RĪGAS TEHNISKĀ UNIVERSITĀTE
Datorzinātnes un informācijas tehnoloģijas fakultāte
Lietišķo datorsistēmu institūts
Lietišķo datorzinātņu katedra

Vladimirs ŅIKUĻŠINS
Doktora studiju programmas „Datorsistēmas" doktorants

PROGRAMMATŪRAS IZSTRĀDES DZĪVES CIKLA MODEĻA TRANSFORMĀCIJAS PIEEJA MODEĻVADĀMĀS ARHITEKTŪRAS KONTEKSTĀ

Promocijas darba kopsavilkums

Zinātniskā vadītāja
Dr. sc. ing., profesore
O.ŅIKIFOROVA

Rīga 2011

UDK 004.2(043.2)
Ņi 805 p

Ņikuļšins V. Programmatūras izstrādes dzīves cikla modeļa transformācijas pieeja modeļvadāmās arhitektūras kontekstā .
Promocijas darba kopsavilkums.-R.:RTU, 2011.-25 lpp.

Iespiests saskaņā ar DITF LD institūta 2011.gada 23.maija lēmumu, protokols Nr.71.

Šis darbs izstrādāts ar Eiropas Sociālā fonda atbalstu Nacionālās programmas „Atbalsts doktorantūras programmu īstenošanai un pēcdoktorantūras pētījumiem" projekta „Atbalsts RTU doktorantūras attīstībai" ietvaros.

ISBN

PROMOCIJAS DARBS
IZVIRZĪTS INŽENIERZINĀTŅUDOKTORA GRĀDA IEGŪŠANAI RĪGAS TEHNISKAJĀ UNIVERSITĀTĒ

Promocijas darbs inženierzinātņu doktora grāda iegūšanai tiek publiski aizstāvēts 2011.g. 7. novembrī Rīgas Tehniskās universitātes Datorzinātnes un informācijas tehnoloģijas fakultātē, Meža ielā 1/3, 202 auditorijā.

OFICIĀLIE RECENZENTI

Profesors, Dr.habil.sc.ing. Jānis Grundspeņķis
Rīgas Tehniskā universitāte

Profesors, Dr.habil.sc.comp. Audris Kalniņš
Latvijas Universitāte

Profesors, Dr.sc.comp. Viktors Taratuhins
Nacionāla pētnieciskā universitāte „Ekonomikas augstskola" (Krievija)
SAP NVS, Skandināvijas un Baltijas universitāšu alianses programmas direktors

APSTIPRINĀJUMS

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

Vladimirs Ņikuļšins …….(Paraksts)

Datums: 13.6.2011

Promocijas darbs ir uzrakstīts latviešu valodā, satur ievadu, 5 nodaļas, nobeigumu, literatūras sarakstu, 3 pielikumus, 47 attēlus, 7 tabulas, kopā 147 lappuses. Literatūras sarakstā ir 151 nosaukums.

VISPĀRĪGS DARBA RAKSTUROJUMS

Pētījuma pamatojums

Pasaulei mums apkārt piemīt dinamisks raksturs. Pēc Alfrēda Norta Vaitheda filozofiskajiem uzskatiem, mūsu pasauli var raksturot ar savstarpēji saistītu lielu un mazu notikumu sistēmām, kuras vienmēr ir mainīgas. No procesu filozofijas viedokļa pasaule nevis vienkārši eksistē, bet nepārtraukti evolucionē [PAL 2006].

Programmatūras izstrādei arī piemīt dinamisks raksturs un uz procesiem orientēta pieeja, ko sauc par programmatūras izstrādes procesu. Tā apskata ar programmatūras izstrādi saistītus aspektus, ieskaitot pārvaldības disciplīnu un procesu uzlabošanu. Programmatūras izstrādes process ir viena no programminženierijas disciplīnām. Programminženierija ir relatīvi jauna joma, un zinātnieku vidū joprojām pastāv noteiktas domstarpības par tās atbilstību klasiskajai inženierijai [Vliet 2006].

Programminženierijas principu praktisku pielietošanu apraksta programmatūras izstrādes metodoloģijas, kas definē koordinētu aktieru vadīto aktivitāšu kopu, ar mērķi izveidot programmatūras produktus [LAN 2004]. Modernām programmatūras izstrādes metodoloģijām piemīt strukturētas, disciplinētas un iteratīvas īpašības [KRU 2000]. Metodoloģijas ļauj strukturizēt un formalizēt programmatūras izstrādi tā, lai to varētu kvantitatīvi novērtēt plānojot vēlamo programmatūras funkcionalitāti, izmantojamos resursus un laiku [MSF 2003]. Modernās programmatūras izstrādes metodoloģijas iedala smagsvara metodoloģijās un spējās metodoloģijās. Smagsvara metodoloģijas uzsvaru liek uz plānošanu, detalizētu dokumentāciju un projektēšanu. Savukārt, spējās izstrādes metodoloģijas, ir orientētas uz rezultāta sasniegšanu, galveno uzmanību pievēršot sadarbībai komandā (nevis sekošanai procesa vadlīnijām) un strādājošam primkodam (nevis dokumentācijai) [KHA 2004].

Neskatoties uz mēģinājumiem sakārtot un formalizēt programmatūras izstrādes procesu, modernās smagsvara un spējās izstrādes metodoloģijas atbalsta tradicionālo uzskatu par programmatūras izstrādi, kur programmatūras pirmkoda rakstīšana ir galvenais programmatūras izstrādes process. Galvenās problēmas tradicionālajā programmatūras izstrādē ir šādas [KLE 2003]:

- Produktivitātes problēma (programmētāji bieži uzskata pirmkodu par produktīvu, bet dokumentāciju nē);
- Pārnesamības problēma (biežas izmaiņas tehnoloģijās);
- Sadarbspējas problēma (starpsistēmu komunikāciju sarežģītība);
- Uzturēšanas un dokumentēšanas problēma (dokumentācijai nav nekādas iedarbības uz kodu un otrādi).

Viena no pieejām, kas ļauj risināt šīs problēmas, ir 2001. gadā objektu vadības grupas (angl. *Object Management Group* – OMG) proklamētā modeļvadāmā arhitektūra (angl. *Model Driven Architecture* – MDA) [SIE 2001]. Tā, atšķirībā no tradicionālas programmatūras izstrādes, par galveno rezultātu uzskata sistēmas modeļa izveidošanu, nevis programmatūras pirmkodu [KLE 2003]. Automatizētās transformācijas dod iespēju sistēmas modeli pārveidot strādājošā kodā. Sistēmas modelis ir neatkarīgs no realizācijas, un ļauj izvērst modeli dažādām platformām. Ar MDA parādīšanos var uzskatīt, ka programmatūras izstrādes process ir iegājis jaunā evolūcijas stadijā un tas iesāk jaunu programmatūras izstrādes ēru. Var uzskatīt, ka pašlaik programmatūras izstrādes industrija atrodas pārejas periodā, kad ar „vecām" metodoloģijām vairs nav iespējams pārvarēt programmatūras sistēmu sarežģītību, bet „jaunās" metodoloģijas vēl nav nobriedušas un stabilizējušās to efektīvai lietošanai. Tādejādi aktuāla kļūst pārejas algoritmu izstrāde, kas dotu iespēju ierastās programmatūras izstrādes metodoloģijas un uzkrāto pieredzi bagātināt ar kādas jaunās metodoloģijas artefaktiem un principiem. Citiem vārdiem sakot, ir nepieciešama transformācijas pieeja pārejai no „tradicionālās" programmatūras izstrādes uz modeļvadāmās programmatūras izstrādi.

Aktualitāte

Attīstoties programmatūras izstrādes procesa organizācijai, ir nostabilizējušās arī vispārpieņemtas programmatūras izstrādes prakses [PEI 2010]. Parasti jaunas tehnoloģijas ieviešana ierastajā procesā ir visai komplicēts un darbietilpīgs process. To apgrūtina ne tikai formālie un objektīvie faktori, bet arī cilvēka konservatīvisms un iekšējā pretestība izmaiņām [SER 2005]. Metodoloģijas, kas atbalsta tradicionālo, uz kodu orientētu izstrādi, ir jāpielāgo modeļvadāmai izstrādei, mainot ierasto programmatūras izstrādes procesu, un jāizstrādā elastīga pieeja pārējai no tradicionālās programmatūras izstrādes uz MDA orientēto. Jauna, MDA bāzēta procesa ieviešana atbilst programmatūras izstrādes uzlabošanas mērķiem, kas ietver sevī programmatūras produkta kvalitātes un produktivitātes uzlabošanu, kā arī samazina izstrādes laiku [STE 1999].

Izmaiņas izstrādes procesā atspoguļo arī izmaiņas arhitektūras līmenī, ko izraisa MDA ieviešana. No finanšu viedokļa šķiet izdevīgi izmantot biznesa modeļus (piem., UML (angl. *Unified Modeling Language*), kas ietilpst MDA) [STA 2006], kas nemainās, vai arī attīstās neatkarīgi no tehnoloģiskām platformas izmaiņām [ERI 2004]. Programminženierijā ar modeļiem var aprakstīt gan sistēmas funkcionēšanu, gan pašu izstrādes procesu. Modeļi formāli nosaka uzvedību jeb aktivitātes (esošo vai plānoto sistēmas stāvokli), kaut gan paša procesa izmaiņas nav izteiktas ar modeļiem. Kā piemēru var minēt CMMI vai ISO 9000, kuri nosaka vēlamo sistēmas stāvokli [STE 1999], [CMM]. Tieši metodoloģijas nosaka, kā jānotiek šīm izmaiņām [SWE 2004].

Kā jebkura „revolucionāra" pieeja, arī modeļvadāmas izstrādes attīstībai piemīt evolucionārs raksturs, un MDA atrodas savas attīstības sākumā [GUT 2007]. 10 gadi ir pagājuši kopš MDA ir pasludināta par alternatīvu pieeju uz kodu orientētai programmatūras izstrādei, taču MDA rīki un metodes evolucionē, un joprojām ir aktuāla nepieciešamība pēc jauniem rīkiem, standartiem un labajām praksēm [FAV 2010]. Tas attiecas gan uz reālo MDA bāzētu projektu analīzi, gan uz teorētiskiem materiāliem [HUS 2011].

Pētījuma problēma

MDA nosaka standartus un principus modeļu transformēšanai, bet tajā pašā laikā nenosaka programmatūras izstrādes metodoloģiju un ar to saistītās aktivitātes, kas ir jāveic, izmantojot modeļvadāmo izstrādi. MDA tehnoloģijas nav tieši saistītas ar tradicionāliem programmatūras izstrādes procesiem, jo šīs tehnoloģijas ir paredzētas adaptēšanai un izmantošanai eksistējošos kompānijas procesos [GAV 2004]. OMG konsorcijs neapraksta modeļvadāmo izstrādi no procesu, aktivitāšu, lomu un artefaktu viedokļa, kas tiek izmantoti formāli aprakstot programmatūras izstrādes procesu. Tāpēc par pētījuma objektu promocijas darba ietvaros ir izvēlēts programmatūras izstrādes organizācijas process un pieeja, kā ir iespējams pāriet no vienas procesu organizācijas citā, saglabājot izstrādājamos artefaktus un to saturu.

Ņemot vērā, ka Latvijas programmatūras izstrādes firmas ir pamatā uz kodēšanas procesu orientētas organizācijas [NIK 2006a], [NIK 2006b], uzdevums definēt pārejas metodoloģiju uz modeļvadāmo programmatūras izstrādi kļūst īpaši aktuāls programminženierijas kontekstā Latvijā. Būtu vērtīgi noteikt ietvaru un stingri definēt metodoloģiju, kas atvieglotu pārēju no kodā sakņotas programmatūras izstrādes uz modeļiem orientētu izstrādi, pielietojot pētījuma izstrādes aprobācijai Latvijas uzņēmumus.

Esošo risinājumu apskats

Pašlaik neeksistē vienotas un pilnīgas metodoloģijas programmatūras izstrādes pārejai no tradicionālā izstrādes procesa uz modeļvadāmo. Eksistē vairāki specifiski, vai arī pārāk vispārīgi pētījumi, kas atbalsta tikai daļu no modeļvadāma izstrādes procesa. Ir pieejami arī pētījumi par specifisku programmatūras izstrādes metodoloģiju pielāgošanu MDA bāzētai izstrādei, piemēram, MASTER projekts (angl. *Model-driven Architecture inSTrumentation, Enhancement and Refinement*) vai MODA-TEL projekts [ESI 2003], [BEL 2002], [STE 2003]. Tomēr, to pielietošana nav universāla – nav iespējams noteikt, kā konkrētas programmatūras izstrādes kompānijas veiktās izstrādes aktivitātes sasaistīt ar modeļvadāmas izstrādes aktivitātēm.

MDA pamatā ir objektorientēta pieeja, un uz komponentēm bāzēta izstrāde, tāpēc teorētiski tā var būt apvienota ar vairākiem programmatūras izstrādes procesiem [CHI 2007]. Pēc OMG konsorcija vīzijas, MDA var tikt pielāgota jebkuram programmatūras izstrādes procesam, tomēr pāreja no viena izstrādes veida uz otru nav formalizēta. Kā iespējamo risinājumu, OMG konsorcijs piedāvā konsultatīvo palīdzību OMG FastStart programmas ietvaros. Šajā programmā iesaistīto ekspertu mērķis ir eksistējoša izstrādes procesa analīze konkrētā kompānijā, un rekomendāciju sniegšana pārejai uz modeļvadāmo izstrādi [GUT 2004].

