RĪGAS TEHNISKĀ UNIVERSITĀTE
Datorzinātnes un informācijas tehnoloģijas fakultāte
Lietišķo datorsistēmu institūts

Artūrs BARTUSEVIČS
Doktora studiju programmas «Datorsistēmas» doktorants

PROGRAMMATŪRAS KONFIGURĀCIJAS PĀRVALDĪBAS MODEĻVADĀMU RISINĀJUMU IZSTRĀDE UN REALIZĀCIJA

Promocijas darba kopsavilkums

Zinātniskais vadītājs
profesors *Dr. habil. sc. ing.*
LEONĪDS NOVICKIS

RTU Izdevniecība
Rīga 2015

Bartusevičs A. Programmatūras konfigurācijas pārvaldības modeļvadāmu risinājumu izstrāde un realizācija. Promocijas darba kopsavilkums. – R.: RTU Izdevniecība, 2015. – 50 lpp.

Iespiests saskaņā ar DITF LDI padomes 2015. gada 8. aprīļa lēmumu Nr. 12300-4.1/2.

Darbs izstrādāts ar daļēju valsts pētījumu programmas *SOPHIS* atbalstu līgumā Nr. 10-4/VPP-4/11.

ISBN 978-9934-10-725-2

PROMOCIJAS DARBS
IZVIRZĪTS INŽENIERZINĀTŅU
DOKTORA GRĀDA IEGŪŠANAI RĪGAS TEHNISKAJĀ
UNIVERSITĀTĒ

Promocijas darbs inženierzinātņu doktora grāda iegūšanai tiek publiski aizstāvēts 2015. gada 21. septembrī plkst. 14.30 Rīgas Tehniskās universitātes Datorzinātnes un informācijas tehnoloģijas fakultātē, Meža ielā 1, 3. korpusā, 202. auditorijā.

OFICIĀLIE RECENZENTI:

Profesors *Dr. habil. sc. ing.* Jānis Osis
Rīgas Tehniskā universitāte, Rīga, Latvija

Profesors *Dr. sc. ing.* Artis Teilāns
Rēzeknes Augstskola, Latvija

Asoc. profesors *Dr. sc. comp.* Antanas Mitašiunas
Viļņas Universitāte, Lietuva

APSTIPRINĀJUMS

Apstiprinu, ka esmu izstrādājis šo promocijas darbu, kas iesniegts izskatīšanai Rīgas Tehniskajā universitātē inženierzinātņu doktora grāda iegūšanai. Promocijas darbs zinātniskā grāda iegūšanai nav iesniegts nevienā citā universitātē.

Artūrs Bartusevičs ……. (paraksts)

Datums: ……

Promocijas darbs ir uzrakstīts latviešu valodā, tajā ir ievads, 5 nodaļas, secinājumi, literatūras saraksts, 2 pielikumi, 55 zīmējumi un ilustrācijas, 30 tabulas, kopā 228 lappuses. Literatūras sarakstā ir 115 nosaukumi.

SAĪSINĀJUMI

MTM	*Model — Transformation — Model* (angļ. val.)
EAF	*Environment — Action — Framework* (angļ. val.)
MDD	*Model-Driven Development* (angļ. val.)
MDA	*Model-Driven Architecture* (angļ. val.)
EM	*Environment Model* (angļ. val.)
PIAM	*Platform Independent Action Model* (angļ. val.)
PSAM	*Platform Specific Action Model* (angļ. val.)
SCBM	*Source Code Branching Model* (angļ. val.)
SM	*Service Model* (angļ. val.)
PIEM	*Platform Independent Environment Model* (angļ. val.)
CM	*Code Model* (angļ. val.)
CIM	*Computing Independent Model* (angļ. val.)
PIM	*Platform Independent Model* (angļ. val.)
PSM	*Platform Specific Model* (angļ. val.)

SATURS

IEVADS

2009. gadā konferencē «*Velocity Conference*» tika prezentēts Džona Allspava (*John Allspaw*) un Pola Hamonda (*Paul Hammond*) referāts «10 instalācijas vienā dienā» (angļ. val. *10 Deploys A Day*). Referātā bija uzsvērta problēma, ka, attīstoties spējo programmatūras izstrādes metodoloģijai (angļ. val. *Agile*) un mākoņskaitļošanas tehnoloģijām, operācijas, kas sagatavo programmatūras būvējumus un laidienus, nespēj savlaicīgi piegādāt pasūtītājam gatavu produktu [CON 2015]. Minēta konference tiek uzskatīta par sākumu *DevOps* metodoloģijai, kuras mērķis ir paātrināt programmatūras būvējumu veidošanu un instalācijas, ka arī uzlabot to kvalitāti [AZO 2014]. Treisija Reigane (*Tracy Ragan*) savā rakstā [RAG 2014] iezīmē mūsdienīgas būvējumu un instalācijas rīku attīstības tendences. Rīkiem, kas atbalsta programmatūras būvējumus un instalācijas, jābūt modeļvadāmiem, jo, attīstoties makoņskaitļošanas tehnoloģijām, statiski skripti vairs nevar nodrošināt ātru un efektīvu programmatūras būvējumu un instalāciju mākoņos [RAG 2014]. Pēdējā laikā tirgū parādījās daudz rīku, kas atbalsta modeļvadāmu programmatūras būvējumu un instalāciju, piemēram, *Serena* un *Open Make* kompānijas produkti, ka arī daudzi citi [AZO 2014].

Konfigurācijas pārvaldības vadošie speciālisti [УДО 2011, AIE 2010] atzīmē, ka mūsdienās jaunie programmatūras izstrādes projekti attīstās ļoti strauji, tāpēc katrā jaunajā projektā nepieciešams pēc iespējas ātrāk ieviest automatizācijas procesus, kas sniedz kvalitatīvu atbalstu programmatūras būvējumu un instalācijas procesam.

Šobrīd lielāka daļa no rīkiem koncentrējas vien uz programmatūras būvējumu un instalāciju, taču pievērš maz uzmanības citiem procesiem, kas tieši ietekmē programmatūras būvējumu. Programmatūras konfigurācijas pārvaldība ir disciplīna, kas apskata visus procesus, kas ietekmē programmatūras būvējumu, atbilstoša būvējuma instalāciju un piegādi pasūtītājam. Kā norāda nozares vadošie speciālisti [AIE 2010, MET 2002, УДО 2011], uzbūvēt no izejas koda kvalitatīvu programmatūru ir iespējams tikai tad, ja kvalitatīvi tiek organizēti visi konfigurācijas pārvaldības procesi kopumā. Tāpēc šajā promocijas darbā pēc iespējas plašāk tiks apskatīts konfigurācijas pārvaldības jēdziens, lai identificētu pēc iespējas vairāk faktoru, kas ietekmē programmatūras būvējumus.

Analizējot mūsdienīgus konfigurācijas pārvaldības automatizācijas risinājumus un to attīstības tendences, jāatzīst kā risinājumi tiecās uz modeļvadāmas arhitektūras formātu (angļ. val. *MDA — Model-Driven Architecture*). Pirmkārt, modeļvadāma pieeja, ko piedāvā *MDA*, ļauj samazināt cilvēcisku faktoru pārejot no prasībām pie implementācijas [OSE 2011]. Otrkārt, attīstoties mākoņskaitļošanas tehnoloģijām, statiski programmatūras būvējumu skripti vairs neder, jo risinājums atrodas mākoņos un skripti nevar operēt ar absolūtām serveru adresēm un citām saistītām vērtībām [RAG 2014].

Tēmas aktualitāte

Konfigurācijas pārvaldības procesa būtiskākais rezultāts ir no izejas koda uzbūvēta programmatūra, kas tiek piegādāta pasūtītājam. Lai to paveiktu, konfigurācijas pārvaldības disciplīna veic programmatūras izejas koda pārvaldību un no tā uzbūvē strādājošu programmatūru. Ja kāda no šim darbībām notiek neveiksmīgi un pasūtītājs saņem nestrādājošu vai nekvalitatīvu programmatūru, zūd pievienotā vērtība attiecīgam programmatūras izstrādes projektam. Programmatūras izstrādes nozares kvalitātes standarti pieprasa sakārtotu un automatizētu konfigurācijas pārvaldību [AIE 2010]. Avotā [УДО 2011] ir minēts, ka viens no konfigurācijas pārvaldības procesa aktualitātes apliecinājumiem ir fakts, ka *CMMI* (angļ. val. *Capability Maturity Model Integration*) standartā konfigurācijas pārvaldības process ir tik pat svarīgs kā sakārtots izstrādes un testēšanas process.

Problēmas nostādne

21. gadsimtā, kad strauji attīstās spējo programmatūras izstrādes pieeja un tiek izstrādātas apjomīgas un sarežģītas programmatūras, bieži jauna projekta sākums ir līdzīgs sprādzienam. Jau pēc dažam dienām programmatūras pasūtītājs grib saņemt pirmo programmatūras versiju. Tikmēr formāls un automatizēts process, kas uzbūvē programmatūru, vēl nav gatavs. Rodas tā saucamais «meistara faktors», kad viens konkrēts speciālists prot sagatavot programmatūras laidienu no lokālas darbstacijas, izmantojot vien savas praktiskas iemaņas. Šāda situācija vēlāk izraisa neparedzētas kļūdas konfigurācijas pārvaldības procesā, ka arī process kļūst ļoti atkarīgs no konkrēta cilvēka kompetences.

Mūsdienās trūkst zinātniski pamatotu pieeju konfigurācijas pārvaldības procesu automatizācijas ieviešanai, kas izmantotu formālu un stingri definētu ceļu no procesa prasībām līdz implementācijai. Papildus tam implementācijas stadijā jāprot atkārtoti izmantot jau esošās implementācijas atsevišķām procesa daļām. Tas varētu paātrināt konfigurācijas pārvaldības procesu automatizācijas ieviešanas laiku, jo no nulles vajadzētu izstrādāt tikai konkrētam projektam specifiskās daļas, nevis pilno automatizācijas implementāciju.

Promocijas darba mērķis

Promocijas darba mērķis ir izstrādāt modeļvadāmu pieeju un metodoloģiju konfigurācijas pārvaldības procesu automatizācijas ieviešanai, kas ļauj samazināt automatizācijas ieviešanas laiku un uzlabot automatizācijas kvalitāti.

Modeļvadāma pieeja konfigurācijas pārvaldības procesu automatizācijas ieviešanai parāda, kā ar modeļu palīdzību var automātiski iegūt izejas kodu procesu automatizācijai. Modeļi atbilst *MDA* (angļ. val. *Model Driven Architecture*) formātam. Pieeja definē katra konfigurācijas pārvaldības modeļa mērķi, galvenos uzdevumus un darbības principus. Pieejas realizācijai izstrādāta jauna metodoloģija, kas realizē pieejas modeļus. Metodoloģijas ietvaros izstrādāti jauni modeļi un metodes, kas realizējot piedāvātas modeļvadāmas pieejas principus, ļauj automātiski iegūt izejas kodu konfigurācijas pārvaldības automatizācijai.

Promocijas darba kontekstā konfigurācijas pārvaldības automatizācijas kvalitātes rādītājs ir kļūdainu programmatūras būvējumu skaits. Programmatūras būvējums ir galvenais rezultāts konfigurācijas pārvaldības automatizācijas procesā, līdz ar to pieņem, ka mazāks kļūdainu būvējumu skaits atbilst kvalitatīvākam automatizācijas procesam.

Darba uzdevumi

Promocijas darba mērķa sasniegšanai ir izvirzīti šādi uzdevumi:

- izpētīt esošus risinājumus konfigurācijas pārvaldības procesu automatizācijas ieviešanai, apzināties galvenās problēmas un risinājumu attīstības tendences;
- identificēt galvenos ieguvumus un trūkumus jaunākajos konfigurācijas pārvaldības automatizācijas risinājumos, kas atbilst mūsdienīgām attīstības tendencēm;
- izstrādāt pieeju, metodoloģiju, modeļus un metodes konfigurācijas pārvaldības procesu automatizācijai. Pieejai jābūt orientētai uz automatizācijas ieviešanas laika samazināšanu un automatizācijas kvalitātes uzlabošanu, atkārtoti izmantojot uzņēmumā esošus automatizācijas risinājumus;
- izstrādāt programmatūras prototipu piedāvātu risinājumu eksperimentālājai pārbaudei;
- definēt kritērijus piedāvātas pieejas eksperimentālai novērtēšanai;
- veikt piedāvātu risinājumu ieviešanu reālos IT projektos un to eksperimentālu novērtēšanu pēc definētiem kritērijiem;
- balstoties uz eksperimentu rezultātiem, noformulēt rekomendācijas piedāvātas pieejas un modeļu ieviešanai programmatūras izstrādes projektos;
- definēt piedāvātas pieejas turpmākus attīstības un uzlabošanas virzienus.

Pētījuma objekts un priekšmets

Pētījuma objekts ir programmatūras izstrādes un uzturēšanas projekti.

Pētījuma priekšmets ir programmatūras konfigurācijas pārvaldības procesi šajos projektos.

Darba hipotēzes

Konfigurācijas pārvaldības pētījums ir balstīts uz faktu, ka, attīstoties spējo programmatūras izstrādes metodoloģijai, programmatūras izstrādes projekts iesākas ļoti strauji salīdzinājumā, piemēram, ar ūdenskrituma metodoloģiju. Šis fakts izraisa nepieciešamību pēc iespējas ātrāk ieviest automatizāciju konfigurācijas pārvaldības procesiem, lai pasūtītājs varētu pēc iespējas ātrāk saņemt programmatūras pirmās versijas.

Izstrādājot jaunu pieeju un modeļus konfigurācijas pārvaldības automatizācijai, tika izvirzītas šādas hipotēzes:

- lai samazinātu konfigurācijas pārvaldības automatizācijas ieviešanas laiku, var atkārtoti izmantot jau esošus automatizācijas risinājumus, kas jau funkcionē citos programmatūras izstrādes projektos;
- jo ilgāk konfigurācijas pārvaldības automatizācijas risinājumi tiek lietoti dažādos projektos, jo efektīvāk var tos lietot atkārtoti, ieviešot konfigurācijas pārvaldības procesu automatizāciju jaunajā programmatūras izstrādes projektā.

Pirmā hipotēze balstās uz faktu, ka pielāgot un konfigurēt jau esošus risinājumus ir ātrāk nekā izstrādāt no nulles pilnīgi jaunus. Jebkura izstrāde prasa laiku, kas tiek tērēta gan izstrādei, gan risinājuma testēšanai. Izmantojot risinājumu atkārtoti, šis laiks ir mazāks, jo izstrāde un risinājuma pilnīga pārtestēšana nav jāveic atkārtoti. Ja konfigurācijas pārvaldības automatizācijas ieviešanai atkārtoti izmanto jau esošus risinājumus, no nulles ir jāizstrādā tikai tās procesa daļas, kas ir specifiskas konkrētam projektam.

Otrā hipotēze balstās uz faktu, ka programmatūrā pilnīgi visas kļūdas nevar atklāt testēšanas posmā. Ir kļūdas, ko ir iespējams atklāt tikai programmatūras reālas ekspluatācijas laikā. Risinājumi, kas automatizē konfigurācijas pārvaldību, nav izņēmums. Tāpēc, jo ilgāk tas risinājums tiek lietots konfigurācijas pārvaldības procesos, jo vairāk kļūdu un nepilnību var atklāt un padarīt to risinājumu stabilāku. Tāpēc konfigurācijas pārvaldības automatizācijas risinājumu atkārtotas lietošanas efektivitāte būs atkarīga no tā, cik ilgi tiek lietots konkrētais risinājums.

Pētījuma metodes

Pētījumā tika izmantotas šādas metodes:

- literatūras analīze;
- modelēšana un metamodelēšana;
- modeļu transformācijas;
- eksperimentu plānošana un organizācija.

Zinātniskais jaunieguvums

Promocijas darbam ir šāds zinātniskais jaunieguvums:

- izstrādāta jauna pieeja *MTM* (Modelis — Transformācija — Modelis) konfigurācijas pārvaldības procesu automatizācijas ieviešanai ar modeļu palīdzību, atkārtoti izmantojot jau esošus automatizācijas risinājumus;
- izstrādāta jauna metodoloģija *EAF* (Vide — Darbība — Ietvars, angļu val.: *Environment — Action — Framework*), kas implementē jaunas *MTM* pieejas principus un definē soļus konfigurācijas pārvaldības automatizācijas ieviešanai;
- izstrādāti modeļi konfigurācijas pārvaldības procesu attēlošanai *EAF* metodoloģijas ietvaros;
- izstrādāta metode konfigurācijas pārvaldības esošu risinājumu glabāšanai.

Teorētiskā vērtība

Promocijas darba teorētiska vērtība ir šāda:

- izanalizētas konfigurācijas pārvaldības definīcijas un definēti konfigurācijas pārvaldības galvenie uzdevumi;
- balstoties uz literatūras analīzi par konfigurācijas pārvaldības uzdevumiem, tika definēta konfigurācijas pārvaldības procesu automatizācija;
- izanalizēti esoši risinājumi konfigurācijas pārvaldības automatizācijai un apkopotas risinājumu attīstības tendences;
- izstrādāta jauna pieeja, metodoloģija, modeļi un metode konfigurācijas pārvaldības automatizācijas ieviešanai, balstoties uz *MDA* formātu;
- izmantojot *MetaEdit+* rīku, tika izstrādāta modelēšanas valoda, kas ļauj implementēt jaunas *MTM* pieejas konceptus, definējot modeļus, transformācijas un papildu elementus pieejas implementācijai;
- izdevās noskaidrot, ka modeļvadāma pieeja konfigurācijas pārvaldības procesu ieviešanai palīdz samazināt cilvēciska faktora risku, pārejot no procesa automatizācijas prasībām uz implementāciju.

Praktiskā nozīmība

Promocijas darbam ir šāda praktiskā nozīmība:

- izstrādāts eksperimentālais programmatūras prototips, kas automatizē *EAF* metodoloģijas modeļu ģenerēšanu un transformāciju;
- iika izveidota kompetences grupa *EAF* metodoloģijas praktiskas testēšanas aktivitātēm. Kompetences grupā piedalījās vecākie un vadošie programmētāji, kas ikdienas darbā saskaras ar konfigurācijas pārvaldības procesiem strādājot uzņēmumā SIA «*Tieto Latvia*»;
- tika izstrādāti kritēriji *EAF* metodoloģijas novērtēšanai, sniegts skaidrojums, kā var aprēķināt kritēriju rādītājus;
- tika veikti eksperimenti, ieviešot konfigurācijas pārvaldības automatizāciju piecos programmatūras izstrādes un uzturēšanas projektos. Balstoties uz eksperimentu rezultātiem, tika definēti *EAF* metodoloģijas praktiskie ieguvumi, atšķirības no citiem konfigurācijas pārvaldības automatizācijas risinājumiem, metodoloģijas ieviešanas riski;
- tika izstrādāta praktisku rekomendāciju kopa, kā var ieviest konfigurācijas pārvaldības automatizāciju pēc jaunās *EAF* metodoloģijas.

Promocijas darba praktiskus rezultātus var izmantot programmatūras izstrādes uzņēmumi, kas vēlas uzlabot konfigurācijas pārvaldības automatizācijas risinājumu efektivitāti, samazināt automatizācijas ieviešanas laiku jaunajos projektos.

Darba aprobācija

Par promocijas darba rezultātiem tika ziņots 10 starptautiskās konferencēs Latvijā, Itālijā, Turcijā, Francijā un Austrijā:

- 2011. g. 13. oktobris. RTU 52. Starptautiskā zinātniskā konference, Rīga, Latvija.
- 2012. g. 12. oktobris. RTU 53. Starptautiskā zinātniskā konference, Rīga, Latvija.
- 2013. g. 17. oktobris. RTU 54. Starptautiskā zinātniskā konference, Rīga, Latvija.
- 2014. g. 14. oktobris. RTU 55. Starptautiskā zinātniskā konference, Rīga, Latvija.
- 2012. g. 27. aprīlis. LLU Applied Information and Communication Tehnology 2012, Jelgava, Latvija.

- 2013. g. 27. aprīlis. LLU Applied Information and Communication Tehnology 2013, Jelgava, Latvija.
- 2014. g. 22.–24. novembris. 3rd International Conference on Systems, Communications, Computers and Applications (CSCCA"14), Florence, Itālija.
- 2014. g. 15.–17. decembris. 13th International Conference on Telecommunications and Informatics TELE-INFO'14, Stambula, Turcija.
- 2015. g. 9.–11. februāris. 3rd International Conference on Model-Driven Engineering and Software Development MODELSWARD 2015, Anžē, Francija.
- 2015. g. 15.–17. martā. International Conference on Applied Physics, Simulation and Computers, Vīne, Austrija.

