TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Kairit Sims 154767 IABB

KASUTAJALIIDESE AUTOMAATTESTIMISE PROTSESSI JA VAHENDI VALIKU VÄLJATÖÖTAMINE OÜ NET GROUP NÄITEL

Bakalaureusetöö

Juhendaja: Inna Švartsman
MSc
Lektor

Tallinn 2018

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Kairit Sims

20.05.2018

Annotatsioon

Käesolev bakalaureusetöö lahendab aktuaalset probleemi tänases tarkvaraarenduse protsessis OÜ Net Group'is, kus kogu testija vastutusalas olev kasutajaliidese testimine käib manuaalselt ning seetõttu ei ole tagatud piisav tarkvara kvaliteet. Üks võimalustest parandada tarkvara kvaliteeti on kasutada kasutajaliidese testimisel automaatteste.

Sellest tulenevalt on käesoleva töö eesmärk välja selgitada tõhusaim kasutajaliidese automaattestimise protsess ning sobivad töövahendid. Uut protsessi ja vahendit rakendati koheselt OÜ Net Group Beeta tiimi töös. Selle tulemusena vähenes manuaalsele testimisele kuluv aeg, suurenes kasutajaliidese testidega kaetus ning seeläbi tõusis ka tarkvara kvaliteet.

Eelnevalt nimetatud Beeta tiim jäi käesoleva töö tulemustega rahule ning autor plaanib kasutuselevõetud protsessi ja vahendit tutvustada ka teistele OÜ Net Group tarkvaraarendustiimidele. Bakalaureusetöö on lisaks OÜ Net Group'ile kasulik ka teistele tarkvaraarendusettevõtetele, kes soovivad oma töös kasutajaliidese testimise protsessi parandada ning teste automatiseerida.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 25 leheküljel, 6 peatükki, 19 joonist, 5 tabelit.

Abstract

Development of User Interface Automated Testing Process and Tool Selection on the Example of OÜ Net Group

The thesis solves an actual problem in OÜ Net Group software development process - the user interface is currently tested manually and this causes problems with software quality. One possible way to solve the problem is to automate user interface tests.

Purpose of the thesis is to find the most effective user interface testing process and tool for test automation. During the study new process and tool were taken into use at OÜ Net Group Beeta team. As a result, the time spent on manual user interface testing was considerably reduced. In addition, test coverage and software quality were increased due to the automated user interface tests.

The new process starts with getting to know project and its requirements, followed by writing the test plan. Next step is to write test cases for current iteration and automate them. During the study two different tools were analyzed and tried out. These tools were Selenium WebDriver and Cypress. To find the most suitable tool for OÜ Net Group analytical hierarchy process technique was used. As a result of evaluation Cypress was chosen. After writing the automated user interface tests they can run continuously and there is no need to test user interface manually on a large scale. Manual user interface testing is needed only for making sure user interface is in accordance with design guide and is comfortable to use.

Beeta team in OÜ Net Group is satisfied with the result of the study and author will continue to introduce the new process and tool to other OÜ Net Group teams. Outside OÜ Net Group the thesis is valuable to other enterprises who wish to improve their testing process and automate user interface tests.

The thesis is in Estonian and contains 25 pages of text, 6 chapters, 19 figures, 5 tables.

Lühendite ja mõistete sõnastik

AHP	*Analytic Hierarchy Process*, analüütiline hierarhia protsess
ID	*Identificator*, identifikaator
IT	Infotehnoloogia
TTÜ	Tallinna Tehnikaülikool
UI	*User interface*, kasutajaliides
URL	*Universal Resource Locator*, universaalne ressursi lokaator ehk internetiaadress

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tarkvaraarenduse protsessi lahutamatuks osaks on testimine, mis tagab tarkvara kvaliteedi. Kiiresti areneva agiilse tarkvaraarenduse tulemit tuleb testida pidevalt ning mitmel tasemel. Üks testitavatest tasemetest on kasutajaliidese testimine, mille eest vastutab käesoleva bakalaureusetöö autor töötades OÜ Net Group'is testijana. OÜ Net Group'i näol on tegu tehnoloogiaettevõttega, kus luuakse klientidele tarkvaralahendusi.

Bakalaureusetööd alustades testib autor projekte terves ulatuses manuaalselt. Manuaalne kasutajaliidese testimine on ajamahukas ning ei taga piisavat kvaliteeti. Kvaliteediprobleemi on välja toonud nii arendustiimi liikmed, kui ka klient. Käesolev bakalaureusetöö lahendab aktuaalset probleemi tänases tarkvaraarenduse protsessis OÜ Net Group'is, kus kogu testija töö käib manuaalselt ja seetõttu ei ole tagatud piisav tarkvara kvaliteet.

Üks võimalustest parandada tarkvara kvaliteeti on kasutada kasutajaliidese testimisel automaatteste. Sellest tulenevalt on käesoleva töö eesmärgiks välja selgitada tõhusaim kasutajaliidese automaattestide loomise protsess ning sobivad töövahendid. Bakalaureusetöö käigus võetakse OÜ Net Group Beeta tiimis kasutusele uus kasutajaliidese testimise protsess, mis hõlmab automaattestimist valitud vahendiga. Töö tulemusena peaks vähenema kasutajaliidese manuaalsele testimisele kuluv aeg, kasutajaliidese testidega kaetus oleks suurem ning seeläbi oleks kõrgem ka tarkvara kvaliteet.

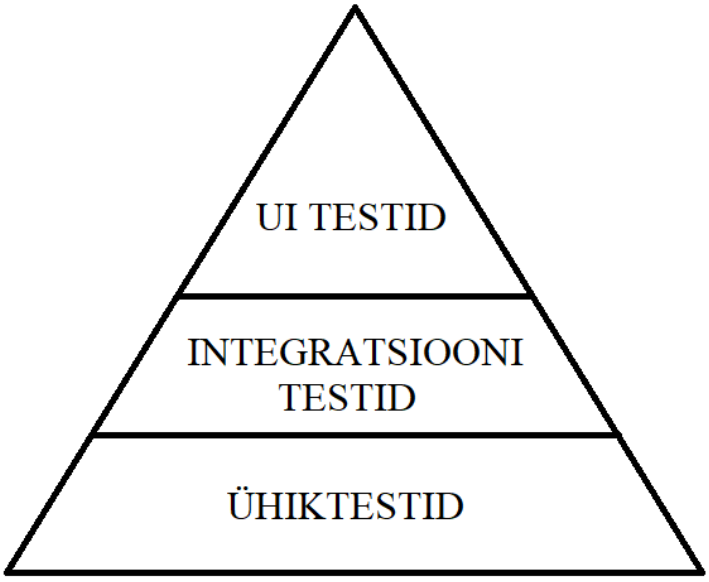
Bakalaureusetöö algab tausta ja probleemi põhjalikuma tutvustamisega, kus selgitatakse miks praegune olukord rahulolematust tekitab. Järgnevalt kirjeldatakse ning analüüsitakse kasutajaliidese automaattestimise protsessi ning automaattestide loomiseks mõeldud vahendeid. Läbi analüütilise hierarhia protsessi selgitatakse välja OÜ Net Group'i jaoks sobivaim töövahend. Seejärel rakendatakse kasutajaliidese automaattestimise protsessi ning valitud vahendit OÜ Net Group tarkvaraarendustiimi töös. Viimaseks kirjeldatakse kuidas uus protsess töö tulemuslikkust ning tarkvaralahenduse kvaliteeti mõjutab.

2 Taust ja probleem

Käesoleva bakalaureusetöö ülesanne on kasutajaliidese automaattestimise protsessi ja vahendite valiku välja töötamine OÜ Net Group jaoks. Teema valik tuleneb aktuaalsest probleemist, mida autor Net Group'i Beeta tiimis testijana töötades täheldanud on.