Eksistē vairāki pētījumi, kuri definētajai problēmai piedāvā specifiskus teorētiskus un tehnoloģiskus risinājumus. Tie arī palīdzēja izveidot promocijas darbā piedāvāto risinājumu. [FEN 2006] piedāvā transformēt SPEM (angl. *Software and Systems Process Engineering Metamodel*) procesu modeļus citās procesu notācijas, tādās kā XPDL (angl. *XML Process Definition Language*), izstrādājot pieeju SPEM2XPDL. [COM 2006] piedāvā uzlabot SPEM ar OCL (angl. *Object Constraint Language*) [OMG 2006], jo SPEM semantika nav pietiekami bagāta, lai to pielietotu matemātiskās operācijās. Tas varētu apgrūtināt SPEM izmantošanu praksē, bet no citas puses, tas ļauj vairāk formalizēt SPEM procesu modeļus. [DEB 2007] apskata SPEM transformācijas uz BPMN (angl. *Business Process Modeling Notation*) ar darbplūsmu automatizēšanu. Daži autori apskata modeļu transformāciju specifiku, un šie risinājumi var būt noderīgi arī procesu modeļu transformācijām. [TRA 2004] apskata OMG konsorcija QVT (angl. *Query/View/Transformation*) valodu [MOF 2008]. Viņš pievēršas trasējamības aktivitātēm un modeļu standartatbilstībai, kas skar gan avota modeli (angl. *source model*), gan mērķa modeli (angl. *target model*). [TRA 2004] arī piedāvā jaunu pieeju modeļu transformācijās, apvienojot deklaratīvu un imperatīvu transformāciju valodu īpašības.

Procesu metamodeļu integrēšanu un unificēšanu apskata [BRE 2001]. Zinātniskais raksts par procesa centrisko modeļu inženieriju (angl. *Process-Centered Model Engineering*) tika uzrakstīts 2001. gadā, tomēr pašlaik tas ir kļuvis vēl vairāk aktuāls līdz ar dažādu atšķirīgu MOF (angl. *Meta-Object Facility*) realizāciju parādīšanos. [DIA 2010] vērš uzmanību uz modeļvadāma izstrādes procesa nepietiekamu atbalstu procesu modelēšanas rīkos. SPEM 2.0 metamodeļu līmenī neatbalsta MDE (angl. *Model-driven Engineering*). Kā risinājumu šai problēmai autori piedāvā izmantot ar MDE specifiku bagātinātu SPEM metamodeli un atbilstošu rīku.

Pētījuma mērķis

Promocijas **darba mērķis** ir izstrādāt pieeju, kas dotu iespēju, izmantojot formālas transformācijas principus, pārveidot programmatūras izstrādes organizāciju no viena programmatūras izstrādes procesa citā, un demonstrēt izstrādāto pieeju ar piemēru pārejai no tradicionālas programmatūras izstrādes uz modeļvadāmo.

Šī mērķa sasniegšanai nepieciešams izpētīt, kādi MDA artefakti un procesi var aizvietot noteiktus tradicionālās programmatūras izstrādes artefaktus un procesus, un definēt metodoloģiju pārejai no tradicionālās programmatūras izstrādes MDA orientētajā.

Pētījuma uzdevumi

1. Aprakstīt tradicionālo programmatūras izstrādes procesu un standartus.
2. Izpētīt modeļvadāmas izstrādes būtību un pamatkoncepcijas.
3. Izpētīt integrācijas iespējas tradicionālās programmatūras izstrādes metodoloģijās, ieviešot MDA principu lietošanu.
4. Definēt formālu pieeju pārējai no tradicionālās programmatūras izstrādes „MDA orientētajā".
5. Aprobēt piedāvāto pieeju programmatūras izstrādes projektā, un novērtēt pieeju iespējas, perspektīvas un ierobežojumus.
6. Izdarīt secinājumus par MDA lietošanas iespējām, problēmām un perspektīvām programmatūras izstrādē un pārejā no tradicionālās programmatūras izstrādes uz modeļvadāmo.

Pētījuma metodes

Informācijas avotu analīze ir veikta balstoties uz pieejamo literatūru par pētāmo problēmsfēru, kas ietver sevī grāmatas, žurnālus, un tematisko konferenču materiālus. Tas ļauj pārzināt problēmvides aktuālo stāvokli un pēdējās aktualitātes. Šī informācija palīdz ģenerēt idejas iespējamai modeļvadāmās programmatūras izstrādes komponentu kartēšanai (angl. *mapping*) tradicionālajā programmatūras izstrādē. Informācijas plūsmas izpētei un modelēšanai starp programmatūras izstrādes procesa soļiem, nepieciešams analizēt informāciju par programmatūras izstrādes dzīves cikliem (un to attēlošanu ar procesu modeļiem). Sasaistot MDA specifiskās aktivitātes ar tradicionālo programmatūras izstrādi, ar analīzes metodēm iespējams definēt vadlīnijas jebkura programmatūras izstrādes procesa pielāgošanai modeļvadāmai izstrādei.

Promocijas darbā ir izmantotas programmatūras izstrādes procesa modelēšanas metodes (analīze, modelēšana un rezultātu novērtēšana). Transformācijai starp modeļiem tiek izmantota imperatīvā pieeja, kas definē programmas stāvokļa izmaiņu (transformācijas tiek aprakstītas valodā QVT Relations). Vēl ir veikta pieejas dzīvotspējas novērtēšana un izstrādātās pieejas pielietošana jeb aprobācija praksē, un izstrādāta risinājuma arhitektūras novērtēšana reālajos darba apstākļos.

Par darba pētījuma bāzi izmantoti zinātniski-pētnieciskie projekti un informācijas sistēmu izstrādes projekti ar autora līdzdalību, kas bija organizēti pēc dažādiem programmatūras izstrādes dzīves cikla modeļiem. Promocijas darba autors piedalījās vairākos industrijas projektos uzņēmumā Accenture, kur autors strādā par SAP konsultantu, un piedalījās vairāku starptautisko projektu implementācijā, sākot no projekta agrīnām fāzēm un beidzot ar gatavas sistēmas ieviešanu un atbalstu. Autors ir piedalījies, kā izpildītājs vai galvenais izpildītājs, šādos zinātniskajos un mācību metodiskajos projektos:

1. LZP lietišķo pētījumu projekta Nr. 09.1269 "Uz izkliedēta mākslīgā intelekta un tīmekļa tehnoloģijām balstītas metodes un modeļi intelektuālas lietišķās programmatūras un datorsistēmu arhitektūras izstrādei", virziena "Programmatūras izstrāde MDA ietvarā" (2009.-pašlaik)
2. RTU pētniecības projekts Nr. FLPP-2009/10 „Konceptuālā modeļa izstrāde pārejai no tradicionālās programmatūras izstrādes MDA orientētajā" (2009.)
3. Līdzdalība ESF līdzfinansētā projektā „Studiju moduļa izstrāde modeļvadāmai programmatūras attīstības tehnoloģijai datorsistēmas programmā" (Līgums Nr. 2007/0080/VPD1/ESF/PIAA/06/APK/3.2.3.2./0008/0007) (2006.-2008.)
4. Līdzdalība ESF līdzfinansētā projektā 3232/15 „RTU studiju programmas "DATORSISTĒMAS" pilnveidošana absolventu profesionālās konkurētspējas paaugstināšanai" kursu pilnveidošana" (2006.-2008.)
5. RTU zinātniskais projekts ZP - 2005/02 „Divpusložu pieejas lietojums elastīgas programmatūras inženierijas zināšanu arhitektūras izstrādē" (2005.-2006.)

Promocijas darba novitāte

Promocijas darba novitāte ir:

1. Modeļvadāmās izstrādes artefaktu kartēšana (angl. *mapping*) tradicionālajā programmatūras izstrādes procesā.
2. Pārejas risinājuma arhitektūra un transformācijas programmatūras izstrādes procesu modeļa pārveidošanai, lietojot QVT Relations.
3. Autora piedāvātās pieejas demonstrācijas piemērs un tās aprobācija vienā no Latvijas programmatūras izstrādes uzņēmumiem.

Promocijas darba praktiskā nozīme

Ar kompānijas „PF" piemēru ir pierādīta piedāvātas pieejas dzīvotspēja. Tas apliecina iespējamību izmantot šo pieeju arī citās programmatūras izstrādes kompānijās, kur programmatūras izstrādes dzīves cikls var būt atšķirīgs.

Iegūto darba rezultātu publikācijas un to prezentācija konferencēs

Pētījuma rezultāti ir iegūti un publicēti laika posmā no 2003. līdz 2011. gadam, darba autoram studējot RTU Datorsistēmu studiju programmas bakalaurantūrā, maģistrantūrā un doktorantūrā. Tie ietver 11 zinātniskās publikācijas starptautiskos izdevumos un konferenču rakstu krājumos. Papildus tam ir divas tēžu publikācijas RTU studentu zinātniski tehniskās konferences materiālos. Promocijas darba pētījuma ietvaros izstrādāto artefaktu apraksti ir ievietoti piecos mācību līdzekļos, un ir izmantoti dažos RTU Lietišķo datorsistēmu institūta mācību priekšmetos. Piedāvātas pieejas aprobācijas rezultāti ir publicēti zinātniskā projektā atskaitē [NIK 2009e]. Autora publikāciju saraksts ir pievienots izmantotās literatūras sarakstam kopsavilkuma beigās.

Promocijas darba autors ir piedalījies ar referātu šādas konferencēs:

1. Nikulsins V. Transformations of Software Process Models to Adopt Model-Driven Architecture. ENASE 2010: 2nd International Workshop on Model-Driven Architecture and Modeling Theory-Driven Development MDA&MTDD. July 22-24, 2010, Athens, Greece
2. Nikulsins V., Nikiforova O. Tool Integration to support SPEM Model Transformations in Eclipse. The 50th RTU International Scientific Conference. October 12-16, 2009, Riga, Latvia
3. Nikulsins V., Nikiforova O. Transformations of SPEM models using query/view/transformation language to support adoption of model-driven software development lifecycle, ADBIS-2009. Model – Driven Architecture: Foundations, Practices and Implications (MDA) workshop. Riga Technical University, September 7-10, 2009, Riga, Latvia
4. Nikulsins V., Nikiforova O. Adapting Software Development Process towards the Model Driven Architecture. The Third International Conference on Software Engineering Advances, ICSEA 2008, October 26-31, Sliema, Malta
5. Nikulsins V., Nikiforova O., Sukovskis U. Mapping of MDA Models into the Software Development Process, The 8th Biennial International Baltic Conference on Databases and Information Systems, Baltic DB&IS, July 2-5, 2008, Tallinn, Estonia
6. Nikulsins V., Nikiforova O., Sukovskis U. Analysis of Activities Covered by Software Engineering Discipline, The 7th Biennial International Baltic Conference on Databases and Information Systems, Baltic DB&IS, July 3-6, 2006, Vilnius, Lithuania
7. Nikulshin V., Nikiforova O. Review on Allocation of Roles and Responsibilities among Software Development Team, The 46th Scientific Conference of Riga Technical University, Computer Science, Applied Computer Systems, October 13-14, 2005, Riga, Latvia

Promocijas darba struktūra

Darba ievadā ir aprakstīta pētījuma aktualitāte, zinātniskā novitāte, darba mērķis, uzdevumi, zinātniskā un praktiskā nozīme. Ir sniegts arī darba struktūras apraksts.

Darba pirmajā nodaļā ir sistematizētas un aprakstītas zināšanas par programmatūras izstrādes procesu, dažādu tā veidojumu definīcijām, standartiem un modernām metodoloģijām, kas tiek pielietotas programmatūras izstrādē.

Darba otrajā nodaļā ir izklāstīta programmatūras izstrādes procesa modernizācija, saistībā ar MDA ieviešanu tajā. Ir definēti galvenie MDA principi un pamatkoncepcijas.

Darba trešajā nodaļā ir aprakstīti autora mēģinājumi veikt programmatūras izstrādes procesa integrāciju ar MDA artefaktiem.

Pirmajās trīs nodaļās izklāstītais teorētiskais materiāls un pētījuma rezultāti ļāva autoram izvirzīt hipotēzi par iespējamo risinājumu, pārejas pieejas izstrādei no tradicionālā programmatūras izstrādes procesa uz modeļvadāmo programmatūras izstrādes organizāciju, kas tiek detalizēti aprakstīta darba ceturtajā nodaļā. Ceturtajā nodaļā arī ir aprakstīts autora izstrādāta rīka prototips, kas atbalsta definēto pieeju.

Darba piektajā nodaļā ir parādīti daži demonstrācijas piemēri piedāvātās pieejas lietošanai praksē. Nobeigumā autors apraksta galvenos pētījuma sasniegumus un izsaka svarīgākos secinājumus.

1. PROGRAMMATŪRAS IZSTRĀDES PROCESA STANDARTIZĀCIJA UN METODOLOĢIJAS

Pirmajā nodaļā ir aprakstīta programmatūras inženierijas vēsture un attīstība, kopš programmatūras inženierija bija pasludināta par inženierijas disciplīnu NATO konferencē, kas notika 1968. gadā [NAU 1969], [VLI 2008]. Tiek sniegtas arī pamatdefinīcijas, kas tiek izmantotas promocijas darbā. Šajā nodaļā ir aprakstīti standartizācijas pasākumi un pazīstamākās programmatūras metodoloģijas.