Saistībā ar promocijas darbu veikto pētījumu rezultāti ir atspoguļoti šādās publikācijās:

1. Bartusevics A., Kotovs V., Novickis L. A Method for Effective Reuse-Oriented Software Release Configuration and Its Application in Insurance Area. In: Scientific Journal of Riga Technical University. Information Tehnology and Management Science, 15th series, RTU Publishing House, 2012, Riga, Latvia, pp. 111–115. (Indeksēts: EBSCO, VINITI, Google Scholar)
2. Bartusevics A., Kotovs V. Towards the effective reuse-oriented release configuration process. In: Proceedings of the 5th International Scientific Conference «Applied Information and Communication Tehnologies», 2012, Jelgava, Latvia, pp. 99–103. (Indeksēts: EBSCO, VINITI)
3. Bartusevics A., A Methodology for Model-Driven Software Configuration Management Implementation and Support. In: Proceedings of the 6-th International Scientific Conference «Applied Information and Communication Tehnologies», 2013, Jelgava, Latvia, pp. 252–258. (Indeksēts: EBSCO, VINITI)
4. Bartusevičs, A., Novickis, L., Bluemel, E. Intellectual Model-Based Configuration Management Conception. In: Scientific Journal of Riga Technical University. Applied Computer Systems. 2014./15, pp. 22.–27. ISSN 2255-8683. e-ISSN 2255-8691. (Indeksēts: EBSCO, VINITI, Google Scholar)
5. Bartusevičs, A., Novickis, L., Model-Driven Software Configuration management and Environment Model. In: Recent Advances in Electrical and Electronic Engineering. In: Proceedings of the 3rd International Conference on Systems, Communications, Computers and Applications (CSCCA"14), Itālija, Florence, 22.–24. novembris, 2014. Italy: WSEAS Press, 2014, pp. 132.-140. ISBN 978-960-474-399-5. ISSN 1790-5117. (Tiks indeksēts: SCOPUS)
6. Bartusevičs, A., Novickis, L., Lesovskis, A. Model-Driven Software Configuration Management and Semantic Web in Applied Software Development. In: Proceedings of the 13th International Conference on Telecommunications and Informatics (TELE-INFO '14), IIstanbul, Turkey December 15–17, 2014, pp. 108.–116. (Tiks indeksēts: *SCOPUS*)
7. Bartusevičs, A., Novickis, L. Models for Implementation of Software Configuration Management. No: Procedia Computer Science. Valmiera, Latvia: 2014, 3.–10. lpp. (Tiks indeksēts: *SCOPUS*)
8. Bartusevičs, A., Novickis, L., Leye, S. Implementation of Software Configuration Management Process by Models: Practical Experiments and Learned Lessons. In: Scientific Journal of Riga Technical University. Applied Computer Systems. Nr. 16, 2014, RTU Press, pp. 26.–32. ISSN 2255-8683. e-ISSN 2255-8691. (Indeksēts: *EBSCO, VINITI, Google Scholar*)
9. Bartusevics, A., Novickis, L. Model-Based Approach for Implementation of Software Configuration Management Process. Starptautiskās konferences

MODELSWARD 2015 rakstu krājums, Francija, Anžē, 9–11 Februāris. (Tiks indeksēts: *SCOPUS*)

10. Bartusevičs, A., Novickis, L. Towards the Model-driven Software Configuration Management Process. In: Scientific Journal of Riga Technical University. Information Technology and Management Science. Vol. 17, 2014, pp. 32–38. ISSN 2255-9086. e-ISSN 2255-9094. (Indeksēts: *EBSCO, VINITI, Google Scholar*)
11. Bartusevičs, A., Lesovskis, A., Novickis, L. Semantic Web Technologies and Model-Driven Approach for the Development and Configuration Management of Intelligent Web-Based Systems. No: Proceedings of the 2015 International Conference on Circuits, Systems, Signal Processing, Communications and Computers, Austrija, Vienna, 15.–17. marts, 2015. Vienna: 2015, 32.–39. lpp. ISBN 978-1-61804-285-9. ISSN 1790-5117. (Tiks indeksēts: *SCOPUS*)

Darba struktūra

Promocijas darbu veido ievads, piecas nodaļas, secinājumi, bibliogrāfiskais saraksts un pielikumi. Promocijas darba pamatteksts ir 228 lappuses, tajā ir 55 attēli un 30 tabulas. Bibliogrāfiskajā sarakstā ir 115 nosaukumu informācijas avoti.

Promocijas darba *ievadā* tika pamatota veiktā pētījuma aktualitāte, formulēts promocijas darba mērķis un uzdevumi, izvirzītas hipotēzes, aprakstītas pētījuma metodes, aprakstīta zinātniska novitāte un iegūto rezultātu praktiska nozīmība, kā arī ir atspoguļota darba aprobācija.

Darba *1. nodaļā* tika definēts programmatūras konfigurācijas pārvaldības jēdziens un konfigurācijas pārvaldības galvenie uzdevumi. Balstoties uz literatūras analīzi, tika definēta konfigurācijas pārvaldības procesu automatizācija. Tika izanalizēti esoši risinājumi konfigurācijas pārvaldības uzdevumiem automatizācijai, definētas galvenās problēmas un mūsdienu risinājumu attīstības tendences.

Promocijas darba *2. nodaļā* tika analizēti esošās pieejas un esošie rīki konfigurācijas pārvaldības automatizācijai, kas izmanto *MDA* formātu un galvenos principus. Balstoties uz analīzes rezultātiem, tika noteikti trūkumi esošajās pieejās. Nodaļas secinājumos ir sniegtas konfigurācijas pārvaldības pieejas pazīmes, kurām jābūt, lai novērstu identificētus trūkumus esošajos risinājumos.

Darba *3. nodaļā* tika aprakstīta jaunizstrādāta *MTM* pieeja konfigurācijas pārvaldības automatizācijas ieviešanai ar modeļu palīdzību. Tika piedāvāta jauna *EAF* metodoloģija MTM pieejas realizācijai, kuras izstrāde ir balstīta uz *MDA* formātu. Jaunas metodoloģijas implementācijai tika definēti jauni modeļi un atbilstoši meta–modeļi konfigurācijas pārvaldības procesu attēlošanai, kā arī modeļu transformācijas likumi, kas ļauj mainīt modeļu abstrakcijas līmeni. Metodoloģija piedāvā ieviest konfigurācijas pārvaldības procesu automatizāciju, izmantojot esošas automatizācijas atsevišķiem konfigurācijas pārvaldības uzdevumiem. Tika izstrādāta metode esošu konfigurācijas pārvaldības automatizācijas risinājumu glabāšanai.

Promocijas darba *4. nodaļā* ir veikta jaunas *EAF* metodoloģijas testēšana. Tika aprakstīti metodoloģijas vērtēšanas kritēriji un teorētisku rezultātu aprobācijas rezultāti. Metodoloģijas testēšanas gaitā konfigurācijas pārvaldība tika ieviesta piecos programmatūras izstrādes projektos. Eksperimentu rezultātā tika noteikti metodoloģijas ieguvumi un trūkumi. Analizējot ieguvumus, trūkumus, recenzentu atsauksmes, kas tika iegūtas, publicējot metodoloģijas teorētiskus pamatus zinātnisku konferenču krājumos, tika konstatēts, ka esošus ieguvumus var palielināt, bet trūkumu skaitu samazināt, veicot uzlabojumus metodoloģijā.

Darba *5. nodaļā* tika aprakstīta *EAF* metodoloģijas uzlabotās versijas izstrāde. Izstrādes galvenais mērķis bija novērst eksperimentu rezultātā atklātus trūkumus un ņemt vērā piezīmes, ko izteica zinātnisku rakstu recenzenti, kas iepazinās ar *EAF* metodoloģiju. Veicot atkārtotus eksperimentus, izdevās parādīt, kā trūkumi tika novērsti un ieguvumi palielinājās. Nodaļas

beigās ir aprakstīti trūkumu novēršanas pasākumi. Balstoties uz eksperimentu pirmās un otrās kārtas rezultātu salīdzinājumiem, ir noteikti metodoloģijas galvenie ieguvumi, atšķirības no citām pieejām konfigurācijas pārvaldības automatizācijas ieviešanai, kā arī noteikti metodoloģijas ieviešanas riski un definēti turpmākie attīstības virzieni.

Promocijas darba *noslēguma daļā* ir izklāstīti promocijas darba galvenie rezultāti, pamatota mērķa un uzdevumu izpilde, hipotēžu pierādījums, kā arī ir uzskaitīti iespējamie turpmāko pētījumu virzieni.

1. PROGRAMMATŪRAS KONFIGURĀCIJAS PĀRVALDĪBAS IZPĒTE

Programmatūras konfigurācijas pārvaldības definīcija

Literatūras analīzes rezultātā [AIE 2010, BER 2003, DEP 2010, PAU 2007, MET 2002, KAN 2005, CON 2002, GLO 2012, BRU 2004, DAR 2001, WES 2005, MEL 2006, BEL 2005, VAC 2006, WIK 2013, ОРЛ 2011, ЛАП 2004, УДО 2011, ЗАМ 2008] tika atrastas vairāk nekā 20 dažādas definīcijas, kas izskaidro konfigurācijas pārvaldības jēdzienu. Atrastajām definīcijām tika apvienotas kopējās daļas un rezultātā iegūta konfigurācijas pārvaldības procesu definīcija.

Programmatūras konfigurācijas pārvaldība ir procesu kopums, kas identificē un kontrolē programmatūras vienumus un to evolūcijas procesu, sniedz vadlīnijas programmatūras būvējuma un instalācijas procesam, kā arī veic programmatūras vienumu statusu uzskaiti.
Programmatūras konfigurācijas pārvaldībai ir šādi galvenie uzdevumi:

- konfigurācijas vienumu identifikācija;
- konfigurācijas vienumu versiju kontrole;
- gatava produkta komplektācija (laidienu vai instalācijas paku sagatavošanas process; angļ. val. *building engineering*);
- gatava produkta instalācija (angļ. val. *deployment*);
- paralēlas izstrādes nodrošinājums (ar konfigurācijas vienumiem vienlaikus strādā vairāki izstrādātāji; angļ. val. *branching*),
- metriku savākšana par konfigurācijas vienumu izmaiņām, versijām un dažādām produkta konfigurācijām;
- konfigurācijas vienumu uzskaite un audits.

Analizējot literatūru, izdevās noskaidrot galvenās konfigurācijas pārvaldības definīcijas un galvenos uzdevumus. Sakarā ar to, ka promocijas darba mērķis ir izstrādāt pieeju un metodoloģiju konfigurācijas pārvaldības automatizācijas ieviešanai, balstoties uz šīs nodaļas ietvaros iegūto informāciju, tiks definēta konfigurācijas pārvaldības automatizācija.

Konfigurācijas pārvaldības automatizācijas risinājumi — programmatūra, kas realizē šajā nodaļā definētos konfigurācijas pārvaldības uzdevumus, minimizējot cilvēka iejaukšanos. Galvenokārt automatizācija ir vērsta uz versiju kontroli, izejas koda pārvaldību, programmatūras būvējumu veidošanu, programmatūras instalāciju.

Līdz ar to formulējums «izstrādāt automatizāciju konfigurācijas pārvaldībai» promocijas darba kontekstā nozīmē izstrādāt programmatūras kopu (skripti, bibliotēkas, ietvari), kas ar minimālu cilvēka iejaukšanos spēj veikt šajā nodaļā definētos konfigurācijas pārvaldības uzdevumus, galvenokārt versiju kontroli, izejas koda pārvaldību, būvējumu un instalācijas veidošanu.

Nodaļā definēta programmatūras konfigurācijas pārvaldības automatizācija. Automatizācijai nepieciešams atrisināt šādus uzdevumus: versiju kontrole, izejas koda pārvaldība, produkta būvējumi un instalācijas, metriku savākšana.

Pētījuma gaitā izdevās identificēt piecas svarīgas iezīmes, kas raksturo mūsdienīgu konfigurācijas pārvaldības procesu: process risina kompleksi visus uzdevumus, process ir modeļvadāms, ir iespēja izmantot esošos rīkus un skriptus jaunajā modeļvadāmajā automatizācijas risinājumā, versiju kontrole strādā ne tikai ar kodu, bet arī ar modeļiem, lai varētu atbalstīt projektus ar *MDD* (angļ. val. *Model Driven Development*) pieeju un process nav pretrunā ar kvalitātes standartiem.

Nākamajā promocijas darba nodaļā tiks analizēti konfigurācijas pārvaldības automatizācijas risinājumi, kas atbilst modeļvadāmas arhitektūras formātam. Katrai pieejai tiks vērtēti šādi kritēriji:

- pieejas atbilstība modeļvadāmas arhitektūras principiem;
- pieejas apgabals, risināmie konfigurācijas pārvaldības uzdevumi;
- iespējamība izmantot esošos rīkus vai skriptus, kā arī radīt jaunus risinājumus, kurus ir iespējams lietot atkārtoti.

2. MODEĻVADĀMA KONFIGURĀCIJAS PĀRVALDĪBA

Modeļvadāmas arhitektūras vispārīgi principi

Modeļvadāma arhitektūra (*MDA*, angļ. val. *Model-Driven Architecture*) sākotnēji tika radīta programmatūras izstrādei. Modeļvadāma programmatūras izstrāde ir sistemātiskā modeļu lietošana programmatūras izstrādes dzīves ciklā. *MDA* ir saistīta ar tādām izstrādes metodēm, kuru pamatā ir programmatūras modeļu izmantošana primāro izstrādes artefaktu un izteiksmes formu veidā, kas paredz zināšanu atspoguļošanu strukturētā veidā ar modelēšanas valodas palīdzību, ievērojot tās noteikumus.

Modelis *MDA* kontekstā ir sistēmas vai tās daļas apraksts valodā ar skaidri definētu formu (sintaksi) un nozīmi (semantiku), kas varētu būt automātiski interpretēta ar datora palīdzību. Neskatoties uz to, ka sistēmai var būt definēti vairāki atšķirīgi modeļi, starp tiem eksistē noteiktas saistības (piemēram, vesels-daļa, kad viens modelis definē kopējo priekšstatu, bet cits — tikai atsevišķu sistēmas daļu detalizētā veidā). Pašlaik modeļvadāmā arhitektūra ir parasti cieši saistīta ar *UML*, kas ļauj novērst nepareizu modeļu iztulkojumu, tomēr specifisko domēnu valodu (*DSL*) izmantošanā ir pieļaujama alternatīva [DON 2011, OSI 2011].

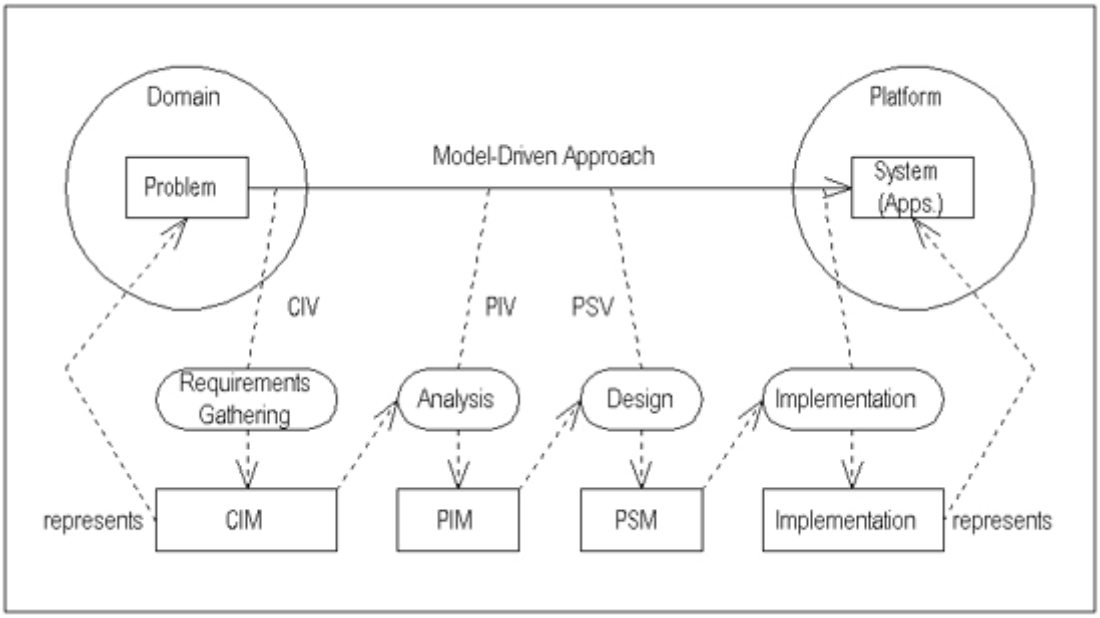
Modeļvadāma pieeja, kas ir bāzēta uz *UML* valodas un citiem programminženierijas industrijas standartiem modeļu un projektējuma vizualizēšanai, glabāšanai un apmaiņai. *MDA* ļauj izveidot augstas abstrakcijas modeļus, kas nav atkarīgi no izpildīšanas platformas un kas glabājas specializētos standartizētos repozitorijos. *MDA* ietver šādas tehnoloģijas: vienota modelēšanas valoda (angļ. val. *Unified Modeling Language — UML*), metaobjektu iespējas (angļ. val. *Meta-Object Facilities — MOF*), *XML* metadatu apmaiņa (angļ. val. *XML Metadata Interchange — XMI*) un kopējais krātuves meta-modelis (angļ. val. *Common Warehouse Metamodel — CWM*) [DON 2011, OSI 2011].

Modeļu transformācijas ir vienots sistēmas process viena modeļa konvertēšanai citā modelī, saglabājot noteikto ekvivalences saistību starp šiem modeļiem. *MDA* arhitektūras pamatā ir modelēšanas process un šo modeļu savstarpējas transformācijas. Modeli var izteikt kā *UML* diagrammu, *OCL* specifikāciju un teksta kopums. Modeļvadāma pieeja nosaka dažādus modeļu tipus, kas var būt abstrakti (specificē sistēmas funkcionalitāti) un konkrēti, kas saistīti ar specifisku platformu, tehnoloģiju un implementāciju. Ir šādi modeļu tipi:

- *CIM* modelis (angļ. val. *Computation Independent Model*) jeb no skaitļošanas neatkarīgais modelis;
- *PIM* modelis (angļ. val. *Platform Independent Model*) jeb no platformas neatkarīgais modelis;

- *PSM* modelis (angļ. val. *Platform Specific Model*) jeb no platformas atkarīgais modelis;
- koda modelis (angļ. val. *Code Model*), retāk tiek lietots apzīmējums *ISM* modelis (angļ. val. *Implementation Specific Model*).

Ir paredzēts, ka iespējams veikt transformācijas no *CIM* uz *PIM*, no *PIM* uz *PSM* un no *PSM* uz koda modeli, kā arī veikt transformācijas viena abstrakcijas līmeņa ietvaros. Vienam iepriekšēja abstrakcijas līmeņa modelim var atbilst vairāki nākamā līmeņa modeļi. Attēlā 2.1. shematiski parādīts programmatūras izstrādes process modeļvadāmas pieejas kontekstā.



2.1. att. Modeļvadāma pieeja.

Konfigurācijas pārvaldības modeļvadāmie risinājumi

Modeļvadāma konfigurācijas pārvaldība [PIN 2009] satur modeļus, dažas meta-modeļu definīcijas un rekomendācijas, kā uzlabot konfigurācijas pārvaldības un programmatūras izstrādes sadarbību. Rezultātā tika izveidots rīks — *Model Driven Configuration Editor*. Tas ir grafiskais redaktors *Eclipse* vidē, kas izmanto *Eclipse Modeling Framework* un *Grafical Modeling Framework* kā pamatus modeļu vizuālai attēlošanai. Transformācijām tiek izmantota *openArchitectureWare* (*oAW*) arhitektūra. Darbs [PIN 2009] sniedz arī detalizētas instrukcijas, kā izveidot patvaļīgu konfigurācijas pārvaldības modelēšanas rīku.

Pieejai [PIN 2009] ir šādi galvenie sasniegumi:

- konfigurācijas pārvaldības un modeļvadāmas izstrādes apvienošanas koncepcija;
- produkta konfigurācijas abstrakts modelis;
- rīks, kas paredzēts konfigurācijas pārvaldības modeļu izveidei;
- instrukcijas, kā radīt un paplašināt rīku, kas atbalsta modeļvadāmo konfigurācijas pārvaldību.

Sakarā ar iegūto informāciju no [PIN 2009] avota, promocijas darba autors secināja:

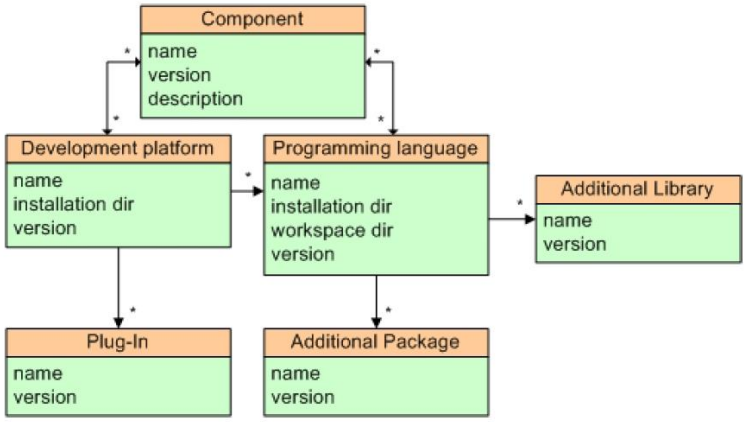
- piedāvātais risinājums ir orientēts uz projektiem, kuru izstrāde notiek pēc modeļvadāmas pieejas. Īsti nav skaidrs, vai metodoloģiju var izmantot, ja projekta komanda strādās pēc citas metodoloģijas, piemēram, pēc ūdenskrituma metodes, kur izstrādes artefakti nebūs modeļi, bet programmatūras kods;
- pieeja ir orientēta galvenokārt uz vienu no konfigurācijas pārvaldības lielajiem uzdevumiem — konfigurācijas vienumu identifikāciju;

- pieeja pilnībā atbilst modeļvadāmai pieejai — ir meta-modelis, ir priekšlikumi, kā meta-modeli veidot katram konfigurācijas vienuma veidam. Ir minēta arī *PSM* un *PIM* modeļu lietošana un transformācijas. Taču, kā jau tika minēts, metodoloģija ir orientēta vien uz konfigurācijas vienumu identifikāciju ar nosacījumu, ka projektā izmanto modeļvadāmo izstrādes metodoloģiju.

