LATVIJAS UNIVERSITĀTE

DATORIKAS FAKULTĀTE

TESTPIEMĒRU PĀRKLĀJUMA ANALĪZES SISTĒMA

KVALIFIKĀCIJAS DARBS

Autors:
Muntis Rudzītis
Stud. apl. nr.: mr10090

Darba vadītāja:
Dr. Mat.
Anda Ādamsone
Testēšanas vadītāja,
SIA „FMS Software”

Rīga 2012

ANOTĀCIJA

Kvalifikācijas darbs apraksta sistēmu, kura nosaka automātisko testu skriptu veikto darbību pārklājumu resursu vadības sistēmai Horizon. Sistēma iegūst testu skriptu veiktās darbības sistēmas saskarnē.

Sistēmas uzdevums ir testu skriptu pirmkodā atpazīt grafiskajā saskarnē veiktās darbības un ievietot izdrukas par tām. Pēc testēšanas skriptu izpildes sistēma apkopo informāciju par testu skriptu veiktajām darbībām un attēlo veiktā funkcionalitātes testa pārklājumu.

Sistēma realizēta Ruby programmēšanas valodā. Atskaišu attēlošanas vietnei izmantots Ruby valodas ietvars Ruby on Rails un SQLite datubāze.

Atslēgvārdi: automātiskie testu skripti, testu pārklājums, Ruby, Ruby on Rails, SQLite.

ABSTRACT

TEST CASE COVERAGE ANALYSIS SYSTEM

Qualification paper describes a system that measures automatic test scripts coverage while testing enterprise resource planning system Horizon. The system collects test script made operations in system user interface.

Systems goal is to read test script source code and identify actions made in graphical user interface and put print functions in source code about them. After test script execution system gathers information about test script actions and represents functional test coverage.

The system is developed in Ruby programming language. Report display site is developed using Ruby programming language framework Ruby on Rails and SQLite database.

Keywords: automatic test scripts, test coverage, Ruby, Ruby on Rails, SQLite.

SATURS

APZĪMĒJUMU SARAKSTS

Horizon – SIA „FMS Software” izstrādātā resursu vadības sistēma.

Rational Robot – IBM izstrādāts automātisko testu izveides un izpildes rīks. SQABasic – programmēšanas valoda, kurā tiek rakstīti Rational robot testu skripti.

Ruby – izmantotā programmēšanas valoda.

Ruby on Rails (RoR) – ietvars, kurš izmantots atskaišu attēlošanas vietnes izveidei.

Skripti – ar rīku Rational robot izveidoti, automātiski izpildāmi testu scenāriji.

Testu scenāriji – noteiktas, secīgi izpildāmas darbības funkcionalitātes testēšanai.

Izdrukas – testu skriptos ievietojamas žurnālu aizpildīšanas komandas par testa veiktajām darbībām.

Parsētājs – programma, kas lielus datu blokus sadala mazākās, vieglāk interpretējamās un tālāk apstrādājamās daļās.

Testēšanas pārklājums – testēšanas laikā veikto darbību salīdzinājums ar kopējo sistēmas darbību kopu.

TPAS – testpiemēru pārklājuma analīzes sistēma.

IEVADS

Uzņēmumā FMS Software papildus testēšanas speciālistu veiktajai manuālajai testēšanai tiek veikta arī automātiska izstrādātās programmatūras testēšana. Ir svarīgi apzināties automātiskās testēšanas pārklājumu un testa veiktās darbības, lai būtu iespējams vienkārši atkārtot un labot automātisko testu atrastās kļūdas, kā arī paplašināt testu veikto pārklājumu veidojot jaunus testu skriptus. Tā kā automātisko skriptu failu skaits ir ļoti liels (kopā aptuveni 700), kā arī skriptu rakstīšanai izmantotās programmēšanas valodas SQABasic pirmkods ir grūti saprotams, ir sarežģīti noteikt katra skripta veiktās darbības. Šo iemeslu dēļ tika nolemts izstrādīt sistēmu, kura spētu atpazīt un vienkāršā cilvēkiem saprotamā veidā attēlot skriptos veiktās darbības. Skriptu izpildes laikā atrastās kļūdas būtu vienkārši atkārtot un izlabot, šos darbību aprakstus iedodot programmētājiem.

Uzņēmuma FMS viena no galvenajām izstrādātajām programmatūrām ir resursu vadības sistēma Horizon. Tā nodrošina ļoti plašu funkcionalitāti, kura nepieciešama darbā ar cilvēkresursiem, grāmatvedību, finansēm, līgumiem, preču un inventāra uzskaiti un citām lietām.

Tā kā sistēmas funkcionalitāte tiek paplašināta un tiek izstrādāti sistēmas moduļi, kuri domāti konkrētiem klientiem, tad ir ļoti svarīgi, lai jaunākie sistēmas uzlabojumi un papildinājumi neveidotu jaunas kļūdas jau esošajā funkcionalitātē. Tādēļ uzņēmums FMS Software papildus sistēmas testēšanai, ko veic testēšanas speciālisti, ir ieviesis IBM Rational Robot rīku, kas ļauj programmēt un atkārtoti izpildīt testus, kas nodrošina lietotāja darbību imitēšanu. Tas sevī ietver ne tikai sistēmas lietošanu, bet arī lauku un tabulu datu pārbaudes, izmantojot verifikācijas punktus, lai noteiktu dažādu aprēķinu pareizību. Testēšana tiek veikta automatizēti, pēc testa skripta izveides, tas tiek atkārtoti izpildīts uz jaunākām sistēmas versijām, tādējādi atrodot kļūdas sistēmā, kuras radušās sistēmas atjauninājumu dēļ un ļauj tās laicīgi izlabot.

Kvalifikācijas darba mērķis ir izveidot sistēmu, kura spētu atpazīt testu skriptu pirmkodos veiktās darbības sistēmas grafiskajā saskarnē un, pēc skriptu izpildes, attēlot skripta veikto darbību sarakstu. Tas palīdzētu atklāt sistēmas daļas, kuras netiek vai tiek vāji testētas, kā arī, iespējams, norādītu uz sistēmas daļām, kurās biežāk tiek atrastas kļūdas, lai tām turpmāk varētu veikt papildus testēšanu.

Darba sekmīgai izpildei tiek izvirzīti šādi uzdevumi:

- Iepazīties ar pašreizējo testēšanas procesu un situāciju.

- Iepazīties ar testēšanas skriptu izmantoto valodu (SQABasic), skriptu programmēšanas veidu un to niansēm.
- Noskaidrot skriptos veiktās darbības, kuras ir nepieciešams atpazīt.
- Izveidot programmatūras prasību specifikāciju.
- Izveidot programmatūras projektējuma aprakstu atbilstoši programmatūras prasību specifikācijai.
- Apgūt programmēšanas valodu Ruby.
- Izstrādāt sistēmas testa skriptu pirmkoda sintaktiskās analizēšanas un izdruku pievienošanas rīka prototipu.
- Atkārtoti veikt izstrādātā rīka testēšanu un uzlabot prototipu.
- Apgrūt tīmekļu vietņu izveides ietvaru Ruby on Rails.
- Izstrādāt sistēmas skriptu veikto darbību attēlošanas vietni.
- Veikt programmatūras testēšanu.
- Ieviest programmatūru.

Sistēma rakstīta Ruby programmēšanas valodā, ievērojot labo programmēšanas praksi. Izveidotas funkcijas, kuras atbild par biežāk izmantotajām darbībām, tādējādi pieturoties pie DRY (Don't repeat yourself) programmēšanas principa. Programma sadalīta moduļos, kur katrs no tiem satur funkcijas, kas kopumā atbild par viena uzdevuma izpildi. Kods, it sevišķi sarežģītākās koda vietas un funkcijas, ir komentēts.

Sistēmas tīmekļa vietnes izstrādē izmantots Ruby tīmekļa vietņu izstrādes ietvars Ruby on Rails (RoR).

Rational Robot testēšanas rīkam pievienotas papildus metodes, kuras veic žurnālu drukāšanu un veido vienotu žurnālu failu struktūru. Plānots testa skriptu pirmkodā ievietot šo funkciju izsaukumus balstoties uz testa skriptos veiktajām darbībām.

Darbs satur programmatūras prasību specifikāciju, programmatūras projektējuma aprakstu, testēšanas dokumentāciju un pirmkoda fragmentus. Iekļauta arī informācija par konfigurāciju pārvaldību, darbietilpības novērtējumu un projekta organizāciju.

Dokumentācija izveidota atbilstoši noteiktajiem valsts standartiem programminženierijā (1, 2), kā arī darba izstrādāšanā ir izmantots Latvijas Universitātes izstrādātais dokuments par noslēguma darbu veidošanu (3.).

1. PROGRAMMATŪRAS PRASĪBU SPECIFIKĀCIJA

1.1 Ievads

1.1.1 Nolūks

Programmatūras prasību specifikācijas (PPS) nolūks ir precīzi un viennozīmīgi aprakstīt testpiemēru pārklājuma analīzes sistēmas prasības. Šis dokuments ir paredzēts sistēmas projektētājam un pasūtītājam produkta funkcionalitātes saskaņošanai un izsekošanai.

Dokuments tiks izmantots par pamatu programmatūras projektējuma apraksta izstrādei.

1.1.2 Darbības sfēra

Šajā dokumentā ir definētas prasības testpiemēru pārklājuma analīzes sistēmai.

Programmatūras produkta mērķis ir analizēt un noteikt automātisko regresijas skriptu kārtējā testēšanas reizē veiktās darbības. Produkta paredzētais pielietojums ir regresijas skriptu veikto darbību atpazinējs un izdruku ievietotājs skriptu pirmkodā, kā arī skriptu veikto darbību attēlošanas vietne.

1.1.3 Saistība ar citiem dokumentiem

Programmatūras prasību specifikācija ir kvalifikācijas darba „Testpiemēru pārklājuma analīzes sistēma" sastāvdaļa.

