoti į HTML, XML ar MHT formatus. Testuotojas netgi gali palikti komentarus, pridėti failus. Taipogi padaromos momentinės nuotraukos. CODED UI irgi padaro nuotraukas, tačiau jas, ir kita informaciją galima rasti aplanke „Test Results“. Selenium WebDriver neturi šios galimybės, tačiau galima naudoti papildinius, kaip TestNG ir Junit, kurie turi šį funkcionalumą. Agile bei DevOps metodikose itin svarbu greiti gauti konkrečias išvadas apie paleistus testus, kad kuo anksčiau būtų galima užrašyta klaidas, jas taisyti ir taip spėti atlikti visą reikiamą testavimą per sprintą/iteraciją. [Lin14]

8. **Ar įrankis yra palaikomas?**

   Šiuo atžvilgiu pirmauja mokami įrankiai – CODED UI bei TestComplete. Jie yra atnaujinami, taisomi rasti defektai. Esant klausimams, galima susisiekti su palaikymo komanda. Selenium įrankio atveju, daugumą problemų reikia spręsti patiems, rasti įvairius apėjimus. Tačiau, atlikus praktinį darbą, pastebėta, kad tiek CODED UI tiek TestComplete įrankiuose rasta nemažai defektų, kurie yra jau keletą metų žinoma, tačiau vis dar neištaisyti. Dėl šių priežasčių, darbe atlikta praktinė dalis yra geras būdas, padėsiantis išsirinkti patogiausią bei kokybiškiausią produktą. Atsivertus šių įrankių puslapius bei perskaičius pagrindinę informaciją apie juos, galima susidaryti apgaulingą įvaizdį, kad įrankiai be trūkumų [Sma173, Msd15]. Norint neapsigauti, darbo autorius

pataria, išsirinkus keletą įrankių, atitinkančių techninius reikalavimus, atsisiųsti laikina versiją ir išbandyti įrankį praktiškai, o ne iš karto pirkti licenciją.

9. **Ar pakankamas kiekis dokumentacijos?**

   Kuo daugiau kokybiškos dokumentacijos bei problemų analizavimo forumuose, tuo daugiau laiko sutaupoma - lengviau išmokstama naudotis įrankiu bei greičiau surandami problemų sprendimo būdai. Laikas yra tas resursas, kurio trūksta kiekvienoje metodikoje, todėl tai itin svarbus aspektas. Selenium yra vienas seniausių automatizavimo įrankių, todėl internete gausų dokumentacijos bei mokymų. Atliekant praktinį darbą, problemų sprendimo būdai buvo randami greitai – forumuose daugybė informacijos, problemos pilnai išanalizuotos. CODED UI įrankis yra kiek naujesnis, dokumentacijos galima rasti nemažai, tačiau forumuose aptartų problemų yra mažoka. Rašant kodą, buvo susidurta su problemoms, kurių sprendimui sugaišta po 3-4valandas. TestComplete yra lyderis, kalbant apie dokumentaciją bei mokymus, tačiau mažiausiai informacijos galima rasti forumuose.

10. **Ar įrankis turi reikiamas technines savybes?**

    Reikia atsižvelgti, ar įrankis palaiko visas reikiamas naršykles. Iš darbe analizuotų įrankių, visi palaiko populiariausias naršykles Firefox, IE, Safari, Opera, Chrome, Edge, išskyrus CODED UI – šis nepalaiko Safari naršyklės. Taip pat verta apmąstyti, ar ateityje neplanuojama testuoti ne tik internetinius puslapius, bet ir mobilias aplikacijas, Windows formas ir kita, tuomet pradėjus testuoti kitas platformas, nereiks mokytis naujo įrankio. Selenium įrankis šiuo atžvilgiu prasčiausias – juo galima testuoti tik internetinius puslapius. TestComplete bei CODED UI turi didesnes galimybes. Jei planuojama naudoti įrašinėjimą, reikėtų išsiaiškinti, kokioms naršyklėms pritaikytas įrašinėjimo funkcionalumas. Darbe susidurta su problema, jog su Selenium galima įrašinėti tik ant FireFox naršyklės, kurios testuojamas internetinis puslapis nepalaiko.