1.1. Termini un definīcijas

Programminženierijai attīstoties, mainās arī tās terminoloģijas definīcijas. Daži no terminiem kļūst divdomīgi. Piemēram, bieži tiek jauktas „izstrādes procesa" un „metodoloģijas" definīcijas. Daži latviskotie termini ir pieejami tīmekļa vietnē [CAU]. Šie termini promocijas darba ietvaros ir uzlaboti vai papildināti.

Programminženierija (angl. *Software engineering*) – sistematizēta, disciplinēta un kvantificējama pieeja zinātnisko un tehnoloģisko zināšanu, metožu un pieredzes izmantošanai funkcionāli efektīvas programmatūras izstrādāšanas, pielietošanas un ieviešanas procesā. Zinātnes nozare par programmatūras izstrādi [CAU], [SCA 2001], [SWE 2004], [IEE 1990].

Programmatūras izstrādes process (angl. *Software development process*) – dažreiz tiek lietots termins programmatūras process (angl. *Software process*), ir process, kurā lietotāja (pasūtītāja) prasības tiek pārveidotas programmatūras produktā. Šīs process iekļauj sevī lietotāja prasību transformēšanu programmatūras prasībās, programmatūras prasību transformēšanu projektējumā, projektējuma implementēšanu kodā, koda testēšanu un dažreiz instalēšanu. Šīs aktivitātes var pārklāties vai izpildīties iteratīvi [IEE 1990].

Programmatūras dzīves cikls (angl. *Software life cycle*) – viss programmatūras pastāvēšanas laiks, no tās izstrādāšanas sākuma līdz brīdim, kad tā ir zaudējusi savu vērtību. Programmatūras dzīves cikla galvenās fāzes ir analīze, projektēšana, izstrādāšana, ekspluatācija un, iespējams, arī modernizēšana [CAU]. Dažreiz tiek uzskatīts, ka cikls beidzas līdz ar sistēmas ieviešanu lietošanā [IEE 1990]. Viens no izplatītākajiem industrijas standartiem ir [IEE 2008].

Programmatūras izstrādes dzīves cikls (angl. *Software development life cycle* – SDLC), dažreiz tiek saukts par sistēmu izstrādes dzīves ciklu (angl. *Systems development life cycle*) – pēc būtības sinonīms programmatūras izstrādes procesam, un ir programmatūras produktu izstrādei paredzēta struktūra. Atšķirībā no programmatūras dzīves cikla, šis termins ir vairāk specifisks, un ir raksturīgs pašai programmatūras izstrādei, nevis programmatūras dzīves ciklam kā tādam.

1.2. Programmatūras izstrādes procesa organizācija

Fundamentālajās zinātnēs, piemēram, fizikā un matemātikā, kā arī sociālajās zinātnēs un ekonomikā, eksistē nostabilizējušās tradīcijas un formāli pamati attiecīgo mehānisko sistēmu, matemātisko vienādojumu vai ķīmisko vielu izstrādē un definēšanā. Taču programmatūras inženierija ir relatīvi jauna, tāpēc heterogēna inženierijas joma, kur formālās normas un likumi programmatūras sistēmu izstrādei vēl nav iedibināti, jo šajā sfērā ir visai sarežģīti noteikt robežas starp nepieciešamo formalizācijas pakāpi un problēmas risināšanas uztveramību un vienkāršību. Turklāt, pētījumi šajā jomā bieži vien prasa ņemt vērā ne tikai tehnoloģiskos, bet arī sociālos un ekonomiskos ar informācijas tehnoloģiju risinājumu izstrādi saistītos aspektus [NIK 2006c]. Īpaši šī problēma ir aktuāla, mēģinot formalizēt pašu programmatūras izstrādes procesu.

Tomēr programmatūras izstrāde ir organizatorisks process, kuru iespējams attēlot ar modeļa palīdzību, šādi sasniedzot minimāli nepieciešamo formalizācijas pakāpi. Pamatelementi šajā modelī ir fāzes, aktivitātes, artefakti un lomas, kas ir sasaistīti savā starpā. Vienam programmatūras izstrādes dzīves ciklam var piemist vairāki programmatūras izstrādes modeļi, kas apraksta uzdevumus un aktivitātes programmatūras izstrādes procesam.

Programmatūras izstrādes procesa izvēle ir atkarīga no vairākiem faktoriem, tādiem kā projekta lielums, izstrādātāju komanda un tās organizācija, klienta prasības, utt. [COC 1999]. Programmatūras izstrādes procesu definējoši standarti un zināšanu korpusi (angl. *body of knowledge*) ļauj noteikt formālus kritērijus gan programmatūras sistēmām, gan to izstrādei.

1.3. Programmatūras izstrādes standarti

Programminženierijas pamatā ir pieņēmums, ka metodiska pieeja programmatūras izstrādē, nodrošina mazāk defektu un kļūdu, un palīdz izveidot programmatūru ātrāk un kvalitatīvāk. Programminženierijas standarti programmatūras izstrādes dzīves ciklā apskata programmatūras izstrādes aspektus un to standartizāciju izstrādes aktivitāšu organizācijā, klasifikācijā un grupēšanā visā programmatūras izstrādes dzīves ciklā, kā arī paša dzīves cikla vispārīgu organizāciju [SWE 2004]. Promocijas darba 1.3. nodaļa apraksta dažas ar programminženierijas standartizācijas jautājumiem saistītas organizācijas un to izstrādātos standartus.

Programmatūras inženierijas standartus ir iespējams izmantot programmatūras izstrādes procesa formalizācijai tādā ziņā, ka tie dod vispārīgu un strukturētu informāciju par dažādām artefaktu grupām. Tas dod pamatu veikt programmatūras izstrādes artefaktu grupēšanu, klasifikāciju, nepieciešamo detalizāciju vai abstrahēšanos no detaļām un tml. Standarti ir neatkarīgi no programmatūras izstrādes metodoloģijas un sniedz vispārīgu ieskatu procesa organizācijā, tādējādi nodrošinot nepieciešamo abstrakcijas līmeni programmatūras procesa modelēšanai.

1.4. Programmatūras izstrādes dzīves cikla modeļi

Programmatūras izstrādes dzīves cikla modeļi nosaka programmatūras izstrādes stratēģiju. Kā piemērus tipiskākiem programmatūras dzīves cikla modeļiem var minēt ūdenskrituma modeli [ROY 1970], spirālveida modeli [JAY 2007], [TSU 2011], inkrementālo modeli [THA 2005], [SCA 2001], [TSU 2011] un V-veida modeli [THA 2005], [VLI 2008]. Tie ir aprakstīti promocijas darba 1.4. sadaļā.

1.5. Programmatūras izstrādes metodoloģijas

Programminženierijā un projektu vadībā par programmatūras izstrādes metodoloģiju sauc programmatūras izstrādes procesa rekomendēto prakšu sistematizētu kopu, kas var būt atbalstīta ar apmācības materiāliem, formālām apmācības programmām un rīkiem [ESS 2009]. Promocijas darba 1.5.sadaļa apskata divas smagsvara metodoloģijas:

1. Microsoft risinājumu ietvaru (angl. *Microsoft Solutions Framework* – MSF) [MIC 2003];
2. Rational vienoto procesu (angl. *Rational Unified Process* – RUP) [RAT 1998].

No spējās (angl. *agile*) programmatūras izstrādes procesu saimes, tiek apskatīta ekstremālā programmēšana (angl. *Extreme programming* – XP) [WAK 2001].

Gan smagsvara, gan spējās izstrādes metodoloģijas ļauj sasniegt vienu un to pašu mērķi – izstrādāt programmatūru, balstoties uz formālu pieeju un pārbaudītām praksēm, minimizējot izmaksas un kontrolējot izstrādes procesu. Smagsvara metodoloģijas precīzi definē izstrādes artefaktus un aktivitātes. Kaut gan pēdējos gados arī smagsvara metodoloģiju vidū ir vērojama spējās izstrādes ietekme. Izstrādājot procesu modeli, ir iespējams izvēlēties tādu procesa abstrakcijas līmeni, kas ļauj izvairīties no atšķirībām starp abām metodoloģijām. Spējo izstrādi arī ir iespējams formalizēt ar procesu modeli, jo spējā izstrāde definē tās pašas fāzes, procesus, aktivitātes un artefaktus, tikai šis process ir vairāk elastīgs.

Programmatūras izstrādes procesu modelim piemīt gan deklaratīvais (kā ir jānotiek programmatūras izstrādei), gan aprakstošais (kā programmatūras izstrāde notiek pašlaik) raksturs. Tādejādi, programmatūras izstrādes modernizācijas un izstrādes procesa maiņas gadījumā, ir iespējams izmantot procesu modeļus, kas ļauj to formāli un deklaratīvi aprakstīt.

2. PROGRAMMATŪRAS IZSTRĀDES PROCESA MODERNIZĀCIJA MODEĻVADĀMAS ARHITEKTŪRAS KONTEKSTĀ

Programmatūras izstrādes nākotnes tendences ir globalizācija, mērogošana un dažādu sistēmu integrācija [BOT 2010]. Noteicošā loma šajā procesā ir sistēmas loģikas un biznesa procesu modelēšanai, ko pašlaik ir iespējams veikt ar modelēšanas valodām, piemēram, ar UML vai ar BPMN. Viens no risinājumiem ir OMG konsorcija 2001. gadā piedāvātā modeļvadāmā arhitektūra jeb MDA [SIE 2001]. MDA konceptuāli maina programmatūras izstrādes prioritātes no koda centriskās pieejas (kods kā galvenais artefakts) uz modelēšanas pieeju (modelis kā galvenais artefakts) [OSI 2010]. Tādejādi projektēšana kļūst par risinājuma daļu, un ir izteikta ar modeļu

palīdzību. Jebkādas izmaiņas projektēšanā ietekmēs modeļus, kurus būs iespējams transformēt izpildāmā kodā. Atšķirībā no tradicionālas uz kodu orientētas izstrādes, kur izpildāmais kods parādās projekta beigās, MDA bāzētā pieeja dod iespēju iegūt modeli (kas jau ir kods) projekta sākumā. Tas dod iespēju izvairīties no pārliekas resursu tērēšanas un situācijas, kad izstrādāta programmatūras sistēma izrādās neatbilstoša biznesa loģikai.

MDA ļauj apskatīt sarežģītas izstrādājamas sistēmas no dažādiem abstrakcijas līmeņiem, gan biznesa modeļa līmenī (neatkarīgi no tehnoloģijas), gan platformas specifiskajā līmenī (ņemot vērā tehnoloģijas specifiku), kas tiek iegūts no biznesa modeļa [OMG], [NIK 2008c]. MDA nosaka kā biznesa modelis jeb no platformas neatkarīgas modelis ir jātransformē uz platformas specifisko modeli.

Līdzīgi var modelēt arī pašu izstrādes procesu un savienot aktivitātes, lomas un artefaktus, un aprakstīt to mijiedarbību savstarpēji saistīta tīkla veidā jeb procesu modeļos [SCA 2001]. Procesu modeļus var izmantot jauna programmatūras izstrādes procesa ieviešanai un darbinieku apmācībai.

2.1. MDA pamatkoncepcijas

MDA ir arhitektūra, kas ir bāzēta uz UML valodas un citiem programminženierijas industrijas standartiem modeļu un projektējuma vizualizēšanai, glabāšanai un apmaiņai. MDA ļauj izveidot augstas abstrakcijas modeļus, kuri nav atkarīgi no izpildīšanas platformas, un kas glabājas specializētos standartizētos repozitorijos.

MDA ietver šādas tehnoloģijas: vienota modelēšanas valoda (angl. *Unified Modeling Language* – UML), metaobjektu iespējas (angl. *Meta-Object Facilities* – MOF), XML metadatu apmaiņa (angl. *XML Metadata Interchange* – XMI) un kopējais krātuves metamodelis (angl. *Common Warehouse Metamodel* – CWM) [OMG].

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. Modelis var būt izteikts kā UML diagrammu, OCL specifikāciju un teksta kopums. MDA izdala dažādus modeļu tipus, kas var būt abstrakti (specificē sistēmas funkcionalitāti) un konkrēti, kas saistīti ar specifisku platformu, tehnoloģiju un implementāciju. MDA modeļu tipi ir šādi [OMG], [FAV 2010], [PIL 2005]:

- CIM modelis (angl. *Computation Independent Model*) jeb no skaitļošanas neatkarīgais modelis.
- PIM modelis (angl. *Platform Independent Model*) jeb no platformas neatkarīgais modelis.
- PSM modelis (angl. *Platform Specific Model*) jeb no platformas atkarīgais modelis.
- Koda modelis (angl. *Code Model*), retāk tiek lietots apzīmējums ISM modelis (angl. *Implementation Specific Model*) [BRO 2005a].

MDA paredz, ka ir 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 nākošā līmeņa modeļi. Piemēram, vienam PIM var atbilst vairāki PSM, kas apraksta sistēmas modeļus dažādām platformām. Transformācijas starp MDA modeļiem notiek ar marķēšanas (angl. *marking*) palīdzību: elementi avota modelī tiek marķēti un saistīti ar elementiem mērķa modelī.

