LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

REGRESA TESTĒŠANAS AUTOMATIZĀCIJAS RĪKA IZVĒLE DARBVIRSMAS LIETOTNĒM

MAĢISTRA DARBS

Autors: Toms Freibergs
Stud. apl. Nr. TF13003
Darba vadītājs: Dr. sc. ing. Edžus Žeiris

RĪGA 2015

ANOTĀCIJA

Programmatūras testēšanai mūsdienās ir arvien lielāka nozīme, tiek izmantotas dažādas testēšanas metodes un rīki. Uzņēmumi, kas nodarbojas ar programmatūras izstrādi, arvien vairāk sāk skatīties uz iespējām automatizēt vismaz daļu no uzņēmuma ierastā darba, ja šis ierastais darbs sastāv no atkārtoti veicamām darbībām.

Darba mērķis ir atrast uzņēmuma SIA ZZ Dats darbvirsmas lietojumprogrammu testēšanas vajadzībām atbilstošāko regresa testu automatizācijas rīku.

Darba apjoms - 64 lpp., 49 attēli, 3 tabulas, 21 bibliogrāfiskā norāde, 1 pielikums. Darbs sastāv no teorētiskās un praktiskās daļas. Teorētiskajā daļā aprakstīta regresa testēšana un kāpēc tā būtu jāveic, kā arī aplūkoti dažādi rīki, kas paredzēti regresa testu automatizēšanai darbvirsmas lietotnēm.

Praktiskajā daļā detalizētāk aplūkoti automatizētas testēšanas rīki, izveidoti konkrēti testpiemēri ar tiem. Testēšana veikta uz SIA ZZ Dats VPS lietojumprogrammas NINO. Maģistra darba rezultātā veikta rīku salīdzināšana pēc vairākiem kritērijiem - funkcionalitāte, lietojamība, izmaksas un rīka atbalstītās vides. Izdarīti secinājumi par to, kurš rīks ir piemērotākais uzņēmuma vajadzībām.

Atslēgvārdi: Regresa testēšana, automatizēta programmatūras testēšana, testēšanas automatizācijas rīki.

ANNOTATION

Title: Regression test automation tool choice for desktop applications.

Nowadays, software testing is becoming more popular and many different methods and testing tools are being used. Companies that develop software are looking into possibilities of automating at least a part of their business, especially if the business consists of tasks that have to be done repetitively.

The goal of the work is to find a suitable regression testing automation tool for desktop applications for the needs of SIA ZZ Dats.

The work consists of 64 pages, 49 images, 3 tables and 21 bibliographic references, 1 attachment. The work consists of a theoretical part and a practical part. In the theoretical part of the work, the author describes regression testing in general and why it should be done. There is also an overview of different software testing tools that are specialized for desktop application regression testing.

In the practical part of the paper the author reviews the tools and also executes specific test cases with them. Testing is done on SIA ZZ Dats VPS software NINO. As a result of the Master's thesis, the author makes conclusions about which tool is the most suitable for SIA ZZ Dats based on criteria such as functionality, usability, costs and environments which the tools support.

Keywords: Regression testing, automated software testing, testing automation tools.

AUTOREFERĀTS

Maģistra darba laikā autors ir veicis sekojošas darbības:

- Pētījis dažādus automatizētās testēšanas rīkus gan no teorētiskā, gan no praktiskā viedokļa;
- Ar apskatītajiem rīkiem izveidojis testu kopas uz SIA ZZ Dats VPS lietojumprogrammas NINO;
- Veicis apskatīto rīku salīdzinājumu un izteicis savu viedokli, kurš no rīkiem varētu būt piemērotākais uzņēmuma vajadzībām.

Darba novitāte ir vienuviet pieejams regresa testēšanas rīku salīdzinājums darbvirsmas lietotnēm. Parasti dažādos avotos ir pieejams divu rīku salīdzinājums, bet šajā darbā praktiski ir salīdzināti 6 rīki.

Literatūras avoti galvenokārt sastāv no tīmeklī pieejamās informācijas par apskatītajiem rīkiem un rīku dokumentācijas. Galvenokārt šī informācija tika iegūta no rīku oficiālajām mājas lapām. Kopumā darbā tika izmantotas 21 atsauces, no kurām 1 ir grāmata, 1 ir akadēmiskā terminu vārdnīca, pārējās ir rīkiem veltītās tīmekļa lapas.

Pēc autora domām darbs ir izklāstīts pietiekami detalizēti. Praktiskajā daļā veidojot testpiemērus, pie katra rīka ir pievienoti vairāki ekrānuzņēmumi, lai lasītājam būtu vieglāk uztvert rīku funkcionalitāti.

Darba lielāko daļu aizņem tieši praktiskais darbs. Katrā no rīkiem izveidotas 2 testu kopas. Citos rīkos tās aizņēma vairāk laika, citos mazāk. Katra rīka izpētīšanai, testu kopu izveidošanai un rīka aprakstīšanai vidēji tika veltīta aptuveni pusotra nedēļa.

Iegūtie rezultāti jeb rīku salīdzinājums ir pietiekami precīzs, jo salīdzinājuma pamatā tika izmantota informācija no rīku oficiālajām mājas lapām. Autora iegūtā personīgā pieredze, strādājot ar šiem rīkiem, sakrīt ar mājas lapās sniegto informāciju.

Darbā valodas kļūdām nevajadzētu būt, jo darbs izstrādes laikā tika vairākas reizes pārlasīts, kā arī tiek izmantots pareizrakstības pārbaudes rīks. Darba struktūra un noformējums atbilst maģistra darba izstrādes un aizstāvēšanas metodiskajiem norādījumiem.

Visa informācija, kas ir aizgūta no citiem avotiem, ir norādīta ar attiecīgo atsauci.

SATURS

APZĪMĒJUMU SARAKSTS

VPS – Vienotā Pašvaldību Sistēma

NINO – Vienotās Pašvaldību Sistēmas nekustamā īpašuma nodokļa administrēšanas lietojumprogramma

IDE (*Integrated Development Environment*) – integrētā izstrādes vide

JVM (*Java Virtual Machine*) – *JAVA* virtuālā mašīna

JRE (*Java Runtime Environment*) – *JAVA* izpildlaika vide

API (*Application Programming Interface*) – lietojumprogrammas programmēšanas saskarne

OpenCV (*Open Source Computer Vision*) – atvērta pirmkoda datora redze

UFT (*Unified Functional Testing*) – *Hewlett-Packard* firmas radīts testēšanas rīks

QTP (*QuickTest Professional*) – *Hewlett-Packard* radīts testēšanas rīks, pārsaukts par UFT

HP – *Hewlett-Packard* firma

JAR (*Java Archive*) – programmēšanas valodas *JAVA* pakotnes faila formāts

AkadTerm – akadēmiskā terminu datubāze

AUT (*Application Under Test*) – lietotne, kura tiek testēta

IEVADS

Autors izvēlējās šo tēmu, jo autora darba vietā – SIA ZZ Dats – pieaug nepieciešamība pēc testēšanas automatizācijas. Uzņēmumā rodas situācijas, kad pēc programmatūras versijas maiņas nākas atkārtoti izpildīt iepriekš veiktos testpiemērus. Šī iemesla dēļ ir nepieciešams atrast testēšanas rīku, kas atvieglos regresa (un iespējams citu, piemēram, vienību, veiktspējas vai stresa) testu veikšanu.

Darba mērķis ir atrast uzņēmuma SIA ZZ Dats darbvirsmas lietojumprogrammu testēšanas vajadzībām atbilstošāko regresa testu automatizācijas rīku. Rīki tiks vērtēti pēc tādiem kritērijiem kā funkcionalitāte, izmaksas, lietojamība, kā arī pēc tā, kādās vidēs rīks ir lietojams. Testēšana tiks veikta uz SIA ZZ Dats VPS lietojumprogrammas NINO, jo pats darba autors visvairāk strādā tieši ar šo lietojumprogrammu. Tas nozīmē, ka izveidotie testpiemēri būs precīzāki un atbilstošāki uzņēmuma vajadzībām, jo autoram ir pieredze ar šo lietojumprogrammu. Rīks, kurš visvairāk atbildīs izvēlētajiem kritērijiem, varētu tikt pielietots arī citām uzņēmuma informācijas sistēmām. Mērķa sasniegšanai nepieciešams veikt šādus uzdevumus:

- Apskatīt vismaz piecus testēšanas automatizācijas rīkus, iepazīties ar to iespējām, aprakstīt šos rīkus teorētiskajā daļā;;
- Praktiskajā daļā detalizētāk aplūkot izvēlētos rīkus, aprakstīt kā strādāt ar rīkiem un kā izveidot testu kopas lietojumprogrammai NINO;
- Nepieciešams izveidot iepriekš definētās testu kopas;
- Testa kopu izveidošanas nolūkos nepieciešams apgūt kādu no aplūkotajos rīkos izmantotajām skriptēšanas valodām;
- Nepieciešams savstarpēji salīdzināt testēšanas rīkus pēc izvirzītajiem kritērijiem un pieņemt lēmumu, kurš rīks varētu būt atbilstošākais uzņēmuma vajadzībām.

1. REGRESA TESTĒŠANA, TEORĒTISKS PĀRSKATS

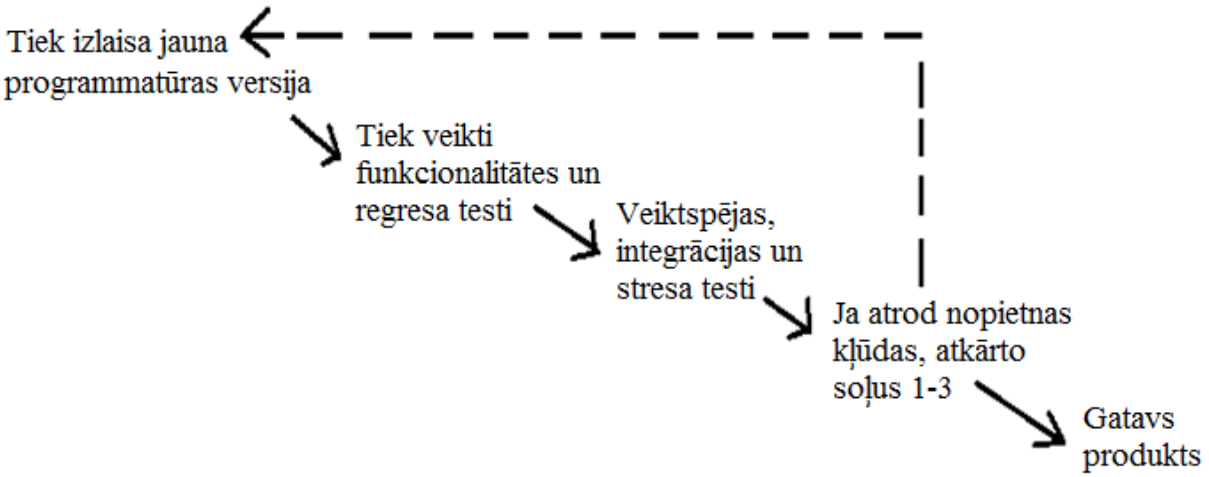
1.1. Regresa testēšanas definīcija un mērķi

Pirms rīku aplūkošanas, ir nepieciešams noskaidrot, kas tad īsti ir regresa testēšana un kāpēc tā ir nepieciešama. Dažādos informācijas avotos ir pieejamas dažādas regresa testēšanas definīcijas, bet pamatideja visām definīcijām ir vienāda. Regresa testēšana nosaka vai iepriekš strādājošā funkcionalitātē tika ieviestas kļūdas, cenšoties izlabot iepriekš atrastu kļūdu vai arī pievienojot lietotnei jaunu funkcionalitāti[1]. Bieži vien notiek tā, ka, veicot kādas kļūdas labojumus, kaut kur citur sistēmā dēļ šiem labojumiem tiek izveidotas jaunas kļūdas. Bez regresa testiem šīs jaunās kļūdas ir grūti pamanīt, jo parasti tiek pārbaudīts tikai tas, vai iepriekš pamanītās kļūdas jaunajā versijā ir izlabotas.

1.2. Regresa testēšanas izmantošana

Parasti tiek izšķirtas divas regresa testēšanas metodes. Viena no tām darbojas pēc principa, ka visas kļūdas tiek dokumentētas un pēc katras jaunas programmatūras versijas sistēma tiek pilnībā pārtestēta, lai gūtu pārliecību, ka kļūdas nav atkal parādījušās. Šī metode visbiežāk tiek veikta manuāli. Otra metode darbojas pēc principa, ka testēšanā izmanto bāzes testpiemēru komplektu. Katru reizi palaižot šos bāzes testus, tiek pārbaudīts, ka sistēmas funkcionalitāte nav mainījusies programmatūras versijas maiņas rezultātā. Arī šo metodi var veikt manuāli, bet šī metode darbojas labāk, ja tā tiek automatizēta, jo bāzes testpiemēri lielā mērā ir nemainīgi. Šajos gadījumos ir jāuzmanās, vai konkrētu testpiemēru ir iespējams izmantot atkārtoti, jo var būt gadījumi, ka tiek veiktas konkrētas darbības, piemēram, personai izdrukāts maksāšanas paziņojums par nodokļa aprēķinu. Programmatūrā var būt iebūvēta kontrole, kas neļauj atkārtoti izdrukāt maksāšanas paziņojumu. Jāņem vērā, ka pēc katras jaunas programmatūras versijas, bāzes testpiemērus būtu vēlams papildināt, jo pastāv iespēja, ka programmatūrā ir ieviestas jaunas funkcijas.

Regresa testēšanas pozīcija kopējā testēšanas ciklā izskatās apmēram šādi:



1.1. att. **Regresa testēšana testēšanas ciklā**

Attēlā 1.1. redzams, ka katru reizi, kā tiek izlaista jauna programmatūras versija, parasti tiek veikti funkcionalitātes un regresa testi, lai pārbaudītu, ka programmatūras funkcionalitāte nav negatīvi mainījusies. Pēc tam tiek veikti integrācijas testi, lai pārliecinātos, ka programmatūras komponentes joprojām ir savstarpēji saderīgas. Papildus veic veiktspējas un stresa testus, lai pārliecinātos, ka pēc izmaiņām programmatūra ir spējīga tikt galā ar paredzēto lietotāju skaitu un datu apjomu, kā arī vai programmatūra tiek galā ar palielinātu lietotāju skaitu un datu apjomu.

Runājot par regresa kļūdām, parasti tiek izdalīti trīs kļūdu veidi:

- Lokālās kļūdas – kļūdas, kuras tiek atrastas „jaunajā" (tikko uzrakstītajā) kodā;
- Atmaskotās kļūdas – kļūdas, kuras tiek atklātas tikko uzrakstītajā kodā, bet kuras programmatūras iepriekšējās versijās nekādu ietekmi neradīja;
- Attālās kļūdas – kļūdas, kuras kādas komponentes koda izmaiņu rezultātā tiek atklātas pavisam citā komponentē.

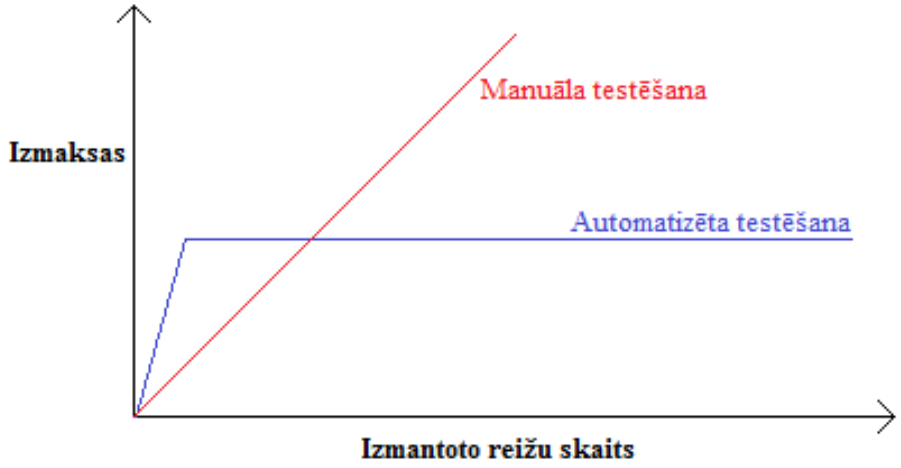
1.3. Regresa testēšanas priekšrocības, trūkumi un labā prakse

Regresa testēšana parasti ir laikietilpīgs process gan no testpiemēru sagatavošanas viedokļa, gan no šo testpiemēru manuālas vai automatizētas izpildes viedokļa, it īpaši tad, ja programmatūras apjoms ir liels un salīdzinoši bieži tiek izlaistas jaunas programmatūras versijas. Jo laikietilpīgāka ir šī testēšana, jo dārgākas ir izmaksas. Ja regresa testēšana tiek veikta manuāli, testētājiem šis process varētu būt nogurdinošs un varētu tikt zaudēta darba kvalitāte, jo bieži vien lielākā daļa testpiemēru kļūdas neatrod, kā arī pēc katras programmatūras versijas maiņas jāveic vienas un tās pašas darbības. Tāpat jāņem vērā, ka

kāda kļūda var netikt pamanīta – šis attiecināms gan uz manuālo regresa testēšanu, gan automatizēto regresa testēšanu. Manuālas regresa testēšanas gadījumā kļūdas var nepamanīt tādēļ, ka testētājam zūd motivācija un darbs netiek veikts tik kvalitatīvi, kā tas būtu tad, ja testētājs ir ieinteresēts un motivēts. Automatizētas testēšanas gadījumā kļūdas var nepamanīt tajos gadījumos, ja šīs kļūdas nav ietvertas bāzes testpiemēru kopā, jo parasti veicot automatizētu testēšanu, tiek uzskatīts, ka programmatūra ir pilnībā pārbaudīta, lai gan patiesībā tā nav.

Neskatoties uz šiem trūkumiem, ir tomēr jāskatās uz regresa testēšanas priekšrocībām – pirmkārt, tiek pārbaudīts tas, ka visas funkcijas strādā un otrkārt, līdzīgi kā vienību testos, izstrādātāji gūst pārliecību, ka pēdējais rakstītais kods ir bijis pareizs un nav izraisījis kādu nopietnu kļūdu. Kopumā regresa testēšana būtiski palielina produkta kvalitāti. No praktiskā viedokļa ir neiedomājama programmatūras versijas palaišana produkcijā bez regresa testu veikšanas.

Ja tiek salīdzinātas priekšrocības starp manuālo un automatizēto regresa testēšanu izmaksu ziņā, ir noderīgi aplūkot attēlu 1.2.



1.2. att. **Manuālās un automatizētās testēšanas izmaksu vispārējs salīdzinājums**

Kā redzams attēlā, jaunam projektam, ja testēšanu paredzēts veikt ātri un tikai dažas reizes, tad manuāla testēšana ir izdevīgāka nekā automatizētā testēšana, jo automatizētas testēšanas gadījumā sākotnēji ir jāvelta laiks testa plāna izveidošanai un testpiemēru rakstīšanai. Ilgtermiņā automatizētā testēšana būtiski atmaksājas, jo iepriekš izveidotos testpiemērus var izmantot atkārtoti un šo testu atkārtošana parasti neaizņem daudz laika, jo visbiežāk šos testus izpilda pēc darbadienas beigām. Ja projekts ir ilgstošs un programmatūra krasi nemainās, tad būtu ieteicams padomāt par testpiemēru automatizēšanu.

Regresa testēšanas labajā praksē tiek izmantoti šādi padomi:

- Jāizstrādā regresa testēšanas plāns, lai tiktu ietverti visi sistēmas būtiskākie testēšanas aspekti;
- Visas atrastās un labotās kļūdas ir jāiekļauj bāzes testpiemēru kopā, jo vietās, kur kļūdas ir kādreiz parādījušās, citās programmatūras versijās šīs pašas kļūdas var atkārtoties;
- Regresa testēšanu vajadzētu sākt jau uzreiz pēc „dūmu" testa. Dūmu tests ir vispārēja programmatūras pārbaude – tiek pārbaudīts, vai strādā pamatfunkcionalitāte un vai vispār ir jēga sākt citu testu izpildi;
- Automatizētos regresa testus vajadzētu veikt reizi dienā (laikā, kas netraucē izstrādes darbus) – šis ieteikums vairāk attiecas uz projektiem, kuriem dienas laikā tiek veikta viena vai vairākas versijas maiņas. Lielākiem projektiem, kuriem versiju maiņa notiek retāk, pietiek, ja regresa testus veic pēc versijas maiņas, neveicot tos katru dienu;
- Ja vien tas ir iespējams, vajadzētu regresa testus pēc iespējas vairāk automatizēt, jo tā kā nāksies šos testus palaist katru reizi, kad ir nomainījusies programmatūras versija, šo testu automatizācija būtiski samazinās testēšanas izmaksas un patērēto laiku.

2. TESTĒŠANAS RĪKU TEORĒTISKAIS PĀRSKATS

Maģistra darba praktiskajā daļā tiks izmantota VPS lietojumprogramma NINO. NINO ir darbvirsmas lietotne, kura galvenokārt bāzēta uz *Visual Basic*. Tas nozīmē, ka teorētiskajā daļā pie potenciālo rīku atlases ir jāņem vērā šie nosacījumi, lai nerastos situācija, ka teorētiskajā daļā apskatītie rīki nemaz nav pielietojami praktiskajā daļā.

Gūstot priekšstatu no dažādiem tīmeklī atrastajiem avotiem par darbvirsmas lietotņu testēšanu, autors ir nolēmis detalizētāk apskatīt tālāk minētos rīkus.

2.1. Sikuli

Rīks *Sikuli*[2] piedāvā automatizēt visu to, kas ir redzams uz ekrāna, izmantojot attēlu atpazīšanas tehnoloģijas. Tas nozīmē, ka šim rīkam nav nekādi ierobežojumi uz to, kāda veida lietotni ir paredzēts testēt – šis rīks derēs gan darbvirsmas, gan tīmekļa, gan mobilajām lietotnēm. Rīks atbalsta dažādas operētājsistēmas – gan *Windows*, gan *Linux*, gan *Mac OSX*.

Idejiski tas notiek tā – lietotājs saglabā kā attēlus uz ekrāna redzamās pogas, izvēlnes, laukus un citus lietotnes elementus, ieliek šos attēlus kodā un pēc tam veic darbības ar šiem attēliem. Pēc pirmā iespaida autors uzskata, ka šis rīks šī darba mērķa sasniegšanai varētu būt noderīgs, jo ir jāņem vērā fakts, ka ne visi uzņēmuma testētāji ir ar profesionālām programmēšanas prasmēm, tādēļ ir svarīgi atrast tādus rīkus, kurus varētu izmantot pēc iespējas lielāks darbinieku skaits. Vēl svarīgi ir pieminēt, ka rīks ir atvērta koda un ir pieejams bez maksas.

