RĪGAS TEHNISKĀ UNIVERSITĀTE

PROMOCIJAS DARBS

2016

RĪGAS TEHNISKĀ UNIVERSITĀTE

Datorzinātnes un informācijas tehnoloģijas fakultāte

Lietišķo datorsistēmu institūts

Gusts Linkevičs

Doktora studiju programmas „Datorsistēmas" doktorants

Spējās paradigmas ieviešanas atbalsts programmatūras izstrādes organizācijās

Promocijas darbs

Zinātniskais vadītājs

Profesors *Dr. sc. ing.*

U. SUKOVSKIS

Rīga 2016

Anotācija

Prasības pret programmatūru nemitīgi attīstās, un programmizstrādes uzņēmumi cenšas sekot līdzi programmatūras izstrādes metožu attīstībai. Izstrādes metodes, kuras tika izmantotas projektu realizēšanai pirms vairākiem gadu desmitiem vairs nespēj nodrošināt moderno projektu izstrādi tādā līmenī, lai projekts pilnībā atbilstu augošajām prasībām. Uzņēmumi regulāri saskārās ar problēmām, kad projekti tiek izstrādāti pārāk ilgi un vairs neatbilst pasūtītāju prasībām. Problēma kļuva tik aktuāla, ka 2001. gada februārī, 17 programmizstrādes speciālisti izstrādāja "Spējās programmatūras izstrādes manifestu", kurā definēja 12 pamatprincipus, kādiem būtu jāatbilst jaunai spējai (angl. *agile*) izstrādes metodei. Ņemot vērā šos 12 pamatprincipus, programmizstrādes procesam bija jākļūst vieglākam, saprotamākam un pieejamākam. Kā augstākā prioritāte tika izvirzīts mērķis, ka pasūtītājam ir jābūt apmierinātam ar izstrādāto produktu un produkts ir jāpiegādā pēc iespējas agrāk un nepārtraukti. Šie pamatprincipi definēja, kādam ir jābūt programmizstrādes procesam, bet nenoteica kā tieši to panākt.

Ņemot vērā, ka nav noteikta veida kā panākt programmizstrādes uzņēmuma pārveidošanu par uzņēmumu, kurš izmanto spējās metodes, tad katrs uzņēmums izvēlas savu ceļu, kā pāriet no tradicionālā izstrādes modeļa uz spējo izstrādes modeli. Pārejai no tradicionālā izstrādes modeļa uz spējo izstrādes modeli ir nepieciešama uzņēmuma transformēšanās, kas ietver daudzus būtiskus aspektus un pamatprocesus. Pārejas procesa ilgums katrā uzņēmumā atšķiras, un to ietekmē dažādi faktori, tādi kā uzņēmuma lielums, projekta specifika u. c. Ņemot vērā, ka uzņēmumu pieejas ļoti atšķiras un uzņēmumi dažādi interpretē literatūrā iegūto informāciju, tad uzņēmumiem ir sarežģīti un dažreiz pat neiespējami noteikt, cik tālu ar transformācijas procesu uzņēmums ir ticis un cik veiksmīgs tas ir bijis.

Līdz šim nav izstrādāts veids kā veiksmīgi noteikt programmatūras izstrādes organizācijas spējības līmeni, lai nepieciešamības gadījumā to varētu uzlabot. Problēmas atrisināšanai ir izveidota organizācijas domēnu spējības noteikšanas metode (ODSN), kuras izmantošana ļauj noteikt esošo programmatūras izstrāds organizācijas spējības līmeni, tādā veidā palīdzot sagatavot spējības uzlabošanas plānu, kura realizācijas rezultātu var atkārtoti izvērtēt, lai veidotu nākamo uzlabojumu plānu. Šāda iteratīva pieeja sniedz iespēju regulāri novērtēt programmatūras izstrādes organizācijas spējības līmeni. Metode ODSN izmanto organizācijas spējības modeli (OSM), kuru ir vērtējuši spējo metožu eksperti (SME), kā arī informāciju par organizāciju, kuru regulāros intervālos sniedz uzņēmuma darbinieki.

Par promocijas darba būtiskākajiem rezultātiem ir nolasīti 5 referāti starptautiskās konferencēs, kā arī tie ir atspoguļoti 5 zinātniskajos rakstos.

Promocijas darbs satur ievadu, 5 nodaļas, secinājumus, 6 pielikumus, literatūras sarakstu (123 avoti), 63 attēlus un 123 tabulas, kopā 227 lappuses.

Annotation

Software development similarly as requirements for software are constantly evolving and software development companies are trying to keep up with software development methods. Development methods used decades ago can`t handle modern development projects at level, which is required by constantly increasing requirements of the customer. Projects in the development often had problems with delivering on time and delivering the product, which corresponds to customer requirements. Problem become so urgent, that in February of 2001, 17 software development gurus created the "Agile Software Development Manifesto" with 12 fundamental principles, to which the new method should correspond. Having regard to the 12 development principles, development should become easier, clearer and more accessible. As a top priority, the goal was that the customer must be satisfied with the developed product and that valuable product is to be delivered as soon as possible and continuously. Twelve principles defined how agile software development process should look like, but did not specify precisely how to achieve it.

Given that there is no certain way exactly to transform software development company into company, which uses agile methods and thus becomes agile, then each company chooses its own path of transformation from traditional software development to agile software development model. The duration of the transformation process in each company is different and it is affected by a variety of factors such as the size of the company, the project, etc. Given that the company approaches are very different and companies interpret the information in literature differently, then it is hard or sometimes even impossible for the company to determine how far the company is with transformation process and how successful is the process.

Until now, there has been no way to successfully identify software development company's agility level so that this can, if necessary, be improved. To solve the problem, the method Organization domain agility (ODA) is created. To some extent, the use of this method may help to determine current agility level of the software development company, thus helping prepare agility improvement plan. After execution of the improvement plan, agility level can be accessed again and new improvement plan can be created. Such iterative approach gives an opportunity to assess the level of organisation agility on regular basis. The ODA method uses Organization agility model (OAM), which has been evaluated by agile method experts (AME) and information about particular organization, which in regular intervals is given by employees of the organization.

Main results of the thesis have been presented at 5 international conferences, as well as they are reflected in 5 scientific papers.

The thesis includes introduction, 5 chapters, and conclusion. It contains 227 pages, 63 figures and 123 tables, 123 titles large bibliography and 6 appendixes.

Saturs

Attēlu sarakts

Tabulu saraksts

Abreviatūru saraksts

Nr.	Saīsinājums	Izvērsums/Paskaidrojums
1	AI	Aktualitātes indeks
2	ATDD	Acceptance Test Driven Development
3	BD	Burndown Chart
4	BDD	Behavior Driven Development
5	CSS	Cascading Style Sheets
6	DAA	Domēni apakšdomēni un atribūti
7	DSDM	Dynamic systems development method
8	DVV	Darbinieku vērtēšanas vērtība
9	FDD	Feature-Driven Development
10	FL	Feature List
11	FOIL	First-Order Inductive Learner
12	HTML	HyperText Markup Language
13	INVEST	Independent, Negotiable, Valuable, Estimatable, Small, Testable
14	KD	Kvalitātes domēns
15	KII	Komandas ietekmes indekss
16	MoSCoW	Must have, Should have, Could have, Won`t have
17	OD	Organizācijas domēns
18	ODSN	Organizācijas domēnu spējības noteikšanas metode
19	OMD	Overall Model Development
20	OSM	Organizācijas spējības modelis
21	PB	Product Backlog
22	PD	Izstrādes domēns
23	PII	Projekta ietekmes indekss
24	PO	Product Owner
25	PRD	Procesu domēns
26	PRJD	Projektu domēns
27	RBD	Release Burndown Chart
28	ROI	Return on Investment
29	RP	Release Plan
30	SB	Sprint Backlog
31	SII	Spējības ietekmes indekss
32	SME	Spējo metožu eksperti
33	STDD	Story Test Driven Development
34	TDD	Test Driven Development
35	UI	User Interface
36	URL	Uniform Resource Locator
37	US	User Stories
38	VD	Vērtību domēns
39	WIP	Work in Progress
40	XP	Extreme programming

Apzīmējumu saraksts

Nr.	Apzīmējums	Vien.	Paskaidrojums
1	$A_{1...n}$	4,9	Darbinieka jautājumu apakškopa, kur *n* ir kopējais darbinieku skaits, kuri piedalās anketēšanā
2	A_m	1	Rakstu skaits, kuru kontekstā ir runa par konkrēto metodi
3	A_y	1	Kopējais rakstu skaits konkrētajā gadā
4	*EG*	2	Ekspertu grupa
5	*Foil_Gain(L,R)*	13	*FOIL* novērtēšanas funkcija
6	IOA_m	1	Metodes AI
7	*K*	2	Eksperti, kuri piedalījušies spējo projektu realizēšanā (parasti spējās komandas sastāvā).
8	L_1 ... L_n	12	Literas, kuras veido konkrētā likuma priekšnosacījumus
9	*M*	2	Eksperti, ar plašām zināšanām par spējām metodēm. Spējās metodes izmanto ikdienā dažādu projektu realizēšanai. Ir pieredzējuši vairāku organizāciju un komandu pārveidošanos uz spējo metožu izmantošanu programmatūras izstrādē
10	*n*	10	Darbinieku skaits, kuri snieguši atbildi par konkrēto atribūtu.
11	n_0	13	Negatīvo sasaišu skaits likumā *R*
12	n_1	13	Negatīvo sasaišu skaits likumā R^1
13	N_n	5	Neatbildētie jautājumi sakārtoti pēc SII, kur *n* ir konkrētais darbinieks. Pēc prioritāro jautājumu pievienošanas kopai, kopai tiek pievienoti jautājumi ar augstu SII vērtību uz kuriem darbinieks vēl nav atbildējis. Šie jautājumi procentuāli aizņem visvairāk un sastāda 60% no jautājumu kopas
14	O_n	5,8	Iepriekš atbildētie jautājumi sakārtoti pēc SII, kur *n* ir konkrētais darbinieks. Eksistē būtiski jautājumi, uz kuriem ir nepieciešams atbildēt biežāk nekā citiem, tādēļ jautājumu kopai visu laiku tiek pievienoti būtiski jautājumi, uz kuriem jau ir atbildēts iepriekš. Šie jautājumi sastāda atlikušos 20% no jautājumu kopas.
15	*P*	5,7	Prioritāro jautājumu kopa – intervēšanas iniciators pirms intervēšanas atzīmē ļoti svarīgus jautājumus, uz kuriem vēlas iegūt atbildi pēc iespējas ātrāk. Prioritārie jautājumi tiek pievienoti visām darbinieku jautājumu kopām, un sastāda 20% no uzģenerētās jautājumu kopas
16	$P(x_1, x_2,....., x_k)$	12	Litera kas veido likuma galvu
17	p_0	13	Pozitīvo sasaišu skaits likumā *R*
18	p_1	13	Pozitīvo sasaišu skaits likumā R^1
19	*Q*	3,4	Visu jautājumu kopa
20	$q_{1...m}$	3	Jautājumi, kur m ir kopējais jautājumu skaits
21	*S*	2	Eksperti, kuri pilda vai ir pildījuši *ScrumMaster* lomu
22	*t*	13	Pozitīvo sasaišu skaits likumā *R,* kuras ir nosegtas pēc literas *L* pievienošanas likumam *R*
23	w_i	11	Atribūta vai apakšdomēna SII vērtība (svars)
24	x_i	11	Atribūta vai apakšdomēna DVV vērtība
25	x_i	10	Atribūta vērtējums, ko sniedzis darbinieks *i*
26	$\ddot{x}$	10	Atribūta DVV vērtība.
27	$\bar{y}$	11	Domēna, apakšdomēna DVV vērtība.
28	*Sk*	14	Saskaņotības līmenis

Ievads

Šodienas uzņēmējdarbības vide ir ļoti dinamiska, un pasūtītāji ir spiesti bieži mainīt prasības programmatūrai, lai tā atbilstu jauniem nosacījumiem. Pasūtītāji pieprasa biežas un ātras programmatūras papildinājumu piegādes un vēlas, lai būtu iespējams jebkurā laikā veikt izmaiņas prasībās. Tradicionālās programmatūras izstrādes metodes nespēj nodrošināt nepieciešamo elastīgumu, un ir radusies vajadzība pēc atšķirīgām izstrādes metodēm, kuras spētu apmierināt pasūtītāju un arī izstrādātāju augošās prasības.

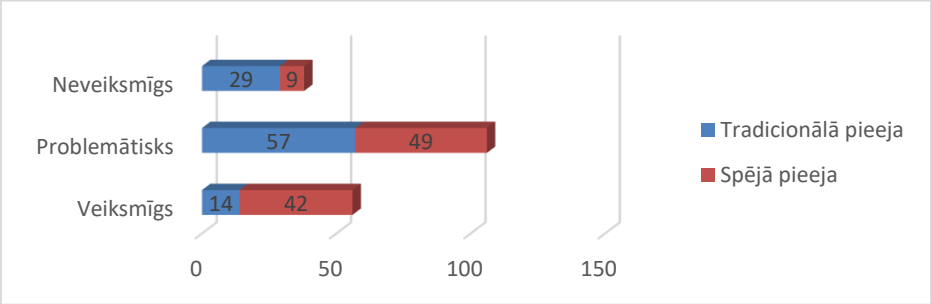
Spējās programmizstrādes pirmsākumi ir atrodami jau 1957. gadā. Pieeja ir minēta *Craig Larman* un *Victor Basili* rakstā [112], kurā viņi citējuši *Gerald M. Weinberg*: "Mēs veicām papildinošu izstrādi jau 1957. gadā, Losandželosā, *Bernie Dimsdale* vadībā strādājot "IBM Service Bureau Corporation". *Bernie Dimsdale* bija *John von Neumann* kolēģis, varbūt viņš to iemācījās tur, vai varbūt viņi to vienkārši uzskatīja par pašsaprotamu. Es atceros, kā mēs un *Herb Jacobs* izstrādājām lielu simulāciju priekš "Motorola", kur mēs izmantojām šo tehniku, vismaz cik es varu spriest … Mēs visi, cik es atceros, uzskatījām, ka liela projekta veidošana, izmantojot ūdenskrituma modeli, ir diezgan muļķīga vai vismaz atrauta no realitātes. Es domāju, ka tas, kas bija aprakstīts "Ūdenskrituma" modelī, noteikti nebija tas, ko darām mēs un ka mēs izmantojam kādu vēl vārdā nenosauktu programmizstrādes metodi" (autora tulkojums).

1970. gadā *Dr. Winstons Royce* prezentē savu rakstu "Lielu programmizstrādes projektu vadība" (angl. *Managing the Development of Large Software Systems*)[106], kurā kritizē secīgu programmizstrādi un papildina, ka programmatūras izstrādi nevar salīdzināt un veikt līdzīgi kā mašīnas izstrādi uz konveijera. Tajā pašā gadā *E. A. Edmonds* ir sagatavojis rakstu "Programmatūras izstrādes process netehniskiem lietotājiem kā adaptīva sistēma" (angl. *A process for the Development of Software for Nontechnical Users as an Adaptive System*), kuru gribēja publicēt žurnālā "Computer Aided Design", bet raksta publicēšana tika atteikta ar komentāriem: "Ja jūs nezināt, ko darīsiet, pirms uzsākat darbu, jums nevajadzētu to uzsākt". Raksts gan tiek publicēts vēlāk 1974. gadā, žurnālā "General Systems" [113].

Neskatoties uz sākotnējo pretestību un nevēlēšanos pieņemt spējās metodes kā alternatīvu, to attīstība turpinājās, un paralēli attīstījās dažādas spējās metodes,

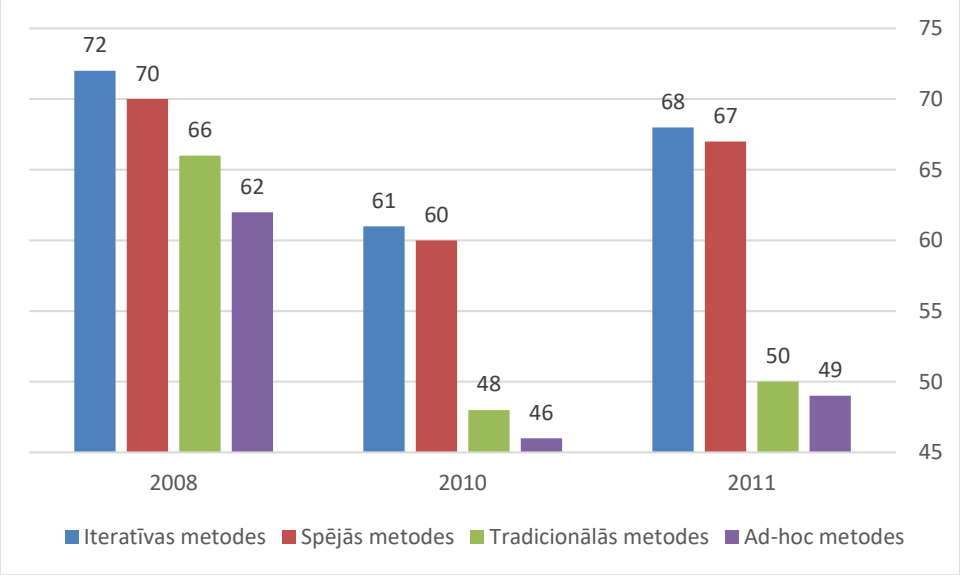
piemēram, *Scrum* (1986) [113], *Extreme Programming* (1999) [19] u. c. Pagrieziena punkts spējo metožu attīstībai ir metožu autoru un ekspertu satikšanās un spējās programmizstrādes manifesta izstrāde 2001. gadā [7], kad speciālisti izstrādāja kopējo spējo metožu pamatprincipu sarakstu.

Spējās metodes piesaista arvien lielāku interesentu loku un tagad dažāda lieluma programmizstrādes uzņēmumi pilnībā vai daļēji pāriet uz spējo izstrādes metožu izmantošanu. Par pamatu šai interesei kalpo lielāks veiksmīgo projektu skaits, ko apliecina arī 2012. gadā veiktais *Standish Group* pētījums [109], kur par veiksmīgiem tiek atzīti 42% no izstrādātajiem spējiem projektiem un tikai 14 % no tradicionālā veidā izstrādātajiem (sk. 0.1. att.).



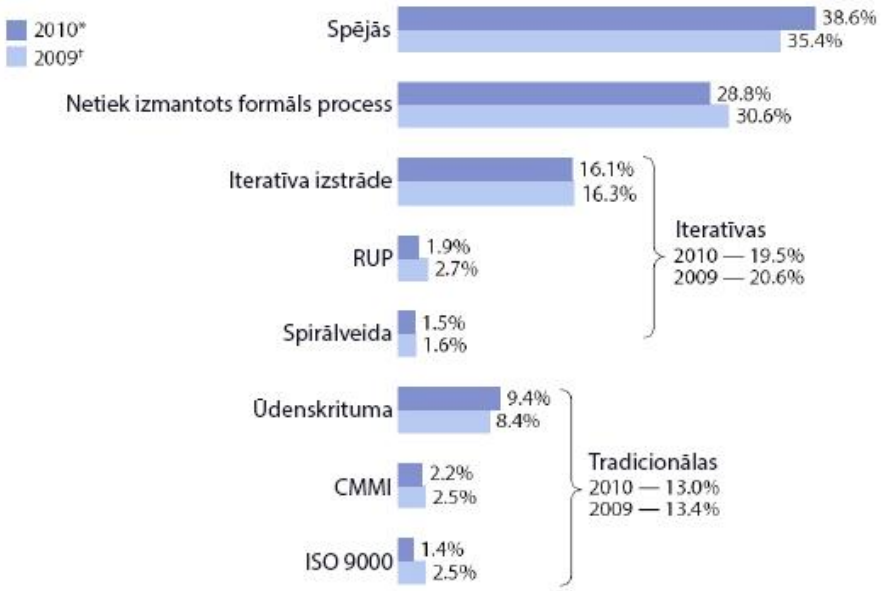
0.1. att. *Standish Group* pētījuma rezultāti 2012. gadā.

Līdzīgas tendences par spējo projektu realizēšanu savā pētījumā ir apkopojis arī *Scott Ambler* no *AmbySoft* [110], kur par veiksmīgiem tiek uzskatīti 68 % iteratīvi realizēti projekti un 67 % spējā veidā realizēti projekti, un tikai puse no tradicionālā veidā realizētajiem projektiem (sk. 0.2. att.).



0.2. att. Veiksmīga projektu realizācija proc. balstoties uz *AmbySoft* pētījumiem.

Forrester Research pētījums [111] parāda, ka spējā veidā realizēto projektu skaits pieaug no 35,4 % 2009. gadā līdz 38,6 % 2010. gadā. Tradicionālā veidā realizēto projektu skaits nedaudz samazinās no 13,4 % uz 13,0 % (sk. 0.3. att.).



0.3. att. *Forrester Research* pētījuma rezultāti par spējo metožu izmantošanu.

Promocijas darbā analizēto pētījumu rezultāti parāda, ka spējo metožu izmantošana ir pieaugusi attiecībā pret tradicionālo metožu izmantošanu un ka spējo projektu realizācija ir bijusi veiksmīgāka, bet norāda arī uz to, ka daļa projektu, izmantojot spējās metodes, tomēr nav bijusi veiksmīga. Pēc *Standish Group* pētījumu

rezultātiem (sk. 0.1. att.) 9 % no spējiem projektiem nav bijuši veiksmīgi un 49 % gadījumu ir bijušas zināmas problēmas.

Tēmas aktualitāte un motivācija

Katra uzņēmuma motivācija ieviest spējo paradigmu var atšķirties, bet pārsvarā tā ir vēlme savā portfelī iegūt pēc iespējas vairāk veiksmīgu programmizstrādes projektu. Daļai uzņēmumu izdodas veiksmīgi transformēties par uzņēmumiem, kuri izmanto spējās metodes, bet ir arī uzņēmumi, kuros transformācijas process nav bijis veiksmīgs.

Ir nepieciešams veids, kā palīdzēt šiem uzņēmumiem sekmīgi un ātri transformēties par programmizstrādes uzņēmumiem, kuri ieviesuši spējo paradigmu, lai uzlabotu programmatūras izstrādes projektu realizāciju.

Viens no nozīmīgākajiem motivatoriem promocijas darba izstrādei ir autora personīgā pieredze organizāciju transformācijā uz spējo metožu izmantošanu. Ar problēmas pētīšanu autors nodarbojas jau sešus gadus. Ar līdzīgām problēmām saskaras daudzi uzņēmumi, un šī tendence atspoguļojas konferencēs par spējām metodēm un to izmantošanu [1] [2][3][4][5].

Kopumā šā darba autors ir strādājis piecos uzņēmumos, kuri ir mēģinājuši ieviest spējās metodes. Par veiksmīgu autors uzskata tikai vienu no šīm transformācijām. Tieši neveiksmīgo transformāciju daudzums ir mudinājis autoru meklēt risinājumu šai problēmai.

Jēdzienam **organizācijas spējība** darba kontekstā ir sašaurināta nozīme un tas ir attiecināms tikai uz programmatūras izstrādes uzņēmumu spēju veikt programmatūras izstrādi, izmantojot spējās metodes.

Promocijas darba mērķis

Promocijas darba mērķis ir izstrādāt metodi un rīku spējās paradigmas ieviešanas atbalstam programmatūras izstrādes organizācijās.

Izvirzītais mērķis pamatojas uz šādām **tēzēm**:

- organizāciju neziņa par spējām metodēm rada problēmas spējo metožu izmantošanā;
- izmantojot metodes un rīkus, ir iespējams noteikt organizācijas spējību;

- organizācijas spējības noteikšana palīdz izstrādāt atbilstošu uzlabojumu plānu organizācijas spējības uzlabošanai.

Darba uzdevumi

Mērķa sasniegšanai ir izvirzīti šādi uzdevumi:

- izpētīt spējo metožu un prakšu priekšrocības un trūkumus, kā arī to nozīmīgākās iezīmes;
- izstrādāt terminu vārdnīcu biežāk izmantotajiem terminiem, lai būtu iespējams definēt aktualitātes indeksu (AI) un noteikt tā vērtību spējām metodēm, praksēm un atslēgas vārdiem;
- izveidot organizācijas spējības definīciju un izstrādāt organizācijas spējības modeli (OSM), kā arī noteikt, kuras prakses ietekmē atbilstošos OSM elementus;
- definēt spējības ietekmes indeksu (SII) un noteikt SII vērtības OSM elementiem;
- izstrādāt organizācijas domēnu spējības noteikšanas metodi (ODSN) un tās darbības nodrošināšanai nepieciešamo jautājumu ģenerēšanas algoritmu;
- aprobēt *FOIL* (angl. *First-Order Inductive Learner*) algoritmu spējības noteikšanas pēdējā posmā;
- veikt metodes un izstrādātā rīka aprobāciju uzņēmumos spējības noteikšanai.

Pētījumu objekti

Promocijas darba pētījumu objekts ir spējo metožu izmantošana programmatūras izstrādes uzņēmumos.