Modeļu transformācijas balstās uz **metamodelēšanas principiem** [EVA 2003]. Metamodelēšana ir MDA bāzes tehnika. MDA ir bāzēts uz platformas modeļiem aprakstītiem ar UML, OCL, kas tiek saglabāti MOF repozitorijā [MOF 2008]. **Metamodelis** (jeb „modeļa modelis") ir modelēšanas valodas modelis, kas definē sintaksi un semantiku modelēšanas valodai un nodrošina sadarbspēju starp modelēšanas procesu un transformācijas rīkiem. **MOF** ir OMG konsorcija standarts, kas ir paredzēts metamodeļu specificēšanai, izstrādei un vadīšanai. Tas definē valodu, kas savukārt definē modelēšanas konstrukciju kopu (jeb modelēšanas valodas sintaksi un semantiku), ko modelētājs var lietot, lai definētu un manipulētu ar sadarbspējīgo metamodeļu kopu. MOF ir starptautisks standarts [ISO 2005]. MOF nodrošina metametamodelēšanu UML metamodeļiem un definē modelēšanas konstrukciju kopu, kas ļauj definēt un manipulēt ar modeļa metadatiem. Pats MOF ir definēts UML valodas notācijā, un ir UML 2.x bāzes paplašinājums [FAV 2010].

2.2. MDA artefaktu lietošana programmatūras izstrādes fāzēs

No procesu viedokļa MDA nepiedāvā izstrādes metodoloģiju, motivējot to ar iespējamību izmantot to jebkurā izstrādes procesā. Tā ir gan MDA priekšrocība, gan trūkums, jo ir definēta pati arhitektūra un tās īpašības, nevis šīs arhitektūras izmantošanas process vai pārejas vadlīnijas izmantošanai no t.s. tradicionālas (uz kodu orientētas) izstrādes, modeļvadāmajā [MEL 2004].

Vispārīgā gadījumā, modeļvadāmā pieeja tiek izmantota vairākos dzīves cikla posmos – prasību strukturizēšanā, biznesa analīzē, procesu modelēšanā, sistēmas projektējuma izstrādē, servisu definēšanā, sistēmu integrēšanā, risinājumu projektēšanā, pirmkoda ģenerēšanā, automātisko transformāciju veikšanā, u.tml. MDA apvieno visas šajos posmos iesaistītās aktivitātes, veidojot modeļvadāmās programmatūras izstrādes dzīves ciklu. Turklāt, ir iespējams atdalīt augstāka līmeņa biznesa procesu aprakstošus modeļus no tiem, kas apraksta izstrādājamās sistēmas arhitektūru un ieviešanas platformu, lai pēc tam nodrošinātu to atkārtotu lietošanu citu lietojumu izstrādē.

Tā kā MDA nenosaka programmatūras izstrādes procesu, MDA pielāgošanai tradicionālajam izstrādes procesam ir nepieciešams analizēt ar procesu integrēšanu saistītus aspektus [CHI 2007]. Šīm nolūkam ir veltīti vairāki pētījumi [GAV 2004], [MAN 2006], [BEL 2002], [ESI 2002], [ESI 2003], [VOG 2006], kuru rezultātus autors analizēja promocijas darbā. No aktivitāšu viedokļa un to savstarpējām atkarībām, MDA procesu apskata [MEL 2004]. Šīs aktivitātes tiek apskatītas vienkāršotai hipotētiskai sistēmai, ar vienu avota modeli un vienu mērķa implementēšanas modeli. Vēlāk šis vienkāršais hipotētiskās sistēmas apraksts tiek papildināts, apskatot MDA iteratīvo izstrādi un modeļa realizēšanu vairākās platformās [MEL 2004]. Analizējot vairākus literatūras avotus [MEL 2004], [KLE 2003], [MEL 2002], [SIE 2001] u.c., promocijas darba autors ir nodefinējis MDA aktivitātes ar to ieejām un izejām (sk. promocijas darba 2.1. tabulu). Šīs tabulas informācija dod pamatu veikt MDA aktivitāšu integrāciju ar tradicionālā programmatūras izstrādes procesa aktivitātēm (tās ir aprakstītas promocijas darba 3.1 un 3.2 nodaļā).

2.3. MDA brieduma līmeņu analīze

Sekmīgai pārejai uz modeļvadāmo izstrādi, organizācijā ir nepieciešams novērtēt, cik lielā mērā pašreizējais izstrādes process atbilst modeļvadāmai izstrādei, un cik liels uzsvars ir uz modeļiem un modelēšanas aktivitātēm. Pēc Forrester pētījuma klasifikācijas [FOR 2009] ir iespējams novērtēt brieduma līmeni modeļvadāmai programmatūras izstrādei organizācijā [RIO 2006]. Tas ir aprakstīts promocijas darba 2.3. nodaļā.

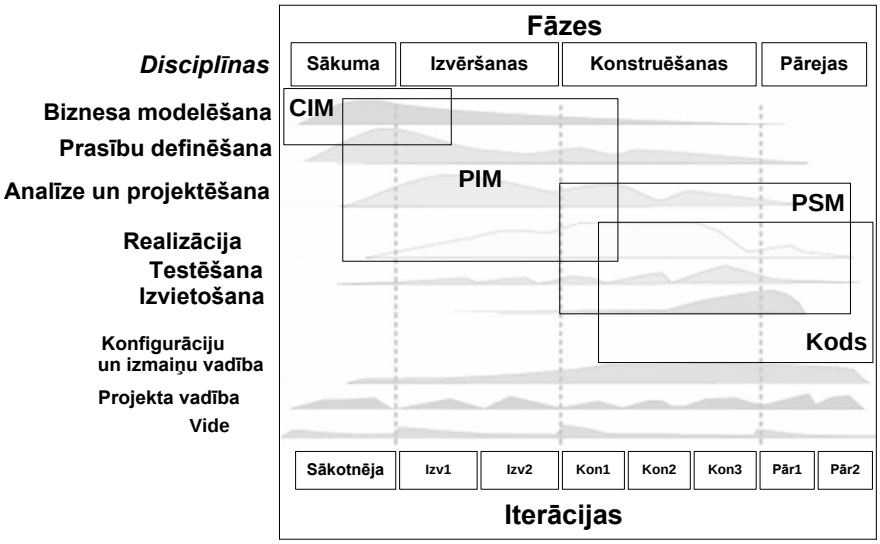
Reālajā dzīvē MDA paņēmienu ieviešana programmatūras izstrādē nav triviāls uzdevums, jo var saskarties ar vairākām problēmām. Veidojot pilnvērtīgu MDA izstrādes dzīves ciklu, ir jāievēro gan OMG konsorcija sniegtās rekomendācijas par programmatūras izstrādi MDA kontekstā (tajā skaitā standarti, vadlīnijas, dzīves cikla posmi, u.tml.), gan arī vispārīgus modeļvadāmā izstrādes procesa aspektus (nodrošināt modeļu repozitoriju, attiecīgo darbības vidi, u.tml.). Tomēr abu šo nosacījumu ievērošana negarantē, ka patvaļīgi izveidotā MDA ieviešana ierastajā programmatūras izstrādes procesa organizācijā būs arī darbspējīga reālajā dzīvē. Saderības nodrošināšanai starp vairākiem MDA artefaktiem un to integrācijai ar tradicionālo programmatūras izstrādes procesu, ir nepieciešams pārbaudīt, kā abu pieeju artefakti kartējas, un kādi tiem ir saskarnes punkti.

3. MDA ARTEFAKTU IEVIEŠANA PROGRAMMATŪRAS IZSTRĀDES METODOLOĢIJAS

Tradicionāli programmatūras izstrādes dzīves ciklu var iedalīt sešās fāzēs: prasības, analīze, projektēšana, kodēšana, testēšana un uzturēšana. Šajās fāzēs rodas problēmas, piemēram, ja esošā platforma jāaizstāj ar citu vai rezultāts atšķiras no prasībām. Modeļvadāmā arhitektūra balstās uz modeļu veidošanu un to transformācijām. Uz katru programmatūras izstrādes fāzi var attiecināt kādu modeli [NIK 2008a]. Autors promocijas darba ietvaros vairāk fokusējas uz smagsvara metodoloģijām un to ievērojamākiem pārstāvjiem, proti RUP un MSF. Abās metodoloģijās ir iespējams iekļaut MDA, attēlojot tās vienkopus, lai redzētu, kurai fāzei atbilst noteikti MDA modeļi [NIK 2008b], [NIK 2009c]. Kopsakarību meklēšana aktivitātēs, palīdz integrētas pieejas izveidē. Tā kā MDA ir iespējams saskatīt kopsakarības ar tradicionālo programmatūras izstrādi, tad atrastās kopīgās aktivitātes grupē pēc MDA principiem.

3.1. MDA realizācija RUP definētajā procesu organizācijā

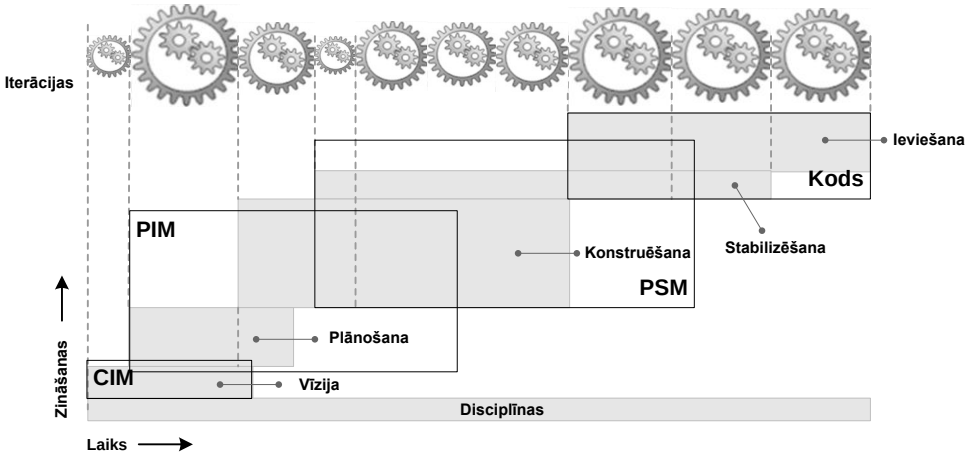
RUP nepiedāvā specifiskus ieteikumus MDA integrācijai ar savu programmatūras izstrādes procesu. Pamatojums šim faktam ir sekojošs: RUP atspoguļo pašreizējās labās prakses un neietver sevī pieejas, kas nav plaši lietotas un pieņemtas. MDA ir jauna un uz nākotni virzīta pieeja, tāpēc nozīmīgs MDA un RUP apvienošanas process nav vēl nodibināts. RUP pamats ir uz arhitektūru virzīts iteratīvs izstrādes process, kas ir ļoti konsekvents ar MDA koncepcijām. Eksistē vairāki pētījumi par MDA integrāciju ar RUP, kas izmanto dažādas pētījuma metodes [ESI 2002], [ESI 2003], [BRO 2005b]. Apkopojot vairāku projektu pieredzi, ir izstrādāti ieteikumi MDA aspektu adaptēšanai RUP procesam, tie ir pārskaitīti promocijas darba 3.1 nodaļā. Kaut gan RUP programmatūras izstrādes procesa fāžu kartēšanai MDA nav oficiālas rekomendācijas, ir iespējams veikt MDA izstrādes etapu un RUP izstrādes procesu kartēšanu, kā tas parādīts 1. attēlā [NIK 2008a]. Pilna informācija par MDA principu integrēšanu RUP programmatūras izstrādes procesa organizācijā ir aprakstīta promocijas darba 3.1. nodaļā.



1. att. MDA un RUP fāžu kartēšana

3.2. MDA realizācija MSF definētajā procesu organizācijā

Oficiāli programmatūras izstrādes metodoloģija MSF neatbalsta MDA. Nav arī veikti atsevišķi pētījumi par MDA un MSF procesu apvienošanu. Tomēr, veicot abu tehnoloģiju analīzi MSF procesu modelim ir iespējams definēt vispārīgus MDA kartēšanas likumus (sk. 2. att.) [NIK 2008a].



2.att. MDA un MSF fāžu kartēšana

Pilna informācija par MDA principu integrēšanu MSF programmatūras izstrādes procesa organizācijā ir aprakstīta promocijas darba 3.2. nodaļā.

3.3. MDA aktivitāšu integrēšana RUP un MSF procesos

MDA principus ir iespējams iekļaut abās (t.i. RUP un MSF) metodoloģijās. Promocijas darba 3.3. nodaļā ir aprakstīti fāzēm atbilstošie modeļi un modeļu pārveide kodā. Tiek attēlots, kurām fāzēm abās metodoloģijās MDA principi atbilst. Kā arī ir aprakstīts, kuri MSF un RUP metodoloģiju procesi atbilst kuriem MDA procesiem. Ar attiecīgajiem RUP un MSF procesiem, tiek attēlots, kādi MDA artefakti var aizvietot vai bagātināt attiecīgos programmatūras izstrādes artefaktus.