Plaša informācija par rīka tehnisko izmantošana pieejama dokumentācijā, kas izvietota rīka mājas lapā.[3]. Šī informācija tiks detalizētāk izmantota darba praktiskajā daļā, kad ar šī rīka palīdzību tiks veidoti testpiemēri.

2.2. TestComplete

TestComplete[4] ir testēšanas rīks, kurš piedāvā veikt testēšanu gan uz darbvirsmas, gan uz tīmekļa lietojumprogrammām, gan uz mobilajām lietotnēm. Jāpiemin, ka šis ir maksas rīks, bet ir pieejama 30 dienu izmēģinājuma versija, kuru autors izmantos praktiskajā daļā.

Rīks atbalsta gan 32-bitu, gan 64-bitu lietotnes, kuras izstrādātas sekojošās vidēs:

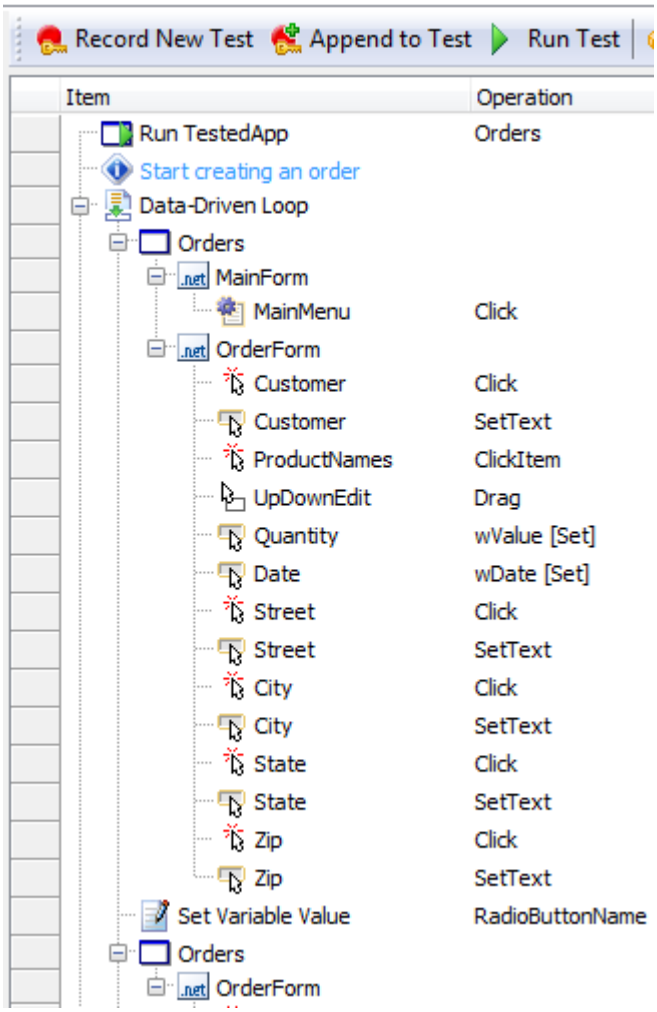
- *C/C++;*
- Jebkura *.NET lietotne;*

- *WPF (XAML);*
- *Visual Basic;*
- *Java;*
- *JavaFX;*
- *Delphi;*
- *C++Builder;*
- *Qt.*

TestComplete rīka apraksts mājas lapā sniedz priekšstatu, ka ar šo rīku var strādāt gan testētāji, kuriem ir skriptu programmēšanas pieredze, gan testētāji, kuriem nav programmēšanas prasmes. Tas iespējams, pateicoties darbību ierakstīšanas un izpildīšanas (*record and playback*) funkcionalitātei. Lietotājs palaiž ierakstīšanas funkciju, kā parametru izvēloties to, kāds tests tiks ierakstīts.

2.1.att. **Tiek izvēlēts, kāds tests tiks ierakstīts[4]**

Piemēram, ja lietotājs izvēlas ierakstīt tā saucamo atslēgvārdu testu, tad pēc ieraksta pārtraukšanas ir iespējams aplūkot un rediģēt darbības atslēgvārdu testu redaktorā. Testa piemērs, kurš darbojas uz atslēgvārdu principa, aplūkojams attēlā 2.2.



2.2. att. **Atslēgvārdu testa piemērs**

Kā redzams attēlā, šajā testā tiek izpildītas darbības, balstoties uz atslēgvārdiem, piemēram, tiek atrasts klienta (*Customer*) lauks, tiek veikts peles klikšķis šajā laukā un tiek ierakstīts teksts. Tāpat tiek atrasts, piemēram, datuma lauks, kurā tiek ievadīts datums.

Ir iespējams izveidot arī objektu bāzētus ierakstus. Arī šie ieraksti saglabā lietotāja veiktās darbības, bet jāņem vērā, ka tā nav tikai peles klikšķu ierakstīšana noteiktās ekrāna koordinātās. Rīks spēj no lietotnes nolasīt objekta vārdu un atribūtus, kā arī ierakstīt, kādas darbības ar šo objektu tika veiktas, piemēram, objekta iezīmēšana, teksta ievade, kā arī dažādas darbības ar slēdžiem. Objektu bāzēti ieraksti ir noderīgi tajos gadījumos, kad lietotnes grafiskā saskarne programmatūras versijas maiņu rezultātā mainās. Šis ir veids, kā nodrošināt, ka testi joprojām strādās arī pēc šīm izmaiņām.

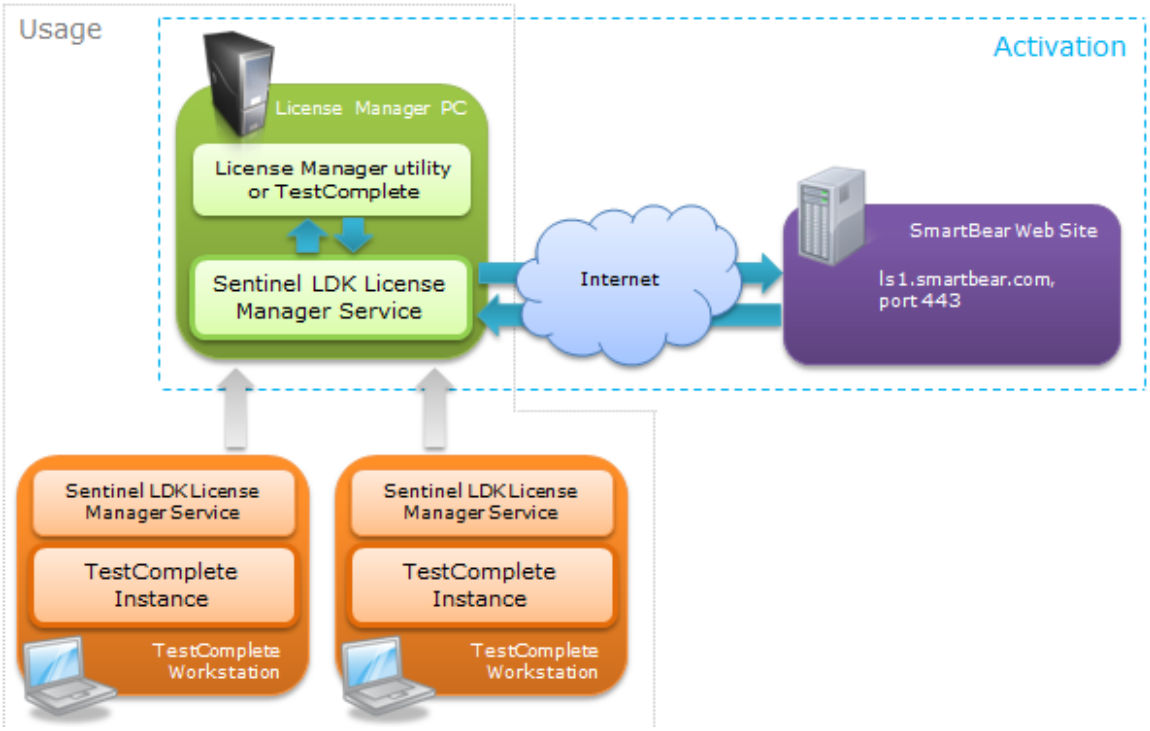
Tāpat ir iespējams ierakstītās darbības izmantot kā pamatu dažādu skriptu veidošanai, piemēram, *VBScript, Jscript, DelphiScript, C++Script*, kā arī *C#Script*.

Noderīgi ir ierakstīt arī zema līmeņa procedūras, kuras ierakstot tiek ņemtas vērā klaviatūras un peles darbības, piemēram, ievadītais teksts, peles kustības, peles klikšķi, peles

ritināšana, kā arī tiek ieturētas tikpat lielas pauzes, kādas tās bija darbību ierakstīšanas laikā. Šīs procedūras ir noderīgas regresa testu veikšanai, it īpaši tad, ja programmatūra, uz kuras tiek izpildītas procedūras, nav mainījusi grafisko saskarni.

Rīka dokumentācija un atbalsts pieejama rīka mājas lapā.[5] Šī informācija tiks izmantota praktiskajā daļā, kad ar šo rīku tiks veidoti testpiemēri.

Lai izmantotu šo rīku, ir jāiegādājas *TestComplete* platforma, kas maksā 889 EUR vienam datoram vai 2229 EUR vairākiem datoriem tīklā (tā saucamā peldošā licence), bet šis datoru skaits tīklā ir ierobežots. Mājas lapā nav norādīts, cik tieši datoriem šī licence strādā, bet teorētiski ir galvenais dators, kas ir licences pārvaldnieks un visi pārējie tīklā esošie datori bez licences var iegūt izmantošanas pozīciju no licences pārvaldnieka datora. Kad darbs ar *TestComplete* tiek pabeigts, izmantošanas pozīcija atbrīvojas un pārvaldnieka datoram var pieslēgties citi datori, kuri vēlas izmantot šo rīku. Šis process izskatās šādi:



2.3. att. ***TestComplete*** **izmantošana uz vairākiem datoriem tīklā**

2.3. Ranorex

Ranorex[6] ir rīks, kurš piedāvā dažādu tehnoloģiju atbalstu, kā arī šis rīks ir pielietojams gan darbvirsmas, gan tīmekļa lietojumprogrammu, kā arī mobilo lietotņu testēšanā. Atbalstīto tehnoloģiju sarakstā ir *.NET, Java, Flash, HTML5, Winforms, Silverlight, Android, iOS, Delphi* u.c. Testēšanu var veikt gan uz 32-bitu, gan uz 64-bitu lietotnēm.

Tāpat kā *TestComplete*, arī *Ranorex* piedāvā divu līmeņu testēšanu – darbinieki bez programmēšanas prasmēm var izmantot darbību ierakstīšanas un atkārtošanas funkcionalitāti, savukārt darbinieki ar programmēšanas prasmēm papildus iepriekš minētajai darbību ierakstīšanai un atkārtošanai var pielietot atslēgvārdu testus, kā arī papildināt kodu, izmantojot *Ranorex* piedāvāto *API*. *API* atbalsta gan *C#,* gan *VB.NET* kodu. Tāpat jāņem vērā, ka tāpat kā *TestComplete, Ranorex* rīks ierakstīšanas funkcijas veic objektu atpazīšanas līmenī.

Arī *Ranorax* ir maksas rīks, bet ir pieejama rīka izmēģināšana uz 30 dienām bez maksas. Izpildes vide pieejama par 690 EUR. Ar šo izpildes vidi nevar izveidot jaunus testpiemērus, bet var izpildīt jau izveidotus testpiemērus.

Rīkam ir pieejama 479 lappušu bieza dokumentācija, kura autoram būtiski noderēs praktiskā darba laikā, kad nāksies izveidot un izpildīt testpiemērus ar šo rīku.[7]

2.4. Squish

Kompānija *froglogic*[8] piedāvā divu veidu *Squish* rīkus. *Squish Coco* ir rīks, kas paredzēts programmatūras pirmkoda pārbaudei. Pēc testu veikšanas *Squish Coco* iezīmē tās koda daļas, kuras netika pārbaudītas, kā arī paziņo par tiem testpiemēriem, kas dublēja kādu citu testpiemēru darbības. Tāpat šis rīks piedāvā salīdzināt testpiemēru pārklājumu starp dažādām konkrētas programmatūras versijām. Rīks paredzēts *C, C++, C#* un *Tcl* kodam.

Otrs piedāvātais *Squish* rīks ir *Squish GUI Tester* jeb *Squish* grafiskās lietotāju saskarnes testētājs. Šis rīks izmantojams gan darbvirsmas, gan tīmekļa, gan mobilajām lietotnēm, kā arī iegultajām lietotnēm. Arī šis rīks piedāvā veikt darbību ierakstīšanu gan pamata, gan objektu atpazīšanas līmenī.

Pēc ierakstu veikšanas ir iespējams ģenerēt *Python, JavaScript, Perl, Ruby* vai *Tcl* skiptēšanas valodu kodu.

Arī *Squish* ir maksas rīks, ar kuru ir iespējams iepazīties izmēģinājuma versijā. *Squish Coco* ir pieejams par sekojošām cenām:

2.1. tabula

***Squish Coco* cenas**

Lietotāji	Cena
5	3000 EUR
10	5880 EUR
15	8640 EUR
20	11280 EUR
25	13800 EUR

30	16200 EUR

Jāņem vērā, ka tabulā minētās cenas attiecas tikai uz vienu no programmēšanas valodu komplektiem. Šie komplekti ir:

- *C/C++;*
- *C#;*
- *Tcl.*

Rīka mājas lapā norādīts, ka gadījumā, ja vēlaties iegādāties vairāk kā 30 lietotāju licences, peldošās vai uzņēmumu licences, tad ir jākontaktējas ar *froglogic* pārstāvi pa e-pastu, lai saņemtu cenas piedāvājumu.

Gan *Squish Coco*, gan *Squich GUI Tester* dokumentācija pieejama rīku mājas lapā.[9] Tā tiks izmantota praktiskajā daļā.

2.5. AutoIt

AutoIt[10] ir skriptēšanas valoda, kas radīta *Windows* grafiskās lietotāju saskarnes automatizēšanai un vispārējai skriptu rakstīšanai. Rīks ir speciāli radīts, lai tas būtu maza izmēra (neaizņemtu daudz vietas uz lietotāja cietā diska), kā arī lai tas darbotos patstāvīgi bez citas programmatūras nepieciešamības. Rīks izmantojams gan uz 32-bitu, gan uz 64-bitu lietotnēm. Jāņem vērā, ka šis ir bezmaksas rīks.

Papildus rīka labums ir fakts, ka ar rīku izveidotos skriptus var pārkonvertēt par neatkarīgām izpildāmām programmām (*.exe* failiem). Tas nozīmē, ka izveidotos skriptus var viegli izmantot arī uz datoriem, uz kuriem nav instalēta *AutoIt* programmatūra. Tas izdarāms ar *Aut2Exe* programmatūru, kas nāk līdzi standarta *AutoIt* instalācijai.

Rīks piedāvā peles un klaviatūras darbību simulāciju, kā arī dažādu darbību veikšanu ar *Windows* formām, piemēram, loga samazināšana, paplašināšana, pārvietošana, atvēršana, aizvēršana utt. Īsumā sakot, ar šo rīku vajadzētu varēt ar *Windows* formām izdarīt visu to, ko varētu izdarīt jebkurš lietotājs. Rīka mājas lapā ir pieejama arī dokumentācija, kura tiks izmantota praktiskajā daļā.[11]

2.6. Unified Functional Testing

Unified Functional Testing (UFT)[12] rīks, iepriekš zināms kā *QuickTest Professional (QTP),* ir *Hewlett-Packard* firmas radīts testēšanas rīks, kurš specializējas visa veida

programmatūras funkcionalitātes testēšanā, tai skaitā arī regresa testu veikšanā. Rīks tiek saukts par apvienoto funkcionalitātes testēšanu, jo izmanto vienu apvienotu grafisko saskarni gan *API* testēšanai, gan *GUI* testēšanai.

Tāpat kā citi rīki, arī *UFT* piedāvā veikt gan darbību ierakstīšanu, gan atslēgvārdu testēšanu, kā arī ir iespējams papildināt testpiemērus ar skriptiem. Kā skriptēšanas valodu *UFT* izmanto *Visual Basic Scripting Edition (VBScript).*

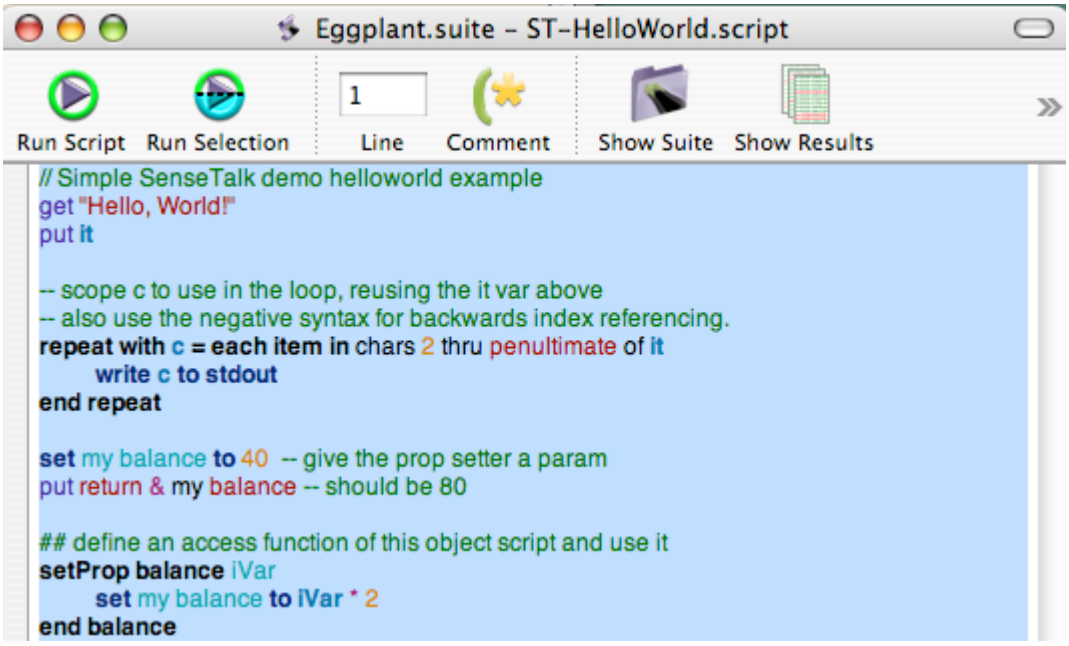
Arī *UFT* ir maksas rīks, kuram pieejama izmēģinājumu versija uz 30 dienām. Arī šis rīks piedāvā gan viena datora licences, gan vairāku datoru (peldošās) licences. Mājas lapā nav norādītas konkrētas cenas, cik katra no licencēm maksā, tādēļ nepieciešams kontaktēties pa e-pastu ar firmas pārdošanas speciālistu, lai iegūtu cenu piedāvājumu konkrētām vajadzībām.

Unified Functional Testing rīka dokumentācija ir grūtāk atrodama, nekā citu rīku dokumentācija, kā arī ir nepieciešama reģistrācija, lai varētu izmantot *HP* pasi, lai piekļūtu dokumentācijai.[13]

2.7. eggPlant Functional

Kompānija *TestPlant*[14] piedāvā dažādus rīkus, bet šī darba ietvaros autors apskata *eggPlant Functional*. Rīks paredzēts grafiskās saskarnes testēšanai. Primāri rīks paredzēts mobilo lietotņu testēšanai, bet, tā kā rīks izmanto attēlu atpazīšanas un analīzes tehnoloģijas līdzīgi kā *Sikuli*, tad rīks ir pielietojams dažādām platformām, tai skaitā *Windows* darbvirsmas lietotnēm. Jāņem vērā, ka attēlu atpazīšana netiek veikta objektu līmenī, kā tas ir citos populāros maksas rīkos. Pateicoties attēlu atpazīšanas un darbību ierakstīšanas un izpildīšanas tehnoloģijām, rīku var izmantot gan ar, gan bez programmēšanas prasmēm.

Rīks ir izmantojams kopā ar dažādām programmēšanas valodām, piemēram, *Ruby, C#, Java*. Skriptu rakstīšanai rīks izmanto speciālu skriptēšanas valodu *SenseTalk. SenseTalk* skriptēšanas valoda ir radīta ar mērķi, lai tā būtu līdzīga angļu valodai, tādējādi cenšoties to padarīt viegli apgūstamu un izmantojamu. *SenseTalk* valodā rakstīts „Sveika pasaule" programmas kods redzams attēlā 2.4.



2.4. att. ***SenseTalk*** **koda piemērs**

Arī *eggPlant Functional* ir maksas rīks, kuram arī ir pieejamas gan viena datora licences, gan grupas (peldošās) licences. Rīka mājas lapā nav precīzi norādīts, kādas tieši ir licenču cenas, bet no citiem avotiem tika iegūta informācija, ka pirmā licence maksā $3400 un katra nākošā uz pusi mazāk, respektīvi, $1700, $850 un pēc tam visas pārējās par $425. Atšķirībā no citu rīku licencēm, viena *eggPlant* licence der gan uz *Windows*, gan *MacOSX*, gan *Linux*. Citu rīku licences katrai no šīm platformām būtu jāiegādājas atsevišķi.

Lai efektīvi izmantotu *eggPlant* rīku, ir jāiegādājas gan *eggPlant Functional Development*, gan *eggPlant Functional Execution* licences – tātad vajag atsevišķas licences testu izstrādes videi un testu izpildes videi. Tāpat ja ir nepieciešams *eggPlant* pārvaldības rīks, ar kura palīdzību ir iespējams kontrolēt *eggPlant Functional* instances, tad par katru instanci ir jāiegādājas atsevišķa *eggPlant Manager* licence.

Rīka mājas lapā ir pieejama arī dokumentācija.[15]

3. TESTĒŠANAS RĪKU PRAKTISKAIS PĀRSKATS

3.1. Praktiskajā daļā izmantotās testu kopas

Ar katru no aplūkotajiem automatizētās testēšanas rīkiem tiks izveidotas divas testu kopas lietojumprogrammā NINO. Katra no šīm kopām sastāvēs no noteiktu darbību skaita un katrā no rīkiem testu kopas tiks veidotas pēc iespējas līdzīgākas cita citai, bet jāņem vērā, ka katrā rīkā var būt kādas nianses, kuru dēļ testu kopas var nedaudz atšķirties.

Testu kopas netika izvēlētas un izveidotas pēc konkrētiem nosacījumiem vai prasībām, bet pēc autora domām šīs kopas būs gana labi piemēri, lai gūtu vispārēju priekšstatu par rīku funkcionalitāti.

3.1.1. Pirmā testu kopa

Pirmās testu kopas mērķis ir pārbaudīt dažādas prasības, ar kurām jebkuram rīkam vajadzētu spēt tikt galā, piemēram, korekti identificēt izvēlnes, laukus un pogas, spēt ierakstīt konkrētu tekstu konkrētos laukos, spēt nospiest citas tastatūras pogas (šajā gadījumā taustiņu „Enter"), spēt uzgaidīt, kamēr kāds process tiks pabeigts, kā arī secīgi aizvērt atvērtās formas, tādējādi atbrīvojot vietu nākošo testa kopu darbībām.