Pētījumu priekšmets

Promocijas darba pētījumu priekšmets ir organizācijas spējības noteikšanas metodes un rīka izstrāde.

Pētījuma metodes

Spējo metožu priekšrocību un trūkumu, kā arī metožu nozīmīgāko iezīmju noteikšanai ir izmantota literatūras avotu analīze.

Terminu vārdnīcas veidošanā ir izmantota literatūras avotu analīze un "Prāta karte" jeb prātojumu shēma (angl. *Mind Map*) terminu atveidošanai.

AI noteikšanai ir veikts konferenču materiālu pētījums laika periodā no 2008. līdz 2012. gadam. Pētījumā ir apkopota informācija par metodēm, praksēm un atslēgas vārdiem, kuri pieminēti konferencēs noteiktā laika periodā. Informācija par metodēm, praksēm un atslēgas vārdiem tika saglabāta speciāli šim nolūkam autora izveidotā datu bāzē.

OSM definēšanai ir izmantota literatūras avotu analīze un autora personīgā pieredze spējo projektu realizācijā. OSM modeļa validācijai tiek izmantots spējo metožu ekspertu (SME) tīkls.

Lai noteiktu OSM elementu spējības ietekmes indeksu (SII) vērtības, tiek izmantots iepriekš izveidotais spējot metožu ekspertu (SME) tīkls un ekspertu aptaujas metode DELPHI.

ODSN metodes darbības rezultāts tiek pārbaudīts, veicot trīs programmizstrādes uzņēmumu spējības līmeņa noteikšanu. Uzņēmumi ir dažāda lieluma un strādā ar dažāda veida projektiem, kā arī viens no uzņēmumiem spējo metodi izmanto tikai viena projekta realizēšanai.

Darba zinātniskais jaunieguvums ir šāds:

- pamatojoties uz terminoloģijas kartes izveidi, ir izdevies precīzāk klasificēt spējās metodēs izmantotos terminus un izveidot AI definīciju, kā arī noteikt spējo metožu, prakšu un atslēgas vārdu AI;
- organizācijas spējības noteikšanai ir definēts termins organizācijas spējība, kuras noteikšanai ir izstrādāts organizācijas spējības modelis (OSM) un spējības noteikšanas metode ODSN;
- izmantojot ekspertu tīklu, ir noteiktas SII vērtības katram OSM elementam, kā arī ir definēts SII jēdziens un izstrādāts domēnu, apakšdomēnu un atribūtu (DAA) vērtību koks SII vērtību attēlošanai;

- metode ODSN spējības noteikšanai ievāc informāciju no darbiniekiem. Lai nepārslogotu darbiniekus ar lielu jautājumu skaitu, ir izveidots jautājumu ģenerēšanas algoritms, kurš katram darbiniekam nodrošina nelielu jautājumu kopu katrā intervēšanas reizē.

Pētījumu praktiskā nozīmība

Promocijas darba praktiskā nozīme ir iespēja palīdzēt programmizstrādes uzņēmumiem sekmīgi transformēties no tradicionālā programmatūras izstrādes modeļa uz spējo izstrādes modeli, noteikt problemātiskos posmus programmatūras izstrādes procesā un organizācijas pieejā, izmantojot izstrādāto metodi ODSN, kura balstās uz izstrādāto OSM modeli un jautājumu ģenerēšanas algoritmu, kuru izmantošana ir realizēta izstrādātajā spējības noteikšanas programmatūras prototipā.

Darba aprobācija

Promocijas darbā veikto pētījumu rezultāti ir atspoguļoti 5 publikācijās.

1. Linkevičs, G., Adopting to Agile Software Development, volume 16 "Applied Computer Systems", 2014. – Rīga, RTU, 2014. – 64.-71. lpp. (EBSCO).
2. Linkevičs, G., Sukovskis, U., Evaluation of the Agility Level of the Organization, volume 18 "Applied Computer Systems", 2015. – Rīga, RTU, 2015. – 21.-26. lpp. (DEGRUYTER).
3. Linkevičs, G., Evaluation of Agility in Software Development Company // Proceedings of the Joint International Conference on Engineering Education & International Conference on Information Technology (ICEE/ICIT 2014), Latvija, Rīga, 2.-6. jūnijs, 2014. - 320.-332. lpp.
4. Linkevičs, G., Sukovskis, U., Determining agility impact index and generating employee based questions to assess organizational agility // Proceedings of the International Conference on Engineering Education (ICEE 2015), Horvātija, Zagreba, 20.-24. jūlijs, 2015.
5. Linkevičs, G., Sukovskis, U., Using ODA Method and *FOIL* Algorithm to Determine Organizational Agility Level // Proceedings of the 10th International Multi-Conference on Computing in the Global Information Technology (ICCGI 2015), Malta, St. Julians, 11.-16. oktobris, 2015. -93.-100. lpp.

Pētijumos iegūtie rezultāti tika prezentēti piecās konferencēs.

1. Linkevičs, G., Adopting to Agile Software Development. RTU 53. Starptautiskā zinātniskā konference, Rīga, Latvija, 11.-12. oktobris, 2012.
2. Linkevičs, G., Sukovskis, U., Evaluation of the Agility Level of the Organization. RTU 56. Starptautiskā zinātniskā konference, Rīga, Latvija, 14.-16. oktobris, 2015.
3. Linkevičs, G., Evaluation of Agility in Software Development Company. Joint International Conference on Engineering Education & International Conference on Information Technology (ICEE/ICIT 2014), Rīga, Latvija, 2.-6. jūnijs, 2014.
4. Linkevičs, G., Sukovskis, U., Determining agility impact index and generating employee based questions to assess organizational agility. International Conference on Engineering Education (ICEE 2015), Horvātija, Zagreba, 20.-24. jūlijs, 2015.
5. Linkevičs, G., Sukovskis, U., Using ODA Method and *FOIL* Algorithm to Determine Organizational Agility Level. 10th International Multi-Conference on Computing in the Global Information Technology (ICCGI 2015), Malta, St. Julians, 11.-16. oktobris, 2015.

Darba struktūra

Promocijas darbs sastāv no ievada, piecām nodaļām, secinājumiem, bibliogrāfijas un sešiem pielikumiem.

Darba ievadā ir pamatota tēmas aktualitāte un motivācija, izvirzīti darba mērķi un uzdevumi, aprakstītas pētījuma metodes, darba zinātniskais jaunieguvums un darba praktiskā nozīme. Ievada beigās ir informācija par darba aprobāciju un darba struktūru.

Promocijas darba 1. nodaļā ir apskatīta spējā metodoloģija, tās priekšrocības un trūkumi. Pētītas spējās metodoloģijas terminoloģijas problēmas un definēti darbā izmantotie spējās metodoloģijas termini. Liela uzmanība ir pievērsta spējo metožu, prakšu un atslēgas vārdu aktualitātes indeksa (AI) noteikšanai. Nodaļas beigās ir sniegta dinamiskas vides definīcija.

Promocijas darba 2. nodaļā ir apskatītas spējās metodes un prakses ar augstāko AI, lai izprastu, kā darbojas spējās metodes un kādas ir to kopīgās un atšķirīgās

raksturiezīmes. Apskatītā informācija tiek izmantota, lai izstrādātu organizācijas spējības modeli (OSM).

Promocijas darba 3. nodaļa fokusējas uz organizācijas spējību. Nodaļā ir izstrādāta organizācijas spējības definīcija un izstrādāts OSM, kas sevī iekļauj vērtējamos domēnus, apakšdomēnus un atribūtus. OSM sastāv no sešiem augstākā līmeņa domēniem: organizācijas domēna, izstrādes domēna, kvalitātes domēna, procesu domēna, vērtību domēna un projektu domēna. Šajā nodaļā ir izstrādāta spējības ietekmes indeksa (SII) definīcija un detalizēts apraksts SII noteikšanai, izmantojot ekspertu tīklu un metodi DELPHI.

Promocijas darba 4. nodaļā ir definēts spējības noteikšanas konceptuālais modelis un organizācijas domēnu spējības noteikšanas metode ODSN. Izstrādāts metodes process un jautājumu kopas ģenerēšanas algoritms, kā arī domēnu apakšdomēnu un atribūtu (DAA) koks, kurš ir nepieciešams SII vērtību un darbinieku vērtēšanas vērtību (DVV) attēlošanai. Ir aprakstīta *FOIL* algoritma izmantošana DAA koka apstrādei un organizācijas spējības noteikšanai, kā arī ir izstrādātas spējības noteikšanas rīka datu plūsmas un arhitektūra.

Promocijas darba 5. nodaļā ir apskatīta ODSN metodes aprobācija trijos dažādos uzņēmumos, nosakot šo uzņēmumu spējības līmeni.

Katras nodaļas secinājumi ir apkopoti nodaļas beigās, noslēguma secinājumi un rezultāti ir apkopoti darba noslēguma sadaļā.

Promocijas darbam ir seši pielikumi. Informācija par spējām metodēm un to AI ir apkopota 1. pielikumā. Informācija par konferencēs izmantoto atslēgas vārdu AI ir apkopota 2. pielikumā. Darba 3. pielikumā ir informācija par spējām praksēm. *FOIL* algoritma veiktā likumu ģenerēšanas iterācija spējības klasēm C_2 un C_3 dota 4. pielikumā. Darba 5. pielikumā ir apkopota informācija ar DAA SII vērtībām pēc spējo ekspertu vērtējuma, bet 6. pielikumā ir metodes ODSN aprobācijas rezultāti, kuros ir atspoguļoti trīs pārbaudes organizāciju rādītāji.

1. Spējā programmatūras izstrāde

Nav vienas konkrētas spējās programmatūras izstrādes definīcijas, dažādos avotos ir pieejami dažādi termina skaidrojumi. Dažos avotos [103][106] tiek minēts, ka spējā programmatūras izstrāde ir metodoloģija, citos [104][105] - ka tas ir kopējs termins spējo metožu un prakšu apvienošanai. Septiņos no astoņiem avotiem par pamata atsauci tiek izmantots spējās programmatūras izstrādes manifests [7]. Darbā vēlētos pievienot trīs definīcijas, kuras pēc maniem uzskatiem vislābāk atspoguļo spējo programmatūras izstrādi:

- *Spējā programmatūras izstrāde ir metodoloģija radošam procesam, kurš paredz nepieciešamību pēc elastīguma un piemēro zināma līmeņa pragmatismu gatava produkta piegādē. Spējā programmatūras izstrāde fokusējas uz vienkāršu kodu, biežu testēšanu un nelielas programmatūras funkcionalitātes nodošanu, tiklīdz tā ir gatava. Spējās programmizstrādes mērķis ir izstrādāt, balstoties uz nelielām pasūtītāja apstiprinātām programmatūras daļām projekta izstrādes laikā, pretstatā tam, lai piegādātu lielu programmatūru projekta beigās [103].*
- *Spējā programmatūras izstrāde ir visaptverošs apzīmējums metožu un prakšu kopai, kuras balstītas uz vērtībām un pamatprincipiem, kuri aprakstīti spējās programmatūras izstrādes manifestā [104].*
- *Spējā programmatūras izstrāde ir programmatūras izstrādes metode, kura balstās uz cilvēkiem un ir komunikāciju orientēta. Tā ir elastīga (gatava pielāgoties, gaidāmajām izmaiņām, jebkurā laikā), ātra (strauja iteratīva izstrāde ar nelieliem laidieniem), plāna (angl. lean) (fokusējas uz saīsinātiem intervāliem un izmaksām, un uzlabotu kvalitāti), reaģējoša (reaģē atbilstoši uz zināmām un nezināmām izmaiņām) un mācoša (fokusējās uz uzlabojumiem projekta izstrādes laikā un pēc tā) [107].*

Par iteratīvās programmizstrādes sākumu tiek uzskatīts 1957. gads, bet popularitātes sākums ir 1990. gadi, kad „vieglās metodes" tiek izveidotas kā pretstats „smagajām metodēm", piemēram, ūdenskrituma modelim [6].

Spējās programmatūras izstrādes manifests tika izveidots 2001. gada 11-13. februārī, kad 17 speciālisti no *Extreme Programming, Scrum, DSDM, Adaptive Software Development, Crystal, Feature-Driven Development, Pragmatic*

Programming un citiem novirzieniem vienojās par spējās metodoloģijas manifestu un 12 spējās metodoloģijas pamatprincipiem [7]:

- mūsu augstākā prioritāte ir apmierināt pasūtītāju izmantojot agras un regulāras vērtīgas programmatūras piegādes;
- mainīgās prasības tiek pieņemtas, pat tad ja tās ir vēlu izstrādes procesā. Pieņemt izmaiņas vēlu izstrādes procesā, lai uzlabotu klienta konkurētspēju;
- piegādāt strādājošu programmatūru regulāri, no pāris nedēļām līdz pāris mēnešiem, priekšroka tiek dota īsākiem periodiem;
- biznesa cilvēkiem un izstrādātājiem ir jāstrādā kopā ik dienu visa projekta garumā;
- veidojiet projektu, izmantojot motivētus indivīdus. Sniedziet viņiem nepieciešamo vidi un atbalstu un uzticieties viņiem, ka darbs tiks padarīts;
- efektīvākā ir klātienes komunikācija;
- strādājoša programmatūra ir primārā progresa metrika;
- spējā metodoloģija veicina ilgtspējīgu attīstību;
- nepārtraukti ir jāpievērš uzmanība tehniskai izcilībai un labai arhitektūrai un dizainam;
- vienkāršība ir māksla maksimizēt to darba daļu, kuru nav nepieciešams realizēt;
- labākās arhitektūras un dizaini tiek veidoti pašorganizētās komandās;
- regulāros intervālos komanda pārskata padarīto, analizē savu darbu, veic nepieciešamās korekcijas.

Spējai metodoloģijai ir savas priekšrocības un trūkumi, kas sīkāk aprakstīti nākamajā apakšnodaļā.

1.1. Spējo metožu priekšrocības

Tiek daudz runāts un spriests, vai spējās metodes ir labākas par tradicionālajām vai nē. Kādas ir to priekšrocības un trūkumi? Spējo metožu treneris *Dave Moran* savā rakstā ir minējis 10 spējo metožu priekšrocības (autora personīgā pieredze un ekspertu viedoklis ir līdzīgs) [8]:

- **sistēmas ieviešanas ātrums** – spējās metodes liek uzsvaru uz ātru un regulāru sistēmas piegādāšanu, tādā veidā panākot, ka bizness var ātrāk atgūt investētos līdzekļus. Protams, ir jārēķinās ar to, ka spējās metodes

pieprasa gan izstrādātāja, gan pasūtītāja pilnīgu iesaistīšanos. Tikai tādā veidā izstrādātājs var iegūt visu nepieciešamo informāciju kārtējai iterācijai un koncentrēties uz svarīgākajām sistēmas iespējām, kuras jāpiegādā agrāk un atbilstošā kvalitātē;

- **mainīgā biznesa vide** – izmantojot spējās metodes, izmaiņas izstrādājamā sistēmā var ieviest katrā iterācijā. Tas palīdz sistēmās ātri ieviest jaunās biznesa vajadzības. Katrā nākamajā iterācijā biznesa vajadzības tiek sakārtotas pēc prioritātēm un tiek realizēta biznesam svarīgākā funkcionalitāte;
- **iespēja samazināt risku** – programmatūras izstrāde ir saistīta ar dažādiem riskiem. Spējās metodes samazina risku, pasūtītājam iegūt to, kas viņam nav vajadzīgs. Spējām metodēm ir iteratīvs raksturs, tāpēc pasūtītājs var ātri identificēt sistēmā neprecizitātes, un tās var tikt ātri novērstas. Nelielās iterācijas palīdz vieglāk sekot līdzi budžetam un nepieciešamības gadījumā veikt korekcijas;
- **spējās metodes palielina produktivitāti** – Ņemot vērā ātrās reaģēšanas nepieciešamību, komandai ir jābūt augstā sagatavotības līmenī un tā koncentrējas uz augstākās prioritātes iespējām, kuras pievieno augstu biznesa vērtību. Komandas strādā kopējā mērķa sasniegšanai, savā starpā dalās zināšanās, kas savukārt ceļ kopējo zināšanu līmeni komandā. Šādā veidā saliedētas komandas spēj būt produktīvākas;
- **spēja nodrošināt izstrādei labvēlīgu vidi** – programmatūras izstrādes projektiem bieži ir tendence novirzīties no plānotajiem grafikiem, kā rezultātā komandas ir spiestas strādāt virsstundas, kas pasliktina kopējo noskaņojumu komandā. Ņemot vērā spējo metožu iteratīvo raksturu, ir iespējams veikt nākamo iterāciju pārplānošanu, ņemot vērā problēmas kādā no iepriekšējām iterācijām. Nākošās iterācijas var tikt plānotas, balstoties uz kārtējo iterāciju produktivitātes rādītājiem. Vide, kurā ir samazināta spriedze, izstrādei ir labvēlīgāka un dod iespēju programmatūru izstrādāt augstākā kvalitātē;
- **inovāciju ieviešana** – jaunu tehnoloģiju ieviešana parasti ir komandas iniciatīva. Gadījumos, ja komanda ir pārslogota, tā neiesaistās jaunu tehnoloģiju ieviešanā, kas savukārt noved pie komandas zināšanu līmeņa

stagnēšanas. Labvēlīga vide sniedz iespēju ieviest dažādas sarežģītības inovācijas, kas palīdz uzturēt augstāku kvalitāti un produkta ilgtspējību;

- **uzticēšanās –** ņemot vērā faktu, ka strādājoša programmatūra tiek piegādāta pasūtītājam daudz biežāk, pastāv iespēja ar pasūtītāju veidot labākas attiecības. Izmaiņas var tikt ieviestas arī vēlākās stadijās, un tas rada pasūtītājos pārliecību, ka tiks izveidota tieši viņiem vajadzīgā programmatūra un tas palielina uzticēšanos komandai;
- **iespēja uzlaboties** – retrospektīvu un koda pārskatu izmantošana sniedz pilnveidošanās iespējas. Pēc katras retrospektīvas (atskats uz iepriekš paveikto, izmanto dažādās spējās metodēs, īpaši pieminēta metodē *Scrum* [22]) var apskatīt, kādas bija problēmas, kādi risinājumi ir izmantoti potenciālo problēmu atrisināšanai. Koda pārskati sniedz iespēju uzrakstīt kvalitatīvāku programmatūras kodu un mazāk pieredzējušiem programmētājiem ātrāk paaugstināt savu kvalifikāciju;
- **spējās metodes ir motivējošas** – katrs komandas loceklis spējā komandā ir par kaut ko atbildīgs, un šī atbildības sajūta var būt ļoti motivējoša. Nekas nevar būt nemotivējošāks, kā situācija, ka tu komandā nēesi atbildīgs par neko, tas liek justies mazsvarīgam, tas noved pie produktivitātes samazināšanās;
- **pielāgošanās projektiem** – katrs projekts savā ziņā ir unikāls, spējās metodes dod iespēju pielāgoties katram individuālajam projektam un komandai, lai tie būtu veiksmīgi.

1.2. Spējo metožu riski

Spējo metožu izmantošanā ir arī riski, ar kuriem uzņēmumiem un komandām ir jārēķinās [9]:

- **neprofesionāla komanda** – risks pastāv gan tradicionālās izstrādes gadījumā, gan spējās izstrādes gadījumā. Ņemot vērā, ka spējās metodes paļaujas uz augstu komandas profesionalitāti, tad riskam var būt liela ietekme uz gala rezultātu. Neprofesionāla komanda nespēj īsā laikā izstrādāt, kvalitatīvu produkta arhitektūru un programmas kodu, kas vēlāk noved pie liela kļūdu skaita un sliktas produkta kvalitātes un tās izrietošajām sekām;

- **slikta komunikācija ar pasūtītāju vai komandas iekšienē –** komunikācija var būt slikta, jeb neefektīva, ne tikai spējos projektos, bet arī tradicionālajos. Spējām metodēm ir nepieciešama pasūtītāja intensīva iesaistīšanās projekta izstrādē. Neizskaidrojot pasūtītājam, ka šāda iesaistīšanās ir nepieciešama, pasūtītājs izstrādes laikā var apjukt un tas var radīt saspīlējumu komunikācijā. Slikta komunikācija ar pasūtītāju noved pie savstarpējās nesaprašanās un pie produkta, kurš neatbilst pasūtītāja prasībām. Komunikācijas problēmas komandas iekšienē noved pie nevajadzīga saspīlējuma izstrādes laikā, kā arī var novest pie dubultas kāda darba darīšanas vai dīkstāves;
- **specifikācija vai prasības ir pārāk abstraktas** – projektiem, kuri izmanto spējās metodes, nav nepieciešami ļoti detalizēti specifikāciju un prasību apraksti, bet pastāv risks, ka tie ir pārāk abstrakti un komanda nevar uzsākt izstrādi vai dara to nepareizi. Slikti definētās prasības, noved pie neprecizitātēm novērtēšanas procesā un pie produkta, kurš neatbilst pasūtītāja prasībām;
- **netiek implementēta vai pareizi izmantota retrospektīva** – retrospektīvai ir liela nozīme spējās metodēs, jo tā dod iespēju komandai uzlaboties [108]. Ir komandas, kuras uzskata retrospektīvu par lieku laika tērēšanu un tās netiek organizētas, liedzot komandai atskatīties uz savu darbu un potenciālajām problēmām. Ja nav informācijas par problēmu, to nevar novērst. Pastāv risks, ka retrospektīva tiek izmantota, bet tas netiek darīts pareizi un sapulcei ir tikai formāls raksturs. Piemēram, sapulcei nav vadītājs, jeb moderators, kā arī sapulces laikā runā tikai daļa no komandas, piemēram, kāds autoritārs tās dalībnieks, kas neļauj izteikties pārējai komandai;
- **nenotiek zināšanu uzglabāšana** – projektu izstrādes laikā, komandas gūst zināmu pieredzi. Pieredze, var būt par kādu no izmantotajām tehnoloģijām vai kādas konkrētas problēmas atrisināšanu. Gadījumos, ja iegūtās zināšanas netiek uzglabātas, vai komanda ar tām nedalās, pastāv risks, atkārtot šo problēmu atkal un atkal nākamajās iterācijās vai citos projektos;
- **nepareizu metriku izmantošana vai to neizmantošana** – spējās metodēs ir nepieciešams noteikt projekta progresu, lai nepieciešamības gadījumā varētu veikt nepieciešamās korekcijas izstrādē un piegādēs. Nepareiza

progresa noteikšana noved pie kavētām projekta piegādēm un slikti saplānotām iterācijām (iterācija ir neliels izstrādes cikls, kura laikā tiek veidots produkta papildinājums). Pastāv risks, ka komanda izvēlas nepareizas metrikas progresa noteikšanai un nevar precīzi noteikt iterācijas vai projekta progresu;