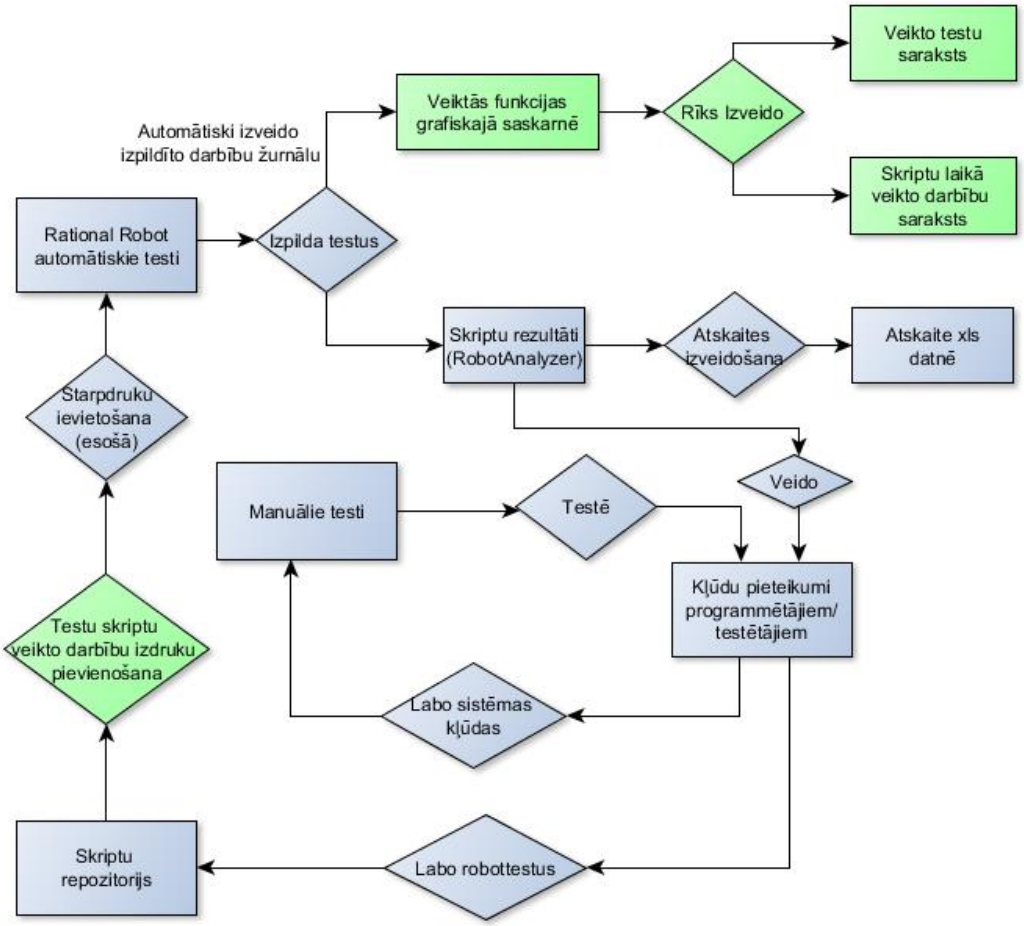
Programmatūras prasību specifikācija ir izstrādāta saskaņā ar standartu „LVS 68:1996 Programmatūras prasību specifikācijas ceļvedis" (1.).

1.2 Vispārējs apraksts

1.2.1 Produkta perspektīva

Izstrādājamais produkts ir programmatūra, kura sastāv no divām lielākām daļām. Pirmā no daļām ir automatizēto regresijas skriptu veikto darbību atpazinējs un informatīvu izdruku ievietotājs, kurš tiks palaists pirms kārtējās skriptu izpildes reizes. Otrā daļa ir skriptu veikto darbību attēlošanas tīmekļa vietne, kura sniegs informāciju par skriptu veiktajām darbībām.

Horizon regresijas testu izpildes shēma attēlā 1.1.



1.1 att. *Horizon regresijas testu izpildes shematisks attēlojums*

1.2.2 Produkta funkcijas

Automatizēto regresijas skriptu veikto funkciju atpazinējam un informatīvu izdruku veidotājam jānodrošina šādu skriptu laikā veikto darbību atpazīšanu:

- izvēlnes atvēršana;
- funkcionālo taustiņu izmantošana;
- darbību izmantošana no sarakstlodziņa;
- izvēlnes rūtiņu izmantošanu;
- pogu nospiešana;
- nākošā skripta izsaukšanu;
- cilnes atvēršanu;
- skripta sākumu un beigas.

Kā arī šim rīkam par katru no izpildītajiem testēšanas skriptiem ir jāveido žurnāla fails, kurā tiek aprakstītas skripta izpildes laikā veiktās darbības un darbību izpildes laiks.

Skriptu veikto darbību attēlošanas rīkam ir jāveic šādas funkcijas:

- Sintaktiski jāapstrādā skriptu veikto darbību žurnālu faili.
- Jāaizpilda datubāzi ar iegūtajiem datiem, ja ir veikti jauni testi pēc iepriekšējo ierakstu pievienošanas.
- Pēc datu saņemšanas, jāattēlo testu skriptu veiktās darbības.

1.2.3 Lietotāja raksturiezīmes

Programmatūras lietotāji ir testēšanas speciālisti, kuri pirms tam ir apmācīti kā rīkoties ar šo rīku.

1.2.4 Vispārējie ierobežojumi

Sistēmai jādarbojas Microsoft Windows vidē.

1.2.5 Pieņēmumi un atkarības

Programmatūra ir speciāli pielāgota konkrētajai darba videi. Lai to izmantotu, to ir nepieciešams konfigurēt atbilstoši darba videi, kā arī ir jābūt uzstādītai vai attālināti jābūt piekļuvei programmatūrai, kuru sistēma izmantos. Lai sistēma pilnvērtīgi darbotos, jābūt pieejamam IBM Rational Robot testēšanas skriptu pirmkodam, testu skriptu laikā veidoto žurnālu failiem.

1.3 Konkrētās prasības

1.3.1 Funkcionālās prasības

Izveidots saraksts ar nepieciešamajām funkcijām un tiem pievienoti identifikatori, lai to sekmīgai izpildei būtu vieglāk izsekot.

Tabula 1.1*Prasību identifikatori*

Identifikators	Prasības nosaukums
FP-1	Darbam nepieciešamās vides sagatavošana
FP-2	Testu skriptu failu meklēšana
FP-3	Žurnāla izveide
FP-4	Žurnāla aizpildīšana
FP-5	Ievades failu veida atpazīšana
FP-6	Klases vai funkcijas definēšanas atpazīšana

Identifikators	Prasības nosaukums
FP-7	Klases beigu atpazīšana
FP-8	Cita skripta izsaukšanas metodes atpazīšana
FP-9	Rindiņas derīguma atpazīšana
FP-10	Rindiņas noformēšana
FP-11	Skripta identificēšanas izdrukas pievienošana
FP-12	Izvēlnes atvēršanas atpazīšana
FP-13	Funkcionālo pogu nospiešanas atpazīšana
FP-14	Pogu nospiešanas atpazīšana
FP-15	Izvēles rūtiņas stāvokļa mainīšanas atpazīšana
FP-16	Izvēles rūtiņas vērtības uzstādīšanas atpazīšana
FP-17	Cilnes atvēršanas atpazīšana
FP-18	Funkciju pogas nospiešanas un darbības izsaukšanas atpazīšana
FP-19	Izveidotās skriptu pirmkoda rindiņas ierakstīšana skripta failā
FP-20	Izpildot automātisko testu jāveido žurnāla fails par veiktajām darbībām
FP-21	Izdruku ievietotājam jābūt savietojamam ar esošo skripta darbību žurnāla
FP-22	Skriptu veikto darbību attēlošanas vietnes datu iegūšana
FP-23	Skriptu veikto darbību attēlošanas vietnes datu attēlošana

FP-1 Darbam nepieciešamās vides sagatavošana

Vispārīga funkcija vai funkciju kopums, kurš sagatavo darba vidi tālāko darbību izpildīšanai, ielasa norādītās sistēmas konfigurācijas.

Ievade

Funkcijai tiek padota norāde uz konfigurācijas failu.

Apstrāde

Funkcija nolasa konfigurāciju failu un attiecīgi izveido, esošo tālākam darbam nepieciešamo failu struktūru, ja tā neeksistē.

Izvade

Izveidota nepieciešamā sistēmas failu struktūra.

FP-2 Testu skriptu failu meklēšana

Tiek atrasti un uzskaitīti visi testu skriptu faili, kurus būs nepieciešams apstrādāt.

Ievade

Funkcijai tiek padota norāde uz testa skriptu atrašanās vietu.

Apstrāde

Funkcija nolasa skriptu atrašanās vietas mapes saturu un sagatavo vidi to apstrādāšanai.

Izvade

Funkcija atgriež datus par atrastajiem testu skriptiem.

FP-3 Žurnāla izveide

Funkcija izveido žurnāla failu testa skriptam, kas tiks apstrādāts.

Ievade

Funkcijai padod apstrādājamā skripta nosaukumu.

Apstrāde

Tiek izveidots teksta fails, kurā tiks glabāta informācija par testa skriptā veiktajām izmaiņām.

Izvade

Sagatavots tukšs žurnāla fails.

FP-4 Žurnāla aizpildīšana

Funkcija aizpilda žurnāla failu ar informāciju par testa skriptā veiktajām izmaiņām.

Ievade

Funkcijai padod informāciju, ko nepieciešams pievienot žurnāla failam.

Apstrāde

Žurnāla failā tiek ierakstīta nepieciešamā informācija.

Izvade

Žurnāla failam pievienots ieraksts.

FP-5 Ievades failu veida atpazīšana

Funkcija atpazīst, kāda veida skripta fails ir jāapstrādā.

Ievade

Funkcijai padod skripta faila atrašanās vietu.

Apstrāde

Tiek atrasts tā paplašinājums, noteikts tā veids.

Izvade

Tiek atgriezti dati par faila tipu.

FP-6 Klases vai funkcijas definēšanas atpazīšana

Funkcija atpazīst, vai saņemtajos datos tiek veikta jaunas klases vai funkcijas parametru deklarēšana, atpazīstot, vai rindiņā ir atslēgas vārds vai *sub* un nav atslēgas vārdi *declare* vai *end*.

Ievade

Funkcijai tiek padoti ielasītās rindiņas dati.

Apstrāde

Funkcija atpazīst, vai padotajos datos tiek veikta klases vai funkcijas definēšana. Ja saņemtajos datos tiek definēta jauna klase vai funkcija, tiek testa fails ielasīts tālāk un tiek meklētas definēšanas beigas (funkciju parametrus mēdz rakstīt katru savā rindiņā).

Izvade

Ierakstīšanai failā tiek nodota rindiņa, kas, izpildot skriptu, izdrukās informāciju par pašreizējo klasi.

FP-7 Klases beigu atpazīšana

Funkcija atpazīst, vai saņemtajos datos tiek beigta klases aprakstīšana. Gadījumu jāatpazīst, jo tas informē arī par sekmīgām klases beigām.

Ievade

Funkcijai tiek padoti ielasītās rindiņas dati.

Apstrāde

Funkcija atpazīst, vai padotie dati satur informāciju par klases deklarēšanas beigām, meklējot tajā atslēgas vārdus.

Izvade

Ierakstīšanai failā tiek nodota rindiņa, kas, izpildot skriptu, izdrukās informāciju par klases beigšanos.

FP-8 Cita skripta izsaukšanas metodes atpazīšana

Funkcija atpazīst, vai saņemtajos datos tiek veikta cita testa skripta izsaukšana, izmantojot funkciju *callscript*.

Ievade

Funkcijai tiek padoti ielasītās rindiņas dati.

Apstrāde

Tiek pārbaudīts, vai rindiņa satur informāciju par citu skripta faila izsaukšanu, meklējot tajā funkcijas *callscript* izsaukumu. Ja šāda informācija atrasta, tiek sagatavota skripta rindiņa, ko ievietot skripta failā.