Pirmā testu kopa veic personas datu pārņemšanu pēc personas koda. Dati lietojumprogrammā NINO tiek iegūti no citas sistēmas, tādēļ parasti sanāk uzgaidīt, līdz process tiek pabeigts. Kopa sastāv no sekojošām darbībām:

- Jāveic peles klikšķis izvēlnē „Dati", lai tiktu attēlota apakšizvēlne;
- Jāveic peles klikšķis uz izvēlnes „Dati" apakšizvēlni „Personas";
- Personu formā jāidentificē lauks „Kods(VRN)" un tajā jāievada vērtība „161290-12501";
- Jānospiež tastatūras taustiņš „Enter";
- Jāatver personas kartiņa, veicot peles dubultklikšķi uz personas formā atrasto personu;
- Personu formā jānospiež poga „  ", kas atbilst personas datu pārņemšanai pēc personas koda;
- Jāsagaida, kad process tiks pabeigts un jānospiež poga „  " (saglabāt izmaiņas), kad tā parādās personas kartiņā;
- Jāsagaida, kad saglabāšana tiks pabeigta un tad jāaizver personas kartiņa;

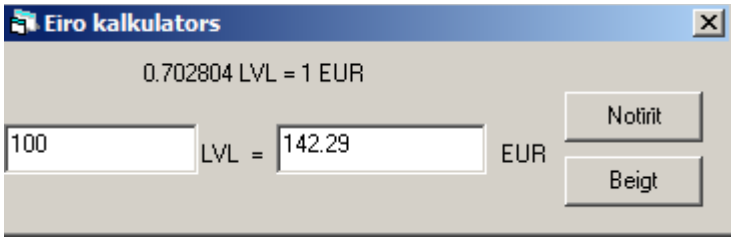
- Jāaizver personu forma.

3.1.2. Otrā testu kopa

Otrās testu kopas mērķis ir noskaidrot, vai rīks spēj pārbaudīt un salīdzināt kāda konkrēta lauka esošo vērtību ar tā paša lauka sagaidāmo vērtību.

Otrā testu kopa izmantos lietojumprogrammā iebūvēto eiro kalkulatoru. Kopa sastāv no šādām darbībām:

- Jāveic peles klikšķis izvēlnē „LVL-EUR", lai atvērtos eiro kalkulators;
- Laukā LVL jāievada vērtība „100";
- Jāpārbauda lauka EUR vērtība. Šai vērtībai jāsakrīt ar skaitli „142.29";
- Jāaizver forma, nospiežot pogu „Beigt".



3.1. att. ***Eiro kalkulators***

3.2. Kritēriji, pēc kuriem pārbaudīti rīki

Rīki pārbaudīti pēc 4 kritēriju kategorijām: funkcionalitāte, lietojamība, rīka atbalstītās vides un izmaksas. Katrā no kritēriju kategorijām ir vairāki kritēriji. Pilns kritēriju saraksts un to paskaidrojumi aplūkojami tabulā 3.1.

Tabula 3.1.

Kritēriji un to paskaidrojumi

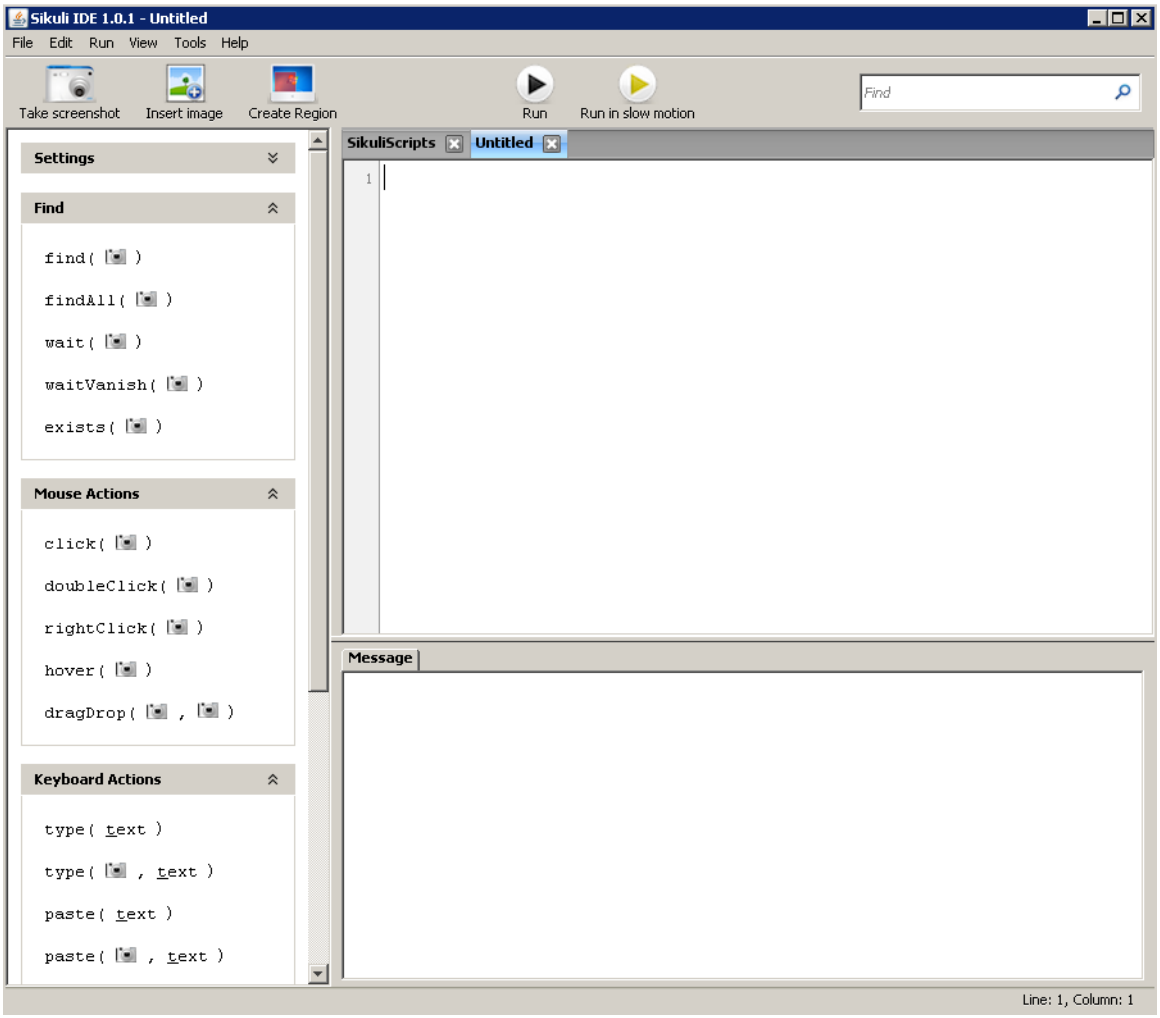
Kritērijs	Paskaidrojums
Funkcionalitāte:	
„Record and playback"	Iespējams ierakstīt darbības un pēc tam šīs darbības atkārtot. Šeit tiks minēti arī ierakstu veidi, ja rīkam tādi ir vairāki.
Skripti	Ir iespējams rakstīt skriptus.
Testēšanas atskaites	Rīks automātiski veido testēšanas atskaites.
Lietojamība:	
Rīka instalācija	Autora viedoklis, cik sarežģīta ir rīka instalācija.
Lietotāja saskarne	Autora viedoklis, cik laba vai slikta ir rīka grafiskā lietotāju

	saskarne, cik ērti ir atrast nepieciešamās pogas un izvēlnes.
Tehniskais atbalsts	Autora viedoklis, cik labs vai slikts ir rīkam pieejamais tehniskais atbalsts.
Testu izveide	Autora viedoklis, cik ērta ir testa kopu izveide.
Testēšanas atskaites	Autora viedoklis, cik saprotama un lasāma ir rīka izveidotā testēšanas atskaite.
Rīka atbalstītās vides:	
Operētājsistēmas	Operētājsistēmas, uz kurām var izmantot rīku.
Programmēšanas valodas	Programmēšanas valodas, kuras var pielietot rīkā.
Tīmekļa lietotnes	Vai rīks atbalsta tīmekļa lietotņu testēšanu (Jā/nē).
Darbvirsmas lietotnes	Vai rīks atbalsta darbvirsmas lietotņu testēšanu (Jā/nē).
Mobilās lietotnes	Vai rīks atbalsta mobilo lietotņu testēšanu (Jā/nē).
Izmaksas:	
Maksas rīks	Vai rīks ir maksas vai bezmaksas.
Cena	Ja rīks ir maksas, tad kāda ir tā cena?
Licences tips	Viena datora licence/peldošā licence.

3.3. Sikuli

Maģistra darba izstrādes laikā rīkam *Sikuli* bija pieejamas 2 versijas. Versija 1.0.1. ir uzskatāma par pārbaudītu un strādājošu versiju, kurai ir pieejams atbalsts no izstrādātājiem. Versija 1.1.0 ir izstrādes stadijā un tai katru nakti tiek veikts būvējums [16] (*nightly build*), kas nozīmē, ka šajā versijā var būt kaut kādas problēmas, kuru dēļ pats rīks vai kāda no rīka komponentēm var nestrādāt. Versiju 1.1.0 paredzēts beigt izstrādāt 2015. gada maijā[17], bet līdz tam laikam maģistra darbs jau būs pabeigts, tātad šīs versijas gala produktu neizdosies pārbaudīt.

Autors sākotnēji mēģināja instalēt versiju 1.1.0, bet rīku neizdevās atvērt, tādēļ tika nolemts strādāt ar stabilāko no versijām – versiju 1.0.1. Ja uz datora ir instalēta salīdzinoši jauna *JAVA* versija (iespējams, ka sākot no 2014. gada, jo 1.0.1. tika pabeigts 2013. gadā), pastāv iespēja, ka pie *system enviroment variables* būs jāpievieno mainīgais *JAVA_HOME*, kurā jānorāda ceļš uz jaunāko *JAVA* versijas *JRE*. Kad tas ir izdarīts, startējot *Sikuli IDE*, tiks izmantota norādītā *JAVA* versija. Startējot *IDE*, tiek atvērts attēlā 3.2. redzamais rīks.

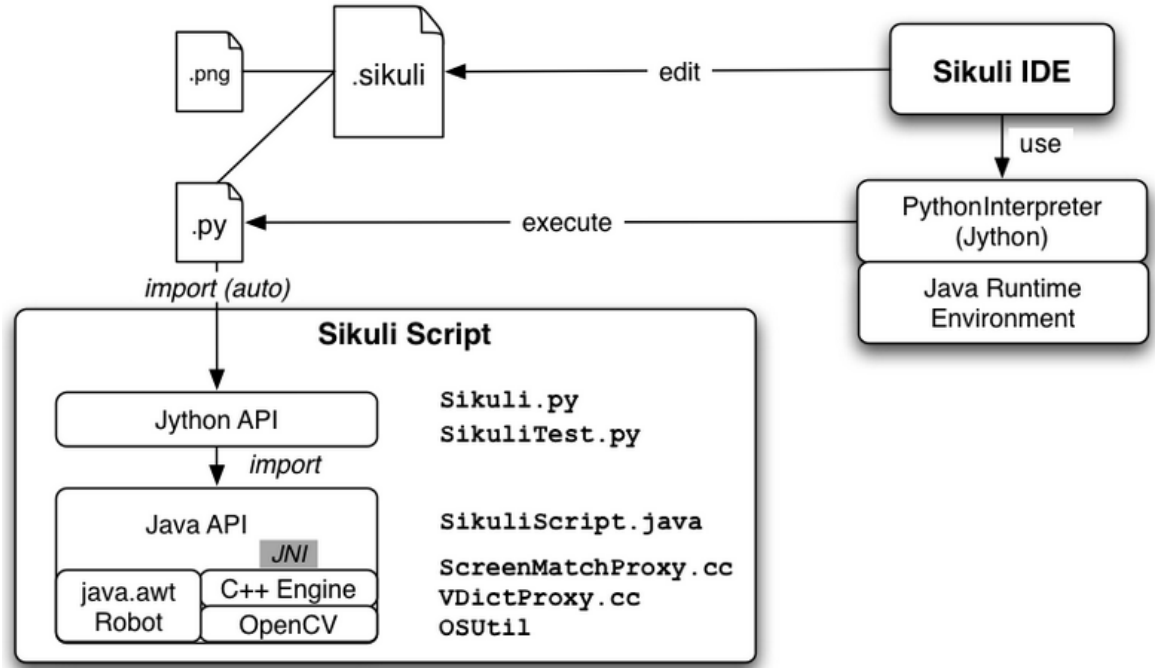


3.2. att. ***Sikuli IDE***

Sikuli kā primāro skriptēšanas valodu atbalsta *Python*, bet ir iespējams izmantot arī *Ruby*. Papildus iespējams izmantot *SikuliX API Java* programmā, pievienojot *sikulixapi.jar* failu *Java* projektā, importējot nepieciešamās bibliotēkas un rakstot kodu.

Teorētiski ir iespējams ieviest arī kādas citas valodas izmantošanu – *Sikuli* mājas lapā minēti tādi kandidāti kā *Beanshell, Scala, JavaScript* – galvenais faktors šiem kandidātiem ir kādas *Java* vai *JVM* bāzētas interpretatora pakotnes pieejamība.

Skatoties uz rīku no tehniskās puses, tas darbojas sekojoši:



3.3. att. **Kā darbojas *Sikuli***

Sikuli IDE izmanto skriptēšanas valodas *Python* interpretatoru (*Jython*) un *JRE,* lai varētu izpildīt *Python* skriptus. *Python* skripti sevī ietver *Sikuli* skriptu un papildus *Python* kodu. *Sikuli* skripti sastāv no divām bibliotēkām – *Jython API* un *Java API*. No *Java API* galvenokārt tiek izmantotas divas daļas – *java.awt.Robot* klase, ar kuras palīdzību var imitēt klaviatūras un peles darbības noteiktās ekrāna vietās, kā arī uz *C++* bāzēto *OpenCV*, ar kura palīdzību uz ekrāna tiek meklēti izvēlētie attēli.

3.3.1. Testa kopu izveidošana ar Sikuli

Tā kā rīkam *Sikuli* nav testa darbību ierakstīšanas un atkārtošanas funkcionalitāte, tad visi testpiemēri ir jākodē pašam. Šī darba ietvaros kodēšana tika veikta skriptēšanas valodā *Python*, kurai papildus tiek pievienota *Sikuli* izmantotā attēlu atpazīšanas tehnoloģija.

Izmantojot *Sikuli[3]* un *Python[18]* dokumentāciju, gala rezultātā tika iegūts 1. pielikumā redzamais kods.

Kā redzams kodā, dažās vietās autors ir norādījis vairākus variantus, kā risināt kādu konkrētu problēmu. Šādā veidā autors parāda, ka vienu lietu var interpretēt dažādi un, kaut arī visi gadījumi sasniedz vēlamo rezultātu, daži gadījumi ir efektīvāki. Kā piemēru var minēt koda rindas 46, 48 un 49.

```
46          wait( Personas , 10)
47 #izmantojot getLastMatch() kopējā izpilde ir ātrāka, jo nav atkārtoti
jāmeklē objekts. Tiek izmantots iepriekš atrastais objekts.
48          object=getLastMatch()
49          click(object)
```

3.4. att. ***Sikuli*** **koda fragments(1)**

46. rindā teikts, ka rīkam jāgaida līdz 10 sekundēm, kamēr parādās poga „Personas". 48. rindā mēs mainīgajam „object" padodam iepriekš atrasto attēlu. 49. rindā mēs izmantojam mainīgo „object" un veicam peles klikšķi uz šo attēlu. Šajā vietā varēja vēlreiz meklēt pogu „Personas", bet šāda veida atkārtota attēlu meklēšana aizņem papildus laiku. Ja šādā veidā atkārtoti meklētu katru testos izmantoto attēlu, tad kopumā skripta izpildes laiks būtu krietni ilgāks.

Līdzīgi varam aplūkot divas definētās funkcijas. Funkcija no 24. līdz 36. rindai ir krietni lēnāka, jo pie katras izpildes iterācijas atkārtoti meklē pogu. Ja atrod divas šādas pogas, tad aizver otro no augšas, tādējādi pašas lietotnes aizvēršanas pogu neaiztiekot.

```
24 def aizvertLogus(reizes):
25     def byY(aizvert):
26         return aizvert.y
27     for x in range(0, reizes):
28         aizvertMasivs = findAll( )
29         sortedIcons = sorted(aizvertMasivs, key=byY)
30         if len(sortedIcons) >= 2: #ja atrod divas aizvēršanas pogas..
31                 click(sortedIcons[1]) #.. tad nospiežam otro aizvēršanas
pogu no augšas
32              #type(Key.F4, KEY_CTRL) #var izmantot click(sortedIcons[1])
vietā
33             aizvertMasivs = []
34             sortedIcons = []
35         else:
36             break
```

3.5. att. ***Sikuli*** **koda fragments(2)**

Vēlāk autors nonāca pie krietni ātrāka un ērtāka risinājuma – izmantojot Windows iestrādāto taustiņu kombināciju Ctrl + F4 var aizvērt visus kāda loga atvasinātos logus jeb tā saucamos „*child windows*".

```
38 #ātrāks veids, kā aizvērt visus lietotnes child logus
39 def aizvertLogus2(reizes):
40     for x in range(0, reizes):
41         type(Key.F4, KEY_CTRL)
```

3.6. att. ***Sikuli* koda fragments(3)**

Tāpat varam salīdzināt otrās testu kopas pārbaudi par to, vai summa 100 LVL eiro kalkulatorā atbilst summai 142.29 EUR. Pirmajā variantā (101. līdz 110. koda rinda) tiek iezīmēts viss lauka EUR saturs pēc tam, kad ir veikta konvertācija un visa šī vērtība tiek nokopēta (ielikta starpliktuvē). Šī starpliktuves vērtība tiek padota mainīgajam „vertiba" un tālāk tiek izpildīta vienkārša pārbaude, vai mainīgais „vertiba" atbilst summai 142.29. Otrs variants (111. līdz 115. koda rinda) ir pa visu ekrānu atrast attēlu, kurā ir skaitlis 142.29. Šajā gadījumā abi risinājumi izpildās vienlīdz ātri, jo abos risinājumos tiek meklēts viens attēls.

Ja aplūkojam to, kā rīkā ir atveidotas sadaļā 3.1. minētās testu kopas, ir redzams, ka pirmā testu kopa ir sadalīta 3 testos. Autors to darīja tādēļ, ka uzskatīja, ka šādā veidā testu atskaitē tiks labāk attēlota visa testu kopa. Jāņem vērā, ka rīks pēc noklusējuma neveido testēšanas atskaites, tādēļ tās nācās integrēt pašam. Kā risinājums tika izmantots *HTMLTestRunner*[19].

Testēšanas atskaitēs pēc noklusējuma ekrānuzņēmumi tika uzņemti tikai pie kļūmēm, bet autors pievienoja kodu, lai ekrānuzņēmumi tiktu izveidoti arī pie kļūdām un arī tajos gadījumos, kad viss strādā korekti. Atskaite redzama attēlā 3.7.

Atskaite

Starta laiks: 2015-05-12 13:37:45
Testu ilgums: 0:00:22.711000
Statuss: Pass 4

NINO testi

Attelot: Kopsavilkums Neizdevusies Visi

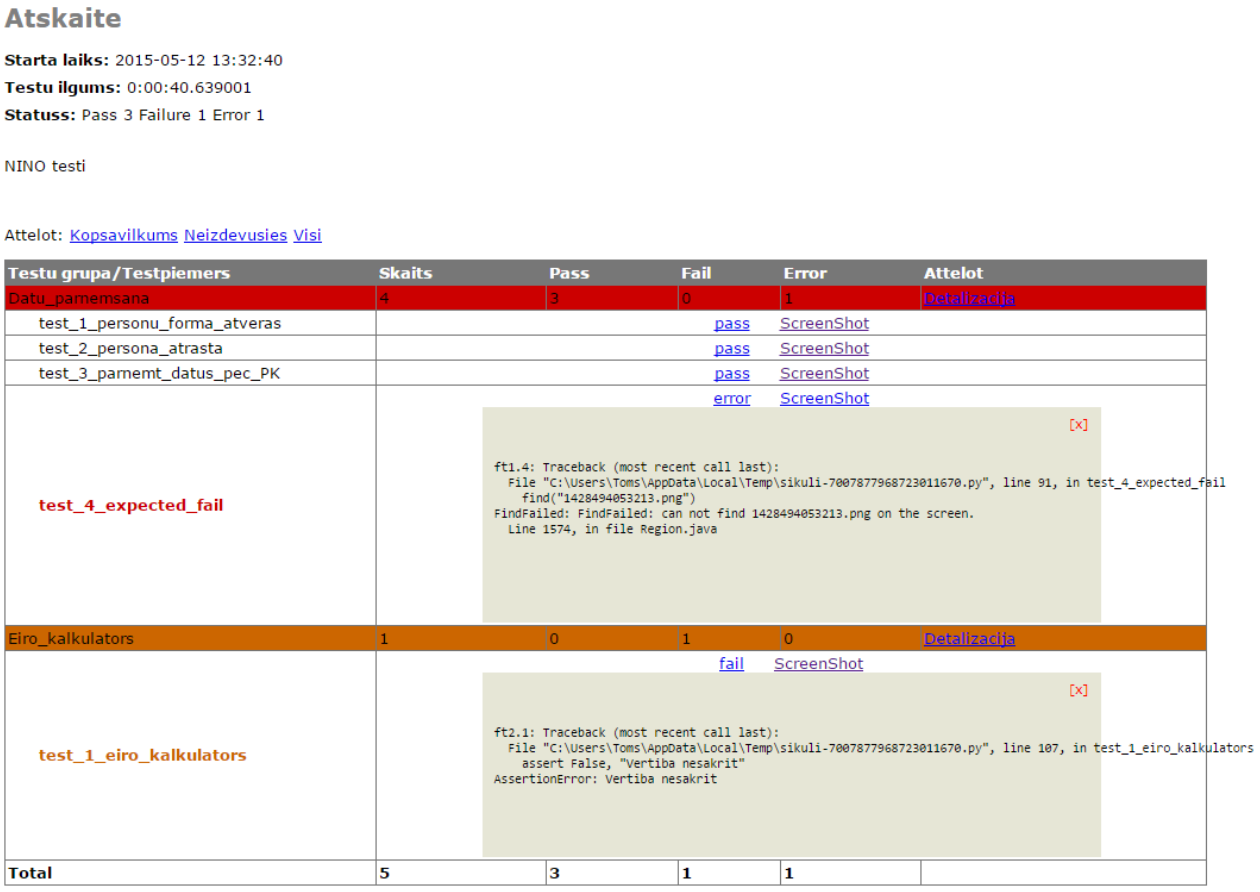
Testu grupa/Testpiemers	Skaits	Pass	Fail	Error	Attelot
Datu_parnemsana	3	3	0	0	Detalizacija
test_1_personu_forma_atveras			pass	ScreenShot	
test_2_persona_atrasta			pass	ScreenShot	
test_3_parnemt_datus_pec_PK			pass	ScreenShot	
Eiro_kalkulators	1	1	0	0	Detalizacija
test_1_eiro_kalkulators			pass	ScreenShot	
Total	**4**	**4**	**0**	**0**	

3.7. att. ***HTMLTestRunner* veidotā testēšanas atskaite**

Virs atskaites tabulas ir pieejami hiperteksti „Kopsavilkums", „Neizdevusies" un „Visi", ar kuru palīdzību var atlasīt nepieciešamo informāciju.