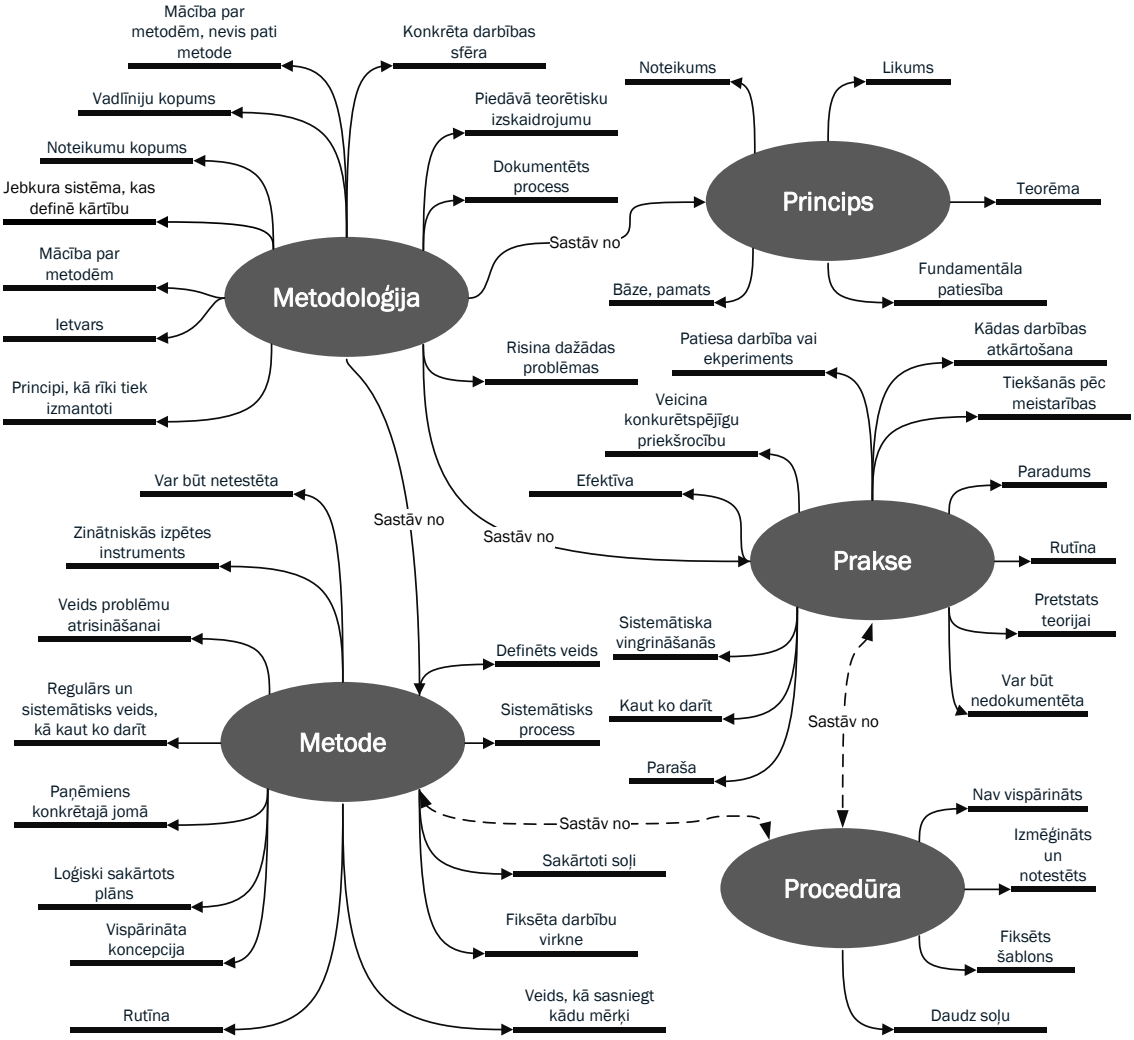
- **neprecīzi saplānots, nepieciešamais laiks –** laika plānošana ir svarīga gan tradicionālajām gan spējām metodēm. Spējās metodēs, sākotnējā laika plānošana notiek pirms darbu uzsākšanas, un saplānotais laiks ir aptuvens (angl. *gestimate*). Precīzāka laika plānošana katrai izstrādājamajai vienībai notiek pirms iterācijas. Pastāv risks, ka nepieciešamā laika novērtējums ir neprecīzs, kā rezultātā, iterācijā ieplānotais darbs netiks pabeigts un netiks atrādīts pasūtītājam. Neprecīza plānošana, noved pie neveiksmīgas iterācijas, kuras rezultātā nepabeigtās lietas ir jāpārliek atpakaļ darāmo darbu saraktā (angl. *Product backlog*) un pie tām būs jāatgriežas, kādā no nākamajām iterācijām. Neprecīzas novērtēšanas risku palielina neprofesionāla komanda, jo spējās metodēs novērtējumus sniedz nevis viens speciālists, bet visa komanda;
- **projekta vai iterācijas robežas ir izplūdušas –** izplūdušu robežu risks pastāv gan tradicionālās, gan spējās metodēs. Spējās metodēs tas nav tik izteikts, kā tradicionālās, kad ir brīdis, kad pasūtītājs saprot, ka pasūtītais nav īsti tas, kas bija nepieciešams un vēlas papildu izmaiņas. Spējās metodēs pasūtītājs regulāri redz produktu un var veikt izmaiņas darāmo darbu sarakstā, bet pastāv risks, ka esošajā iterācijā, kura jau ir sākusies, kāds mēģinās pievienot uzdevumus, kuri sākotnēji tur nav bijuši, ar šādu rīcību mainot iterācijas robežas un tai nepieciešamo laiku;
- **sarežģīti noslēgt līgumu** – tradicionālajās metodēs līguma noslēgšana ir vienkāršāka, jo ir vieglāk aprakstīt, kas tieši, kādā laikā un apjomā tiks piegādāts. Spējās metodēs pastāv variēšanas iespējas, jo piegādājamais saturs var mainīties, kas sarežģī līguma noslēgšanu ar pasūtītāju;
- **nepietiekams zināšanu līmenis** – ir uzņēmumi un komandas, kuri vēlas izmantot spējās metodes, bet to zināšanu līmenis par spējām metodēm ir nepietiekams. Pastāv risks, ka komanda, realizējot projektu, nezina, ka process tiek veikts nepareizi un tas noved pie projekta izgāšanās.

Katrs uzņēmums un komanda, kuri vēlas izmantot spējās metodes var izvērtēt šos riskus un spējo metožu priekšrocības, pirms uzsākt spējo metožu ieviešanu. Jāņem vērā, ka ir iespējams mazināt definētos riskus, izmantojot atbilstošas prakses.

1.3. Terminoloģijas problēma

Spējo programmatūras izstrādi cenšas ieviest daudzi dažāda izmēra uzņēmumi, kuros strādā dažādas komandas un indivīdi, kuru zināšanu līmenis ievērojami atšķiras. Problēma rodas faktā, ka dažādi avoti apraksta vienu un to pašu situāciju, izmantojot dažādas definīcijas. Piemēram, daļā avotu tiek runāts par to, ka *Scrum* ir ietvars, bet citos ka tā ir metodoloģija. Un dotais piemērs nav vienīgais, ar ko jāsaskaras, ieviešot spējo metodoloģiju. Problēmas atrisināšanai ir nepieciešams veikt terminu analīzi un nonākt pie kāda noteikta kopsaucēja.

Problēmas atrisināšanai tika apkopotas terminu definīcijas no 28 dažādiem avotiem un veikta definīciju sadalīšana atsevišķos terminu aprakstos, kurus iespējams izmantot terminu un to savstarpējo relāciju vizualizācijas kartē. Pamatā tiek izmantoti termini „Metodoloģija", „Metode", „Princips", „Prakse" un „Procedūra" (sk. 1.1. att.).



1.1. att. Spējo terminu karte.

Termins „metodoloģija" tiek aprakstīts kā „konkrēta darbības sfēra", „piedāvā teorētisku izskaidrojumu", „ietvars", „dokumentēts process" utt. Metodoloģija ietver metodes un prakses, kā arī ir konkrētu principu vadīta.

Termins „metode" tiek aprakstīts kā „zinātniskās izpētes instruments", „veids problēmu risināšanai", „loģiski sakārtots plāns", „vispārināta koncepcija", „definēts veids". Metode var sastāvēt no procedūrām, kuras pareizā veidā sakārtojot un izpildot, var sasniegt kādu definētu rezultātu.

Termins „prakse" labi tiek aprakstīts ar frāzēm „rutīna", „paradums", „pretstats teorijai" un „var būt nedokumentēta". Prakses sastāv no dažādām procedūrām, tās nav vispārinātas un ir sistemātiskas.

Pirmās problēmas ir ar terminiem „metode” un „procedūra”. Piemēram, „Pirms izpildīt B un C ir jāizpilda darbība A”. Konkrētais apraksts apzīmē metodi vai praksi? Kad mēs sakām, ka metode, lai atrisinātu šo problēmu, ir izpildīt A, B un C, mēs patiesībā runājam par procedūru. Metode problēmas atrisināšanai ir sekot konkrētajai procedūrai. Terminoloģijas kartē (sk. 1.1. att.) šo relāciju var redzēt vizuāli, ka metode sastāv no procedūrām. Viens no vieglākajiem veidiem atšķirt terminus „metode” un „procedūra” ir iegaumēt, ka procedūra netiek vispārināta un ir ļoti konkrēta, bet metode var būt vispārināta.

Nākamā problēma ir ar terminiem „metodoloģija” un „metode”, problēma rodas, jo pēdējā laikā autori ir sākuši izmantot terminu „metodoloģija” kā alternatīvu terminam „metode”, kas savukārt noved pie nepareizas izpratnes par šo terminu izmantošanu. Ir sastopami darbi, kuros autors darba laikā izmanto abus šos terminus, aprakstot vienu un to pašu problēmu. Un šis fakts vēl vairāk var samulsināt komandas un organizācijas, kuras izmanto literatūru par spējo metodoloģiju ieviešanu un praktizēšanu. Definīciju dekompozīcija un sadalīšana aprakstošās frāzēs, kā arī terminu vizualizācija sniedz iespēju labāk atšķirt šos abus terminus. Viens no šādiem aprakstiem ir „metodoloģija ir mācība par metodēm, nevis metode” [43]. Lai gan šis apraksts ir precīzs un labi paskaidrojošs, pastāv arī citi apraksti, kas var radīt apjukumu terminu skaidrojumā. Viens no šādiem skaidrojumiem ir „jebkura sistēma, kas definē kārtību” [49]. Definēta soļu kārtība ir atrodama arī termina „metode” aprakstā, kur metode, tiek definēta kā „loģiski sakārtots plāns” vai „fiksētu darbību virkne” [50]. Ņemot vērā faktu, ka autoriem ir dalītas domas kā izmantot terminu „metodoloģija” un „metode”, bet tālākam darbam ir nepieciešama skaidrība, ir izlemts apstāties pie biežāk izmantotajiem skaidrojumiem. Metodoloģiju aprakstīt kā „metodoloģija ir metožu un principu sistēma, kas tiek izmantota konkrētā darbības sfērā” [51] un „metodoloģija ir mācība par metodēm nevis metode” [52]. Termins „metode” tiek aprakstīta kā „regulārs un sistemātisks veids, kā sasniegt mērķi, it īpaši sistemātisks un regulārs” [50].

Izveidotā terminu karte tiek izmantota nākamajās apakšnodaļās, lai varētu veikt spējo metožu, prakšu un atslēgas vārdu analīzi.

1.4. Aktualitātes indekss

Aktualitātes indekss (AI) ir indikators, kurš nosaka, cik ieinteresēti konkrētajā metodē, praksē vai problēmā ir organizācijas un komandas. AI, kas noteikts, balstoties uz konferenču materiālu analīzi, ir izteikts ar vienādojumu

$$IOA_{\text{m}} = \frac{100 \cdot A_{\text{m}}}{A_{\text{y}}}, \tag{1}$$

kur IOA_{m} – metodes AI;

A_{m} – rakstu skaits, kuros pieminēta metode;
A_{y} – rakstu skaits gadā.

Prakšu un atslēgas vārdu AI arī tiek aprēķināts izmantojot doto formulu A_{m} aizvietojot atbilstoši ar prakšu vai atslēgas vārdu skaitu konkrētajā gadā.

1.4.1. Spējo metožu AI

Vadoties pēc terminoloģijas kartes, ir noteikts, ka metodes ir *Scrum*, *Extreme Programming*, *Lean*, *Dynamic System Development*, *Crystal*, *Feature Driven Development* un *Agile Model Driven Development*.

Dotā apakšnodaļa apraksta spējās metodes ar augstāko AI. Augstākais AI metodēm ir noteikts periodā no 2008. līdz 2012. gadam, par pamatu ņemot datus no konferencēm par spējo metodoloģiju, kuras organizē *Agile Alliance*. *Agile Alliance* ir bezpeļņas organizācija un ir viens no būtiskākajiem spēlētājiem šajā nozarē. *Agile Alliance* organizētās konferences pilnā mērā atspoguļo nozares tendences. Zināšanas par metožu AI var palīdzēt uzņēmumam un komandai transformācijas un adoptēšanas procesā.

AI iegūšana sastāv no pieciem posmiem. Pirmais posms ir spējo metožu saraksta izveidošana, otrais posms ir elementa no saraksta pārbaude pret terminoloģijas karti (sk. 1.1. att.). Trešais posms ir datubāzes un programmatūras izveidošana. Datubāze un programmatūra ir nepieciešama, lai saglabātu un koriģētu informāciju par rakstiem. Ceturtais posms ir datu par konferencēm iegūšana un saglabāšana. Par rakstiem tiek saglabāta šāda informācija:

- raksta virsraksts,
- apraksts,

- gads,
- raksta grupa vai apgabals;
- vietrādis *URL* (angl. *Uniform Resource Locator*) – raksta adrese.

Atslēgas vārdu identificēšana rakstos ir manuāls process, kurā visi atslēgas vārdi tiek identificēti, ņemot vērā raksta kontekstu, nevis atslēgas vārda parādīšanos rakstā.

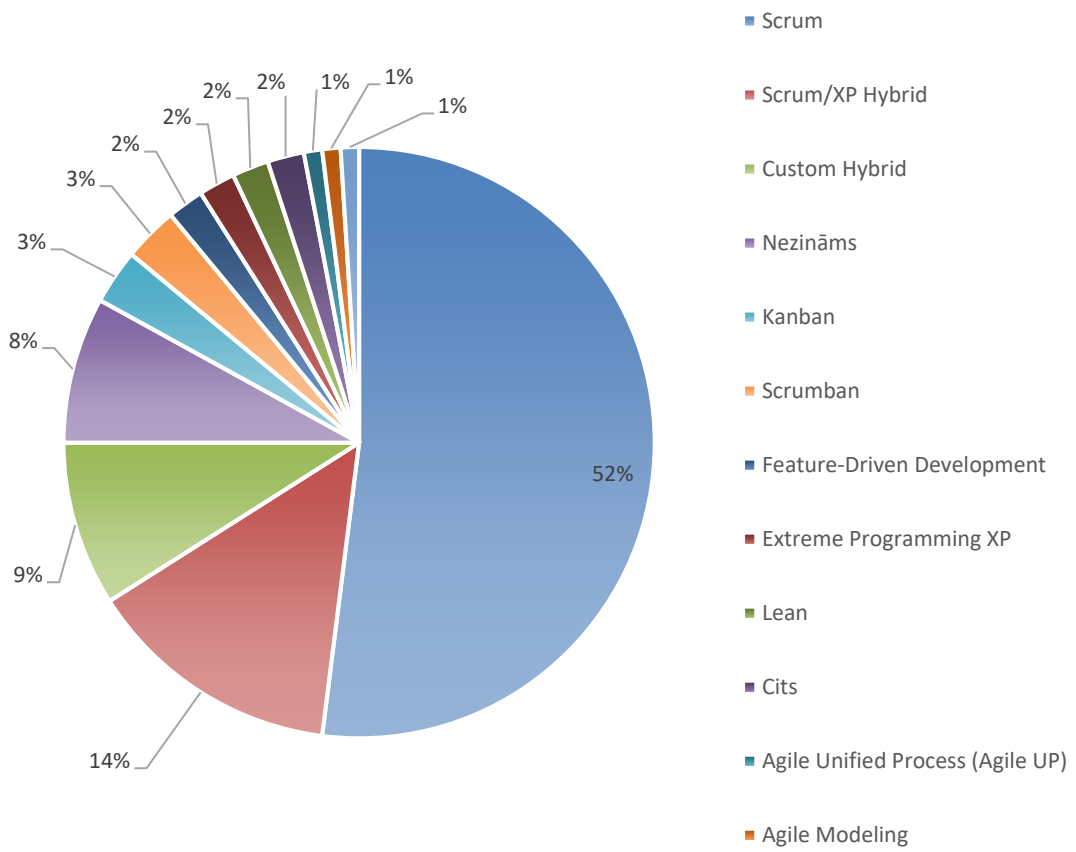
Piektais posms ir iegūto datu grupēšana, kārtošana un apvienošana, kā rezultātā ir iegūts spējo metožu AI (sk. 1.1. tabula).

1.1. tabula

Spējo metožu AI

Metode	**2008**	**2009**	**2010**	**2011**	**2012**
Scrum	18,75	16,78	15,00	13,51	16,82
Extreme programming	8,16	5,77	3,26	1,93	1,66
Lean	7,91	5,77	7,39	8,69	11,14
Dynamic System Development Method	0,51	0,70	0,00	0,00	0,00
Crystal	1,02	0,35	0,00	0,00	0,00
Feature Driven Development	0,00	0,52	0,22	0,00	0,24
Agile Model Driven Development	0,26	0,00	0,43	0,00	0,00

Rezultāti, kas apkopoti 1.1. tabula, norāda uz to, ka populārākā metode ir *Scrum*, kurai seko *Lean* pieeja. Līdzīgus secinājumus ir ieguvuši arī *VersionOne*. Viņu 2011. gada pētījuma rezultātā ir noteikts, ka populārākā metode ir *Scrum* (sk. 1.2. att.), kuru izmanto 52 % aptaujāto, kā arī 14 % izmanto kādu no *Scrum* hibrīdu metodēm [115]. Metožu aktualitātes dinamiku ir iespējams aplūkot P.1.1. att.



1.2. att. Spējo metožu izmantošanas rādītāji.

Scrum metodei ir augstākais AI visā periodā no 2008. līdz 2012. gadam, bet *Lean* pieejas AI palielinās katru gadu, turklāt *Extreme Programming* savu AI vērtību zaudē gadu no gada. Pārējo metožu AI ir ievērojami zemāks un nebūtu ieteicams šīs metodes izmantot organizācijas un komandas pārējai uz spējo programmatūras izstrādi. Spējās metodes izvēle ir svarīgs posms pārējai uz spējo metodi, tomēr metodes izvēle nav vienīgais, ar ko jāsastopas uzņēmumiem un organizācijām. Svarīgi ir izvēlēties atbilstošas spējās prakses, kuras kombinējot ar izvēlēto metodi, varētu sniegt nepieciešamo rezultātu. Detalizētāk spējās prakses un to AI ir aprakstītas nākamajā apakšnodaļā.

1.4.2. Spējo prakšu AI

Spējās metodes sastāv no dažādām procedūrām un definē vispārinātu vai konceptuālu veidu, kā sasniegt kādu noteiktu rezultātu. Metodes pašas par sevi negarantē veiksmīgu projekta iznākumu. Lai sasniegtu vēlamo rezultātu, ir nepieciešams izmantot spējās prakses. Spējo prakšu kopas izvēle ir atkarīga no

organizācijas vai komandas. Vairums gadījumu spējo prakšu izvēle ir komandas atbildība, kuru var ietekmēt dažādi ārējie faktori, piemēram, konkrēts projekts, konkrēts pasūtītājs vai kāds cits faktors.

Dotās apakšnodaļas mērķis ir noteikt spējo prakšu AI gadu griezumā. AI noteikšanai tiek izmantota informācija, kura iegūta iepriekšējos soļos, nosakot spējo metožu AI. Datu pētīšanas un analizēšanas rezultātā ir iegūts saraksts ar 35 spējām praksēm (sk. 1.2. tabula). Spējo prakšu aktualitātes dinamiku grafiskā veidā ir iespējams aplūkot P.2.7. att.

1.2. tabula

Spējo prakšu AI

Prakse	**2008**	**2009**	**2010**	**2011**	**2012**
Test Driven Development	12,88	10,84	3,04	6,56	5,69
Retrospective	11,48	15,73	3,91	2,70	4,74
Review	8,55	8,22	0,87	1,74	2,61
INVEST	8,42	7,34	0,87	1,16	5,69
Code Refactoring	7,14	5,59	1,96	2,90	3,79
User stories	6,76	6,12	3,48	4,44	6,64
Continuous deployment	6,63	8,04	1,30	1,54	1,42
Backlog usage	6,38	7,34	4,13	1,93	6,64
Pair programming	6,12	5,77	2,61	3,47	3,08
Measurements	5,36	6,47	3,48	1,54	3,32
Automated testing	4,08	1,31	0,43	1,54	0,00
Automated unit testing	3,95	2,45	1,30	1,16	1,90
Continuous integration	3,83	6,99	0,87	1,93	1,42
Acceptance testing	3,57	2,27	1,30	2,32	0,95
Release Planning	3,57	2,27	0,00	0,39	0,95
Iteration Planning	3,32	2,80	1,30	0,00	0,47
Estimation	2,81	1,40	1,74	0,77	2,37
Daily Stand up meeting	2,81	1,40	1,30	0,39	0,47
Behavior Driven Development	2,81	3,50	1,74	1,35	1,42
Planning game	2,30	1,57	0,65	0,00	0,95
Working software	1,79	2,10	1,30	0,77	1,42
Source Control	1,53	1,40	0,00	0,77	0,47
Active Stakeholder Participation	1,28	2,80	0,43	0,39	1,90
Definition of Done	1,02	0,52	0,22	0,19	2,13
Emergent Design	1,02	1,57	0,00	0,19	1,66
Exploratory testing	1,02	0,70	0,43	0,77	0,47
Facilitation	0,89	2,10	0,22	0,39	1,18
Cross-functional team	0,26	1,40	0,00	0,77	3,32
Usability testing	0,38	1,05	1,30	0,39	0,00
Code review	0,26	1,05	0,43	0,00	0,47
Story mapping	0,26	0,00	0,43	0,39	0,95
Sustainable pace	0,26	0,70	0,43	0,00	0,47
Kanban board	0,26	0,70	0,43	0,00	0,95
Acceptance Test Driven Development	0,38	0,52	0,00	1,16	1,66
Automated build	0,38	0,87	0,43	0,00	1,42

Spējo prakšu AI iegūšana sastāv no vairākiem soļiem. Pirmais solis ir izveidot spējo prakšu sarakstu. Sākotnējais saraksts sastāvēja no 176 spējām praksēm, un to iegūšanai tika izmantoti 22 avoti. Otrais solis ir dublējošo prakšu izņemšana no saraksta. Dažādos avotos pēc satura ir aprakstītas vienas un tās pašas prakses, izmantojot citu nosaukumu. Apvienošanas rezultātā iegūstam sarakstu ar 111 praksēm, konkrētais saraksts tālāk tiek izmantots AI noteikšanai. Trešais solis ir izvēlēto prakšu pārbaude attiecībā pret datu bāzē saglabātajiem atslēgas vārdiem, kā rezultātā iegūstam informāciju par tām spējām praksēm, kurām ir iespējams noteikt AI. Pēc prakšu pārbaudes ir iegūts saraksts ar 70 spējām praksēm, kuru AI ir iespējams noteikt. Ņemot vērā, ka saraksts ar 70 praksēm ir pārāk liels, lai to ievietotu darbā, tad ir izlemts darbā ievietot tikai 30 prakses ar augstāko AI gada griezumā. Sakarā ar to, ka kāda prakse var iekļūt 30 praksēs ar augstāko AI vienā gadā, bet nebūt šajā sarakstā nākošajā, tad ir izlemts, ka prakses, kuras ir iekļuvušas top 30 prakšu sarakstā ar augstāko AI kādā no gadiem, tad informācija par šādām praksēm ir jāiekļauj arī pārējos atskaites gados. Šādu darbību rezultātā, pēdējais prakšu saraksts sastāv no 35 (sk. 1.2. tabula) nevis 30 praksēm, kā bija izlemts sākotnēji.

Informācija, kura apkopota 1.2. tabula ir skaidri redzams, ka prakse ar augstāko AI 2008. gadā ir TDD, kurai seko *Retrospective* un *Review*, *INVEST* (angl. *Independent, Negotiable, Valuable, Estimable, Small, Testable*) un *Code Refactoring* prakse. 2009. gadā prakses ar augstāko AI ir *Retrospective*, TDD, *Review*, *Continuous deployment*, *Backlog usage* un *INVEST*. Līdzīga situācija atkārtojas arī ar praksēm 2010. gadā, piemēram, *Backlog usage*, *Retrospective*, TDD, *User stories* un *Measurement system* prakses. Vērā ņemamas atšķirības nav arī 2011. gadā, kur augstākais AI ir praksēm TDD un *User stories*. 2012. gadā prakses ar augstāko AI ir *User stories*, TDD, *INVEST* un *Backlog usage*. Visā pārbaudes periodā prakšu sarakstā augšgalā figurē tās pašas prakses, tikai nedaudz mainās to secība gadu griezumā.

Prakšu AI noteikšana var palīdzēt organizācijām un komandām noteikt, kādas prakses varētu ieviest vai izmēģināt nākamajās iterācijās. Metožu un prakšu AI noteikšana nav vienīgais, ko var iegūt, analizējot konferenču materiālus. Nākamajā apakšnodaļā ir detalizētāk apskatīti ar spējo metodoloģiju saistītie atslēgas vārdi un noteikts to AI.

1.4.3. Atslēgas vārdu AI

Atslēgas vārdu AI noteikšana ir labs veids organizācijām un komandām, kā iegūt informāciju par to, kas interesē citas organizācijas un komandas. Atslēgas vārdi ir termini, kuri identificē interesējošo virzienu, piemēram *Learning* vai *Coaching*. Šīs apakšnodaļas nolūks ir atslēgas vārdu AI noteikšana.

Atslēgas vārdu datu sagatavošana sastāvēja no trīs soļiem. Pirmais solis ir atslēgas vārdu manuāla identificēšana konferenču materiālos. Manuāla identificēšana ir atslēgas vārda identificēšana, ņemot vērā raksta kontekstu, nevis vienkāršu atslēgas vārda parādīšanos tekstā. Šāda identificēšana uzlabo atrasto atslēgas vārdu kvalitāti. Kopumā promocijas darbā veiktā pētījuma rezultātā tika identificēti 1257 atslēgas vārdi. Izveidotā programmatūra sniedz iespēju par rakstu saglabāt n daudzumu atslēgas vārdu. Ņemot vērā, ka daži atslēgas vārdi ir sinonīmi, nākamais solis ir sinonīmu apvienošana. Trešais solis ir atslēgas vārdu grupēšana gadu griezumā, katrā gadā izvēloties 30 atslēgas vārdus ar augstāko AI. Ņemot vērā, ka gadu griezumā top 30 nav vieni un tie paši atslēgas vārdi, tad informācija par iztrūkstošajiem atslēgas vārdiem kādā no gadiem arī tiek pievienota. Iztrūkstošo datu pievienošanas rezultātā ir iegūts atslēgas vārdu saraksts ar augstāko AI, kurš sastāv no 37 ierakstiem (sk. 1.3. tabula). Vizuāli atslēgas vārdu dinamiku ir iespējams aplūkot P.2.8. att.