Publikācijā [PIN 2009] tiek piedāvāta konceptuāla realizācija modeļvadāmai konfigurācijas pārvaldībai. Metode orientēta uz konfigurācijas vienumu identifikāciju un attiecību noteikšanu augstākajā līmenī. Konfigurācijas vienumi varētu būt šādi:

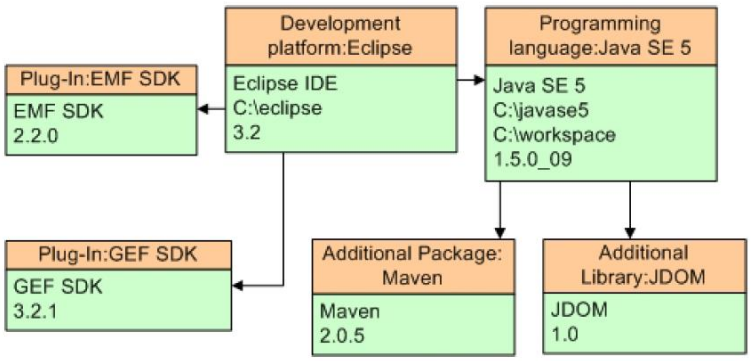
- aparātu komponentes (serveri);
- programmatūras komponentes (datubāzes, aplikācijas, operētājsistēmas);
- dokumentācija un izejas koda faili;
- organizācijas komponentes (reglamentē lietotāju pieejas tiesības utt.).

Katrai konfigurācijas vienumu grupai tiek veidots meta-modelis. Pēc tam tiek veidots *PIM* (no platformas neatkarīgais) modelis no iegūtā meta-modeļa. Vēlāk ar transformāciju palīdzību iespējams modeli transformēt *PSM* (no platformas atkarīgs) modelī un pat *XML* failos, kas apraksta konfigurācijas vienumus, struktūru un saiknes. Attēlā 2.2. var redzēt *PIM* modeļa piemēru izstrādes videi.



2.2. att. *PIM* modeļa piemērs izstrādes videi.

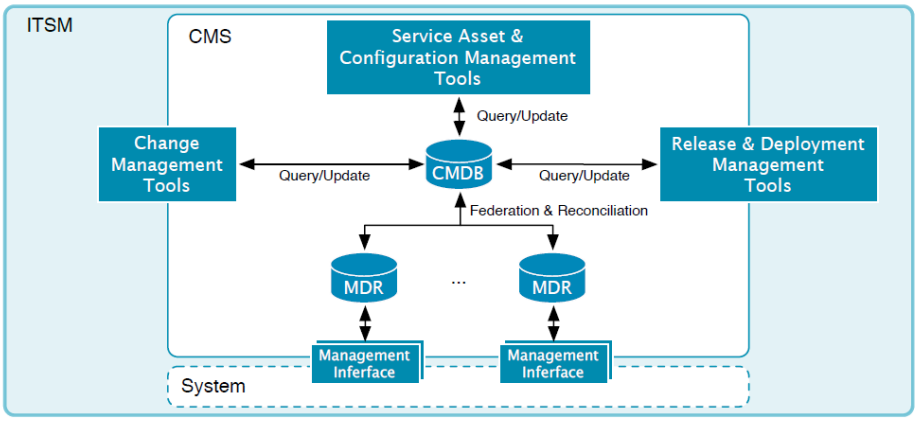
Transformējot minēto *PIM* modeli *PSM* modelī, attēlā 2.3. var redzēt piemēru platformas atkarīgajam modelim, kas šajā gadījumā tiek veidots *Eclipse* integrētai izstrādes videi.



2.3. att. *PSM* modeļa piemērs izstrādes videi.

Vēlāk modeli, kas ir redzams attēlā 2.3., nepieciešamības gadījumā iespējams transformēt uz *XML* failu, kas būs saprotams konfigurācijas pārvaldības rīkam.

Atšķirībā no tikko apskatīta darba, kur uzsvars bija likts uz konfigurācijas vienumu identifikāciju un atkarību noteikšanu, avotā [GIE 2009] tiek piedāvāta metodoloģija, kas apskata konfigurācijas pārvaldības procesu kopumā. Konfigurācijas pārvaldības principi tiek ņemti no *ITIL* (*Information Technology Infrastructure Library*) un vēlāk tiek izveidoti modeļi, no kuriem savukārt varētu izveidot abstraktu konfigurācijas pārvaldības procesu un vēlāk transformēt to no platformas atkarīgajā modelī. Pieeja izmanto modeļvadāmas izstrādes galvenos principus. Uz meta-modeļiem balstīti modeļi piedāvā nepieciešamu abstrakciju, kas uzlabo konfigurācijas procesa pārvaldību, pārskatāmību un ļauj lietotājam nepieciešamības gadījumā implementēt modeli kādai noteiktai tehnoloģijai, piemēram, veicot modeļa transformāciju. Ir izveidots sistēmas prototips, kas implementē modeļvadāmo konfigurācijas pārvaldību. Konfigurācijas pārvaldības abstraktais modelis tiek izveidots *ITSM* (*IT Service Management*) kontekstā un ir attēlots 2.4. attēlā.



2.4. att. Konfigurācijas pārvaldības abstrakts modelis.

Pieeja paredz, ka eksistē šādi modeļi (sk. attēlu 2.4.):

- pārvaldības rīku (management tools) modeļi;
- konfigurācijas pārvaldības datubāzes (*CMDB*) modelis;
- pārvaldāma datu repozitorija (*MDR*) modelis.

Saites, kas redzamas attēlā 2.4. (*Query/Update* utt.), piedāvā realizēt kā operācijas ar modeļiem — transformācijas, apvienošana, papildināšana ar atribūtiem utt. Taču konkrētas realizācijas netiek piedāvātas. Darbā [GIE 2009] ir iekļauta implementācija piedāvātai modeļvadāmai konfigurācijas pārvaldībai. Taču risinājums ir orientēts tikai uz vienu tehnoloģiju (*JAVA*).

Viena no modeļvadāmajiem konfigurācijas pārvaldības pieejām [CAL 2012] ir orientēta uz rīku savstarpēju integrāciju, cenšoties paaugstināt procesa automatizācijas līmeni. Lai uzturētu konfigurācijas pārvaldības pilnu procesu, ir vajadzīgi vairāki rīki: versiju kontroles sistēmas, problēmu pārvaldības sistēmas, būvējumu serveri, nepārtrauktās integrācijas serveri un daudzi citi. Praksē bieži ir tā, ka visi minēti rīki strādā atsevišķi viens no otra. Lai atvieglotu konfigurācijas pārvaldības procesu, tiek piedāvāta pieeja integrēt kopā visus šos rīkus. Taču, lai integrētu konfigurācijas pārvaldības dažādus rīkus kopā, ir nepieciešams definēt katra integrējama rīka vispārīgu konceptu [CAL 2012]. Publikācija piedāvā uzdevumu ontoloģiju konfigurācijas pārvaldības procesiem. Šī ontoloģija tiek izmantota kā konfigurācijas pārvaldības modelis, kas parāda, kādā veidā tiks integrēti dažādi konfigurācijas pārvaldības rīki. Ontoloģija galvenokārt ir balstīta uz izmaiņu kontroli, kas ir viena no galvenajām koncepcijām konfigurācijas pārvaldībā. Ontoloģija piedāvā informāciju par konfigurācijas pārvaldības apakšprocesu savstarpējām saitēm [CAL 2012].

Pēc publikācijas [CAL 2012] izpētes tika secināts:

- tiek piedāvāta uzdevuma ontoloģija, kas vispārīgi apraksta konfigurācijas pārvaldības procesu konkrētu apakšuzdevumu kontekstā. Risinājums nav atkarīgs no kādām noteiktām tehnoloģijām vai platformām;
- ontoloģija modeļvadāmas pieejas kontekstā var būt kā avots no platformas neatkarīgajam modelim, taču mulsina fakts, ka ontoloģijas izveidošanai tika izmantoti vien konkrēti standarti (*ISO*) un rīki (*Subversion*). Tomēr trūkst apraksta, kā pašam veidot ontoloģijas elementus;
- ontoloģija pārsvarā tiek domāta konfigurācijas pārvaldības dažādu rīku integrācijai, taču integrācijai jābūt saistītai ar procesu. Kamēr nav abstrakta procesa modeļa, par rīku instalāciju un integrāciju domāt ir pāragri [AIE 2010];
- nav rekomendāciju, kā ontoloģiju varētu izmantot, lai iegūtu, piemēram, *PSM* modeli konfigurācijas pārvaldībai, kur, ņemot vērā publikācijas kontekstu, varētu būt informācija arī par rīku savstarpēju integrāciju.

Mēģinājumi lietojot formālas mākslīga intelekta metodes rīku konfigurēšanai ir ievēroti ne tikai konfigurācijas pārvaldībā. Arī programmatūras projektu vadībā ir nepieciešams efektīvi konfigurēt projekta pārvaldības sistēmas. Kā jau tika minēts promocijas darba ievadā, mūsdienās programmatūras izstrādes projekti sākās ļoti ātri. Arī no projekta vadības viedokļa ir ārkārtīgi svarīgi ātri un efektīvi nokonfigurēt pārvaldības sistēmu. Plaši šo problēmu apskata un risina RTU zinātniece Solvita Bērziņa [BER 2012, BER 2011].

2.1. tabula

Konfigurācijas pārvaldības modeļvadāmu risinājumu salīdzinājums

Risinājuma identifikators	Meta-modelis	Modeļi ar atšķirīgu abstrakcijas līmeni	Transformāciju risinājumi	Rīku atbalsts	Komentāri
[PIN 2009]	+	+	+/–	+/–	Saturiskajā ziņā labākais no visiem minētajiem risinājumiem, jo ir daļējs risinājums meta-modelim, rīks, kas veic modeļu transformācijas.
[GIE 2009]	+/–	+/–	–	+/–	Tīri teorētisks risinājums, nav minētas konkrētas detaļas, kā šādu risinājumu var realizēt. Pieeja ir orientēta tikai uz vienu tehnoloģiju.
[BUC 2009]	–	–	–	+/–	Risinājums ir orientēts vien uz versiju kontroli nevis uz konfigurācijas pārvaldības procesu kopumā.
[CAL 2012]	+/–	+/–	+/–	+/–	Lai gan risinājums neatbilst vispārīgiem modeļvadāmas pieejas principiem, ir uzsvērta svarīga problēma, kas obligāti jāņem vērā modeļvadāmas konfigurācijas pārvaldības risinājuma izstrādē — rīku savstarpēja integrācija. Teorētiskā līmenī ir piedāvāts risinājums, kā izveidot abstraktu integrācijas modeli rīkiem, kas atbalsta konfigurācijas pārvaldības procesu.

Apkopojot informāciju par konfigurācijas pārvaldības modeļvadāmajiem risinājumiem, tika izpētītas pieejas un to attīstības tendences [PIN 2009, GIE 2009, BUC 2009, CAL 2012, KR 2014, FIT 2014, FUG 2014, CRA 2008], kā arī jaunākie rīki modeļvadāmas pieejas praktiskai realizācijai [OPE 2014, SER 2014, AZO 2014]. Risinājumu atbilstība *MDA* formātam un komentāri atrodami tabulā 2.1.

Analizējot jaunākos rīkus, kas ievieš modeļvadāmu konfigurācijas pārvaldības procesu [OPE 2014, SER 2014, AZO 2014], tika apkopoti galvenie labumi un trūkumi, ko konstatēja promocijas darba autors. Būtiskākie sasniegumi rīkos [OPE 2014, SER 2014, AZO 2014] ir šādi:

- lielāka daļa no analizētiem rīkiem atbilst galvenajiem modeļvadāmas pieejas principiem. Rīki ļauj salīdzinoši ātri modelēt konfigurācijas pārvaldības procesu programmatūras izstrādes projektā un pēc tam implementēt to konkrētajām tehnoloģijām un platformām;
- konfigurācijas pārvaldības process ir pārskatāms un viegli konfigurējams, pateicoties intuitīvi saprotamai lietotāju grafiskai saskarnei. Konfigurācijas pārvaldnieks veido produkta būvējumu scenārijus ar peles klikšķiem, nevis rakstot milzīgus skriptus;
- rīki pilnībā atbilst mūsdienīgām programmatūras izstrādes nozares tendencēm. Tika realizētas iespējas veidot paralēlus būvējumus, konfigurēt sistēmu, kas spēj veikt vairākus desmitus būvējumus dienā. Pārsvarā visos rīkos ir iebūvētas funkcijas, kas atbalsta būvējumu veidošanas procesus arī mākoņos. Līdz ar to vairs nav jāraksta statiski skripti katram projektam atsevišķi.

Apkopojot informāciju no avotiem [OPE 2014, SER 2014, AZO 2014], tika secināts, ka rīkiem ir arī savi trūkumi:

- pārsvarā visi rīki ir orientēti uz šādiem konfigurācijas pārvaldības uzdevumiem: būvējumu un instalāciju pārvaldība, produkta laidienu sagatavošana pasūtītājam un metriku savākšana. Taču reti kurš rīks pievērš pietiekamu uzmanību izejas koda pārvaldības automatizācijai. Savukārt bez pārdomātas izejas koda pārvaldības būvējumu un instalācijas process nevar būt kvalitatīvs [AIE 2010];
- realizējot konfigurācijas pārvaldības procesu ar rīkiem, kas minēti šajos avotos [OPE 2014, SER 2014, AZO 2014], zemāka abstrakcijas līmeņa modeļiem (skripti, projektu struktūra, kompilācijas algoritmi utt.) ir definēti konkrēti priekšnosacījumi. Ja tos neievēro, risinājums nestrādās korekti. Taču nereti uzņēmumam ir sava specifika un pieeja dažādu skriptu un projektu konfigurācijai. Uzņēmums diez vai būs gatavs attiekties no risinājumiem un pieejām, kas ir pārbaudīti gadiem. Piemēram, ja, ieviešot kādu no jaunajiem būvējumu un instalācijas rīkiem, visiem *JAVA* projektiem vajadzēs pārtaisīt klašu un pakotņu struktūru, diez vai uzņēmums būs tam gatavs, savukārt pasūtītājs šādu aktivitāti visticamāk negribēs apmaksāt.

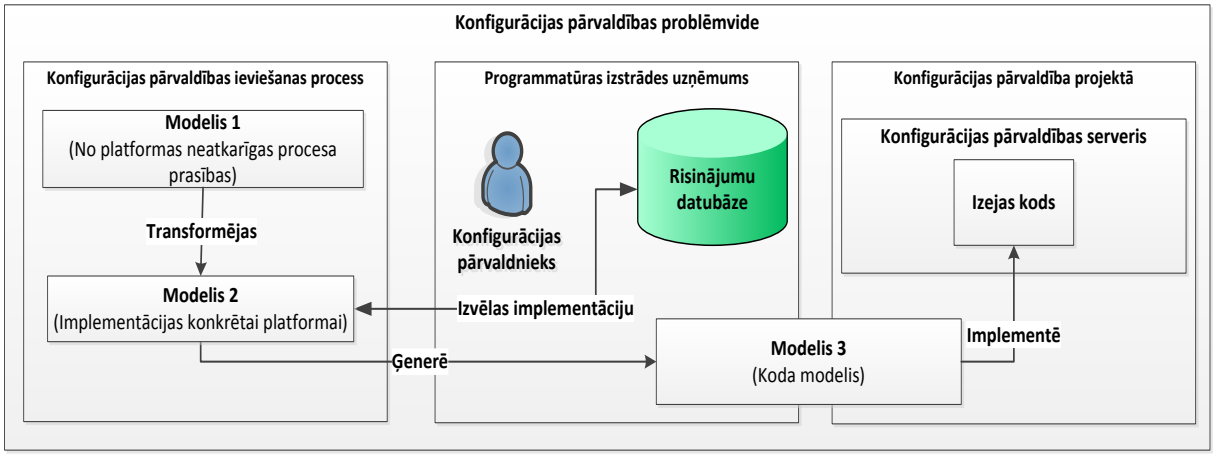
3. *MTM* PIEEJAS UN *EAF* METODOLOĢIJAS IZSTRĀDE

MTM pieejas definīcija un vispārīgs apraksts

MTM (no angļu val. *Model — Transformation — Model*) — jaunizstrādāta pieeja izejas koda iegūšanai programmatūras konfigurācijas pārvaldības procesu automatizācijai. *MTM* pieeja paredz, ka visus konfigurācijas pārvaldības procesus pārvalda atkārtoti izpildāms izejas kods no speciāla konfigurācijas pārvaldības servera. Šis izejas kods tiek iegūts automātiski, secīgi modelējot konfigurācijas pārvaldības automatizācijas procesus. Modeļi atbilst *MDA* formātam. Ir paredzēts, ka programmatūras izstrādes uzņēmumā, kas lieto *MTM*, ir realizēta risinājumu datubāze, kas glabā sevī atkārtoti lietojamas izejas koda vienības atsevišķiem konfigurācijas pārvaldības uzdevumiem atsevišķām platformām. Risinājumu datubāze glabā minētas izejas

koda vienības pēc noteiktās metodes, kas ir izstrādāta un vēlāk modernizēta šajā promocijas darbā.

MTM pieejas paredz, ka sākumā konfigurācijas pārvaldnieks modelē konfigurācijas pārvaldības procesu neatkarīgi no kādas konkrētas platformas. Vēlāk modelis tiek papildināts ar implementācijas detaļām, kuras konfigurācijas pārvaldnieks iegūst no risinājumu datubāzes. Rodas platformas specifiskais modelis konkrētam konfigurācijas pārvaldības procesam. Visbeidzot no šī modeļa automātiski tiek uzģenerēts izejas kods konfigurācijas pārvaldības procesu automatizācijai. Attēlā 3.1. var redzēt *MTM* pieejas shēmu.



3.1. att. *MTM* pieejas elementi un saites.

***EAF* metodoloģija *MTM* pieejas realizācijai**

Metodoloģijas mērķis ir definēt konfigurācijas pārvaldības automatizācijas ieviešanas soļus un sniegt iespēju jaunajos procesos izmantot jau esošus risinājumus. *EAF* ir saīsinājums no metodoloģijas angliskā nosaukuma «*Environment — Action — Framework*», latviskais nosaukums ir «Vide — Darbība — Ietvars». *EAF* metodoloģija realizē *MTM* pieejas principus, realizējot 3.1. attēlā redzamu modeļus Modelis 1, Modelis 2 un Modelis 3, ka arī rumu datubāzi un modeļu transformācijas likumus. Pakāpeniska pāreja no viena modeļa uz otru, izmantojot modeļu transformācijas likumus, definē konfigurācijas pārvaldības automatizācijas izejas koda veidošanas soļus. Risinājumu atkārtota izmantošana ļauj samazināt automatizācijas ieviešanas laiku un mazina risku neparedzētām kļūdām, kas rodas gadījumos, kad visi risinājumi ir jāizstrādā no nulles. *EAF* metodoloģijas izstrāde notika pakāpeniski, un starprezultāti tika publicēti zinātniskajos rakstos [BAR 2012a, BAR 2012b, BAR 2013, BAR 2014f].

EAF metodoloģijas izstrādes gaitā ieviestie jēdzieni

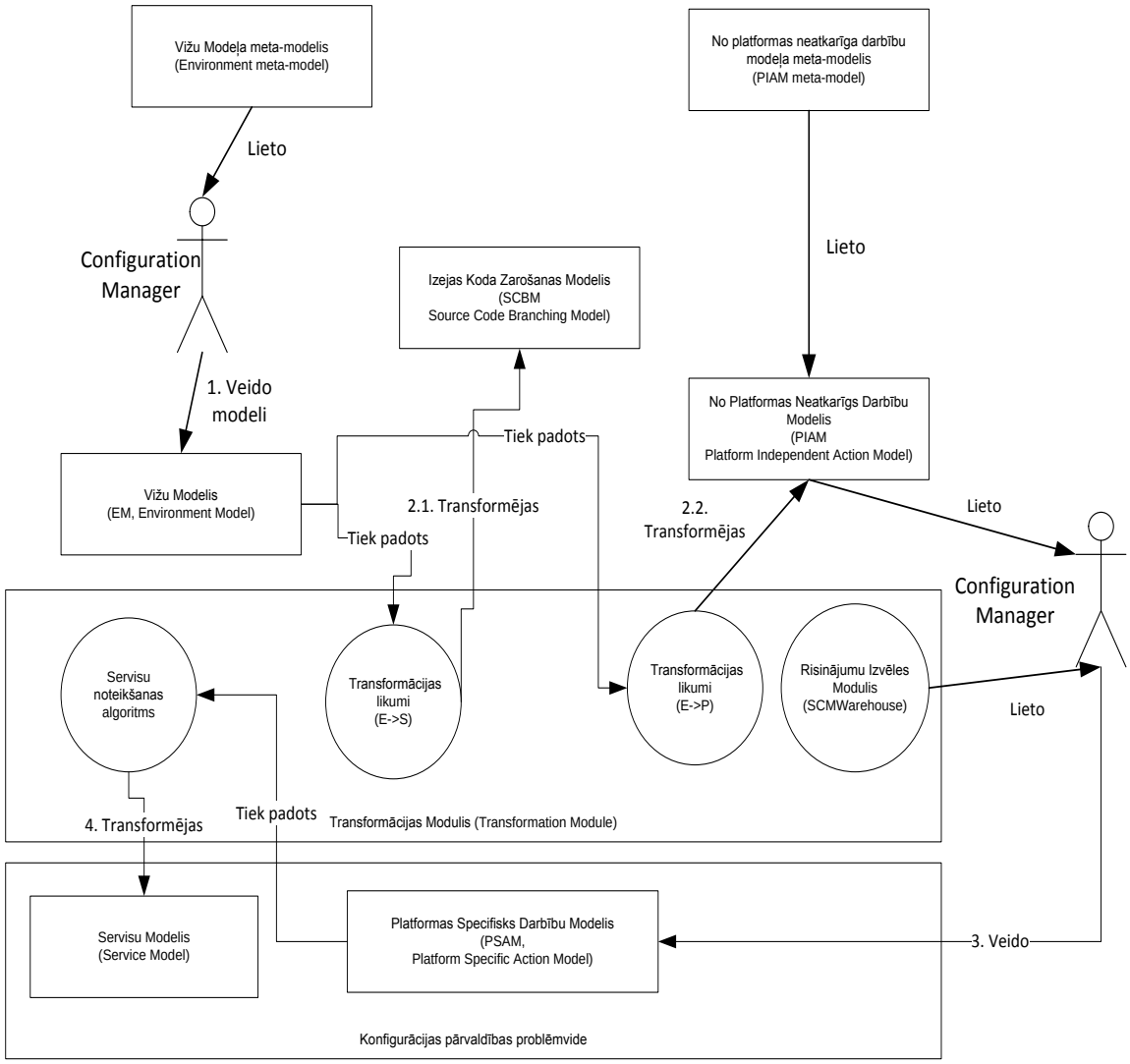
1. **Projekts** — programmatūras izstrādes projekts, kurā tiek aprakstīta konfigurācijas pārvaldība.
2. **Kompānija** — konkrēts uzņēmums, kas realizē programmatūras izstrādes projektus.
3. **Konfigurācijas pārvaldnieks (*Configuration Manager*)** — lietotājs, kas, izmantojot *EAF* metodoloģiju, modelē un implementē konfigurācijas pārvaldības automatizācijas procesu projektā.
4. **Konfigurācijas pārvaldības risinājumu glabātuve (*SCMWarehouse*)** — struktūra, kur glabājas visi kompānijas konfigurācijas pārvaldības automatizācijas risinājumi.
5. **Konfigurācijas pārvaldības risinājumu glabātuves pārvaldības sistēma** — aplikācija, kas pārvalda informāciju glabātuvē *SCMWarehouse*.
6. **Platforma (*Platform*)** — konkrēta operētājsistēma, kurā tiek implementēts konfigurācijas pārvaldības procesa izejas kods.