Izvade

Ierakstīšanai failā tiek nodota rindiņa, kas, izpildot skriptu, izdrukās informāciju par jauna skripta faila izsaukšanu.

FP-9 Rindiņas derīguma atpazīšana

Funkcijai jāatpazīst, vai saņemtā skripta rindiņa ir aizkomentēta un tiek izpildīta. Tas nepieciešams, jo daļā skriptu fragmenti ir aizkomentēti, jo ir pielāgoti kādai konkrētai testējamās sistēmas darbībai.

Ievade

Funkcijai tiek padoti ielasītās rindiņas dati.

Apstrāde

Funkcija atpazīst, vai padotā skripta rindiņa nav aizkomentēta.

Izvade

Funkcija atgriež vērtības:

- True – ja rindiņa ir aizkomentēta.
- False – ja rindiņa nav aizkomentēta.

FP-10 Rindiņas noformēšana

Funkcija domāta, lai, pirms pievieno testa skripta failos izdrukas funkcijas, pareizi noformētu pievienojamo rindiņu. Tas tiek darīts tāpēc, lai, pēc rindiņas pievienošanas tiktu saglabāta skripta lasāmība, ja, neskaidrību gadījumā, to ir nepieciešamas darīt pēc skripta izpildes.

Ievade

Funkcijai tiek padota pievienojamā izdrukas komanda un rindiņa, kura satur funkcijas izmantošanu.

Apstrāde

Tiek noteikts atkāpes izmērs padotajai testa skripta rindiņai un tāds pats tiek pievienots izdrukas komandai.

Izvade

Funkcija atgriež noformētu pievienojamo izdrukas rindiņu.

FP-11 Skripta identificēšanas izdrukas pievienošana

Funkcija domāta, lai izdrukātu žurnāla failā šī brīža strādājošā skripta nosaukumu. Tai jādarbojas arī izmantojot skriptu izsaukšanas metodi kādā citā skriptā (jāizdrukā jaunu informāciju par jauno skripta failu).

Ievade

Funkcijai tiek padota skripta klases sākšanas rindiņas dati.

Apstrāde

Tiek noskaidrots skripta nosaukums un tas pievienots skripta pirmkodam.

Izvade

Funkcija ir pievienojusi skriptam identificēšanas izdrukas.

FP-12 Izvēlnes atvēršanas atpazīšana

Funkcija atpazīst, vai saņemtā skripta rindiņa atver izvēlnes punktu, izmantojot *MenuSelect* metodi.

Ievade

Funkcijai tiek padoti ielasītās rindiņas dati.

Apstrāde

Tiek atpazīts, vai rindiņā tiek pielietota viena no izvēlnes punktu atvēršanas funkcijām. Ja tā tiek pielietota, tiek sagatavota rindiņa, kuru pievienot testa skripta failam.

Izvade

Ierakstīšanai failā tiek nodota rindiņa, kas, izpildot skriptu, izdrukās informāciju par izvēlnes punkta atvēršanu.

FP-13 Funkcionālo pogu nospiešanas atpazīšana

Funkcija atpazīst, vai padotajā rindiņā tiek izmantota kāda no funkcionālo pogu (F2, F3, F4) nospiešanas metodēm vai poga nospiesta tiešā veidā (izmantojot *sendkeys* SQABasic metodi).

Ievade

Funkcijai tiek padoti ielasītās rindiņas dati.

Apstrāde

Tiek atpazīts, vai rindiņā tiek pielietota kāda no funkcionālajām pogām. Ja tā tiek pielietota, tiek sagatavota rindiņa, kuru pievienot testa skripta failam.

Izvade

Ierakstīšanai failā tiek nodota rindiņa, kas, izpildot skriptu, izdrukās informāciju par pielietoto funkcionālo pogu.

FP-14 Pogu nospiešanas atpazīšana

Funkcija atpazīst, vai padotajā rindiņā tiek nospiesta poga, izmantojot *Buttonclick* vai *Pushbutton* metodi.

Ievade

Funkcijai tiek padoti ielasītās rindiņas dati.

Apstrāde

Tiek atpazīts, vai rindiņā tiek veikta kādas pogas nospiešana, pielietojot kādu no pogas nospiešanas funkcijām. Ja tā tiek pielietota, tiek sagatavota rindiņa, kuru pievienot testa skripta failam.

Izvade

Ierakstīšanai failā tiek nodota rindiņa, kas, izpildot skriptu, izdrukās informāciju pogas nospiešanu.

FP-15 Izvēles rūtiņas stāvokļa mainīšanas atpazīšana

Funkcija atpazīst, vai padotajā rindiņā tiek mainīts izvēles rūtiņas stāvoklis izmantojot *CheckBoxClick* metodi.

Ievade

Funkcijai tiek padoti ielasītās rindiņas dati.

Apstrāde

Tiek atpazīts, vai rindiņā tiek veikta stāvokļa mainīšana kādai izvēles rūtiņai. Ja tas tiek mainīts, tiek sagatavota rindiņa, ko satur izvēles rūtiņas identifikācijas datus, kuru pievienot testa skripta failam.

Izvade

Ierakstīšanai failā tiek nodota rindiņa, kas, izpildot skriptu, izdrukās informāciju par izvēles rūtiņu stāvokļa mainīšanu.

FP-16 Izvēles rūtiņas vērtības uzstādīšanas atpazīšana

Funkcija atpazīst, vai padotajā rindiņā tiek uzstādīts kādas izvēles rūtiņas stāvoklis, izmantojot *setCheckBox* metodi.

Ievade

Funkcijai tiek padoti ielasītās rindiņas dati.

Apstrāde

Tiek atpazīts, vai rindiņā tiek veikta noteikta stāvokļa uzstādīšana kādai izvēles rūtiņai. Ja tas tiek uzstādīts, tiek sagatavota rindiņa, kura satur izvēles rūtiņas identifikācijas datus un nomainīto stāvokli, ko pievienot testa skripta failam.

Izvade

Ierakstīšanai failā tiek nodota rindiņa, kas, izpildot skriptu, izdrukās informāciju par izvēles rūtiņu, kurai tika uzstādīts stāvoklis, kā arī par to, kāds ir jaunais stāvoklis.

FP-17 Cilnes atvēršanas atpazīšana

Funkcija atpazīst, vai padotajā rindiņā tiek atvērta kāda no cilnēm, izmantojot *TabControl* vai *SelectTab* metodi.

Ievade

Funkcijai tiek padoti ielasītās rindiņas dati.

Apstrāde

Tiek atpazīts, vai rindiņā tiek atvērta kāda no cilnēm. Ja tā tiek atvērta, tiek sagatavota rindiņa, kura satur cilnes identifikācijas datus, kuru pievienot testa skripta failam.

Izvade

Ierakstīšanai failā tiek nodota rindiņa, kas, izpildot skriptu, izdrukās informāciju par to, kura cilne tika atvērta.

FP-18 Funkciju pogas nospiešanas un darbības izsaukšanas atpazīšana

Funkcija atpazīst, vai padotajā rindiņā tiek izmantota darbību pogas izmantošana izmantojot darbību izsaukšanas metodi *PopupMenuSelect*.

Ievade

Funkcijai tiek padoti ielasītās rindiņas dati.

Apstrāde

Tiek atpazīts, vai rindiņā tiek pielietota viena no darbību pogas izvēlēšanās komandām. Ja tā tiek pielietota, tiek sagatavota rindiņa, kuru pievienot testa skripta failam.

Izvade

Ierakstīšanai failā tiek nodota rindiņa, kas, izpildot skriptu, izdrukās informāciju par darbību pogas izmantošanu un konkrēto izmantoto darbību.

FP-19 Izveidotās skriptu pirmkoda rindiņas ierakstīšana skripta failā

Funkcija veido jaunu skriptu pirmkodu, rakstot tajā jau esošās skripta rindiņas un pievienojot jaunās izdrukas.

Ievade

Funkcijai tiek padoti ielasītās rindiņas dati.

Apstrāde

Ja ir atpazīta kāda no skripta veiktajām darbībām grafiskajā saskarnē un izveidota izdruka par to, tā tiek nodota ierakstīšanai failā.

Izvade

Testa skripta failā tiek izdrukātas skriptu oriģinālā pirmkoda un izdruku rindiņas.

FP-20 Izpildot automātisko testu jāveido žurnāla fails par veiktajām darbībām

Funkcijām, kuras tiek ievietotas skriptu pirmkodā ir jāizveido žurnāla fails, kura nosaukumā ir skripta nosaukums un sākšanas laiks, lai tos būtu iespējams integrēt jau esošajā sistēmā.

Ievade

Funkcijai nav ievaddatu.

Apstrāde

Izpildot katru no izdrukas funkcijām, to ir nepieciešams pievienot skripta izdruku failā. Ja izdruku fails neeksistē, to jāizveido.

Izvade

Tiek izveidots izdruku fails un tajā tiek ierakstīta informācija par automātiskā skripta veiktajām darbībām.

FP-21 Izdruku ievietotājam jābūt savietojamam ar esošo skripta darbību žurnāla izveidotāju

Sistēmai jāspēj darboties un jābūt savietojamai ar esošo skripta pirmkodu izdrukāšanas rīku.

Ievade

Sistēmai pieejamie faili var būt arī iepriekš apstrādāti ar esošo rīku.

Apstrāde

Sistēma darbojas vienādi neatkarīgi no tā, vai tā tiek darbināt a pirms vai pēc esošā žurnāla failu izveidotāja.

FP-22 Skriptu veikto darbību attēlošanas vietnes datu iegūšana

Skriptu veikto darbību attēlošanas vietnei jāiegūst datus par skriptu laikā veiktajām darbībām

Ievade

Funkcijai ievaddatu nav.

Apstrāde

Tiek apstrādāti visi pieejamie skriptu veikto darbību žurnālu faili, ar to datiem tiek aizpildīta datu bāze.

Izvade

Attēlošanai ir pieejami jauni skriptu failu veikto darbību apraksti.

FP-23 Skriptu veikto darbību attēlošanas vietnes datu attēlošana

Skriptu veikto darbību attēlošana.

Ievade

Tiek saņemts pieprasījums par attēlojamo informāciju.

Apstrāde

Tiek atlasīti datu bāzes ieraksti, kas atbilst pieprasījumam

Izvade

Attēlo atlasītos datus.

1.3.2 Ārējās saskarnes prasības

Testa skriptu analizēšanas un izdruku pievienošanas rīkam nav nepieciešama saskarne, to jāvar palaist no komandrindas vai izmantojot tam speciāli izveidotu pakešdatni (*BAT* failu).

Atskaišu attēlošanas vietnes lietotāja saskarnei jābūt intuitīvai un viegli saprotamai. Jābūt iespējai izvēlēties noteiktus testa skriptus, kuru veiktās darbības attēlot.

1.3.3 Nefunkcionālās prasības

Programmatūrai jāatbalsta Ruby interpretatora 1.9.1.