- Hiperteksts „Kopsavilkums” attēlo testu kopas, kā arī to, cik testpiemēri ir izdevušies, cik gadījumos bijušas kļūmes, cik gadījumos bijušas kļūdas. Šajā skatā pēc noklusējuma netiek detalizēti attēlota informācija par katru no testpiemēriem (nav nospiesti hiperteksti „pass”, „ScreenShot”, kā tas redzams attēlā 3.7.);
- Hiperteksts „Neizdevusies” attēlo tikai tos ierakstus, kuriem ir vai nu kļūda vai kļūme;
- Hiperteksts „Visi” attēlo visus ierakstus detalizētā līmenī, tieši tāpat, kā tas ir redzams attēlā 3.8.

Lai parādītu, kā izskatās atskaite, kad ir notikusi kāda kļūda vai kļūme, autors ir atkomentējis 90. un 91. rindā redzamo testpiemēru, kā arī 2. testu kopas testpiemēram norādījis, ka starpliktuves vērtība ir jāsalīdzina ar summu 142.39, kas, protams, nebūs pareizi, jo pareizā summa bija 142.29. Šāda atskaite redzama attēlā 3.8.



3.8. att. ***HTMLTestRunner*** **veidotā testēšanas atskaite, kurā ir kļūdas un kļūmes**

Kā redzams attēlā 3.8., nospiežot uz hipertekstiem „error” vai „fail”, ir iespējams apskatīt kļūdas vai kļūmes cēloni, kā arī nepieciešamības gadījumā var nospiest uz hipertekstu

„ScreenShot", tādā veidā atverot ekrānuzņēmumu, kurā būs redzams ekrāns, kāds tas izskatījās brīdī, kad notika kļūda vai kļūme.

3.3.2. Autora atsauksmes par Sikuli un rīka novērtējums

Sikuli ir konkurētspējīgs rīks, ar kuru var strādāt gan programmētāji, gan cilvēki bez profesionālām programmēšanas prasmēm. Autors uzskata, ka pateicoties attēlu atpazīšanas tehnoloģijai, ir interesanti ar šo rīku eksperimentēt, bet tai pašā laikā tiek apgūtas vai padziļinātas programmēšanas prasmes un sasniegts rezultāts. Rīku ir viegli integrēt ar citiem rīkiem, kā arī rīkam ir pieejamas dažādas bibliotēkas, kas atvieglo darbu ar šo rīku. Rīkam veltītajā jautājumu un atbilžu forumā[20] ir pieejama plaša informācija par visdažādākajiem ar rīku saistītajiem jautājumiem, uz kuriem regulāri atbild pats rīka izstrādātājs. Tā kā rīks ir atvērta koda, tad šī rīka dažādu risinājumu izstrādē iesaistās dažādas personas. Priekš *Sikuli* ir izveidoti dažādi satvari [16] (angliski *framework*), kuri atvieglo grafiskās lietotāju saskarnes automatizēšanu. Kā piemēru var minēt *SikuliFramework*[21].

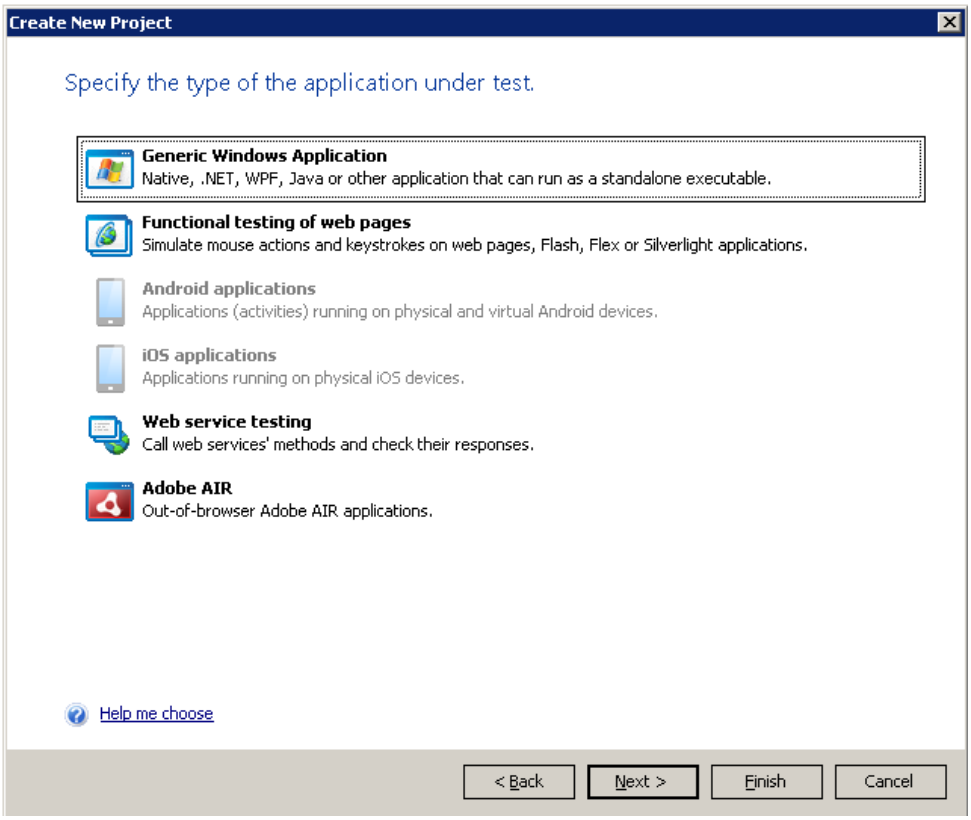
3.4. TestComplete

3.4.1. TestComplete projekta izveidošana un lietotnes piesaistīšana

Rīka sākuma lapā ir atrodami noderīgi padomi, kā izveidot pirmo projektu un pirmos testus. Citu neskaidrību gadījumā ir iespēja caur pašu rīku piekļūt forumam, kurā var uzdot sev interesējošus jautājumus par rīka funkcionalitāti. Pēc autora novērojumiem, uz jautājumiem šajā forumā tiek saņemtas atbildes pietiekami ātri.

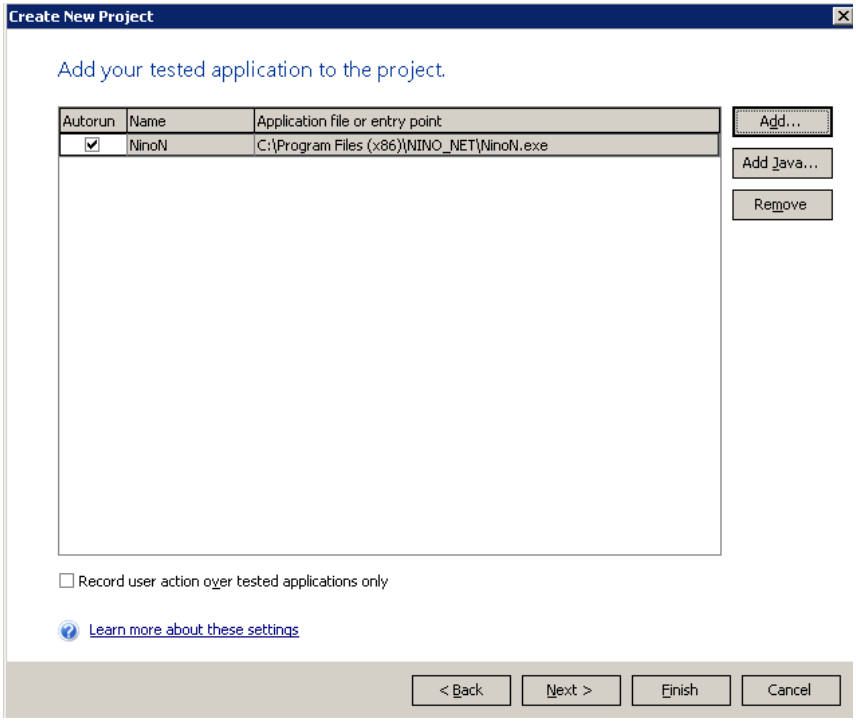
Bez jebkādas dokumentācijas lasīšanas vai pamācību skatīšanās, autoram neradās nekādas problēmas atrast sākotnēji nepieciešamās izvēlnes, lai izveidotu jaunu projektu un projektam piesaistītu testējamo lietotni.

Veidojot jaunu projektu, lietotājam ir jānorāda, kāda veida lietotne tiks testēta. Autora gadījumā bija jāizvēlas opcija *Generic Windows Application* (attēls 3.9.).



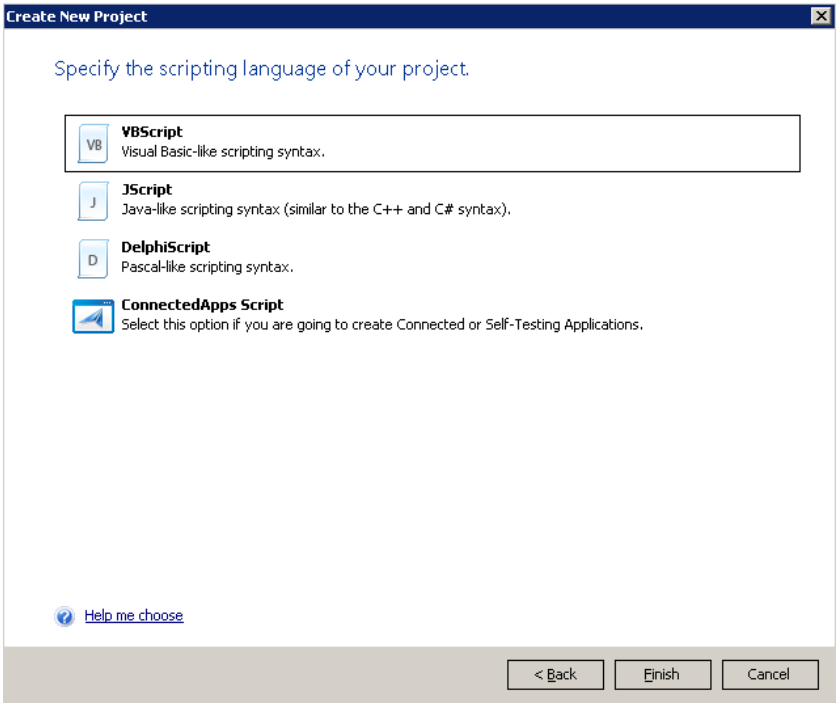
3.9. att. **Projekta lietotnes veida norādīšana**

Kad tas tika izdarīts, nācās norādīt ceļu uz vietu, kur lietotne glabājas uz cietā diska. Ja lietotne ir izstrādāta programmēšanas valodā *JAVA*, tad jāspiež poga „Add Java" un tur jānorāda ceļš uz lietotnes *JAR* faila atrašanās vietu uz cietā diska, bet, tā kā VPS lietojumprogramma NINO nav izstrādāta programmēšanas valodā *JAVA*, tad šī lietotne bija jāpievieno ar pogu „Add" (attēls 3.10.).



3.10. att. **Lietotnes pievienošana projektam**

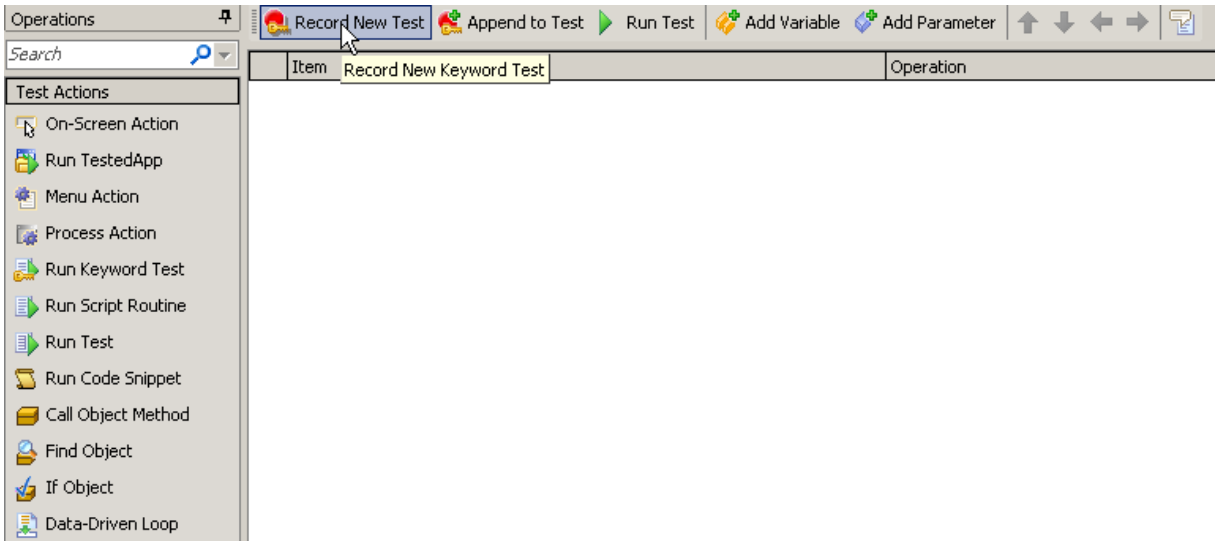
Tālāk jānorāda, kāda skriptēšanas valoda tiks izmantota projektā. No piedāvātajām skriptēšanas valodām autoram ir bijusi pieredze ar *VBScript*, tādēļ tika izvēlēta šī valoda.



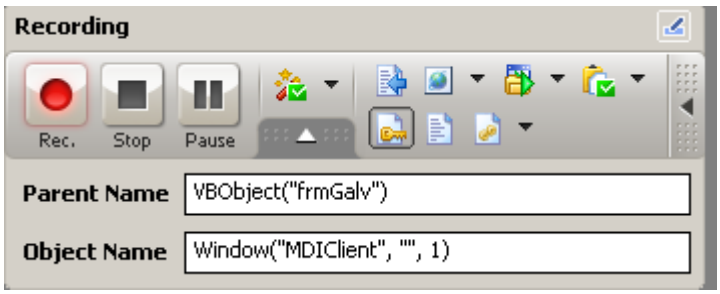
3.11. att. **Projekta skriptēšanas valodas norādīšana**

3.4.2. Testa kopu izveidošana ar TestComplete

Kad projekts ir izveidots un lietotne piesaistīta, ir nepieciešams izveidot testpiemērus. Tas ir viegli izdarāms, izmantojot darba teorētiskajā daļā minēto atslēgvārdu testu. Vispirms jānospiež attēlā 3.12. redzamā poga, un pēc tam parādīsies attēlā 3.13. redzamais logs.



3.12. att. **Jauna atslēgvārdu testa ierakstīšana**

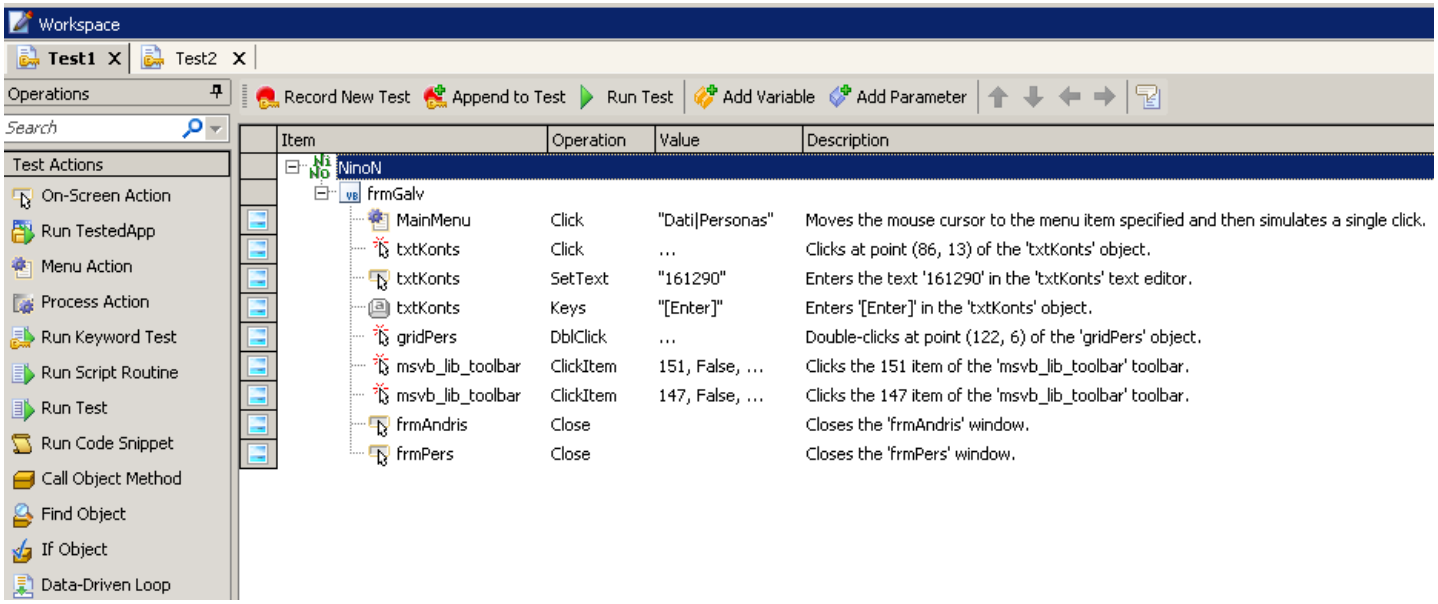


3.13. att. **Jauna testa ierakstīšanas logs**

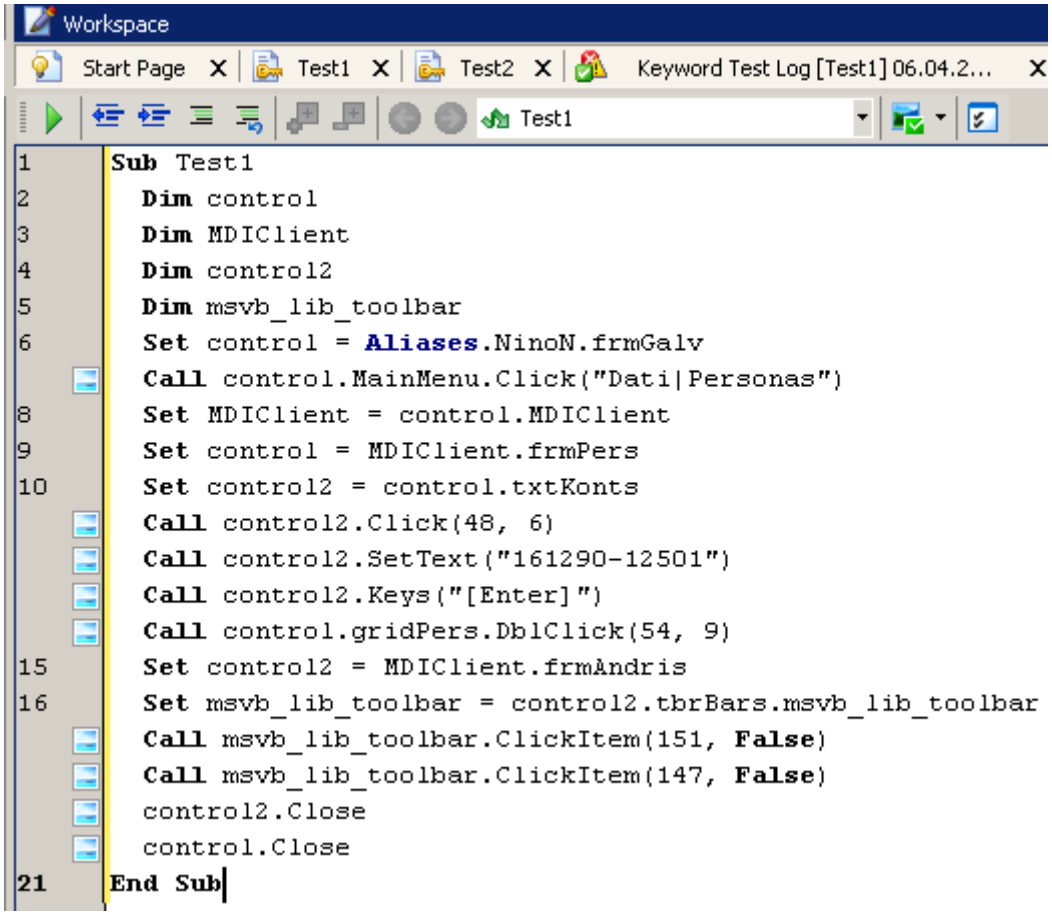
Attēlā 3.13. ir redzams ierakstīšanas logs paplašinātā režīmā, kas nozīmē, ka šajā logā, atšķirībā no attēlā 2.1. redzamā loga, ir izmantojama papildus funkcionalitāte. Kā būtiskāko papildus funkcionalitāti autors min iespēju aplūkot to, kā lietotnē ir nosaukts konkrēts objekts, uz kura tajā brīdī lietotājs ir uzvirzījis peles kursoru. Tas varētu būt noderīgi tajos gadījumos, kad jāpārliecinās, ka testēšanas rīks tik tiešām pareizi spēj identificēt konkrētu objektu un tā vecākobjektu (no angļu val. *parent*[16]).

Papildus tam, caur ierakstīšanas logu testam var pievienot komentārus, automātiski startēt piesaistīto lietotni, ja tā nebija startēta pirms testa ierakstīšanas uzsākšanas, kā arī pārliecināties par starpliktuves saturu, pievienojot testam starpliktuves kontrolpunktu (*clipboard checkpoint*).

Tālāk tika ierakstīts pirmais tests, kas atbilst sadaļas 3.1. pirmajai testu kopai. Šis tests tika ierakstīts gan kā atslēgvārdu tests (attēls 3.14.), gan kā skripts (attēls 3.15.).