REZULTATAI IR IŠVADOS

Šiame darbe atlikta teorinė bei praktinė analizė apie testavimą programų sistemų kūrimo gyvavimo cikle, remiantis Krioklio, Agile bei DevOps metodikomis. Sprendžiant iškeltus uždavinius gauti šie rezultatai:

1. Pirmajame skyriuje „Testavimo fazės vieta skirtingose metodikose“ apžvelgta, kaip skiriasi testavimo fazė skirtingose metodikose. Krioklyje testavimas pradedamas tik po programavimo fazės, kas yra rizikinga, nes kuo vėliau rasti defektai, tuo didesnė jų taisymo kaina. Agile bei DevOps metodikose, siekiama, jog testavimas būtų įtraukiamas nuo pradinių projekto fazių, tokiu būdu greičiau surandant bei ištaisant defektus. Tačiau negalima teigti, jog Krioklio metodiką reikėtų atmesti, o rinktis pastarąsias. Krioklys puikiai tinka, kai projektas mažas, su aiškiais bei nekintančiais projekto reikalavimais. Agile metodika tinka dideliems, su dažnai kintančiais reikalavimais, projektams. Jei įmonė turi pakankamai techninių žinių turinčių testuotojų ar resursų, leidžiančių darbuotojus apsimokyti, pasisamdyti naujų, DevOps metodika stipriai pagreitintų testavimo procesą.
2. Antrajame skyriuje „Automatinio testavimo nauda“ Išnagrinėta automatinio testavimo svarba, jo teikiama nauda bei trūkumai. Nors žmogaus išmonės bei patirties testuojant rankiniu būdų nepakeis jokie testavimo automatizavimo įrankiai, tačiau jie gali padėti spėti atlikti visą testavimą per vieną Agile metodikos sprintą bei įgyvendinti vieną iš DevOps tikslų – artėti prie 100% testų automatizavimo. Automatinis testavimas reikalauja daug pradinių investicijų – testavimo įrankiai brangūs, jų kokybė ganėtinai prasta, reikia samdyti gerus specialistus arba apmokyti esamus testuotojus. Nors tai nemaži automatizavimo trūkumai, tačiau, pagal 2016m. atliktą apklausą [Est16], 81% apklaustųjų teigė, jog pamatė automatinių testų atsipirkimą per vienerius metus.
3. Praktinėje dalyje buvo atliktas internetinės aplikacijos https://connect.bentley.com dalies vartotojo sąsajos automatinis testavimas, naudojant skirtingus automatizavimo įrankius – Selenium, CODED UI bei TestComplete. Darbe pateikta šių įrankių pagrindinė informacija, praktiniai aspektai bei išryškintos jų silpnosios bei stipriosios pusės.
4. Trečiajame skyriuje „Automatinio testavimo įrankiai“ naudoti automatizavimo įrankiai palyginti pagal techninius bei praktinius aspektus (1, 2 priedai). Pagal technines galimybes, Selenium įrankis (22 balai) pirmauja pagal daugumą vertinimo kriterijų, išskyrus palaikomas platformas - šis įrankis apsiriboja tik internetinių puslapių testavimu. Antrąją vietą užima TestComplete (19 balų), tačiau kaina stipriai sumažina jo pranašumą. Paskutinė vieta tenka

CODED UI įrankiui (17 balų), kurio didžiausias apribojimas – mažai palaikomų programavimo kalbų. Vertinant praktinę dalį visi įrankiai ganėtinai panašūs. Pirmoje vietoje TestComplete (19 balų), jo silpnosios vietos - vykdymo greitis bei kokybė. Antrąją vietą dalinasi Selenium su CODED UI (18 balų). Selenium silpnosios vietos yra testų įrašinėjimas, klaidų registravimas bei reikalingos nemažos techninės žinios. CODED UI minusai - lėtas testų vykdymas bei prastas tetų įrašinėjimas.

5. Ketvirtame skyriuje „Rekomendacijos renkantis automatizavimo įrankį" praktinės dalies atlikimo metu išanalizavus testavimo fazę bei automatinio testavimo svarbą, buvo pateiktos rekomendacijos, renkantis automatizavimo įrankį. Negalima teigti, kad vienas įrankis yra blogas, o kitas geras – skirtingi įrankiai naudojami, atsižvelgiant į projekto aplinkybes. Pradžioje itin svarbu pasitikrinti, ar aplikacijos testavimas apskritai gali būti automatizuotas. Nustatyti sumą, skirtą automatizavimo įrankių įsigijimui, esamų darbuotojų apmokymui arba kvalifikuotų specialistų samdymui. Toliau reikėtų remtis įrankių techninėmis savybėmis – palaikomomis platformomis/naršyklėmis, programavimo kalbomis. Galiausiai reikia atkreipti dėmesį į įrankio palaikymą, dokumentacijos be mokymų kiekį.