OÜ Net Group on tarkvaraarendusega tegelev tehnoloogiaettevõte, kus töötatakse väikestes tarkvaraarendustiimides. Iga tiimi juurde kuulub ka testija, kelle ülesandeks on kontrollida tarkvara nõuetele vastavust ehk leida tuleb vead nii tarkvaralahenduse funktsionaalsuses kui kasutajaliidese disainis. Seega on testijal väga oluline roll arendatava tarkvara kvaliteedi tagamises.

Tarkvara testimine on väga lai valdkond ning kvaliteetse tarkvara tagamise nimel tuleb luua erinevat tüüpi teste. Et anda ülevaade erinevatest testimise kihtidest on joonisel 1 kujutatud Mike Cohni poolt välja töötatud testimise püramiidi. Cohni püramiid jagab testimise kihtideks ning näitab kui suur osakaal erinevatel kihtidel olema peaks [1].



Joonis 1. Testimise püramiid.

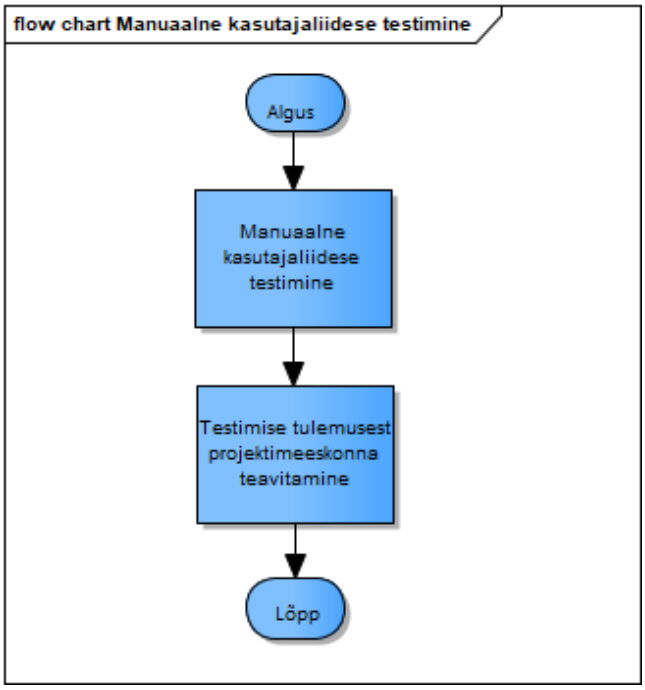
Esimesena tuleks luua ühikteste, mis on joonisel 1 kujutatud kõige madalama kihina. Ühiktestimisega kontrollitakse kas tarkvara ühik töötab nõuetele vastavalt. Ühik on väikseim testitav osa tarkvarast. Seega on ühiktestid kitsa haardega, kuid peaksid katma võimalikult suure osa koodist. Nad on üksteisest sõltumatud ning neid saab tihti ja kiiresti jooksutada [1].

Järgmisena kihina tuleb luua integratsioonitestid, mis võimaldavad testida rakenduse ühilduvust väliste liidestega ja eraldiseisvate komponentidega (näiteks andmebaas). Integratsioonitestid on ühiktestidest keerukamad ning ka nende jooksutamine nõuab rohkem aega [1].

Viimaseks kihiks on kasutajaliidese testimine ehk UI (*User interface*) testimine, mille abil saab hinnata kas rakenduse kasutajaliides töötab korrektselt. Kasutaja poolt tulev sisend (näiteks hiireklikk või välja täitmine) peab välja kutsuma nõuetele vastavad sündmused. Kokkuvõttes peavad kasutajani jõudma õiged andmed ning kasutajaliidese vaade peab muutuma ootuspäraselt [1].

OÜ Net Group’is on ühiktestide ja integratsioonitestide kirjutamine arendajate ülesanne. Testija ülesandeks on testida kasutajaliidest ning teostada üldist nõuetele vastavuse kontrolli. Lisaks toetab testija klienti faasis, kus ka klient ise uut arendust testida saab.

Käesolev lõputöö keskendub testija ülesannetest lähtuvalt kasutajaliidese testimisele. Algselt testitakse kogu kasutajaliidest manuaalselt. Manuaalse testimise protsess on kujutatud järgneval joonisel 2.



Joonis 2. Manuaalne kasutajaliidese testimise protsess.

Manuaalne testimine hõlmab endas kõikide kasutajaliidese funktsionaalsuste läbiproovimist. Testide tulemusest teavitatakse projekti meeskonda, et arendajad saaksid tekkinud vead parandada. Manuaalset testimist peab pidevalt kordama ning kuna see võtab kaua aega on testija töökoormus suur. Testide ühekordsuse tõttu peab testija pidevalt samu funktsionaalsusi üle testima, et veenduda süsteemi korrektses toimimises. Tihti samu kasutusjuhte korrates harjub testija kasutajaliidesega ning võib juhtuda, et osad vead jäävad märkamata. Sama võib juhtuda inimlikust eksimusest, tähelepanematusest või ka suurest töökoormusest. See aga toob kaasa madalama kvaliteediga tarkvaralahenduse.

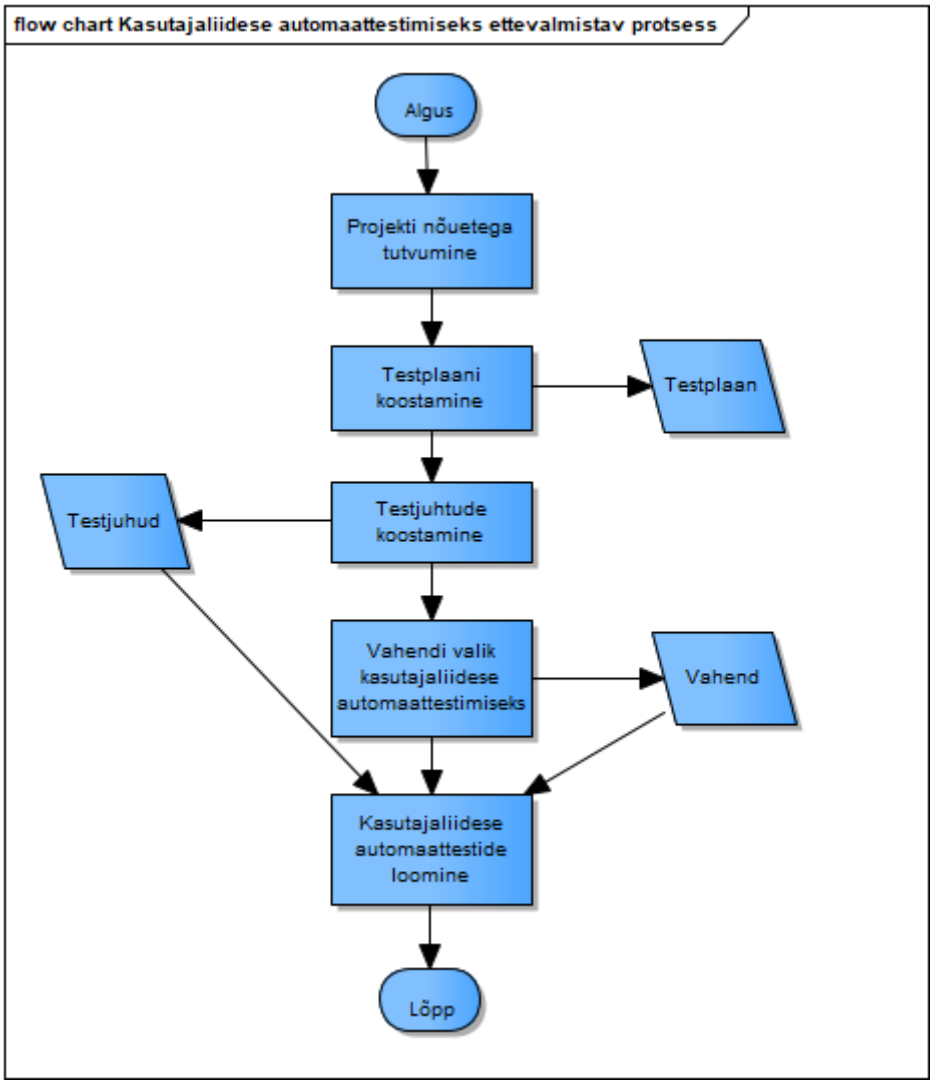
Kirjeldatud probleemi lahendamiseks ning OÜ Net Group'is kõrgema kvaliteediga tarkvaralahenduste tagamiseks keskendub käesolev bakalaureusetöö kasutajaliidese automaattestimise protsessi ja vahendite valiku välja töötamisele. Järgnevas peatükis analüüsitav protsess ning vahendid võetakse esmalt kasutusele OÜ Net Group Beeta tiimi töös.