Atskaišu attēlošanas sistēmai jādarbojas uz interneta pārlūku Internet Explorer 8.0, Mozilla Firefox 12.0, Google Chrome 18.0 vai analoģiskām versijām. Attēlošanai uz visām

pārlūkprogrammām jāstrādā līdzvērtīgi un to noformējumiem jābūt līdzīgiem, ja nav iespējams tos nodrošināt pilnīgi vienādus.

1.3.4 Veiktspējas prasības

Testa piemēru analizēšanas un izdruku pievienošanas rīkam netiek izvirzītas precīzas ātrdarbības prasības, jo testa skriptu nepieciešams modificēt un tam pievienojot izdrukas tikai pēc skripta izmaiņām vai uzlabojumiem.

PP-1 Visu testa skriptu apstrādes laiks

Visu testa skriptu apstrādei (~750) un izdruku pievienošanai nevajadzētu prasīt ilgāk kā 20 minūtes.

PP-2 Skriptu veikto darbību attēlošanas laiks

Atskaišu attēlošanas sistēmai vienkāršas darbības (pieejamu datu attēlošana pēc to izvēles) jāveic ne ilgāk kā 7 sekunžu laikā, pie nosacījuma, ka dati ir pieejami un nav problēmas ar tīkla pieslēgumu.

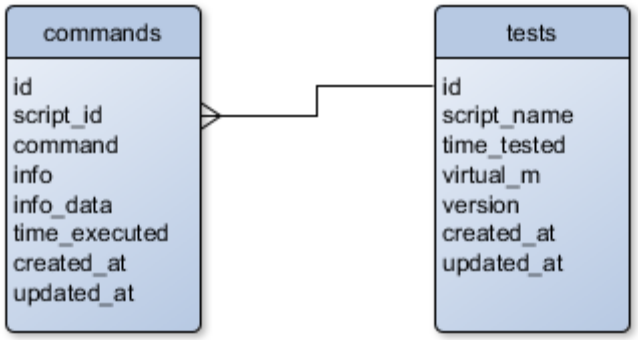
PP-3 Jaunu datu pievienošanas

Jaunu datu pievienošanai, atkarībā no pievienojamo datu daudzuma, vidēji nevajadzētu prasīt vairāk kā 10 minūtes.

1.3.5 Citas prasības

Datu bāze

Datu bāzes fiziskais ER modelis.



Att. 1.2 *datu bāzes fiziskais modelis*

Tabula commands

Tabula 1.2 *Tabulas commands lauki*

Lauka nosaukums	**Datu tips**	**Obligāts**	**Atslēga**	**Ierobežojumi, īpašas pazīmes**
Id	INTEGER	Jā	Primārā atslēga	RoR automātiski izveidots lauks
Script_id	INTEGER	Jā	Ārējā atslēga	
Command	VARCHAR(255)	Jā		
Info	VARCHAR(255)	Nē		
Info_data	VARCHAR(255)	Nē		
Time_executed	TIME	Nē		
Created_at	DATETIME	Jā		RoR automātiski izveidots lauks
Updated_at	DATETIME	Jā		RoR automātiski izveidots lauks

Tabula tests

Tabula 1.3 *Tabulas tests lauki*

Lauka nosaukums	**Datu tips**	**Obligāts**	**Atslēga**	**Ierobežojumi, īpašas pazīmes**
Id	INTEGER	Jā	Primārā atslēga	RoR automātiski izveidots lauks
Script_name	VARCHAR(255)	Jā		
Time_tested	TIME	Nē		
Virtual_m	VARCHAR(255)	Nē		
Version	VARCHAR(255)	Nē		
Created_at	DATETIME	Jā		RoR automātiski izveidots lauks
Updated_at	DATETIME	Jā		RoR automātiski izveidots lauks

2. PROGRAMMATŪRAS PROJEKTĒJUMA APRAKSTS

2.1 Ievads

2.1.1 Nolūks

Programmatūras projektējuma apraksta (PPA) nolūks ir aprakstīt, kā Programmatūras prasību specifikācijā izvirzītās prasības tiks realizētas produkta izstrādē. Dokuments paredzēts programmprodukta izstrādātājam, kā tehnisko detaļu apraksta materiāls produkta izstrādes laikā.

2.1.2 Darbības sfēra

Šajā dokumentā tiek aprakstīts testpiemēru pārklājuma analīzes sistēmas projektējums.

Programmatūras produkta mērķis ir veikt testpiemēru veikto darbību atpazīšanu un veidot atskaites par tām.

2.2 Saistība ar citiem dokumentiem

Programmatūras projektējuma apraksts ir kvalifikācijas darba „Testpiemēru pārklājuma analīzes sistēma" sastāvdaļa.

Programmatūras projektējuma apraksts ir izstrādāts balstoties uz TPAS programmatūras prasību specifikāciju un ir lietojams kopā ar to.

Programmatūras projektējuma apraksts ir izstrādāta saskaņā ar standartu LVS 72:1996 „Ieteicamā prakse programmatūras prasību aprakstīšanai" (2).

2.3 Dekompozīcijas apraksts

2.3.1 Ievads

Balstoties uz programmatūras specifikāciju un esošās situācijas izpēti, tika nolemts programmatūras kodu strukturēt šādi:

- Veido moduli Main, kurš pārvaldīs galvenās programmatūras funkcijas un izsauks tās no citiem moduļiem.
- Veidot moduli FileMod, kurš veiks darbības, kuras ir saistītas ar jaunu failu sameklēšanu, failu dzēšanu, modificēto failu izveidošanu.

- Veidot moduli ParseMod, kurš veiks failu sintaktisko analizēšanu, saturēs dažādas parsētāja funkcijas, saturēs funkciju kopumu, kuras nodrošinās rindiņas satura atpazīšanu un noteikšanu, vai tās veic darbības ar grafisko saskarni.
- Veidot moduli InsertMod, kurš nodrošinās izdruku pievienošanu skripta failiem SQABasic programmēšanas valodas pareizajā sintaksē, izmantojot pievienotās Rational Robot metodes, kā arī koda noformēšanas un informatīvo izdruku pievienošanu.
- Rational robot inpututils.sbl klasē pievienot funkcijas par žurnāla failu izveidi un aizpildīšanu.
- Veidot attēlošanas vietnes moduli ParseMod, kurš veiks skriptu veikto darbību failu sintaktisko analizēšanu, savāks un apkopos datus par tām.

2.3.2 Moduļu dekompozīcija

Starpdruku ievietotāja modulis – Main.rb

Nolūks

Galvenais modulis, kurš pārvalda pārējo procesu darbību.

Funkcija

Identificē apstrādājamos failus, pielieto tiem apstrādes funkcijas.

Starpdruku ievietotāja modulis – FileMod.rb

Nolūks

Pārvaldīt darbības ar regresijas skriptu failiem.

Funkcija

Izsauc funkcijas, lai dzēstu iepriekšējos skriptu failus, kuri atrodas mapē, kura saturēs jaunos skripta failus ar pievienotām papildus izdrukām. Iegūst un atgriež informāciju par jaunajiem skripta failiem, kuros būs jāievieto papildus izdrukas.

Atkarības

Modulis Main.

Starpdruku ievietotāja modulis – ParseMod.rb

Nolūks

Atbild par dažādām testu skriptu atpazīšanas funkcijām.

Funkcija

Satur saņemtā testa skripta rindiņas sintaktiskās analīzes funkciju kopu. Apstrādā saņemtos testa skriptu failus pa rindiņām, atpazīst un attiecīgi rīkojas, ja sastop atpazīstamās rindiņas testa skripta kodā. Izsauc funkcijas, kuras veido izdrukas par rindiņā esošo informāciju. Failā pievieno funkciju izsaukumu izdrukas, pēc to saņemšanas no *InsertMod* moduļa. Veido žurnāla failu un uzskaiti par veiktajām darbībām.

Veido jaunas rindiņas par saņemtajā testa skripta rindiņā atrodamajām (testējamajām) funkcijām tā, lai atgriezto vērtību varētu ievietot testa pirmkodā un tas būtu sintaktiski pareizs un, to izpildot, izdrukātu informāciju par rindiņā notiekošo. Veic arī dažas koda noformēšanas funkcijas.

Atkarības

Modulis Main.

Starpdruku ievietotāja modulis – InsertMod.rb

Nolūks

Atbild par sintaktiski pareizu Rational Robot izdrukas funkciju koda ģenerēšanu. Satur dažas koda noformēšanas funkcijas, kuras domātas, lai saglabātu testa skripta koda lasāmību.

Funkcija

Satur dažāda veida izdruku ģenerēšanas funkciju kopu. No saņemtās informācijas parametros atrod sev nepieciešamo informāciju par funkcijas izsaukumu, ģenerē un atgriež sintaktiski pareizu Rational Robot izdrukas funkcijas kodu. Rūpējas par to, lai ģenerētais kods netiktu izdrukāts vietā, kur tas sabojātu skriptu darbību. Satur vispārīgu informatīvu izdrukas izveidošanu un arī funkcijas koda noformēšanai.

Šis pievienos izdrukas.

Atkarības

Moduļi Main, ParseMod.

Atskaišu attēlošanas vietnes modulis – parsemod.rb

Nolūks

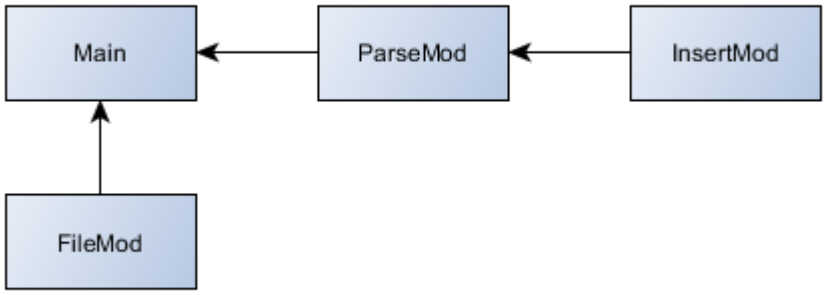
Atbild par dažādām testa veikto darbību žurnāla failu apstrādes funkcijām.

Funkcija

Satur saņemtā testa skriptu veikto darbību žurnāla failu sintaktiskās analīzes funkciju kopu. Apstrādā saņemtos failus. Iegūst skripta nosaukumu un testēšanas laiku. No rindiņām iegūst veiktās darbības aprakstu un veikšanas laiku. Ar iegūtajiem datiem aizpilda datu bāzi.

2.4 Atkarību apraksts

Teksta sintaktiskās analizēšanas rīka moduļu atkarības redzamas attēlā 2.1.



2.1 att. Teksta sintaktiskās analizēšanas rīka moduļu atkarības.

2.5 Saskarnes apraksts

Katram no moduļu dekompozīcijas nodaļā aprakstītajiem moduļiem saskarnes aprakstā norādītas to saturošās metodes – to izsaukšanas komanda, izpildītās prasības identifikators.

2.5.1 Modulis Main

Tabula 2.1 *Moduļa Main apraksts*

Izsaukšana	Prasības identifikators	Funkcija
-	FP-1	Sagatavo nepieciešamo darba vidi. Izveido mainīgos un ielādē moduļus.