Pasinaudojus darbe pateikia testavimo fazės analize galima pagal savo poreikius išsirinkti tinkamiausią darbo metodiką ir išgauti didžiausią naudą testuojant produktą. Atsižvelgiant į rankinio bei automatinio testavimo palyginimą bei praktinę darbo dalį, pagal pateiktas autoriaus rekomendacijas galima išsirinkti testų automatizavimo įrankį, išvengiant didesnių nuostolių, pasirinkus netinkamą.

ŠALTINIAI

[Ale13] Aleksey Savkin, Limitation of Automation Software, 2013, [žiūrėta 2017-03-25]. Prieiga per internetą: <https://dzone.com/articles/limitations-automation>

[Agi01] Agile Manifesto dokumentas, 2001, žiūrėta 2017-03-25]. Prieiga per internet: <http://agilemanifesto.org/>

[Ama16] A. Amaradri Continuous Integration, Deployment and Testing in DevOps, 2016 žiūrėta 2017-03-31]. Prieiga per internetą: <http://www.diva-portal.org/smash/get/diva2:1044691/FULLTEXT02.pdf>

[Amb12] Scott W. Ambler, Results from the November 2012 Agile Testing Survey, 2012 žiūrėta 2017-03-25]. Prieiga per internetą: <http://www.ambysoft.com/surveys/agileTesting201211.html>

[Amr15] Chintan Amrit, DevOps Literature Review, 2015 [žiūrėta 2017-04-11]. Prieiga per internetą: <https://www.researchgate.net/profile/Chintan_Amrit/publication/267330992_Report_DevOps_Literature_Review/links/544ba33f0cf2bcc9b1d6bd8a.pdf>

[Bas14]Sukanta Basak, Software Testing Process Model from Requirement Analysis to Maintenance, 2014, [žiūrėta 2017-04-22]. Prieiga per internetą: <http://search.proquest.com/openview/9f73acdfba3254d2763d47592c5953f4/1?pq-origsite=gscholar&cbl=136216>

[Bla17] Rex Black, Fully Leveraging Agile Test Automation, 2017, [žiūrėta 2017-03-12]. Prieiga per internetą: <http://www.pduotd.com/2017/03/22/fully-leveraging-agile-test-automation/>

[Boe02] Boehm, Get ready for agile methods, with care. Computer, 2002

[BT04] Boehm, B., & Turner, R., Balancing agility and discipline: Evaluating and integrating agile and plan-driven methods. In Software Engineering, 2004

[Dem88] W. E. Deming, "Out of the crisis: quality, productivity and competitive position". Cambridge University Press, 1988.

[Des17] Aniket Deshpande, “DevOps” an Extension of Agile Methodology – How It will Impact QA?, 2017 [žiūrėta 2017-04-03]. Prieiga per internetą: <http://www.softwaretestinghelp.com/devops-and-software-testing/>

[Est16] European Software Testing Benchmark Report 2016: Test Management, 2016

[Gur15] Gur99, Agile Testing Guide: Process, Strategies, Test Plan, Quadrants, Life Cycle, 2015, [žiūrėta 2017-04-04]. Prieiga per internetą: < http://www.guru99.com/agile-testing-a-beginner-s-guide.html>

[Gur162] Guru99, Coded UI Test (CUIT) , 2016, [žiūrėta 2017-04-02]. Prieiga per internetą: <http://www.guru99.com/coded-ui-test-cuit.html>

[Gur163]Guru99, Banking Domain Application Testing, 2016, [žiūrėta 2017-04-05]. Prieiga per internetą:
< http://www.guru99.com/banking-application-testing.html >

[HM12] Httermann, Michael, DevOps for Developers, 2012

[Hig12] Highsmith, Agile software development ecosystems, 2012

[Iso99] International Organization for Standardization, "ISO/IEC 9001: Quality management systems -- Requirements," 1999.