3 Kasutajaliidese automaattestimise protsess

Kasutajaliidest automaattestides jooksutab testjuhte arvuti. Kasutajaliidese automaatteste on küll keerukas luua, kuid neid on võimalik korduvalt käivitada ja kokkuvõttes on ajakulu väiksem, kui manuaalselt testides [2]. Kui automatiseerida lisaks eelpool kirjeldatud ühiktestidele ja integratsioonitestidele ka kasutajaliidese testid on tulemuseks kvaliteetsem tarkvara, mille testidega katvus on kõrge. Manuaalsete testide elimineerimisega saavutatav väiksem ajakulu ja kõrgema kvaliteediga tarkvara tähendavad omakorda madalamaid kulusid arendusele. Kasutajaliidese automaattestid suurendavad tarkvaraarenduse protsessi tõhusust, teste saab käivitada tihedalt ja seejuures ei ole vaja testija kohalolu [3].

Kasutajaliidese automaattestide loomine on keeruline protsess ja võib ebakorrektse lahenduse puhul tuua kaasa palju rohkem probleeme, kui manuaalne testimine [4]. Meetodeid ja lähenemisi kasutajaliidese automaattestide loomiseks on erinevaid. Näiteks võib alustada kasutajaliidese suitsutestimise automatiseerimisest [5]. Käesolev bakalaureusetöö keskendub OÜ Net Group vajadustest lähtuvalt projekti kasutuslugude põhjal kasutajaliidese testide automatiseerimisele, kuna ka tarkvaraarendus käib OÜ Net Group'is kasutuslugude põhjal.

Enne kasutajaliidese automaattestide loomist on tarkvaraarenduse kvaliteedi tagamiseks oluline läbida testimiseks ettevalmistavad sammud. Testimiseks ettevalmistamine algab nõuetega tutvumisest, kus õpitakse projekti tundma ja selgitatakse välja, kui suures mahus antud projektile automaatteste luua. Edasi koostatakse testplaan ning testjuhud. Seejärel on vaja otsustada, milliseid tööriistu kasutajaliidese automaattestide loomisel kasutada. Järgmiseks sammuks on esimeste kasutajaliidese automaattestide loomine selliselt, et automaattestid oleks võimalikult vähe sõltuvad muutustest kasutajaliideses. Sellega on testimiseks ettevalmistav osa kasutajaliidese automaattestimise protsessist lõppenud [4]. Ettevalmistav osa kasutajaliidese automaattestimise protsessist on toodud järgneval joonisel 3.



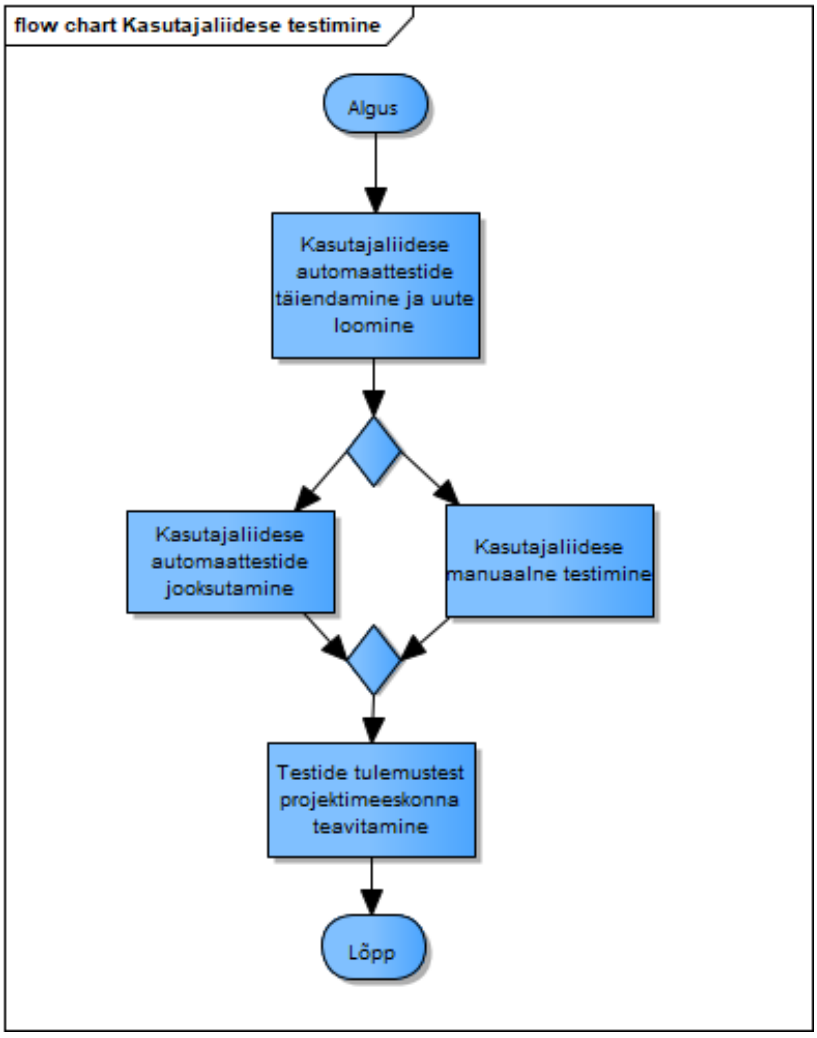
Joonis 3. Kasutajaliidese automaattestimiseks ettevalmistav protsess.

Loomulikult saab kasutajaliidese automaattestimise protsessile läheneda erinevalt, kuid peamiselt seisnevad erinevused samade sammude erinevas järjekorras või mõningate sammude kõrvale jätmises. Näiteks on mõnikord soovitatud testimisprotsessi alustada vahendi valikuga [6]. Võttes aga arvesse OÜ Net Group erinevate tiimide kogemust on peale projekti nõuetega tutvumist mõislik alustada testplaani väljatöötamisega ning alles seejärel jätkata teiste testimiseks ettevalmistavate sammudega. Testplaani koostamise olulisus tuleneb ka OÜ Net Group'i projektide mahukusest.

Peamine osa kasutajaliidese automaattestimise protsessist on ikkagi testimine. Kui ettevalmistavat osa protsessist läbitakse iga projekti või selle etapi puhul ühe korra, siis testimise osa kasutajaliidese testimise protsessist korratakse pidevalt. See tähendab

automaattestide jooksutamist ja testide tulemusest raporteerimist. Lisaks tuleb uute funktsionaalsuste lisandumisel kirjutada kasutajaliidese automaatteste juurde või täiendada olemasolevaid.

Samas tuleb arvestada, et kogu kasutajaliidese testimise protsessi ei saa automatiseerida, läbi tuleb viia ka manuaalset testimist. Ainult läbi manuaalse testimise saab öelda kas kasutajaliides on ilus, vastab disainiõuetele ja on mugav kastutada [1]. Lisades kasutajaliidese automaattestimise protsessile manuaalse testimise saame protsessi, mida on kujutatud järgneval joonisel 4.