3.4. SPEM pamatkoncepcija

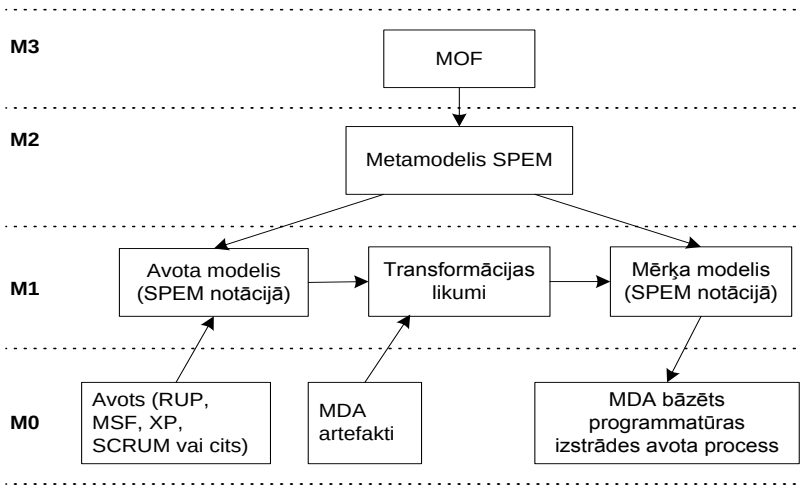
Programminženierijas procesa metamodelis (angl. *Software Process Engineering Metamodel* – SPEM) ir OMG konsorcija specifikācija, kas ir izveidota, lai aprakstītu noteiktu programmatūras izstrādes procesu, vai ar to saistīto programmatūras izstrādes procesu grupu. SPEM 2.0 ir definēts kā metamodelis, kā arī UML 2 profils. OMG standarti, kas ir saistīti ar MDA izmantošanu, ir definēti metamodeļu līmenī. Līdzīgā veidā OMG definē SPEM metamodeli: SPEM ir definēts ar metamodeli UML valodā, kas, savukārt, pats ir definēts kā MOF eksemplāra metametamodelis.

SPEM 2.0 ir paredzēts programmatūras un sistēmu izstrādes procesu, kā arī tās komponentu definēšanai. SPEM ir ierobežots ar minimālu elementu skaitu, kas nepieciešami programmatūras izstrādes procesu attēlošanai. Tas neietver specifisku elementu izmantošanu specializētiem domēniem vai disciplīnām (piemēram, programmas vadību). SPEM mērķis ir pielāgot dažādas izstrādes metodes, procesus, formalizācijas līmeņus, dzīves cikla modeļus un sabiedrības. Tomēr galvenais SPEM uzsvars ir izstrādes projekti. SPEM 2.0 nav vispārēja procesu modelēšanas valoda, un nepiedāvā savus modelēšanas konceptus.

SPEM ļauj atspoguļot programmatūras izstrādes procesu, izmantojot lomas, aktivitātes, uzdevumus, artefaktus un darba produktus. Papildus, SPEM piedāvā vispārpieņemtu sintaksi un struktūru vairākām tehnikām un rīkiem. Izmantojot MDA transformācijas iespējas (piem., MOF un QVT), SPEM var būt pārveidots un transformēts BPMN valodā vai arī citās procesu valodās.

3.5. Programmatūras izstrādes procesa pārejas pieejas koncepcija

Promocijas darba autors piedāvā veikt MDA ieviešanas procesu tradicionālajā programmatūras izstrādē, izmantojot formālus modeļu transformācijas paņēmienus. Par pamatu tam var kalpot autora pētījumu rezultāti šajā jomā: programmatūras izstrādes process SWEBOK kontekstā ir analizēts [NIK 2006b], programmatūras izstrādes procesa struktūras izstrāde, vadoties no standartiem (ISO, CMMI) programmatūras projektu izstrādātāju lomu kontekstā ir aprakstīta [NIK 2006c] un daži rezultāti par RUP un MSF metodoloģiju lietošanu ir aprakstīti [NIK 2008a], [NIK 2008b], [NIK 2008c]. Hipotētiskā risinājuma kopēja uzbūve ir parādīta 3. attēlā [NIK 2010a].



3. att. Hipotētiskā risinājuma attēlojums MDA četru līmeņu arhitektūrā

SPEM notācija ir OMG standartizēts un formāls programmatūras izstrādes procesa modelēšanas veids, taču tā nav paredzēta programmatūras izstrādes procesa modernizācijai vai viena programmatūras izstrādes procesa pārejai citā programmatūras izstrādes procesā. Tomēr darba

autors uzskata, ka SPEM elementi var tikt izmantoti arī risinājumam, kas ir domāts tieši pārejai no viena programmatūras izstrādes procesa citā.

Ņemot vērā to, ka SPEM galvenie artefakti ir modeļi, var pieņemt, ka modeļvadāmas programmatūras izstrādes principu izmantošana varētu arī būt šādas pieejas pamatā. Autors izvirza hipotēzi par to, ka, specifiski integrējot šos divus principiālos risinājumus, proti, SPEM un MDA, var definēt pieeju, pārējas problēmas risināšanai no viena programmatūras izstrādes procesa modeļa citā. MDA arhitektūras kontekstā, par avota modeli kalpo problēmvides attēlojums, bet par mērķa modeli kalpo programmatūras komponenti (piem., CIM un PIM vai PIM un PSM). Ar transformācijas palīdzību viens modelis tiek transformēts citā modelī.

Izvēloties programmatūras izstrādes dzīves cikla modeli (kas atbilst kādas specifiskas organizācijas tradicionālajam izstrādes procesam) kā avota modeli, var pieņemt ka eksistē tāds programmatūras izstrādes process, kas apraksta programmatūras izstrādes procesu atbilstoši modeļvadāmajai paradigmai. Tāds tiek uzskatīts par mērķa modeli. Savukārt SPEM ļauj aprakstīt jebkuru programmatūras izstrādes procesu ar modeļa palīdzību [SPE 2008], tādejādi gan avota, gan mērķa modelis var būt aprakstīts SPEM notācijā, jo abi modeļi atbilst SPEM metamodelim.

Katram programmatūras izstrādes procesam var specificēt gan statiskus aspektus (aktivitātes, artefaktus, lomas), gan dinamiskus (cikli, fāzes, iterācijas). Salīdzinot divus izstrādes procesus, ir iespējams atrast procesu likumsakarības, procesu kopīgos elementus, un definēt vispārīgas procesu grupas [Ņikuļšins 2006], kas nodrošina abu procesu loģisku savienošanu. Loģiska elementu savienošana MDA kontekstā ir pamats transformāciju definēšanai. Apskatot procesu aspektus kā atribūtus transformācijām, ir iespējams definēt priekšnosacījumus transformāciju izpildīšanai, respektīvi, definēt SPEM modeļu transformācijas likumus. Šo likumu definēšanai ir nepieciešams iegūt tradicionālā programmatūras izstrādes procesa formalizētus aspektus, kas būtu tipiski visiem programmatūras izstrādes procesiem un kuru elementus varētu viennozīmīgi sasaistīt ar MDA bāzēta izstrādes procesa elementiem.

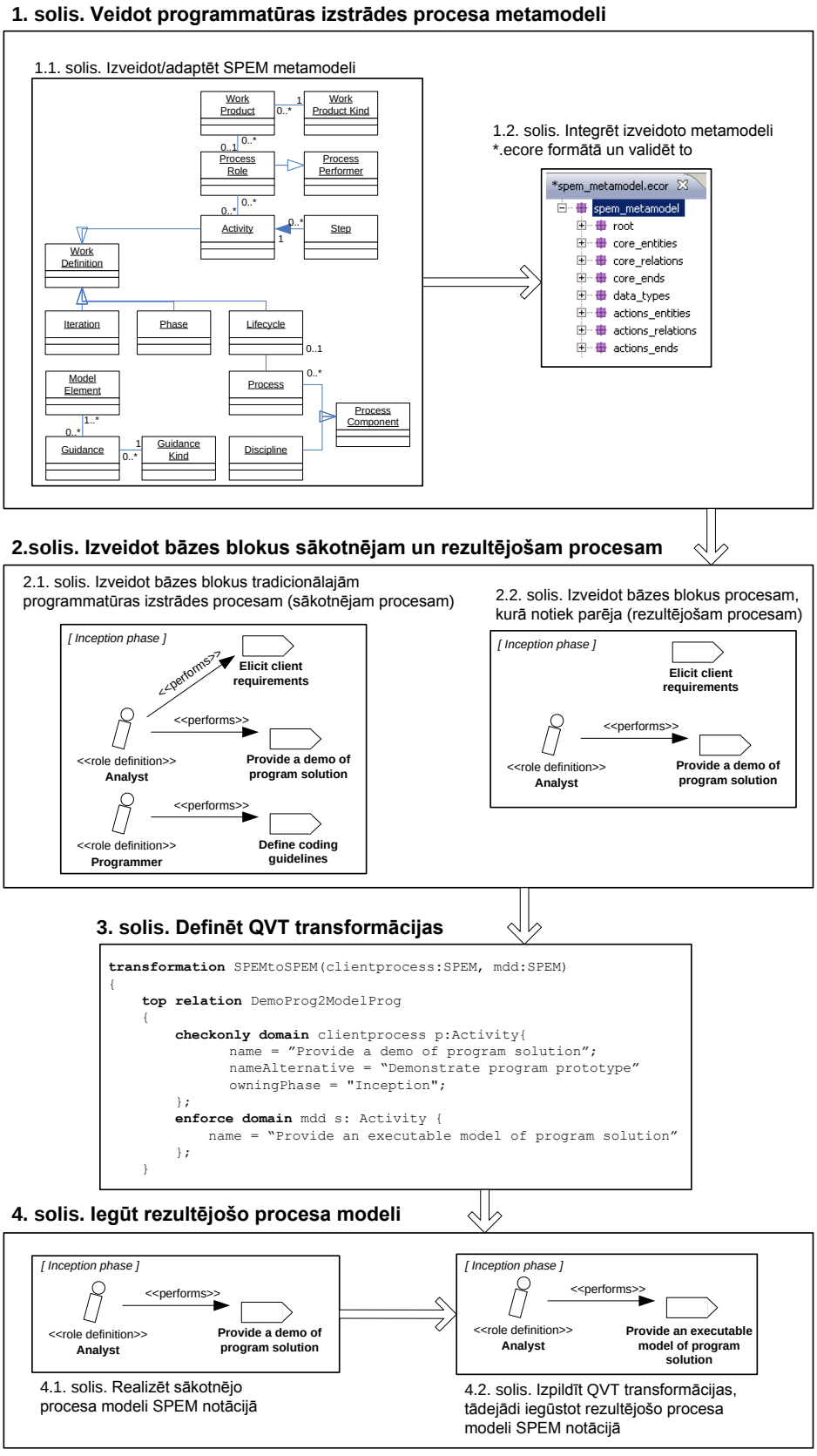
Izvirzītā uzdevuma risināšanai ir nepieciešamas gan teorētiskās zināšanas par izstrādes procesa specifiku, gan tehnoloģiskais atbalsts SPEM modeļu formālajam pierakstam un starpmodeļu transformāciju realizēšanai.

Piedāvātais risinājums ļauj aprakstīt avota modelī jebkuru tradicionālo programmatūras izstrādes procesu (piem., RUP, MSF, SCRUM), kas piemīt izvēlētajai organizācijai ar savu specifiku, un, ar MDA transformāciju palīdzību, iegūt modeļvadāmo programmatūras izstrādes procesu jeb mērķa modeli, tādejādi bagātinot esošo programmatūras izstrādes procesu ar modeļvadāmas izstrādes artefaktiem. Šī pieeja pilnībā atbilst MDA četru līmeņu arhitektūrai, jo reālais programmatūras izstrādes process tiek formalizēts SPEM modelī, kas atbilst SPEM metamodelim, savukārt, SPEM metamodelis ir aprakstīts ar MOF metametamodeli.

4. PROGRAMMATŪRAS IZSTRĀDES PROCESA TRANFORMĀCIJAS APRAKSTS

Eksistējošie MDA risinājumi darbam ar modeļiem ļauj pielietot šos pašus risinājumus arī citā kontekstā, aizvietojot biznesa modeļus ar programmatūras izstrādes dzīves cikla modeļiem. Vadoties pēc MDA karkasa, pirmais solis ir SPEM metamodeļa izveidošana vai eksistējoša SPEM metamodeļa pielāgošana (sk. 4. attēlu). SPEM ir UML profils, kas ir definēts ar MOF palīdzību. SPEM procesu metamodelis ļauj attēlot jebkuru programmatūras izstrādes procesu [SPE 2008]. Risinājuma koncepcijas realizēšanai, SPEM ir speciāli ierobežots ar dažiem būtiskākiem programmatūras izstrādes dzīves cikla konceptiem, lai padarītu modeļus pēc iespējas vienkāršākus un samazinātu transformāciju skaitu. Ja nepieciešams, ir iespējams definēt papildus elementus un papildus transformācijas, jo tas neietekmē kopējo risinājuma koncepciju, bet gan papildina to ar papildus izstrādātiem artefaktiem.

Kā risinājuma arhitektūru šai MDA bāzētai pieejai, darba autors piedāvā izmantot atklātā pirmkoda (angl. *open source*) platformu Eclipse un EMF (angl. *Eclipse Modeling Framework*) datu modelēšanas un integrēšanas karkasu [NIK 2010c].



4. att. Algoritma kopēja shēma

EMF pēc būtības atbilst vienam no MOF attīstības zariem – EMOF (angl. *Essential MOF*). EMF arhitektūras ietvaros Eclipse metamodeļi tiek saglabāti ECore formātā. Papildus ECore metamodelēšanas valodai, EMF nodrošina arī koda ģenerēšanas karkasu, kas ļauj ģenerēt Java kodu no ECore modeļiem. ECore metamodeļa klašu nosaukumi ir tuvi UML terminoloģijai, jo pamatā ECore ir neliela UML apakškopa, ierobežota ar specifisku pielietošanu EMF ietvaros.