1.3. tabula

Atslēgas vārdi ar augstāko AI

Atslēgas vārds / termins	**2008**	**2009**	**2010**	**2011**	**2012**
Agile adoption	16,58	21,33	13,48	20,08	23,70
Experience report	16,33	11,19	2,61	9,65	10,90
Agile team	16,07	6,29	13,91	11,58	18,01
Practices	14,80	14,69	31,74	20,08	21,33
Testing	14,80	14,34	9,13	7,34	9,00
Leadership	14,54	13,99	12,61	6,56	8,53
Organizational culture	14,29	13,64	8,26	10,04	13,74
Tools	11,99	11,54	9,57	15,83	8,53
Business value	10,97	8,74	3,04	3,47	9,95
Customer	10,46	6,64	2,61	6,95	11,85
Distributed agile	10,20	8,04	7,83	1,93	3,79
Development	9,69	11,54	9,13	6,18	8,53
Learning	8,93	7,69	3,48	12,74	6,16
Large scale agile	7,40	6,64	10,00	5,79	8,53
Transition	7,40	6,29	15,22	15,44	16,59
Quality	7,40	4,55	2,61	7,34	7,58
Collaboration	6,89	8,39	7,83	17,37	20,85
Communication	6,89	5,59	5,22	5,41	5,21
Organization	6,63	5,94	11,74	4,25	1,42
Coaching	4,08	11,54	5,22	10,04	12,32

Atslēgas vārds / termins	2008	2009	2010	2011	2012
Enterprise	3,57	6,99	16,96	9,27	12,32
User experience	5,10	6,99	6,96	5,41	8,06
Environment	4,85	6,64	7,83	5,79	7,58
Planning	3,06	6,29	3,04	3,47	4,27
Product management	0,77	5,94	6,09	0,39	0,47
Requirements	1,79	4,90	10,00	3,86	4,27
Teambuilding	1,02	2,45	6,96	1,93	2,84
Project management	2,81	2,45	6,96	6,56	0,95
Problems	4,08	4,20	6,52	5,79	5,69
Research	2,55	1,75	5,65	5,79	4,74
Hands on labs	0,26	2,10	3,04	8,49	6,64
Business	0,51	1,05	1,30	7,34	1,42
Mentoring	0,77	2,10	0,43	6,18	7,58
Innovation	1,53	4,55	2,17	5,79	5,21
Principles	0,00	0,70	2,17	1,93	7,11
Scaling agile	2,30	5,59	4,35	3,47	6,64

Atslēgas vārdi ar augstāko AI 2008. gadā ir *agile adoption*, *experience report*, *agile team*, *practices* un *testing*. Pieredzes ziņojumi (*experience report*) pēdējā laikā ir ieguvuši vērā ņemamu vietu konferencēs par spējo metodoloģiju, kas ir izskaidrojams ar to, ka organizācijas un komandas vēlas dalīties ar savu pieredzi ar pārējiem. Šajā gadījumā tiek ņemta vērā gan pozitīvā, gan negatīvā pieredze. Līdzīga situācija ir arī 2009. gadā, atslēgas vārdi ar augstāko AI ir *agile adoption*, *practices* un *testing*. Jaunpienācējs sarakstā ir *leadership* un *organizational culture*.

Līderības (angl. *Leadership*) jēdziens komandās tiek interpretēts dažādi. Dažas komandas to interpretē tā, ka līderim ir noteikti jābūt, citas, ka noteikts līderis neeksistē, jo viens no spējo metožu pamatprincipiem ir tāds, ka komandas ir pašorganizējošas. Komandu interpretācija par līderības un pašorganizējošas komandas pamatprincipu mēdz būt kļūdaina, jo pašorganizējoša nenozīmē bez līdera [119][120], un ir nepieciešams uzņemties līderību komandas ietvaros, lai sasniegtu vēlamo rezultātu. Viens no vienkāršākajiem piemēriem līderības nepieciešamībai ir situācija, kad bariņam indivīdu, kuri tiek uzskatīti par spējo komandu, liek vienlaikus palekties. Novērojot komandas darbību, iesākumā komandas dalībnieki saskatās, līdz kāds uzņemas līderību un sāk komandēt, koordinētu komandas vienlaicīgu palekšanos. Šādā kontekstā ir skaidrs, ka atsevišķos momentos bez līderības nav iespējams organizēt koordinētu komandas darbu, un katrā atsevišķā posmā ir jābūt definētam līderim, kurš spējās komandas gadījumā var mainīties.

Rakstos par spējām praksēm, uzmanība tiek pievērsta, konkrētās prakses pielietošanai un kombinēšanas iespējām ar citām praksēm, konkrētas problēmas risināšanai.

2010. gadā atslēgas vārdi ar augstāko AI ir *practices* un *enterprise*. *Enterprise* šajā kontekstā ir spējās metodes izmantošana liela izmēra organizācijās. Viens no jaunpienācējiem ir *transition*, kurš apraksta organizācijas transformāciju uz spējo projektu izstrādi. Organizācijas transformācijas periods var būt dažāds un ir atkarīgs no pašas organizācijas un komandas vai komandām, kuras ir iesaistītas transformācijas periodā. Vēl viens jaunpienācējs ir *large scale agile,* kas ir tas pats *enterprise agile*.

2011. gadā atslēgas vārdi ar augstāko AI ir *agile adoption, practices*, kā arī atslēgas vārdi *collaboration, learning* un *coaching* ir palielinājuši savu AI šajā periodā. Tieši interese par *learning* un *coaching* ir interesanta, jo norāda uz organizāciju un komandu tiekšanos uz izglītošanos un vēlmi uzzināt vairāk par spējo metodoloģiju. Papildu tam, interese par *coaching* norāda arī uz to, ka organizācijas un komandas sāk pastiprināti interesēties par treneru iesaistīšanu mācīšanās procesā, lai pēc iespējas ātrāk uzlabotu spējās metodes izmantošanu.

Līdzīga situācija ir arī 2012. gadā, kur atslēgas vārdi ar augstāko AI ir *agile adoption, practices, transition, agile team* un *organizational culture*.

Pēc visu atslēgas vārdu analīzes, ir pamanāms, ka organizācijas un komandas interesējās par vienām un tām pašām lietām pārbaudes periodā, nedaudz tikai mainās ieinteresētības prioritātes. Atslēgas vārdu *agile adoption* un *practices* AI ir augsts visu pārbaudes periodu, kas norāda uz paaugstinātu interesi par spējo metožu apgūšanu un pareizu prakšu izvēli konkrētajā situācijā. Atslēgas vārdiem *learning* un *coaching* arī ir liela nozīme, un tieši šie atslēgas vārdi pārbaudes periodā ir palielinājuši savu AI. Kā jau iepriekš tika minēts, paaugstināta interese par *learning* un *coaching* norāda uz organizāciju un komandu vēlmi ātrāk apgūt spējās metodes un pēc iespējas ātrāk izprast savas kļūdas un tās labot. Sakarā ar to, ka iespēja pieaicināt spējo metožu trenerus nav visiem pieejama augsto izmaksu dēļ, tāpēc ir nepieciešams rast kādu alternatīvu risinājumu. Darba ietvaros tiek piedāvāts izstrādāt jaunu metodi organizācijas spējības noteikšanai, lai to varētu izmantot organizācijas transformācijas procesā, kā arī pēc transformācijas procesa beigām, kura palīdzētu noteikt problemātiskos apgabalus spējo metožu ieviešanā, neiesaistot spējos ekspertus katrā konkrētajā gadījumā.

1.5. Dinamiska vide

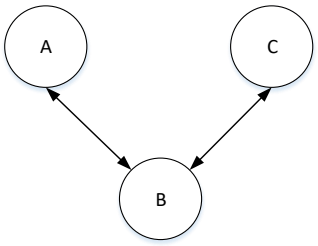
Spējo metožu ieviešana parasti ir saistīta ar vēlmi būt veiksmīgākiem un nodrošināties, ka projekti, pie kuriem strādā organizācija, neizgāžas. Izgāšanās un problēmas bieži ir saistītas ar dinamisko vidi, jo organizācija un komanda ir gatava vienam scenārijam, bet nav gatava kādam citam. Organizācijām un komandām ir nepieciešams pielāgoties pie konkrētā projekta, jo nevar apgalvot, ka viena un tā pati pieeja darbosies vienādi dažādos projektos ar dažādām komandām. Vide, kura veidojas konkrētā projekta un konkrētās komandas ietvaros, ir mainīga. Ir vairākas dinamiskas vides definīcijas:

Dinamiska vide sastāv no mainīgas apkārtējās vides, kurā darbojas aģents. Aģenta uzdevums ir pielāgoties jaunām situācijām, lai pārvarētu neprognozējamus šķēršļus [82].

Dinamiska vide ir strauji mainīga darba vide organizācijā. Dažas no šīm izmaiņām ietver vadības problēmas, tādas kā strauja lēmumu pieņemšana, agresīvs marketings, augsta riska iniciatīvas un citas. Augsti konkurējošās nozarēs ir nepieciešams uzturēt dinamisku organizācijas vidi [83].

Vide E ir statiska attiecībā pret aģentu A, ja A ievades dati no E ir pilnībā atkarīgi no aģenta A izejas uz vidi E. A dinamiskā vide ir tāda, kas nav statiska [84]. Piemēram, lampa, kura ieslēdzas, kad tiek pievienota pie strāvas, un izslēdzas, kad tiek atslēgta, ir statiska vide. Tai pašā laikā, ja aģents A pārvalda lampu ar gaismas sensora starpniecību, kurš ieslēdz lampu tikai tad, ja istabā ir tumšs, šajā gadījumā runa ir par dinamisku vidi.

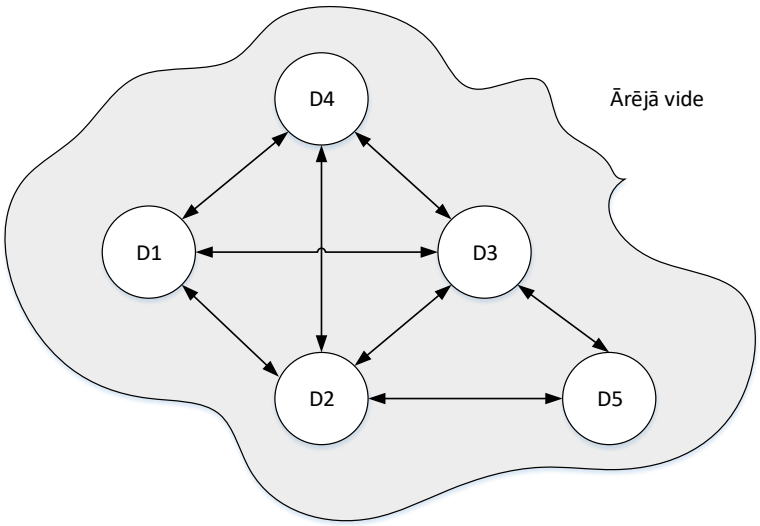
Dinamiska vide tiek raksturota ar to spējību autonomi mainīties. Ja aģenti A un C abi mijiedarbojas ar aģentu B, bet nav mijiedarbības starp aģentiem A un C, un C mijiedarbība ar B ietekmē B aģenta uzvedību, tad B veido dinamisku vidi attiecībā pret A (sk. 1.3. att.).



1.3. att. Dinamiska vide.

Vide ir pastāvīga, ja tās izvades dati ir atkarīgi ne tikai no tūlītējiem ievadiem, bet arī no agrākiem ievadiem. Vide ir amnēziska, ja tā nav pastāvīga [84]. Piemēram, ja lampa tiek ieslēgta ar slēdzi, un vēlāk ieslēgtā lampa tiek izslēgta ar to pašu slēdzi, tad runa ir par pastāvīgu vidi.

Ņemot vērā visas iepriekš minētās dinamiskās vides definīcijas, varam šīs definīcijas attiecināt uz programmatūras izstrādes vidi organizācijā.



1.4. att. Programmizstrādes vide.

Kā apskatāms 1.4. att. , katru darbinieku varam uzskatīt par aģentu, kur tie viens ar otru mijiedarbojas tiešā vai netiešā veidā, ietekmējot citus aģentus un to darbību. Darbinieki D1, D2, D3, D4 mijiedarbojas viens ar otru tiešā veidā, bet D5 mijiedarbojas tikai ar D2 un D3, bet, ņemot vērā, ka D2 un D3 var tikt ietekmēti no D1 un D4 puses, tad šīs mijiedarbības rada dinamiskumu arī uz D5. Tam visam papildu jāņem vērā, ka eksistē arī ārējā vide, kura var ietekmēt kādu vai visus aģentus, tādā veidā padarot šo sistēmu dinamiskāku. Ņemot vērā šo analoģiju, varam droši teikt, ka programmizstrādes vide ir dinamiska un pret to ir pilnā mērā jāattiecas kā pret dinamisku vidi.

2. Spējās metodes un prakses

Šajā nodaļā ir informācija par spējām metodēm un praksēm, kuru AI ir augstāks par 0 pētījuma periodā. Dažām no metodēm *DSDM* un *Crystal* pēdējos divos gados AI vērtība ir 0, bet informācija par tām tiek iekļauta, lai sniegtu lasītājam priekšstatu par to, ka aprakstītajās metodēs ir līdzīgas iezīmes, neskatoties uz to, ka interese par viņām ir mazinājusies.

2.1. Metodes

Programmatūras izstrādes uzņēmi parasti par pamatu spējai izstrādei izmanto, kādu no spējām metodēm. Spējās metodes izvēle tiek atstāta uzņēmuma ziņā.

2.1.1. *Scrum*

Ņemot par pamatu aprēķinātās AI vērtības, metode ar augstāko AI ir *Scrum* (sk. 1.1. tabula).

2.1.1.1. Vispārīga informācija

Scrum, kā spējā metode ir iteratīva [22]. Pirmo reizi tās pielietošana ir minēta 1987. gadā [18], un pirmie to izmantoja *Ikujiro Nonaka* un *Hirotaka Takeuchi*. *Scrum* ir paredzēts, lai ātri reaģētu uz izmaiņām. Un divi galvenie *Scrum* stūrakmeņi ir pielāgošanās spējas un komandas iespējas. Lai nodrošinātu šīs pielāgošanās spējas, *Scrum* nosaka dažādus likumus un noteikumus, piemēram, komandai ir jābūt mazai, un komandu iterācijas laikā nav ieteicams pārtraukt.

Kā redzams 2.1. att. , *Scrum* izstrāde sastāv no iteratīviem posmiem, kurus metode dēvē par „Sprintiem". Katrs sprints ilgst 2 līdz 4 nedēļas, sprinta garumu definē komanda. Programmas iespējas, kuras tiks implementētas konkrētajā sprintā, nāk no "Produkta darāmo lietu saraksta" (angl. *Product Backlog - PB*), kurā sakārtotā veidā atrodas "Lietotāju stāsti" (angl. *User story - US*) [21][22].

Liela nozīme *Scrum* metodē ir sanāksmēm [21][22]. Sanāksmēm atkarībā no to veida ir dažādi raksturlielumi. Izšķir 5 sanāksmju veidus:

- **ikdienas sapulces (angl. *Daily Scrum*)** :
    - sprinta darbības laikā Ikdienas sapulces norisinās katru dienu un to var uzskatīt par projekta statusa sanāksmi (sk. 2.1. att.). Sekojošas vadlīnijas ir jāievēro veicot ikdienas statusa sanāksmes:

        - sanāksmei vienmēr jāsākas noteiktā laikā;
        - ierasties var jebkurš, bet runā tikai pamata lomu pārstāvji;
        - sanāksmes ilgums ir 15 minūtes;
        - sanāksmei jānotiek vienā un tanī pašā vietā, un vienā un tanī pašā laikā.
    - sanāksmes laikā katrs komandas dalībnieks atbild uz trīs jautājumiem:
        - ko tu esi izdarījis kopš vakardienas?
        - ko tu plānot darīt šodien?
        - vai eksistē kādas problēmas, kuras var traucēt sasniegt uzstādītos mērķus (*ScrumMaster* pienākums ir atrast norādītajai problēmai risinājumu. Risinājums tiek meklēts ārpus Ikdienas sapulces laika, un tas ir saistīts ar Ikdienas sapulces garuma ierobežojumu - 15 minūtes).
- **sprinta plānošanas sanāksme (angl. *Sprint Planning Meeting*)** – katra sprinta sākumā [21][22][23] tiek novadīta sprinta plānošanas sanāksme, kuras mērķis ir:
    - noteikt darbu, kas tiks paveikts plānojamā sprintā;
    - sagatavot „Sprinta darāmo lietu sarakstu" (angl. *Sprint Backlog - SB*), kurā definēts, cik laika nepieciešams katrai izvēlētajai funkcionalitātei;
    - identificēt un apņemties izpildīt izvēlēto darba apjomu, kurš ir paveicams sprintā;
    - ievērot laika ierobežojumus:
        - pirmajās stundās (~4h) Produkta Īpašnieki (angl. *Product Owner - PO*) un komanda dialoga veidā sakārto PB pēc prioritātes;
        - nākamajās stundās (~4h) komanda izmantojot izveidoto PB izveido SB.
- **sprinta pārskata sanāksme (angl. *Sprint Review Meeting*)**:
    - pārskatīt paveikto un nepaveikto darbu sarakstu;
    - prezentēt paveikto darbu ieinteresētajām pusēm;
    - nepabeigtais darbs nevar tikt demonstrēts;

    - sprinta pārskats ilgst ~4h.
- **sprinta retrospektīva (angl. *Sprint Retrospective*):**
    - komanda pārrunā iepriekšējā sprintā padarīto;
    - mēģina veikt regulārus procesa uzlabojumus;
    - komandai ir jāatbild uz diviem galvenajiem jautājumiem:
        - kas iepriekšējā sprintā bija labi?
        - ko var uzlabot nākamajā sprintā?
    - retrospektīvas sapulces garums ir ~3h.
- **sagatavošanas sapulce (angl. *Grooming meeting*)** – norisinās paralēli izstrādes procesam. Sapulces mērķis ir sagatavot US tādā līmenī, lai tos varētu iekļaut PB. Sapulcē ir ieteicams piedalīties visai komandai, jo savlaicīgi izrunājot un vērtējot US, tie PB sarakstā nonāks labāk sagatavoti, un komanda daļēji būs ar tiem iepazinusies. Iepriekšējā iepazīšanās ar US palīdz novērst situāciju, ka komanda par US neko nezina un ir neprecīzi nepieciešamā laika novērtēšanā.

Scrum ir vairāk orientēts uz komandām, kuras atrodas vienā vietā un var brīvi komunicēt. Ir organizācijas un komandas, kuras piekopj *Scrum* arī tad, ja atrodas dažādās vietās, šajā gadījumā ir papildu jāpiedomā pie efektīvas komunikācijas nodrošināšanas. *Scrum* metodes funkcionēšanai izdala vairākas lomas:

- ***Scrum* procesa vadītājs** (angl. ***ScrumMaster*)** – ir procesa vadītājs, kurš arī darbojas kā buferis starp komandu un pasūtītāju;
- **produkta īpašnieks (PO)** – pasūtītāja jeb biznesa puses pārstāvis;
- **komanda (angl. *Team*)** – komanda sastāv no cilvēkiem, kura veic patieso darbu, analīzi, izstrādi un testēšanu [21][22].

Lomas *Scrum* metodē var iedalīt divās daļās [21][22]:

- **pamatlomas** – lomas, kuras ir tiešā veidā iesaistītas projekta izstrādē:
    - **produkta īpašnieks (PO)** – pārstāv klientu un ir atbildīgs par klienta vēlmju un biznesa vajadzību ievērošanu. Produkta īpašnieks definē klienta prasības, parasti lietotāja stāstu formā un sakārto tās pēc prioritātes PB sarakstā. PO lomu nav ieteicams apvienot ar *ScrumMaster* lomu;

- **komanda** – komandas atbildība ir produkta izstrāde. Komanda parasti sastāv no 5-9 cilvēkiem ar dažādām iemaņām, un ir patiesie darba darītāji, kas veic analīzi, izstrādi, testēšanu utt.. Komandai ir jābūt pašorganizējošai un pašvadāmai;
- ***ScrumMaster* –** darbojas kā buferis starp produkta īpašnieku un komandu. *ScrumMaster* sargā komandu no ārējas iedarbības, lai tā varētu pildīt uzliktos uzdevumus.

- **papildu lomas** - ir lomas, kuras tiešā veidā neietekmē izstrādes procesu:
  - **ieinteresētās personas (pasūtītāji, pārdevējs)** - tie ir cilvēki, kuri pasūta projektu un ir ieinteresēti projekta rezultātā, jo no tā gūs biznesa labumu. Ieinteresētās personas parasti piedalās sprinta pārskata sapulcēs sprinta beigās;
  - **vadītāji (ieskaitot projekta vadītāju) –** tie ir cilvēki, kas nodarbojas ar izstrādes vides veidošanu un sakārtošanu.

Lai nodrošinātu veiksmīgu *Scrum* metodes procesu, tiek izmantoti vairāki artefakti [21][22][23]:

- **produkta darāmo lietu saraksts (PB)** – augsta līmeņa saraksts, kurš tiek uzturēts visu projekta darbības laiku. Tas ir saraksts, kurā tiek apkopotas visas potenciālās iespējas sakārtotas pēc prioritātes (parasti ņemot vērā augstāko biznesa vērtību). Saraksts ir labojams un iekļauj virspusējus novērtējumus, cik konkrētā funkcionālā prasība prasīs laiku un kāda ir tās biznesa vērtība. Šādā veidā sagatavots saraksts palīdz PO sakārtot implementējamās funkcijas nākamajā sprintā. Katra elementa biznesa vērtību un prioritāti nosaka PO, bet nepieciešamo izstrādes laiku definē komanda. Saraksta sakārtošanai var izmantot atgriezeniskās investīcijas (angl. *Return on Investment - ROI*) metodi;
- **sprinta darāmo lietu saraksts (SB) –** sprinta darāmo lietu sarakstā ir tās funkcionālās prasības, kuras tiks ieviestas konkrētajā sprintā. Funkcionālās prasības tiek sadalītas uzdevumos, uzdevumu ieteicamais garums ir no 4 līdz 16 stundām [22]. Šāds detalizācijas līmenis palīdz komandai precīzāk izvērtēt nepieciešamo darba apjomu. Uzdevumi no sprinta netiek komandai piešķirti, tos izvēlas pašorganizētā komanda pati, ņemot vērā konkrēto uzdevumu īpašības. SB lietu saraksts ir komandas īpašums, un komanda nosaka katra elementa

izstrādei nepieciešamo laiku, kā arī maina uzdevumu statusus sarakstā pēc nepieciešamības;

- **sadegšanas diagramma (angl. *Burn Down Chart*) –** publiski pieejams saraksts ar elementiem, kurš norāda, cik daudz darba ir palicis sprintā. Saraksts tiek atjaunināts katru dienu un dod iespēju sekot līdzi sprinta progresam, un vai ir nepieciešamas kādas darbības veiksmīgai sprinta pabeigšanai.