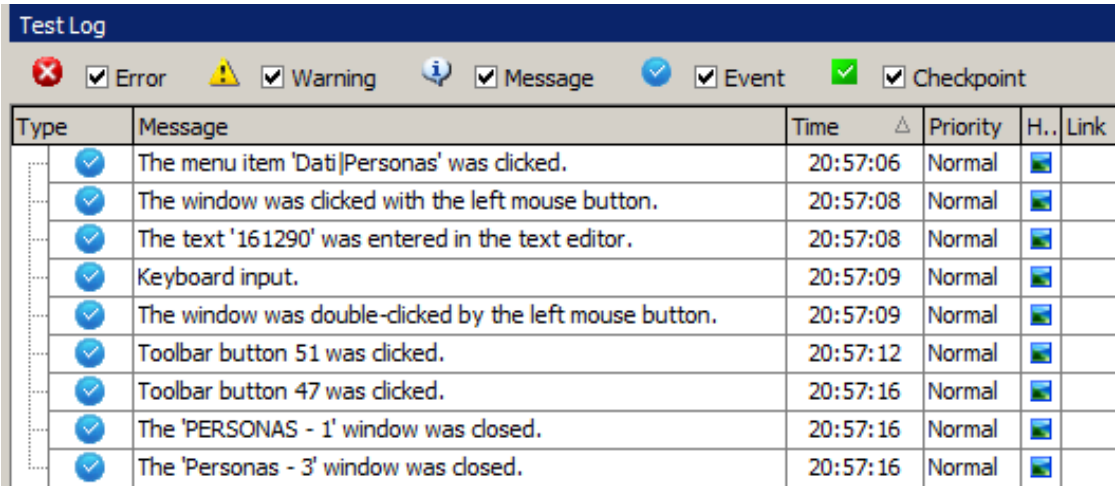
3.14. att. **Pirmās testu kopas ierakstītais atslēgvārdu tests**



3.15. att. **Pirmās testu kopas ierakstītais skripts**

Ierakstītais skripts ir *VBScript* valodā, jo šādu valodu autors norādīja projekta izveides laikā. Gan atslēgvārdu testā, gan skriptā pie katras darbības ir pievienots attēls, kurš tika iegūts darbību ierakstīšanas laikā.

Pēc tam palaižot ierakstīto atslēgvārdu testu, rīks veic testā norādītās darbības un pēc darbību veikšanas izveido testēšanas atskaiti.

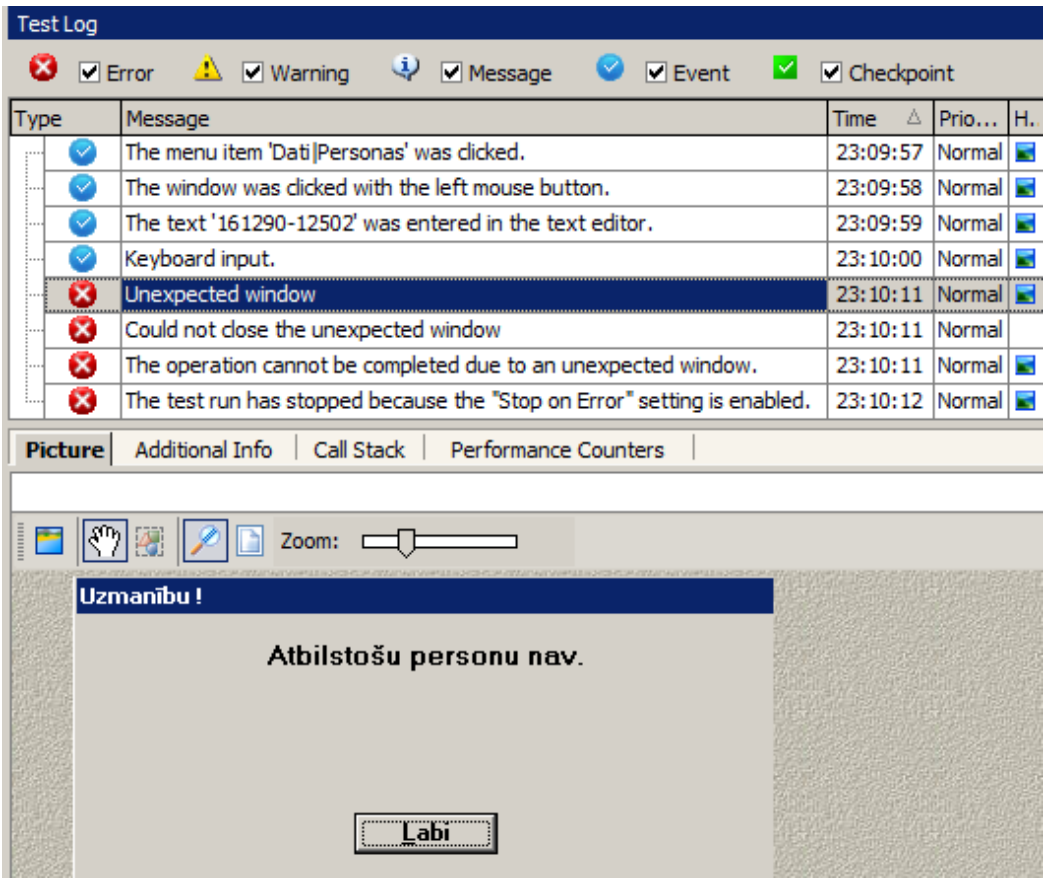


Type	Message	Time	Priority	H..	Link
	The menu item 'Dati|Personas' was clicked.	20:57:06	Normal		
	The window was clicked with the left mouse button.	20:57:08	Normal		
	The text '161290' was entered in the text editor.	20:57:08	Normal		
	Keyboard input.	20:57:09	Normal		
	The window was double-clicked by the left mouse button.	20:57:09	Normal		
	Toolbar button 51 was clicked.	20:57:12	Normal		
	Toolbar button 47 was clicked.	20:57:16	Normal		
	The 'PERSONAS - 1' window was closed.	20:57:16	Normal		
	The 'Personas - 3' window was closed.	20:57:16	Normal		

3.16. att. **Atslēgvārdu testa atskaite**

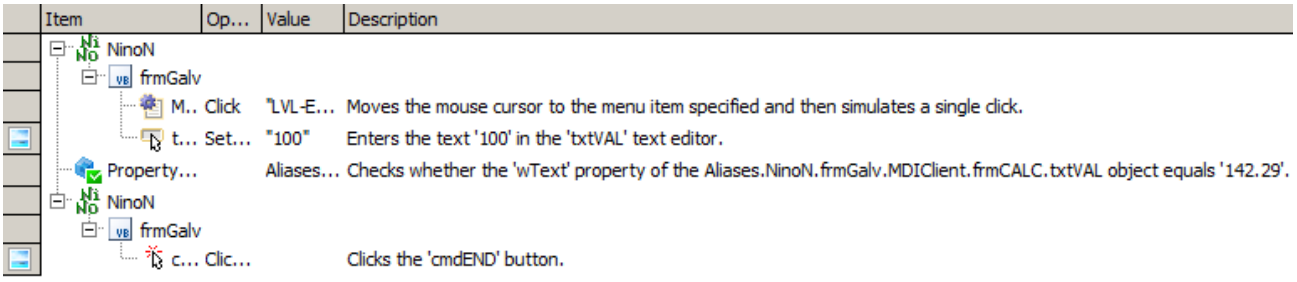
Arī pēc skripta palaišanas tika iegūta tieši tāda pati atskaite, kas nozīmē, ka rīks prot korekti ierakstīt un izpildīt gan atslēgvārdu testus, gan skriptus. Papildus tika salīdzināti izpildes laiki un tika konstatēts, ka skripts ar vienādu darbību skaitu izpildījās par aptuveni 3 sekundēm ātrāk nekā atslēgvārdu tests. Vidēji atslēgvārdu tests izpildījās 11 sekundēs, skripts izpildījās 8 sekundēs.

Kopumā šī rīka testēšanas atskaites ir viegli lasāmas un saprotamas. Pie katra testa soļa ir iespējams aplūkot divu veidu attēlus, kur viens ir sagaidāmais rezultāts (attēls, kas tika izveidots testa darbību ierakstīšanas laikā) un otrs ir esošais rezultāts (attēls, kas tika iegūts testa darbību izpildes laikā). Tas noder tajās situācijās, kad kāds no testēšanas soļiem nav izdevies un ir jāsaprot, kur tieši ir bijusi problēma. Piemēram, attēlā 3.17. redzams, ka testa solis nav izpildījies un rīks ir automātiski uzņēmis ekrānuzņēmumu ar kļūdas situāciju. Arī pašu testa soļu ziņojumi ir viegli lasāmi. Ņemot par piemēru šo pašu gadījumu, ir skaidri redzams, ka lietotnē parādījās neparedzēts logs, ar kuru rīkam neizdevās tikt galā, jo šādas situācijas apstrāde nebija izveidota. Korekti šādā situācijā būtu bijis jāveido *if…then* nosacījums, bet tā kā tas nozīmētu, ka būtu nepieciešams pārveidot iepriekš definētās testu kopas, tad šī darba ietvaros šādas apstrādes netika izveidotas.



3.17. att. **Neizdevies testa solis**

Pēc tam tika ierakstītas otrās testu kopas darbības. Arī šīs darbības tika saglabātas gan atslēgvārdu testa formātā (attēls 3.18.), gan skripta formātā (attēls 3.19.).

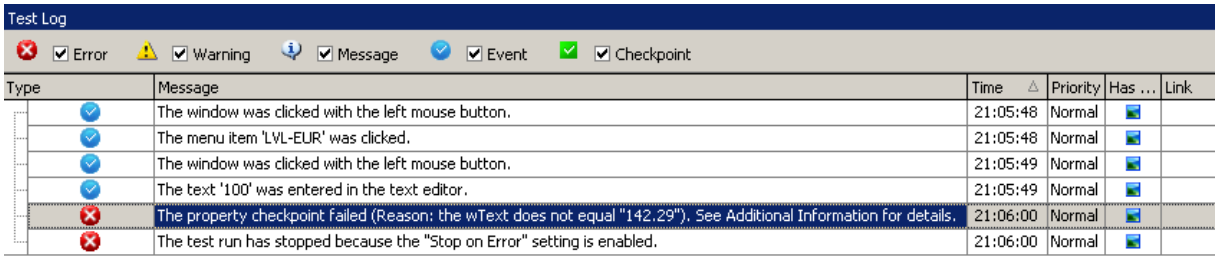


3.18. att. **Otrās testu kopas ierakstītais atslēgvārdu tests**

```
23  Sub Test2
24    Dim control
25    Dim control2
      Call Aliases.explorer.wndShell_TrayWnd.ReBarWindow32.MSTaskSwWClass.MSTaskListWClass.Click(224, 28)
27    Set control = Aliases.NinoN.frmGalv
28    Call control.MainMenu.Click("LVL-EUR")
29    Set control2 = control.MDIClient.frmCALC
30    Set control = control2.txtVAL
      Call control.Click(47, 9)
      Call control.SetText("100")
33    Call aqObject.CheckProperty(Aliases.NinoN.frmGalv.MDIClient.frmCALC.txtVAL, "wText", cmpEqual, "142.29")
      control2.cmdEND.ClickButton
35  End Sub
```

3.19. att. **Otrās testu kopas ierakstītais skripts**

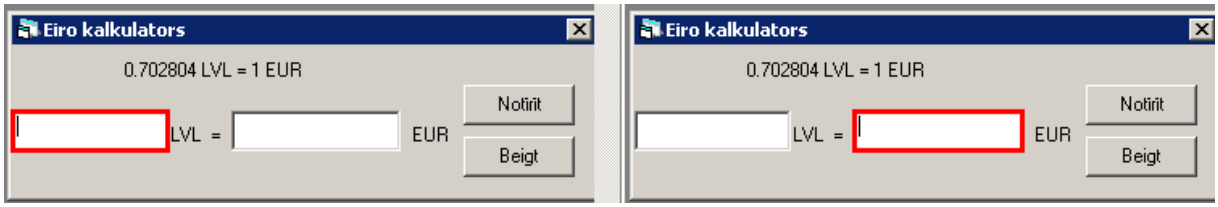
Pēc veicamo darbību skaita, otrā testu kopa ir daudz vienkāršāka, tādēļ pēc loģikas liktos, ka šim testam nekādas problēmas nevajadzētu radīt, jo ar pirmo testu kopu rīks tika galā veiksmīgi. Praktiski, kaut kāda iemesla dēļ rīks nespēja korekti identificēt, kurā laukā jāievada vērtība „100” un gan atslēgvārdu testa gadījumā, gan skripta gadījumā nespēja korekti izpildīt šo testu (attēls 3.20.).



3.20. att. **Otrā testu kopa nav izpildījusies**

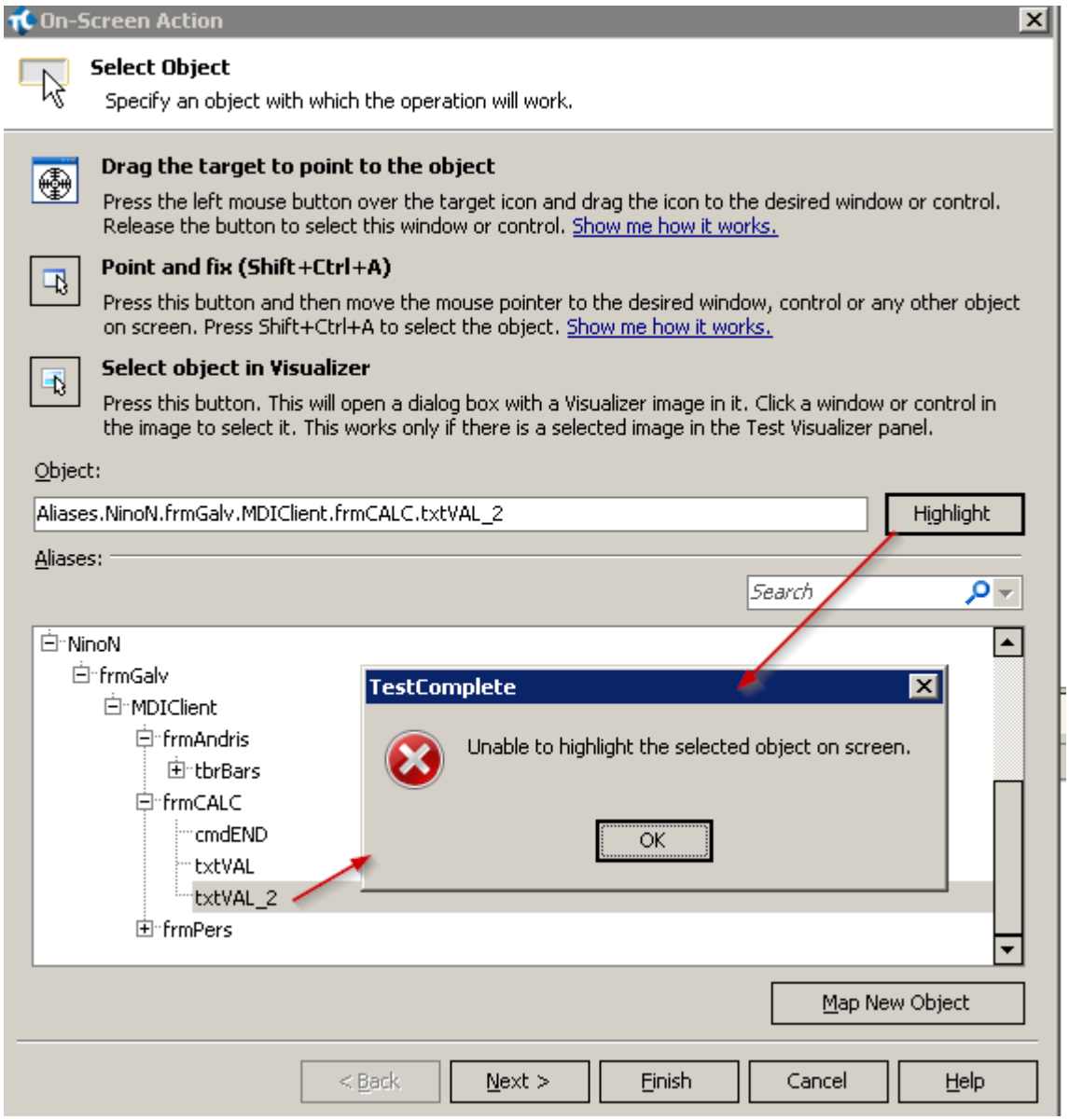
Cenšoties noskaidrot problēmas cēloni, autors sākumā pārbaudīja, vai testu ierakstītājs korekti atpazīst mainīgos, kuri tiek izmantoti testu kopā. Rīks korekti identificēja atšķirīgos teksta laukus un attēloja, ka LVL lauks ir *txtVAL_2* un EUR lauks ir *txtVAL*.

Pēc tam aplūkojot testa atskaitei pievienotos attēlus (attēls 3.21.), ir redzams, ka rīks sagaidāmā rezultāta attēlā (attēls pa kreisi) saprot, ka peles klikšķis ir jāizdara un vērtība jāievada LVL laukā, bet esošā rezultāta attēlā (attēls pa labi) ir redzams, ka peles klikšķis tiek izdarīts un vērtība ievadīta EUR laukā.



3.21. att. **Otrās testu kopas sagaidāmā un esošā rezultāta attēli**

Papildus fakts, ar ko autors saskārās, cenšoties konstatēt problēmas cēloni, ir rīka iezīmēšanas (*highlight*) funkcionalitāte. Šī funkcionalitāte strādā sekojoši – uzspiežot uz kādu no identificētajiem mainīgajiem un nospiežot pogu „Highlight”, lietotnē tiek parādīts, kur šis objekts atrodas. Autoram dīvaini šķiet, ka rīks spēja identificēt, ka šāds objekts eksistē (jo tika iegūts objekta nosaukums), bet tai pašā laikā nespēja objektu attēlot formā, izmantojot pogu „Highlight”. Šī situācija redzama attēlā 3.22.



3.22. att. **Rīks nespēj attēlot objektu *txtVAL_2***

3.4.3. Autora atsauksmes par TestComplete un rīka novērtējums

TestComplete no lietojamības viedokļa ir ērts un viegli saprotams rīks. Automātiski izveidotie atslēgvārdu testi un skripti ir viegli lasāmi un rediģējami. Rīka izveidotās testēšanas atskaites ir informatīvas un no lietojamības viedokļa šīs atskaites ir vienas no labākajām starp visiem apskatītajiem rīkiem.

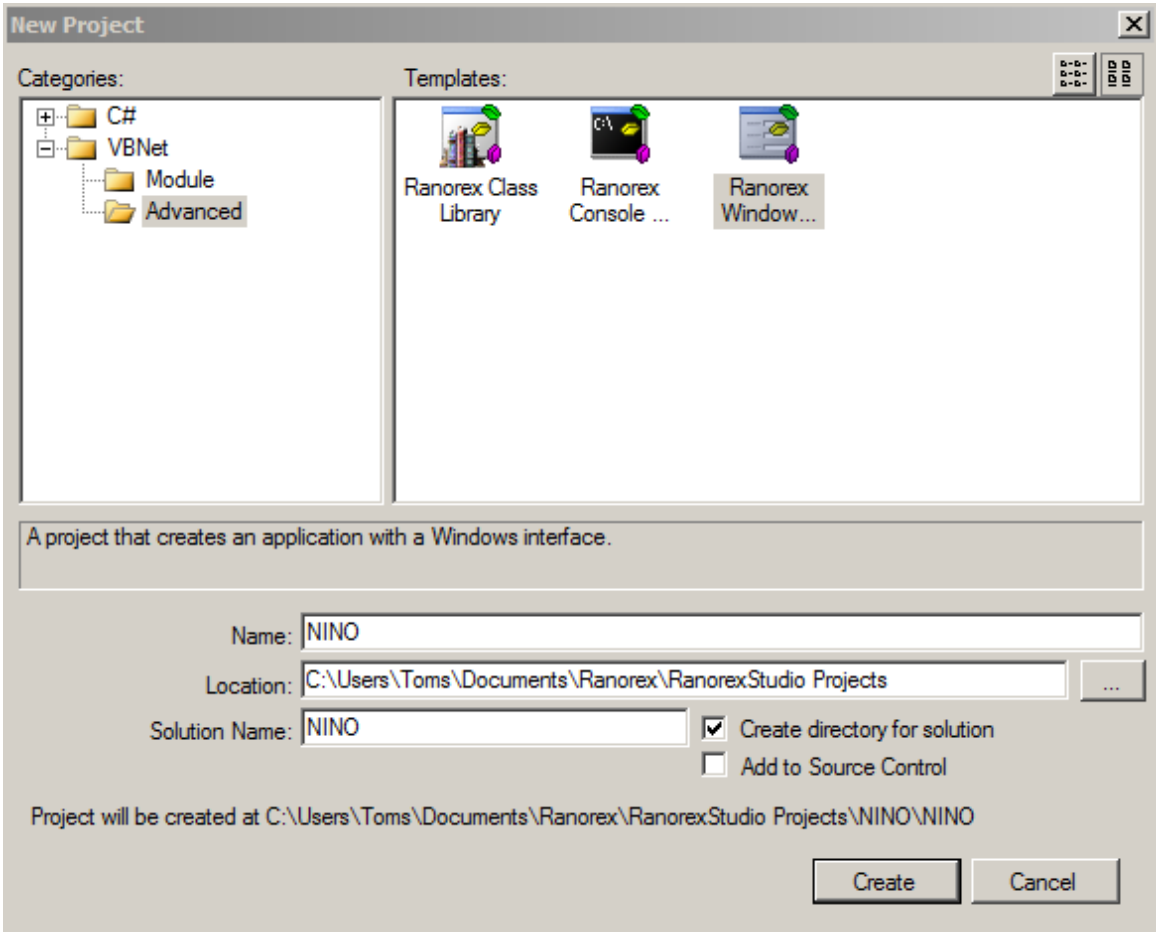
Atskaitot neveiksmi ar otro testa kopu, autors tomēr uzskata, ka objektu atpazīšana rīkā ir labā līmenī. Salīdzinot ar citiem rīkiem, pirmā testu kopa tika ierakstīta un izpildīta bez problēmām. Autors pamēģināja savādāku metodi, kā realizēt otro testu kopu, un tā izdevās. Otrā testa kopa tika ierakstīta daļēji kā atslēgvārdu tests, daļēji kā zema līmeņa procedūra. Tas nozīmē, ka rīks identificēja eiro kalkulatora formu un pēc tam veica darbības pēc formā

esošajām koordinātām. Šajā gadījumā šī metodei nostrādāja, jo eiro kalkulatora formai ir noteikts izmērs, kuru nevar mainīt. Kopumā par rīku rodas iespaids, ka ar to varētu tikt galā ar vairumu situāciju.