4.2. Pirmais solis. Veidot programmatūras izstrādes procesa metamodeli

Pirmā soļa rezultāts ir izstrādāts SPEM metamodelis, kas ir aprakstīts izmantojot ECore metamodeli. Metodoloģijas elementu standartizācija metamodeļos programminženierijas kontekstā atbalsta programmatūras izstrādes procesu un ar to pārstāvēto metodoloģiju, jo metamodelis ir nepieciešams šo procesu modeļu eksemplāru veidošanai [HUG 2009]. Izmantojot šo metamodeli, tiek nodrošināta iespēja ģenerēt SPEM modeļus programmatūras izstrādes dzīves cikla formalizēšanai.

4.3. Otrais solis. Veidot bāzes blokus

Otrajā solī ir nepieciešams izveidot bāzes blokus sākotnējam un rezultējošam procesam. Šī promocijas darba kontekstā procesa bāzes bloks ir programmatūras izstrādes procesa sastāvdaļa, kas apvieno kādu specifisku procesa aktivitātes kopu transformācijas kontekstā. Veicot dekompozīciju programmatūras izstrādes dzīves ciklam, iespējams iegūt procesa bāzes bloku kopu. Un otrādi, programmatūras procesu var uzskatīt par savstarpēji saistītu bāzes bloku kopu [NIK 2010b], [NIK 2011]. Par sākotnējo procesu tiek uzskatīts avota process, kas atbilst organizācijā pieņemtajam procesam, t.i. tradicionālajām programmatūras izstrādes procesam. Par rezultējošo procesu tiek uzskatīts mērķa process, uz kuru notiek pāreja, t.i. MDA bāzēts process.

Programmatūras izstrādes procesi katrā organizācijā ir unikāli. Turklāt pat vienai un tai pašai procesa sastāvdaļai var lietot dažādas definīcijas. Tomēr tiem piemīt standarta aktivitātes, kuras ir iespējams sastādīt no iepriekš definētiem procesu bāzes blokiem [Ņikuļšins 2006]. Šī pieeja ir realizēta vairākos rīkos – Eclipse EPF (angl. *Eclipse Process Framework*), IBM Rational Method Composer, Enterprise Architect, MagicDraw UML (SPEM Package) un Objecteering SPEM Modeler [NIK 2010c].

Otrā soļa rezultāts ir abu procesu (gan sākotnējā, gan rezultējošā) vienotas bāzes bloku kopas, kas raksturo, attiecīgi, tradicionālo programmatūras izstrādes procesu un MDA bāzēto procesu.

4.4. Trešais solis. Definēt QVT transformācijas

Trešais solis ir transformāciju definīcija. Transformāciju definīciju pierakstam ir paredzēts izmantot vienu no QVT valodām – QVT Relations valodu. QVT Relations ir OMG standarta deklaratīva valoda, paredzēta no modeļa uz modeli (angl. *model-to-model*) transformācijām. MDA ietvaros šai valodai ir definēta gan tekstuālā sintakse, gan grafiskais pieraksts [MOF 2008], [RED 2006].

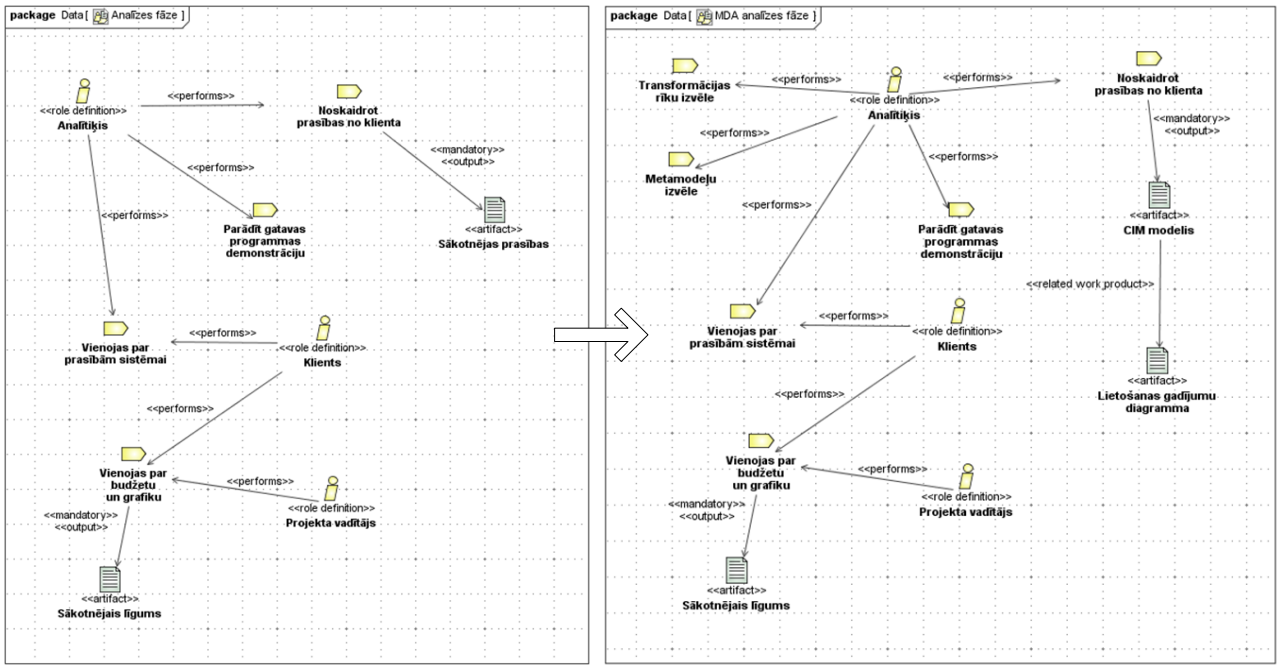
Transformācijas definīcijai ir nepieciešama teorētiskā bāze, proti, sākotnējā procesa elementu kartēšana uz rezultējošā procesa elementiem [NIK 2009a]. Kā jau tika minēts otrā soļa aprakstā, transformācijas tiek definētas procesa bāzes bloku kopai. Transformāciju definīcijai ir veikta abu procesu detalizēta analīze, ar mērķi identificēt modeļvadāmas programmatūras izstrādes ietekmi uz tradicionālo programmatūras izstrādes procesu.

4.5. Ceturtais solis. Iegūt rezultējošo procesa modeli

Ceturtais solis pēc būtības ir izstrādātās pieejas praktiska pielietošana. Izmantojot programmatūras izstrādes procesa bāzes blokus, nepieciešams izveidot SPEM modeli, kas atbilst tradicionālajam programmatūras izstrādes procesam pētāmajā organizācijā. Šim SPEM modelim ir jāatbilst ECore metamodelim. Izpildot QVT Relations transformācijas, sākotnējais modelis tiek transformēts rezultējošā, kurš ir bagātināts ar modeļvadāmas pieejas artefaktiem un ir ceturtā soļa rezultāts.

5. PIEDĀVĀTAS PIEEJAS PRAKTISKĀ LIETOŠANA

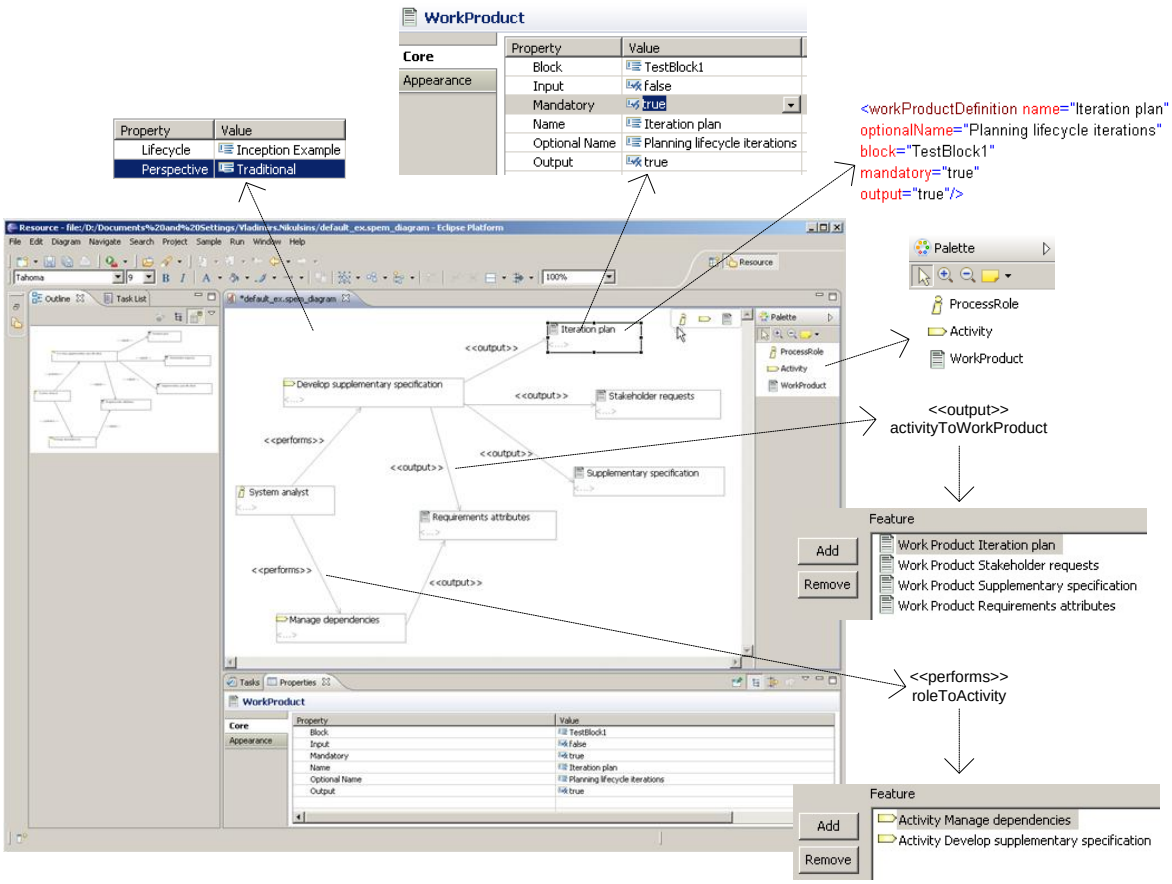
Autora piedāvāta pieeja, kas ir detalizēti aprakstīta darba 4. nodaļā, ir praktiski pielietota programmatūras izstrādes uzņēmumā „PF”, ar mērķi pārveidot uzņēmumā izmantoto programmatūras izstrādes procesa organizāciju modeļvadāmajā [NIK 2010b]. Uzņēmumā eksistējošam programmatūras izstrādes procesam ir veidots procesa modelis, kam tika pielietota autora definēta pieeja un iegūts rezultējošais procesa modelis, bagātināts ar modeļvadāmas programmatūras izstrādes artefaktiem [NIK 2009e]. Tā fragments ir parādīts 5. att.



5. att. Sākotnēja un attiecīga rezultējoša procesu modeļu fragmentu demonstrācija

Autora piedāvātās pieejas atbalstam ir izstrādāts rīka prototips, kas dod iespēju automatizēt piedāvātās pieejas praktisko pielietošanu (sk. 6. att.). Šis prototips ir izstrādāts, izmantojot Eclipse GMF (angl. *Graphical Modeling Framework*) tehnoloģiju, kas ir paredzēta grafisko redaktoru izstrādei modeļvadāmās pieejas kontekstā. Transformācijas ir uzdotas ar citu Eclipse spraudni (angl. *plug-in*) mediniQVT.

Šis prototips var tikt ņemts par pamatu pilnvērtīga spraudņa veidošanai programmatūras izstrādes procesa transformāciju realizācijā.



6. att. RUP aktivitāšu grupas „definēt sistēmu" bāzes bloks (promocijas darba autora izstrādātajā rīka prototipā)

DARBA REZULTĀTI UN SECINĀJUMI

Attīstoties programmatūras izstrādes procesam, programminženierija atrodas tādā kā pārejas periodā, kurā tradicionālie jeb uz kodu orientētā programmatūras izstrāde vairs nevar pilnībā tikt galā ar pieaugošo programmatūras sistēmu sarežģītību. Līdz ar to ir parādījusies pakāpeniska tendence pāriet uz modeļvadāmu programmatūras izstrādes procesa organizāciju. Tāpēc kļūst aktuālas šādu pāreju metodiskas pieejas, kas nodefinē vadlīnijas tradicionālā programmatūras izstrādes procesa transformācijai modeļvadāmajā.