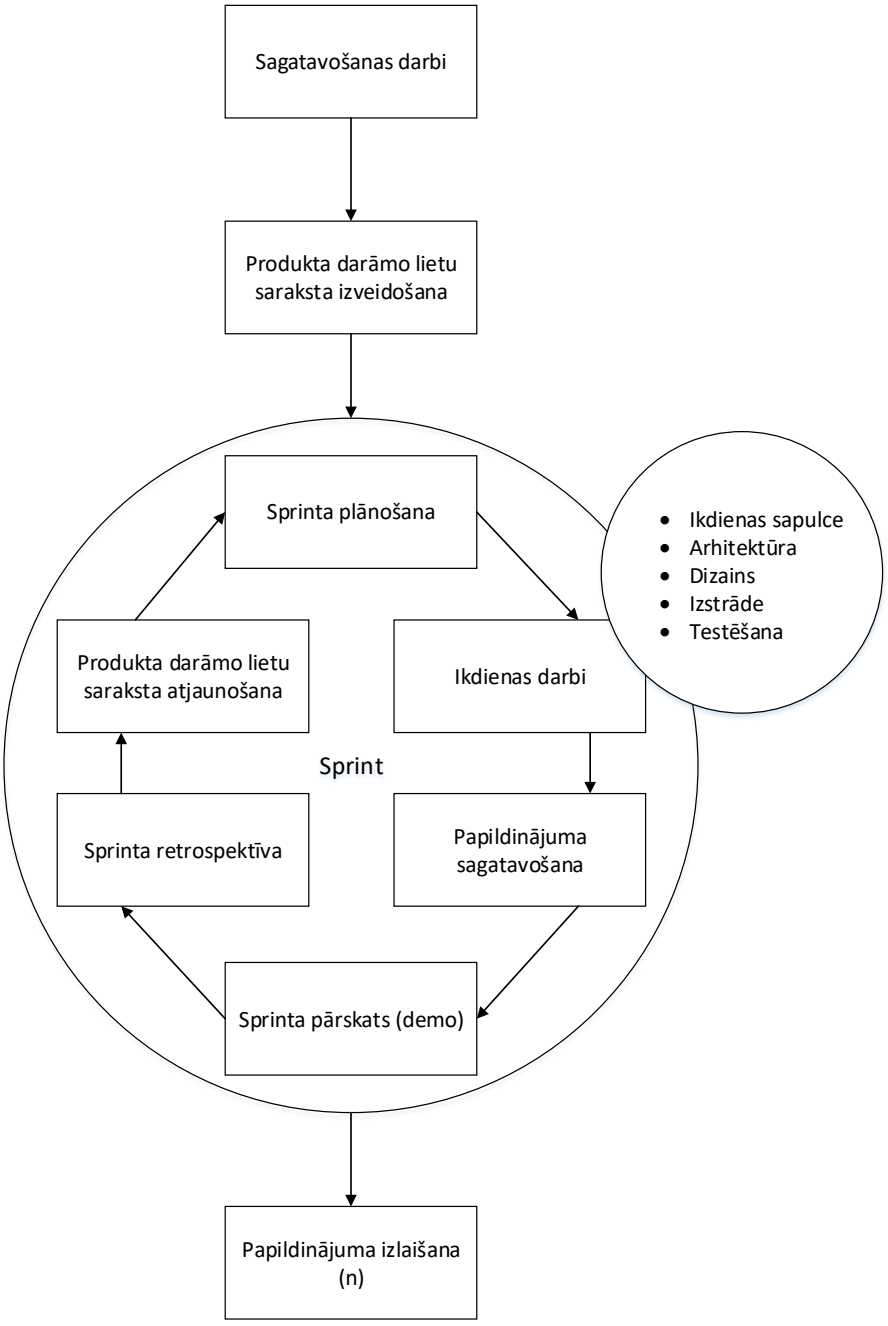
Norādītie saraksti nav vienīgie, eksistē arī *Release Burndown List, Earned Value Chart* utt., kas var palīdzēt sekot sprinta statusam dažādos griezumos.

Šajā apakšnodaļā ir apskatītas galvenās *Scrum* metodes iezīmes. Sīkāka informācija ir plaši pieejama internetā un literatūrā [21][22][23].

2.1.1.2. Process

Scrum metodes process sastāv no 9 pamataktivitātēm, kuras cieši saistītas ar iepriekšējā apakšnodaļā minētajām sapulcēm. Ir sastopamas dažādas *Scrum* procesa interpretācijas, no kurām darbā ir izveidota viena kopīga (sk. 2.1. att.).

Projekts, kurš tiks realizēts, izmantojot *Scrum* metodi, sākas ar sagatavošanas darbiem un vēlāk pāriet PB izveidošanas aktivitātē. Aktivitātes laikā norisinās vairākas apakšaktivitātes, tādas kā US sakārtošana pēc prioritātes, pieņemšanas kritēriju izveidošana utt.. Kad PB ir sagatavots un sakārtots, tad izveidotā komanda gatavojas iterācijai (“Sprintam”). Gatavošanās sprintam notiek aktivitātē “Sprinta plānošana”, kuras laikā komanda izlemj kādus US realizēt nākamajā iterācijā.



2.1. att. *Scrum* process.

Kad SB ir sagatavots, un uzdevumu saraksts izveidots, komanda var uzsākt darbu. Katrs komandas dalībnieks izvēlas sev atbilstošu US un uzsāk darbu. Ikdienas darba ciklā ir iekļauta sanāksme Ikdienas sapulce. Ikdienas darbu laikā tiek nodrošināta gan izstrāde, gan testēšana, jo pēc *Scrum* pamatprincipiem šiem procesiem ir jābūt nepārtrauktiem. Kad papildinājumi jaunajam laidienam ir sagatavoti, tiek organizēta sprinta pārskata sapulce, kuras laikā sagatavotie US tiek demonstrēti ieinteresētajām pusēm. Ja pēc demonstrācijas, daļa vai visi US tiek pieņemti, tie tiek iekļauti laidienā un atbilstošajā datumā pievienoti produkcijai. US, kuri nav pieņemti, atgriežas PB un

var tikt paņemti izstrādei nākamajā iterācijā. Pēc demonstrācijas komanda organizē retrospektīvas sapulci, kuras laikā tiek apspriests, kas iterācijā bija labi un kas slikti, lai nākamajās iterācijās nepieļautu tādas pašas kļūdas. Procesa beigās PB tiek modificēts un atjaunināts, lai nākamajā iterācijā šis saraksts būtu aktuāls. Paralēli standarta procesam norisinās sagatavošanas sapulce, kuras uzdevums ir sagatavot, un izvērtēt jaunos US, lai par tiem būtu zināma visa iespējamā informācija.

2.1.2. *Lean*

Balstoties uz vērtībām 1.1. tabula, metode *Lean* ieņem otro vietu pēc AI vērtības un pēdējos gados tiek izmantota arvien biežāk.

2.1.2.1. Vispārīga informācija

Minimāla izstrāde (angl. *Lean Software Development*) ir metode, kura balstīta uz 7 pamatprincipiem, kuri saistīti ar spējām praksēm [94]:

- novērst atkritumus (angl. *Eliminate Waste*):
    - ieraudzīt atkritumus – pirmais solis atkritumu novēršanā ir to identificēšana. Atkritumi ir daļēji pabeigts darbs, nevajadzīgi procesi, pārslēgšanās starp darbiem utt.;
    - vērtību plūsmas kartēšana (angl. *Value Stream Mapping*) – veids kā dokumentēt, analizēt un uzlabot informācijas plūsmu, kas nepieciešama, lai piegādātu produktu vai pakalpojumu.
- palielināt mācīšanos (angl. *Amplify Learning*):
    - atgriezeniskā saite – izmantojot atgriezenisko saiti iegūstam informāciju no lietotājiem, kas palīdz nākamajās iterācijās uzlabot produktu vai procesus;
    - iterācijas – iteratīva procesa nodrošināšana paātrina atgriezenisko saiti un informāciju no lietotāja ātrāk nonāk līdz izstrādātājiem;
    - sinhronizēšana – gadījumos, kad vairāki cilvēki vai komandas nodarbojas ar vienas lietas izstrādi, ir nepieciešama sinhronizēšana. Ir jānodrošina biežāki būvējumi. Būvējumiem un testiem ir jābūt automatizētiem. Indivīdu un komandu koda izmaiņām ir jābūt integrētām centrālā glabātuvē regulāri;
    - *Set-Based* izstrāde – gadījumos, ja jārisina sarežģīta problēma, ir nepieciešams izdomāt vairākas alternatīvas un definēt to ierobežojumus.

Izmēģinot alternatīvas, var izvēlēties kādu no tām, vai apvienot vairākas vienā, lai apmierinātu pasūtītāja prasības.

- izlem pēc iespējas vēlāk (angl. *Decide as late as possible*) – izstrādes laikā, komandas var saskarties ar problēmām, kad kāda no iespējām tiek izstrādāta sīkās detaļās, bet izrādās, ka augstākā līmenī risinājums vairs neder. Mērķis ir sākotnēji izprast visu funkcionalitāti augšējā līmenī, pirms doties sīkās detaļās. Izvēles iespējas ir jāatstāj tik ilgi, cik tas ir saprātīgi, ne ilgāk:
    - iespēju domāšana – ir saistīta ar vairāku iespēju izstrādi un izvēli;
    - pēdējais brīdis lēmumu pieņemšanai – ja ir iespēja, lēmums jāpieņem vēlāk, bet ne vēlāk kā brīdis, pēc kura var pazaudēt labu alternatīvu. Jāiemācās atrādīt līdz galam nepabeigts darbs, lai iegūtu ātrāku atgriezenisko saiti. Jāfokusējas uz to, kas konkrētajā brīdī ir būtisks;
    - lēmumu pieņemšana – ja ir iespējams, tad jāiemācās lēmumi pieņemt ne tikai racionāli, bet arī intuitīvi. To iespējams panākt, ja komandā ir eksperta līmeņa darbinieki.
- piegādāt pēc iespējas ātrāk (angl. *Deliver as fast as possible*) – jo ātrāk tiek piegādātas izmaiņas, jo labāk. Ātrāki laidieni sniedz ātrāku atgriezenisko saiti un ļauj klientam ātrāk atgūt iegūto naudu. Nedrīkst aizmirst, ka piegādāt ātri nenozīmē piegādāt sliktas kvalitātes darbu, runa ir par ātru vērtības piegādi lietotājam:
    - vilkšanas sistēma – tradicionālajās metodēs, vadītājs pasaka darbiniekam, kurš darbs jādara. Efektīvas komandas gadījumā, komandas dalībnieki paši zina, kuru no US viņi var izdarīt ātrāk un labākā kvalitātē;
    - rindu teorija – viens no rindas rādītājiem ir cikla laiks, kas nepieciešams elementa apstrādei. Jo īsāks ir šis laiks, jo labāk. Kad pieprasījums ir vienmērīgi sadalīts un atbilst iespējamai kapacitātei, cikla laiku ir iespējams samazināt. Viena no iespējām panākt vienmērīgu darba pieplūdi, ir veidot nelielus laidienus. Nelielus darbus ir vieglāk sadalīt izstrādātājiem un komandām, tādā veidā panākot to paralēlu izpildi. Gadījumos, ja kāda no komandām kavējas, liela daļa darba tiek vienalga paveikta;

    - kavēšanās izmaksas – izstrādājot jaunus produktus, vērā ņemams faktors ir produkta piegāde tirgū. Kādas būs izmaksas, ja mūsu produkts tirgū nonāks vēlāk? Aprēķinu veikšanai talkā var nākt ekonomiskie modeļi. Ekonomiskie modeļi palīdzēs izlemt, kas ir tas, ko nepieciešams piegādāt visātrāk.
- pilnvarot komandu (angl. *Empower the team*) – komandas pilnvarošana ir būtiska izmaiņa no tradicionālām izstrādes metodēm, kur lēmumu pieņemšana ir atstāta tikai vadītāju ziņā. Komandas pilnvarošanas gadījumā komandai ir jāļauj darīt to, ko viņi dara visslabāk:
    - pašnoteikšanās – dažreiz komandai ir jāpasaka vadībai, kā viņi gribētu strādāt un kas viņiem traucē veikt savu darbu;
    - motivācija – motivēts darbinieks ir labs darbinieks. Eksistē 4 motivācijas veidošanas stūrakmeņi: Piederība, Drošība, Kompetence, Progress;
    - līderība – katras labas komandas pamatā ir labs līderis. Tādam līderim nav pilnvaras pār cilvēkiem, kas strādā projektā, bet viņš palīdz izstrādāt sākotnējo konceptu un nosaka izstrādes tempu;
    - kompetence – jo augstāks ir komandas kompetences un zināšanu līmenis, jo labāk. Zinošākajiem izstrādātājiem ir jādalās ar savām zināšanām. Komandai ir jāievēro izstrādes standarti. Tie var būt vispārzināmie standarti vai standarti, par kuriem ir vienojusies pati komanda, labāk gan izmantot vispārzināmos kodēšanas standartus. Šādu standartu ievērošana palīdzēs gadījumos, kad komandai pievienojas jauni cilvēki.
- veidot integritāti (angl. *Build integrity in*) – integritātes nodrošināšanai ir jārūpējas, lai visas komponentes darboties viena ar otru. Jārūpējas par koda kvalitāti, lai jaunas funkcionalitātes pievienošana nebojātu jau esošo funkcionalitāti. Mērķa sasniegšanai var izmantot vispārzināmas izstrādes prakses, kā "Koda uzlabošana" (angl. *Refactoring*), automātiskie būvējumi, automātiskā testēšana, koda pieraksta vienkāršība utt.;
- redzēt visu kopumā (angl. *See the whole*) – redzēt visu kopumā nenozīmē ignorēt detaļas, bet saprast, ka izlaižot vai aizmirstot par kādu detaļu varam nodarīt vairāk ļauna. Piemēram, ja rodas kādas problēmas, vadība var likt

stingrāk ievērot kādus noteikumus, kuri patiesībā ir problēmu iemesls. Ļoti laba analoģija ir medicīnā, kur ārsti var ārstēt simptomus, nevis slimības cēloni. Mērķis ir tikt līdz problēmas cēloņiem, un viens no veidiem ir uzdot jautājumu "Kāpēc?" tik ilgi, kamēr iegūsim kādu saprātīgu atbildi:

- mērījumi – ir nepieciešams mērīt visu, kas ir svarīgi. To gan ir grūti panākt, bet svarīgi ir atcerēties, ka slikti mērījumi nodarīs vairāk ļauna nekā laba. Mērījumiem labāk būt apkopojošiem, nevis sadalītiem. Ja runa ir par sniegumu, tad svarīgāk ir vērtēt kopējo sniegumu, nevis individuālo;
- līgumi – spējo metožu kontekstā arī ir jāizmanto līgumi, tikai to saturam ir jābūt savādākam. Parasti ir 4 mainīgie: laiks, izmaksas, kvalitāte un robežas. Spējo metožu gadījumā ir ieteicams fiksēt pirmos trīs, nevis robežas [94]. Iespējām jābūt kārtotām pēc svarīguma, bet precīzam to sarakstam nevajadzētu būt iekļautam līgumā. Ja iespēju saraksts ir labi sakārtots, klientam svarīgās iespējas tiks piegādātas, pirms viss saraksts būs realizēts. Pēc *Standish Group* pētījumu rezultātiem līdz 45 % iespēju netiks izmantotas ikdienā [94].

2.1.2.2. Process

Neeksistē konkrēts *Lean* izstrādes process. Izstrādes process var tikt organizēts kā metodē *Scrum* vai praksē *Kanban*. *Lean* izstrādes galvenais pamats ir ievērot 7 pamatprincipus un spējās prakses, kuras tiek izmantotas pamatprincipu nodrošināšanai. *Scrum* procesa modelis ir aprakstīts apakšnodaļā *Scrum*. Informāciju par *Kanban* var atrast apakšnodaļā par praksēm.

2.1.3. Ekstrēmā programmēšana (XP)

Ekstrēmā programmēšana ir viena no pirmajām spējām metodēm, bet laika gaitā savas pozīcijas ir zaudējusi (sk. 1.1. tabula). Vērtīgākās prakses no šīs metodes ir pārnākušas uz jaunākām spējām metodēm. Konferencēs dažreiz ir minētas hibrīdmetodes, kuras ir *Scrum* un XP sajaukums.

2.1.3.1. Vispārīga informācija

Ekstrēmā programmēšana ir programmizstrādes metode, kas paredzēta, lai uzlabotu programmatūras kvalitāti un spētu reaģēt uz mainīgajām klientu prasībām [17].

Galvenās XP raksturiezīmes ir [19][20]:

- programmatūra tiek piegādāta ātri. Tā tiek piegādāta īsās iterācijās;
- tiek izmantota papildinoša plānošana, kas ātri nodrošina kopējo plānu, kurš projekta izstrādes gaitā attīstās;
- ātri spēj pielāgoties jaunām funkcionālām prasībām, kuras izriet no jaunām biznesa prasībām;
- paļaušanās uz automatizētiem testiem, kurus izveido izstrādātāji un klienti, lai pārbaudītu programmatūras darbību, ļautu sistēmai attīstīties un atrastu kļūdas pēc iespējas agrāk;
- paļaušanās uz mutiskām komunikācijām, testiem un programmas kodu;
- paļaušanās uz mainīgu izstrādes procesu, kurš turpinās tik ilgi, cik sistēma darbojas;
- paļaušanās uz tuvu programmētāju sadarbību;
- paļaušanās gan uz izstrādātāja īstermiņa instinktiem, gan uz ilgtermiņa projekta interesēm.

Ekstrēmā programmēšana strādā ar četriem mainīgo veidiem [19][20]:

- izmaksas;
- laiks;
- kvalitāte;
- robežas (angl. *Scope*).

Manipulējot mainīgo vērtības, tiek mēģināts iegūt vēlamo rezultātu, piemēram, izdalot pārāk maz laika, nevarēsim iegūt vajadzīgo kvalitāti. Izmantojot mazākas robežas, iegūsim labāku kvalitāti izstrādājamajai funkcionalitāti, bet atklāts paliek jautājums, vai pietiek ar izstrādātajām iespējām, lai iegūtu vajadzīgo vērtību.

Sākotnēji XP manipulēja ar četrām vērtībām, vēlāk tika pielikta vēl piektā:

- komunikācija – bieža komunikācija palīdz identificēt pareizas biznesa vajadzības un pareizu izstrādes virzienu;
- vienkāršība – uzsvars tiek likts, uz vienkāršāko risinājumu. Papildu funkcionalitāte var tikt pievienota vēlāk, pēc nepieciešamības. Māksla ir nedarīt neko lieku, līdz brīdim, kad tas patiešām būs vajadzīgs;

- atgriezeniskā saite – informācijai par sistēmas darbību ir jābūt nepārtrauktai. Pateicoties komunikācijai un testēšanai visu laiku tiek iegūta informācija par sistēmas darbību;
- drosme – dažreiz ir nepieciešama drosme, lai pārrakstītu sliktu kodu un izmēģinātu kaut ko jaunu, kas var sniegt jūtami labākus rezultātus nākotnē;
- cieņa – ir jāciena savs un cita darbs, izstrādātājiem nekad nevajadzētu ievietot izmaiņas, kas potenciāli var sabojāt sistēmu. Katrs cenšas izdarīt savu darbu visaugstākajā kvalitātē.

Galvenie ekstrēmās programmēšanas pamatprincipi, kas var palīdzēt izstrādes procesā, ir [19][20]:

- ātra atgriezeniskā saite – informācija kas iegūta, ir ātri jāliek lietā, jo vairāk laika paliek no informācijas iegūšanas līdz ieviešanai, jo vairāk tā izkropļojas;
- uzņemties vienkāršību – jebkuru problēmu vajadzētu apskatīt pēc iespējas vienkāršāk;
- papildinošas izmaiņas – lielu izmaiņu veikšana uzreiz parasti noved pie problēmām, tādēļ izmaiņas labāk ieviest pakāpeniski;
- pielāgoties pārmaiņām;
- kvalitatīvi padarīts darbs – padarītajam darbam ir jābūt ar iespējami augstāku kvalitāti, kvalitāte nedrīkst būt zema.

Eksistē arī mazāk svarīgi ekstrēmās programmēšanas principi [19][20], bet to apskatīšana dotā darba kontekstā nav paredzēta.

Četras XP bāzes aktivitātes ir:

- kodēšana – dienas beigās ir jābūt tādam programmas kodam, kuru var notestēt un uzstādīt klientam;
- testēšana – uzrakstītajam kodam ir jābūt testējamam. Programmas iespējas, kuras nav notestētas, nevar uzskatīt par pabeigtām programmas iespējām, kamēr testēšanas rezultāti nav pierādījuši pretējo;
- klausīšanās – sākotnēji vajadzētu pieņemt, ka izstrādātāji neko nezina, un viņiem ir jāklausās klients un viņa biznesa problēmas, lai pēc iespējas labāk saprastu, kas viņam ir jāuzprogrammē;

- projektēšana – nepietiek ar to, ka notiek klausīšanās, programmēšana un testēšana. Ir problēmas, kuru realizācijai ir nepieciešama projektēšana un strukturizēšana. Labs sistēmas daļas projektējums ir tāds, kura izmaiņas neietekmē citas sistēmas daļas.

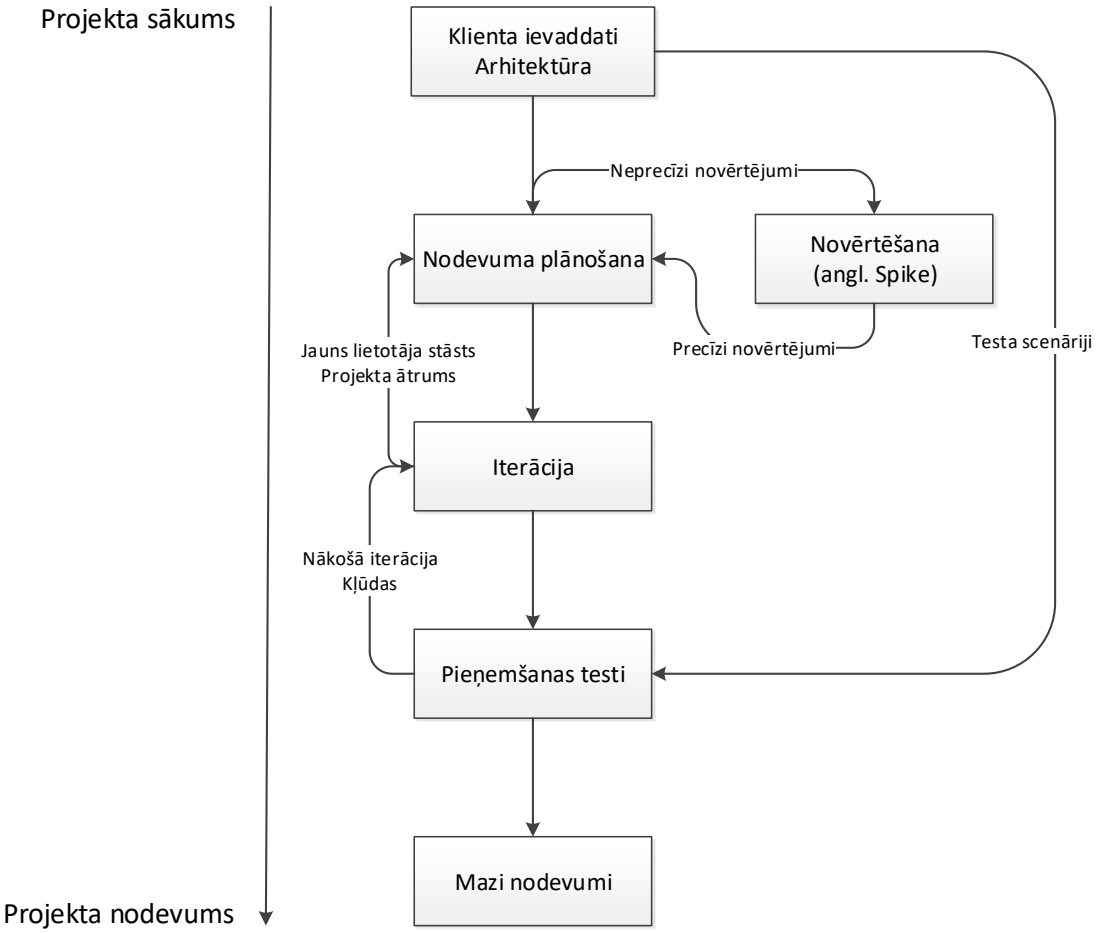
Svarīgākās XP prakses:

- plānošanas spēle – ir nepieciešams ātri noteikt nākamā laidiena robežas, kombinējot biznesa prioritātes un tehniskos novērtējumus. Ja realitāte maina sākotnējo plānu, plāns ir jāpielāgo. Spēles plānošana sastāv no trim fāzēm:
  - izpēte – noteikt kādas jaunas iespējas sistēmai ir nepieciešamas;
  - apņemšanās – noteikt kādas no definētajām iespējām izveidot nākamajā iterācijā;
  - vadīt – vadīt izstrādi, lai nonāktu līdz plānotajam rezultātam.
- nelieli laidieni – vienkāršs, strādājošs risinājums jāievieš produkcijas vidē ātri, jauni papildu laidieni ir jāievieš īsos izstrādes ciklos;
- vienkāršs projektējums – sistēma ir jāprojektē pēc iespējas vienkāršāk;
- testēšana – programmētāji raksta moduļu testus, kurus tie izmanto visu laiku. Klienti raksta testus, lai pārliecinātos, ka programmas iespējas ir pabeigtas;
- pārstrukturēšana – izstrādātāji pārstrukturizē kodu, nemainot sistēmas uzvedību, izvairoties no koda dublēšanas, ieviešot vienkāršību un pievienojot dinamiskumu;
- pāru programmēšana – svarīgas koda daļas ir iespējams programmēt divatā pie viena datora;
- kolektīvās īpašumtiesības – jebkurš var samainīt jebkuru kodu sistēmā, jebkurā vietā;
- nepārtraukta integrācija – koda būvējumam ir jānotiek vairākas reizes dienā, ik reizi, kad kāds uzdevums ir pabeigts;
- 40 stundu darba nedēļa – nestrādāt vairāk par 40 stundām nedēļā ir jābūt kā likumam. Nedrīkst strādāt virsstundas divas nedēļas pēc kārtas;
- klients uz vietas – visu laiku ir jābūt pieejamam klienta pārstāvim, kurš var atbildēt uz nepieciešamajiem jautājumiem;

- kodēšanas standarti – izstrādātāji raksta visu kodu atbilstoši kopīgi izstrādātiem noteikumiem, jo tas palīdz ātri izprast cita izstrādātāja izveidotu kodu.

2.1.3.2. Process

Ekstrēmās programmēšanas procesa modelis ir apskatāms zemāk (sk. 2.2. att.) un to var sadalīt 5 posmos [18]:



2.2. att. Ekstrēmās programmēšanas procesa modelis.