7. **Konfigurācijas pārvaldības serveris (*SCMServer*)** — centralizēta vietne, no kuras tiek pārvaldīta konfigurācijas pārvaldības izejas koda izpilde. Serveris ir orientēts uz kādu konkrētu platformu.
8. **Vide (*Environment*)** — infrastruktūras kopa, kurā atrodas izstrādājama programmatūra (aplikāciju serveri, datubāzes, ārēju sistēmu interfeisi utt.). Katra vide ir paredzēta konkrētai aktivitātei programmatūras izstrādes dzīves ciklā, piemēram, izstrādei, testēšanai, kvalitātes akcepttestēšanai, ekspluatācijai utt.
9. **Darbība (*Action*)** — aktivitāte konfigurācijas pārvaldības automatizācijas procesā. Parasti aktivitāte atrisina kādu no galvenajiem konfigurācijas pārvaldības uzdevumiem, piemēram, programmatūras būvējuma veidošana, izejas koda pārvaldība, programmatūras instalācija kādā no vidēm utt.

EAF metodoloģija satur šādus elementus:

- **vižu modeļa meta-modelis** — modelēšanas valoda vižu modeļa veidošanai;
- **vižu modelis (*EM*)** — konfigurācijas pārvaldības procesa modelis, kas attēlo visas konkrēta projekta vides, starp kurām notiek programmatūras izmaiņu pārnešana;
- **izejas koda zarošanas modelis (*SCBM*)** — modelis, kas ilustrē programmatūras izejas koda pārvaldības likumus atkarībā no vižu modeļa, parāda, kādi izejas koda zari atbilst kādām vidēm un kādā veidā notiek izejas koda izmaiņu pārnešana (angļu val. *merge*) starp dažādiem zariem;
- **no platformas neatkarīgs darbību modelis (*PIAM*)** — modelis, kas parāda, kādas darbības ir nepieciešams veikt, lai pārnestu programmatūras izmaiņas starp instancēm vižu modelī. Šajā modelī darbības nesatur nekādas implementācijas detaļas un nav atkarīgas no jebkādām platformām;
- **no platformas neatkarīga darbību modeļa meta-modelis** — modelēšanas valoda *PIAM* modeļa veidošanai;
- **platformas specifiskais darbību modelis (*PSAM*)** — paplašinātais variants *PIAM* modelim. Atšķirībā no, šīs modelis satur visu informāciju par darbību implementāciju: platformu, konkrētus rīkus, skriptus, instrukcijas;
- **servisu modelis (*Service Model*)** — modelis, kas attēlo rīku savstarpēju integrāciju. Modelis satur rīku pārus. Katram rīkam, kas atrodas konkrētajā pārī, ir funkciju vai metožu kopa, ko var izsaukt otrais pāra rīks. Servisu Modelis ir vajadzīgs dažādu rīku integrācijas aprakstam. Ja *PSAM* modelī var redzēt, kādi rīki ir nepieciešami konfigurācijas pārvaldības darbību analīzei, tad servisu modelis parāda, kā rīki savā starpā sadarbojas (integrējas), lai varētu uzturēt pilnvērtīgu konfigurācijas pārvaldības darbību plūsmu;
- **servisu noteikšanas algoritms** — algoritms, kas atkarībā no rīkiem *PSAM* modelī nosaka rīku pārus jeb servisus. *PSAM* modeļa implementācijas laikā konfigurācijas pārvaldniekam sakumā jārealizē servisi, ko noteiks servisu noteikšanas algoritms;
- **transformācijas likumi «E–>S» —** likumu kopa, kas operē ar vižu modeli un sagatavo atbilstošu izejas koda zarošanas modeli;
- **transformācijas likumi «E–>P» —** likumu kopa, kas operē ar vižu modeli un sagatavo atbilstošu *PIAM* modeli;
- **risinājumu izvēles modulis (*SCMWarehouse*) —** glabātuve, kur atrodas visi kompānijā esoši konfigurācijas pārvaldības risinājumi.

Attēlā 3.2. var redzēt *EAF* metodoloģijas vispārīgu shēmu. Ar bultiņām ir apzīmētas darbības un metodoloģijas galvenie soļi. «*Configuration Manager*» ir lietotājs, kas ievieš kompānijā konfigurācijas pārvaldības procesus un veido dažādus modeļus saistībā ar šo metodoloģiju.



3.2. att. *EAF* metodoloģijas vispārīga shēma.

Ir paredzēts, ka kompānijā ir izstrādāts risinājumu izvēles modulis, kas strukturētā veidā glabā visus kompānijas risinājumus konfigurācijas pārvaldības darbībām. *EAF* metodoloģijā ir četri galvenie soļi:

- konfigurācijas pārvaldnieks veido vižu modeli konkrētam programmatūras izstrādes projektam;
- transformācijas likumi «E–>S» un «E–>P» pārveido vižu modeli *SCBM* un *PIAM* modeļos. Šajā brīdī lietotājs zina, kādi izejas koda repozitorija zari ir jāveido, lai uzturētu izejas koda bāzi katrai vižu modeļa videi. Ir zināmas arī konfigurācijas pārvaldības darbības, kas nepieciešamas, pārnesot izmaiņas starp vidēm;
- izmantojot darbības *PIAM* modelī, konfigurācijas pārvaldnieks no risinājumu izvēles moduļa izvēlas vienu konkrētu risinājumu katrai darbībai. Rezultātā *PIAM* modelis paplašinās līdz *PSAM* modelim, kas satur informāciju par platformām, rīkiem, skriptiem utt.;
- *PSAM* modeli apstrādā servisu noteikšanas algoritms, kas nosaka rīku pārus integrācijai.

Visbeidzot — gan rīku integrācija, gan *PSAM* modelis tiek implementēti projekta konfigurācijas pārvaldības problēmvidē. *EAF* metodoloģija beidzas tad, kad konfigurācijas pārvaldības problēmvidē tiek implementēta izejas koda pārvaldības sistēma atbilstoši *SCBM*

modelim, realizētas visas integrācijas, kas attēlotas *SM* modelī, un ir implementēts *PSAM* modelis.

Vižu modeļa meta-modelis

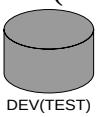
Meta-modelim jāveido vižu modeļi, kuru mērķis ir parādīt visas programmatūras izmaiņu plūsmas pa vidēm. Katra plūsma, pa kuru programmatūras vienumu versijas nonāk no vienas vides uz otru, pieder pie konkrēta notikuma. Vienam notikumam var piederēt vairākas plūsmas. Piemēram, ja notikums ir «Pārnest konfigurāciju no instances *DEV* uz instanci *TEST*», tad, iespējams, šajā notikumā būs iesaistītas divas plūsmas. Vienā plūsmā konfigurācija tiks pārnesta uz instanci *TEST1*, lai pārliecinātos, ka konfigurācija ir korekta, bet nākamajā plūsmā tā pati konfigurācija tiks pārnesta uz *TEST* instanci. Līdz ar to modelim vajadzētu parādīt visus notikumus, kas ir iesaistīti konfigurācijas pārnešanā starp dažādām instancēm, visas notikumu plūsmas, to secību un visas instances, kas glabā jebkāda veida programmatūras konfigurāciju. Vižu modeli veidos konfigurācijas pārvaldnieks, kuram būs iespēja pievienot, mainīt un dzēst notikumus, plūsmas un vides.

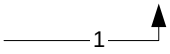
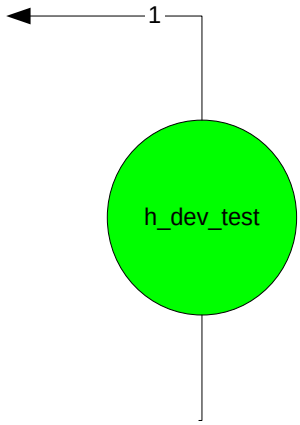
Meta-modeļa grafiski elementi un modeļu piemēri

Tabulā 3.1. ir dots apraksts visiem vižu modeļa meta-modeļa elementiem. Katram elementam ir noteikti atribūti, kuriem jābūt aizpildītiem modelēšanas gaitā, citādi pārveidošanas algoritms nevarēs izveidot *XML* dokumentu.

3.1. tabula

Vižu Modeļa elementi un to apraksts

Nosaukums/Izskats	Atribūti un to apraksts
Aktieris (*Actor*) 	Programmatūras izstrādātājs, kas veic izmaiņas programmatūras vienumos. Aktierim ir šādi atribūti: **Nosaukums (*Name*)** **Apraksts (*Description*)** Veidojot jaunu aktieri vižu modelī, lietotājam uzreiz tiek piedāvāts aizpildīt šos atribūtus.
Vide (*Environment*) 	Instance jeb vide, kurā atrodas izstrādājama programmatūra. Šim elementam ir šādi atribūti: **Nosaukums (*Name*)** — vides nosaukums **Apraksts (*Description*)** **Pazīme, kas nosaka, vai vidi uztur pasūtītājs (*CustumerSupportFlag*)** — pazīme, kas nosaka, vai konkrētu instanci uztur pasūtītājs vai izpildītājs. Ja vidi uztur pasūtītājs, tad izstrādātāju komandai nav pieejas, lai mainītu programmatūras konfigurāciju. Šajā gadījumā, lai nomainītu konfigurāciju, izstrādātājs sagatavo un nodod pasūtītājam laidienu, kuru izpildot, ir iespējams ieviest konkrētas izmaiņas vidē **Izstrādes vides pazīme (*DevelopmentFlag*)** — pazīme, kas nosaka, vai tā ir vai nav izstrādes vide. Izstrādes vidē aktieris (izstrādātājs) drīkst veikt izmaiņas brīvā veidā, visās citās vidēs izmaiņas nonāk tikai ar būvējumiem **Oriģinālas vides pazīme.** Vižu modelī ir divi vižu veidi. Pirmais veids ir oriģinālas vides, kuras lieto projekta konkrētiem mērķiem, tādiem kā izstrāde, testēšana, akcepttestēšana, ekspluatācija. Otrais veids ir vižu kopijas jeb neoriģinālas vides. Katra no šīm vidēm pēc būtības ir kādas oriģinālas vides kopija. Neoriģinālas vides jeb kopijas ir vajadzīgas, lai notestētu programmatūras

	izmaiņu pārnešanu starp vidēm. Praksē tas izpaužas, piemēram tad, kad uz ekspluatācijas vides jāuzliek kārtējais laidiens ar izmaiņām programmatūrā, taču pirms tam to pašu laidienu instalē uz ekspluatācijas precīzas kopijas, lai pārliecinātos, ka laidiens ir kvalitatīvs un pēc instalācijas īstajā ekspluatācijas vidē, negaidītas problēmas nerādīsies **Oriģinālas vides nosaukums (*OriginalEnvironmentName*)** — ja vide nav oriģināla, šeit norāda atbilstošas oriģinālas vides nosaukumu
Konfigurācijas vienumu plūsma (*ConfigurationItemFlow*) 	Apzīmē ceļu, pa kuru programmatūras izmaiņas nonāk no vienas vides uz otru vai no aktiera uz vidi, gadījumā, ja notiek izstrāde. Atribūti: **Nosaukums (*Name*)** — teksts, kas sniedz nelielu aprakstu par to, no kuras uz kuru vidi nonāk konfigurācijas vienumi **Kārtas numurs (*Sequence*)**. Katra plūsma pieder kādam konkrētam notikumam. Viena notikuma ietvaros var būt vairākas plūsmas. Atribūts parāda plūsmas kārtas numuru notikuma ietvaros **Avots (*Source*)** — vide vai aktieris, no kura nonāk programmatūras izmaiņas **Mērķa vide (*Goal*)** — vide, kur nonāk programmatūras izmaiņas **Apraksts (*Description*)** — papildu informācija par konkrētu plūsmu
Notikums (*Event*) 	Apzīmē notikumu, kura ietvaros programmatūras izmaiņas nonāk no vienas vides uz citām. Notikumam ir unikāls nosaukums, apraksts un vismaz viena konfigurācijas plūsma. Tādējādi viena notikuma ietvaros konfigurācijas vienumi var nonākt no vienas vides uz citām vidēm. Visām plūsmām viena notikuma ietvaros ir vienāda avota vide, tikmēr mērķa vides atšķiras. Viena konfigurācijas plūsma var piederēt tikai vienam notikumam. Notikuma atribūti ir šādi: **Nosaukums (*Name*)** **Plūsmas (*ConfigurationItemFlows*)** — norādes uz eksistējošām konfigurācijas vienumu plūsmām (*ConfigurationItemFlow*) **Apraksts (*Description*)** — papildu informācija par konkrētu notikumu **Tiek pārnestas visas pieejamas izmaiņas (*AllChangesMoveFlag*)** — atribūts, kas norāda, vai tiek pārnestas visas avota vidē pieejamas izmaiņas, vai tikai kādas noteiktas

No platformas neatkarīga darbību meta-modeļa izstrāde

No platformas neatkarīgs darbību modelis (turpmāk tekstā *PIAM*) parāda, kādas darbības ir jāveic, lai nodrošinātu visu plūsmu implementāciju vižu modelī. Līdz ar to modeļa galvenais mērķis ir parādīt visas darbības, kas ir nepieciešamas visu plūsmu nodrošināšanai, darbību

savstarpējas atkarības un darbību atribūtus. Atribūtu vērtības šajā modelī netiek aizpildītas, jo modelis nedrīkst būt atkarīgs no kādas konkrētas platformas vai tehnoloģijas.

PIAM meta-modeļa elementi atbilst konfigurācijas pārvaldības principam, ka visu disciplīnu var sadalīt noteiktos uzdevumos [AIE 2010, BER 2003] un rekomendāciju, ka visām darbībām jānotiek centralizēti — vienā vietā, lai izpilde nebūtu atkarīga no cilvēka darbstacijas, kurš izpilda kādu no darbībām, un lai visas atbildīgas personas varētu vienādi izpildīt vienas un tās pašas darbības [AIE 2010, PAU 2007, MET 2002]. Līdz ar to *PIAM* modelī vajadzētu iekļaut elementu, kas apraksta šādu centralizētu vietu, kur notiek visas konfigurācijas pārvaldības darbības. Tabulā 3.2. ir redzamas visas konfigurācijas darbības, kas varētu tikt iekļauti *PIAM* modelī.

3.2. tabula

***PIAM* meta-modeļa konfigurācijas pārvaldības darbības un atribūti**

Nosaukums	Apzīmējums	Apraksts
Izmaiņu izstrāde	*DEVELOPMENT*	Izmaiņu izstrādes darbība ir iekļauta *PIAM* modelī, jo konfigurācijas pārvaldība reglamentē, ka jābūt izstrādes noteikumiem [AIE 2010, MET 2002].
Izmaiņu saglabāšana versiju kontroles sistēmā	*COMMIT_CHANGES*	Darbība, kas nosaka izstrādātu izmaiņu saglabāšanu centralizētā repozitorijā. Darbību reglamentē versiju kontroles uzdevums, kas tika aprakstīts promocijas darba pirmajā nodaļā.
Versijas bāzes līnijas sagatavošana	*PREPARE_BASELINE*	Šī darbība atkarībā no versiju kontroles sistēmas reglamentē procedūru bāzes līnijas sagatavošanai. Konfigurācijas pārvaldība reglamentē, ka jebkādām izmaiņām ir jādefinē bāzes līnija, attiecībā pret kuru tiks veiktas izmaiņas [BER 2003]. Vižu modelis paredz, ka katrai oriģinālai videi ir sava bāzes līnija — produkta izejas koda stāvoklis, kas atbilst vides konfigurācijai.
Produkta būvējums	*COMPILE_BUILD*	Darbība, kas no atbilstoša izejas koda uzbūvē produktu.
Produkta instalācija	*INSTALL_BUILD*	Darbība, kas uzinstalē uzbūvētu produktu noteiktā vidē.
Produkta piegāde pasūtītājam	*PRODUCT_DELIVERY*	Darbība, kas sagatavo un nosuta pasūtītājam uzbūvētu produktu. Šī darbība ir nepieciešama, lai produktu varētu uzinstalēt vidē, ko uztur pasūtītājs un kurai izstrādātāju komandai nav

		pieejas, piemēram, ekspluatācijas vidē.
Informācijas saņemšana par vides atjaunošanu	*ENV_UPDATE_NOTIFICATION*	Darbība, kas nogādā izstrādātāju komandai signālu par to, ka kādā no vidēm, ko uztur pasūtītājs, ir uzinstalēta jauna produkta versija. Šajā momentā izstrādātāju komandai vajadzētu piefiksēt šo faktu, atbilstoši papildinot konfigurācijas vienumu informāciju versiju kontroles sistēmā, kā arī jāuzinstalē tādu pašu produkta versiju visās vidēs, kas ir kopijas atbilstošai pasūtītāja videi.

Tabulā 3.3. ir redzami *PIAM* darbību atribūti un to apraksts.

3.3. tabula

***PIAM* darbību atribūti un to apraksts**

Atribūts/apzīmējums	Darbības apraksts
Platforma (*Platform*)	Specifiskās platformas nosaukums, kas ir nepieciešams darbības realizācijai.
Risinājuma nosaukums (*SolutionName*)	Vienai un tai pašai darbībai var būt vairāki risinājumi.
Nepieciešami rīki (NeededTools)	Rīki, kas ir nepieciešami risinājuma implementācijai.
Risinājumu glabātuve (*LocationsOfSolutions*)	Šīs atribūts paredzēts, lai glabātu jebkādu informāciju par jau izstrādātajiem risinājumiem konkrētai darbībai. Te var glabāties, piemēram, skriptu nosaukumi, speciālu programmu atrašanas vietas utt.
Apraksts (*Description*)	Apraksts. Papildu apraksts konkrētai darbībai projekta vai risinājuma kontekstā, kas varētu sniegt papildu informāciju konfigurācijas pārvaldniekam.

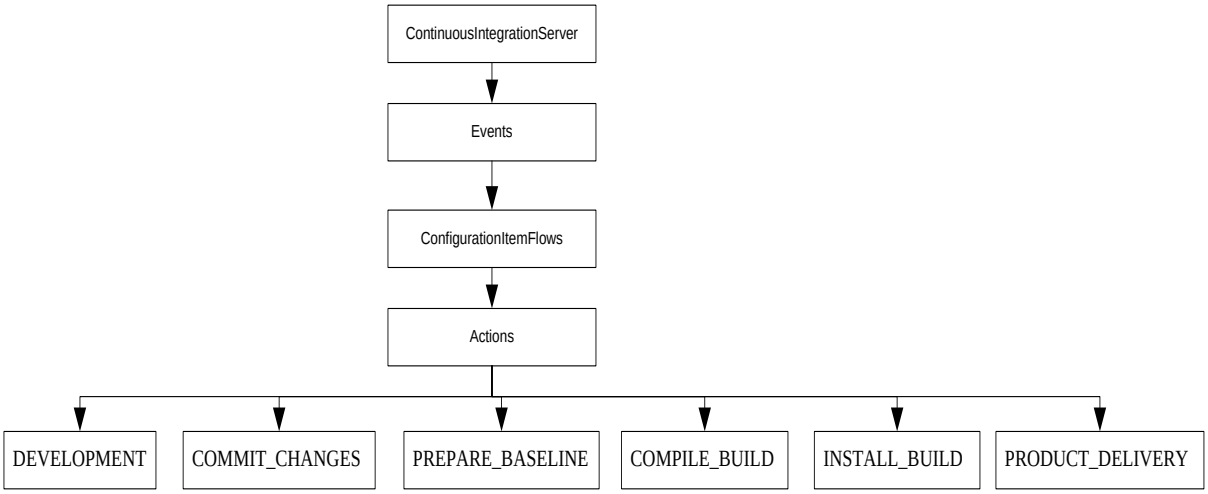
Konfigurācijas pārvaldības darbībām jānotiek vienā vietā. Šāda vieta parasti ir nepārtrauktās integrācijas serveris, kas arī tiks modelēts *PIAM* modelī. Līdz ar to *PIAM* meta-modelī tiek ieviests elements «*Continuous Integration Server*» ar šādiem atribūtiem:

- platforma (*Platform*) — konkrēta platforma, kurā tiek uzinstalēts serveris;
- rīka nosaukums (*ToolName*) — servera nosaukums;
- instalācijas norādījumi (*InstallationNotes*) — norādījumi par servera instalāciju;
- risinājumu glabātuve (*LocationOfSolutions*) — gatavu risinājumu atrašanas vietas (ja tādi ir).