3.5. Ranorex

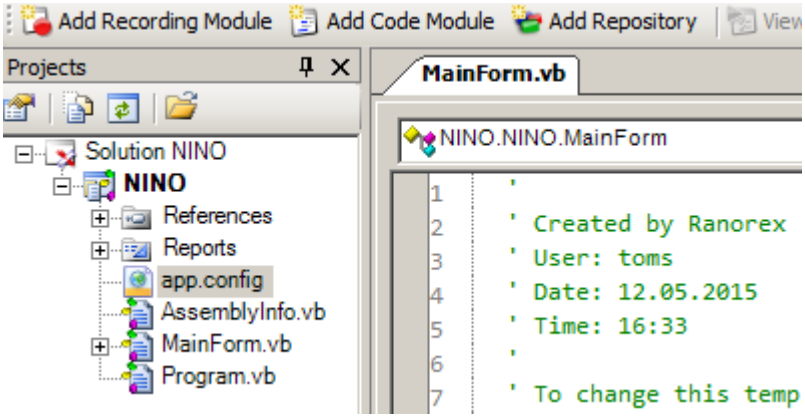
3.5.1. Ranorex risinājuma izveide un lietotnes piesaistīšana

Izmantojot izvēlnes opcijas „File" -> „New" -> „Solution", tiek atvērta attēlā 3.23. redzamā forma. Šajā formā jāizvēlas *Ranorex Windows Application.*



3.23. att. **Jauna Ranorex projekta izveide**

Kad projekts ir izveidots, tiek uzģenerēts risinājums (*solution*) „NINO", kurā izveidoti daži *Visual Basic* faili, kurus nepieciešamības gadījumā var rediģēt manuāli. Risinājums redzams attēlā 3.24.



3.24. att. **Risinājums NINO**

Nospiežot pogu „Add Recording Module", tiek izveidots *Ranorex* repozitorija fails un *Ranorex* ieraksta fails. *Ranorex* ieraksta failā var veikt ierakstu, nospiežot pogu „Record". Nospiežot pogu „Record", parādās logs, kurā jāizvēlas, kāda veida tehnoloģija tiks ierakstīta. Iespējamie ierakstu veidi ir:

- „Desktop" – ieraksta darbvirsmas lietotnē veiktās darbības. Izvēloties šo opciju, jānorāda ceļš uz lietotni, kura tiks testēta;
- „Web" – ieraksta tīmekļa pārlūkprogrammā veiktās darbības. Izvēloties šo opciju, jānorāda tīmekļa pārlūkprogramma un mājas lapa, kurā tiks veikta testēšana;
- „Mobile" – ieraksta mobilajā ierīcē veiktās darbības. Izvēloties šo opciju, ir iespējams pievienot gan mobilo ierīci, gan lietotni, kura tiks testēta;
- „Instant recording" – izvēloties šo opciju, var sākt ierakstīt testēšanas darbības jebkurai lietotnei vai mājas lapai. Atšķirībā no iepriekšējām opcijām, kuras automātiski atver norādīto lietotni vai tīmekļa pārlūkprogrammu/mājas lapu, šīs opcijas nosacījums ir ka testējamā vide ir jau atvērta. Tā netiek atvērta automātiski.

Šī darba izpildei bija jāizvēlas opcija „Dekstop" un jānorāda *NinoN.exe* atrašanās vieta uz cietā diska.

3.5.2. Testa kopu izveidošana ar Ranorex

Tad, kad projektam ir piesaistīta lietotne un atkārtoti nospiesta poga „Record", tiek sākta veikto darbību ierakstīšana. Nospiežot pogu „Validate", tiek uzsākta kādas darbības pārbaude. Poga „Validate" izmantota 2. testu kopā, lai pārbaudītu konvertēto EUR valūtu. Poga „Pause/Continue" veic ierakstītāja nopauzēšanas un ieraksta atsākšanas funkcionalitāti.

Poga „Stop” pabeidz ierakstu un veiktās darbības nosūta uz rīku *Ranorex*. Atzīmējot slēdzi „Image based”, ir iespējams ierakstīt uz attēliem balstītas testa darbības. Šī funkcionalitāte ir noderīga tajos gadījumos, kad no objekta nav iespējams iegūt nepieciešamo informāciju, lai to varētu korekti identificēt. Atzīmējot slēdzi „Enable Hotkeys”, ieraksta laikā ir iespējams izmantot karstos taustiņus, lai veiktu konkrētas darbības. Piemēram, ieraksta laikā nospiežot pogu ‘V’, tiks veikta tāda pati funkcionalitāte, kādu veic ar pogu „Validate”. *Ranorex* veikto darbību ierakstītājs ar atzīmētu slēdzi „Enable Hotkeys” redzams attēlā 3.25.

3.25. att. **Ranorex veikto darbību ierakstītājs**

Pēc pirmās testu kopas darbību ierakstīšanas, tika iegūts attēlā 3.26. redzamais rezultāts. Pēc otrās testu kopas darbību ierakstīšanas, tika iegūts attēlā 3.27. redzamais rezultāts.

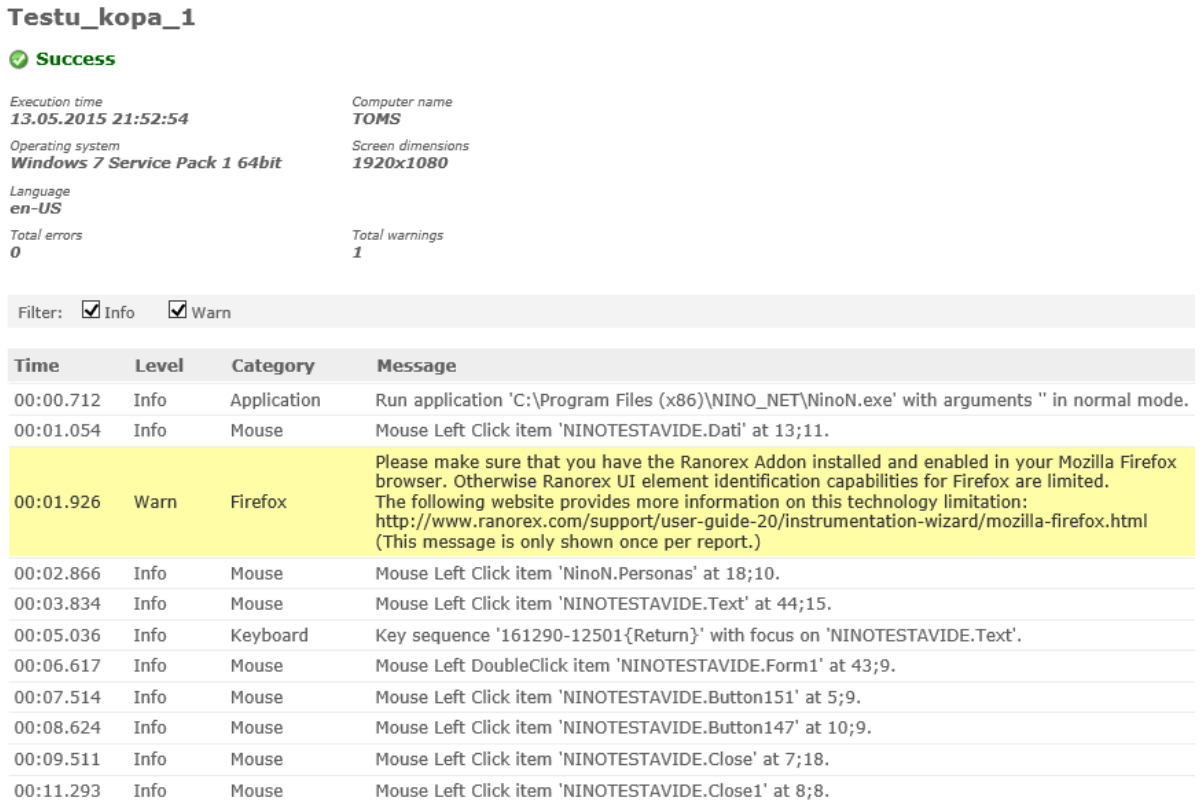
#	Action					Comment
1	Run Application	C:\Program Files (x...		C:\Program Files (x86)...		
2	Mouse	Click	Left	13;11	Dati	
3	Mouse	Click	Left	18;10	Personas	
4	Mouse	Click	Left	44;15	Text	
5	Key Sequence	161290-12501{Return}			Text	
6	Mouse	DoubleClick	Left	43;9	Form1	
7	Mouse	Click	Left	5;9	Button151	
8	Mouse	Click	Left	10;9	Button147	
9	Mouse	Click	Left	7;18	Close	
10	Mouse	Click	Left	8;8	Close1	

3.26. att. **Pirmās testu kopas ierakstītās darbības**

#	Action					Comment
1	Mouse	Click	Left	10;6	LVLEUR	
2	Mouse	Click	Left	60;8	Text	
3	Key Sequence	100			Text	
4	Validate	AttributeEqual	Text	142.29	Text1	
5	Mouse	Click	Left	10;5	Close2	

3.27. att. **Otrās testu kopas ierakstītās darbības**

Pirmās testu kopas testēšanas atskaite redzama attēlā 3.28. Otrās testu kopas testēšanas atskaite redzama attēlā 3.29.



Testu_kopa_1

Success

Execution time
13.05.2015 21:52:54

Computer name
TOMS

Operating system
Windows 7 Service Pack 1 64bit

Screen dimensions
1920x1080

Language
en-US

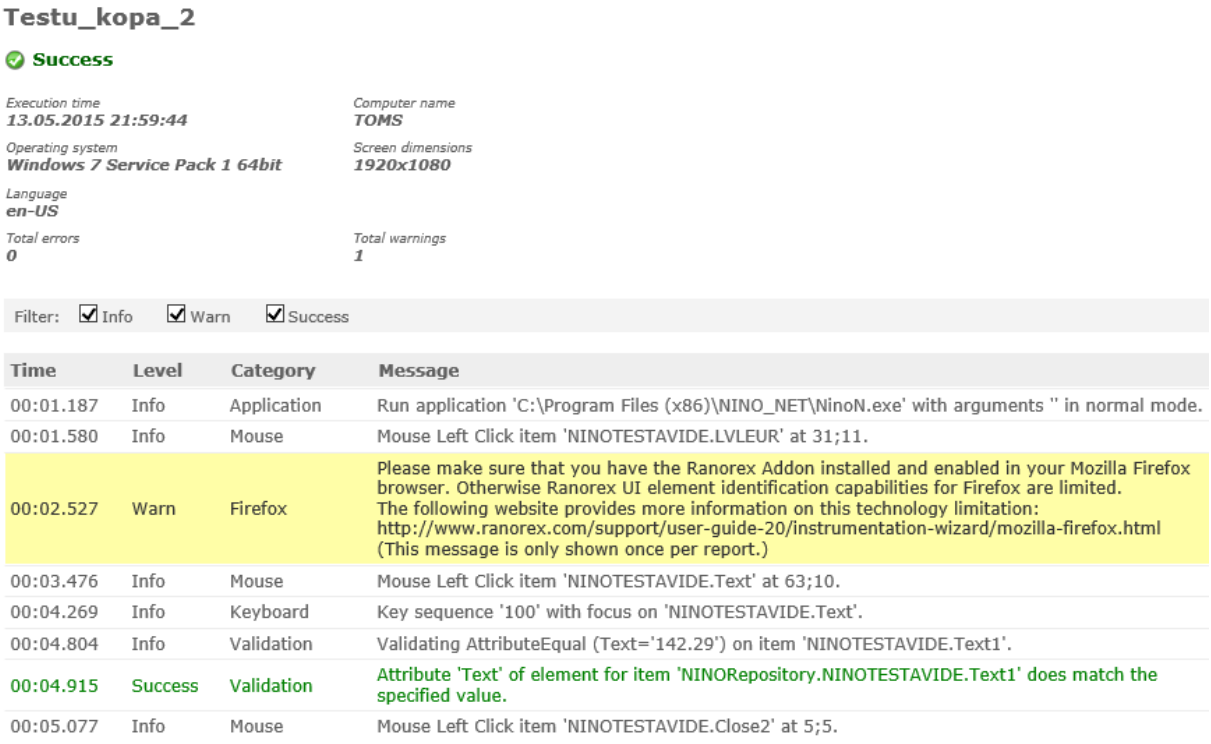
Total errors
0

Total warnings
1

Filter: ☑ Info ☑ Warn

Time	Level	Category	Message
00:00.712	Info	Application	Run application 'C:\Program Files (x86)\NINO_NET\NinoN.exe' with arguments '' in normal mode.
00:01.054	Info	Mouse	Mouse Left Click item 'NINOTESTAVIDE.Dati' at 13;11.
00:01.926	Warn	Firefox	Please make sure that you have the Ranorex Addon installed and enabled in your Mozilla Firefox browser. Otherwise Ranorex UI element identification capabilities for Firefox are limited. The following website provides more information on this technology limitation: http://www.ranorex.com/support/user-guide-20/instrumentation-wizard/mozilla-firefox.html (This message is only shown once per report.)
00:02.866	Info	Mouse	Mouse Left Click item 'NinoN.Personas' at 18;10.
00:03.834	Info	Mouse	Mouse Left Click item 'NINOTESTAVIDE.Text' at 44;15.
00:05.036	Info	Keyboard	Key sequence '161290-12501{Return}' with focus on 'NINOTESTAVIDE.Text'.
00:06.617	Info	Mouse	Mouse Left DoubleClick item 'NINOTESTAVIDE.Form1' at 43;9.
00:07.514	Info	Mouse	Mouse Left Click item 'NINOTESTAVIDE.Button151' at 5;9.
00:08.624	Info	Mouse	Mouse Left Click item 'NINOTESTAVIDE.Button147' at 10;9.
00:09.511	Info	Mouse	Mouse Left Click item 'NINOTESTAVIDE.Close' at 7;18.
00:11.293	Info	Mouse	Mouse Left Click item 'NINOTESTAVIDE.Close1' at 8;8.

3.28. att. **Pirmās testu kopas testēšanas atskaite**



Testu_kopa_2

Success

Execution time
13.05.2015 21:59:44

Computer name
TOMS

Operating system
Windows 7 Service Pack 1 64bit

Screen dimensions
1920x1080

Language
en-US

Total errors
0

Total warnings
1

Filter: ☑ Info ☑ Warn ☑ Success

Time	Level	Category	Message
00:01.187	Info	Application	Run application 'C:\Program Files (x86)\NINO_NET\NinoN.exe' with arguments '' in normal mode.
00:01.580	Info	Mouse	Mouse Left Click item 'NINOTESTAVIDE.LVLEUR' at 31;11.
00:02.527	Warn	Firefox	Please make sure that you have the Ranorex Addon installed and enabled in your Mozilla Firefox browser. Otherwise Ranorex UI element identification capabilities for Firefox are limited. The following website provides more information on this technology limitation: http://www.ranorex.com/support/user-guide-20/instrumentation-wizard/mozilla-firefox.html (This message is only shown once per report.)
00:03.476	Info	Mouse	Mouse Left Click item 'NINOTESTAVIDE.Text' at 63;10.
00:04.269	Info	Keyboard	Key sequence '100' with focus on 'NINOTESTAVIDE.Text'.
00:04.804	Info	Validation	Validating AttributeEqual (Text='142.29') on item 'NINOTESTAVIDE.Text1'.
00:04.915	Success	Validation	Attribute 'Text' of element for item 'NINORepository.NINOTESTAVIDE.Text1' does match the specified value.
00:05.077	Info	Mouse	Mouse Left Click item 'NINOTESTAVIDE.Close2' at 5;5.

3.29. att. **Otrās testu kopas testēšanas atskaite**

Apskatot atskaites, nav skaidrs, kādēļ abās testēšanas atskaitēs ir pieminēts brīdinājums par *Ranorex* pievienojumprogrammu tīmekļa pārlūkprogrammā *Firefox*. Abas testu kopas izdevās veiksmīgi izpildīt, kā arī EUR lauka vērtība tika pārbaudīta korekti. Autoram radās problēmas ar pirmās testu kopas izpildi – to nācās ierakstīt un izpildīt vairākas reizes, jo dažos gadījumos netika identificēts kāds no objektiem. Autors uzskata, ka šādā situācijā veicamās darbības ir jāieraksta lēnākā tempā, jo šķiet, ka rīks nespēj korekti izsekot līdzi visām darbībām, ja tās izpilda pārāk ātri.

3.5.3. Autora atsauksmes par Ranorex un rīka novērtējums

Atskaitot problēmu ar pirmās testu kopas ierakstīšanu (to nācās veikt vairākas reizes), kopumā rīks ar testa kopām tika galā veiksmīgi. Ierakstītās testa darbības ir ērti lasāmas un rediģējams, kā arī rīka lietojamība ir ērta un viegli saprotama. Fakts, ka darbības tiek ierakstītas objektu līmenī nodrošina testu kopu atkārtojamību gadījumos, kad testi tiek palaisti uz kādu citu datoru, kurā, piemēram, ir savādāka ekrāna izšķirtspēja vai NINO formas ir novietotas citās koordinātās.

Pēc testēšanas ģenerētās atskaites ir viegli saprotamas. Ja testpiemēru atkārtošanas laikā radās kādas problēmas, rīks uzņem ekrānuzņēmumu ar lietotnē redzamo situāciju un pievieno paskaidrojošu tekstu.

3.6. Squish GUI Tester

3.6.1. Squish projekta izveide un testpiemēru izveidošana

Jaunu *Squish* projektu var izveidot, nospiežot uz izvēlni „File” -> „New” -> „Squish Test Suite”. Pēc tam ir jāizvēlas viena no piedāvātajām skriptēšanas valodām, kura tiks izmantota testa skriptos. Piedāvātās valodas – *JavaScript*, *Perl*, *Python*, *Ruby* un *Tcl*. Tā kā autors ar *Python* iepazinās izstrādājot *Sikuli* skriptu, tad arī šī projekta ietvaros tika izvēlēta valoda *Python*. Nākošajā logā tika norādīts ceļš uz lietotnes *NinoN.exe* atrašanās vietu uz cietā diska.

Pēc tam testu komplektam nepieciešams pievienot testpiemēru. Tas izdarāms, nospiežot pogu „New Test Case”. Nospiežot pogu „Record Test Case”, tiek atvērta lietojumprogramma NINO un parādās attēlā 3.30. redzamais veikto darbību ierakstītājs. Līdzīgi kā citu rīku

ierakstītājiem, arī Squish piedāvā ieraksta laikā veikt dažādas pārbaudes, piemēram, var pārbaudīt vai kāda objekta kāds no atribūtiem atbilst konkrētai vērtībai..

3.30. att. **Squish veikto darbību ierakstītājs**

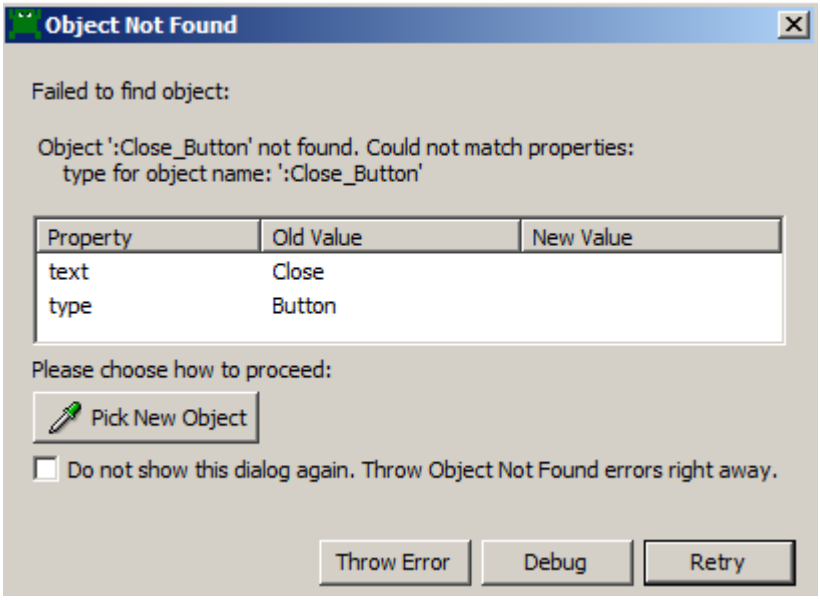
Pēc vairākiem testu ierakstīšanas un atkārtošanas mēģinājumiem, tika iegūts sekojošs skripts:

```
import time
def main():
    startApplication("NinoN")
    mouseClick(waitForObject(":Autentifikācija_Edit"), 17, 9, MouseButton.PrimaryButton)
    type(waitForObject(":Autentifikācija_Edit"), "[redacted]")
    type(waitForObject(":Autentifikācija_Edit"), "<Tab>")
    type(waitForObject(":Autentifikācija_Edit_2"), "[redacted]")
    type(waitForObject(":Autentifikācija_Edit_2"), "<Return>")
    mouseClick(waitForObjectItem(":NINO-TESTA VIDE_MenuBar", "Dati"))
    mouseClick(waitForObjectItem(":Dati_MenuItem", "Personas"))
    mouseClick(waitForObject(":NINO-TESTA VIDE_Edit"), 90, 16, MouseButton.PrimaryButton)
    type(waitForObject(":NINO-TESTA VIDE_Edit"), "161290-12501")
    type(waitForObject(":NINO-TESTA VIDE_Edit"), "<Return>")
    doubleClick(waitForObject(":_Pane"), 57, 7, MouseButton.PrimaryButton)
    mouseClick(waitForObject(":151_ToolbarItem"))
    mouseClick(waitForObject(":147_ToolbarItem"))
    time.sleep(2)
    if object.exists(":147_ToolbarItem"): #šajā brīdī šāds objekts nav ekrānā
        test.fail("Tests nav izpildījies korekti")
    else:
        test.passes("Tests veiksmīgi izpildījies")
    type(waitForObject(":NINO-TESTA VIDE_Tree"), "<Ctrl+F4>")
    type(waitForObject(":_Pane"), "<Ctrl+F4>")
```

3.31. att. **Pirmās testu kopas skripts**

Testa darbību ierakstīšanas laikā rīks dažreiz nereģistrēja peles darbības. Tāpat bija problēmas ar teksta laukā ievadītajām vērtībām, ja tās ierakstīja pārāk ātri – kāda no teksta daļām mēdza pazust. Lai korekti reģistrētu visas darbības, tās nācās veikt lēnā tempā.

Papildus tika konstatētas problēmas ar formu aizvēršanu – rīks nespēja identificēt aizvēršanas pogas. Šādā situācijā tika attēlots attēlā 3.32. redzamais logs. Cenšoties rīkam norādīt, kura poga jāspiež, tika izmantota poga „Pick New Object", bet arī šajā gadījumā rīks nereģistrēja aizvēršanas pogu kā objektu, uz kura būtu jāizpilda peles klikšķis. Šo gadījumu nācās risināt, izmantojot taustiņu kombināciju Ctrl + F4, kuru nospiežot ik reizi, tiek aizvērts lietotnes atvasinātais logs.



3.32. att. ***Squish*** **nespēj identificēt formu aizvēršanas pogu**

Izpildot pirmo testu kopu, tika iegūta attēlā 3.33. redzamā atskaite. Atšķirībā no citiem iepriekš apskatītajiem rīkiem (*TestComplete* un *Ranorex*), *Squish* atskaitē netiek attēlotas visas veiktās darbības. Attēlots tiek tas, kura pārbaude ir izpildījusies un kura nē. Šim iemeslam parasti izmanto pārbaudes funkcijas „test.pass", „test.fail", „test.fatal" u.c. Autora gadījumā funkcijas „test.pass" vietā bija jāraksta „test.passes", jo „pass" ir skriptēšanas valodas Python rezervētais vārds, tādēļ to nevarēja pielietot šādā funkcijā.