2.5.2 Modulis FileMod

Tabula 2.2 *Moduļa FileMod apraksts*

Izsaukšana	Prasības identifikators	Funkcija
collectfiles	FP-2	Atrod visus skripta failus, kurus būs nepieciešams apstrādāt.
deletelogfiles	FP-1	Dzēš žurnālu failus par iepriekšējo skriptu modificēšanas reizi, lai to vietā radītu jaunus.
deleteoldfiles	FP-1	Dzēš iepriekšējos skriptu failus, ja tie palikuši modificēto skripta failu mapē.
readconfig	FP-1	Ielasa konfigurācijas failu un uzstāda parametrus.

2.5.3 Modulis ParseMod

Tabula 2.3 *Moduļa ParseMod apraksts*

Izsaukšana	Prasības identifikators	Funkcija
parsefiles	FP-3, FP-4	Galvenā parsētāja sākšanas funkcija. Atpazīst failos funkcijas, izsauc atbilstošu izdruku izveidošanas funkciju.
iscomment	FP-9	Nosaka, vai saņemtā rindiņa ir komentārs.
insertscriptname	FP-5, FP-6, FP-11	Pievieno identifikācijas izdrukāšanu skripta failā.
containsbuttonclick	FP-13	Nosaka, vai saņemtā rindiņa satur buttonclick metodes izsaukumu.
containscallscript	FP-8	Nosaka, vai saņemtā rindiņa satur cita skripta izsaukšanas nosaukumu.
containscbclick	FP-16	Nosaka, vai saņemtā rindiņa satur izvēlnes rūtiņas nospiešanu.
containskey	FP-13	Nosaka, vai saņemtā rindiņa satur funkcionālo pogu nospiešanu.
containsmenu	FP-12, FP-18	Nosaka, vai saņemtā rindiņa atver izvēlni.
containspushbutton	FP-13	Nosaka, vai rindiņa satur pogas nospiešanas metodi.
containsselecttab	FP-17	Nosaka, vai rindiņa satur izvēlnes aktivizēšanu.
containssetcb	FP-16	Nosaka, vai rindiņa satur izvēlnes rūtiņas vērtības uzstādīšanas metodi.
containssub	FP-11	Nosaka, vai rindiņa satur klases deklarēšanu.
containstabcontrol	FP-17	Nosaka, vai rindiņa satur tiešā veida cilnes aktivizēšanu.

2.5.4 Modulis InsertMod

Tabula 2.4 *Moduļa InsertMod apraksts*

Izsaukšana	Prasības identifikators	Funkcija
insertbtnoutput	FP-13	Izveido izdruku par tiešā veida pogas nospiešanu.
insertpushbuttonoutput	FP-13	Izveido izdruku par pogas nospiešanu.
insertchbclickoutput	FP-16	Izveido izdruku par izvēlnes rūtiņas vērtības mainīšanu.
insertsetchboutput	FP-16	Izveido izdruku par izvēlnes rūtiņas vērtības uzstādīšanu un tās uzstādīto vērtību.
insertkeyoutput	FP-13	Izveido izdruku par funkcionālā taustiņa nospiešanu.
insertmenuoutput	FP-12, FP-18	Izveido izdruku par izvēlnes atvēršanu.
insertselecttaboutput	FP-17	Izveido izdruku par cilnes aktivizēšanu.
inserttabctrloutput	FP-17	Izveido izdruku par tiešā veida cilnes aktivizēšanu.
addcallscript	FP-8	Izveido izdruku par nākošā skripta izsaukšanas metodes izmantošanu.
addspaces	FP-10	Atgriež tik tukšumzīmju saturošu string tipa mainīgo, cik satur saņemtā rindiņa (koda noformēšanai).
arraytodownc	FP-10	Atgriež saņemto masīvu pārveidojot visus burtus par mazajiem burtiem.
formline	FP-10	Noformē izveidoto izdrukas rindiņu tā pat, kā saņemto skripta pirmkoda rindiņu (koda noformēšanai).

2.5.5 Atskaišu attēlošanas vietnes modulis – parsemod

Tabula 2.5 *Moduļa parsemod apraksts*

Izsaukšana	Prasības identifikators	Funkcija
collectfiles	FP-19	Atrod visus skriptu veikto darbību failus, kurus būs nepieciešams apstrādāt.
getfileinfo	FP-19	Ielasa un apstrādā faila pirmo rindiņu, kas satur informāciju par skripta nosaukumu un izpildes sākšanas laiku.
parserezfiles	FP-20	Galvenā parsētāja sākšanas funkcija. Ielasa faila datus (ar funkciju getfileinfo) un izveido datu bāzē ierakstus par skripta nosaukumu un sākšanas laiku. Ielasa rindiņas (ar funkciju parseline) un datu bāzē pievieno ierakstus par skripta veiktajām darbībām.

getresultsdir	FP-19	Atgriež rezultātu failu mapes atrašanās vietu.
parseline	FP-20	Sintaktiski apstrādā faila rindiņu, no tās iegūstot darbības izpildes laiku, darbības nosaukumu un papildus parametrus, ja tādi ir.

2.6 Detalizētais apraksts

2.6.1 Modulis Main

Tiek ielādēti citi moduļi, uzstādīti vajadzīgie mainīgie un izsauktas apstrādājamo skriptu failu atrašanas un apstrādāšanas funkcijas.

2.6.2 Modulis FileMod

Funkcija collectfiles

Ievade

Viens *String* tipa arguments *scriptlocation*. Norāda uz skriptu failu atrašanās vietu.

Apstrāde

Tiek atlasīti visi *rec* un *sbl* faili, kas atrodas direktorijā. No visu failu saraksta tiek izņemtas *Modified* un *Excluded* mapes.

Izvade

Funkcija uz ekrāna izdrukā atrasto failu sarakstu un atgriež masīvu, kurš satur visus apstrādājamo failu nosaukumus.

Funkcija deletelogfiles

Ievade

Funkcijai nav ievaddatu.

Apstrāde

Funkcija dzēš žurnāla failus par iepriekšējā reizē veiktajām izmaiņām failos.

Izvade

Funkcija uz ekrāna izdrukā dzēsto failu sarakstu.

Funkcija deleteoldfiles

Ievade

Funkcijai nav ievaddatu.

Apstrāde

Tiek dzēsti iepriekšējie skriptu faili, ja tie vēl atrodas modificēto skriptu mapē.

Izvade

Funkcija uz ekrāna izdrukā dzēsto failu sarakstu un ziņo par sekmīgu darbības beigšanu.

Funkcija readconfig

Ievade

Funkcijai nav ievaddatu.

Apstrāde

Funkcija ielasa konfigurācijas faila saturu, uzstāda nepieciešamos mainīgos tālākai darbībai.

Izvade

Uzstādīti mainīgie tālākai darbībai.

2.6.3 Modulis ParseMod

Funkcija parsefiles

Ievade

Masīvs *allfiles*, kurš satur visu apstrādājamo skripta failu sarakstu.

Apstrāde

Pēc kārtas tiek ielasīti visi faili pa rindiņām. Katrai no rindiņām pēc kārtas tiek izsauktas funkcijas, kuras pārbauda, vai skripta rindiņā tiek veikta kāda darbība ar grafisko saskarni. Ja tā tiek veikta, tad tiek izsaukta attiecīgā funkcija, kura atbild par šī gadījuma aprakstošas Rational Robot funkcijas izsaukuma ģenerēšanu, ko pēc tam ierakstīt skripta failā.

Izvade

Modificēti testēšanas skripti, kuriem ir pievienotas darbību veikšanas izdrukas.

Funkcija iscomment

Ievade

Viens *string* tipa arguments *line* – testa skriptu faila rindiņa.

Apstrāde

Funkcija pārbauda, vai saņemtā rindiņa testa pirmkodā ir aizkomentēta.

Izvade

- False – ja rindiņa nav aizkomentēta.
- True – ja rindiņa ir aizkomentēta.

Funkciju kopums Contains

Tā kā funkcijas rindiņas satura atpazīšanai (containsbuttonclick, containscallscript, containscbclick, coontainskey, containsmenu, contains pushbutton, containsselecttab, containssetcb, containssub, containstabcontrol) ir līdzīga rakstura, aprakstu tās visas vienā sadaļā.

Ievade

Visu funkciju ievadē ir viens *string* tipa arguments *line* – testa skripta faila rindiņa.

Apstrāde

- Funkcija *containsbuttonclick* pārbauda, vai rindiņa satur *buttonclick* funkcijas izmantošanu.
- Funkcija *containscallscript* pārbauda, vai rindiņa satur *callscript* funkcijas izmantošanu.
- Funkcija *containscbclick* pārbauda, vai rindiņa satur *CheckBoxClick* funkcijas izmantošanu.
- Funkcija *containskey* pārbauda, vai rindiņa satur funkcionālo pogu nospiešanu.
- Funkcija *containsmenu* pārbauda, vai rindiņa satur *MenuSelect* vai *PopupMenuSelect* funkcijas izmantošanu.
- Funkcija *containspushbutton* pārbauda, vai rindiņa satur *PushButtonClick* funkcijas izmantošanu.
- Funkcija *containsselecttab* pārbauda, vai rindiņa satur *SelectTab* funkcijas izmantošanu.

- Funkcija *containssetcb* pārbauda, vai rindiņa satur *setCheckBox* funkcijas izmantošanu.
- Funkcija *containssub* pārbauda, vai rindiņa satur *sub* atslēgas vārdu un nesatur atslēgas vārdus *end* un *declare*.
- Funkcija *containstabcontrol* pārbauda, vai rindiņa satur *TabControlClick* funkcijas izmantošanu.

Izvade

Funkciju izvade ir:

- True – ja rindiņa satur funkciju, par kuru tikta veikta pārbaude.
- False – ja rindiņa nesatur funkciju, par kuru tikta veikta pārbaude.

2.6.4 Modulis InsertMod

Funkcija insertscriptname

Ievade

Divi *string* tipa argumenti:

- *line* – ielasītā testu skripta rindiņa;
- *logname* – apstrādājamā faila žurnāla faila nosaukums.

Apstrāde

Tiek pārbaudīts, vai šajā rindiņā var pievienot faila identificējošu rindiņu. Ja nē, failu ielasa tālāk un tiek atrasta vieta, kur to var izdarīt.

Izvade

Failā tiek ievietota izdruka, kas identificē skriptu pēc tā nosaukuma.

Funkciju kopums InsertOutput

Tā kā funkcijas testu skriptu pirmkoda ģenerēšanai (insertchbclickoutput, insertkeyoutput, insertpushbtnoutput, insertselecttaboutput, insertsetchboutput, inserttabctrloutput) ir līdzīgas savā darbībā, aprakstu tās visas vienā sadaļā.

Ievade

Visu funkciju ievadē ir viens *string* tipa arguments *line* – testa skripta koda rindiņa, no kuras tiks ņemta informācija testa skriptu pirmkoda rindiņas ģenerēšanai.