[Ist15] ISTQB, What is Waterfall model- advantages, disadvantages and when to use it?, 2015, [žiūrėta 2017-04-08]. Prieiga per internetą: <http://istqbexamcertification.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/>

[Isa15] Malcolm Isaacs, 4 simple steps to prioritizing your DevOps automation, 2015, [žiūrėta 2017-04-08]. Prieiga per internetą: < https://techbeacon.com/devops-automation-4-simple-steps-prioritizing>

[KG15] Harpreet Kaur, Dr.Gagan Gupta , Comparative Study of Automated Testing Tools: Selenium, Quick Test Professional and Testcomplete, 2015, [žiūrėta 2017-03-25]. Prieiga per internetą:
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.448.6743&rep=rep1&type=pdf>

[Kev12] Kevin Martin, Automated Testing - The Advantages and Disadvantages , 2012

[LCR+13] Maurizio Leotta, Diego Clerissi, Filippo Ricca, Paolo Tonella, Capture-Replay vs. Programmable Web Testing: An Empirical Assessment during Test Case Evolution, 2013, [žiūrėta 2017-04-02]. Prieiga per internetą:
<http://sepl.dibris.unige.it/publications/2013-leotta-WCRE.pdf>

[Lar14] Larson,I still don't have time to manage requirements: My project is later than ever, 2014

[Lew08] William E. Lewis, Software Testing and Continuous Quality Improvement, Third Edition, 2008.

[Lin14] Tilo Linz “TESTING IN SCRUM” A Guide for Software Quality Assurance in the Agile World, 2014.

[MS13] M.Mahalakshmi, DR. M. Sundararajan, Traditional SDLC Vs Scrum Methodology – A Comparative Study, 2013

[Msd15] Msdn dokumentacija, 2015, [žiūrėta 2017-03-28]. Prieiga per internetą:
< https://msdn.microsoft.com/en-us/library/dd286726.aspx
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.413.2992&rep=rep1&type=pdf>

[MKM15] Lakshman Mahadevan, William J. Kettinger, Thomas O. Meservy, Running on Hybrid: Control Changes when Introducing an Agile Methodology in a Traditional "Waterfall" System Development Environment, 2015, [žiūrėta 2017-04-01]. Prieiga per internetą:
<http://aisel.aisnet.org/cgi/viewcontent.cgi?article=3832&context=cais>

[Mun14] JÜRGEN MÜNCH, Data- and Value-Driven Software Engineering with Deep Customer Insight, 2014 , [žiūrėta 2017-04-10]. Prieiga per internetą:
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.448.6743&rep=rep1&type=pdf>

[MV14] Marcel de Vries, Test Automation with CodedUI, 2014. Prieiga per internetą:
< https://joecolantonio.com/testtalks/27-marcel-de-vries-test-automation-with-codedui-testtalks/>

[Olu14] Dele Oluwole, "The Tester's Role in Agile Is More than "Just Testing", 2014, [žiūrėta 2017-04-10]. Prieiga per internetą:
<https://www.scrumalliance.org/community/articles/2014/june/the-tester%E2%80%99s-role-in-agile-is-more-than-just-testi >

[Peu14] T. Peuraniemi. Devops, value-driven principles, methodologies and tools. Data-and Value-Driven Software Engineering with Deep Customer Insight, psl. 43, 2014

[RM15] Priyanka Rathi Vipul Mehra, Analysis of Automation and Manual Testing Using Software Testing Tool, 2015

[Sel17] Selenium dokumentacija, 2017, [žiūrėta 2017-03-31]. Prieiga per internetą:
<http://www.seleniumhq.org/docs/06_test_design_considerations.jsp#page-object-design-pattern>

[Sha14] R. M. Sharma, Quantitative Analysis of Automation and Manual Testing, 2014
[žiūrėta 2017-03-29]. Prieiga per internetą:
< http://www.ijeit.com/Vol%204/Issue%201/IJEIT1412201407_46.pdf>
http://www.academicscience.co.in/admin/resources/project/paper/f201503151426402494.pdf

[She31] W. A. Shewhart, Economic control of quality of manufactured product. Van Nostrand, 1931.

[Sil16] Jason Silberman, The Advantages of Manual VS. Automated Testing, 2016

[SMS+14] Jagannatha S, Niranjanamurthy M, Manushree SP, Chaitra GS**,** Comparative Study on Automation Testing, 2014,

[žiūrėta 2017-03-22]. Prieiga per internetą:

<http://s3.amazonaws.com/academia.edu.documents/35162080/V3I10201485.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1491922332&Signature=YyZMPRFOgH2Ay7Z3W0fqJ6prtqo%3D&response-content-disposition=inline%3B%20filename%3DComparative_Study_on_Automation_Testing.pdf>

[Sma17] SmartBear dokumentacija, 2017, [žiūrėta 2017-04-20]. Prieiga per internetą: <https://support.smartbear.com/testcomplete>

[SNP15] J. Smeds, K. Nybom, and I. Porres. Agile Processes in Software Engineering and Extreme Programming: 16th International Conference, XP 2015,

[Sma172] SmartBear dokumentacija, 2017, [žiūrėta 2017-03-05]. Prieiga per internetą:

<https://smartbear.com/product/testcomplete/features/test-reporting/>

[Sma173] SmartBear dokumentacija, 2017, [žiūrėta 2017-04-20]. Prieiga per internetą: <https://smartbear.com/product/testcomplete/overview/ >

[Spa16] Penna Sparrow, Manual Testing: Advantages and Disadvantages of Manual Software Testing, 2016

[STC15] Deepti Singh1 , Ankit Thakur2 , Abhishek Chaudhary3, A Comparative Study between Waterfall and Incremental Software Development Life Cycle Model, 2015, [žiūrėta 2017-04-15]. Prieiga per internetą: < http://ijetst.in/article/v2-i4/9%20ijetst.pdf >

[VBP14] Jyoti Verma, Sunita Bansal, Himanshu Pandeym, International Journal of Scientific & Engineering Research, Volume 5, Issue 4, 201, [žiūrėta 2017-04-02]. Prieiga per internetą:

<http://www.ijser.org/researchpaper/Develop-Framework-for-Selecting-Best-Software-Development.pdf >

[Vri15] Marcel de Vries, Test Automation with CODED UI, 201, [žiūrėta 2017-03-22]. Prieiga per internetą:

<https://app.pluralsight.com/player?course=codedui-test-automation&author=marcel-devries&name=codedui-m1-courseintro&clip=0&mode=live >

[Zen14] ZenQ ,Test Automation Process, 2014, [žiūrėta 2017-04-11]. Prieiga per internetą:

<http://zenq.com/Portals/0/TestAutomationWhitePaper.pdf >

[Wri14] Christopher Wright, Agile Governance and Audit: An overview for auditors and agile teams, 2014, [žiūrėta 2017-04-02]. Prieiga per internetą:
< **http://www.jstor.org/stable/j.ctt7zsx7z**>

PRIEDAI

1 priedas. CODED UI, Selenium, TestComplete įrankų techninių savybių palyginimas

	Palaikomos naršyklės	**Testuojamos platformos**	**Programavimo kalba**	**Programavimo aplinka**	**Kaina**
CODED UI	Firefox, IE, Opera, Chrome, Edge	Internetiniai puslapiai, Windows formos, kai kurios mobiliosios Windows aplikacijos	VBScript, C#	Įeina į Visual Studio Ultimate/Premium/ Enterprise paketus – nuo 2010m.	Visual Studio aplinkos kaina, priklauso nuo paketo ir licencijos, prasideda nuo 500e
Selenium	Firefox, IE, Safari, Opera, Chrome, Edge	Internetiniai puslapiai	C#, Groovy, Java, Perl, PHP, Python, Ruby, Scala	Pasirinktinai pagal programavimo kalbą	Nemokamas
TestComplete	Firefox, IE, Safari, Opera, Chrome, Edge	Internetiniai puslapiai, Windows, Android OS, iOS	VBScript, JScript, C++Script, C#Script, DelphiScript	Kartu su įrankiu, gausus funkcionalumas	Priklauso nuo pasirinkto paketo ir licencijos. Kainos nuo 1000e

2 priedas. CODED UI, Selenium, TestComplete įrankų palyginimas iš praktinės pusės

	Įrašinėjimas	**Klaidų registravimas**	**Dokumentacija**	**Vykdymas**	**Kokybė**	**Techninės žinios**
Selenium WebDriver	FireFox	Reikia atsisiųsti papildinį (log4j)	Plati dokumentacija, daugelis problemų išspręsta forumuose	Testų rašymas užtrunka (~7d.), testų vykdymas greitas (~7min)	Nėra itin stabilus dėl atsitiktinių defektų	Objektinio programavimo pagrindai, kitos techninės žinios
CODED UI	IE	Išsamiai eksportuoja į "TestResults" failą	Plati dokumentacija, tačiau trūksta problemų sprendimo	Kodo rašymo greitis vidutinis (~5d), testų vykdymas vidutinis (~9min)	Yra defektų, tačiau juos galima apeiti	Objektinio programavimo pagrindai
Test Complete	Ant IE veikia gerai, ant kitų naršyklių blogai fokusuoja obejektus	Išsami informacija matoma pačioje įrankio aplinkoje	Labai plati dokumentacija su video mokymais, tačiau problemų sprendimo trūksta	Testų rašymas greitas(~3d), testų vykdymas vidutinis(8min)	Nėra itin stabilus dėl atsitiktinių defektų. IDE veikimas lėtas	Nebūtinos