Joonis 4. Kasutajaliidese testimise protsess.

Järgnevates alapeatükkides on käsitletud eelpool kirjeldatud testimiseks ettevalmistamise (Joonis 3) ja kasutajaliidese testimise protsessi osi põhjalikumalt.

3.1 Nõuetega tutvumine

Uue projektiga tööd alustades on esimeseks sammuks projekti ja selle nõuetega tutvumine. Kasutajaliidese testide automatiseerimise osas on projekti nõuetega tutvumise faasis hea läbi mõelda, kui suur osa testidest automatiseerida. Tuleb arvestada, et esialgne hinnang võib töö käigus muutuda kui selgub, et teatud funktsionaalsusele automaattestide kirjutamine on liigse keerukusega või ei ole muul moel otstarbekas [4].

Kasutajaliidese automaattestide loomine on ajamahukas investeering kõrgema kvaliteediga tarkvarasse. Seetõttu tuleb enne testide loomist põhjalikult analüüsida, kas automaattestide loomine on antud olukorras üldse mõistlik. Võib öelda, et lühiajalistele projektidele ei tasu kasutajaliidese automaatteste luua, sest testide loomisele kuluv aeg võib olla samas suurusjärgus projekti kestvusega ja kasutegur on väike. Samas pikema perioodiga projektidele, kus automaattestide loomise aeg on võrreldes projekti kestvusega marginaalse tähendusega, on kasulik automaatteste luua [7].

Teise aspektina tasub mõelda, kui stabiilse projektiga tegu on. Ebastabiilsele kasutajaliidesele, kus elemendid tihti muutuvad ei ole automaatteste mõtet luua. Sellisel juhul tasub oodata testide loomisega seni, kuni kasutajaliides on muutunud stabiilseks [7].

3.2 Testplaani koostamine

Testplaan on enne testimisega alustamist loodav dokument, mis kirjeldab testimise eesmärgid, skoobi ja prioriteedid. Testplaanis on kirjas ka testimiseks vajaminevad ressursid ja ajakava [3]. Testplaani võib luua eraldiseisva tekstidokumendina, kuid OÜ Net Gropis kasutusel olev versioonihaldustarkvara Team Foundation Server võimaldab testplaani koostada ka süsteemis defineeritud põhja peal.

Testplaani võib iga testija oma nägemuse ja vajaduste järgi koostada, kuid järgnevalt on parema ülevaate saamiseks loetletud võimalikud testplaani osad [3]:

- ajakava;
- töötajad, kes on testimise protsessi kaasatud;

- vajaminevad vahendid, milleks võivad olla nii arvutid, serverid, kui ka automaattestide loomiseks vajalik tarkvara;
- testkeskkonnad, näiteks operatsioonisüsteem, mobiilsed seadmed või veebilehitsejad;
- tarkvaralahenduse põhifunktsionaalsused;
- planeeritud tehnikad ja meetodid;
- testimise eesmärgid ja veaolukordades käitumine.

3.3 Testjuhtude koostamine

Testjuht on rakenduse üht funktsionaalsust kirjeldav tekst. Testjuht koosneb ID-st (*Identificator*), pealkirjast, ehk funktsionaalsusest mida testitakse, samm-sammulisest juhendist, kuidas testjuht läbida, vajaminevatest testandmetest ja oodatud tulemusest [8]. Illustreerimiseks on järgnevas tabelis 1 toodud näidis testjuht Tallinna Tehnikaülikooli (TTÜ) kodulehelt Infotehnoloogia (IT) teaduskonna kodulehele jõudmisest.

Tabel 1. Näidis testjuht.

ID	14862
Pealkiri	TTÜ kodulehelt IT teaduskonna kodulehele jõudmine
Juhend	1. Mine veebilehele 2. Vali menüüst etteantud menüüpunkt 3. Vali avanenud rippmenüüst etteantud teaduskond 4. Kontrolli, kas lehekülge aadress võrdub etteantud URL-iga (*Universal Resource Locator*)
Andmed	Veebileht – „https://www.ttu.ee“ Menüüpunkt – „Teadustöö“ Teaduskond – „Infotehnoloogia teaduskond“ URL – „https://www.ttu.ee/?id=146605“
Oodatud tulemus	Oled jõudnud IT teaduskonna kodulehele

Käesolev bakalaureusetöö keskendub OÜ Net Group vajadustest lähtuvalt kasutuslugude põhjal testjuhtude loomisele ning nende automatiseerimisele. Järgnevalt on välja toodud soovitused, milliseid testjuhte automatiseerida [6], [3]:

- äriliselt kõige olulisemad positiivsed ja negatiivsed juhud;
- juhud, mida on vaja testida korduvalt;
- ajamahukad testjuhud;
- tehniliselt keerukad juhud;
- juhud, mida on vaja testida erinevates veebilehitsejates.

3.4 Vahendid automaattestide loomiseks

Kasutajaliidese automaattestide loomiseks ja haldamiseks mõeldud vahendeid on erinevaid. Käesolev bakalaureusetöö keskendub kahele neist. Esimeseks on Selenium WebDriver, mis on üks populaarsemaid kasutajaliidese automaattestimiseks mõeldud vahendeid. Teine vaadeldav vahend on Cypress, mis on uus ja moderne vahend, ning millega saab lisaks kasutajaliidese testidele luua ka ühikteste ja integratsiooniteste. Cypress'i loojad rõhuvad just sellele, et Cypress on tööpõhimõtetelt Selenium WebDriver'i täielik vastand [9]. Just seetõttu on autor otsustanud need kaks vahendit vaatluse alla võtta. Järgnevates alapeatükkides on tutvustatud lähemalt mõlemat vahendit ning selgitatud välja sobivaim vahend.

3.4.1 Selenium WebDriver

Selenium WebDriver on kogum erinevatest automaattestide loomist toetavatest tarkvaraarenduse tööriistadest. Kasutades Selenium WebDriver'i funktsionaalsust on võimalik ära katta kõik vajadused kasutajaliidese automaattestide loomisel. Antud tööriista kõige suuremaks eeliseks teiste tööriistade ees on erinevate veebilehitsejate tugi. Toetatud veebilehitsejad on Google Chrome, Internet Explorer, Firefox, Safari, Opera, HtmlUnit ja phantomjs [10].

Vahend on tasuta kättesaadav ning sobib ka mobiilsetele seadmetele mõeldud tarkvaralahenduste testimiseks. Küll aga on vajalik programmeerimise oskus, teste saab

kirjutada järgnevates keeltes: Java, C#, Python, Ruby, Perl, PHP, JavaScript. Selenium WebDriver'il puudub kasutajaliides, seda saab kasutada endale mugavas programmeerimiskeskkonnas ning ka vigade haldamine toimub seal. Selenium WebDriver'i installimine käib samuti läbi programmeerimiskeskkonna oma projekti teeke täiendades [10]. Järgnevalt on joonisel 5 toodud tabelis 1 (Tabel 1) kirjeldatud testjuhu põhjal loodud näidistest, mis on kirjutatud Javas.

```
class Test {

	private WebDriver driver;

	@BeforeEach
	void setUp() throws Exception {
		driver = new ChromeDriver();
		System.setProperty("webdriver.chrome.driver",
	System.getProperty("user.dir") + "/chromedriver.exe");
	}

	@Test
	void test14862() {
		driver.get("https://www.ttu.ee");
		driver.findElement(By.className("mainNav")).findEle
	ment(By.xpath("//a[3]")).click();
		driver.findElement(By.linkText("Infotehnoloogia
	teaduskond")).click();
		assertEquals(driver.getCurrentUrl(),
	"https://www.ttu.ee/?id=146605");
	}
}
```

Joonis 5. Näidis testjuhu põhjal Selenium WebDriver'iga koostatud kasutajaliidese automaattest.