Apstrāde

- Funkcija *insertchbclickoutput* apstrādā rindiņas, kurās tiek veikta izvēlnes rūtiņas stāvokļa mainīšana un ģenerē skripta pirmkoda rindiņu par veikto darbību.
- Funkcija *insertkeyoutput* apstrādā rindiņas, kurās tiek veikta funkcionālo taustiņu nospiešana un ģenerē skripta pirmkoda rindiņu par veikto darbību.
- Funkcija *insertpushbtnoutput* apstrādā rindiņas, kurās tiek pogas nospiešana un ģenerē skripta pirmkoda rindiņu par veikto darbību.
- Funkcija *insertselecttaboutput* apstrādā rindiņas, kurās tiek izmantota cilnes aktivizēšanas funkcija un ģenerē skripta pirmkoda rindiņu par veikto darbību.
- Funkcija *insertsetchboutpu* apstrādā rindiņas, kurās tiek veikta izvēlnes rūtiņas stāvokļa uzstādīšana un ģenerē skripta pirmkoda rindiņu par veikto darbību.
- Funkcija *inserttabctrloutput* apstrādā rindiņas, kurās tiek veikta cilnes aktivizēšana un ģenerē skripta pirmkoda rindiņu par veikto darbību.

Izvade

Visas šīs funkcijas atgriež ģenerētu koda rindiņu pēc viena principa – rindiņas sākumā ir *coverConsoleOutput* funkcijas izsaukšana un tai seko informatīvs teksts, kurš katrai rindiņai atšķiras atkarībā no veiktās darbības:

- Funkcijai *insertchbclickoutput* teksts ir *CHECKBOX:*, kam seko izvēlnes rūtiņas nosaukums.
- Funkcijai *insertkeyoutput* teksts ir *KEY:*, kam seko nospiestā taustiņa nosaukums.
- Funkcijai *insertpushbtnoutput* teksts ir *BUTTON:*, kam seko nospiestās pogas nosaukums.
- Funkcijai *insertselecttaboutput* teksts ir *TABSELECT:*, kam seko cilnes nosaukuma teksts, tad vārds *as* un cilnes nosaukums.
- Funkcijai *insertsetchboutput* teksts ir *CHECKBOX_ON:*, kam seko izvēlnes rūtiņas nosaukums, ja tā tiek aktivizēta vai *CHECKBOX_OFF:*, kam seko izvēlnes rūtiņas nosaukums, ja tā tiek deaktivizēta.
- Funkcijai *inserttabctrloutput* teksts ir *TABSELECT:*, kam seko cilnes nosaukums un indekss.

Funkcija insertbtnoutput

Ievade

Masīvs *orglineparts,* kurš satur testa skripta rindiņu sadalītu masīva elementos, atdalītājsimbols ir tukšumzīme.

Apstrāde

Masīvs tiek apstrādāts un no tā tiek iegūta informācija par nospiesto pogu. Tiek ģenerēta skripta pirmkoda rindiņu par pogas nospiešanu.

Izvade

Tiek atgriezta testa skriptu pirmkoda rindiņa *coverConsoleOutput "BUTTON:"* kam tālāk seko pogas nosaukums vai tās mainīgā apzīmējums. Tādējādi, izpildot skriptu, tiks izdrukāts pogas nosaukums pat tad, ja skriptā tiks tam izmantots mainīgais.

Funkcija insertmenuoutput

Ievade

Masīvs *orglineparts,* kurš satur testa skripta rindiņu sadalītu masīva elementos, atdalītājsimbols ir tukšumzīme.

Apstrāde

Masīvs tiek apstrādāts un no tā tiek iegūta informācija par atvērto izvēlni vai darbības veikšanu. Tiek ģenerēta skripta pirmkoda rindiņa par veikto darbību.

Izvade

Tiek atgriezta testa skriptu pirmkoda rindiņa *coverConsoleOutput "MENU: "* izvēlnes atvēršanas gadījumā vai *coverConsoleOutput "POPUPMENU: "* darbības pogas izmantošanas gadījumā. Tālāk seko izvēlnes atvēršanas ceļš vai ceļa mainīgā apzīmējums. Tādējādi, izpildot skriptu, tiks izdrukāts atvēršanas ceļš pat tad, ja skriptā ceļa apzīmēšanai tiks izmantots mainīgais (bieži izmanto mainīgo ar nosaukumu *path*).

Funkcija addcallscript

Ievade

Viens *string* tipa arguments *line* – testa skripta koda rindiņa, no kuras tiks ņemta informācija testa skriptu pirmkoda rindiņas ģenerēšanai.

Apstrāde

Rindiņa tiek apstrādāta un no tās tiek iegūta informācija par izsauktā skripta nosaukumu.

Izvade

Tiek atgriezta testa skriptu pirmkoda rindiņa *coverConsoleOutput "CALLSCRIPT:" &* kurai seko izsaucamā skripta nosaukums.

Funkcija addspaces

Ievade

Viens *string* tipa arguments *line* – testa skripta koda rindiņa, no kuras tiks ņemta informācija testa skriptu pirmkoda rindiņas ģenerēšanai.

Apstrāde

Rindiņa tiek apstrādāta pa simbolam un tiek izveidots tukšumzīmju skaits tik pat, cik saņemtās rindiņas sākumā.

Izvade

Tiek atgriezts *string* tipa mainīgais, kurš sastāv no tukšumsimboliem.

Funkcija arraytodownc

Ievade

Masīvs *stringarray,* kurš satur testa skripta rindiņu sadalītu masīva elementos, atdalītājsimbols ir tukšumzīme.

Apstrāde

Katru masīva *string* elementu apstrādā un pārveido tā visus simbolus par mazajiem burtiem.

Izvade

Atgriež masīvu ar saņemtās rindiņas datiem.

Funkcija formline

Ievade

Viens string tipa parametrs *line*, viens masīvs *lineparts*.

Apstrāde

Masīva *lineparts* saturs tiek noformēts atbilstoši *line* noformējumam. Tiek pievienotas nepieciešamās tukšumzīmes.

Izvade

Tiek atgriezta viena *string* tipa vērtība, kura ir noformēta testa skripta pirmkoda rindiņa tā pat kā padotā argumenta *line* vērtība.

2.6.5 Atskaišu attēlošanas vietnes modulis – parsemod

Funkcija collectfiles

Ievade

Masīvs ar *string* tipa elementiem *rezdir*. Norāda uz testa skriptu veikto darbību žurnāla failu atrašanās vietām.

Apstrāde

Tiek atlasīti visi žurnāla faili un noformēti nepieciešamajā formātā.

Izvade

Funkcija atgriež masīvu, kurš satur visus apstrādājamo failu pilnos nosaukumus.

Funkcija getfileinfo

Ievade

Viens *string* tipa arguments *line.*

Apstrāde

Funkcija sintaktiski analizē saņemto žurnāla faila rindiņu. Rindiņā ir informācija par skripta nosaukumu un tā darbības sākšanas laiku.

Izvade

Funkcija atgriež *hash* tipa vērtību, kura satur divus elementus. Viena no tām ir *time_tested* – skripta darbības sākšanas laiks, otra ir *script_name* – skripta nosaukums.

Funkcija parserezfiles

Ievade

Masīvs – *rezfiles.* Satur apstrādājamo failu sarakstu.

Apstrāde

Katram failam tiek izsaukta metode *getfileinfo,* kas noskaidro skripta nosaukumu un izpildes laiku. Tālāk tiek ielasītas faila rindiņas, kurām tiek izsaukta metode *parseline,* kas atgriež informāciju par rindiņā veiktajām darbībām.

Izvade

Ar visu šo informāciju tiek aizpildīta datu bāze. Tiek pievienots ieraksts par skriptu un laiku, kad tas paveikts, kā arī tiek aizpildīta tabula par skriptā veiktajām darbībām.

Funkcija getresultsdir

Ievade

Funkcijai nav ievaddatu.

Apstrāde

Funkcija izveido masīvu ar visu automātisko testēšanas mašīnu testu darbību veikto žurnāla failu atrašanās vietām.

Izvade

Funkcija atgriež masīvu, kurš satur visu automātisko testēšanas mašīnu skriptu veikto darbību atrašanās vietas.

Funkcija parseline

Ievade

Viens *string* tipa arguments *line.*

Apstrāde

Katra rindiņa tiek sintaktiski analizēta, no tās tiek iegūta informācija par testa skripta darbības izpildīto laiku, darbības nosaukumu un aprakstu.

Izvade

Funkcija atgriež *hash* tipa vērtību, kura satur četrus elementus.

c*ommand* – veiktās darbības nosaukums.

Info – veiktās darbības apraksts.

info_data – papildus veiktās darbības apraksts.

timetested – laiks, kad darbība tika izpildīta.

3. TESTĒŠANAS DOKUMENTĀCIJA

Programmprodukta izstrāde notika pēc iteratīvā prototipēšanas modeļa, programmatūras testēšana notika paralēli izstrādes procesam, katra prototipa izveides laikā.

Ņemot vērā izstrādājamā produkta īpatnējo specifiku – nepieciešams ģenerēt citas programmēšanas valodas pirmkodu par veiktajām darbībām ar grafisko lietotāja saskarni – tika nolemts testēšanas procesu veikt izpildot testu skriptu analizēšanas un izdruku ievietošanas rīku, iegūstot modificētus testēšanas skriptu pirmkodus, pēc tam tika mēģināts šos skriptu pirmkodus kompilēt un meklēt, vai izdruku ievadīšanas process nav testos ieviesis kļūdas vai kādas citas nepilnības un vai tests izpilda tās pašas darbības, kā pirms izdruku pievienošanas, kā arī vai izdrukas tiek ievietotas pareizi arī pēc noformējuma.

Tika izveidots speciāls automātisko testu skripts, kurā tika ievietotas darbības, kuras nepieciešams atpazīt testa failu analizējot ar rīku, tādējādi vienkāršai visu funkciju notestēšanai atlika tikai šim skriptam pievienot izdrukas un to palaist.

3.1 Testu scenāriji

3.1.1 Testa skriptu fails Test TCCS.rec

Testa skripts, kuru izpildot tas piesakās sistēmā, atver izvēlni „Sistēma→Uzstādījumi→Uzskaites parametri”, atvērtajā logā izvēlas cilni „Algas”, apakšcilni „Parametri” un nomaina izvēlnes rūtiņas „Iekļaut izdienas pakāpes” vērtību. Pēc tam spiež pogu saglabāt un atkārtoti piesakās sistēmā.

Funkcijas tiek izpildītas izmantojot iebūvētās Rational robot metodes.

3.1.2 Testa skriptu fails Test TCCS2.rec

Testa skripts, kuru izpildot tas piesākas sistēmā, atver izvēlni „Sistēma→Uzstādījumi→Uzskaites parametri”, atvērtajā logā izvēlas cilni „Algas”, apakšcilni „Parametri” un nomaina izvēlnes rūtiņas „Iekļaut izdienas pakāpes” vērtību. Pēc tam spiež pogu saglabāt un atkārtoti piesākas sistēmā.