Kaut arī pēc autora domām skripts tika uzrakstīts pareizi (pēc analoģijas skripts ir līdzīgs *Sikuli* skriptam, kur arī tika pārbaudīta objekta esamība), *Squish* atgrieza rezultātu, ka šāds objekts tomēr eksistē, kā rezultātā atskaitē redzams „fail".

Result	Message	Time
⊟ Start	tst_1_testu_grupa	2015.14.5 21:08:49
⊟ Fail	Tests nav izpildijies	2015.14.5 21:09:07
Detail		

3.33. att. **Pirmās testu kopas atskaite**

Šim rīkam piemīt nianse, ka pie katras testu grupas izpildes *AUT* tiek startēts un pēc testu pabeigšanas lietotni izslēdz. Tātad, ja projektā ir divas testu grupas, kā tas ir šajā praktiskajā darbā, tad lietotni startēs un izslēgs divas reizes. Rīks nedarbojās tad, kad lietotne jau bija startēta. Dēļ šī iemesla skriptiem ir pievienota autentifikācija. Drošības apsvērumu dēļ autors ir aizsedzis izmantoto lietotājvārdu un paroli.

Pēc tam tika izveidota otrā testu kopa, kurā tika pievienota rīka piedāvātā objekta atribūtu pārbaude. Otrās testu kopas skripts redzams attēlā 3.34.

```
def main():
    startApplication("NinoN")
    mouseClick(waitForObject(":Autentifikācija_Edit"), 46, 7, MouseButton.PrimaryButton)
    type(waitForObject(":Autentifikācija_Edit"), "[illegible]")
    type(waitForObject(":Autentifikācija_Edit"), "<Tab>")
    type(waitForObject(":Autentifikācija_Edit_2"), "[illegible]")
    type(waitForObject(":Autentifikācija_Edit_2"), "<Return>")
    mouseClick(waitForObjectItem(":NINO-TESTA VIDE_MenuBar", "LVL-EUR"))
    mouseClick(waitForObject(":NINO-TESTA VIDE_Edit"), 57, 7, MouseButton.PrimaryButton)
    type(waitForObject(":NINO-TESTA VIDE_Edit"), "100")
    waitFor("object.exists(':NINO-TESTA VIDE_Edit_2')", 20000)
    test.compare(findObject(":NINO-TESTA VIDE_Edit_2").text, "142.29")
    clickButton(waitForObject(":NINO-TESTA VIDE.Beigt Button"))
```

3.34. att. **Otrās testu kopas skripts**

Izpildot izveidoto skriptu, tika iegūta attēlā 3.35. redzamā atskaite. Ar šo testa kopu rīks tika galā veiksmīgi.

Result	Message	Time
⊟ Start	tst_2_testu_grupa	2015.14.5 21:36:49
⊟ Pass	Comparison	2015.14.5 21:37:17
Detail	'142.29' and '142.29...	

3.35. att. **Otrās testu kopas atskaite**

3.6.2. Autora atsauksmes par Squish un rīka novērtējums

No lietojamības viedokļa rīks nebija pārāk ērti lietojams. Šajā rīkā nepieciešamās izvēlnes bija grūtāk atrast, nekā citos rīkos. Veikto darbību ierakstīšana varēja būt labāka, jo uz aizvēršanas pogām rīks vispār nestrādāja, kā arī dažreiz rīks nereģistrēja ar peli veiktās darbības. Izveidotie skripti nav ērti lasāmi, jo nav īsti skaidrs, kurš objekts konkrētajā rindiņā tiek izmantots. Šim rīkam pietrūkst funkcionalitāte, kas pie katras darbības uzņem ekrānuzņēmumu, kā arī pašā skriptā parāda kā izskatās konkrēta poga vai izvēlne. Tāpat arī pati skriptu rakstīšana nebija pārāk ērta, jo ne vienmēr varēja saprast, kuru objektu kodā izmantot. Rīks piedāvā sarakstu ar visiem lietotnes objektiem, bet autoram nepieciešamo tā arī neizdevās atrast, kā arī manuāli rīkam parādot, kur šis objekts atrodas, rīks nenostrādāja.

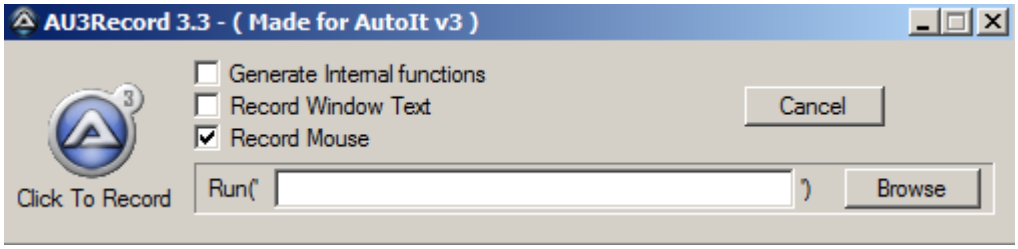
Squish ierakstītās darbības veica ātrāk par citiem rīkiem. Šis ir gan pluss, gan tai pašā laikā šī rīka mīnuss, jo pie pirmās testu kopas izpildes autoram šķita, ka rīks nekorekti pārbauda, vai konkrētais objekts eksistē. Tajā brīdī, kad rīkam bija jāpārbauda objekta esamība, forma jau bija aizvērta un skripta izpilde beigusies. Šī iemesla dēļ autors pie šīs darbības veikšanas pievienoja 2 sekunžu pauzi, bet arī šāds risinājums nenostrādāja un rīks joprojām atgrieza rezultātu, ka objekts eksistē.

Kopumā šis rīks ir vairāk paredzēts cilvēkiem ar programmēšanas prasmēm. Rīka priekšrocība ir tāda, ka to var izmantot dažādās vidēs.

3.7. AutoIt

3.7.1. Testa kopu izveidošana ar AutoIt

Skriptēšanas valoda *AutoIt* izmanto skriptu redaktoru *SciTE*. Veikto darbību ierakstīšanu veic ar iebūvēto rīku *AU3Recorder*.



3.36. att. **AU3Recorder**

Ierakstot veiktās darbības, rīks izmanto ekrāna koordinātas, lai noteiktu, kur jāveic peles klikšķi. Salīdzinot ar objektu atpazīšanas metodi, šī metode ir sliktāka, jo ierakstītos skriptus praktiski nav iespējams pielietot atkārtoti. Autors novēroja, ka pēc pāris atkārtošanas reizēm peles klikšķi tika izdarīti uz neīstajām pogām vai arī vispār ārpus konkrētas formas. To, kāds skripts tika izveidots, var aplūkot attēlā 3.37. Jāņem vērā, ka šajā skriptā manuāli ir pierakstītas vairākas darbības, jo sākotnējais skripts bija mazāk lietojams, nekā tas ir šobrīd. Piemēram, pēc noklusējuma pēc katras darbības netika ieturēta pauze, bet uzreiz tika veikta nākamā darbība. Šādā gadījumā konkrētas formas vēl nebija atvērušās, bet rīks jau izdarīja peles klikšķus vietās, kur pēc kāda laika vajadzētu parādīties kādam no lietotnes objektiem. Tāpat nebija iespējams pārbaudīt eiro konvertācijā iegūto vērtību, tādēļ šo kontroli nācās pievienot manuāli.

```
Func _Au3RecordSetup()
    Opt('WinWaitDelay',100)
    Opt('WinDetectHiddenText',1)
    Opt('MouseCoordMode',0)
EndFunc
Func _WinWaitActivate($title,$text,$timeout=0)
    WinWait($title,$text,$timeout)
    If Not WinActive($title,$text) Then WinActivate($title,$text)
    WinWaitActive($title,$text,$timeout)
EndFunc
_AU3RecordSetup()

#comments-start
1. testu kopa
#comments-end
#region --- Au3Recorder generated code Start (v3.3.9.5 KeyboardLayout=A0000426)  ---
_WinWaitActivate("NINO-TESTA VIDE","")
MouseClick("left",20,30,1)
MouseClick("left",29,67,1)
sleep(1000)
MouseClick("left",1042,177,1)
Send("161290-12501{ENTER}")
MouseClick("left",1079,199,2)
sleep(2000)
MouseClick("left",700,279,1)
sleep(2000)
MouseClick("left",651,270,1)
sleep(2000)
Send("{CTRLDOWN}{F4}{F4}{CTRLUP}")
#endregion --- Au3Recorder generated code End ---

#comments-start
2. testu kopa
#comments-end
#region --- Au3Recorder generated code Start (v3.3.9.5 KeyboardLayout=A0000426)  ---
_WinWaitActivate("NINO-TESTA VIDE","")
MouseClick("left",661,28,1)
_WinWaitActivate("NINO-TESTA VIDE","Eiro kalkulators")
MouseClick("left",612,388,1)
Send("100")
MouseClick("left",771,388,2)
sleep(500)
Send("{CTRLDOWN}c{CTRLUP}")
$vertiba = ClipGet()
If $vertiba = "142.29" Then
    MsgBox(0, "Pazinojums", "Vertiba sakrit ar paredzeto.", 3)
Else
    MsgBox(0, "Pazinojums", "Vertiba nesakrit ar paredzeto.", 3)
endif
MouseClick("left",869,399,1)
#endregion --- Au3Recorder generated code End ---
```

3.37. att. ***AutoIt* skripts ar abām testu kopām**

Rīkā nav iebūvēta veikto soļu atskaite. Vienīgais veids, kā iegūt atskaiti par veiktajām darbībām, ir integrēt *AutoIt* ar kādu citu rīku, vai arī pievienot kādu bibliotēku, kas nodrošina šādu funkcionalitāti.

3.7.2. Autora atsauksmes par AutoIt un rīka novērtējums

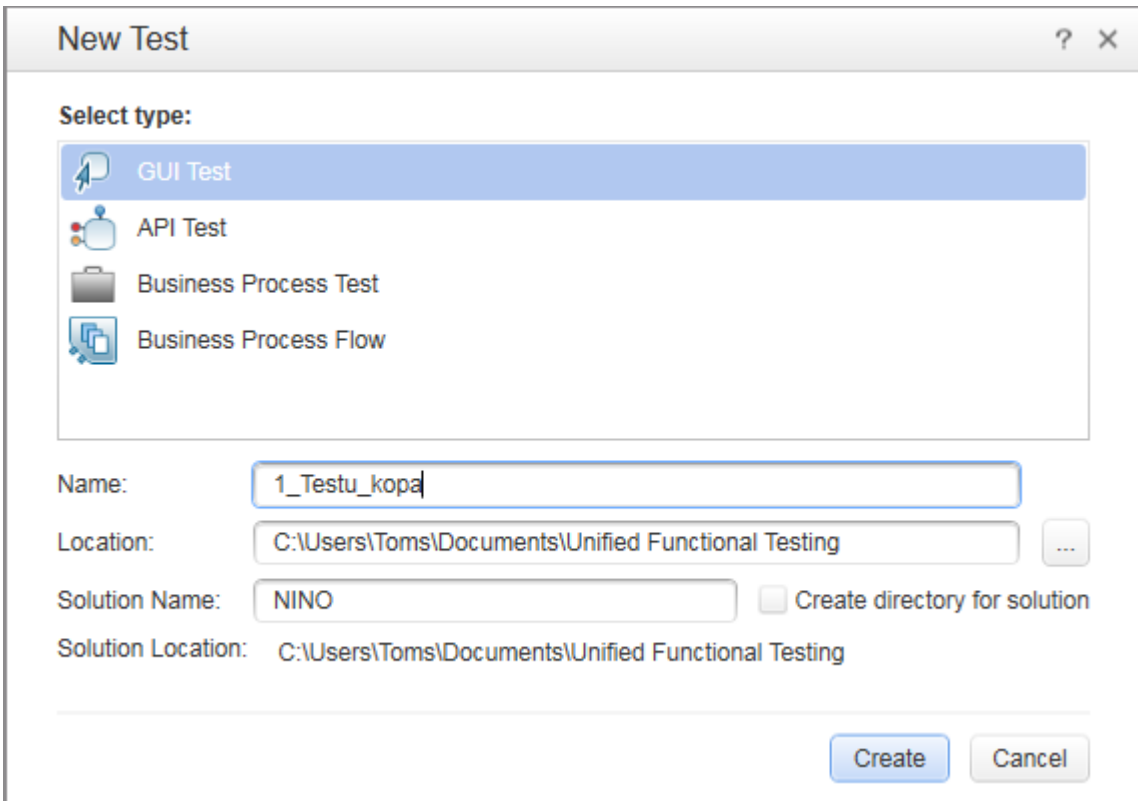
AutoIt iebūvētā veikto darbību ierakstīšana ir ne pārāk noderīga, jo peles darbību veikšana noteiktās ekrāna koordinātās ir pārāk novecojusi tehnoloģija. Skriptus, kuri izveidoti šādā veidā, nav iespējams izmantot ilgtermiņā, jo bieži vien radīsies situācija, kad konkrētās ekrāna koordinātas vairs neatbilst objektam, ar kuru nepieciešams veikt kādu darbību. Lai no skriptēšanas valodas *AutoIt* varētu izveidot konkurētspējīgu testēšanas rīku, *AutoIt* ir jāintegrē ar dažādiem citiem rīkiem vai arī jāimportē dažādas bibliotēkas. Kopumā šis process aizņem pārāk daudz laika – gan no izmantojama rīka izveides viedokļa, gan no testpiemēru izveides viedokļa.

Pozitīvi, ka izveidotos skriptus var pēc tam konvertēt kā izpildāmas programmas un izmantot uz datoriem, uz kuriem nav instalēts *AutoIt*.

3.8. Unified Functional Testing

3.8.1. UFT projekta un testa kopu izveide

Jaunu testu var izveidot, nospiežot izvēlnes pogas „File” -> „New” -> „Test”, vai arī nospiežot taustiņu kombināciju Ctrl + N. Parādās attēlā 3.38. redzamais logs, kurā var norādīt, kāda tipa tests tiks izveidots. Šajā formā jānorāda arī testa kopas un projekta nosaukums. Autora gadījumā tika izvēlēts testa tips „GUI Test”, kas atbilst grafiskās lietotāja saskarnes testam.



3.38. att. **UFT jauna testa izveide**

Pēc tam pie darbību ierakstīšanas un izpildes iestatījumiem tika norādīts, ka tiks ierakstītas darbības darbvirsmas lietotnē, kā arī norādīts ceļš uz *NINON.exe* atrašanās vietu uz cietā diska.

Tālāk tika izveidotas abas testu kopas. Darbību ierakstītājs aplūkojams attēlā 3.39. Darbību ierakstītājā var pārslēgties no vienas testu kopas uz otru, pievienot dažādas objektu kontroles, kā arī norādīt kāda veida ieraksts tiks veikts. Iespējamie ierakstu veidi ir sekojoši:

- „Default" – noklusētais ieraksta veids;
- „Analog Recording" – ar šo veidu tiek ierakstītas ne tikai klaviatūras darbības, bet arī peles kustības;
- „Low-level Recording" – ar šo ieraksta veidu tiek ierakstītas darbības pēc ekrāna koordinātām;
- „Insight Recording" – ieraksta darbības, balstoties uz objekta izskatu, nevis uz objekta parametriem.

3.39. att. **UFT darbību ierakstītājs**

Veicot darbību ierakstīšanu, tika konstatēts, ka rīks ne īpaši veiksmīgi tiek galā ar situāciju, ja ekrānā ir divas vienādas pogas – šajā gadījumā 2 aizvēršanas pogas. Tika nolemts rediģēt skriptu un logu aizvēršanai izmantot Ctrl + F4 taustiņu kombināciju. Izmēģinot šo variantu, tika konstatēts, ka rīks automātiski nereģistrē šādas klaviatūras darbības, tāpēc nācās manuāli pierakstīt nepieciešamo kodu. Iegūtais pirmās testu kopas kods *VBScript* valodā aplūkojams attēlā 3.40.

```
Sub aizvertLogus(reizes)
    For i = 1 To reizes
        poguKombinacijas.SendKeys("^{F4}")
    Next
End Sub

Dim poguKombinacijas
set poguKombinacijas = CreateObject("WScript.shell")

VbWindow("VbWindow").WinMenu("Menu").Select "Dati;Personas"
VbWindow("VbWindow").VbWindow("VbWindow_4").VbEdit("VbEdit").Set "161290-12501"
VbWindow("VbWindow").VbWindow("VbWindow_4").VbEdit("VbEdit").Type  micReturn
VbWindow("VbWindow").VbWindow("VbWindow_3").WinObject("TG60.ApexGrid32.20").DblClick 53,7
wait(1)
VbWindow("VbWindow").VbWindow("VbWindow").WinToolbar("msvb_lib_toolbar").Press 10
wait(2)
VbWindow("VbWindow").VbWindow("VbWindow").WinToolbar("msvb_lib_toolbar").Press 8
aizvertLogus(2)
Reporter.ReportEvent micPass, "Datu parnemsana pec pers. koda", "Tests izpildijas"
```

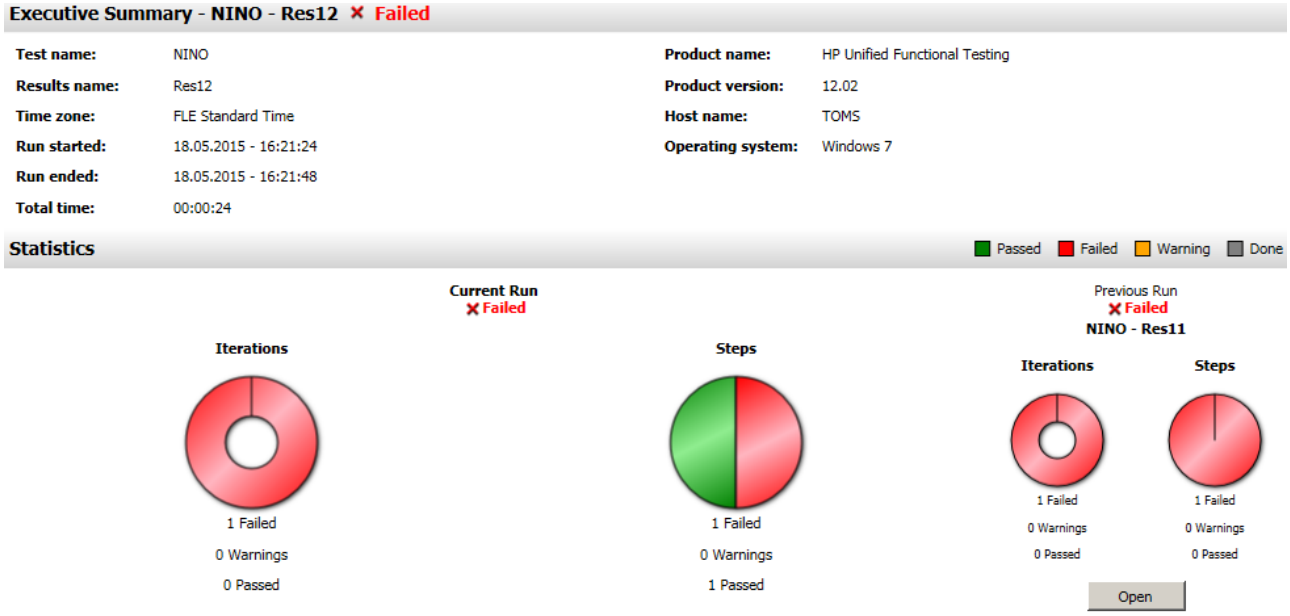
3.40. att. **Pirmās testu kopas skripts**

Pēc tam tika ierakstīta otrā testu kopa, kurai tika pievienota teksta lauka kontrole, lai pārbaudītu, ka EUR konvertācijā ir iegūta pareiza summa. Rīks pēc noklusējuma teksta laukos vērtības iekopē, bet NINO eiro kalkulatorā vērtības vajag ievadīt, tādēļ skriptā ierakstīto funkciju „Set” vajadzēja nomainīt uz „Type”. Iegūtais skripts aplūkojams attēlā 3.41.

```
VbWindow("VbWindow").WinMenu("Menu").Select "LVL-EUR"
VbWindow("VbWindow").VbWindow("VbWindow").VbEdit("VbEdit").Type "100"
VbWindow("VbWindow").VbWindow("VbWindow").VbEdit("VbEdit_2").Check CheckPoint("VbEdit")
VbWindow("VbWindow").VbWindow("VbWindow").VbButton("Beigt").Click
```

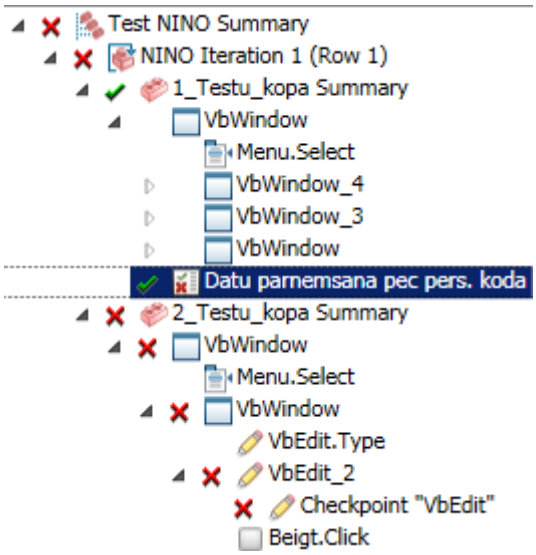
3.41. att. **Otrās testu kopas skripts**

Aplūkojot rīkā ģenerētās atskaites, redzams, ka viena testu kopa ir izdevusies, bet otrā bijusi kļūme. Pirmā testu kopa izpildījās korekti, otrā testu kopā nepareizi nostrādāja eiro summas kontrole. Iegūtā atskaite redzama attēlos 3.42. un 3.43.



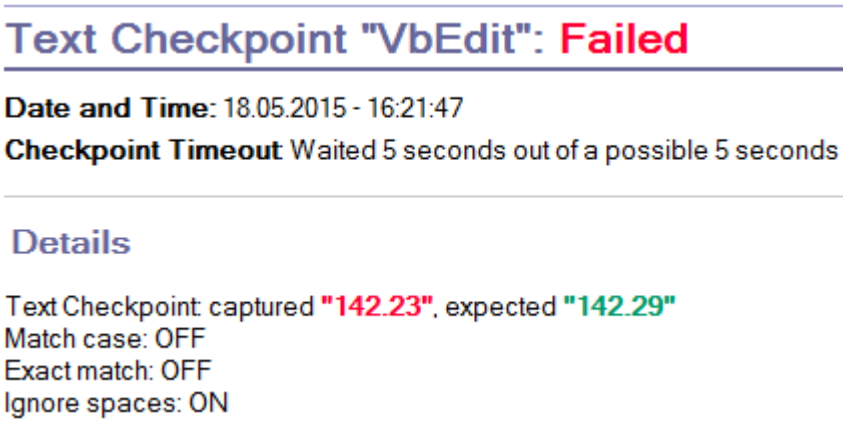
3.42. att. ***UFT*** **testēšanas atskaite(1)**

Attēlā 3.43. redzamajā kokā pie katra soļa var ērti aplūkot, vai šis solis ir izpildījies vai nē. Tāpat kā citos rīkos, arī *UFT* pie testa soļiem tiek uzņemts ekrānuzņēmums.