Enne testimist on vaja määratleda veebilehitseja, milles soovitakse testi jooksutada. Seetõttu on joonisel 5 setUp meetodis öeldud kus asub soovitava veebilehitseja draiver testija arvutis. Test ise algab TTÜ veebilehele minekuga, kus otsitakse menüüd ning avanenud rippmenüüst teaduskonda. IT teaduskonnale klikates peaks kasutaja olema jõudnud IT teaduskonna kodulehele.

3.4.2 Cypress

Cypress on modernsete veebirakenduste testimiseks mõeldud vahend, mis võimaldab kirjutada ühikteste, integratsiooniteste ja kasutajaliidese teste. Cypress on vabatarkvaraline testimisvahend, mille saavad kõik endale alla laadida. Arhitektuuri

poolest jookseb cypress veebilehitsejas ning töötab võrgukihi peal, mis võimaldab kontrollida võrgu poole tehtavaid päringuid [9].

Järgnevalt on loetletud Cypress'i funktsioone [9] :

- Cypress teeb automaatteste jooksutades kõikidest sammudest kuvatõmmiseid, et kasutaja näeks kogu teekonda ja veaolukordade tekkimist;
- Cypress oskab teha kogu testjuhu läbimisest video;
- Cypress'i veateated on detailirohked ja muudavad vigadest arusaamise lihtsaks;
- Cypress'i automaattestidesse ei ole vaja kirjutada ooteaega, mis kulub näiteks faili üles laadimiseks. Cypress saab sellest ise aru;
- Cypress võimaldab kontrollida serveri poole tehtavaid päringuid.

Cypress'i kasutamiseks tuleb see käsurealt oma projekti kaustas installida ning avada. Vastavad käsud on toodud järgnevatel joonistel 6 ja 7 [9].

```
npm install cypress --save-dev
```

Joonis 6. Cypress'i installimine.

```
./node_modules/.bin/cypress open
```

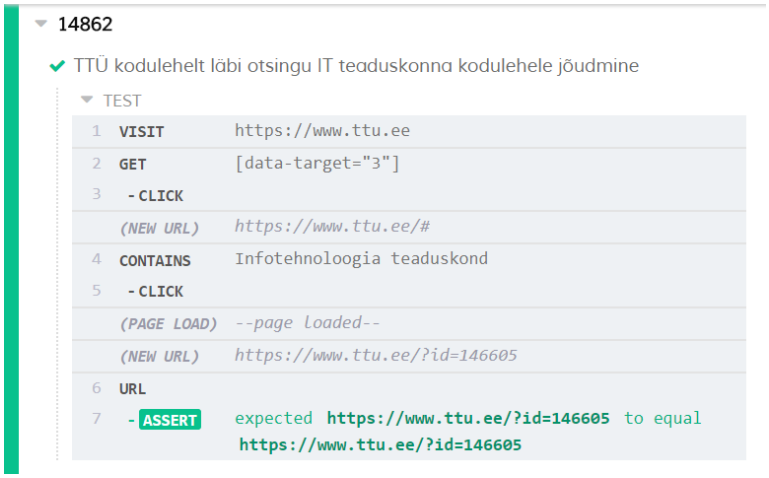
Joonis 7. Cypress'i avamine.

Järgnevalt avaneb Cypress'i kasutajaliides, mis võimaldab teste jooksutada ning tulemusi hallata. Testid kirjutatakse TypeScript keeles JavaScript failidena projektis asuvasse Cypress kausta. Tabelis 1 (Tabel 1) toodud testjuhule vastava testi kood on toodud järgneval joonisel 8.

```
describe('14862', function() {
	it('TTÜ kodulehelt läbi otsingu IT teaduskonna kodulehele
	jõudmine', function() {
		cy.visit('https://www.ttu.ee')
		cy.get('[data-target="3"]').click()
		cy.contains('Infotehnoloogia teaduskond').click()
		cy.url().should('eq', 'https://www.ttu.ee/?id=146605')
	})
})
```

Joonis 8. Näidis testjuhu põhjal Cypress'iga koostatud kasutajaliidese automaattest.

Cypress’i kasutajaliideses näeb antud test välja nagu näidatud järgneval joonisel 9. Roheline värv näitab, et test läbis ehk oli edukas. Mitteläbivad testid tuuakse välja punasena.



Joonis 9. Näidis testjuhu põhjal koostatud kasutajaliidese automaattest Cypress’i kasutajaliideses.

Antud kasutajaliidese automaattesti sisu on identne testiga jooniselt 5 (Joonis 5).

3.4.3 Vahendi valik

Selgitamaks millist vahendit eelpool kirjeldatutest eelistada on kasutatud analüütilist hierarhia protsessi - AHP (*Analytic Hierarchy Process*). AHP on erinevate kriteeriumite põhjal otsuse langetamise meetod, mis seisneb kriteeriumipaaride võrdlemises [11]. Kasutajaliidese automaattestimiseks mõeldud vahendite kriteeriumiteks on valitud järgnevad omadused: õpitavus, mugavus, vigade haldamine ja funktsionaalsus.

AHP hindamismeetod näeb ette kõikvõimalike kriteeriumipaaride võrdlemist vahemikus 1-9, mis on esitatud alljärgnevas tabelis 2. Hinnangud on antud autori poolt. Hinnangute andmisel tugines autor vahenditega tutvumisest ja kasutamisest saadud kogemustele.

Tabel 2. Kriteeriumite võrdlemine.

Kriteerium	**9**	**7**	**5**	**3**	**1**	**3**	**5**	**7**	**9**	**Kriteerium**
Õpitavus							x			Mugavus
Õpitavus								x		Vigade haldamine
Õpitavus									x	Funktsionaalsus
Mugavus						x				Vigade haldamine
Mugavus								x		Funktsionaalsus
Vigade haldamine								x		Funktsionaalsus

Võrdluse tulemustest tuleb koostada maatriks, leida veergude summad, jagada kõik veerus olevad väärtused veeru summaga ning summeerida read korrutades maatriksit kriteeriumite arvu pöördväärtusega [11]. Maatriks on toodud järgneval joonisel 10.

$$\begin{matrix} \\ Õpitavus \\ Mugavus \\ Vigade\ haldamine \\ Funktsionaalsus \end{matrix}\begin{vmatrix} 1 & \frac{1}{5} & \frac{1}{7} & \frac{1}{9} \\ 5 & 1 & \frac{1}{3} & \frac{1}{7} \\ 7 & 3 & 1 & \frac{1}{7} \\ 9 & 7 & 7 & 1 \end{vmatrix} = \frac{1}{4}\begin{vmatrix} \frac{1}{22} & \frac{1}{56} & \frac{3}{178} & \frac{7}{88} \\ \frac{5}{22} & \frac{5}{56} & \frac{7}{178} & \frac{9}{88} \\ \frac{7}{22} & \frac{15}{56} & \frac{21}{178} & \frac{1}{22} \\ \frac{9}{22} & \frac{5}{8} & \frac{147}{178} & \frac{1}{22} \end{vmatrix} = \begin{vmatrix} 0.04 \\ 0.12 \\ 0.20 \\ 0.64 \end{vmatrix}$$

Joonis 10. Kriteeriumite võrdlemise tulemus.

Selgub, et autori jaoks kõige olulisemaks kriteeriumiks on funktsionaalsus, sellele järgneb vigade haldamine ning mugavus. Kõige vähem olulisem kriteerium on õpitavus. Järgmise sammuna viiakse läbi kriteeriumite põhjal vahendite võrdlemine [11]. Tabelis 3 antud hinnangud põhinevad sarnaselt tabelile 2 autori kogemusel.

Tabel 3. Testimisvahendite võrdlemine.