Funkcijas tiek izpildītas izmantojot testēšanas speciālistu izveidotas metodes kurām tiek padoti parametri, kas nosaka to darbības.

3.2 Testēšanas žurnāls

3.2.1 Dažādu veidu metožu izsaukšanas pārbaude

Testēšanai tika izmantoti iepriekš sagatavotie testēšanas faili, kas veic dažādas darbības. Testa skripti pirms izdruku ievietošanas izpildās pareizi.

Tabula 3.1 *Testa „Dažādu veidu metožu izsaukšanas pārbaude" dati*

Ievaddati	**Vēlamais rezultāts**	**Iegūtais rezultāts**
Test TCCS.rec	Testu skripts sekmīgi kompilējas pēc izdruku ievades.	Testu skripts sekmīgi kompilējas pēc izdruku ievades.
Test TCCS2.rec	Testu skripts sekmīgi kompilējas pēc izdruku ievades.	Testu skripts sekmīgi kompilējas pēc izdruku ievades.

3.2.2 Kļūdaina skripta apstrādāšanas tests

Testēšanai tika izmantoti iepriekš sagatavotie testēšanas faili. Otrajā testēšanas failā tika ar nodomu ieviesta kļūda.

Tabula 3.2 *Testa „Kļūdaina skripta apstrādāšanas tests" dati*

Ievaddati	**Vēlamais rezultāts**	**Iegūtais rezultāts**
Test TCCS.rec	Testu skripts sekmīgi kompilējas pēc izdruku ievades.	Testu skripts sekmīgi kompilējas pēc izdruku ievades.
Test TCCS2.rec	Testu skripts nekompilējas.	Testu skripts nekompilējas.

3.2.3 Skripta faila ar nepieejamu metodi izpildīšanas tests

Testēšanai tika izmantoti iepriekš sagatavotie testēšanas faili. Otrajā testēšanas failā netika iekļauta inpututils.sbh klase, kas nodrošina izdruku veidošanas funkciju iekļaušanu.

Tabula 3.2 *Testa* Skripta *faila ar nepieejamu metodi izpildīšanas tests" dati*

Ievaddati	**Vēlamais rezultāts**	**Iegūtais rezultāts**
Test TCCS.rec	Testu skripts sekmīgi kompilējas pēc izdruku ievades.	Testu skripts sekmīgi kompilējas pēc izdruku ievades.
Test TCCS2.rec	Testu skripts nekompilējas, trūkst funkcijas deklarācijas.	Testu skripts nekompilējas, trūkst funkcijas deklarācijas.

4. PIRMKODA FRAGMENTI

4.1 Moduļa *InsertMod* funkcija *insertscriptname*

```
  def insertscriptname(line, logname)
    # pārbauda, vai rindiņa satur deklarāciju, kā arī vai esošajā vietā
drīkst
    # pievienot izdruku vairāk-rindiņu deklarācijas gadījumā.
    # Pievieno izdruku par faila identificēšanu
    linecharary = line.chars.to_a
    if linecharary.include? '(' and not linecharary.include? ')'
      while tmpline = $curfile.gets and not tmpline.chars.to_a.include? ')'
        $newfile.write(tmpline)
      end
        $newfile.write(tmpline)
        #newfile.write('   coverConsoleOutput "STARTOF: ' + afile + '"' +
"\n")
    end


    # ieliek izdruku par to, kādu failu šobrīd apstrāda.
    #  Tālāk tas nepieciešams, lai ieliktu izdrukas pareizajā failā,
redzētu, kādu
    # failu skripts izsaucis.
    if File.extname(@filename) == ".rec"
      $newfile.write('  coverConsoleScriptName = "'+ logname + '"' + "\n")
      print '<'
      #izveidos izdruku par to failu, kur šobrīd tiek apstrādāts log
direktorijā
      $newfile.write('   coverConsoleOutput "INFILE: ' + $afile + '"' +
"\n")
    end
  end
```

4.2 Moduļa *InsertMod* funkcija *insertmenuoutput*

```
    def insertmenuoutput(orglineparts)
      lineparts = orglineparts
      # atgrieztajā stringā rindiņa pārveidota tā, ka tā satur menu ceļu,
ja
      #  tā  ir  menu  atvēršanas  komanda  un  darbību  ceļu,  ja  tas  ir
funkcionālā
      # taustiņa "darbības" komanda.

      for linepart in lineparts
        if linepart == 'MenuSelect'
          indx = lineparts.index('MenuSelect')
          if indx != nil
            lineparts[indx] = 'coverConsoleOutput "MENU: " + '
            #break
          end
        elsif linepart == 'PopupMenuSelect'
          indx = lineparts.index('PopupMenuSelect')
          if indx != nil
            lineparts[indx]= 'coverConsoleOutput "POPUPMENU: " + '
            #break
          end
        end
      end
      return lineparts
    end
```

4.3 Moduļa *InsertMod* funkcija *insertchbclickoutput*

```
    def insertchbclickoutput(line)
      # apstrādā rindiņas, kurās ir checkbox nospiešanas funckija rakstīta
      # "sliktajā" stilā, piem: CheckBox Click, "Name=cbExec;Type=CheckBox"
      matchrez = line.match(/"Name=.*;/i) #jāpārbauda vai nav uz type ar
"Type=CheckBox;Index=1"
      if not matchrez == nil
        rezstr = matchrez.to_s
        rezstr = rezstr.chomp(";")
        rezstr = rezstr.delete('"')
        rezstr = rezstr.gsub(/Name=/i, "")
        newline = 'coverConsoleOutput "CHECKBOX:  ' + rezstr + '"'
      elsif matchrez == nil
        # ļoti slikto gadījumu apstrāde, kad aprakstīts ar šādu stilu:
        # CheckBox Click, "Type=CheckBox;Index=2" (neko nevar pateikt)
        matchrez = line.match(/".*"/)
        rezstr = matchrez.to_s
        rezstr = rezstr.gsub(/"/, "")
        newline = 'coverConsoleOutput "CHECKBOX:  ' + rezstr + '"'
      end

      if newline == nil
        newline = 'coverConsoleOutput "WARNING! CHECKBOX INFO NOT FOUND!"'
      end

      return newline
    end
```

4.4 Moduļa *ParseMod* funkcija *containssub*

```
  def containssub(line)
    # funkcija pārbauda, vai padotā rindiņa satur jaunas klases deklarēšanu
    # Atgriež true, ja satur, false - ja nesatur.
    if line =~/sub\s/i and not line =~ /end/i
     # rindiņa satur sub nosaukums (parametri), bet nesatur end (end sub ir
deklarēšanas beigas)
      if not line =~ /declare/i
        # tā pat jāpārbauda, vai tā nav priekš-deklarēšana (uzskaitīti
tikai parametri header failā)
        return true
      end
    end
    return false
  end
```

5. PROJEKTA ORGANIZĀCIJA

Testpiemēru pārklājuma analīzes sistēmas izstrāde notika atbilstoši prototipēšanas modelim, kurā lielākais uzsvars tiek likts uz ātru programmatūras prototipu izveidi un funkcionalitātes ieviešanu un pielāgošanu.

Sākumā tika formulētas vispārīgas prasības par produkta funkcionalitāti. Pēc prasību formulēšanas tika izstrādāta programmatūras prasību specifikācijas pirmā versija. Balstoties uz to, tika izveidots pirmais programmatūras projektējums un izstrādāts pirmais teksta sintaktiskās analizēšanas rīka prototips. Pēc tā izveides tika noskaidrotas precīzākas detaļas par to, kādas vēl darbības tam jāveic un tika papildināta programmatūras prasību specifikācija un projektējums, pēc kurā, arī tika turpināta produkta izstrāde. Katrā iterācijā tika papildināta un uzlabota rīka funkcionalitāte atbilstoši jaunajām prasībām.

Kad sintaktiskās skriptu pirmkoda analizēšanas rīks tika izveidots tik tālu, ka tā pamata funkcionalitāte strādā, tika veikti pirmie apjomīgie testi. Pēc pirmo testu veikšanas tika iegūti dati tālākai sistēmas izstrādei, jo tika iegūti automātisko testēšanas skriptu darbību apraksti. Tā kā tīmekļa vietnes funkcionalitāte ir diezgan vienkārša, tās pamata funkcijas tika izveidotas jau pirmajā prototipa versijā un vēlāk pielāgotas vieglākai konfigurēšanai.

Programmprodukta izstrādē tika izmantota programmēšanas vide NetBeans 6.9.1 teksta sintaktiskās analizēšanas rīka izveidei, Ruby on Rails ietvars tīmekļa vietnes izstrādei, datu bāzu pārvaldības rīks SQLite administrator, teksta redaktors Notepad++ un automātisko skriptu izstrādes rīks Rational Robot.

Programmatūras pirmkoda un dokumentācijas konfigurāciju pārvaldībai tika izmantota Subversion versiju pārvaldības sistēma, lietojot grafiskās vides rīku TortoiseSVN.

6. KONFIGURĀCIJU PĀRVALDĪBA

Programmatūras produkta pirmkoda un dokumentācijas konfigurāciju pārvaldībai tika izmantota Subversion versiju pārvaldības sistēma, lietojot grafiskās vides rīku TortoiseSVN (8).

Konfigurāciju repozitorijs tika izveidots uz tīkla diska, kura rezerves kopijas tiek veidotas katru dienu, nodrošinot pirmkoda glabāšanu gan uz cietā diska gan tīkla diskā, tādējādi paaugstinot drošību un koda atgūšanas iespēju, ja kāda no diskiem vairs nebūtu pieejams vai tiktu bojāts.

Veicot konfigurācijas saglabāšanu (*commit*) tika pievienots komentārs par veiktajām izmaiņām kopš iepriekšējās konfigurācijas saglabāšanas, lai būtu vieglāk noteikt un sekot programmatūras uzlabojumiem un papildinājumiem pirmkodā. Konfigurāciju saglabāšana tiek veikta tikai brīžos, kad ir pabeigtas vēlamās izmaiņas un pirmkods kompilējas un ir izpildīti testi, kas minēti nodaļā testēšanas .

7. KVALITĀTES NODROŠINĀŠANA

Lai nodrošinātu izstrādātās sistēmas kvalitāti, tika nolemts ievērot šādus noteikumus:

- programmatūras prasību specifikācija tika izstrādāta atbilstoši valsts izvirzītajiem standartiem (1);
- programmatūras projektējuma apraksts tika izstrādāts atbilstoši valsts izvirzītajiem standartiem (2);
- pirmkods izstrādāts atbilstoši Latvijas Universitātē izstrādātajiem ieteikumiem par labo programmēšanas stilu (4);
- pirms programmatūras izstrādes tika apgūtas Ruby programmēšanas īpatnības un ieteicamais stils (5, 6);
- veidojot testu skriptu darbību atpazīšanas metodes tās tika izstrādātas tādā veidā, lai tās būtu viegli pielāgojamas arī kādai citai automātisko skriptu rakstīšanas valodai, jo darbību atpazīšanas funkcijas tika izdalītas atsevišķi;
- veidojot testu skriptu darbību atpazīšanu tika nolemts to darīt cenšoties izmanto regulārās izteiksmes (*regexp*), lai būtu pēc iespējas vieglāk izmainīt metodes testu kodu izmaiņu gadījumā (7);
- izstrādātā programmatūra tika testēta atbilstoši nodaļā „Testēšanas dokumentācija" atbilstošajai kārtībai.