Pētījuma rezultātā ir izstrādāta pieeja programmatūras izstrādes procesa pārejai no tradicionālās procesa organizācijas uz modeļvadāmo, par pamatu izmantojot procesa modelēšanas paņēmienus un modeļu formālas transformācijas principus, kas ir definēti modeļvadāmas arhitektūras ietvarā, un šādas pārejas tehnoloģiskajam atbalstam pielietojot esošus rīkus, kas atbalsta modeļvadāmo pieeju. Lai pārbaudītu pieejas lietderību, tā tika praktiski aprobēta vienā no Latvijas programmatūras izstrādes uzņēmumiem, kurā tika izanalizēts uzņēmuma esošais programmatūras izstrādes process un tika izveidots procesa modelis. Savukārt, lai nodrošinātu pāreju uz modeļvadāmu programmatūras izstrādes procesu, izstrādātajam procesu modelim tika pielietotas autora definētās transformācijas. Izvirzītā mērķa sasniegšanai tika izpildīti šādi **uzdevumi**:

- aprakstīti tradicionālā programmatūras izstrādes procesa pamatprincipi, to noteicošie standarti un zināšanu korpuss SWEBOK [SWE 2004];
- izklāstīti modeļvadāmas izstrādes galvenie koncepti, proti, kas ir modelis, modeļu transformācijas un metamodelēšana, kā arī ir izpētītas iespējas izmantot modeļvadāmas programmatūras izstrādes konceptus programmatūras izstrādē;
- izpētītas tradicionālā programmatūras izstrādes procesa integrācijas iespējas ar modeļvadāmas arhitektūras artefaktiem;
- piedāvāta jauna pieeja pārējai no tradicionālās programmatūras izstrādes MDA orientētajā, kas balstās uz procesu modelēšanas formālajiem pamatiem, modeļu transformācijas likumiem un esošajiem tehniskajiem līdzekļiem, kas atbalsta modeļvadāmo arhitektūru;
- izstrādātā pieeja tika pielietota informācijas tehnoloģijas uzņēmumā, tajā esošā programmatūras izstrādes procesa modernizācijai, piedāvājot ar MDA artefaktiem bagātināta procesu modeļa ieviešanu;
- izteikti secinājumi par MDA lietošanas iespējām, problēmām un perspektīvām programmatūras izstrādē, kā arī diskutēti aspekti pārejai no tradicionālā programmatūras izstrādes procesa, uz modeļvadāmo.

Promocijas darba galvenais rezultāts *ir izstrādāta programmatūras izstrādes procesa pārejas pieeja, kas piedāvā uz kodu orientēto programmatūras izstrādi aizvietot ar modeļvadāmas programmatūras izstrādes artefaktiem. Definētā pieeja balstās uz formāliem procesu modelēšanas principiem, transformācijas likumu definēšanas teoriju, un izmanto eksistējošos rīkus, kas atbalsta procesu modelēšanu un modeļu transformācijas. Izstrādāts piedāvātās pieejas atbalsta rīka prototips.*

Promocijas darba ***svarīgākie rezultāti***:

1. Sistematizēta un izklāstīta informācija par programmatūras izstrādes procesu vēsturi, terminoloģiju, programmatūras izstrādes metodoloģijām, kā arī ir uzskaitīti ietekmīgākie programmatūras izstrādes standarti un zināšanu korpusi.
2. Demonstrēta procesu modeļu un modeļu transformācijas lietošanas iespējamība programmatūras izstrādes procesu apstrādē un pārvaldībā.
3. Definēta modeļvadāmas programmatūras izstrādes terminoloģija un modeļvadāmas izstrādes artefaktu kartēšana programmatūras izstrādes dzīves cikla fāzēs un aktivitātēs.
4. Definēti transformācijas likumi, kurus ir iespējams izmantot, pārveidojot programmatūras izstrādes procesa aktivitātes, bagātinot tās ar modeļvadāmas izstrādes artefaktiem.

5. Definēts algoritms tradicionālā programmatūras izstrādes procesa pārveidošanai modeļvadāmajā, kuru modificētā veidā ir iespējams izmantot kāda noteikta programmatūras izstrādes procesa pārveidošanai citā procesā (piem., no MSF uz RUP).
6. Balstoties uz programmatūras izstrādes procesa analīzi informācijas tehnoloģijas uzņēmumā, izstrādāta ieteikumu kopa uzņēmumā izmantotā procesa bagātināšanai ar modeļvadāmas programmatūras izstrādes artefaktiem.

Promocijas darba ietvaros veiktie pētījumi un iegūto rezultātu analīze, ļauj izteikt šādus secinājumus:

- Programminženierijas disciplīnu, kā vienu no inženierijas nozarēm joprojām ir sarežģīti formalizēt. It īpaši tas attiecas uz programmatūras izstrādes procesu, kur īpaša loma ir atvēlēta cilvēciskajam faktoram. Tomēr viens no programmatūras izstrādes procesa formalizācijas paņēmieniem šāda procesa analīzei un apstrādei ir izmantot modelēšanas iespējas.
- Procesu modeļu transformācijas ir sevišķi aktuāls uzdevums smagsvara metodoloģijās un liela izmēra programmatūras izstrādes procesos, salīdzinājumā ar triviālo programmatūras izstrādes projektu realizāciju, kam pārsvarā tiek lietotas spējas programmatūras izstrādes metodes.
- Izstrādes procesu modelēšanai eksistē vairākas notācijas, piemēram, programmatūras izstrādes specifiku ir iespējams attēlot ar t.s. SPEM notāciju. Tomēr, SPEM notācijā nav iebūvētu mehānismu, lai veiktu procesu modeļu pārveidošanu, pārejot no vienas programmatūras izstrādes organizācijas citā. Tomēr darba autoram ir izdevies noteikt veidu, kā tieši pārejas uzdevuma realizācijā, iespējams strādāt arī ar SPEM notācijā attēlotiem modeļiem.
- MDA piedāvā iespējas, lai izvairītos no platformas specifikas, līdz ar to paaugstinot abstrakcijas līmeni programmatūras izstrādē. Tomēr MDA atbalstošo rīku daudzums apmulsina šo rīku potenciālos lietotājus, it īpaši ņemot vērā līdz šim nepietiekami izstrādātos artefaktu apmaiņas mehānismus dažādu rīku savstarpējā integrācijā.
- Vieni no pamatelementiem, ko piedāvā darba autors savā pieejā, ir SPEM notācijas bāzes bloku izmantošana, jo ar tiem var uzdot dažādus abstrakcijas līmeņus, tādā veidā attēlojot programmatūras izstrādes procesa aktivitātes un darbplūsmas. Tieši uz bāzes blokiem var balstīt programmatūras izstrādes procesa zināšanu formalizāciju, piemēram, meklējot dažādos programmatūras izstrādes procesos līdzīgus, bet dažādi nosauktus artefaktus.
- Novērojot programmatūras izstrādes procesu attīstību modeļvadāmas arhitektūras atzīšanas virzienā, jāsaka, ka pašlaik vēl nav pieejami pietiekami kvalitatīvi instrumenti, pilnvērtīgai modeļvadāmas izstrādes paņēmienu lietošanai programmatūras industrijā. Tomēr promocijas darba autora piedāvātais risinājums noteikti ir viens no soļiem programmatūras izstrādes procesu evolūcijā, tieši modeļvadāmas izstrādes virzienā.

Turpmākie pētījumu virzieni var būt saistīti ar:

- procesu modeļa atbalsta rīka prototipa realizāciju pilnvērtīgā vidē;
- SPEM notācijas bagātināšanu ar jauniem elementiem, kas nepieciešami procesu modeļu transformāciju uzdevumu risināšanai.

Izstrādāto metodi autors iesaka pielietot ar programmatūras izstrādi saistītajos uzņēmumos, tajos esošo izstrādes procesu modernizācijai. Kā arī liela mēroga informācijas tehnoloģiju kompānijās, kur programmatūras sistēmu izstrāde tiek organizēta programmatūras fabriku (angl. *software factories*) veidā, un līdz ar to ir aktuāla vairāku programmatūras sistēmu izstrāde pēc tipveida procesa organizācijas. Šāda tipa uzņēmuma piemērs ir kompānija *Accenture*, kurā pašlaik strādā promocijas darba autors. Šajā uzņēmumā sākotnējā programmatūras izstrādes procesa modelēšana jau tiek izmantota dažādos pārvaldības uzdevumos, bet joprojām pastāv nepieciešamība pēc tehnoloģijas, kas nodrošinātu programmatūras izstrādes procesu apstrādi to transformācijas mērķiem.

IZMANTOTĀ LITERATŪRA

Autora publikācijas starptautiski recenzējamos zinātniskajos krājumos:

[NIK 2006b] Nikulsins V,. Nikiforova O., Sukovskis U. Analysis of Activities Covered by Software Engineering Discipline // Databases and Information Systems, Seventh International Baltic Conference on Databases and Information Systems, Communications, Materials of Doctoral Consortium, O. Vasilecas, J. Eder, A. Caplinskas (Eds.), pp. 130-138, VGTU Press „Technika" scientific book No 1290, Vilnius, Lithuania – 2006. – pp. 130–138

[NIK 2006c] Nikulsins, V., Nikiforova, O. Review on Allocation of Roles and Responsibilities among Software Development Team // The 46th Scientific Conference of Riga Technical University, Computer Science, Applied Computer Systems, October 13-14, Riga, Latvia, 2005, published in the 5th Series Computer Science. Applied Computer Systems, – 2006. – Vol. 26 – pp. 54–65

[NIK 2008a] Nikulsins, V., Nikiforova, O., Sukovskis, U. Mapping of MDA Models into the Software Development Process // Databases and Information Systems, Proceedings of the Eighth International Baltic Conference Baltic DB&IS 2008, H.-M. Haav and A. Kalja (Eds.), Tallinn University of Technology Press, Tallinn, Estonia, June 2-5, – 2008. – pp. 217–226

[NIK 2008b] Nikulsins V., Nikiforova O. Adapting Software Development Process towards the Model Driven Architecture // Proceedings of The Third International Conference on Software Engineering Advances (ICSEA), International Workshop on Enterprise Information Systems (ENTISY), 2008, Mannaert H., Dini C., Ohta T., Pellerin R. (Eds.), Sliema, Malta, October 26-31, 2008., Published by IEEE Computer Society, Conference Proceedings Services (CPS) – 2008. – pp. 394–399 (available at IEEE Xplore Digital Library; ACM Digital Library)

[NIK 2008c] Nikiforova O., Sukovskis U., Nikulsins V. Principles of Model Driven Architecture for the Task of Study Program Development // Joining Forces in Engineering Education Towards Excellence Proceedings, SEFI Annual Conference, 2008, F.K. Fink (Ed.), CD-ROM of papers – 2008. – paper ID 1162, pp 1–8. (available at SEFI Digital Library)

[NIK 2009a] Nikulsins, V., Nikiforova, O. Transformations of SPEM Models Using Query/View/Transformation Language to Support Adoption of Model-driven Software Development Lifecycle // The 13th East-European Conference on Advances in Databases and Information Systems (ADBIS), September 2009. Riga, Latvia, JUMI Publishing House Ltd. – 2010. – pp. 416–423

[NIK 2009c] Nikiforova O., Nikulsins V., Sukovskis U. Integration of MDA Framework into the Model of Traditional Software Development // In the series "Frontiers in Artificial Intelligence and Applications", Databases and Information Systems V, Selected Papers from the Eighth International Baltic Conference Baltic DB&IS 2008, Haav H.-M., Kalja A. (Eds.), IOS Press, - 2009. – Nr. 187. – pp. 229–242 (available at Google Books; IOS Press)

[NIK 2010a] Nikulsins V. Transformations of Software Process Models to Adopt Model-Driven Architecture // International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2010), Proceedings of the 2nd International Workshop on Model Driven Architecture and Modeling Theory Driven Development (MDA&MTDD 2010), Osis J., Nikiforova O. (Eds.), Greece, Athens, July 2010, SciTePress, Portugal. – 2010. – pp. 70–79 (available at Thomson Reuters; Inspec; dblp.uni-trier.de Computer Science bibliography)

[NIK 2010b] Nikulsins V., Nikiforova O., Kornijenko J. SPEM model transformations to adopt MDA in practice. Databases and Information Systems // Proceedings of the Ninth International Baltic Conference Baltic DB&IS 2010. Databases and Information Systems, 2010. Barzdins J., Kirikova M. (Eds.). 2010. July, Latvia, Riga. Riga: University of Latvia Press, Riga, Latvia, – 2010. – pp. 295–307

[NIK 2010c] Nikulsins V., Nikiforova O. Tool Integration to Support SPEM Model Transformations in Eclipse // Scientific Journal of Riga Technical University, Computer Science,

Applied Computer Systems. The 50th International Scientific Conference of Riga Technical University, Riga, Latvia: RTU Publishing, – 2010. – Vol. 43. – pp. 60–67 (available at EBSCO)

[NIK 2011] Nikulsins V., Nikiforova O., Kornijenko J. An Approach for Enacting Software Development Process: SPEM4MDA // Frontiers in Artificial Intelligence and Applications, Vol. 224, Databases and Information Systems VI – Selected Papers from the 9th International Baltic Conference, DB&IS 2010. Barzdins J., Kirikova M. (Eds.). Amsterdam: IOS Press, – 2011. Vol. 224. – pp. 55–65 (available at IOS Press; ACM Digital Library)

Autora zinātniskā projekta atskaite:

[NIK 2009e] Ņikiforova O., Ņikuļšins V., Korņijenko J. u.c. Modeļvadāma programmatūras izstrādes procesa ieviešana kompānijā „PF", RTU pētniecības projekta Nr. FLPP-2009/10 „Konceptuālā modeļa izstrāde pārejai no tradicionālās programmatūras izstrādes MDA orientētajā" atskaite. – 2009.

Citi kopsavilkumā izmantotie avoti:

[BEL 2002] Belaunde M. Initial identification of issues for further research. Deliverable 2.2. MODA-TEL. Interactive Objects Software GmbH – 2002. / Internets: http://www.modatel.org/~Modatel/pub/deliverables/D2.2-final.pdf

[BOT 2010] Botteri P., Cowan D., Deeter B. u.c. Bessemer's Top 10 Laws of Cloud Computing and SaaS. Bessemer Venture Partners – 2010. / Internets: http://www.bvp.com/downloads/saas/BVPs_10_Laws_of_Cloud_SaaS_Winter_2010_Release.pdf

[BRE 2001] Breton E., Bezivin J. Process-Centered Model Engineering. Fifth IEEE International Enterprise Distributed Object Computing Conference, 2001. – France: IEEE – 2001. / Internets: http://www.sciences.univ-nantes.fr/lina/atl/www/papers/edoc.pdf

[BRO 2005a] Brown A., Conallen J., Tropeano D. Models, Modeling, and Model Driven Development // Model-Driven Software Development, Springer, Berlin – 2005. – pp. 1–17.