Selenium WebDriver	**9**	**7**	**5**	**3**	**1**	**3**	**5**	**7**	**9**	**Cypress**
Õpitavus			x							Õpitavus
Mugavus								x		Mugavus
Vigade haldamine									x	Vigade haldamine
Funktsionaalsus						x				Funktsionaalsus

Tabelist 3 järeldub selgelt, et autori jaoks olulisemad kriteeriumid on paremini esindatud automaattestide loomise vahendis Cypress, ning täiendavad arvutused tulemuse selgitamiseks ei ole vajalikud. Ainuke omadus, mis on paremini esindatud Selenium WebDriver'it kasutades on õpitavus. Kuid nagu selgus jooniselt 10 on õpitavus autori jaoks kõige vähem olulisem kriteerium.

3.5 Testide loomine

Varem koostatud testplaani ja testjuhtude järgi testide loomine on kõige olulisem samm automaattestimise protsessis, sest ainult korrektselt loodud testid tagavad testitava tarkvaraarenduse kvaliteedi [12]. Silmas tuleb pidada, et kasutajaliidese automaattestid ei dubleeriks madalama kihi teste ehk integratsiooniteste ja ühikteste. Üldine põhimõte on, et testid tuleb kirjutada võimalikult madalal kihil testimise püramiidis. Kui kõrgema kihi test leiab vea, mida madalama kihi test ei leidnud, tuleb madalama kihi teste täiendada. Mida kõrgema kihi testiga on tegu, seda keerulisem on testi kirjutada ja seda kauem võtab testi jooksutamine aega [1].

Selleks, et kasutajaliidese automaattestide loomine oleks mugavam ja loodavad testid ei oleks sõltuvad muutustest kasutajaliideses on soovitatav kasutusele võtta Page Object muster [12]. Page Object muster on loodud eesmärgiga vähendada kasutajaliidese automaattestide loomisel koodi dubleerimist. Page Object on objektorienteeritud klass, mis kapseldab veebilehe kõik detailid, ning testides saab veebilehe elementide poole pöörduda objektidena. Ühtlasi muudab Page Object mustri kasutamine testid vähem sõltuvamaks muutustest kasutajaliideses, sest testid pöörduvad kasutajaliidese poole läbi Page Object klasside [13].

Kahes järgnevas alapeatükis on kirjeldatud Page Object mustri kasutamist mõlema vaatluse all oleva kasutajaliidese automaattestimiseks mõeldud vahendiga.

3.5.1 Selenium WebDriver'i kasutamine Page Object mustriga

Page Object muster sai populaarseks just tänu Seleniumi kasutajatele, ning ka Selenium WebDriver'i dokumentatsioon soovitab testide loomisel Page Object mustrit kasutada [13]. Page Object mustri rakendamine Selenium WebDriver'i testides on järgnevalt kirjeldatud läbi näite.

Esmalt on vaja luua Page Object klass, mida kõik teised loodavad objektid liidestama hakkavad. Page Object klass on toodud järgneval joonisel 11.

```
public class PageObject {
	protected WebDriver driver;

	public PageObject(WebDriver driver){
		this.driver = driver;
		PageFactory.initElements(driver, this);
	}
}
```

Joonis 11. PageObject klass [14].

Näidis testjuhu (Tabel 1) läbimine algab TTÜ koduleheküljelt. Page Object mustri rakendamiseks on vaja luua TTÜ koduleheküljele vastav klass, nagu näidatud järgneval joonisel 12. Lisaks konstruktorile on loodud meetod TTÜ kodulehekülje avamiseks ning meetod menüü poole pöördumiseks.

```
public class TTUHomePage extends PageObject{
	private final HeaderMenu headermenu;
	private static final String TTU_HOME_PAGE_URL =
	"http://www.ttu.ee";

	public TTUHomePage(WebDriver driver) {
		super(driver);
		this.headermenu = new HeaderMenu(driver);
	}

	public TTUHomePage open() {
		driver.get(TTU_HOME_PAGE_URL);
		return this;
	}

	public HeaderMenu getHeaderMenu() {
		return headermenu;
	}
}
```

Joonis 12. TTÜ kodulehele vastav Java klass.

Kuna antud testjuhu läbimisel kasutatakse TTÜ kodulehel olevat menüüd selleks, et navigeerida IT teaduskonna veebilehele, on joonisel 13 loodud eraldi klass ka selle menüü jaoks. Selles klassis pöördutakse kasutajaliidese poole ning lisaks konstruktorile on loodud meetod IT teaduskonna kodulehe avamiseks.

```
public class HeaderMenu extends PageObject {

	@FindBy(xpath="//*[@id=\"dropdownHeader\"]/div/nav/a[3]")
	private WebElement teadustoo;

	@FindBy(xpath="//*[@id=\"dropdownWrapper\"]/div[3]/div[1]
/nav/ul/li[1]/a")
	private WebElement infotehnoloogia_teaduskond;

	public HeaderMenu(WebDriver driver) {
		super(driver);
	}

	public ITTeaduskond openITTeaduskondHomepage() {
		this.teadustoo.click();
		this.infotehnoloogia_teaduskond.click();
		return new ITTeaduskond(driver);
	}
}
```

Joonis 13. TTÜ kodulehel olevale menüüle vastav Java klass.

Samuti on loodud joonisel 14 Page Object mustri rakendamiseks klass, mis vastab IT teaduskonna veebilehekülje.

```
public class ITTeaduskond extends PageObject{

	public ITTeaduskond(WebDriver driver) {
		super(driver);
	}
}
```

Joonis 14. IT teaduskonna veebilehekülgele vastav Java klass.

Selliselt loodud klassid vastavad Page Object mustrile ning lihtsustavad testide koostamist. Lisaks on testid vastupidavamad muutustele kasutajaliideses, sest elemendi aadress kasutajaliideses on koodis ainult ühes kohas, ning seega toimub ka muudatuste tegemine ainult ühes kohas.

Nüüd saab sama testi, mis enne joonisel 5 (Joonis 5) oli loodud ilma Page Object mustrit kasutamata, kirjutada järgnevalt, nagu näidatud joonisel 15. Enam ei pöörduta otse kasutajaliidese poole vaid Page Object mustrit järgides loodud klasside poole.

```
class TestPageObject {
	private WebDriver driver;

	@BeforeEach
	void setUp() throws Exception {
		driver = new ChromeDriver();
		System.setProperty("webdriver.chrome.driver",
System.getProperty("user.dir") + "/chromedriver.exe");
	}

	@Test
	public void test14862(){
		TTUHomePage ttu = new TTUHomePage(driver);
		ttu.open();
		ttu.headerMenu().openITTeaduskondHomepage();
		assertEquals("https://www.ttu.ee/?id=146605",
driver.getCurrentUrl());
	}
}
```

Joonis 15. Näidis testjuhu põhjal Selenium WebDriver'i ja Page Object mustriga koostatud automaattest.

Page Object mustri järgi klasside loomiseks võib kasutada ka Apogeni, mis on tööriist mis genereerib veebilehest automaatselt Page Object mustri järgi Java objektid. Apogeni loojad väidavad, et 75% Apogeni poolt loodud klasse on korrektsed ning ülejäänud 25% klassides tuleb teha minimaalseid muudatusi. Apogeni miinuseks on tõsiasi, et seda saab kasutada ainult Selenium WebDriver'iga automaatteste kirjutades [15].

3.5.2 Cypress'i kasutamine Page Object mustriga

Cypress'i dokumentatsioon ütleb, et eraldi Page Object mustri järgi objektide loomine on ebavajalik ning nimetab Page Object mustri ümber funktsioonide kasutamiseks ja kohandatud käskude loomiseks. Kohandatud käsk sisselogimiseks on näiteks cy.login('email', 'password'), mis pole Cypress'i algselt sisse ehitatud ning mille loomise näide on toodud joonisel 16 [16].

```
Cypress.Commands.add("login", (email, password) => {
   cy.get('input[name=email]').clear().type(email)
   cy.get('input[name=password]').clear().type(password+'{enter}')
 })
```

Joonis 16. Kohandatud käsk sisselogimiseks vahendiga Cypress.