Vēl viens elements *PIAM* meta-modelī ir «Notikumi» (*Events*), kas ietver sevī visus notikumus no vižu modeļa. Katrs notikums satur arī visas plūsmas, kas arī tiek paņemtas no vižu modeļa. Tādējādi, iegūstot *PIAM* modeli no *EM* modeļa, galvenais uzdevums ir katra notikuma (*Event*) katrai plūsmai noteikt nepieciešamas darbības no *PIAM* meta-modeļa un attēlot šo informāciju strukturētā veidā.

PIAM modelī būs redzamas tikai konfigurācijas pārvaldības darbības, kas ir vajadzīgas, lai implementētu visus notikumus vižu modelī, un darbību atribūti. Atribūtu vērtības tiks

aizpildītas tikai *PSAM* modelī, kur tiks specificēta platforma, tehnoloģijas, rīki utt. Attēlā 3.3. ir redzama *PIAM* meta-modeļa elementu hierarhija.



3.3. att. *PIAM* meta-modeļa elementu hierarhija.

PIAM meta-modeļa izstrādes gaitā tiek piedāvāts grafiskais attēlojums *PIAM* modelim, kas ir redzams attēlā 3.4.

ContinuousIntegrationServer			
Platform: <name>	**ToolName:** <name>	**InstallationNotes:** <notes>	**LocationsOfSolutions:** <locations>
Event: <name>		**Event:** <name>	
ConfigurationItemFlow: <name>	**Action:** <name> **Action:** <name>	**ConfigurationItemFlow:** <name>	**Action:** <name> **Action:** <name>
ConfigurationItemFlow: <name>	**Action:** <name> **Action:** <name>	**ConfigurationItemFlow:** <name>	**Action:** <name> **Action:** <name>
All Actions: Action1 Action2 Action3 .... ActionN			

3.4. att. *PIAM* modeļa grafiskais attēlojums.

Platformas specifiskā darbību modeļa realizācija

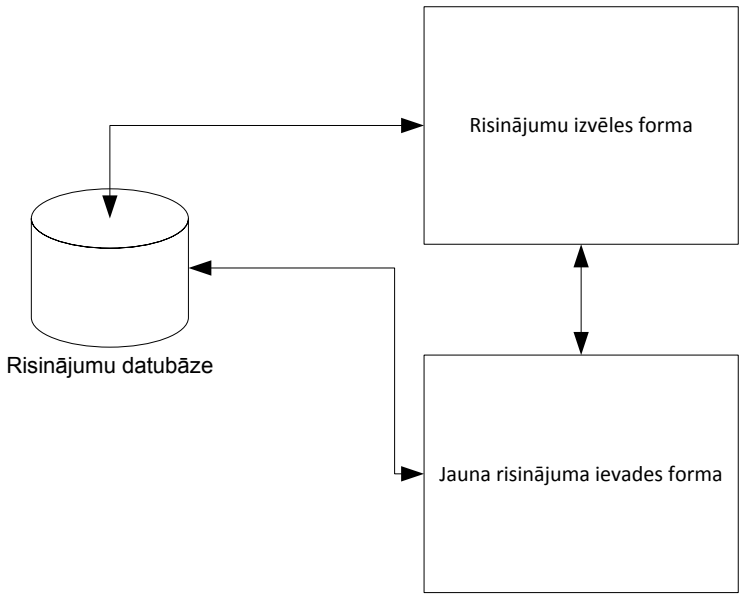
PSAM modeļa mērķis ir definēt implementāciju katrai darbībai *PIAM* modelī. *PSAM* modelis satur tās pašas komponentes kā *PIAM* modelis, tikai komponenšu atribūti jau ir aizpildīti ar konkrētām vērtībām, kas norāda uz konkrētu platformu, nepārtrauktas integrācijas servera nosaukumu, konkrētām programmām, skriptiem un citiem rīkiem, kas implementē konkrētus konfigurācijas pārvaldības uzdevumus.

PSAM modelim ir trīs galvenie mērķi:

- glabāt informāciju par esošajiem risinājumiem katrai konfigurācijas pārvaldības darbībai no *PIAM* modeļa;
- ļaut pārskatīt risinājumus un katrai darbībai no *PIAM* modeļa, ļaut izvēlēties vienu no esošajiem risinājumiem;
- ļaut ievadīt pilnīgi jaunu risinājumu jebkurai konfigurācijas pārvaldības darbībai un pēc tam attiecināt šo risinājumu pret darbību no *PIAM* modeļa;

- izveidot pārskatu, kurā tiek uzskaitītas visas darbības no *PIAM* modeļa un katrai darbībai ir definēts risinājums. Visi atribūti, kas *PIAM* modelī ir tukši, šajā pārskatā ir aizpildīti.

Lai īstenotu *PSAM* modeļa noteiktus mērķus, tika izstrādāts risinājumu izvēles modulis, kas ir redzams attēlā 3.5.

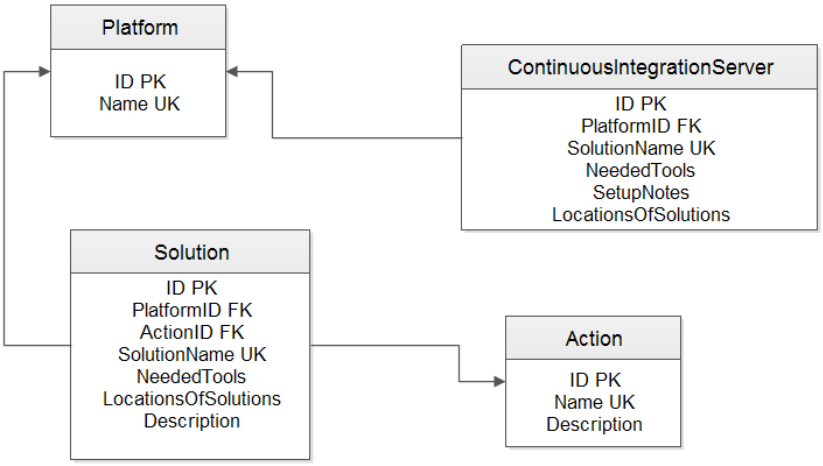


3.5. att. Risinājumu izvēles modulis.

Attēlā 3.5. redzamo elementu nozīme

- **Risinājumu datubāze** — konfigurācijas pārvaldības darbību risinājumu datubāze. Šajā datubāzē atrodas informācija par visiem jau realizētiem un pieejamiem risinājumiem katrai konfigurācijas pārvaldības darbībai, kas ir definēta *PIAM* meta-modelī.
- **Risinājumu izvēles forma** — konfigurācijas pārvaldnieka forma, kas ļauj katrai konfigurācijas pārvaldības darbībai izvēlēties konkrētu risinājumu no datubāzes.
- **Jauna risinājuma ievades forma** — lietotāja forma, kas ļauj ievadīt jaunu risinājumu kādai konkrētai konfigurācijas pārvaldības darbībai no *PIAM* modeļa. Ievadot risinājumu, tas kļūst pieejams risinājumu izvēles formā un to ir iespējams piesaistīt konkrētai konfigurācijas pārvaldības darbībai.

Attēlā 3.6. ir redzama risinājumu datubāzes *ER* diagramma, kurā tiek attēlotas minimālas prasības pret risinājumu datubāzi saskaņā ar *PIAM* meta-modeļa galvenajiem darbības principiem un mērķiem.



3.6. att. Konfigurācijas pārvaldības darbību risinājumu datubāze.

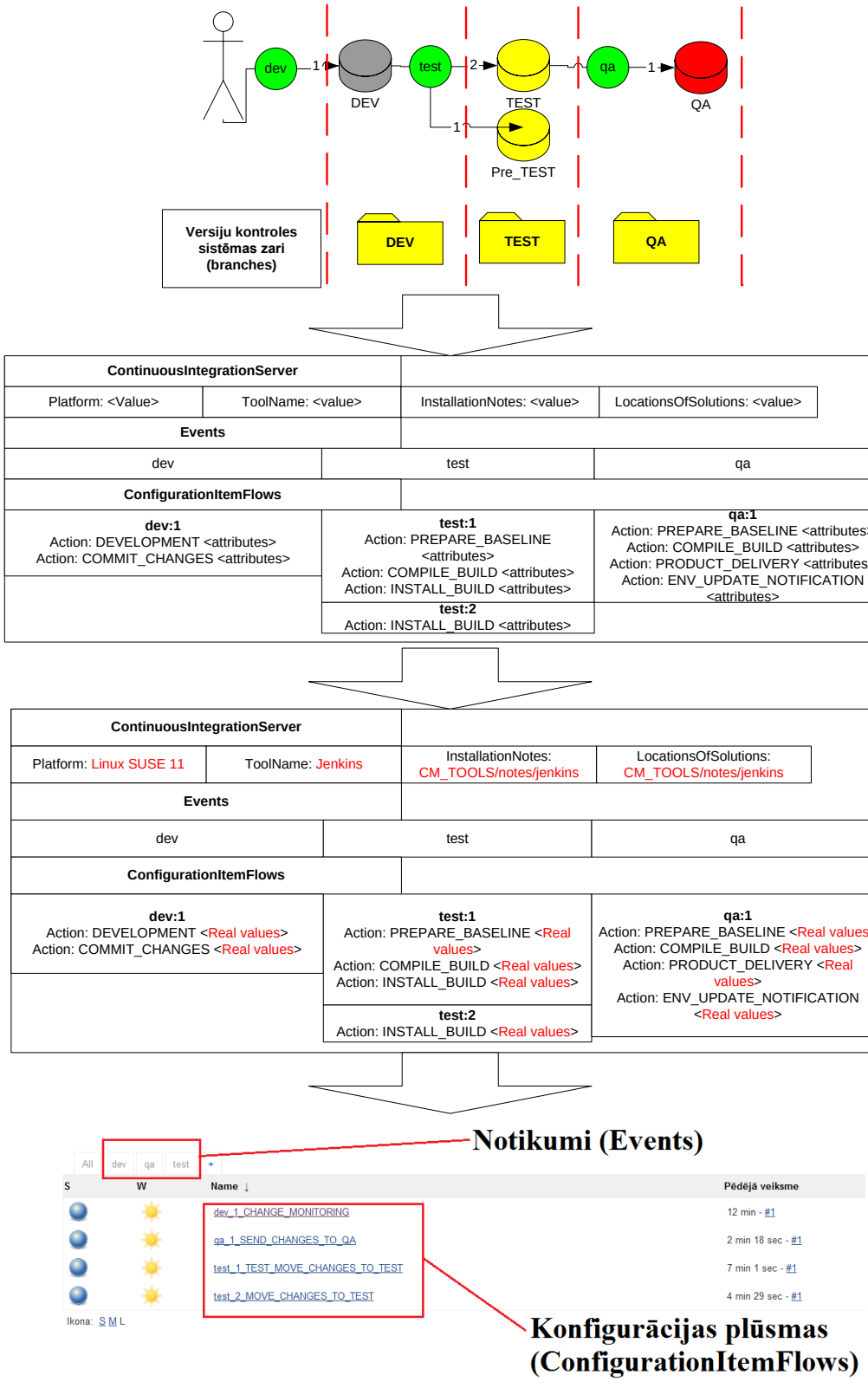
PSAM modeļa veidošanas algoritms

1. Saņem *PIAM* modeli *XML* formātā.
2. Nolasa elementu «*Actions*» un katru darbību ieliek risinājumu izvēles formā.
3. Lietotājs strādā ar risinājumu izvēles formu. Katrai darbībai (*Action*) no risinājumu datubāzes tabulas «*Solution*» tiek atlasīti visi risinājumi, kas atbilst šai darbībai un piedāvā lietotājam izvēlēties vienu no risinājumiem. Ja lietotājs visām darbībām ir izvēlējies risinājumus, tad *PIAM* modeļa *XML* fails tiek papildināts ar informāciju no risinājumu datubāzes un algoritms beidz savu darbību, atgriežot atjaunotu *XML* failu, kurā glabājas jau *PSAM* modelis.
4. Ja lietotājs pieņem lēmumu, ka kādai darbība nav piemērota risinājuma vai risinājumu datubāze ir tukša, tad notiek darbs ar risinājumu ievades formu. Lietotājs ievada jaunu risinājumu. Pēc risinājuma ievades lietotājs atgriežas algoritma 3. solī.

Attēlā 3.7. var redzēt *EAF* modeļu lietojumu projektam ar šādām vidēm:

- *DEV* — izstrādes vide;
- *TEST* — testa vide;
- *Pre_TEST* — vide, kur tehniski testē programmatūras būvējumus pirms likt tos *TEST* vidē,
- *QA* — akcepttesta vide.

Katrai no minētajām vidēm ir savs izejas koda zars versiju kontroles sistēmā. Attēlā 3.7. vižu modelim seko *PIAM*, *PSAM* modeļi un visu modeļu realizācija *Jenkins* nepārtrauktās integrācijas serverī.



3.7. att. *EAF* modeļu piemērs.

4. MODEĻVADĀMAS KONFIGURĀCIJAS PĀRVALDĪBAS METODOLOĢIJAS APROBĀCIJA UN TESTĒŠANA

Eksperimentu sagatavošana un plāns

EAF modeļu veidošanai ir izstrādāts programmatūras prototips. Izstrādē piedalījās bakalaurantūras students, pildot individuālu uzdevumu programmēšanā. Prototips izstrādāts, izmantojot *.NET, HTML5, CSS* un *JavaScript* tehnoloģijas. Papildus *JavaScript* tika izmantots arī *jQuery* un bibliotēka, kas ir paredzēta grafisku elementu zīmēšanai — *KineticJs*, kas atvieglo un uzlabo darbu ar *HTML5 Canvas* elementiem. Turpmāk tiek apsvērta iespēja projektu «pārnest» uz *AngularJs*, kas atvieglotu projekta uzturēšanu. Ar modeļu validāciju nodarbojās promocijas darba autors. Pēc veiktajām pārbaudēm prototipa darbība tika atzīta par korektu, lai gan tika izteikti vairāki priekšlikumi uzlabojumiem.

Prototips atvieglo šādu modeļu veidošanu:

- vižu modelis;
- *PIAM* modelis;
- *PSAM* modelis.

Papildus tam prototips ļauj veikt automātisku transformāciju no *EM* uz *PIAM* modeli atbilstoši «E–>P» transformācijas likumiem, kas tika aprakstīti promocijas darba iepriekšējā nodaļā.

Eksperimentiem tika izveidota kompetences grupa uzņēmumā SIA «*Tieto Latvia*». Grupā tika iekļauti vecākie un vadošie tehniski speciālisti, kas ikdienā strādā ar konfigurācijas pārvaldības automatizācijas procesiem.

Eksperimentu mērķi

- Salīdzināt konfigurācijas pārvaldības automatizācijas ieviešanas laiku pēc vecām metodēm un pēc *EAF* metodoloģijas.
- Salīdzināt kļūdainu būvējumu skaitu projektos pirms un pēc *EAF* metodoloģijas ieviešanas.
- Balstoties uz salīdzināšanu, noteikt, kā mainās konfigurācijas pārvaldības automatizācijas ieviešanas laiks, kļūdainu būvējumu skaits projektā, kā arī būvējumu kopējais skaits.

Eksperimentu nosacījumi

- Eksistē vismaz viens aktīvs programmatūras izstrādes projekts, kam ir kā minimums viena testa vide.
- Projektā ir ieviesta programmatūras konfigurācijas pārvaldība, ir realizēti galvenie konfigurācijas pārvaldības uzdevumi, kas ir aprakstīti promocijas darba pirmajā nodaļā.
- Konfigurācijas pārvaldības process vismaz daļēji ir automatizēts.

Eksperimentu metodika un darbības

- Eksperimentiem tika izvēlēti pieci programmatūras izstrādes un uzturēšanas projekti. Lai paaugstinātu eksperimentu ticamību, tika izvēlēti projekti ar dažādām izstrādes tehnoloģijām.
- Speciālistu kompetences grupai uzņēmumā SIA «*Tieto Latvia*» tika organizētas apmācības, kurās speciālisti tika iepazīstināti ar piedāvātu metodoloģiju un modeļiem.
- Tika izstrādāts risinājumu izvēles modulis konfigurācijas pārvaldības risinājumu glabāšanai, kas tika aprakstīts promocijas darba iepriekšējā nodaļā. Vadošie programmētāji izstrādāja programmatūru, kas ļauj pārvaldīt automatizācijas risinājumus konfigurācijas pārvaldības uzdevumiem. Programmatūrai bija šādi elementi:
  - *Oracle* datubāze atkārtoti lietojamu automatizācijas risinājumu glabāšanai;

- *Oracle ADF* forma, kas ļauj ievadīt datubāzē jaunu konfigurācijas pārvaldības automatizācijas risinājumu;
- *Oracle ADF* forma, kas saņem gatavu *PIAM* modeli un ļauj no minētas datubāzes izvēlēties automatizācijas risinājumu katrai konfigurācijas pārvaldības darbībai. Rezultātā tiek iegūts *PSAM* modelis.

- Izveidotais risinājumu izvēles modulis tika papildināts ar risinājumiem, veicot šādus soļus:
  - tabulas «*ContinuousIntegrationServer*» papildināšana ar risinājumiem konfigurācijas pārvaldības serveriem. Šajā tabulā tika ievietota informāciju par katru konfigurācijas pārvaldības serveri, kas tika lietots eksperimentos. Konfigurācijas pārvaldniekam, kas bija atbildīgs par katru konkrētu eksperimentālu projektu, vajadzēja sagatavot instrukciju, kā uzinstalēt konfigurācijas pārvaldības serveri, noteikt rīkus, kas ir vajadzīgi papildus, ka arī apkopot esošus risinājumus, kas ļauj atvieglot konfigurācijas pārvaldības servera sagatavošanu. Kad tas tika izdarīts, visu tikko minētu informāciju vajadzēja ielikt datubāzē, tabulā «*ContinuousIntegrationServer*». Tika aizpildīti šādi atribūti:
    - ***Platform*** — platforma, kurā funkcionē konkrētais konfigurācijas pārvaldības serveris;
    - ***SolutionName*** — servera unikāls nosaukums;
    - ***NeededTools*** — rīku uzskaitījums, ko ir jāuzinstalē, lai varētu aktivizēt konfigurācijas pārvaldības serveri;
    - ***SetupNotes*** — konfigurācijas pārvaldības servera detalizēta instalācijas instrukcija;
    - ***LocationsOfSolutions*** — gatavu risinājumu atrašanas vietas (ja tādi ir);
  - tabulas «*Solution*» aizpildīšana. Katram konfigurācijas pārvaldniekam, kas atbild par konkrētu programmatūras projektu, vajadzēja restrukturizēt atbilstošus automatizācijas risinājumus tādā veidā, kā to paredz *PSAM* modelis un risinājumu izvēles modulis. Veicot esošu automatizācijas risinājumu restrukturizāciju, katrs no tiem tiek ielikts tabulā «*Solution*», aizpildot šādus tabulas atribūtus:
    - ***Platform*** — platforma, kurā funkcionē dotais automatizācijas risinājums;
    - ***Action*** — kuru konfigurācijas pārvaldības darbību automatizē;
    - ***SolutionName*** — automatizācijas risinājuma unikāls nosaukums;
    - ***NeededTools*** — rīki, kas nepieciešami risinājuma ieviešanai un lietošanai;
    - ***LocationsOfSolutions*** — atkārtoti izpildāma koda atrašanas vieta;
    - ***Description*** — papildu norādījumi par risinājuma ieviešanu.
- Tika izstrādāti *EAF* metodoloģijas vērtēšanas kritēriji un rādītāji, kas ir nepieciešami kritēriju aprēķinam. Par katru no pieciem projektiem no SIA «*Tieto Latvia*» korporatīvas darba laika uzskaites sistēmas tika iegūti šādi dati:
  - laiks, kas tika patērēts konfigurācijas pārvaldības procesu sākotnējai ieviešanai;
  - vidējais laiks nedēļā, kas tika patērēts konfigurācijas pārvaldības procesu regulārai uzturēšanai;
  - nedēļu skaits līdz paredzētam projekta beigu datumam.

No katra projekta konfigurācijas pārvaldības datubāzes tika iegūta šāda informācija:

- programmatūras būvējumu skaits;
- programmatūras kļūdainu būvējumu skaits.

Katrā no pieciem projektiem tika veikts *EAF* metodoloģijas ieviešanas eksperiments pēc šāda plāna:

- tika izveidots vižu modelis. Modelī tika iekļautas visas izstrādes un testa vides, ka arī ekspluatācijas vide;
- vižu modelis tika transformēts *PIAM* modelī un *SCBM* modelī;

- konfigurācijas pārvaldnieks, strādājot ar risinājumu izvēles moduli, papildināja konfigurācijas pārvaldības darbības *PIAM* modelī ar implementācijas detaļām. Rezultāta tika iegūts *PSAM* modelis;
- no *PSAM* modeļa tika iegūts *SM* modelis, kas parādīja visus rīkus, ko vajadzēja integrēt savā starpā;
- konfigurācijas pārvaldnieks izstrādāja servisu modeli (SM) un izejas koda pārvaldības sistēmu atbilstoši *SCBM* modelim;
- *PSAM* modelis tika implementēts konfigurācijas pārvaldības serverī;
- tika piefiksēts laiks, kas tika patērēts, sākot ar vižu modeļa veidošanu un beidzot ar *PSAM* modeļa implementāciju;
- programmatūras konfigurācijas pārvaldības process funkcionēja pēc *EAF* metodoloģijas triju mēnešu laikā. Pēc tam tika piefiksēti šādi rādītāji:
  - vidējais laiks nedēļā, kas bija nepieciešams procesu uzturēšanai un procesa kļūdu labojumiem;
  - programmatūras būvējumu kopējais skaits;
  - programmatūras kļūdainu būvējumu skaits.
- Tika organizēta sapulce, kurā kompetences grupas dalībnieki izvērtēja sākotnēji iegūtus un eksperimentu gaitā piefiksētus rādītājus par patērētu laiku un būvējumu skaitu.