- **klienta ievaddati, arhitektūras smaile** (angl. *Architectural spike*), spike jeb smaile ir eksperiments, ko var izdarīt viens izstrādātājs – projekta sākumā, prasības tiek iegūtas no US. US ir funkcionālais apraksts lietām, ko klients vēlas, un ir veidots klientam saprotamā valodā. Papildu US tiek definēta sistēmas metafora, kas parastā valodā apraksta, kas sistēmai ir jādara;

- **laidienu plānošanas fāze** – izstrādātāji izskata US, un novērtē, cik daudz piepūles ir nepieciešams, konkrētā darba veikšanai. Klients veic US sakārtošanu pēc prioritātes. Sakārtotie US tiek izmantoti sākotnējā projekta plāna izveidošanai. Sākotnējs projekta plāns parasti nav ļoti precīzs, bet ar to pietiek, lai varētu sākt izstrādi. Projekta plāns izstrādes laikā mainīsies, ņemot vērā jauniegūtās zināšanas no katras iterācijas;
- **iterācija** – katra iterācija sākas ar konkrētās iterācijas plānošanu. Parasti iterācija nav garāka par divām nedēļām un iterācijas beigās tiek piegādāta strādājoša sistēma. Klients pastāsta izstrādātājiem par jaunām funkcionālām iespējām, kuras viņam nepieciešamas iegūt nākamajā iterācijā. Izstrādātāji sadala US darbos un veic darbietilpības novērtējumus, balstoties uz zināšanām, kas iegūtas iepriekšējās iterācijās. Darbi, kas palikuši pāri no iepriekšējās iterācijas, parasti tiek ieplānoti nākamajā iterācijā. Projekta ātrums (angl. *Project velocity*) tiek pielāgots, vadoties no jauniegūtās informācijas;
- **pieņemšanas testi** – klients testē konkrētā iterācijā piegādāto funkcionalitāti. Funkcionalitāte var būt gan pieņemta, gan noraidīta. Pieņemtā funkcionalitāte var tikt uzstādīta produkcijā. Nepieņemtā funkcionalitāte atgriežas atpakaļ izstrādē problēmu novēršanai;
- **nelieli laidieni –** sistēma tiek papildināta izmantojot nelielus laidienus, kuri var tikt uzstādīti ik pēc divām nedēļām.

2.1.4. Iezīmju vadītā izstrāde (FDD)

Iezīmju vadītā izstrāde salīdzinājumā ar tādām metodēm kā *Scrum, Extreme programming* vai *Lean* tiek izmantota ievērojami retāk (sk. 1.2. att. un 1.1. tabula).

2.1.4.1. Vispārīga informācija

Līdzīgi kā pārējās spējās metodes, Iezīmju vadītā metode (angl. *Feature Driven Development*) arī ir iteratīva [24][25]. FDD ir vadāms īsu iterāciju process, kurš sastāv no piecām pamata aktivitātēm. Par svarīgākajām FDD raksturiezīmēm tiek uzskatītas:

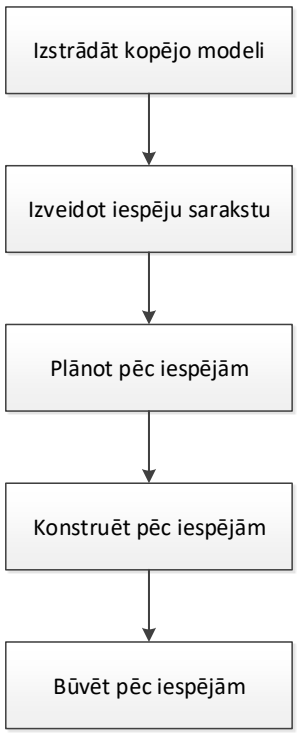
- **robežatzīmes** – Neskatoties uz to, ka iespēja var būt neliela un tās realizācijai pietiek ar nelielu uzdevumu, ir nepieciešams sekot iespējas izstrādes statusam. FDD definē sešus robežpunktus, kuri secīgi ir jāsasniedz katrai iespējai:
    - projektēt balstoties uz iespējām:

        - problēmdomēna aplūkošana (angl. *Domain Walkthrough*) – 1%;
        - projektēšana (angl. *Design*) – 40%;
        - projektējuma pārbaude (angl. *Design Inspection*) – 3%.
    - izstrādāt balstoties uz iespējām:
        - izstrāde (angl. *Code*) – 45%;
        - izstrādes pārbaude (angl. *Code Inspection*) – 10%;
        - iekļaušana kopējā projektā (angl. *Promote To Build*) – 1%.
- **labākās FDD prakses** – par labākajām FDD programmizstrādes praksēm tiek uzskatītas [24][25][27]:
    - **domēna objektu modelēšana (angl. *Domain Object Modeling*)** – tiek apskatīts problēmsfēras domēns un izveidots domēna objektu modelis, kas nodrošina kopējo ietvaru, kurā iespējas tiks pievienotas;
    - **izstrāde balstoties uz iespējām (angl. *Developing by Feature*) –** sistēma tiek sadalīta funkcijās. Ja kāda funkcija ir pārāk liela, lai to izstrādātu 2 nedēļu laikā, tā tiek uzskatīta par sarežģītu un tiek sadalīta sīkāk, kamēr iegūstam izstrādājamo iespēju sarakstu;
    - **individuāla klašu piederība (angl. *Individual Class Ownership*) –** individuāla klašu piederība nodrošina, ka par vienu klasi ir atbildīgs viens cilvēks. Klases īpašnieks ir atbildīgs par klases konsistenci, ātrdarbību un tās integritāti ar pārējām komponentēm;
    - **iespēju komandas (angl. *Feature teams*) –** iespēju komandas ir nelielas, dinamiskas izstrādātāju komandas, kas izstrādā atbilstošo iespēju. Iespējas komanda pieņem lēmumus par izstrādājamo iespēju kopā, izvēloties atbilstošāko izstrādes veidu;
    - **pārbaudes (angl. *Inspections*) –** regulāras pārbaudes tiek veiktas, lai nodrošinātu izstrādājamās iespējās augstāko kvalitāti;
    - **konfigurācijas pārvaldība (angl. *Configuration Management*) -** palīdz identificēt kodu katrai iespējai, iegūt informāciju par rakstītā koda vēsturi un vēsturiskajām izmaiņām tajā;
    - **regulāri būvējumi (angl. *Regular Builds*) –** nepieciešamība regulāri veikt koda būvējumus, lai pārliecinātos, ka jaunās funkcionalitātes pievienošana nav salauzusi būvējumu;

- **progresa rezultātu pieejamība (angl. *Visibility of progress and results*) –** izmantojot regulāras un precīzas atskaites, kuru pamatā ir padarītais darbs, vadība ir vienmēr informēta par darbu progresu.

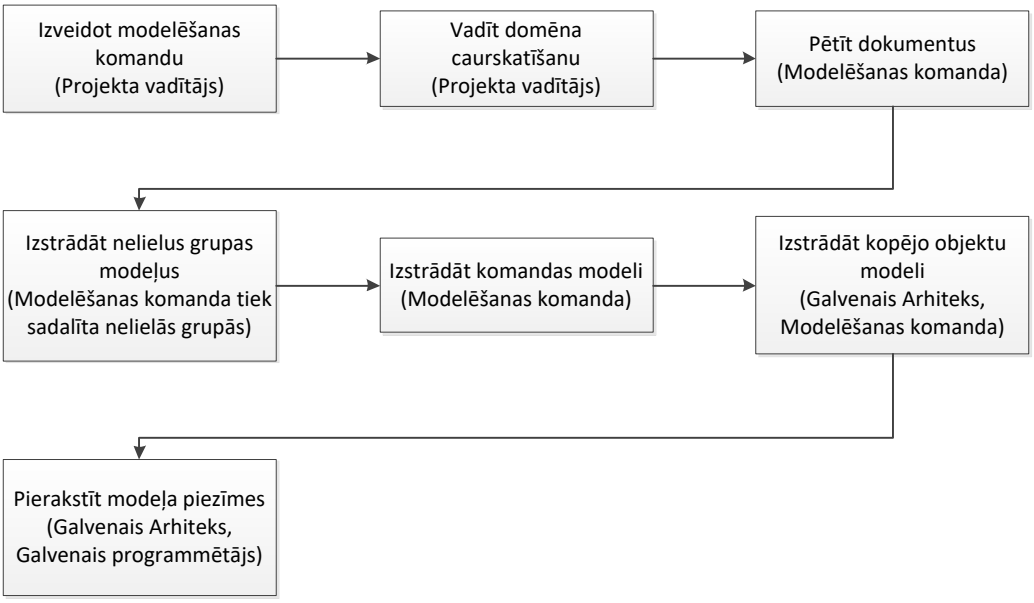
2.1.4.2. Process

FDD process sastāv no 5 pamata aktivitātēm. (sk. 2.3. att.) [24][25]. Pirmo trīs aktivitāšu laikā tiek izstrādāts kopējais sistēmas modelis. Pārējās divas aktivitātes tiek izmantotas, lai izveidotu katru atsevišķo iespēju.



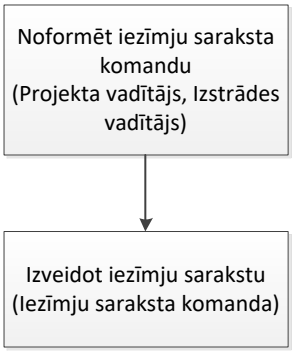
2.3. att. FDD procesa modelis.

Vispārīgā modeļa izstrāde (angl. *Overall Model Development - OMD*) – projekta sākumā tiek analizētas augsta līmeņa prasības sistēmas modeļa izveidošanai. Nākamajos soļos tiek veikta detalizēta domēna analīze. Detalizētu analīzi veic katram modelējamam apgabalam. Tiek izstrādāti vairāki modeļi katram domēnam. Diskutējot un modificējot sākotnējos modeļus, tiek izveidots piemērotākais, kurš vēlāk tiek izmantots par atbilstošā domēna pamatu. Domēna apgabalu modeļi tiek ievietoti kopējā modelī. OMD process ir apskatāms 2.4. att. [26].



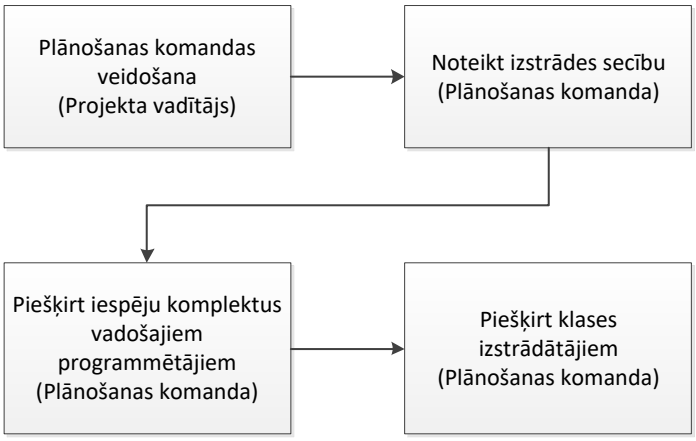
2.4. att. Vispārīgās modeļa izstrādes process.

Iespēju saraksta veidošana (angl. *Building of Feature List*) – zināšanas, kuras tiek iegūtas kopējā modeļa veidošanas laikā, tiek izmantotas, lai identificētu iespēju sarakstu (angl. *Feature List - FL*). Katrs domēnu apvidus (angl. *Subject areas*) ietver biznesa aktivitātes (angl. *Business activities*), soļi no katras biznesa aktivitātes veido sakārtotu iespēju sarakstu (sk. 2.5. att.). Iespējas šajā kontekstā ir klientam vērtīgas funkcijas (angl. *Client-valued functions*), kuras ir definētas sekojošā formātā: <darbība> <rezultāts> <objekts>, piemēram, „Aprēķināt kopsummu pārdošanām". Katras iespējas izstrāde nedrīkst aizņemt vairāk par 2 nedēļām. Gadījumos, kad kāda iespēja aizņem vairāk laika, tā tiek sadalīta sīkākās iespējās [26].



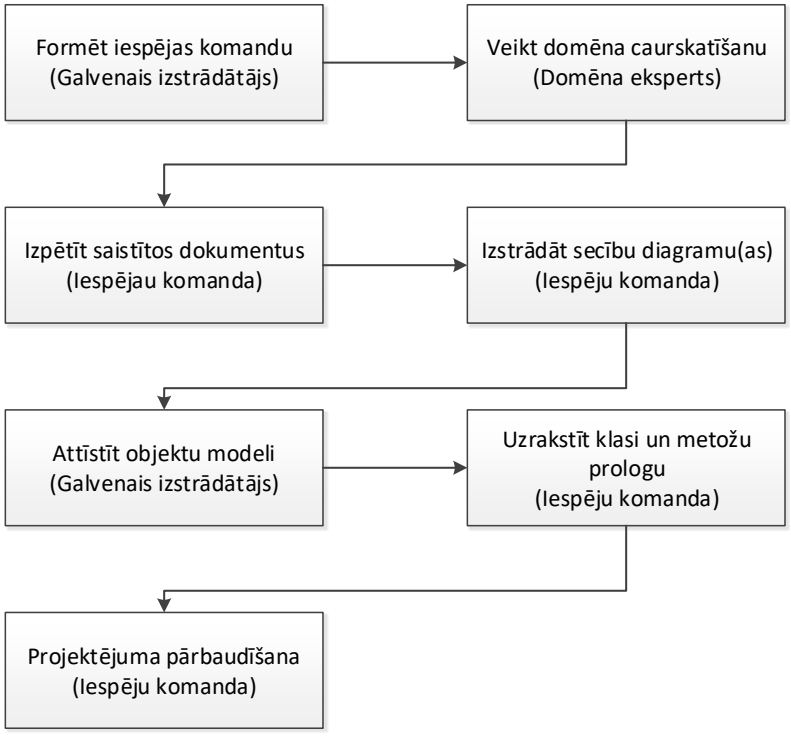
2.5. att. FL veidošana.

Plānošana balstoties uz iespējām (angl. *Plan by Feature*) – kad FL ir pabeigts, nākamais solis ir izstrādāt izstrādes plānu. Iespējas tiek sakārtotas un piešķirtas vadošajiem izstrādātājiem kā klases (sk. 2.6. att.) [26].



2.6. att. Plānot balstoties uz iespējām.

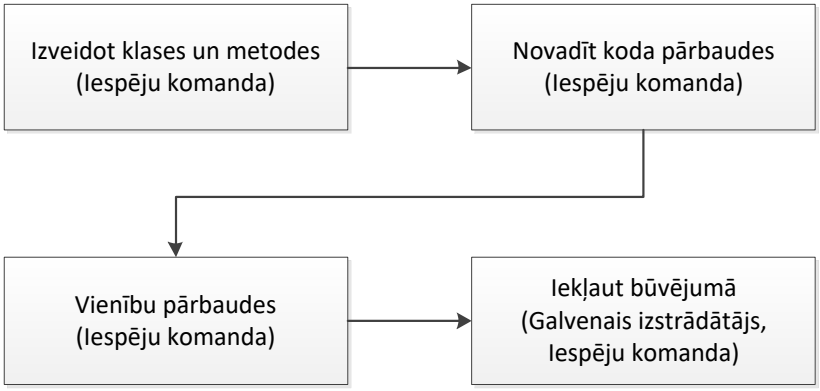
Projektēt balstoties uz iespējām (angl. *Design by Feature*) – vadošais izstrādātājs, izvēlas nelielu iespēju grupu, kuru ir iespējams izstrādāt 2 nedēļās. Kopā ar atbilstošās klases īpašnieku, vadošais izstrādātājs izstrādā detalizētu secību diagrammu (angl. *Sequence diagram*) katrai iespējai un papildina kopējo modeli. Nākamais solis ir klašu un metožu prologu pierakstīšana un pārbaude (sk. 2.7. att.) [26].



2.7. att. Projektēt balstoties uz iespējām.

Izstrādāt balstoties uz iespējām (angl. *Build by Feature*) – pēc veiksmīgas projektējuma pārbaudes, katra iespēja tiek izstrādāta. Klases īpašnieks veic atbilstošās

klases programmēšanu. Pēc veiksmīgas, vienību testu (angl. *Unit test*) un koda kvalitātes pārbaudes pabeigšanas iespēja tiek iekļauta kopējā būvējumā (sk. 2.8. att.) [26].



2.8. att. Izstrādāt balstoties uz iespējām.

Procesa beigās tiek iegūts papildinājums, kurš ir iekļaujams laidienā un kuru ir iespējams izvietot produkcijā.

2.1.5. *Dynamic system development method* (DSDM)

Pētījuma periodā metodei DSDM ir zems AI un tā vērtība 2012. gadā ir 0 (sk. 1.1. tabula). Spriežot pēc informācijas konferencēs un citos avotos, metode praktiski netiek vairs izmantota un interese par viņu ir zudusi.

2.1.5.1. Vispārīga informācija

DSDM ir ietvars, kurš paredzēts gan spējai, gan tradicionālai izstrādei [93]. DSDM būtiskākās raksturiezīmes ir:

- mērķa sasniegšanas orientēts ietvars, kura pamatā ir labākās prakses projekta realizēšanai;
- vienkāršs;
- paplašināms;
- netiek pozicionēts kā labākā metode visu projektu veidiem.

Salīdzinot DSDM ar tradicionālām izstrādes metodēm, lielākā starpība būs, ka izmantojot noteiktu laika sprīdi un resursus piegādājam mainīgu funkcionalitātes apjomu (izstrādājamās funkcionalitātes apjoms ir saistīts ar pieejamo laiku un resursiem).

Galvenie iemesli izmantot DSDM metodi ir līdzīgi tiem, kādi tie ir arī pārējām spējām metodēm (Pēc konferenču datu analīzes, izskatās, ka DSDM pēdējo gadu laikā nav piesaistījis izstrādātājus):

- izstrādes rezultāts ir ātri pieejams un redzams;
- lietotāji tiek vairāk iesaistīti izstrādes procesā, kā rezultātā lietotāji jauninājumus pieņem ātrāk un nav psiholoģiskā faktora, ka ar papildinājumiem neviens negrib strādāt vai tos pieņemt;
- pamatfunkcionalitāte tiek piegādāta ātri, pārējā funkcionalitāte tiek piegādāta regulāros intervālos;
- samazina birokrātijas apjomus un uzlabo komunikāciju starp iesaistītajām pusēm;
- funkcionalitāte tiek piegādāta regulāri un vairāk atbilst lietotāju prasībām;
- galalietotājiem ir lielāka ietekme pār to, kas tiks piegādāts.

DSDM pamatā ir 9 pamatprincipi, kurus veiksmīgai procesa nodrošināšanai ir jāizmanto:

- aktīva lietotāju iesaistīšanās;
- komandai ir jābūt tiesībām pieņemt lēmumus;
- nepieciešams nodrošināt biežus laidienus;
- jābūt izstrādātiem pieņemšanas kritērijiem;
- iterāciju izmantošana papildinošiem laidieniem ir obligāta;
- visām izmaiņām izstrādes procesā ir jābūt atgriezeniskām;
- prasības tiek ieskicētas augstā līmenī;
- testēšana ir integrēta visā dzīvesciklā;
- sadarbībai ir augsta nozīme.

Funkcionalitātes nodrošināšanai DSDM paļaujas uz 3 pamata praksēm, divas no kurām tiek plaši izmantotas arī citās spējās metodēs:

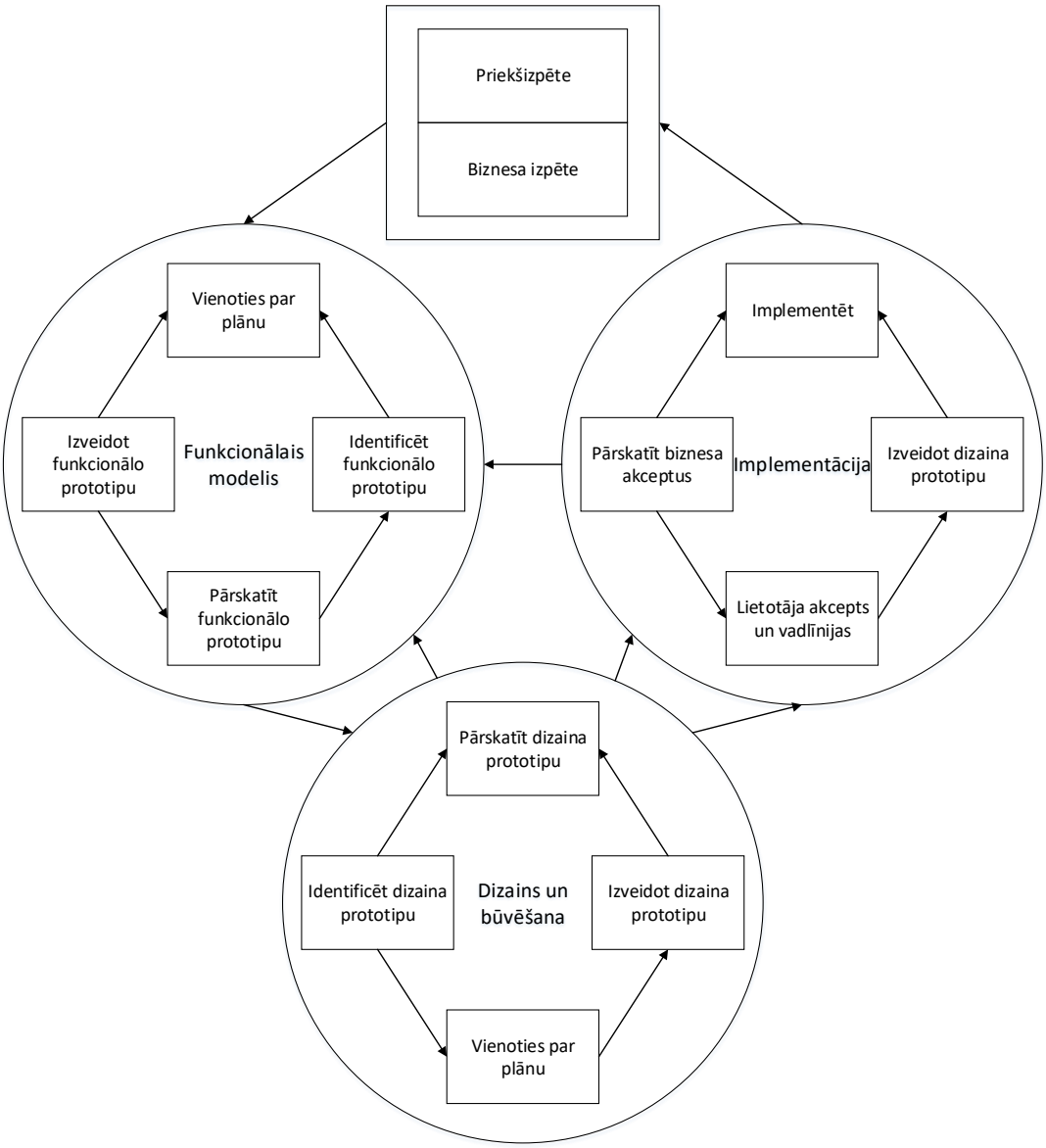
- ierobežošana laikā (angl. *Timeboxing*) – prakse, kura nosaka, ka visām aktivitātēm ir jābūt laikā ierobežotām. Piemēram, ja iterācijas laikā esam izvēlējušies realizēt konkrētu funkcionalitāti, bet, kādu daļu no funkcionalitātes nevar paspēt uztaisīt, tad šī funkcionalitāte tiek pārlikta uz nākamajām

iterācijām, jo izstrādes cikla garums netiks pagarināts. Noteiktā laika robežā ir jānodrošina, ka beigās tiks kaut kas piegādāts;

- MoSCoW likumi – palīdz iedalīt funkcionalitāti grupās:
    - Ir jābūt – visa funkcionalitāte no šīs grupas ir jārealizē, jo ja tas netiks izdarīts, sistēma vienkārši nedarbosies;
    - Vajadzētu būt – funkcionalitāte no šīs grupas ir ļoti svarīga, bet, ja rodas problēmas realizēt to laikā, tad to var izlaist;
    - Varētu būt – šāda funkcionalitāte papildina sistēmu, bet to var mierīgi realizēt nākamajās iterācijās;
    - Gribētos, lai būtu – funkcionalitāte no šīs grupas ir nepieciešama konkrētiem lietotājiem un tai ir neliela vērtība.
- prototipu veidošana – prakse tiek aktīvi izmantota un palīdz ātri pārbaudīt funkcionalitāti, jo svarīgās funkcionalitātes prototips tiek izveidots pirmais, un tas palīdz ātrāk iegūt atgriezenisko saiti no lietotājiem. Izšķir četrus prototipu veidus:
    - biznesa prototips – nodrošina sistēmas novērtējumu;
    - lietojamības prototips – tiek veiktas lietotāja saskarnes pārbaudes un lietojamības testēšana;
    - ātrdarbības prototips – tiek pārbaudīts, vai risinājums spēs nodrošināt nepieciešamo ātrdarbību;
    - spēju prototips - pārbauda un vērtē iespējamos risinājumus.