Joonisel 16 kirjeldatud kohandatud käsk otsib kasutajaliidesest e-maili ja parooli sisestamise väljad, tühjendab nad ja kirjutab etteantud e-maili ja parooli vastavatesse lahtritesse. Peale parooli sisestamist vajutatakse sisestusklahvi enter, mis aktiveerib sisse logimise.

Samuti annab Cypress neli soovitust neile, kes soovivad kohandatud käskudega Page Object mustrit jäljendada [16]:

- kohandatud käskude loomisel ja kasutamisel tuleb olla mõõdukas;
- kohandatud käsud võiksid olla triviaalsed;
- kohandatud käske peab saama taaskasutada;
- vältida liigset kasutajaliidese poole pöördumist;

4 Protsessi ja vahendi kasutuselevõtt OÜ Net Group'is

Käesolevas peatükis on kirjeldatud uue kasutajaliidese testimise protsessi kasutuselevõttu OÜ Net Group Beeta tiimi töös. Samuti on kirjeldatud, kuidas kulges töö uue testimisvahendiga Cypress.

Uue kasutajaliidese testimise protsessi kasutuselevõtt langes kokku uue projekti, Tallinna Linnavalitsuse õigusaktide menetlemise infosüsteemi arenduse algusega. Arendus, mis on kasutusloo põhine, jagati sprintideks ehk kahe nädalasteks etappideks. Testija vaatenurgast toimus uue funktsionaalsuse testimine sprindi teises pooles. Sprindi esimene pool jääb ettevalmistusteks ja eelnevates sprintides valminud funktsionaalsuse testimiseks.

Esmalt tutvus testija analüüsiga ja sealhulgas nõuetega ning koostas testplaani. Testplaani koostamise käigus mõtestas testija enda jaoks lahti kogu töö protsessi. Autori arvates on testplaani koostamine projekti alguses vajalik, sest testplaani koostamiseks tuleb aru saada kogu projekti olemusest. Peale testplaani koostamist koostas testija kasutuslugude põhjal testjuhud. Näidisena on tabelis 4 toodud kasutuslugu Tallinna Linnavalitsuse õigusaktide menetlemise infosüsteemis dokumendi leidmisest. Tabelis 5 on näidisena üks selle kasutusloo põhjal koostatud testjuht.

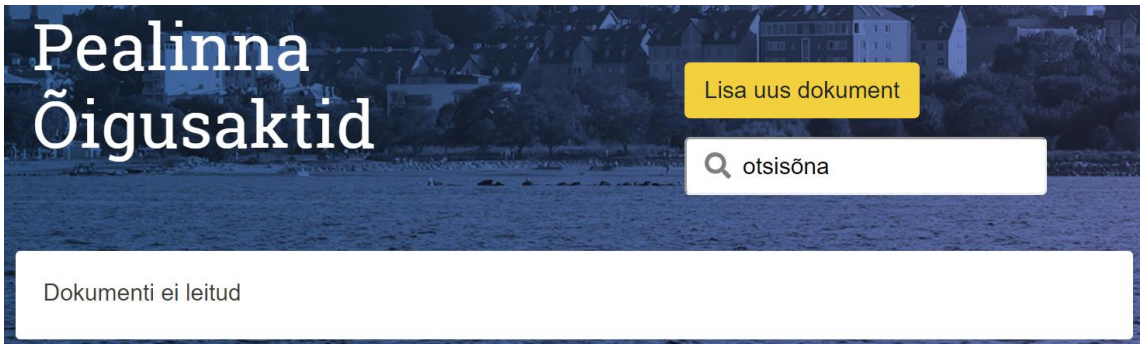
Tabel 4. Kasutuslugu.

Nimetus	Dokumendi leidmine
Tegutsejad	Autoriseeritud kasutaja
Eesmärk	Leida soovitud dokument
Eeltingimused	Kasutaja on autoriseeritud
Päästik	Erinevad tööülesanded
Põhivoog	Kasutaja täidab otsingusüsteemis dokumendi leidmiseks vajalikud väljad kasutades määratud süntaksit. Infosüsteem väljastab otsingutulemuse alusel otsitava. Kasutaja valib leitud dokumendiga teostatava operatsiooni.
Alternatiivvoog	-
Järeltingimused	Dokument on leitud

Tabel 5. Testjuht.

ID	864
Pealkiri	Dokumendi leidmine, negatiivne stsenaarium
Juhend	1. Mine veebilehele 2. Kirjuta otsingulahtrisse otsisõna 3. Kontrolli, kas otsingutulemuste asemele on kirjutatud „Dokumenti ei leitud“
Andmed	Otsisõna – „otsisõna“
Oodatud tulemus	Dokumenti ei leitud

Joonisel 17 on näidatud ekraanipilti olukorrast, mis peale eelnevalt kirjeldatud testjuhu läbimist kasutajale kuvatud peaks olema. Joonisel on näha otsing kuhu on kirjutatud soovitud otsisõna ning teade, et sellist dokumenti ei leitud.



Joonis 17. Ekraanipilt kasutajaliidesest.

Peatükis 3 analüüsitud protsessi põhjal tuleb projekti alguses ettevalmistuse faasis luua kogu projekti testjuhud. Töö käigus aga selgus, et efektiivsem on testjuhte luua igas sprindis eraldi seal aktuaalsetele kasutuslugudele. Põhjus seisneb selles, et testjuhud on väga detailsed, ning projekti alguses on väga keeruline ette näha detaile, mida ei ole veel olemaski. Seetõttu otsustas autor protsessi ümber kujundada selliselt, et testjuhtude koostamine toimuks korduvalt projekti vältel.

Testjuhtude põhjal loodi automaattestid kasutades kasutajaliidese automaattestimise tarkvara Cypress. Automaattestide loomise juures oli autori jaoks kõige raskem vahendi sobivate funktsionaalsuste leidmine. Cypress pakub väga palju testimist lihtsustavaid lahendusi, kuid õige funktsionaalsuse leidmine ja selle kasutama õppimine võtab aega. Vahendit tundma õppides on tegu väga mitmekülgse töövahendiga.

Näiteks on järgmisel joonisel 18 loodud lihtne kasutajaliidese automaattest sisselogimise funktsionaalsuse testimiseks. Automaattest kasutab varasemalt loodud kohandatud käsku (Joonis 16). Joonisel 18 on kaks testjuhtu. Esimene neist on negatiivne stsenaarium, mille puhul sisestatakse sisse logimisel vale parool ning kasutaja jääb samale veebilehele. Positiivse stsenaariumi puhul sisestatakse õige parool ning kasutaja ei tohi enam sisselogimise leheküljel asuda.

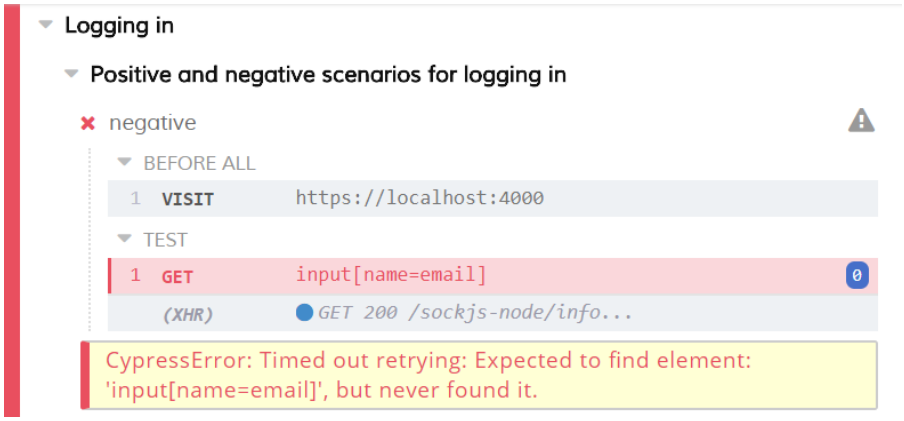
```
describe('Logging in', function(){
 context('Positive and negative scenarios for logging in',
function(){
  before(function(){
   cy.visit('https://localhost:4000')
  })

  it('negative', function(){
   cy.login('kairit.sims@netgroup.ee', 'valeparool')
   cy.url().should('include', '/login')
  })

  it('positive', function(){
   cy.login('kairit.sims@netgroup.ee', '6igeparool')
   cy.url().should('not.include', '/login')
  })
   })
})
```

Joonis 18. Kasutajaliidese automaattest sisselogimise funktsionaalsuse testimiseks.