Modeļvadāmas pieejas vērtēšanas kritēriji

EAF metodoloģijas novērtēšanai tika izstrādāti vērtēšanas kritēriji

- **Procesu ieviešanas laika starpība.** Kritērijs, kas procentuāli parāda starpību starp laiku, kas tika patērēts konfigurācijas pārvaldības automatizācijas ieviešanai pēc vecām metodēm un pēc jaunās EAF metodoloģijas. Ja vērtība ir pozitīva, tas nozīmē, ka procesu ieviešana pēc jaunās metodoloģijas aizņēmu vairāk laika nekā ieviešana pēc veciem paņēmieniem.
- **Regulāra uzturēšanas laika starpība.** Kritērijs, kas procentuāli parāda starpību starp laiku, kas bija nepieciešams procesu manuālai uzturēšanai pirms un pēc *EAF* metodoloģijas ieviešanas. Ja vērtība ir negatīva, tas nozīmē, ka pēc *EAF* metodoloģijas ir nepieciešams mazāk laika automatizācijas procesu manuālai uzturēšanai. Savukārt pozitīvas vērtības gadījumā nepieciešams patērēt vairāk laiku procesu uzturēšanai pēc *EAF* metodoloģijas.
- **Kopēja uzturēšanas laika starpība.** Kritērijs, kas ļauj spriest par *EAF* metodoloģijas ieviešanas ilgtermiņa ieguvumu. Ņem vērā laiku līdz projekta noslēgumam, laiku, kas bija nepieciešams *EAF* ieviešanai, un laiku, ko vidēji patērē procesu manuālai uzturēšanai pirms un pēc *EAF* ieviešanas. Procentuāli parāda starpību starp kopēju laiku, kas būtu nepieciešams procesu uzturēšanai līdz projekta beigām pēc veciem paņēmieniem un starp kopēju laiku, kas būtu nepieciešams procesiem sakarā ar *EAF* ieviešanu.
- **Kļūdainu būvējumu starpība.** Kritērijs procentuāli parāda, kā izmainījās kļūdainu būvējumu skaits projektā pēc *EAF* metodoloģijas ieviešanas.
- **Būvējumu kopēja skaita starpība.** Kritērijs procentuāli parāda, kā izmainījās kopējais būvējumu skaits projektā pēc *EAF* metodoloģijas ieviešanas.

Tabulā 4.1. ir redzami aprēķināti projektu vērtēšanas kritēriji.

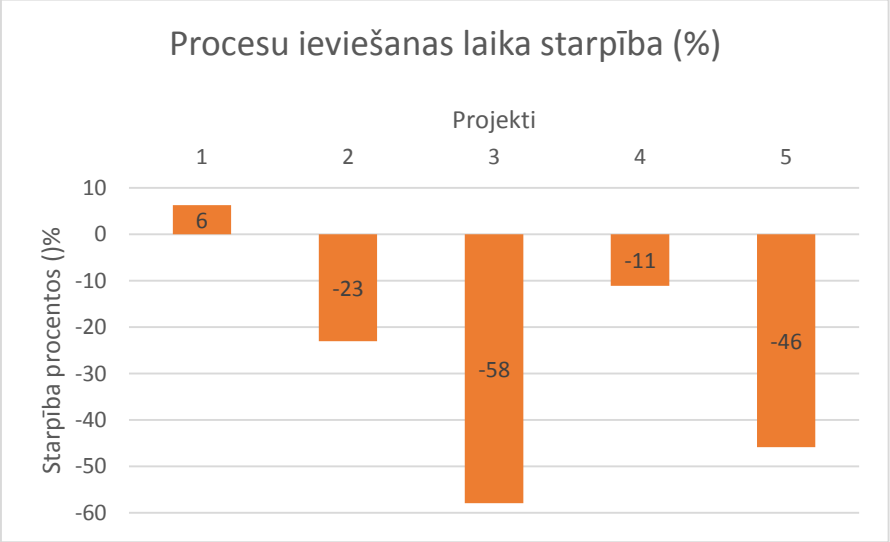
4.1. tabula

Projektu vērtēšanas kritēriji

Projekts	Kritēriji				
	Procesu ieviešanas laika starpība (%)	Regulāra uzturēšanas laika starpība (%)	Kopēja uzturēšanas laika starpība (%)	Kļūdainu būvējumu starpība (%)	Būvējumu kopēja skaita starpība (%)
1	6	–10	23	–27	4
2	–23	–7	36	–40	–43
3	–58	–25	–10	–33	3
4	–11	–38	–28	–60	–3
5	–46	0	71	–29	3

Automatizācijas procesu ieviešanas laika starpība

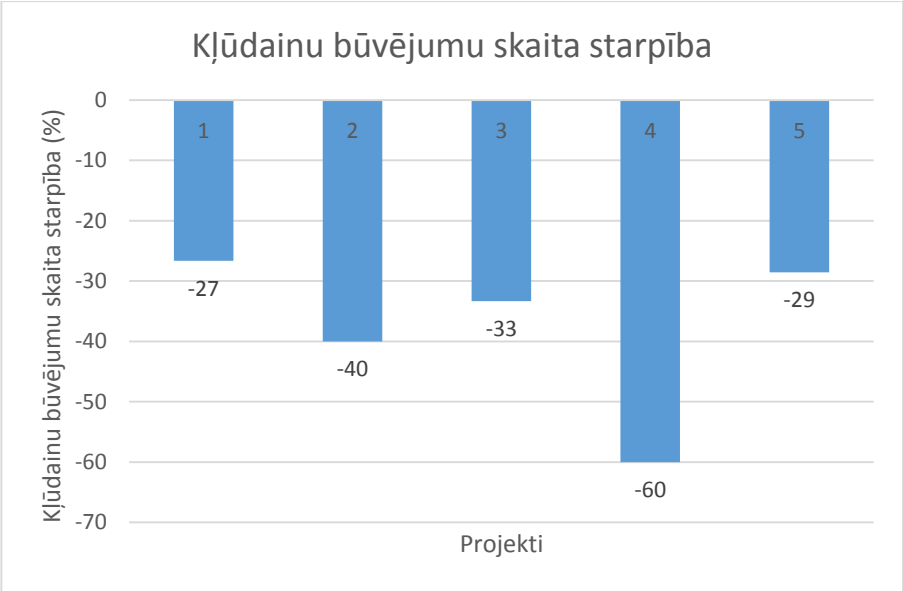
Attēlā 4.1. ir redzams grafiks, kur horizontāla ass satur projekta numuru, bet vertikāla ass — ieviešanas laika starpību procentos.



4.1. att. Automatizācijas procesu ieviešanas laika starpības salīdzinājums pa projektiem.

Kļūdainu būvējumu skaita izmaiņu analīze

Attēlā 4.2. ir redzams, kā mainījās kļūdainu būvējumu skaits pēc *EAF* metodoloģijas ieviešanas visos piecos projektos. Grafiks atklāja vēl vienu *EAF* metodoloģijas būtisku ieguvumu: kļūdainu būvējumu skaita samazināšanās. Samazināšanas tendenci parāda fakts, ka vidēji par 38 % samazinājās kļūdainu būvējumu skaits.



4.2. att. Kļūdainu būvējumu skaita starpība.

Veicot minētos eksperimentus un analizējot to rezultātus, tika konstatēti šādi *EAF* metodoloģijas ieguvumi:

- metodoloģija samazina laiku, kas ir nepieciešams konfigurācijas pārvaldības procesu ikdienas uzturēšanai. Pateicoties tam, ka visas konfigurācijas pārvaldības darbības izpildās no centralizētas vietas, pieaug automatizācijas un pārskatāmības līmenis. Visām darbībām ir atbilstošs izejas kods, kas ļauj izvairīties no manuālām darbībām. Veicot eksperimentus piecos projektos, vidēji par 16 procentiem samazinājās laiks, kas bija nepieciešams procesu ikdienas uzturēšanai;
- metodoloģija krietni samazina kļūdainu būvējumu skaitu. Veidojot izejas kodu katrai konfigurācijas pārvaldības darbībai, praksē tiek pārskatīti daži soļi, pievienotas papildu kvalitātes pārbaudes, kas nebija līdz šim, uzlabota kļūdu apstrāde un žurnālifikācijas sistēma. Tas ļāva samazināt kļūdainu būvējumu skaitu vidēji par 38 procentiem;
- konfigurācijas pārvaldības automatizācijas ieviešana aizņem mazāk laika nekā ieviešana pēc veciem paņēmieniem ar nosacījumu, ka risinājumu izvēles modulis satur gatavus un notestētus risinājumus atsevišķu konfigurācijas pārvaldības darbību automatizācijai. Eksperimenti parādīja, ka, ja risinājumu izvēles modulis satur implementācijas konfigurācijas pārvaldības darbībām, tad procesu ieviešanas laiks pēc *EAF* metodoloģijas ir vidēji par 34 procentiem mazāks.

Tabulā 4.2. var redzēt *EAF* metodoloģijas konstatētu trūkumu apkopojumu.

4.2. tabula

EAF metodoloģijas trūkumi

Trūkums kārtas numurs	Apraksts
1	Risinājumu izvēles moduļa struktūra. Eksperimentu rezultāti atklāja, ka strukturēt konfigurācijas pārvaldības izejas kodu pēc konfigurācijas pārvaldības galvenajiem uzdevumiem (*compile, deploy, prepare baseline*) ir pārāk plaši. Šajā gadījumā funkcijas satur ļoti daudz parametru, un funkcijas ķermenis satur daudz atzarojumu.
2	Vižu modeļa struktūra. Esošā vižu modeļa interpretācija ļoti ierobežo projektus. Pirmkārt, vižu modelī jāparedz iespēja, ka programmatūras pārnešanas starp vidēm notiks vairākos notikumos (*Event*) un konfigurācijas plūsmas (*ConfigurationItemFlow*) arī var būt sadalītas sīkāk atkarībā no projektu specifikas. Otrkārt, kā atzīmēja konferenču rakstu recenzenti un tehniski speciālisti, jēdzieni Notikums (*Event*) un Konfigurācijas plūsma (*ConfigurationItemFlow*) nav intuitīvi saprotami. Līdz ar to būtu nepieciešams atrast veidu, kā vienkāršāk strukturēt konfigurācijas pārvaldības darbības, kas pārnes programmatūras izmaiņas starp vidēm. Papildus tam, vižu modelēšanas gaitā konfigurācijas pārvaldniekam būtu jāsniedz iespēja brīvāk strukturēt darbības pa notikumiem un plūsmām. Visbeidzot, jāpārskata notikumu un plūsmu jēdzienu, lai konfigurācijas pārvaldniekiem jēdzieni būtu intuitīvi saprotamāki.
3	*PIAM* modeļa būtība. Darbību kopa, kas ir aprakstīta *PIAM* meta-modelī, nav pilnīga. Ir jābūt iespējai pielikt klāt jaunas darbības. Papildus tam, transformācija no *EM* uz *PIAM* modeli ļoti ierobežo projektus, kuriem ir jāveic vēl citas darbības, kas nav definētas transformācijas likumos. Iesniedzot modeļu aprakstus zinātniskajai konferencei *MODELSWARD 2015*, tika saņemts ieteikums apvienot *EM* un *PIAM* modeļus, ļaujot lietotājam pašam izvēlēties darbības, kā arī paplašināt darbību kopu meta-modelī.
4	Izejas koda zarošanas modelis neatspoguļo dažādas izejas koda pārvaldības stratēģijas. Ir projekti, kam jau ir citas stratēģijas, un šajā gadījumā *EAF* metodoloģijas ieviešanu apgrūti fakts, ka metodoloģija paredz noteiktu zarošanas pieeju. Līdz ar to radās ieteikums zarošanas pieeju pasniegt vien kā rekomendāciju, taču atstāt projektiem zināmu brīvību zaru nosaukumu un zarošanas pieejas izvēlē.
5	Servisu modelis neparedz darbību ar vairākām instancēm un tehnoloģijām. Pieņemsim, ir situācija, kad konkrēta programmatūras laidiena aprakstam ir nepieciešama informācija no vairākām dažādām pieteikumu apstrādes sistēmām. Šajā gadījumā servisu modelim jābūt pietiekami elastīgam, lai ļautu pieslēgties dažādām sistēmām tā, lai funkcijām nevajadzētu ņemt vērā projektu specifiku.

EAF metodoloģijas pamati un testēšanas rezultāti ir atspoguļoti publikācijās [BAR 2014a, BAR 2014b, BAR 2014c, BAR 2014d, BAR 2014e, BAR 2015].

***EAF* metodoloģijas otrās kārtas izstrādes un atkārtotu eksperimentu nepieciešamības pamatojums**

Eksperimentu rezultāti parādīja, ka *EAF* metodoloģija ļauj samazināt konfigurācijas pārvaldības automatizācijas ieviešanas laiku. Ieviešot konfigurācijas pārvaldības automatizāciju piecos projektos, vidēji par 34 % samazinās ieviešanas laiks salīdzinājumā ar automatizācijas ieviešanu pēc vecām metodēm. Šī tendence no vienas puses ļautu uzskatīt, ka promocijas darba mērķis ir sasniegts. Taču gan eksperimentu gaitā, gan arī, publicējot *EAF* metodoloģijas pamatus starptautisku konferenču rakstu krājumos, tika konstatēti būtiski priekšnosacījumi otrajai izstrādes kārtai:

- risinājumu izvēles modulis. Kad konfigurācijas pārvaldības automatizācija tika ieviesta pēdējā no pieciem projektiem, tika konstatēts, ka atkārtoti izpildāms izejas kods palika grūti uzturams. Apspriežot rezultātus ar vadošajiem kompetences grupas speciālistiem, kas piedalījās eksperiemntu organizācijā, tika konstatēts fakts, ka pie esošās realizācijas *EAF* metodoloģija nespēj pilnībā sniegt atkartoti lietojamu izejas kodu automatizācijas procesiem;
- recenzenta atsauksmes rakstam [BAR 2015]. Jāatīmē, ka šis raksts tiks pieteikts konferencei *MODELSWARD 2015*, kas bija veltīta gan *MDA*, gan *MDD* jaunākajiem sasniegumiem. Lai gan raksts tika akceptēts, viens no recenzentiem atzīmē būtiskus trūkumus *PIAM* modelī, kas būtībā ierobežo programmatūras konfigurācijas pārvaldniekus veidot jaunas darbības, kā arī mainīt to secību. Recenzents ieteica apvienot vižu modeli un no platformas neatkarīgu darbību modeli vienā, lai lietotājs varētu brīvi modelēt ne tikai vides, bet arī darbības.

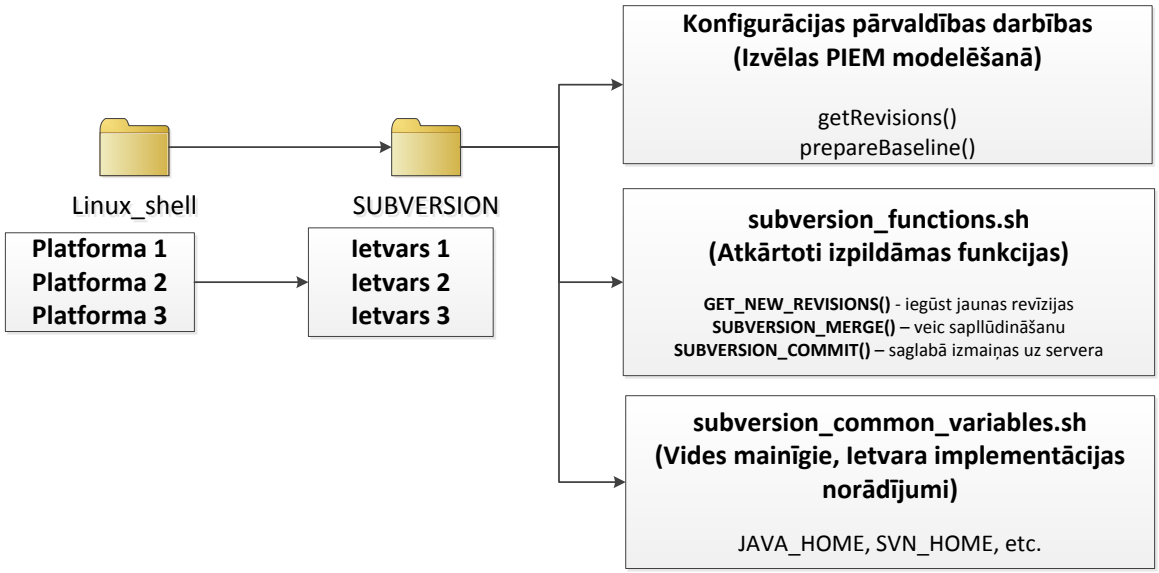
Sakarā ar to, ka risinājumu izstrādes modulis, *EM* un *PIAM* modeļi ir pamata elementi *EAF* metodoloģijai un šajos elementos bija nepieciešamas izmaiņas, tika pieņemts lēmums organizēt *EAF* metodoloģijas otru izstrādes kārtu. Galvenais mērķis bija uzlabot risinājuma izvēles moduļa struktūru un apvienot *EM* un *PIAM* modeļus.

Ņemot vērā, ka uz eksperimentu noslēguma brīdi *EAF* metodoloģijas pamati un eksperimentu rezultāti bija publicēti zinātniskajos rakstos un metodoloģijā bija nepieciešams mainīt pamatelementus, tika pieņemts lēmums veikt arī atkārtotus eksperimentus. Otrā eksperimentu kārta ir vajadzīga, jo tiks mainīti vairāki pamatelementi *EAF* metodoloģijai, kā rezultātā tā kļūs citādāka. Līdz ar to vajadzēs pārliecināties ne tikai par to, vai ir novērsti pirmajā versijā konstatēti trūkumi, bet arī par to, ka nav nograuti kādi no ieguvumiem.

5. *EAF* METODOLOĢIJAS UZLABOŠANA

Risinājumu datubāze

Risinājumu datubāze — metode, kas parāda, kā glabāt atkārtoti lietojamus automatizācijas risinājumus konfigurācijas pārvaldības darbībām un lietot šos risinājumus *EAF* modeļos. Metode ietver sevī atkārtoti lietojamu risinājumu struktūru un risinājumu atlases algoritmu. Attēlā 5.1. ir redzama risinājumu datubāzes struktūra.



5.1. att. Risinājumu datubāzes uzlabotā struktūra.

Kā var redzēt attēlā 5.1., visi atkārtoti lietojamie risinājumi ir sagrupēti pa platformām un ietvariem. Savukārt katram ietvaram ir šādi galvenie atribūti:

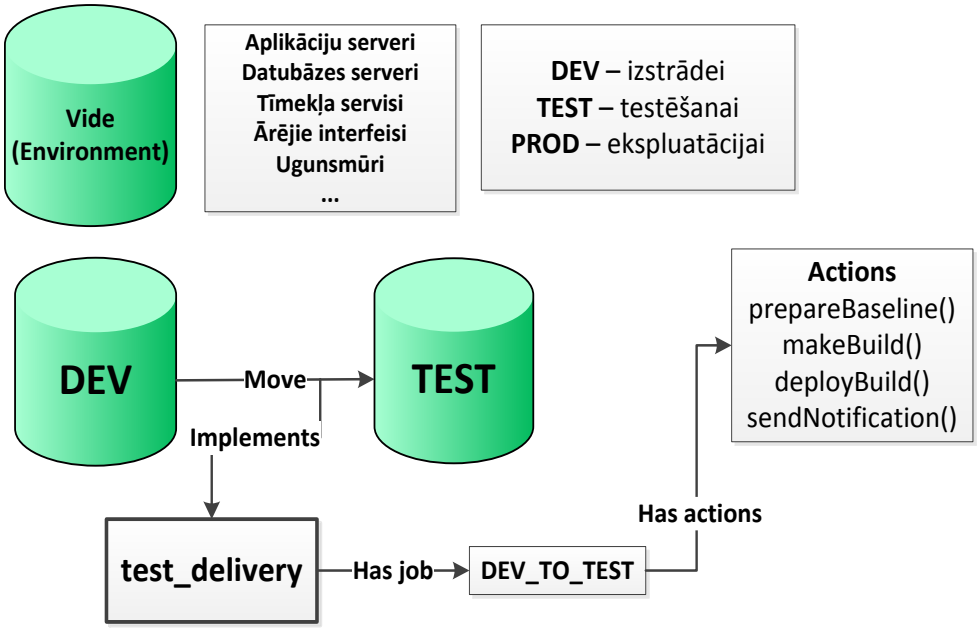
- konfigurācijas pārvaldības darbības, ko automatizē ar *EAF* metodoloģijas palīdzību un ko definē, modelējot projekta vides;
- atkārtoti izpildāmas funkcijas;
- vides mainīgie un ietvara implementācijas norādījumi.

Strādājot ar 5.1. attēlā redzamu risinājumu datubāzi *EAF* ietvaros, konfigurācijas pārvaldnieks veic šādas darbības:

- izvēlas platformu visu konfigurācijas pārvaldības darbību implementācijai. Šajā brīdī konfigurācijas pārvaldniekam kļūst pieejamas tikai tādi ietvari, kas atbilst izvēlētai platformai;
- katrai darbībai tiek izvēlēts ietvars. Šajā brīdī konfigurācijas pārvaldnieks saņem ietvara funkcijas kā atkārtoti lietojamu izejas kodu noteiktai platformai un ietvara implementācijas norādījumus.