2.1.5.2. Process

DSDM process sastāv no 5 pamata aktivitātēm (sk. 2.9. att.). Papildu attēlā norādītājam eksistē vēl divas, “Pirms projekta aktivitātes” un “Pēc projekta aktivitātes”, kuras nav tik tieši saistītas ar pašu izstrādes procesu.



2.9. att. DSDM process.

DSDM pamata aktivitātes ir:

- priekšizpēte;
- biznesa izpēte;
- funkcionālā modeļa iterācija – notiek vienošanās par funkcionālo plānu, tiek izveidots funkcionālais modelis, iterāciju rezultātā, funkcionālais modelis tiek uzlabots;
- dizaina un būvēšanas iterācija – notiek vienošanās par projekta plānu, un tiek izveidots un apspriests projekta prototips, veikti nepieciešamie prototipa uzlabojumi;

- implementācijas iterācija – tiek implementēta apspriestā funkcionalitāte, un lietotājs to validē un pieņem. Var tikt pārskatīti biznesa pieņemšanas kritēriji.

2.1.6. *Crystal*

Pētījuma periodā metodei *Crystal* ir zems AI un tā vērtība 2012. gadā ir 0 (sk. 1.1. tabula). Spriežot pēc informācijas konferencēs un citos avotos, metode praktiski netiek vairs izmantota un interese par viņu ir zudusi.

2.1.6.1. Vispārīga informācija

Crystal Clear ir vairāku metožu apkopojums. Konferenču laikā pēdējos gados, metode nav izpelnījusies īpašu izstrādātāju interesi. Katra apakš metode tiek izmantota atbilstoša projekta veidam, vienīgais kopīgais ir apakš metožu pamatelementi. Projekti tiek iedalīti izmantojot krāsu kodus (clear, yellow, orange, red, utt). Kā arī projekti tiek iedalīti pēc riska pakāpes un svarīguma (quartz, diamond, utt.) [92].

Krāsu iedalījums iedala projektu pēc tajā iesaistīto cilvēku skaita. “Clear” ir komandām līdz 8 cilvēkiem. “Yellow” ir komandām no 10 līdz 20 cilvēkiem. “Red” ir komandām no 50 līdz 100 cilvēkiem un tā var turpināt līdz tumšākām krāsām un lielākiem komandu izmēriem.

Projektu iedalīšana pēc cietības ir saistīta ar to risku faktoriem. Piemēram, “Quartz” var atteicināt uz parastu rēķinu pārvaldības sistēmu, bet “Diamond” ir attiecināma uz tādu programmatūru, kas nodrošina atomreaktora darbību. Palielinoties cietībai, palielinās nepieciešamo validāciju un verifikāciju skaits.

Crystal pamatā ir ekonomiskās-sadarbības spēles modelis, kurš sastāv no:

- prioritātēm:
  - drošība projekta galarezultātā;
  - efektivitāte ar kādu tiek veikta izstrāde;
  - komandas vienošanās, ar kurām komanda var sadzīvot.
- atribūtiem:
  - biežas piegādes;
  - cieša sadarbība – paredz, ka informācijas plūsma komandā notiek fonā un komanda darbojas vienā kopējā telpā, lai nodrošinātu ātru informācijas apmaiņu. Piemēram, ja rodas, kāds jautājums komandas biedri vienkārši pagriežas viens pret otru un apspriež problēmu;

- regulāra uzlabošana – šajā kontekstā regulāra uzlabošana ir līdzīga retrospektīvai metodē *Scrum*, jo paredz izvērtēt konkrētās iterācijas darbību un nepieciešamības gadījumā paredz veikt izmaiņas, lai nākamā iterācija būtu veiksmīgāka nekā iepriekšējā;
- personiskā drošība – ir drošība runāt par kādu sasāpējušu problēmu, nebaidoties un neuztraucoties par kritiku. Piemēram, pateikt projekta vadītājam, ka ar atvēlēto laiku nav pietiekoši, lai pabeigtu kādu uzdevumu utt.;
- fokuss – pirmkārt fokuss ir zināt, kas jādara, otrām kārtām, vai ir atvēlēts vajadzīgais laiks darbības veikšanai, lai to varētu izpildīt bez lieka stresa. Uz fokusa atribūtu attiecas arī konteksta maiņa, kad darbiniekam liek pārslēgties no vienas iesāktas lietas uz citu un vēlāk atgriezties pie kādas no iesāktajām lietām;
- viegla piekļuve superlietotājiem (lietotāji ar lielu sistēmas lietošanas pieredzi). Ja tiek nodrošināti bieži laidieni un informācija no superlietotājiem ātri nonāk pie izstrādātājiem, tas lielā mērā var uzlabot produkta kvalitāti;
- tehniska vide ar automātisku testēšanu, konfigurācijas pārvaldību un biežu integrāciju.

- principiem – izmantojamie principi ir jāizvēlas atkarībā no projekta, jo katrā projektā izmantojamo principu kopa var atšķirties:
  - mazāk dokumentācijas;
  - cieša komandas sadarbība;
  - strādājoša programmatūra;
  - biežas piegādes;
  - citas komandas izvēlētas prakses.
- metodes – *Crystal* ietver sevī vairākas paraugmetodes, bet nepieprasa, konkrētas metodes izmantošanu.

Metodes un stratēģijas *Crystal* kontekstā ir pielīdzināmas praksēm citās metodēs. Tiek izceltas piecas stratēģijas:

- izpētes 360^0 – projekta sākuma stadijā tiek veikta izpēta dažādos virzienos. Piemēram, vai, izmantojot esošo tehnoloģiju, var sasniegt vēlamo rezultātu, kāda būs biznesa vērtība, kādas ir prasības, projekta plāns utt.;

- ātra uzvara – uzvara ir spēks, kas apvieno komandu un palielina pašpārliecinātību. Ir nepieciešams nodrošināties, ka darba gaitā komandai ir mazās uzvaras un kāds sasniedzams redzams rezultāts;
- "Staigājošs skelets" (angl. *Walking Skeleton*) – ir neliela sistēmas implementācija, kura pilnībā izpilda nelielu funkcionalitāti no sākuma līdz galam. Šajā gadījumā sistēmas arhitektūra varētu būt vēl nepabeigta, bet tai ir jābūt ļoti pietuvinātai tai, kāda tā būs projekta beigās;
- papildinoša arhitektūra – sistēmas arhitektūra izstrādes laikā var mainīties. Papildinošas arhitektūras pamatmērķis ir panākt, ka izmaiņas arhitektūrā tiek ieviestas pamata izstrādes laikā, nevis atsevišķi izdalītā procesā;
- informācijas radiatori – ir vietas, kur komanda var iegūt visu nepieciešamo aktuālo informāciju. Informācijas radiatori var būt gan papīra, gan elektroniskā formā, būtiskākais, lai tos ir viegli atjaunot;

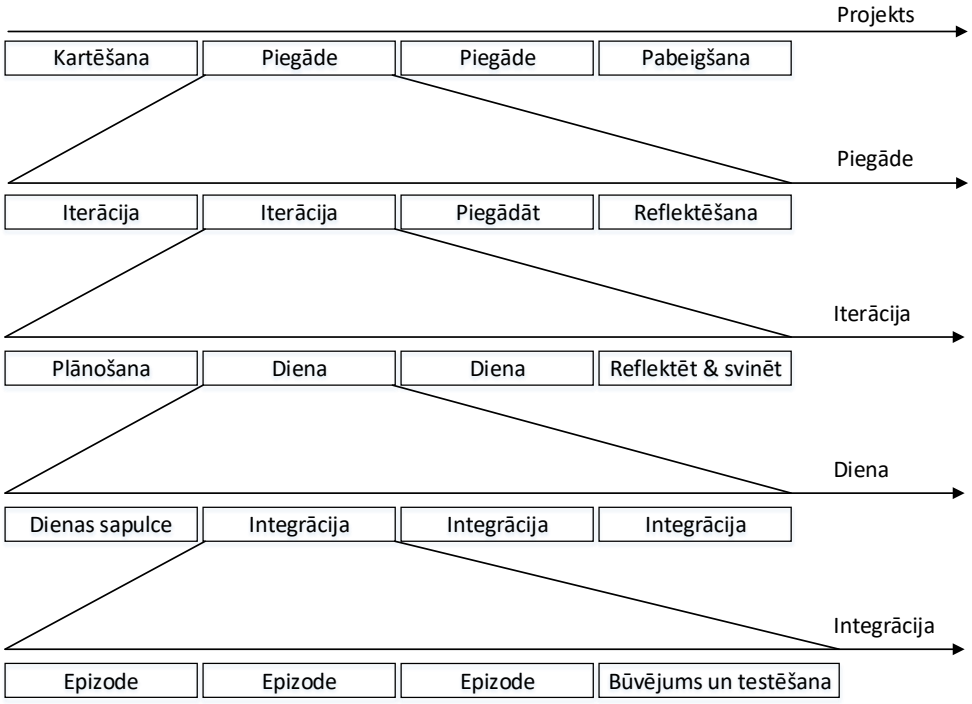
Papildu stratēģijām *Crystal* izmanto dažādas prakses. Zemāk ir minētas 9 ieteicamās prakses:

- metožu pielāgošana – izmantojot intervijas un savstarpējās sarunas, tiek izvēlēts, kādas metodes būs jāizmanto un ko tieši no metodēm ir jāizmanto. Kādas bija pozitīvās, un kādas negatīvās pieredzes izmantojot kādu no metodēm;
- pārdomu darbnīca (angl. *Reflection Workshop*) – ir prakse, kura līdzinās retrospektīvai, un tās darbības principi ir līdzīgi. Iterācijas beigās komanda satiekas un izrunā lietas, kuras bijušas labas un kuras sliktas;
- ātrā plānošana (angl. *Blitz Planning*) – *Crystal* metodes kontekstā "Ātrā plānošana" līdzinās "Grooming" sapulcei metodē *Scrum*, kuras ideja ir apspriest un novērtēt US, kurus būs jārealizē nākamajā iterācijā;
- DELPHI vērtēšana – sākoties projektam un projekta laikā pastāv mūžīgais jautājums, cik daudz laika un piepūles ir nepieciešams, lai realizētu kādu no US. Katru lietotāja stāstu izvērtē eksperti dodot tam vērtējumu. Ja ekspertu vērtējums atšķiras, ekspertiem ir jāpārrunā konkrētais US un jāvērtē atkārtoti, līdz brīdim, kamēr starpība vērtējumos ir neliela. DELPHI vērtēšanas prakse ļoti līdzinās "Poker game" praksei, kura tiek izmantota metodē *Scrum*;

- ikdienas sapulce (angl. *Daily Stand-Up*) – ikdienas sapulce ir sapulce, kura notiek katru dienu noteiktā laikā un ir ļoti īsa. Tāpat kā *Daily Scrum* tiek apspriesti trīs jautājumi:
    - Ko darīji vakar?
    - Ko darīsi šodien?
    - Vai eksistē kādas problēmas?
- spējais mijiedarbības dizains (angl. *Agile Interaction Design*) – ir veids kā panākt, ka sistēma ir lietojama. Prakse ir saistīta ar pieeju pie "superlietotājiem";
- procesu miniatūra (angl. *Process Miniature*) mācīšanās metode – darbs lielos projektos ir sarežģīts it īpaši jauniem izstrādātājiem, kuri tiek piesaistīti procesā vēlāk. Lai atvieglotu procesa izprašanu, tiek veiktas izstrādes procesa miniatūras sesijas, kuru laikā tiek veikts pilnīgs izstrādes cikls, tikai tas ir mazāks;
- blakus programmēšana (angl. *Side-by-Side Programming*) – programmēšanas veids līdzinās pāru programmēšanai, bet nav tieši tas pats. Pāru programmēšanas gadījumā divi cilvēki strādā pie vienas darbstacijas, bet blakus programmēšanas gadījumā izstrādātāji strādā katrs pie sava datora, bet pietiekoši tuvu, lai varētu ieskatīties viens otra datorā. Kā arī izstrādātāji strādā katrs pie sava uzdevuma, nevis pie viena kopīga;
- darba progresa grafiki (angl. *Burn Charts*) – dažāda veida grafiki, kuri palīdz noteikt iterācijas un projekta statusu.

2.1.6.2. Process

Šajā apakšnodaļā ir apkopota informācija par *Crystal* procesu un tā sastāvdaļām. Process ir hierarhisks un sastāv no pieciem līmeņiem jeb cikliem (sk. 2.10. att.).

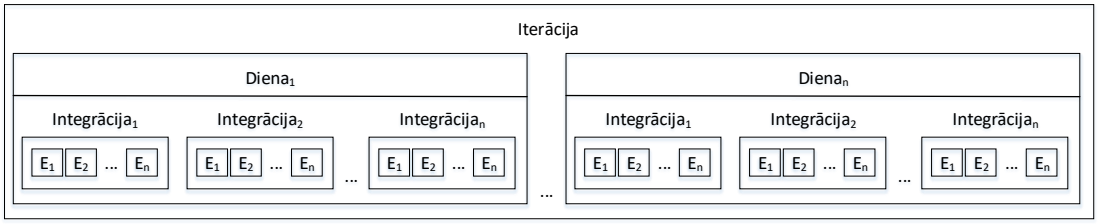


2.10. att. *Crystal* metodes process.

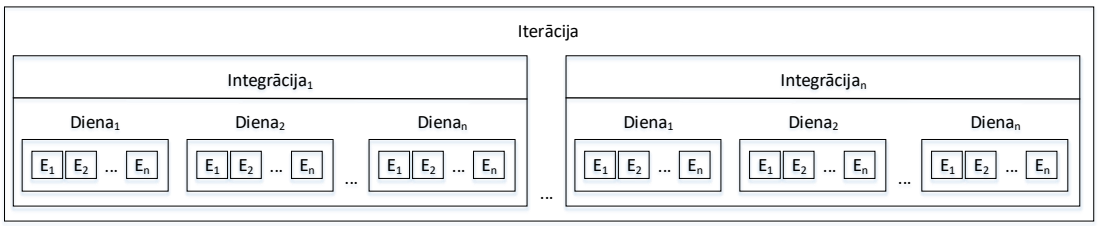
Augstākajā projekta ciklā ir projekta kartēšana, piegādes (piegāde$_1$, piegāde$_2$, ... piegāde$_n$) un projekta pabeigšanas aktivitātes.

Piegādes ciklā ir iterācijas (iterācija$_1$, iterācija$_2$ ... iterācija$_n$), piegādes un reflektēšanas aktivitātes.

Iterācijas ciklā ir plānošanas, dienas (diena$_1$, diena$_2$ ... diena$_n$) un reflektēšanas aktivitātes. Šajā līmenī ir iespējamas vairākas izmaiņas, jo iterācija var ilgt vairākas dienas vai iterācija var dalīties vairākās integrācijās (sk. 2.11. att. , 2.12. att.).



2.11. att. Iterācijas dalījums vairākās dienās.



2.12. att. Iterācijas dalījums integrācijās.

Dienas vai integrācijas cikls sastāv no vairākām epizodēm jeb lietotāju stāstiem, kuri attiecīgajā aktivitātē ir jārealizē un aktivitātes beigās jāveic būvējums un testēšana.

2.2. Prakses

Spējā metodoloģija nav iedomājama bez spējām praksēm. Spējās prakses ar augstāko AI ir apskatītas 1.4.2 apakšnodaļā. Detalizētāka informācija par trijām praksēm ir pieejama darba pamatteksā, bet par pārējām 3. pielikumā. Par visām praksēm ir iekļauts neliels apraksts, to angliskais nosaukums, zināmie sinonīmi, kā arī pozitīvās un negatīvās īpašības.

2.2.1. Pieņemšanas testu vadīta izstrāde (angl. *Acceptance Test Driven Development - ATDD*)

Analogs TDD, dotā prakse sastāv no automatizētiem pieņemšanas testiem [53]. Ideālā gadījumā, testu (angl. *Test case*) var uzrakstīt PO, klients vai domēna eksperts tam nepiesaistot izstrādātāju.

Sinonīmi:

- stāstu vadīta izstrāde (angl. *Story Test Driven Development - STDD*).

Pozitīvās īpašības:

- līdzīgi kā TDD palīdz programmām veidot vienumu testus, ATDD sniedz iespēju veikt funkcionālo testēšanu. (Testēšana izmantojot UI testus, ir uzskatāma par mazāk efektīvu).

Trūkumi:

- automatizētie pieņemšanas testi ir saistīti ar specifiskiem rīkiem, tādiem kā *Fit/FitNess, Cucumber* vai citiem, kā rezultātā ir nepieciešams iegūt papildu zināšanas par specifisko rīku, kas liedz produkta īpašniekam, klientam vai domēna ekspertam vieglā veidā veidot pieņemšanas testus.

2.2.2. Pieņemšanas testi (angl. *Acceptance testing*)

Pieņemšanas tests ir formāls programmatūras produkta uzvedības apraksts, izteikts kā pielietojuma gadījums [54]. Eksistē dažādi testu definēšanas veidi, kuros figurē kāds konkrēts rīks. Līdzīgi kā vienumu testos, pieņemšanas testu rezultāts var būt patiess vai aplams. Aplams rezultāts var norādīt uz kļūdas esamību, bet to pilnībā

nepierāda. Dažas spējās komandas izmanto pieņemšanas testus, kā vienīgo specifikācijas dokumentu, kas palīdz atbrīvoties no daļas formalitāšu.

Sinonīmi:

- funkcionālais tests (angl. *Functional test*);
- klienta tests (angl. *Customer test*);
- stāsta tests (angl. *Story test*).

Pozitīvās īpašības:

- uzlabo sadarbību starp izstrādātājiem no vienas puses un klientiem, lietotājiem un domēna ekspertiem no otras;
- tiek definēts skaidrs līgums starp klientu un izstrādātāju. Produkts, kura pieņemšanas testi iziet, var tikt uzskatīts par atbilstošu;
- samazina iespēju, ka nopietni defekti paliek nepamanīti.

Trūkumi:

- gadījumā, ja pieņemšanas testi ir uzrakstīti pārāk tehniski, tos būs ļoti grūti saprast klientiem un domēna ekspertiem, kā rezultātā šos testus sanāk uzturēt izstrādātājiem. Lai šo kļūdu nepieļautu, pieņemšanas testu veidošanā ir jāpiedalās domēna ekspertiem un klientiem;
- pieņemšanas testi var būt viegli sabojājami, piemēram, kādā formā ir samainījies teksts, pēc kura tika veikta meklēšana vai salīdzināšana.

2.2.3. Aktīva klienta dalība (angl. *Active Stakeholder Participation*)

Aktīva klienta iesaistīšana savā ziņā ir gan prakse, gan viens no spējās metodes stūrakmeņiem, jo bez aktīvas klienta iesaistīšanas spējās metodes var nesniegt vēlamo rezultātu. Aktīva klienta iesaistīšanās šajā kontekstā ir klienta piedalīšanās sapulcēs, kā arī klienta ātra pieejamība nepieciešamības gadījumā. Pastāv iespēja kādu no klienta pārstāvjiem projekta laikā pārvietot pie izstrādātājiem, lai informācijas apmaiņa būtu ātrāka un efektīvāka.

Sinonīmi:

- klients uz vietas (angl. *On site customer*).

Pozitīvās īpašības:

- iespēja ātri iegūt informāciju par biznesa prasībām;
- iespēja ātri veikt pieņemšanas pārbaudes no klienta puses;
- klients ir labi informēts par progresu un nepieciešamības gadījumā var palīdzēt pieņemt lēmumus, kas saistīti ar produktu.

Trūkumi:

- lielākā daļa klientu vienmēr nav gatavi, ļoti iesaistīties izstrādes procesā;
- klienta pārstāvis nejūtas omulīgi svešā kolektīvā;
- klienta pārstāvis var vēlēties iesaistīties pašā izstrādes procesā un mēģināt to kontrolēt, kas neatbilst spējās metodes pamatvērtībām. Šajā gadījumā ir jābūt labam *ScrumMaster*, kas neļauj ietekmēt izstrādes komandas darbu.

2.3. Secinājumi

Uzņēmumi un komandas izmanto dažādas spējās metodes. Daži uzņēmumi izmanto eksistējošas un aprakstītas metodes, bet ir arī tādi uzņēmumi, kuri savām vajadzībām ir atvasinājuši un izveidojuši savu metodi. Pētot informāciju, dažādos avotos ir izdevies secināt, ka dotajā brīdī biežāk izmantotā metode ir *Scrum*. Šī metode tiek izmantota arī visos trijos pārbaudes uzņēmumos, kuri aprakstīti 0. nodaļā.

Spējās metodes parasti netiek izmantotas vienas pašas. Vairumā gadījumu, to izmantošana ir saistīta ar vairāku spējo prakšu izmantošanu. Spējās prakses nodrošina veiksmīgāku metodes izmantošanu. Prakšu izvēle vairumā gadījumu tiek atstāta komandas ziņā, bet ir arī prakses, kuru izmantošana ir saistīta ar lēmumiem organizācijas līmenī. Gadījumos, ja kāda no praksēm nesniedz vēlamo rezultātu, tā tiek aizvietota ar kādu citu.

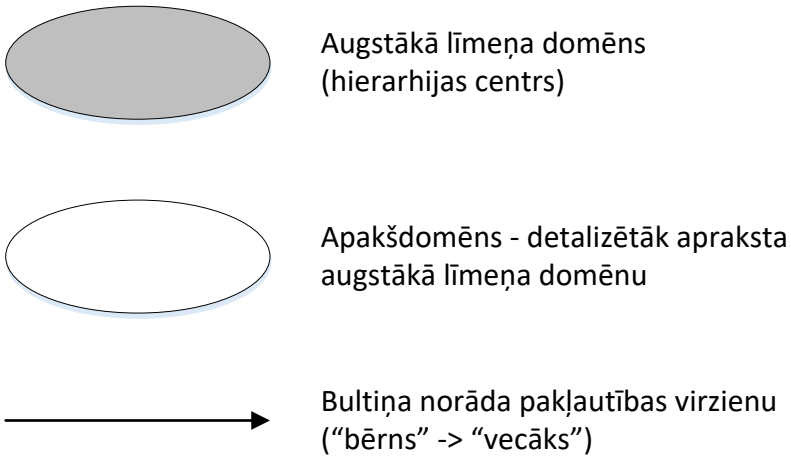
Sākotnējais prakšu saraksts sastāvēja no 176 spējām praksēm. Sarakstu apstrādājot un apvienojot līdzīgās prakses, tika iegūts saraksts ar 111 praksēm. Darba pamattekstā ir aprakstītas 35 prakses ar augstāko AI un to priekšrocības un trūkumi.

3. Organizācijas spējība

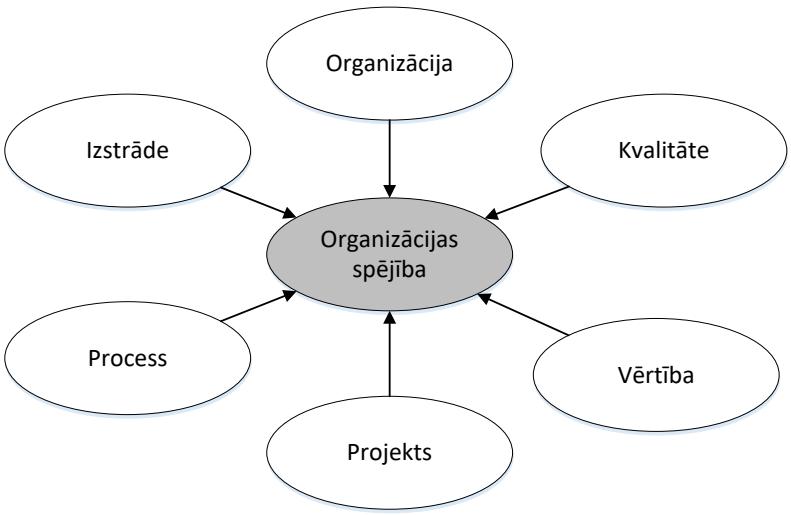
Organizācijas spējība ir programmizstrādes uzņēmuma spēja transformēties no tradicionālā izstrādes modeļa uz spējo izstrādes modeli un veiksmīgi veikt dažādu programmatūras izstrādes projektu realizēšanu, izmantojot kādu no spējām programmizstrādes metodēm, un nepieciešamības gadījumā ātri pielāgoties mainīgās vides mainīgajiem apstākļiem.

3.1. Organizācijas spējības modelis (OSM)

Organizācijām, kuras vēlas izmantot spējās metodes, ir svarīgi noteikt savu spējības līmeni. Spējības līmeņa noteikšana palīdz organizācijām saprast, kādā līmenī organizācijas atrodas, kādas papildu zināšanas jāiegūst, un kādas darbības ir nepieciešamas organizācijai veikt, lai veiksmīgāk ieviestu spējās metodes. Ja organizāciju aplūko kopumā, tad tās spējību noteikt ir sarežģīti, jo organizācijas ir sarežģīts un komplicēts veidojums. Darba ietvaros tiek piedāvāts organizāciju aplūkot no vairāku domēnu aspektiem un veikt katra domēna novērtēšanu atsevišķi. Darba kontekstā termins **domēns** apraksta kādu konkrētu programmatūras izstrādes organizācijas apgabalu. Piemēram, izstrādes domēns apraksta un analizē organizācijas apgabalu, kura galvenie elementi ir saistīti ar programmatūras izstrādi. Organizācijas spējības modelis (sk. 3.2. att.) sastāv no sešiem domēniem. Modeļa attēlošanai tiek izmantoti 3 veidu apzīmējumi (sk. 3.1. att.).