[BRO 2005b] Brown A., Conallen J. An introduction to model-driven architecture. Part III: How MDA affects the iterative development process – May 2005. / Internets: http://www.ibm.com/developerworks/rational/library/may05/brown/index.html

[CAU] Cauna E. Lielā terminu vārdnīca / Internets: www.termini.lv

[CHI 2007] Chitforoush F., Yazdandoost M., Ramsin R. Methodology Support for the Model Driven Architecture // Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. 14th Asia-Pacific Software Engineering Conference APSEC 2007. – 2007.

[CMM] Software Engineering Institute. Capability Maturity Model Integration (CMMI) / Internets: http://www.sei.cmu.edu/cmmi/

[COC 1999] Cockburn A. A Methodology Per Project. / Internets: http://alistair.cockburn.us/crystal/articles/mpp/methodologyperproject.html

[COM 2006] Combemale B., Cregut, X. Towards a Rigorous Process Modeling With SPEM. – France: Rennes University – 2006. / Internets: http://www.combemale.net/research/phd/2006/iceis250406-CCCC-poster401.pdf.

[DEB 2007] Debnath, N., Zorzan, F.A., Montejano u.c. Transformation of BPMN subprocesses based in SPEM using QVT // Electro/Information Technology, 2007 IEEE International Conference. – 2007. – pp. 146–151

[DIA 2010] Diaw S., Lbath R., Thai V. u.c. SPEM4MDE: a Metamodel for MDE Software Processes Modeling and Enactment // Third Workshop on Model-Driven Tool & Process Integration. MDTPI 2010, Paris, France. – 2010. – pp. 109-121.

[ERI 2004] Eriksson H.-E., Penker M., Lyons B. u.c. UML2 Toolkit. Wiley Publishing, Indianapolis, Indiana, USA. – 2004. – 511 p.

[ESI 2002] Steinhau R. Model Driven Architecture Definition and Methodology. Deliverable 3.1. MODA-TEL. Interactive Objects Software GmbH – 2002 / Internets: http://www.modatel.org/public/deliverables/D3.1.htm

[ESI 2003] Process Model to Engineer and Manage the MDA: Model-driven Architecture inSTrumentation, Enhancement and Refinement Approach. European Software Institute – 2003 / Internets: http://modeldrivenarchitecture.esi.es/pdf/Deliverable-D32.zip

[ESS 2009] Essvale Corporation Limited. Business Knowledge for IT in Insurance: A Complete Handbook for IT Professionals. 63 Apollo Building, 1 Newton Place, London E14 3TS. Essvale Corporation – 2009. – 248 p.

[EVA 2003] Evans A., Sammut P., Willans J.S. (eds.) Metamodelling for MDA // First International Workshop Proceedings, York, UK – November 2003. / Internets: http://www.cs.york.ac.uk/metamodel4mda/onlineProceedingsFinal.pdf

[FAV 2010] Favre L. Model Driven Architecture for Reverse Engineering Technologies: Strategic Directions and System Evolution. 701 E. Chocolate Avenue, Hershey PA 17033, United States of America: IGI Global publishing – 2010. – 459 p.

[FEN 2006] Feng Y., Mingshu L., Zhigang, W. SPEM2XPDL: Towards SPEM Model Enactment. Bejing, China: The Chinese Academy of Sciences – 2006 – 6 p.

[FOR 2009] Forrester C., Buteau E. L., Shrum, S. B. CMMI for Services, Version 1.2. Technical Report, Software Engineering Institute – 2009.

[GAV 2004] Gavras A., Belaunde M., Pires L.F. u.c. Towards an MDA-based development methodology for distributed applications – 2004. / Internets: http://eprints.eemcs.utwente.nl/7100/01/paper3-3.pdf

[GUT 2004] Guttman M. Getting Started With OMG's MDA. Application Development. 2004. / Internets: http://www.softwaremag.com/L.cfm?doc=2004-09/2004-09mda

[GUT 2007] Guttman, M., Parodi, J. Real-Life MDA. Solving Business Problems With Model Driven Architecture. Morgan Kaufmann Publishers – 2007.

[HUG 2009] Hug C., Front A., Rieu D. u.c. A method to build information systems engineering process metamodels. Journal of Systems and Software. Volume 82, Issue 10. – 2009. – pp. 1730-1742.

[HUS 2011] Hussman H., Meixner G., Zuehlke D. (Eds.). Model-Driven Development of Advanced User Interfaces. Studies in Computational Intelligence, First Edition. – Berlin, Heidelberg, Germany: Springer-Verlag – 2011. – 324 p.

[IEE 1990] IEEE Std 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology. The Institute of Electrical and Electronics Engineers – 345 East 47th Street, New York, NY 10017, USA: IEEE Standards Board. – 1990. – 84 p. / Internets: http://www.idi.ntnu.no/grupper/su/publ/ese/ieee-se-glossary-610.12-1990.pdf

[IEE 2008] IEEE Std 12207-2008. ISO/IEC 12207. IEEE Systems and software engineering - Software life cycle processes. The Institute of Electrical and Electronics Engineers. IEEE Standards Activities Department - 445 Hoes Lane, Piscataway, NJ 08854, USA: IEEE Standards Board. – 2008. – 122 p. / Internets: http://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-2008.pdf

[ISO 2005] ISO/IEC 19502:2005 International Standard. Information Technology – Meta Object Facility (MOF). – Switzerland, Geneva. – 2005.

[JAY 2007] Jayaswal B.K., Patton P.C. Design for Trustworthy Software: Tools, Techniques, and Methodology of Developing Robust Software. First Edition. Prentice Hall, 2007. 840 p. http://ptgmedia.pearsoncmg.com/images/0131872508/samplechapter/0131872508_ch01.pdf

[KHA 2004] Balbo S., Khan A. A Tale of two Methodologies: Heavyweight versus Agile // Department of Information Systems, The University of Melbourne. AusWeb04. The Tenth Australian World Wide Web Conference. – 2004 / Internets: http://ausweb.scu.edu.au/aw04/papers/edited/balbo/

[KLE 2003] Kleppe A., Warmer J., Bast W. MDA Explained: The Model Driven Architecture: Practice and Promise. Addison Wesley – April 2003. – 167 p.

[KRU 2000] Kruchten P. The Rational Unified Process An Introduction, Second Edition, Addison Wesley – 2000. – 320 p.

[LAN 2004] Langlois B., Exertier D. MDSoFa: a Model-Driven Software Factory. Thales Research & Technology France. – October 2004. / Internets: www.softmetaware.com/oopsla2004/langlois.pdf

[MAN 2006] Mansell J., Bediaga A., Vogel R. u.c. Process Framework for the Successful Adoption of Model Driven Development. A. Rensink and J. Warmer (Eds.): ECMDA-FA 2006, LNCS 4066. – 2006. – pp. 90–100.

[MEL 2002] Mellor S. J., Balcer J. M. Executable UML: A Foundation for Model-Driven Architecture. Addison Wesley. – 2002. 416. p.

[MEL 2004] Mellor S.J., Scott K., Uhl A. u.c. MDA Distilled: Principles of Model-Driven Architecture. Addison Wesley – 2004. - 176 p.

[MIC 2002] Microsoft. Microsoft Solutions Framework: MSF Process Model v. 3.1. White Paper, Microsoft Corporation – June 2002.

[MIC 2003] 2710B: Analyzing Reuirements and Defining Microsoft .NET Solution Architectures. Microsoft Official Course. – April 2003.

[MOF 2008] Meta Object Facility (MOF) 2.0 Query/View/Transformation, v1.0, 2008 / Internets: http://www.omg.org/docs/formal/08-04-03.pdf

[NAU 1969] Naur P., Randell B. (Eds.) Software Engineering. Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11th of October 1968. / Internets: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.pdf

[OMG 2006] Object Constraint Language Specification, version 2.0. Prepared by OMG: OMG – 2006 / Internets: http://www.omg.org/cgi-bin/apps/doc?formal/06-05-01.pdf

[OMG] OMG Model Driven Architecture. / Internets: http://www.omg.org/mda

[OSI 2010] Osis J., Asnina E. Model-Driven Domain Analysis and Software Development: Architectures and Functions. IGI Publishing. – 2010. – 487 p.

[PAL 2006] Palomäki J., Keto H. A Process-Ontological Model for Software Engineering // Philisophiocal Foundations on Information Systems Engineering. Tampere University of Technology/Pori. Department of Information Technology / Internets: http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-240/paper3.pdf

[PEI 2010] Peixoto D. C. C., Batista V. A., Resende R. F. u.c. How to Welcome Software Process Improvement and Avoid Resistance to Change // Federal University of Minas Gerais, Brazil – 2010. / Internets: http://homepages.dcc.ufmg.br/~vitor/artigos/icsp2010_paper.pdf

[PIL 2005] Pilone D., Pitman N. UML 2.0 in a Nutshell. First Edition. – Sebastopol, USA. O'Reilly Media Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. – 2005. – 240 p.

[RAT 1998] Rational Software Corporation. Rational Unified Process. Best Practices for Software Development Teams // Rational Software White Paper. – 1998. – 21 p.

[RED 2006] Reddy S., Venkatesh R., Ansari Z. A relational approach to model transformation using QVT Relations // Tata Research Development and Design Centre, Pune, India. – 2006. / Internets: http://www.iist.unu.edu/~vs/wiki-files/QVT-TRDCC.pdf

[RIO 2006] Rios E., Bozheva T., Bediaga A. u.c. MDD Maturity Model: A Roadmap for Introducing Model-Driven Development // European Conference on Model Driven Architecture – Foundations and Applications, Rensink A., Warmer J. (Eds.), ECMDA-FA 2006 – Bilbao, Spain: Springer – 2006. – pp. 78-89.

[ROY 1970] Royce W. Managing the Development of Large Software Systems. IEEE WESCON, 1970. / Internets: http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf

[SCA 2001] Scacchi W. Process Models in Software Engineering. Institute for Software Research, University of California, Irvine – 2001. – 24 p. / Internets: http://www.ics.uci.edu/~wscacchi/Papers/SE-Encyc/Process-Models-SE-Encyc.pdf

[SER 2005] Serour M.K., Henderson-Sellers B. Resistance to Adoption of an OO Software Engineering Process: An Empirical Study // University of Technology, Sydney, Australia. European and Mediterranean Conference on Information Systems (EMCIS) –2005.

[SIE 2001] Siegel J., OMG Staff Strategy Group. Developing in OMG's Model-Driven Architecture. Object Management Group White Paper. – 2001.

[SPE 2008] Software Process Engineering Metamodel Specification (SPEM), Version 2.0, OMG specification. – 2008. / Internets. – http://www.omg.org/cgi-bin/doc?formal/2008-04-01

[STA 2006] Stahl T., Voelter M. Model-Driven Software Development: Technology, Engineering, Management. John Wiley & Sons, Ltd., 2006. 428 p.

[STE 1999] Stelzer D., Mellis W. Success Factors of Organizational Change in Software Process Improvement // Software Process Improvement and Practice, Volume 4, Issue 4. John Wiley & Sons Ltd –1999. / Internets: http://informationsmanagement.wirtschaft.tu-ilmenau.de/forschung/documents/successf.pdf

[STE 2003] Steinhau R. Guidelines for the Application of MDA and the Technologies covered by it. Deliverable 3.2. MODA-TEL. Interactive Objects Software GmbH – 2003. / Internets: http://www.modatel.org/~Modatel/pub/deliverables/D3.2-final.pdf

[SWE 2004] Abran A., Moore J.W., Bourque P. u.c. (Eds.) Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE. – Los Alamitos, California: IEEE Computer Society Press – 2004. – 302 p.

[THA 2005] Thayer R.H., Christensen M.J. (Eds.) Software Engineering, Volume 1: The Development Process, Third Edition – 11 River Street, Hoboken, NJ 07030, ASV: John Wiley & Sons. – 2005. – 540 p.

[TRA 2004] Tratt L. Model transformations and tool integration. – London, UK: Department of Computer Science, King's College London, Springer-Verlag – 2004. –12 p.

[TSU 2011] Tsui F., Karam O. Essentials of Software Engineering, Second Edition. Southern Polytechnic State University, Marietta, Georgia, USA. John and Bartlett Publishers, 40 Tall Pine Drive, Sudbury, MA 01776. – 2011. – 409 p.

[VLI 2008] Vliet H. Software Engineering: Principles and Practice, Third Edition. The Atrium, Southern Gate, Chester, West Sussex PO19 8SQ, England – John Wiley & Sons. – 2008. – 713 p.

[VOG 2006] Vogel R., Mantell K. MDA adoption for a SME: evolution, not revolution Phase II. // The Second European Conference on Model Driven Architecture (ECMDA 2006). Foundations and Applications. Bilbao, Spain. – 2006. – 13 p.

[WAK 2001] Wake W. C. Extreme Programming Explored. The XP series, Addison-Wesley Professional – 2001. – 192 p.