No platformas neatkarīgai vižu modelis (angļ. val. *Platform Independent Environment Model, PIEM*)

Šis modelis ir apvienojums iepriekšējā nodaļā definētiem *EM* un *PIAM* modeļiem, kas ne tikai parāda projekta vides līdzīgi kā *EM* modelis, bet arī ļauj konfigurācijas pārvaldniekam definēt konfigurācijas pārvaldības darbību struktūru. Līdzīgi kā *EM* modelis, arī jaunais *PIEM* modelis nesatur nekādas detaļas par konfigurācijas pārvaldības darbību implementāciju konkrētai platformai. Attēlā 5.2. var redzēt piemēru jaunizstrādātājam *PIEM* modelim.



5.2. att. No platformas neatkarīgais vižu modelis.

EAF metodoloģijas uzlabotās versijas darbības piemērs

EAF metodoloģijas uzlabotajā versijā tika izdarīts:

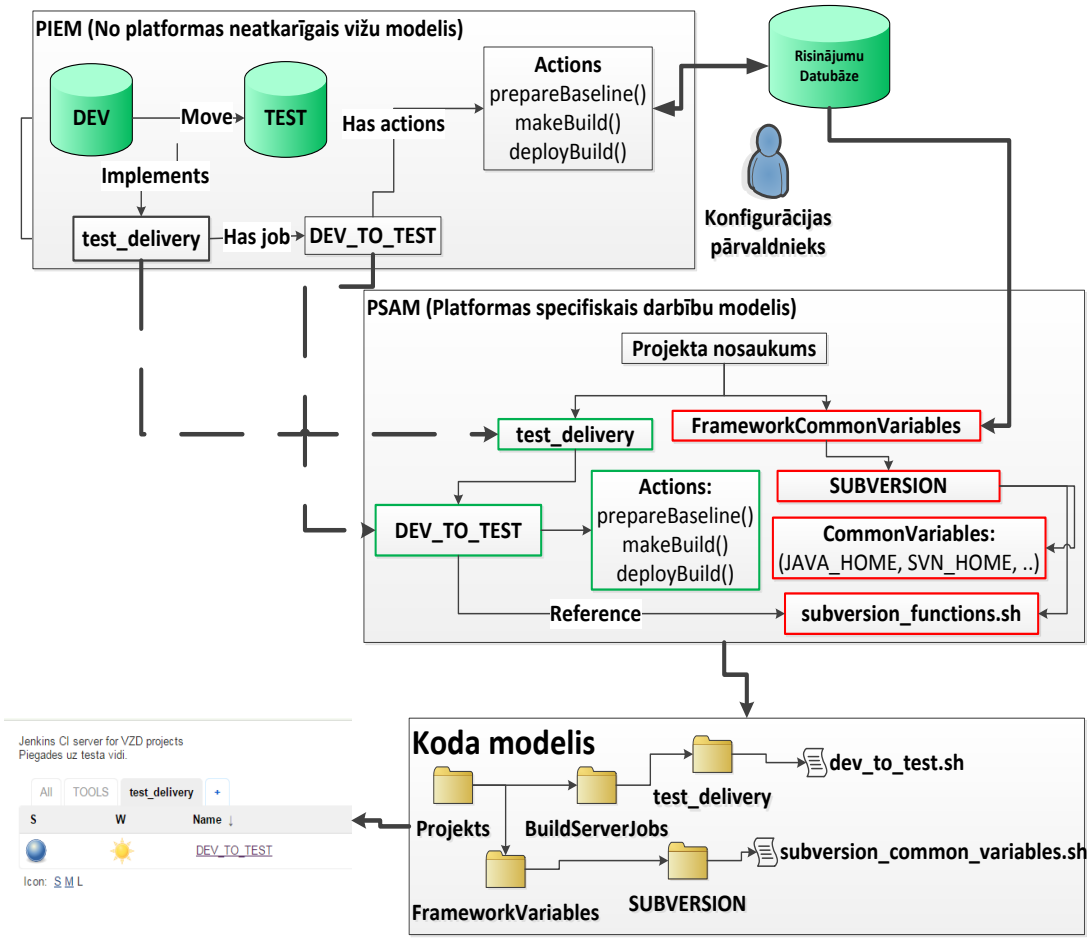
- pārveidota risinājumu datubāze;
- apvienoti *EM* un *PIAM* modeļi, izveidojot *PIEM* (no platformas neatkarīgais vižu modelis). Galvenais mērķis bija ļaut konfigurācijas pārvaldniekam pašam definēt struktūru automatizējamām konfigurācijas pārvaldības darbībām;
- ieviests koda modelis — direktoriju un failu struktūra, kas automātiski uzģenerējas no platformas specifiskā darbību modeļa, ņemot vērā konkrētas platformas un konkrētas programmēšanas valodas likumus.

Attēlā 5.3. var redzēt *EAF* uzlabotās versijas modeļu lietošanas piemēru.

Piemērs, kas ir redzams attēlā 5.3., ilustrē situāciju, kad kādā programmatūras izstrādes projektā ir divas vides: *DEV* — izstrāde vide un *TEST* — testa vide. Ir nepieciešams regulāri pārnest programmatūras izmaiņas no izstrādes uz testa vidi. Procesam ir jābūt pilnībā automatizētam. Tika pieņemts lēmums, ka automatizācija tiks īstenota ar *Jenkins* nepārtrauktās integrācijas servera palīdzību. Serveris tiks instalēts *Linux* platformā, un konfigurācijas pārvaldības darbības tiks automatizētas ar *Linux Shell* skriptiem. *EAF* uzdevums ir uzģenerēt izejas kodu šādiem skriptiem.

Attēlā 5.3. var redzēt *EAF* modeļus un darbības soļus:

- konfigurācijas pārvaldnieks, modelējot vides un konfigurācijas pārvaldības darbības, veido *PIEM* modeli;
- no *PIEM* modeļa veidojas *PSAM* modelis (platformas specifiskais darbību modelis). Konfigurācijas pārvaldības darbību struktūra, kas *PSAM* modelī apzīmēta ar zaļo krāsu, tiek nokopēta no *PIEM* modeļa. Savukārt implementācijas detaļas piešķir konfigurācijas pārvaldnieks, izvēloties katrai darbībai ietvarus no risinājumu datubāzes;
- no *PSAM* modeļa tiek uzģenerēts koda modelis, kas ir *Linux Shell* skriptu komplekts *PIEM* modelī definēto konfigurācijas pārvaldības darbību automatizācijai.



5.3. att. No platformas neatkarīgais vižu modelis.

***EAF* metodoloģijas uzlabotās versijas testēšana**

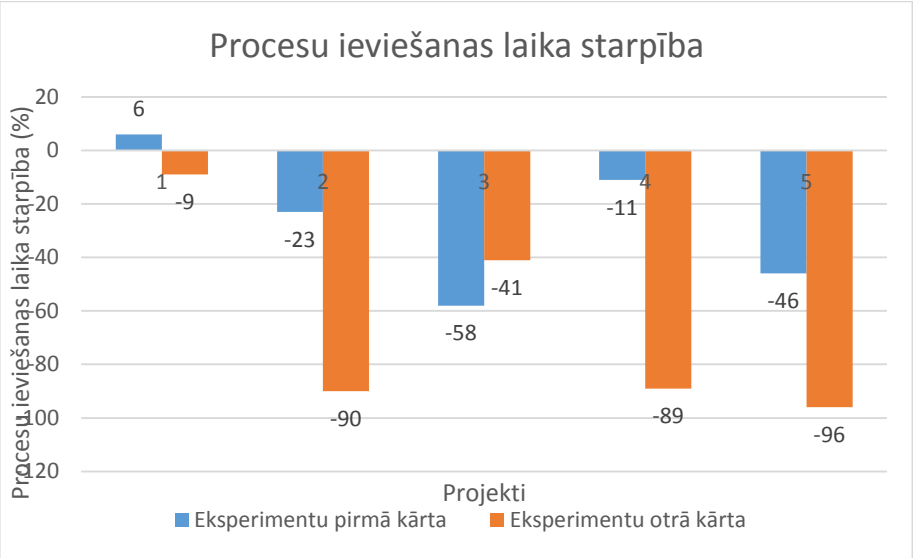
Testējot *EAF* metodoloģijas jaunu versiju, tika veikti eksperimenti ar tiem pašiem projektiem, kas tika aprakstīti promocijas darba iepriekšējā nodaļā. Lai eksperimentu rezultātus varētu salīdzināt ar pirmās kārtas eksperimentiem, tika izmantoti tādi paši vērtēšanas kritēriji un tādi paši rādītāji. Tabulā 5.1. ir redzami kritēriji eksperimentu otrajai kārtai.

5.1. tabula

Otrās kārtas eksperimentu rezultātu apkopojums

Kritēriji				
Procesu ieviešanas laika starpība (%)	Regulāra uzturēšanas laika starpība (%)	Kopēja uzturēšanas laika starpība (%)	Kļūdainu būvējumu starpība (%)	Būvējumu kopēja skaita starpība (%)
–9	–20	8	–33	–2
–90	–7	–1	–20	–39
–41	–25	–3	–67	–2
–89	–50	–49	–70	3
–96	0	5	–57	2

Pateicoties EAF metodoloģijas uzlabojumiem, izdevās krietni samazināt procesu ieviešanas laiku. Vidējais rādītājs procesu ieviešanas laika samazināšanai visos piecos projektos ir 65 procenti, kas ir ļoti labs rādītājs, ņemot vērā, ka sākotnēji risinājumu datubāze (*Solution Database*) bija tukša. Attēlā 5.1. var redzēt grafiku, kurā tiek salīdzināta ieviešanas laika starpība eksperimentu pirmajā un otrajā kārtā.



5.2. att. Procesu ieviešanas laika salīdzinājums divām eksperimentu kārtām.

Analizējot otrās kārtas eksperimentu rezultātus, tika konstatēti *EAF* metodoloģijas ieguvumi.

- Ieviešot *EAF* metodoloģiju minētajos piecos projektos, vidēji par 65 procentiem samazinājās konfigurācijas pārvaldības automatizācijas ieviešanas laiks. Tas rāda tendenci, ka esošu automatizācijas risinājumu izmantošana tiešām ļauj būtiski samazināt automatizācijas ieviešanu jaunajos projektos.
- *EAF* metodoloģija palīdz atbrīvot projektu no manuālām darbībām konfigurācijas pārvaldības procesu uzturēšanā. Pateicoties metodoloģijas principam, ka konfigurācijas pārvaldība izpildāms izejas kods, manuālas darbības vairs netiek veiktas. Eksperimenti rāda, ka projektos ar salīdzinoši zemu automatizācijas līmeni *EAF* metodoloģija krietni to uzlabo.
- *EAF* metodoloģija eksperimentālajos projektos par 49 procentiem samazināja kļūdainu būvējumu skaitu. Eksperimenti parādīja, ka, pateicoties pārdomātai kļūdu apstrādei un automatizētai rīku savstarpējai integrācijai, kļūdainu būvējumu skaits projektā samazinās. Pateicoties risinājumu izvēles moduļa pilnīgai restrukturizācijai, kļūdainu būvējumu skaits kļuva vēl aptuveni par 10 % mazāk. Šī tendence apliecina faktu, ka, glabājot atkārtoti lietojamu izejas kodu konfigurācijas pārvaldības automatizācijai pēc jaunās metodes, uzlabojas koda atkārtota lietojamība.

EAF metodoloģijas uzlabotās versijas pamati un otrās kārtas eksperimentu rezultāti tika publicēti zinātniskajā rakstā [BAR 2015a].

DARBA KOPĒJIE REZULTĀTI, SECINĀJUMI UN TURPMĀKIE PĒTĪJUMI

Promocijas darba mērķis bija izstrādāt modeļvadāmu pieeju un metodoloģiju konfigurācijas pārvaldības procesu automatizācijas ieviešanai, kas ļautu samazināt automatizācijas ieviešanas laiku un uzlabot automatizācijas procesa kvalitāti.
Mērķa sasniegšanai izpildītie uzdevumi

- Izpētīti esošie risinājumi un pieejas konfigurācijas pārvaldības procesu automatizācijai.
- Identificēti galvenie ieguvumi un trūkumi jaunākajos konfigurācijas pārvaldības automatizācijas risinājumos.
- Izstrādāta pieeja un metodoloģija konfigurācijas pārvaldības procesu automatizācijai.
- Izstrādāts prototips jaunās metodoloģijas ieviešanas automatizācijai.
- Izstrādāti kritēriji jaunas metodoloģijas novērtēšanai.
- Ieviesta konfigurācijas pārvaldības automatizācija programmatūras izstrādes projektos, un pēc izstrādātiem kritērijiem noteikti metodoloģijas ieguvumi un trūkumi.
- Izstrādāta metodoloģijas uzlabotā versija, kas novērš eksperimentu rezultātā identificētus trūkumus.
- Veikti atkārtoti eksperimenti, un iegūti praktiski apliecinājumi tam, ka metodoloģijas uzlabotā versijā ir novērsti sākotnēji identificēti trūkumi.

Promocijas darbā izstrādātā metodoloģija un visi jaunie modeļi tika eksperimentāli analizēti, lai varētu pārbaudīt izvirzītas hipotēzes. Eksperimentu rezultāti parādīja:

- pirmā hipotēze tika pierādīta, salīdzinot konfigurācijas pārvaldības automatizācijas ieviešanas laiku pēc vecām metodēm, kad netika atkārtoti lietoti esošie risinājumi, un automatizācijas ieviešanas laiku, lietojot atkārtoti jau esošus risinājumus. Eksperimentu rezultāti parādīja tendenci, ka, lietojot *EAF* metodoloģiju, var ieviest automatizāciju aptuveni divreiz ātrāk. Eksperimentos vidējais rādītājs bija 65 %;
- otrā hipotēze tika pierādīta, salīdzinot konfigurācijas pārvaldības automatizācijas ieviešanas laiku pēc jaunas metodoloģijas dažādiem projektiem. Eksperimenti parādīja, ka sākotnēji, kad esošu risinājumu datubāze ir tukša un visus risinājumus bija jāveido no nulles, ieguvums ir tikai 9 procenti. Savukārt — jo ilgāk risinājumi eksistē datubāzē un attīstās, jo stabilākie tie kļūst, un automatizācijas ieviešanas laiks samazinās. Pēdējā projektā, kurā tika ieviesta automatizācija, ieviešanas laiks samazinājās par 96 procentiem salīdzinājumā ar ieviešanu bez *EAF* modeļvadāmas pieejas.

Veicot literatūras analīzi, strādājot ar dažādiem rīkiem konfigurācijas pārvaldības uzdevumu risināšanai un izstrādājot jaunu metodoloģiju, tika secināts:

- mūsdienās bieži konfigurācijas pārvaldības procesus definē nepilnīgi, akcentējot vien dažus uzdevumus, ko min nozares speciālisti, kvalitātes standarti un zinātniski pētījumi;
- vēl viena tendence, ko ir ievērojis promocijas darba autors ir tā, ka konfigurācijas pārvaldību mēdz uzskatīt vienkārši par rīku kopu. Dažreiz nozares speciālistiem rodas ilūzija, ka, uzinstalējot rīkus, procesus var uzskatīt par ieviestiem un vairs par tiem nav jāuztraucas. Šāda nostāja radīja zaudējumus ne vienam vien projektam gan Latvijā, gan arī visā pasaulē. Nav svarīgi, kādus rīkus projekts lieto, bet ir svarīgi, cik efektīvas ir rīku ieviešanas metodoloģijas, kas spētu efektīvi izvēlēties, nokonfigurēt rīkus, kā arī sniegt rekomendācijas, veicot konfigurācijas pārvaldības procesa darbības.

Darba rezultāti tika izmantoti zinātniskajos projektos un RTU mācību priekšmeta «Lietišķo datorsistēmu programmatūra» studiju procesā:

- LZP projekts «Modeļu un metožu izstrāde lietišķai intelektuālai programmatūrai pamatojoties uz izkliedētu mākslīgu intelektu, zināšanu pārvaldību un progresīvām tīmekļa tehnoloģijām» izpildē (vad. prof. J. Grundspeņķis); programmatūras pārvaldības modeļvadāmo metožu izstrāde;
- Eiropas Komisijas 7. IP projektā *eINTERASIA «ICT Transfer Concept for Adaptation, Dissementation and Local Exploitation of European Research Results in Central Asia's Countries*», 2013 – 2015 (projekta koordinators — prof. Leonīds Novickis); Programmatūras ietvara pārvaldības modeļu izstrāde;
- priekšmeta «Lietišķo datorsistēmu programmatūra» mācību procesā. Tika sagatavota mācību līdzekļa daļa (Metodiski norādījumi priekšmetā «Lietišķo datorsistēmu programmatūra»/L. Novickis, V. Kotovs, A. Lesovskis, A. Bartusevičs», RTU, 2012. – 67 lpp,; nodaļa: Lietišķās programmatūras konfigurācijas pārvaldība);
- valsts pētījumu projektā VVP Y8089 «Kiberfizikālās sistēmas, ontoloģijas un biofotonita drošai un vieglai pilsētai un sabiedrībai (no 2014. g.) — programmatūras konfigurācijas pārvaldība».

Promocijas darba turpmākie attīstības virzieni

- Vižu sākotnējas instalācijas procesu formalizācija. Šobrīd *EAF* metodoloģija paredz, ka visas vides jau ir izveidotas. Taču reāli programmatūras izstrādes projekta pašā sākumā vides veido no nulles: instalē operētājsistēmas, aplikāciju serverus, datubāzes, konfigurē ugunsmūrus utt.
- Jānovērtē *EAF* metodoloģijas atbilstība populārākiem kvalitātes standartiem un vadlīnijām, piemēram *CMMI, ISO, ITIL* u. c.
- Jāizstrādā vēl viens modelis, kas ļautu no *PIEM, PSAM* un *CM* modeļiem automātiski ģenerēt konfigurācijas pārvaldības plānu.

BIBLIOGRĀFISKAIS SARAKSTS

[ABO 2014] About CMMI Institute. 2014. [ONLINE] Available at: http://whatis.cmmiinstitute.com/about-cmmi-institute. [Apskatīts 02 Septembrī 2014].

[AIE 2010] Aiello, R. Configuration Management Best Practices: Practical Methods that Work in the Real World (1st ed.). Addison-Wesley, 2010.

[ALT 2008] Altmanninger K. Models in conflict — towards a semantically enhanced version control system for models. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 2008;5002 LNCS:293-304.

[ALT 2010] Bruce Altner, Brett Lewinski. A Roadmap to Continuous Integration. Proceedings of the 2010 IT Summit, NASA, 2010.

[ASN 2010] Asnina, E. & Osis, J. 2010, «Computation independent models: Bridging problem and solution domains», Proceedings of the 2nd International Workshop on Model-Driven Architecture and Modelling Theory-Driven Development, MDA and MTDD 2010, in Conjunction with ENASE 2010, pp. 23.

[AZO 2008] Azoff M., The Benefits of Model Driven Development. MDD in Modern Web-based Systems, Published March, Butler Direct Limited, 2008.

[AZO 2014] Azoff, R., DevOps: Advances in Release Management and Automation. [ONLINE] Available at: http://electric-cloud.com/wp-

content/uploads/2014/06/EC-IAR_Ovum-DevOps.pdf [Apskatīts 20 Oktobrī 2014].

[BAM 1995] Bamford, R., 1995. *Configuration Management and ISO 9001*. Software Systems Quality Consulting, DO-25 V6, 7., 1995.

[BAR 2012a] Bartusevics A., Kotovs V., Novickis L. A Method for Effective Reuse-Oriented Software Release Configuration and Its Application in Insurance Area. Proceedings of Riga Technical University «Information Tehnology and Management Science», 15th series, RTU Publishing, 2012, Riga, Latvia, pp. 111–115.

[BAR 2012b] Bartusevics A., Kotovs V. Towards the effective reuse-oriented release configuration process. Proceedings of the 5-th International Scientific Conference «Applied Information and Communication Tehnologies», 2012, Jelgava, Latvia, pp. 99–103.

[BAR 2013] Bartusevics A., AMethodology for Model-Driven Software Configuration Management Implementation and Support. Proceedings of the 6-th International Scientific Conference «Applied Information and Communication Tehnologies», 2013, Jelgava, Latvia, pp. 252–258.

[BAR 2014a] Bartusevičs, A., Novickis, L. Model-Driven Software Configuration management and Environment Model. No: Recent Advances in Electrical and Electronic Engineering. Proceedings of the 3rd International Conference on Systems, Communications, Computers and Applications (CSCCA"14), Itālija, Florence, 22.-24. novembris, 2014. Italy: WSEAS Press, 2014, 132.–140. lpp. ISBN 978-960-474-399-5. ISSN 1790-5117.

[BAR 2014b] Bartusevičs, A., Novickis, L., Leye, S. Implementation of Software Configuration Management Process by Models: Practical Experiments and Learned Lessons. Applied Computer Systems. Nr.16, 2014, 26.–32. lpp. ISSN 2255-8683. e-ISSN 2255-8691. Pieejams: doi:10.1515/acss-2014-0010

[BAR 2014c] Bartusevičs, A., Novickis, L. Models for Implementation of Software Configuration Management. No: Procedia Computer Science. Valmiera, Latvia: 2014, 3.–10. lpp.

[BAR 2014d] Bartusevičs, A., Lesovskis, A., Novickis, L. Model-Driven Software Configuration Management and Semantic Web in Applied Software Development. Proceedings of the 13th International Conference on Telecommunications and Informatics (TELE-INFO '14), IIstanbul, Turkey December 15–17, 2014.

[BAR 2014e] Bartusevičs, A., Novickis, L. Towards the Model-driven Software Configuration Management Process. Information Technology and Management Science. Nr.17, 2014, 32.-38. lpp. ISSN 2255-9086. e-ISSN 2255-9094.