3.43. att. ***UFT*** **testēšanas atskaite(2)**

Aplūkojot iemeslu, kādēļ eiro summas kontrole neizdevās, autors noskaidroja, ka rīks vērtību „142.29” uztvēra kā 142.23” (attēls 3.44.), lai gan pie šī soļa pievienotajā ekrānuzņēmumā redzams, ka laukā ir vērtība „142.29”.



3.44. att. ***UFT*** **nepareizi nolasīja lauka vērtību**

Pēc tam autors pārbaudīja, cik precīzi strādā ieraksta veids „Insight Recording". Tā kā otrā testu kopa pirms tam neizdevās, tad šis ieraksta veids tika pielietos otrajai testu kopai. Rezultātā pēc nelielas koda rediģēšanas tika iegūts skripts, kas redzams attēlā 3.45. Atskaitē tika parādīts rezultāts, ka vērtība sakrīt.

```
VbWindow("VbWindow").InsightObject( LVL-EUR ).Click
VbWindow("VbWindow").InsightObject( LVL ).Click
Window("NiNo").Window("NINO-TESTA VIDE").WinObject("Text").Type "100"
If VbWindow("VbWindow").InsightObject( 142.29 ).Exist Then
    Reporter.ReportEvent micPass, "Eiro kalkulators", "Vertiba sakrit"
    Else
    Reporter.ReportEvent micFail, "Eiro kalkulators", "Vertiba nesakrit"
End If

VbWindow("VbWindow").InsightObject( Beigt ).Click
```

3.45. att. **Otrā testu kopa ar „Insight Recording"**

3.8.2. Autora atsauksmes par UFT un rīka novērtējums

Rīka objektu identificēšana varēja būt labāka. Pirmo testa kopu izdevās korekti ierakstīt un palaist ar astoto reizi. Labi, ka rīkam ir iebūvēta uzturēšanas palaišanas (*maintenance run*) funkcionalitāte, ar kuras palīdzību pie soļa, kas nav izdevies, var rīkam konkrēti parādīt, ar kuru objektu jāveic kāda darbība. Jāņem vērā, ka ne vienmēr šī funkcionalitāte palīdzēja – dažreiz rīks vienalga nespēja korekti identificēt objektu. Šo problēmu lielā mērā atrisina fakts, ka skriptus var ierakstīt dažādos veidos. Pēc autora novērojumiem, „Insight Recording" ierakstīšanas metode ir uzticama, kā arī ja kāds attēls darbību ierakstīšanas laikā nav precīzi uzņemts, to var viegli rediģēt pašā rīkā.

Kopumā par rīku rodas iespaids, ka ar to varētu tikt galā ar vairumu situāciju, ja atbilstoši situācijai tiek pielietotas dažādas rīkā iebūvētās tehnoloģijas.

3.9. eggPlant Functional

Praktiskā darba izstrādes laikā tika noskaidrots, ka rīkam *eggPlant Functional* nepieciešamas divas ierīces, lai varētu izmantot šo rīku. Var izmantot gan citu fizisku ierīci, gan virtuālo datoru. Paredzēts, ka uz vienas ierīces ir sistēma, kura tiek testēta, bet uz otras ierīces ir palaists pats testēšanas rīks. Starp abām šīm ierīcēm ir jāizveido *VNC (Virtual Network Computing)* savienojums. Tā kā autoram nav pieejas otrai fiziskai vai virtuālai ierīcei, uz kuras būtu instalēta lietojumprogramma NINO, tika nolemts ar šo rīku testpiemērus neveidot.

REZULTĀTI UN SECINĀJUMI

Rezultāti:

Darba praktiskajā daļā tika detalizētāk aplūkoti teorētiskajā daļā izvēlētie rīki. Ar katru apskatīto rīku tika izveidotas 2 testu kopas lietojumprogrammā NINO. Tika sasniegts mērķis apskatīt vismaz 5 regresa testēšanas automatizācijas rīkus darbvirsmas lietotnēm (teorētiskajā daļā apskatīti 7 rīki, praktiskajā daļā 6).

Darba noslēgumā izstrādāts testēšanas rīku salīdzinājums:

Tabula 3.2.

Testēšanas rīku salīdzinājums

Kritērijs/ Testēšanas rīks	*Sikuli*	*TestComplete*	*Ranorex*	*Squish*	*AutoIt*	*UFT*
Funkcionalitāte:						
„Record and playback”	Nē	Jā. Zema līmeņa procedūras, skripti, atslēgvārdu testi, slodzes testi.	Jā. Atslēgvārdu testi, uz attēliem bāzēti testi.	Jā	Jā. Zema līmeņa procedūras	Jā. Zema līmeņa procedūras, atslēgvārdu testi, uz attēliem bāzēti testi
Skripti	Jā	Jā	Jā	Jā	Jā	Jā
Testēšanas atskaites	Nē, atskaite ir jākodē	Jā	Jā	Jā	Nē, atskaite ir jākodē	Jā
Lietojamība:						
Rīka instalācija	Vidēji sarežģīta, it īpaši tad, ja uz datora nav aktuāla *JAVA* versija.	Vienkārša	Vienkārša	Vienkārša	Vienkārša. Nācās instalēt gan pašu *AutoIt*, gan skriptu redaktoru *SciTE*	Vienkārša
Lietotāja saskarne	Viegli saprotama un ērti lietojama.	Viegli saprotama un ērti lietojama	Viegli saprotama un ērti lietojama	Bija nelielas grūtības atrast nepieciešamās pogas un izvēlnes. Kopumā darbošanās ar rīku nelikās pārāk ērta.	Ērti lietojama, bet tai pašā laikā pirmajā brīdī nav skaidrs, kādu funkcionalitāti veic katrs no iebūvētajiem rīkiem	Viegli saprotama un ērti lietojama.
Tehniskais atbalsts	Ļoti labs. Uz jautājumiem ātri tiek saņemtas	Ļoti labs. Rīkam veltītajā forumā notiek aktīvas	Ļoti labs. Forums ir aktīvs un arī dokumentā	Rīkam nav pieejams oficiālais forums, kurā uzdot	Aktīvs forums. Dokumentācija ir labā līmenī.	Aktīvs forums. Papildus atbalstu var saņemt,

	atbildes. Pieejamā dokumentācija labā līmenī.	diskusijas par dažādākajiem jautājumiem. Pieejamā dokumentācija labā līmenī.	cija ir labā līmenī	jautājumus. Galvenokārt atbalsts notiek caur e-pastu. Dokumentācija ir labā līmenī.		zvanot pa tālruni Dokumentācijai var piekļūt ar *HP* pasi..
Testu izveide	Skriptēšana ar attēliem būtiski atvieglo testu izveidi. Testu izveidošana ir diezgan vienkārša.	Nebija sarežģīti izveidot testus. Tā kā ir pieejami dažādi darbību ierakstīšanas veidi, tad ir iespējams tikt galā ar vairumu situāciju	Abas testu kopas tika ierakstītas un izpildītas salīdzinoši veiksmīgi. Kopumā testus izveidot ir vienkārši.	Testus izveidot nebija grūti, bet pēc tam tos ir grūti rediģēt, jo nav skaidrs, kuru objektu izmantot.	Testu ierakstīšana nav pārāk laba, jo ieraksta ekrāna koordinātas. Visu nepieciešams kodēt.	Bija problēmas ar abām testu kopām. Beigās tās izdevās korekti izpildīt, pielietojot rīka piedāvātās dažādās tehnoloģijas.
Testēšanas atskaites	Pašā rīkā atskaites nav, bet ir vienkārši integrēt kādu citu risinājumu. Praktiskajā daļā izmantotais *HTMLTestrunner* ir gana labs risinājums.	Ļoti labas atskaites. Pie katra soļa pievienotais ekrānuzņēmums būtiski uzlabo atskaišu pārskatīšanu.	Ļoti labas atskaites. Ja testa laikā kaut kas neizdodas, tiek uzņemts ekrānuzņēmums.	Atskaite ir ne īpaši laba. Tajā netiek attēloti visi veiktie soļi, kā arī netiek uzņemti ekrānuzņēmumi.	Nav komentāri. Praktiskajā darbā netika integrēts atskaišu veidošanas risinājums	Atskaite ir gana laba. Gadījumos, kad testu izpildes laikā radās problēmas, atskaite bija informatīva.
Rīka atbalstītās vides:						
Operētājsistēmas	*Microsoft Windows, Mac OSX, Linux*	*Microsoft Windows*	*Microsoft Windows, Android, iOS*	*Microsoft Windows, Mac OSX, Linux, Android, iOS, Embedded Linux, Windows CE*	*Microsoft Windows*	*Microsoft Windows*
Programmēšanas valodas	*Python, Ruby. Sikuli API* viegli integrējams valodā *JAVA.*	*VBScript, JScript, DelphiScript, C++Script, C#Script*	*C#, VB.NET*	*JavaScript, Perl, Python, Ruby, Tcl*	*AutoIt*	*VBScript*
Tīmekļa lietotnes	Jā	Jā	Jā	Jā	Nē	Jā
Darbvirsmas lietotnes	Jā	Jā	Jā	Jā	Jā	Jā
Mobilās lietotnes	Jā, izmantojot mobilo ierīču	Jā	Jā	Jā	Nē	Jā

	emulatoru.					
Izmaksas:						
Maksas rīks	Nē	Jā	Jā	Jā	Nē	Jā
Cena	-	Vienam datoram 889 EUR, peldošā licence 2229 EUR.	Vienam datoram 1990 EUR, peldošā licence 3490 EUR.	Cena nav norādīta. Lai noskaidrotu precīzu cenu, jākontaktējas ar firmas pārstāvi.	-	Cena nav norādīta. Lai noskaidrotu precīzu cenu, jākontaktējas ar firmas pārstāvi.
Licences tips	-	Viena datora licence, peldošā licence.	Viena datora licence, peldošā licence.	Viena datora licence, grupu licence.	-	Viena datora licence, peldošā licence.

Autora vērtējums, kurš no rīkiem varētu būt piemērotākais uzņēmumam SIA ZZ Dats: Tā kā visiem rīkiem jebkurā gadījumā būs nepieciešama papildus kodēšana, lai to varētu pilnvērtīgi izmantot uzņēmuma vajadzībām, tad pēc autora domām uzņēmumam vajadzētu detalizētāk apskatīt un izmēģināt *Sikuli*. *Sikuli*, integrēts ar visdažādākajiem jau esošajiem risinājumiem, ir tikpat spējīgs kā jebkurš maksas rīks. Ņemot vērā, ka vienkāršu skriptu izveidošanai nav nepieciešamas profesionālas programmēšanas prasmes, rīku varētu pielietot jebkurš uzņēmuma darbinieks, tai pašā laikā apgūstot programmēšanas pamatus.

Secinājumi:

1. Teorētiski darbību ierakstīšana un automātiska izpilde šķiet labs veids, kā veidot testpiemērus, bet praktiski bieži vien ierakstītais tests nestrādā, kā vajag, tādēļ nepieciešams ierakstītās darbības pārskatīt un pielabot ar kodu.
2. Tā kā testēšanas automatizācijas rīka ieviešana uzņēmumā nav tikai 1 personas darbs, tad ir tikai loģiski pieņemt, ka pilnīga rīku funkcionalitāte netika pārbaudīta. Efektīva metode, kā uzņēmumā pārbaudīt rīka funkcionalitāti, ir katrā uzņēmumā esošajā projektā izveidot nelielu komandu, kura sastāv no vismaz 1 testētāja un 1 programmētāja. Katra komanda varētu apskatīt rīku uz savām projektā esošajām sistēmām un pieņemt lēmumu, vai viņu projektam rīks atvieglo darbu.

IZMANTOTĀ LITERATŪRA UN AVOTI

1. Elfriede Dustin. *Effective Software Testing: 50 Specific Ways to Improve Your Testing*, Addison-Wesley Professional, 2002. 39. nodaļa *Automate Regression Tests When Feasible*.
2. *Sikuli/SikuliX* mājas lapa [tiešsaiste]. – [atsauce 23.01.2015.] Pieejams: http://www.sikuli.org/ un http://www.sikulix.com/
3. *Sikuli/SikuliX* dokumentācija [tiešsaiste]. – [atsauce 24.01.2015.] Pieejams: http://sikulix-2014.readthedocs.org/en/latest/index.html
4. *TestComplete* mājas lapa [tiešsaiste]. – [atsauce 24.01.2015.] Pieejams: http://smartbear.com/product/testcomplete/overview/
5. *TestComplete* dokumentācija/atbalsts [tiešsaiste]. – [atsauce 24.01.2015.] Pieejams: http://support.smartbear.com/viewarticle/65670/
6. *Ranorex* mājas lapa [tiešsaiste]. – [atsauce 25.01.2015.] Pieejams: http://www.ranorex.com/
7. *Ranorex* dokumentācija [tiešsaiste]. – [atsauce 25.01.2015.] Pieejams: http://www.ranorex.com/Documentation/Ranorex-Tutorial.pdf
8. *Squish* mājas lapa [tiešsaiste]. – [atsauce 25.01.2015.] Pieejams: http://www.froglogic.com/index.php
9. *Squish Coco* un *Squich GUI Tester* dokumentācija [tiešsaiste]. – [atsauce 25.01.2015.] Pieejams: http://doc.froglogic.com/
10. *AutoIt* mājas lapa [tiešsaiste]. – [atsauce 25.01.2015.] Pieejams: https://www.autoitscript.com/site/autoit/
11. *AutoIt* dokumentācija [tiešsaiste]. – [atsauce 25.01.2015.] Pieejams: https://www.autoitscript.com/autoit3/docs/
12. *UFT* mājas lapa [tiešsaiste]. – [atsauce 26.01.2015.] Pieejams: http://www8.hp.com/us/en/software-solutions/unified-functional-automated-testing/
13. *UFT* dokumentācija [tiešsaiste]. – [atsauce 26.01.2015.] Pieejams: http://support.openview.hp.com/selfsolve/document/KM00794319 (nepieciešama HP Pase (*HP Passport*))
14. *TestPlant* mājas lapa [tiešsaiste]. – [atsauce 26.01.2015.] Pieejams: http://www.testplant.com/
15. *EggPlant* dokumentācija [tiešsaiste]. – [atsauce 26.01.2015.] Pieejams: http://docs.testplant.com/?q=Documentation-Home

16. Akadēmiskā terminu datubāzē *AkadTerm* [tiešsaiste] Pieejams: http://termini.lza.lv/term.php
17. Informācija par *SikuliX* versijām [tiešsaiste]. – [atsauce 11.05.2015.] Pieejams: http://www.sikulix.com/
18. Python dokumentācija [tiešsaiste]. – [atsauce 14.03.2015.] Pieejams: https://www.python.org/doc/
19. *Sikuli* papildinājums *HTMLTestRunner* [tiešsaiste]. – [atsauce 11.05.2015.] Pieejams: https://answers.launchpad.net/sikuli/+question/176005
20. *Sikuli* jautājumu un atbilžu forums [tiešsaiste]. – [atsauce 12.05.2015.] Pieejams: https://answers.launchpad.net/sikuli
21. *SikuliFramework.* Satvars priekš *Sikuli* [tiešsaiste]. – [atsauce 24.05.2015.] Pieejams: https://github.com/smysnk/sikuli-framework

PIELIKUMI

Sikuli skripts

```
1 #norādam, kur tiks glabāta testēšanas atskaite un ekrānuzņēmumi
2 dir = "C:\\Users\\Toms\\Desktop\\SikuliScripts\\Reports"
3 import os
4 fp = file(os.path.join(dir, "atskaite.html"), "wb")
5
6 from sikuli import *
7 import unittest
8  import  HTMLTestRunner  #šis  imports  atbild  par  testēšanas  atskaišu
veidošanu
9 reload(HTMLTestRunner) #ja importā veicu kādas izmaiņas, tad nepieciešams
pārlādēt
10
11 #norādot ceļu uz app, te varētu iestrādāt programmas auto-open un auto-
login
12 #NINO_exe = App("C:\\Program Files (x86)\\NINO_NET\\NinoN.exe")
13
14 #auto-open
15 #NINO_exe.open()
16
17 setShowActions(False)
18 Settings.ActionLogs = True
19 Settings.InfoLogs = True
20 Settings.DebugLogs = True
21
22 #funkcija netiek izmantota, jo šis bija pirmais veids, kā realizēju logu
aizvēršanu
23 #nav pārāk ātrs risinājums, jo katru reizi meklē pa visu ekrānu konkrētu
pogu
24 def aizvertLogus(reizes):
25     def byY(aizvert):
26         return aizvert.y
27     for x in range(0, reizes):
28         aizvertMasivs = findAll(  )
29         sortedIcons = sorted(aizvertMasivs, key=byY)
30         if len(sortedIcons) >= 2: #ja atrod divas aizvēršanas pogas..
31                 click(sortedIcons[1]) #.. tad nospiežam otro aizvēršanas
pogu no augšas
```

```
32              #type(Key.F4, KEY_CTRL) #var izmantot click(sortedIcons[1])
vietā
33              aizvertMasivs = []
34              sortedIcons = []
35          else:
36              break
37
38 #ātrāks veids, kā aizvērt visus lietotnes child logus
39 def aizvertLogus2(reizes):
40      for x in range(0, reizes):
41          type(Key.F4, KEY_CTRL)
42
43 class Datu_parnemsana(unittest.TestCase): #atbilst pirmajai testu kopai
44      def test_1_personu_forma_atveras(self):
45          click( Dati )
46          wait( Personas , 10)
47 #izmantojot getLastMatch() kopējā izpilde ir ātrāka, jo nav atkārtoti
jāmeklē objekts. Tiek izmantots iepriekš atrastais objekts.
48          object=getLastMatch()
49          click(object)
50          if not exists( Personas ):
51                  #assert False un True ir vajadzīgs, lai atskaitē iegūtu
ekrānuzņēmumus
52              assert False, "Personu forma netika atverta!"
53          else:
54              assert True, "Personu forma tika veiksmigi atverta."
55
56      def test_2_persona_atrasta(self):
                                  Kods(VRN):
57          wait(Pattern(                ).targetOffset(-51,9), 10)
58          object=getLastMatch()
59          click(object)
60          type("161290-12501")
61          type(Key.ENTER)
62          wait( Freibergs Toms , 10)
63          object=getLastMatch()
64          doubleClick(object)
65          if not exists( Freibergs Toms ):
66                  #assert False un True ir vajadzīgs, lai atskaitē iegūtu
ekrānuzņēmumus
```

```
67              assert False, "Persona netika atrasta!"
68          else:
69              assert True, "Persona tika atrasta."
70
71      def test_3_parnemt_datus_pec_PK(self):
72              #šo if nosacījumu vairs nevajag, jo logi tiek aizvērti ar
Ctrl+F4.
73          #if exists ("1427200255074.png"): #ar šo attēlu var identificēt,
ka logs nav pilnā izmērā (maximized)
74          #     object=getLastMatch()
75          #     doubleClick(object) #logs tiek maximizēts
76          wait(  , 120)
77          object=getLastMatch()
78          click(object)
79          wait(  , 120)
80          object=getLastMatch()
81          click(object)
82          waitVanish(  , 120)
83
84          if exists(  ):
85              assert False, "Neizdevas parnemt datus pec personas koda!"
86          else:
87              assert True, "Datu parnemsana tika veiksmigi pabeigta!"
88
89  #Piemērs, lai redzētu kā izskatās atskaite, ja tests neizdodas. Tests
neizdosies, jo šajā brīdī uz ekrāna šāda poga neeksistē.
90  #    def test_4_expected_fail(self):
91  #        find("Dokumenti")
92
93  class Eiro_kalkulators(unittest.TestCase): #atbilst otrajai testu kopai
94      def test_1_eiro_kalkulators(self):
95          aizvertLogus2(2)
96          click(LVL-EUR)
97          wait(Pattern(  ).targetOffset(-6,1), 10)
98          object=getLastMatch()
99          click(object)
100         type("100")
101 #viens variants ir saglabāt lauka vērtibu starpliktuvē (clipboard) un
pēc tam parbaudīt šo vērtību
```

```
102         doubleClick(Pattern(  ).targetOffset(-17,0))
103         type("c", KeyModifier.CTRL)
104         vertiba = Env.getClipboard().strip()
105 
106         if vertiba != "142.29":
107             assert False, "Vertiba nesakrit"
108         else:
109             assert True, "Vertiba sakrit"
110         aizvertLogus2(1)
111 #otrs variants ir pārbaudīt, vai eksistē šāds attēls, kurā ir redzama vērtība "142.29"
112         #if not exists("  "):
113         #    assert False, "Vertiba nesakrit"
114         #else:
115         #    assert True, "Vertiba sakrit"
116 
117 suite = unittest.TestLoader().loadTestsFromTestCase(Datu_parnemsana) #ielādējam testus no pirmās testu kopas
118 
suite.addTests(unittest.TestLoader().loadTestsFromTestCase(Eiro_kalkulators)) #pievienojam testus no otrās testu kopas
119 #padodam testu izpildītājam parametrus..
120 runner = HTMLTestRunner.HTMLTestRunner(stream=fp, title='Atskaite', description='NINO testi', dirTestScreenshots=dir)
121 runner.run(suite) #palaižam visus ielādētos testus
122 fp.close() #aizveram testēšanas atskaites failu
```

Maģistra darbs: **Regresa testēšanas automatizācijas rīka izvēle darbvirsmas lietotnēm.**

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: _____
(Autora paraksts)

Ar savu parakstu apliecinu, ka esmu lasījis augstāk minēto maģistra darbu un atzīstu to par **p i e m ē r o t u / n e p i e m ē r o t u** (nevajadzīgo svītrot) aizstāvēšanai Latvijas Universitātes datorzinātņu maģistrantūrā.

Darba vadītājs: _____
(Vadītāja paraksts)

Darbs iesniegts **maģistrantūras sekretariātā** _____.
(Iesniegšanas datums)

Ar šo es apliecinu, ka darba elektroniskā versija ir augšupielādēta LU informatīvajā sistēmā.

Studiju metodiķe: _____.
(Metodiķes paraksts)

Recenzents: _____
(Akad.amats, zin.grāds, vārds, uzvārds)

Darbs aizstāvēts maģistra gala pārbaudījuma komisijas sēdē

_____ prot. Nr. _____
(Darba aizstāvēšanas datums)

Komisijas sekretārs: _____
(Sekretāra paraksts)