8. DARBIETILPĪBAS NOVĒRTĒJUMS

Darbietilpību prognozēju izmantojot funkcijpunktu aprēķināšanas modeli, tā izmantošanas apraksts apgūts programminženierijas kursā (9). Tabulā 8.1 attēlots funkcijpunktu sadalījums.

*Tabula 8.1*Funkcijpunktu sadalījums

Prasības identifikators	Prasības nosaukums	Iedalījums	FP
FP-1	Darbam nepieciešamās vides sagatavošana	Ievads	3
FP-2	Testu skriptu failu meklēšana	Ievads	3
FP-3	Žurnāla izveide	Izvads	4
FP-4	Žurnāla aizpildīšana	Izvads	4
FP-5	Ievades failu veida atpazīšana	Vaicājums	3
FP-6	Klases vai funkcijas definēšanas atpazīšana	Vaicājums	3
FP-7	Klases beigu atpazīšana	Vaicājums	3
FP-8	Cita skripta izsaukšanas metodes atpazīšana	Vaicājums	3
FP-9	Rindiņas derīguma atpazīšana	Vaicājums	3
FP-10	Rindiņas noformēšana	Vaicājums	3
FP-11	Skripta identificēšanas izdrukas pievienošana	Ievads	3
FP-12	Izvēlnes atvēršanas atpazīšana	Vaicājums	3
FP-13	Funkcionālo pogu nospiešanas atpazīšana	Vaicājums	3
FP-14	Pogu nospiešanas atpazīšana	Vaicājums	3
FP-15	Izvēles rūtiņas stāvokļa mainīšanas atpazīšana	Vaicājums	3
FP-16	Izvēles rūtiņas vērtības uzstādīšanas atpazīšana	Vaicājums	3
FP-17	Cilnes atvēršanas atpazīšana	Vaicājums	3
FP-18	Funkciju pogas nospiešanas un darbības izsaukšanas atpazīšana	Vaicājums	3
FP-19	Izveidotās skriptu pirmkoda rindiņas ierakstīšana skripta failā	Izvads	4
FP-20	Izpildot automātisko testu jāveido žurnāla fails par veiktajām darbībām	Izvads	4
FP-21	Izdruku ievietotājam jābūt savietojamam ar esošo skripta darbību žurnāla	Vaicājums	3
FP-22	Skriptu veikto darbību attēlošanas vietnes datu iegūšana	Ievads	3
FP-23	Skriptu veikto darbību attēlošanas vietnes datu attēlošana	Izvads	4
-	Skriptu attēlošanas vietnes datu bāze	Iekšējais fails	7

Prasības identifikators	Prasības nosaukums	Iedalījums	FP
-	Testu skriptu pirmkodi (ap 750)	Ārējais fails	10
		Kopā	91

Šīs prasības kopā veido 91 nepielāgotu funkcijpunktu.

Tālākiem programmatūras izstrādes darbietilpības aprēķiniem nepieciešams zināt rindiņu skaits vienam funkcijpunktam programmēšanas valodā Ruby. Tā kā neizdevās atrast vidējo funkcijpunktu rindiņu skaitu šai programmēšanas valodai, apskatīju līdzīgas programmēšanas valodas (Perl, Java, Visual Basic) un aprēķinos ieguvu aptuveni rezultātu 22*83=1826 pirmkoda rindiņas (*SLOC*).

Tā kā darbietilpība aprēķināta balstoties uz PPS un PPA, neizmantojamā (pārstrādātā) koda apjoms varētu būt līdz 15%. Pielietojot šo korekciju iegūstam aptuveni 1552 koda rindiņas. Jāpiebilst gan, ka izmantojot iteratīvās izstrādes modeli pārstrādāto koda rindiņu skaits varētu būt arī lielāks, ja izstrādes gaitā tika papildinātas prasības.

Lietojot tiešsaistē pieejamo COCOMO II (10) aprēķina kalkulatoru ievadot 8.2 tabulā redzamos parametrus iegūstam aptuveno novērtējumu 3.79 personmēneši.

Tabula 8.2 COCOMO II ievadītie parametri

Parametrs	Ļoti zems	Zems	Vidējs	Augsts	Ļoti augsts	Ekstrēmi augsts
Product Attributes						
Required Reliability			X			
Database Size		X				
Product Complexity				X		
Computer Attributes						
Execution Time Constraint			X			
Main Storage Constraint			X			
Virtual Machine Volatility				X		
Computer Turnaround Time		X				
Personnel Attributes						
Analyst Capability		X				
Applications Experience			X			
Programmer Capability				X		

Parametrs	Ļoti zems	Zems	Vidējs	Augsts	Ļoti augsts	Ekstrēmi augsts
Virtual Machine Experience			X			
Programming Language Experience		X				
Project Attributes						
Modern Programming Practices			X			
Use of Software Tools			X			
Required Development Schedule			X			

Realizētā koda apjoms ir aptuveni 720 rindiņas. Atšķirību starp COCOMO II modeļa un reālo rindiņu skaitu varētu ietekmēt dažādi faktori, kā piemēram nav pieejams precīzs Ruby programmēšanas valodas funkcijpunkta rindiņu skaits, kā arī COCOMO II modelis nav piemērots maza apjoma programmatūrai. Kā arī koda izstrādes laikā tika izmantota regulāro izteiksmju bibliotēka priekš teksta apstrādes, kura ļauj ļoti sarežģītus nosacījumu aprakstīt vienā koda rindiņā.

REZULTĀTI

Darba izstrādes gaitā ir izveidots rīks automātisko regresijas testu skriptu pirmkodu izdruku ievietošanai un vietne veikto darbību apkopošanai un testa skriptu veikto darbību attēlošanai.

Izveidotais rīks spēj apstrādāt visus skripta failus, kuri tiek izmantoti regresa testēšanai. Ievietotās izdrukas nesabojā skriptu pieraksta sintaksi, tos ir iespējams kompilēt.

Izpildot testu skriptus tie izveido žurnāla failu par veiktajām darbībām. Šo žurnāla failu ir ērti izmantot skriptu veikto darbību noskaidrošanai un atkārtot skripta atrastās kļūdas.

Skripta izpildīto darbību sarakstu ir iespējams apskatīt izveidotajā vietnē.

Izstrādes laikā tika apgūta programmēšanas valoda Ruby, versiju kontroles rīks Subversion izmantojot grafiskās pārvaldības rīku TortoiseSVN. Projekta izstrādes laikā apguvu arī programmēšanas valodu SQABasic, ar kuru tiek programmēti testēšanas skripti. Tika gūta vērtīga pieredze programmatūras dokumentēšanas izstrādē.

SECINĀJUMI

Ņemot vērā, ka iepriekšējās pieredzes ar programmēšanas valodu Ruby nebija, tas netraucēja programmatūras izstrādē, jo šī programmēšanas valoda ir gana viegli saprotama un ļauj darbības veikt dažādos veidos, kas līdzinās citām programmēšanas valodām.

Programmprodukta specifiskās funkcionalitātes dēļ programmatūras testēšanu veikt bija ļoti vienkārši, jo kļūdas gadījumā testēšanas skriptus vairs nebija iespējams nokompilēt un tika paziņots par kļūdu.

Testu skriptu pirmkoda izdruku ievietošanas rīkam jau atrasts tālāks pielietojums – tas tiks integrēts esošajā sistēmā, kas ļaus testu veikto darbību aprakstu pievienot jau šobrīd automātiski veidotajiem pieteikumiem par testu atrastajām kļūdām. Pieteikumi tiek reģistrēti uzdevumu un kļūdu izsekošanu sistēmā JIRA, šobrīd tiem tiek pievienots kļūdas paziņojums un aptuvena atrašanās vieta, taču, pievienojot arī testa veikto darbību aprakstu kļūdas atrašanās vietu un cēloni būs daudz vieglāk, kas arī ir lielākais guvums no šīs sistēmas izstrādes.

Testu darbību attēlošanas vietni nolemts tālāk integrēt jau esošajā automātiskajā testu izpildes rīkā, kas ļaus arī iegūt precīzāku informāciju par skriptu un moduļu sasaisti. Tā kā vietni būs īpaši jāpielāgo esošajai funkcionalitātei (jānomaina arī datu bāzi no SQLite uz MSSQL), šobrīd tā atstāta prototipa stadijā.

IZMANTOTĀ LITERATŪRA UN AVOTI

1. LVS 68:1996 „Programmatūras prasību specifikācijas ceļvedis”
2. LVS 72:1996 „Ieteicamā prakse programmatūras prasību aprakstīšanai”
3. Noslēguma darbu izstrādāšanas un aizstāvēšanas kārtība [tiešsaiste] [atsauce – 17.04.2012] http://estudijas.lu.lv/mod/resource/view.php?id=3495
4. Programmēšanas stils [tiešsaiste] [atsauce – 26.03.2012] http://estudijas.lu.lv/mod/resource/view.php?id=44939
5. *Little book of Ruby*. [e-grāmata] [atsauce – 26.03.2012] Pieejams: http://www.sapphiresteel.com/IMG/pdf/LittleBookOfRuby.pdf
6. Ruby Programming [tiešsaiste] [atsauce – 28.03.2012] http://en.wikibooks.org/wiki/Ruby_programming_language
7. Regexp bibliotēka [tiešsaiste] [atsauce – 14.04.2012] http://www.ruby-doc.org/core-1.9.3/Regexp.html
8. TortoiseSVN [tiešsaiste] [atsauce – 26.03.2012] http://tortoisesvn.tigris.org/
9. COCOMO lietošanas piemērs [tiešsaiste] [atsauce – 21.05.2012] http://estudijas.lu.lv/mod/resource/view.php?id=128556
10. COCOMO II - Constructive Cost Model [tiešsaiste] [atsauce – 21.05.2012] http://diana.nps.edu/~madachy/tools/COCOMO.php

DOKUMENTĀRĀ LAPA

Kvalifikācijas darbs „***Testpiemēru pārklājuma analīzes sistēma***” izstrādāts Latvijas Universitātes Datorikas fakultātē.

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

Autors: ***Muntis Rudzītis*** _____ **___.05.2012**.

Rekomendēju darbu aizstāvēšanai

Darba vadītāja: *Dr. mat. Anda Ādamsone* _____ ___.05.2012.

Recenzents: M. Dat. Jana Ceriņa-Bērziņa

Darbs iesniegts 24.05.2012.

Kvalifikācijas darbu pārbaudījumu komisijas sekretārs: *Dainis Dosbergs* _____

Darbs aizstāvēts kvalifikācijas darbu pārbaudījuma komisijas sēdē

___.06.2012. prot. Nr. _____, vērtējums _____ (_____)

Komisijas sekretārs(-e): _____