Ka vigade haldamine käesoleva vahendiga oli mugav. Automaattestid, mis ei läbi on selgelt välja toodud ning võimalik on vaadata ka kuvatõmmiseid testi jooksutamisest, et aru saada, kus viga tekkis. Joonisel 19 on näidatud vigane test kus elementi ei leitud kasutajaliideses üles.



Joonis 19. Vigane kasutajaliidese automaattest.

Antud viga oli tingitud veast automaattesti loomisel, mitte rakenduse funktsionaalsusest. Kõik vead, mis testimise käigus välja tulid kontrollis testija üle ning kui tegu oli arendusest tingitud veaga teavitas testija projektimeeskonda registreerides veakirjelduse versioonihaldustarkvaras Team Foundation Server.

5 Tulemused

Kuigi Tallinna Linnavalitsuse õigusaktide menetlemise süsteemi arendus ei ole lõppenud, on autor kasutajaliidese testide automatiseerimise tulemusega rahul. Võrreldes manuaalselt testitud projektidega on käesoleva projekti tarkvaraarenduse tõhusus suurem ja tarkvaralahenduse kvaliteet kõrgem. Uue kasutajaliidese automaattestimise protsessi ja vahendi valikuga on rahul ka õigusaktide menetlemise süsteemi arenduse projektijuht Birgit Ärm, kelle arvamus on toodud järgnevalt.

„Varasemalt testisime kasutajaliidest manuaalset ja ilma konkreetse plaanita. Tundub, et uue kasutajaliidese automaattestimise vahendi kasutuselevõtt ning automaattestide loomine õigustab ennast, sest juba praegu on näha, et arendustöö on efektiivsem. Uuel protsessil on positiivne mõju ka projekti eelarvele. Loodan, et suudame kliendile üle anda kvaliteetse lahenduse, sest kasutajaliidese automaattestimine aitab vead avastada koheselt ning kliendini need ei jõuagi.“ – Birgit Ärm.

Beeta tiimis edukalt rakendatud uus kasutajaliidese testimise protsess ja vahend äratas huvi ka teistes OÜ Net Group tiimides. Autoril on plaanis tutvustada käesoleva bakalaureusetöö raames analüüsitud ja ellu rakendatud protsessi ning käsitletud vahendeid ka OÜ Net Group ülesel testijate ümarlaual. Autori eesmärgiks on aidata kaasa testimise protsessi parandamisele ka teistes tarkvaraarendustiimides.

6 Kokkuvõte

Käesolev bakalaureusetöö lahendab aktuaalset probleemi tänases tarkvaraarenduse protsessis OÜ Net Group'is, kus kogu testija vastutusalas olev kasutajaliidese testimine käib manuaalselt ning seetõttu ei ole tagatud piisav tarkvara kvaliteet. Üks võimalustest parandada tarkvara kvaliteeti on kasutada kasutajaliidese testimisel automaatteste.

Sellest tulenevalt on käesoleva töö eesmärk välja selgitada tõhusaim kasutajaliidese automaattestide loomise protsess ning sobivad töövahendid. Bakalaureusetöö käigus võeti kasutusele uus kasutajaliidese testimise protsess ja vahend, mille tulemusena vähenes manuaalsele testimisele kuluv aeg, suurenes kasutajaliidese testidega kaetus ning seeläbi tõusis ka tarkvara kvaliteet.

Kasutusele võetud protsess algas projekti nõuetega tutvumisest ja testplaani koostamisest. Seejärel tuli koostada käesolevas tarkvaraarenduse etapis aktuaalsetele kasutuslugudele testjuhud ning kirjutada nende põhjal kasutajaliidese automaattestid. Antud töö raames analüüsiti ja prooviti kaht kasutajaliidese automaattestimiseks mõeldud vahendit - Selenium WebDriver ja Cypress. OÜ Net Group'i jaoks sobiva vahendi leidmiseks kasutas autor analüütilise hierarhia hindamisprotsessi ning seeläbi osutus valituks Cypress. Loodud automaatteste saab projekti jooksul käivitada pidevalt ning kaob vajadus nii suures mahus tarkvara manuaalselt testida. Manuaalne testimine peaks säilima üksnes selleks, et kontrollida kasutajaliidese vastavust disainiõuetele ning kasutusmugavust.

OÜ Net Group'i Beeta tiim jäi käesoleva töö tulemustega rahule ning autor plaanib kasutuselevõetud protsessi ja vahendit tutvustada ka teistele OÜ Net Group tarkvaraarendustiimidele. Bakalaureusetöö on lisaks OÜ Net Group'ile kasulik ka teistele tarkvaraarendusettevõtetele, kes soovivad oma töös kasutajaliidese testimise protsessi parandada ning teste automatiseerida.

7 Kasutatud kirjandus

[1] The Practiccal Test Pyramid. [WWW] https://martinfowler.com/articles/practical-test-pyramid.html (22.03.2018)

[2] Kaur, H., Gupta, G. Comparative Study of Automated Testing Tools: Selenium, Quick Test Professional and Testcomplete. - *Journal of Engineering Research and Applications*, 2013, 3 (5), 1739-1743. [Online] Semantic Scholar (16.03.2018)

[3] GUI Testing. [WWW] https://www.ranorex.com/resources/testing-wiki/gui-testing/ (31.03.2018)

[4] Pettichord, B. Seven Steps to Test Automation Success. - *STAR West conference, California, San Jose, 1999.*

[5] Markvardt, M. Tarkvara testimist käsitlev juhendmaterjal. Tallinn, 2006.

[6] Automation Testing Tutorial: Process, Planning and Tools. [WWW] https://www.guru99.com/automation-testing.html (04.03.2018)

[7] Step by Step Guide to Implement Proof of Concept in Automation Testing. [WWW] http://www.softwaretestinghelp.com/implement-proof-of-concept-poc-in-automation-testing/ (05.03.2018)

[8] How to Write Test Cases: Sample Template with Examples. [WWW] https://www.guru99.com/test-case.html (31.03.2018)

[9] Why cypress? [WWW] https://docs.cypress.io/guides/overview/why-cypress.html (05.04.2018)

[10] Introduction to Selenium. [WWW] https://www.seleniumhq.org/docs/01_introducing_selenium.jsp#test-automation-for-web-applications (14.03.2018)

[11] Sehra, S., Brar, Y., Kaur, N. Applications of Multi-criteria Decision Making in Software Engineering. - *International Journal of Advanced Computer Science and Applications,* 2016, 7 (7), 472-477. [Online] The Science and Information Organization (05.04.2018)

[12] Leotta, M., Clerissi, D., Ricca, F., Spadaro, C. Improving Test Suites Maintainability with the Page Object Pattern: An Industrial Case Study. Luxembourg, 2013.

[13] PageObject. [WWW] https://martinfowler.com/bliki/PageObject.html (01.03.2018)

[14] Selenium. [WWW] https://github.com/SeleniumHQ/selenium (09.05.2018)

[15] Stocco, A., Leotta, M., Ricca, F., Tonella, P. Why Creating Web Page Objects Manually If It Can Be Done Automatically? - *Automation of Software Test 10th International Workshop, Italy, Florence, 2015.*

[16] Using Cypress. [WWW] https://docs.cypress.io/faq/questions/using-cypress-faq.html (18.04.2018)