[BAR 2014f] Bartusevics Arturs, Leonids Novickis and Eberhard Bluemel. 2014. Intellectual Model-Based Configuration Management Conception. Applied Computer Systems. 15(1): 5-41. Retrieved 28 Nov. 2014, from doi:10.2478/acss-2014-0003

[BAR 2015] Bartusevičs, A., Novickis, L. Model-based Approach for Implementation of Software Configuration Management Process. No: MODELSWARD 2015: Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development, Francija, Angers, 9.–11. februāris, 2015. Lisbon: SciTePress, 2015, 177.-184. lpp. ISBN 978-989-758-083-3.

[BAR 2015a] Bartusevičs, A., Lesovskis, A., Novickis, L. Semantic Web Technologies and Model-Driven Approach for the Development and Configuration Management of Intelligent Web-Based Systems. No: Proceedings of the 2015 International Conference on Circuits, Systems, Signal Processing, Communications and Computers, Austrija, Vienna, 15.–17. marts, 2015. Vienna: 2015, 32.–39. lpp. ISBN 978-1-61804-285-9. ISSN 1790-5117.

[BEL 2005] Bellagio, M. What Is Software Configuration Management? Internet http://ptgmedia.pearsoncmg.com/images/0321200195/samplechapter/bellagio_ch01.pdf, 2005.

[BER 2003] Berczuk, Appleton. Software Configuration Management Patterns: Effective TeamWork, Practical Integration (1st ed.). Addison-Wesley, 2003.

[BER 2011] Berziša, S. & Grabis, J. 2011, «Combining project requirements and knowledge in configuration of project management information systems», ACM International Conference Proceeding Series, pp. 89.

[BER 2012] Bērziša, Solvita. Application of Knowledge and Best Practices in Configuration of Project Management Information Systems : promocijas darbs / S.Bērziša ; zinātniskais vadītājs J.Grabis ; Rīgas Tehniskā universitāte. DATORZINĀTNES UN INFORMĀCIJAS TEHNOLOĢIJAS FAKULTĀTE. Informācijas tehnoloģijas institūts. Vadības informācijas tehnoloģijas katedra. Rīga : [RTU], 2012. 196 pp.

[BIL 2014] Bill Chamberlin's HorizonWatching. 2014. Top 18 Trends in Application Software Development for 2014. [ONLINE] Available at: http://www.billchamberlin.com/top-18-trends-in-application-software-development-for-2014/. [Apskatīts 20 Oktobrī 2014].

[BRA 2008] Bastian Braun. SAVE — Static Analysis on Versioning Entities. ICSE: International Conference on Software Engineering, 2008.

[BRO 2002] Brouse, Peggy S. Configuration Management Interenet: http://www.eolss.net/Sample-Chapters/C15/E1-28-03-02.pdf , 2002.

[BRO 2005] A. B. Brown, A. Keller, and J. L. Hellerstein, A model of configuration complexity and its application to a change management system, IFIP/IEEE International Symposium, Integrated Network Management, pp. 631–644, 2005.

[BRU 2004] Brugge, B., Dutoit, A. Software Configuration Management. Internet https://files.ifi.uzh.ch/rerg/amadeus/teaching/courses/software_engineering_hs08/folien/Kapitel_23_Addendum_SCM.pdf, 2004.

[BUC 2009] Buchmann T., Dotor A., Westfechtel B. MODEL-DRIVEN DEVELOPMENT OF SOFTWARE CONFIGURATION MANAGEMENT SYSTEMS. ICSOFT 2009 – 4th International Conference on Software and Data Technologies 2009.

[BUS 2011] Bushehrian O., Automatic object deployment for software performance enhancement. The Institution of Engineering and Technology 2011, Vol. 5, Iss. 4, pp. 375–384, 2011.

[CAL 2012] Calhau R., Falbo R. A Configuration Management Task Ontology for Semantic Integration. Proceedings of the 27th Annual ACM Symposium on Applied Computing Pages 348–353 ACM New York, NY, USA, 2012.

[CLE 2012] Clemencic M., Mato P., A CMake-based build and configuration framework. Journal of Physics: Conference Series 396 (2012) 052021, 2012.

[CMC 2014] CMCrossroads | Three Major Trends in Software Release Management You Should Adopt . 2014. [ONLINE] Available at:

http://www.cmcrossroads.com/article/three-major-trends-software-release-management-you-should-adopt. [Apskatīts 20 Oktobrī 2014].

[COM 2011] Comas J., Mostashari A., Mansouri M., Turner R. A Software Deployment Risk Assessment Heuristic for Use in a Rapidly-Changing Business-to-Consumer Web Environment International Journal of Software Engineering and Its Applications Vol. 5 No. 4, October, 2011.

[CON 2002] Configuration Management Training. Section 1: Explaining Configuration Management, EESA, 2002. Internet: http://esamultimedia.esa.int/docs/industry/SME/Configuration/Section_1-CM.pdf

[CON 2015] The Convergence of DevOps « IT Revolution IT Revolution. [ONLINE] Available at: http://itrevolution.com/the-convergence-of-devops/. [Apskatīts 28 Janvārī 2015].

[CRA 2008] Cravino P., Enterprise Software Configuration Management Solutions for Distributed and System z. 1st ed. USA: Redbooks. 2008.

[DAR 2001] Dart, S. Concepts in Configuration Management Systems. Internet http://sceweb.uhcl.edu/boetticher/swen5230/concepts-in-configuration-management.pdf, 2001.

[DEP 2010] Department of Defense, USA Military Handbook. Configuration management guidance (rev. A) (MIL-HDBK-61A). Retrieved January 5, 2010, from http://www.everyspec.com/MIL-HDBK/MIL-HDBK-0001-0099/MIL-HDBK-61_11531/

[DEV 2014] DevOps Implementation | Giga Promoters. 2014. [ONLINE] Available at: http://gigapromoters.com/offerings/services/it-services/devops-implementation/. [Apskatīts 10 Novembrī 2014].

[DOD 2014] Do DevOps tools really exist?. 2014. Do DevOps tools really exist?. [ONLINE] Available at: http://www.scriptrock.com/blog/devops-tools-exist. [Apskatīts 11 Novembrī 2014].

[DON 2011] Doniņš Uldis. Topoloģiskā biznesa sistēmu modelēšana un programmatūras sistēmu projektēšana. Metodiskais līdzeklis. RTU Izdevniecība. Rīga 2011.

[EIL 2006] T. Eilam, M. H. Kalantar, A. V. Konstantinou, G. Pacifici, and J. Pershing, «Managing the Configuration Complexity of Distributed Applications in Internet Data Centers,» IEEE Communications Magazine, vol. 44, pp. 166–177, 2006.

[EST 2013] Estler, H.-Christian, Unifying Configuration Management with Merge Conflict Detection and Awareness Systems. In 22nd Australian Conference on Software Engineering. Austalia, 4–7 June 2013. Austalia: IEEE. 201-210.

[FIT 2014] Fitzgerald B., Stol J., Continuous software engineering and beyond: trends and challenges. Proceeding in RCoSE 2014 Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, Pages 1-9. ACM New York, NY, USA, 2014.

[FUG 2014] Fuggetta A., Nitto E., Software process. Proceeding in FOSE 2014 Proceedings of the on Future of Software Engineering, Pages 1–12, ACM New York, NY, USA, 2014.

[GAL 2009] Galup, S. D., Dattero, R., Quan, J.J., Conger, S., An Overview of IT Service Management. *Commun.* ACM, 2009, vol. 52, no. 5, pp. 124–127., 2009.

[GHE 2012] Giacomo Ghezzi, Michael Würsch, Emanuel Giger, Harald Gall. An Architectural Blueprint for a Pluggable Version Control System for Software (Evolution) Analysis, In: 2nd Workshop on Developing Tools as Plug-ins, Zurich, 03 June 2012–03 June 2012.

[GIE 2009] Giese Holger, Seibel Andreas, Vogel Thomas. A Model-Driven Configuration Management System for Advanced IT Service Management. Available at: http://www.hpi.unipotsdam.de/giese/gforge/publications/pdf/GSV-MRT09_paper_7.pdf, 2009.

[GLO 2012] IT Glossary. Defining The IT Industry. SCM Software Configuration Management. Internet http://www.gartner.com/it-glossary/scm-software-configuration-management/, 2012.

[GRO 2007] H. Gronniger, H. Krahn, B. Rumpe, M. Schindler, and S. Volkel. Textbased Modeling. In 4th International Workshop on Software Language Engineering, 2007.

[GUO 2005] Guozheng Ge, E., Whitehead, Jr. Automatic Generation of Rule-based Software Configuration Management Systems. ICSE'05, May 15–21, 2005, St. Louis, Missouri, USA.

[HAG 2010] Hagen, S., Kemper, A., Model-Based Planning for State-Related Changes to Infrastructure and Software as a Service Instances in Large Data Centers. Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, On page(s): 11–18, Volume: Issue: , 5–10 July 2010

[HAT 2012] Hideaki Hata, Osamu Mizuno, Tohru Kikuno. Bug Prediction Based on Fine-Grained Module Histories. ICSE: International Conference on Software Engineering, Feb2012, p200–210.

[HIS 2014] History of software configuration management - Wikipedia, the free encyclopedia. 2014. [ONLINE] Available at: http://en.wikipedia.org/wiki/History_of_software_configuration_management. [Apskatīts 05 Novembrī 2014].

[HIST 2014] A History of Version Control. [ONLINE] Available at: http://ericsink.com/vcbe/html/history_of_version_control.html. [Apskatīts 11 Novembrī 2014].

[HUA 2009] Shi-Ming Huang, Chih-Fong Tsai, Po-Chun Huang. Component-based software version management based on a Component-Interface Dependency Matrix, The Journal of Systems and Software, 2009.

[JIA 2009] Jiang, L., Eberlein, A., An Analysis of the History of Classical Software Development and Agile Development. Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics San Antonio, TX, USA – October 2009.

[JOH 2011] Johnsen E., Schlatte R. Integrating Aspects of Software Deployment in High-Level Executable Models, presented at the NIK-2011 conference, 2011.

[JUI 2002] Juite Wanga, Yung-I Lin, A fuzzy multicriteria group decision making approach to select configuration items for software development. MathematicsWEB, Fuzzy Sets and Systems, 2002.

[KAN 2005] Ronald Kirk Kandt. Configuration Management Principles and Practices. Jet Propulsion Laboratory, 4800 Oak Grove Dr., Pasadena, CA 91 109, USA Internet: http://trs-new.jpl.nasa.gov/dspace/bitstream/2014/10507/1/02-2525.pdf , 2005.

[KAP 2008] Kapitza R, Baumann P, Reiser HP. Using object replication for building a dependable version control system. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 2008;5053 LNCS:86-99.

[KAR 2009] G. Karsai, H. Krahn, C. Pinkernell. Design Guidelines for Domain Specific Languages. Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling DSM'09, page 7–13., 2009.

[KEL 2008] S. Kelly and J.-P. Tolvanen. Domain-Specific Modeling: Enabling Full Code Generation. Wiley, 2008.

[KR 2014] Krusche S., Alperowitz L., Introduction of continuous delivery in multi-customer project courses. Proceeding in ICSE Companion 2014 Companion Proceedings of the 36th International Conference on Software Engineering, Pages 335-343. ACM New York, NY, USA, 2014.

[LAV 2011] Jannik Laval, Simon Denier, Stéphane Ducasse, Jean-Rémy Falleri. Supporting simultaneous versions for software evolution assessment. Science of Computer Programming, 2011.

[LES 2014] Lesson 11: Devops & Configuration Management Intro — OSU DevOps Bootcamp 0.0.1 documentation. 2014. [ONLINE] Available at: http://devopsbootcamp.readthedocs.org/en/latest/11_devops.html. [Apskatīts 10 Novembrī 2014].

[LI 2012] Jingyue Li, Michael D. Ernst. CBCD: Cloned Buggy Code Detector. ICSE: International Conference on Software Engineering, Feb2012, p310–320.

[MAL 2012] Malek S. An Extensible Framework for Improving a Distributed Software System's Deployment Architecture. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 38, NO. 1, JANUARY/FEBRUARY 2012.

[MEL 2006] Mellon, K. A Framework for Software Product Line Practice, Version 5.0. Internet: http://www.sei.cmu.edu/productlines/frame_report/config.man.htm, 2006.

[MER 2005] M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. Technical Report SEN-E0309, Centrum voor Wiskunde en Informatica, Amsterdam, 2005.

[MET 2002] Anne Mette Jonassen Hass. Configuration Management Principles and Practice, Addison-Wesley Professional. Part of the Agile Software Development Series series, 2002, pages 432.

[MUR 2008] Leonardo Murta, Chessman Correa, Joao Gustavo Prudencio. Towards Odyssey-VCS 2: Improvements over a UML-based Version Control System. ICSE: International Conference on Software Engineering, 2008.

[NIK 2008] Nikulsins, V. & Nikiforova, O. 2008, «Adapting software development process towards the model driven architecture», Proceedings – The 3rd International Conference on Software Engineering Advances, ICSEA 2008, Includes ENTISY 2008: International Workshop on Enterprise Information Systems, pp. 394.

[NIK 2009] Nikiforova, O., Cernickins, A. & Pavlova, N. 2009, «Discussing the difference between model driven architecture and model driven development in the context of supporting tools the projection of two-hemisphere model into the component model of MDA/MDD», 4th International Conference on Software Engineering Advances, ICSEA 2009, Includes SEDES 2009: Simposio para Estudantes de Doutoramento em Engenharia de Software, pp. 446.

[OPE 2014] OpenMake Products. [ONLINE] Available at: http://www.openmakesoftware.com/build-management. [Apskatīts 22 Novembrī 2014].

[OSE 2002] Oject-Oriented Software Engineering Using UML, Patters and JAVA «Software Configuration Management» Internet: http://www.bilkent.edu.tr/~bakporay/cs_413/Bruegge_L28_Configuration Management_ch12lect1.ppt , 2002

[OSI 2008] Osis, J., Asnina, E. & Grave, A. 2008, Formal problem domain modeling within MDA. Proceedings of the 2nd International Conference on Software and Data Technologies, ICSOFT 2007; Barcelona; Spain; Volume 22 CCIS, 2008, Pages 387–398.

[OSI 2010] Osis, J. & Donins, U. 2010, «Platform independent model development by means of topological class diagrams», Proceedings of the 2nd International Workshop on Model-Driven Architecture and Modelling Theory-Driven Development, MDA and MTDD 2010, in Conjunction with ENASE 2010, pp. 13.

[OSI 2011] Osis J., Asnina E. Model-Driven Domain Analysis and Software Development: Architectures and Functions. IGI Global, Hershey – New York, 2011, 514 p.

[PAI 1999] R. Paige, J. Ostroff, and P. Brooke. Principles for Modeling Language Design. Technical Report CS-1999-08, York University, December 1999.

[PAU 2007] Paul M. Duvall, Steve Matyas, and Andrew Glover. Continuous Integration: Improving Software Quality and Reducing Risk. (1st ed.). Addison-Wesley Professional, 2007.

[PFA 1997] P. Pfahler and U. Kastens. Language Design and Implementation by Selection. In Proc. 1st ACM-SIGPLAN Workshop on Domain-Specific-Languages, DSL ’97, pages 97–108, Paris, France, January 1997. Technical Report, University of Illinois at Urbana-Champaign.

[PIN 2009] Pindhofer Walter, Model Driven Configuration Management. Master work of Wien University, Wien, 2009.

[RAG 2014] Ragan, T., 21st-Century DevOps--an End to the 20th-Century Practice of Writing Static Build and Deploy Scripts, Linux Journal, 230, pp. 116–120, Computers & Applied Sciences Complete, EBSCOhost, viewed 22 October 2014.

[RAZ 2007] Saad Razzaq, Fahad Maqbool, Bilal Anjum. The Challenges & Case for Mining Software Repositories. International MultiConference of Engineers and Computer Scientists, 2007.

[ROS 2010] Alessandro Rossini, Adrian Rutle, Yngve Lamo, UweWolter. A formalisation of the copy-modify-merge approach to version control in MDE. The Journal of Logic and Algebraic Programming, 2010.

[RUA 2003] Ruan Li, Zhong Yong, A New Configuration Management Model for Software Based on Distributed Components and Layered Architecture. Parallel and Distributed Computing, Applications and Technologies, 2003. PDCAT'2003. 27–29 Aug. 2003.

[SAR 2008] Anita Sarma, David Redmiles, André van der Hoek. Empirical evidence of the benefits of workspace awareness in software configuration management. Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, 2008

[SAT 2011] Laika Satish, Identifying the Dissimilarities based on Working of Programs among Versions in DVCS (Distributed Version Control Systems). International Journal of Computer Applications (0975 – 8887) Volume 36– No. 6, December 2011.

[SCH 2010] Holger Schackmann, Horst Lichter. Process assessment by evaluating configuration and change request management systems. Proceedings of the Warm Up Workshop for ACM/IEEE ICSE 2010

[SCM 2001] Software Configuration Management Internet: http://dogbert.mse.cs.cmu.edu/charlatans/References/Configuration%20Management/0130912972.pdf 2001.

[SEK 2012] Atsuji Sekiguchi, Kuniaki Shimada, Yuji Wada, Akio Ooba, Ryouji Yoshimi, Akiko Matsumoto. Configuration management technology using tree structures of ICT systems. Proceedings of the 15th Communications and Networking Simulation Symposium Publisher: Society for Computer Simulation International, 2012

[SER 2014] Serena Deployment Automation Overview. Serena Deployment Automation Overview. [ONLINE] Available at: http://www.serena.com/index.php/en/products/featured-products/serena-deployment-automation/overview/. [Apskatīts 22 Novembrī 2014].

[SHI 2010] Shih C., Huang S. Exploring the relationship between organizational culture and software process improvement deployment, Information & Management 47 (2010) 271–281p., 2010.

[SIN 2008] Sindhuja P. N., Surajit Ghosh Dastidar. Software Deployment: Concepts and Technologies. ICFAI Journal of Systems Management, 2008.

[SIN 2010] Sinan Si Alhir. Understanding the Model Driven Architecture (MDA). Available at: http://www.methodsandtools.com/archive/archive.php?id=5, 2010.

[SIY 2008] Harvey Siy, Parvathi Chundi, Daniel J. Rosenkrantz, Mahadevan Subramaniam. A segmentation-based approach for temporal analysis of software version repositories. Jornal of Software Maintenance and Evolution: Research and Practice, 2008.

[SOF 2014] Software configuration management - Wikipedia, the free encyclopedia. 2014. [ONLINE] Available at: http://en.wikipedia.org/wiki/Software_configuration_management. [Apskatīts 05 Novembrī 2014].

[STA 2008] Glen Stansberry. 7 Version Control Systems Reviewed. At http://www.smashingmagazine.com/2008/09/18/the-top-7-open-source-version-control-systems/, 2008.

[TAK 2014] Taking Release Management to the Next Level. 2014. [ONLINE] Available at: http://www.slideshare.net/xebialabs/taking-releasemanagementtothenextlevel. [Apskatīts 20 Oktobrī 2014].

[TAR 2011] Alexander Tarvo, Thomas Zimmermann, Jacek Czerwonka. An integration resolution algorithm for mining multiple branches in version control systems. IEEE international conference on software maintenance, ICSM; 2011. 402 p.

[THA 2009] Tung Thanh Nguyen, Hoan Anh Nguyen, Nam H. Pham, Jafar M. Al-Kofahi, Tien N. Nguyen. Clone-Aware Configuration Management. ASE '09: Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering, 2009.

[TOL 2005] J. Tolvanen, S Kelly. Defining Domain-Specific Modeling Languages to Automate Product Derivation: Collected Experiences. Proceeding SPLC'05 Proceedings of the 9th international conference on Software Product Lines Pages 198-209. 2005.

[TRE 2014] Trends in Software Engineering - Dice News. 2014. [ONLINE] Available at: http://news.dice.com/software-engineering-talent-community/trends/. [Apskatīts 20 Oktobrī 2014].

[VAC 2006] VACCAPERNA Systems Limited. Software Configuration Management (SCM). Internet http://www.vaccaperna.co.uk/scm/about_scm.html, 2006.

[VAS 2013] Vasiljevics, I., Milosavljevics, G., Dejanovics, I., Filipovics, M., COMPARISON OF GRAFICAL DSL EDITORS. The 6th PSU-UNS International Conference on Engineering and Technology (ICET-2013), Novi Sad, Serbia, May 15–17, 2013.

[WET 2012] Wettinger J., Concepts for Integrating DevOps Methodologies with Model-Driven Cloud Management Based on TOSCA. Institute of Architecture of Application Systems University of Stuttgart, 2012.

[WHA 2014] What Are Current Hot Trends In The Field Of Software Engineering?. 2014. [ONLINE] Available at: http://bloggless.com/it/software-engineering/what-is-currently-popular-in-software-engineering/. [Apskatīts 20 Oktobrī 2014].

[WIL 2003] D. Wile. Lessons Learned from Real DSL Experiments. Proceedings of the 36th Hawaii International Conference on System Sciences, 2003.

[WIL 2004] D. Wile. Lessons learned from real DSL experiments. Science of Computer Programming, 51(3):265–290, June 2004.

[WES 2005] Westfechtel, B., Conradi, R. Software Architecture and Software Configuration Management Internet http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13.3085&rep=rep1&type=pdf, 2005.

[ЗАМ 2008] Заметки о Software Configuration Management. Управление конфигурацией программного обеспечения. Интернет: http://scm-notes.blogspot.com/p/scm-books.html, 2008.

[ЛАП 2004] Лапыгин, Д. Новичков, А. Конфигурационное управление проектами разработки программного обеспечения. Интернет: http://citforum.ru/SE/quality/configuration_management/, 2004.

[ОРЛ 2011] Орлик, С. Программная инженерия. Конфигурационное управление Перевод главы из SWEBOK с комментариями. Архивировано из первоисточника 14 марта 2012. Проверено 18 июня 2011.

[УДО 2011] Удовиченко, Ю. Управление изменениями и кессонная болезнь проектов. Интернет: http://experience.openquality.ru/software-configuration-management/, 2011.