3.1. att. OSM modeļa attēlošanas shēmās izmantotie apzīmējumi.



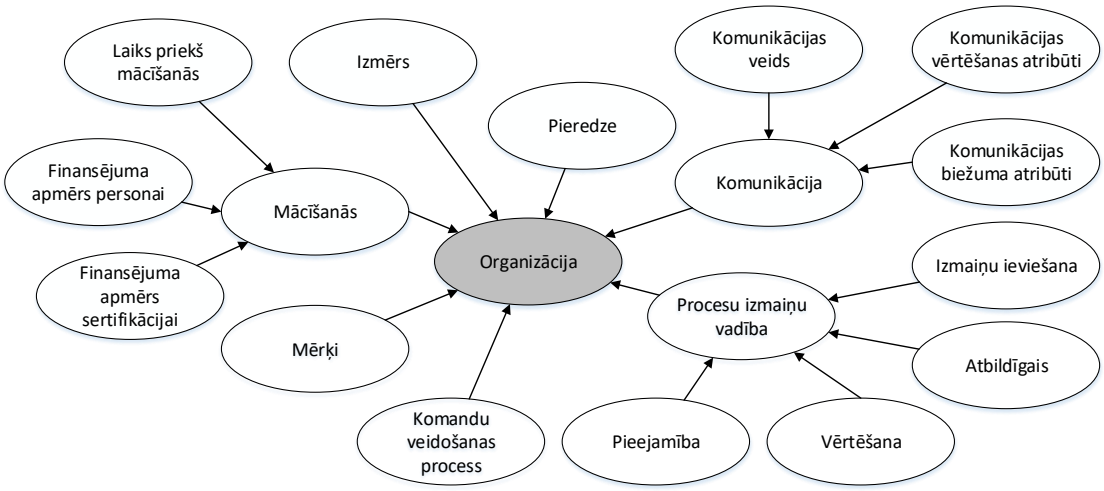
3.2. att. Organizācijas spējības modelis (OSM).

Darbā aprakstītais modelis veidots, balstoties uz autora pieredzi, un ir ievērojami papildināts ar informāciju, kura iegūta no ekspertiem, ar kuriem autors sadarbojās spējības ietekmes indeksa (SII) vērtības noteikšanas laikā. Pētot literatūras avotus, līdz līdzīgam augšējā līmeņa domēnu sadalījumam ir nonākuši arī pētnieki no *Scrum.org* [121].

Organizācijas domēns apraksta organizācijas apgabalu, kurš saistīts ar organizācijas kopējiem atribūtiem, piemēram, organizācijas izmēru vai organizācijas pieredzi ar spējām metodēm. Izstrādes domēns raksturo produkta izstrādes apgabalu. Kvalitātes domēns apraksta apgabalu, kura uzdevums ir parūpēties par veidojamā produkta kvalitāti. Procesu domēns apraksta *Scrum* metodes un tās elementu precīzu izmantošanu. Vērtību domēns apraksta elementus, kuri saistīti ar biznesa ieguvumiem, kuri tiks iegūti pēc kārtējā programmatūras laidiena. Projektu domēns apraksta elementus, kuri saistīti ar konkrēto projektu. Nākamajās nodaļās katrs no domēniem ir apskatīts sīkāk.

3.1.1. Organizācijas domēns

Organizācijas domēns raksturo organizācijas apgabalu, kura pārziņā ir nepārtraukta organizācijas atribūtu spējības izvērtēšana un uzlabošana mainīgā vidē (3.3. att.).



3.3. att. Organizācijas domēns.

Organizācijas domēns sastāv no astoņiem apakšdomēniem, bet var tikt paplašināts nepieciešamības gadījumā:

- procesu izmaiņu vadība ir nepieciešama, lai vērtētu procesu izmaiņu pārvaldību organizācijā. Procesu izmaiņu vadībai ir nepieciešams atbildīgais;
- komunikācija spēlē nozīmīgu lomu spējās. Ir nepieciešams izvērtēt gan iekšējo, gan ārējo komunikāciju, kā arī komunikāciju veidu, piemēram, komunikācija attālinātās komandās ievērojami atšķiras no komunikācijas vienā telpā, tiešā redzamībā;
- mācīšanās apakšdomēns raksturo organizācijas spēju mācīties un tas ir cieši saistīts ar zināšanu pārvaldības apakšdomēnu;
- organizācijas izmērs ir jāņem vērā, jo dažāda izmēra organizācijās var būt nepieciešamas dažādas spējās prakses;
- organizācijas pieredzei ir vērā ņemama loma tās spējības līmeņa noteikšanā, jo organizācijas, kuras izmanto spējās metodes ilgāku laiku un ir izmēģinājušas dažādas pieejas, var būt spējīgākas par tām, kuras tikai uzsākušas spējo metožu izmantošanu;
- komandu organizēšanas procesa apakšdomēns ir atbildīgs par komandu veidošanas procesa organizēšanu.

Katru domēnu un apakšdomēnu raksturo atribūti.

3.1.1.1. Izmērs

Izmēra apakšdomēns iedalās divās daļās:

- cilvēku skaits organizācijā;
- izstrādātāju skaits organizācijā.

Organizācijas izmēra atribūtu vērtības var atšķirties dažādos reģionos un valstīs. Darbā definētie izmēri ir Eiropas Savienībā noteiktie (sk. 3.1. tabula) [95].

3.1. tabula

Cilvēku skaits organizācijā

Nosaukums	**Papildu vērtība 1**	**Papildu vērtība 2**
Liela	x >250	>50 miljoni EUR
Vidēja	50 < x < 250	≤50 miljoni EUR
Maza	11 < x < 50	≤10 miljoni EUR
Mikro	x < 10	≤2 miljoni EUR

Izmēra apakšdomēnā vienīgais atribūts nav "organizācijas izmērs", apakšdomēna otrs atribūts ir izstrādātāju skaits organizācijā, kurš izteikts procentos no kopējā skaita. Lai "izstrādātāju skaits" atribūtu būtu vieglāk izmantot, veicot intervēšanu, tad ir izlemts veikt gradāciju solī pa 10%. Iespējamās vērtības ir: 10, 20, 30, 40 utt..

3.1.1.2. Mācīšanās

Organizācijas domēna kontekstā mācīšanās ir saistīta ar apmācības procesa organizēšanu organizācijā:

- darbiniekiem atvēlētais laiks mācīties;
- apmācību finansējums uz vienu darbinieku;
- sertifikācijas eksāmenu finansējuma apmērs.

3.2. tabula

Darbiniekiem atvēlētais laiks mācīties (stundas/gadā)

Vērtība	**Apraksts**
x = 0	Laiks netiek atvēlēts
1 < x < 8	Viena diena
9 < x < 16	Divas dienas
16 < x < 40	Viena nedēļa
40 < x < 80	Divas nedēļas
x > 80	Vairāk par divām nedēļām
Nav zināms	Finansējuma apmērs nav zināms

3.3. tabula

Apmācību finansējums vienam darbiniekam (EUR/gadā)

Vērtība
x = 0
1 < x < 100
100 < x < 200
200 < x < 300
300 < x < 400
400 < x < 500
500 < x 1000
1000 < x < 2000
2000 < x < 3000
x > 3000
Nav zināms

3.4. tabula

Sertifikācijas eksāmenu finansējuma apmērs vienam darbiniekam (EUR/gadā)

Vērtība
x = 0
1 < x < 100
100 < x < 200
200 < x < 300
300 < x < 400
x > 400
Nav zināms

3.1.1.3. Pieredze

Organizācijas kontekstā pieredze ir laiks gados, cik ilgi konkrētā organizācija izmanto spējās metodes. 3.5. tabula ir apkopotas iespējamās atribūtu vērtības, kuras ir jānoskaidro par katru metodi, kuras AI pēdējos gados ir lielāks par 0. Šajā gadījumā runa ir par *Scrum*, *Extreme Programming*, *Lean* un FDD.

3.5. tabula

Pieredze ar metodi (metode/gadi)

Vērtība (gadi)	Apraksts
x = 0	Spējā metode netiek izmantota
x < 1	Metode tiek izmantota mazāk par gadu
1 < x < 2	No 1 līdz 2 gadiem
2 < x < 5	No 2 līdz 5 gadiem
x > 5	Vairāk par 5 gadiem
Nav zināms	Darbinieks nezina cik sen un vai metode tiek vai tika izmantota

3.1.1.4. Komunikācija

Komunikācijas apakšdomēns sastāv no vairākām daļām:

- komunikācijas veids;
- komunikācijas vērtēšanas atribūti;
- komunikācijas biežuma atribūti.

Komunikāciju veids ir ļoti būtisks aspekts, jo komunikācija ir viens no spējo metožu stūrakmeņiem. Organizācijas kontekstā mēs apskatām pārsvarā ārējo komunikāciju, kas nav tiešā veidā saistīta ar izstrādi.

3.6. tabula

Komunikāciju veids

Nosaukums	**Apraksts**
Aci pret aci	Komunikācija notiek, kad abas puses atrodas vienā vietā
Telefoniski	Sarunas notiek telefoniski
Rakstiski	Komunikācija norisinās pārsūtot e-pastus vai vēstules
Skype vai alternatīva sarunu noorganizēšana	Pēdējā laikā aktīva komunikācija norisinās izmantojot Skype vai līdzīgu alternatīvu, kurā ir iespējams ne tikai rakstiski aprunāties, bet veikt arī dalīšanos ar ekrānu, kas ļoti atvieglo kļūdu demonstrēšanu, vai sarežģītāku procesu izskaidrošanu
Videokonferences	Mūsdienu aparatūra un interneta ātrums sniedz iespēju veikt augstas kvalitātes videozvanus un video konferences, tās gan pilnībā nevar aizstāt "Aci pret aci" komunikāciju, bet ikdienas steigā dalītas komandās var ļoti izpalīdzēt

3.7. tabula

Komunikācijas vērtēšanas atribūti

Nosaukums
Vērtēšanas kritēriji ir definēti
Komunikācijas vērtēšana tiek veikta periodiski
Vērtēšanas rezultāti ir pieejami visiem
Vērtēšanas rezultāti tiek saglabāti

Komunikācijas veidam ir būtiska ietekme uz spējību, bet spējību ietekmē arī komunikācijas biežums, kam jāpievērš īpaša uzmanība.

3.8. tabula

Komunikācijas biežuma atribūti

Nosaukums
Vairākas reizes dienā
Reizi dienā
Vairākas reizes nedēļā
Reizi nedēļā
Retāk nekā reizi nedēļā
Dažas reizes mēnesī
Reizi mēnesī
Retāk nekā reizi mēnesī

3.1.1.5. Procesu izmaiņu vadība

Procesu izmaiņu vadība ir nepieciešama, lai nodrošinātu kontrolētu procesu izmaiņu veikšanu. Procesu izmaiņas ir jāveic, jo esošie procesi var neatbilst jaunajām prasībām un procesi ir jāpārskata. Procesu izmaiņu vadība sastāv no vairākiem apakšdomēniem:

- pieejamība;
- atbildīgais;
- vērtēšana;
- izmaiņu ieviešana.

3.9. tabula

Procesu pieejamības atribūti

Nosaukums
Procesi uzņēmumā eksistē
Procesa dokumentācija ir pieejama visiem
Procesi ir skaidri un dokumentēti visiem saprotamā veidā

3.10. tabula

Atbildīgā atribūti

Nosaukums
Ir noteikts atbildīgais, kurš rūpējas par procesa dokumentāciju
Visiem ir zināms, kurš ir atbildīgais
Atbildīgajam ir visas tiesības veikt izmaiņas

3.11. tabula

Vērtēšanas atribūti

Nosaukums
Ir definēti procesu vērtēšanas kritēriji
Ir definēta procesu vērtēšanas procedūra
Informācija par procesiem un to darbības rezultātiem tiek saglabāta, lai būtu pieejama nākošā vērtēšanas iterācijā
Procesi tiek periodiski vērtēti
Procesu vērtēšanas rezultāti ir pieejami visiem

3.12. tabula

Izmaiņu ieviešanas atribūti

Nosaukums
Ir definētas izmaiņu ieviešanas procedūras

Informācija par veiktām izmaiņām tiek piereģistrēta
Ir definēts atbildīgais par konkrēto izmaiņu ieviešanu

3.1.1.6. Komandu organizēšanas procesa atribūti

Komandu veidošana un pārformēšana ir nozīmīgs programmatūras izstrādes procesa posms, neatkarīgi no tā vai tiek izmantotas spējās vai tradicionālās metodes. Pareizas komandas izvēle konkrētā projektā var nodrošināt veiksmīgāku projekta realizāciju.

3.13. tabula

Komandu organizēšanas procesa atribūti

Nosaukums
Komandu organizēšanai ir piesaistīts atbildīgais
Piesaistītajam atbildīgajam ir pieredze komandu veidošanā
Informācija par darbinieku zināšanām tiek uzglabāta
Informācija par darbinieku zināšanām tiek regulāri atjaunota
Tiek ņemta vērā darbinieka pieredze
Tiek ņemtas vērā darbinieka zināšanas
Tiek ņemta vērā iepriekšējā dalībnieku sadarbošanās
Iepriekšējās sadarbības rezultāti tiek uzglabāti
Tiek veidotas komandas ar augstu motivācijas līmeni
Tiek veidota pašorganizējoša komanda

Organizācijas mērķiem ir jāatbilst spējo metožu ideoloģijai [7]. Organizācijas kontekstā var izdalīt trīs no šiem mērķiem.

3.14. tabula

Mērķi

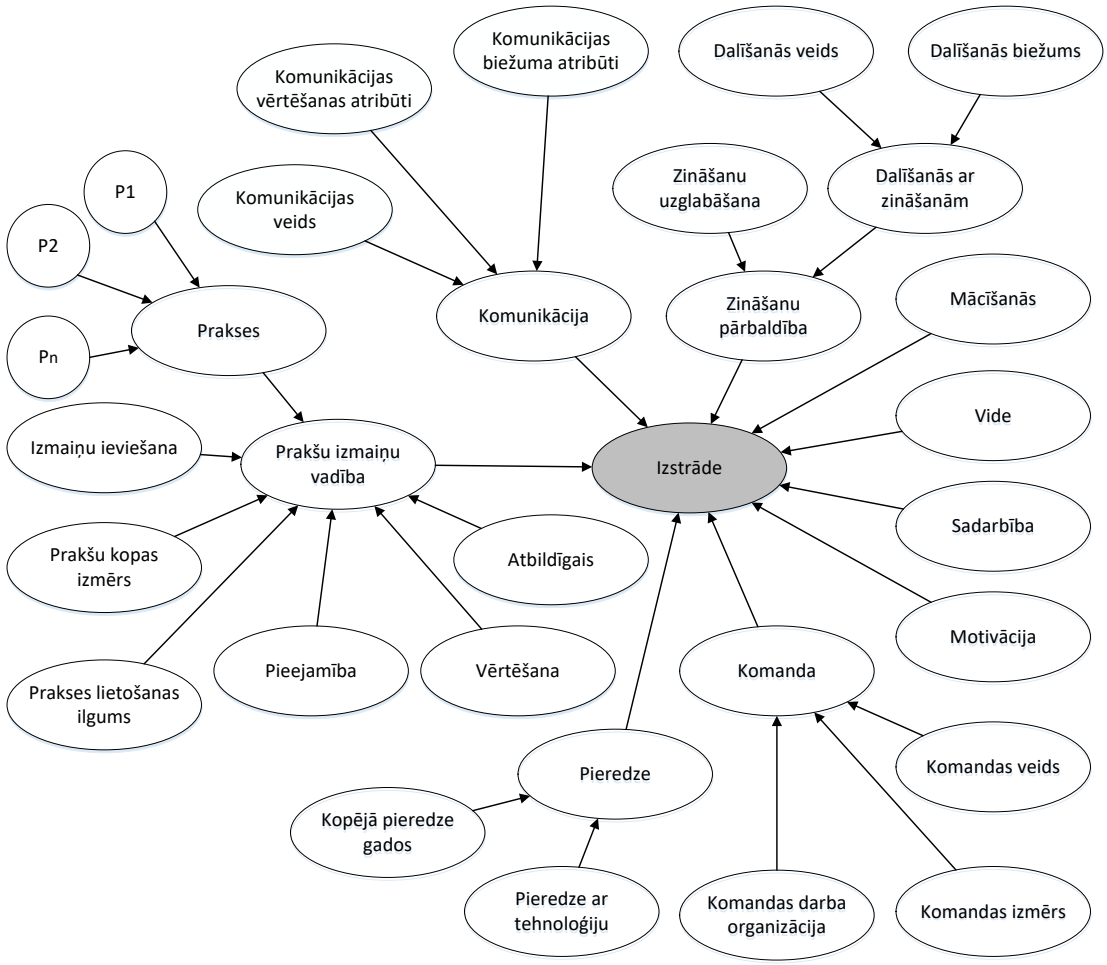
Nosaukums
Viena no augstākajām prioritātēm ir apmierināt klientu
Būtiska ir agra un nepārtraukta piegāde
Piegādāt vērtīgu programmatūru pēc iespējas biežāk

3.1.2. Izstrādes domēns

Izstrādes domēns raksturo produkta izstrādi, un uzsvars tiek likts uz izstrādes komandu. Izstrādes domēns sastāv no 9 apakšdomēniem (3.4. att.):

- vides komponente raksturo darba vidi;
- komunikācijas komponente izstrādes domēnā raksturo komunikācijas vērtēšanu komandas ietvaros. Komunikāciju līmenis ir jāvērtē katrā komandā atsevišķi, jo komandu sastāvs mēdz būt ļoti atšķirīgs;

- sadarbības komponente raksturo, cik labi sadarbojas komandas dalībnieki;
- līdzīgi kā organizācijas domēnā ir būtiski vērtēt organizācijas izmēru, arī izstrādes domēnā ir svarīgi vērtēt komandas izmēru;
- spējo komandu veidi var būt dažādi. Piemēram, komanda var strādāt vienā telpā, vienā birojā, dažādos birojos vai dažādās valstīs. Dažādām komandām ir nepieciešama pielāgota pieeja, it īpaši ja komandas neatrodas vienā birojā;
- pieredzes komponente raksturo, cik pieredzējusi ir komanda un kādā mērā tā ir gatava darboties ar spējām metodēm un praksēm;
- mācīšanās ir neatņemama izstrādes domēna sastāvdaļa un tā tiek izmantota, lai vērtētu komandas spēju mācīties un pilnveidoties;
- visas komandas ir dažādas un katrā komandā ir cilvēki, kuriem ir lielāka pieredze kādā no jautājumiem. Ir būtiski, lai vairāk pieredzējušie komandas dalībnieki būtu spējīgi nodod savas zināšanas mazāk pieredzējušiem;
- zināšanu uzglabāšanas komponentes uzdevums ir vērtēt procesus, kuri saistīti ar zināšanu uzglabāšanu un uzglabātās informācijas izmantošanu;
- prakšu izmaiņu vadība nodrošina esošo prakšu izvērtēšanu un jaunu prakšu ieviešanu. Līdzīgi kā procesu izmaiņu vadībā, arī prakšu izmaiņu vadībai ir nepieciešama atbildīgā persona, kura var iesākt jaunu prakšu ieviešanu un veco aizvietošanu.



3.4. att. Izstrādes domēns.

Ja veikt izstrādes un organizācijas domēna salīdzinājumu, tad varam apgalvot, ka izstrādes domēns ir jāizvērtē biežāk nekā organizācijas domēns, jo organizācijas domēnam ir globālāks raksturs, kā arī izstrādes domēna izvērtēšana var sniegt ātrāku atgriezenisko saiti, jo izmaiņas var ieviest katrā iterācijā.

3.1.2.1. Komunikācija

Komunikācija ir viens no svarīgākajiem spējo metožu stūrakmeņiem, un tās vērtējums var sniegt būtisku informāciju par spējības līmeni [7]. Komandas līmeņa komunikācijas novērtēšanai ir jāņem vērā komunikācijas veids (sk. 3.6. tabula). Kā arī papildu ir jāņem vērā komunikācijas vērtēšanas kritēriji (sk. 3.7. tabula) un biežuma atribūti (sk. 3.8. tabula). Organizācijas domēna kontekstā komunikāciju atribūti ir saistīti ar komunikāciju organizācijas iekšienē un ārēji ar klientiem augstākā līmenī. Komandas domēna kontekstā runa ir par komunikāciju izstrādes komandā un izstrādes komandas komunikāciju ar *ScrumMaster*, PO vai citu PO pārstāvi. Šajā sakarā

komunikāciju atribūtus (sk. 3.6. tabula, 3.7. tabula, 3.8. tabula), ir jānovērtē atbilstošā komunikācijas līmenī:

- komandas;
- *ScrumMaster*;
- PO vai PO pārstāvja.

Iegūstot informāciju no dažādiem komunikācijas līmeņiem, var rast labu priekšstatu par komunikācijas spējību komandā.

3.1.2.2. Zināšanu pārvaldīšana

Zināšanas, kuras reiz ir iegūtas, ir nepieciešams uzglabāt, lai tās varētu ātri atrast un atkārtoti izmantot nepieciešamības gadījumā. Komanda uzglabāšanas veidus parasti izvēlas pati, un vienlaikus var eksistēt vairāki uzglabāšanas veidi. Zināšanu pārvaldību ir iespējams iedalīt apakšdomēnos:

- kopējie zināšanu pārvaldības atribūti;
- dalīšanās ar zināšanām atribūti:
  - dalīšanās veids;
  - dalīšanās biežums.

3.15. tabula

Zināšanu uzglabāšanas atribūti

Nosaukums
Zināšanas tiek uzglabātas
Zināšanas tiek regulāri atjaunotas
Zināšanas var saglabāt jebkurš no komandas
Zināšanas ir viegli atrast
Zināšanas ir labi strukturētas
Zināšanu saglabāšana ir vienkārša
Visai komandai ir pieeja saglabātajām zināšanām
Atbildīgais par zināšanu uzglabāšanas sistēmu ir definēts
Komanda dalās ar zināšanām

3.16. tabula

Dalīšanās veida atribūti

Nosaukums
Blakus sēdošā programmētāja konsultēšana
Pārējo programmētāju konsultēšana
Kopēju lekciju organizēšana
Koda pārskata organizēšana
Kopēja koda rakstīšanas pasākumu organizēšana

Dalīšanās biežums attiecas uz visiem dalīšanas veida atribūtiem un katram atribūta veidiem ir jānovērtē atsevišķi.

3.17. tabula

Dalīšanās biežuma atribūti

Nosaukums
Reizi stundā
Vairākas reizes stundā
Vairākas reizes dienā
Reizi nedēļā
Vairākas reizes nedēļā
Reizi mēnesī
Vairākas reizes mēnesī
Retāk nekā reizi mēnesī

3.1.2.3. Mācīšanās

Mācīšanās ir neatņemama izstrādes domēna sastāvdaļa. Komandām ir jāiemācās kaut kas jauns katrā iterācijā. Cik daudz un vai komanda kaut ko iemācās ir atkarīgs no katras komandas un tās spējas mācīties, ko palīdz noteikt mācīšanās atribūti.

3.18. tabula

Mācīšanās atribūti

Nosaukums
Komanda runā par problēmām
Komanda runā par pozitīvām lietām
Problemātiskās lietas un to risinājumi tiek piereģistrēti
Pozitīvās lietas tiek piereģistrētas
Pozitīvās lietas tiek ņemtas vērā nākamajās iterācijās
Problēmas un to risinājumi tiek ņemti vērā nākamajās iterācijās
Piereģistrētā informācija ir ērti pieejama

3.1.2.4. Pieredze

Katra komandas dalībnieka pieredze ietekmē kopējo komandas pieredzi, kā rezultātā ir nepieciešams novērtēt katra dalībnieka pieredzi, izmantojot pieredzes atribūtus:

- kopējā pieredze gados programmatūras izstrādē;
- pieredze konkrētas tehnoloģijas izmantošanā.

3.19. tabula

Kopējā pieredze gados

Vērtība
X < 0,5
0,5> X >1
1 > X > 3
3 > X > 5
5 > X > 10
X > 10

3.20. tabula

Pieredze konkrētās tehnoloģijas izmantošanā

Vērtība
X < 0,5
0,5> X >1
1 > X > 3
3 > X > 5
5 > X > 10
X > 10

3.1.2.5. Komanda

Komanda ir tiešā darba darītāja, kas nodrošina, ka tiek izveidots un piegādāts programmatūras papildinājums. Komandas apakšdomēnu var iedalīt sīkāk:

- komandas veids;
- komandas lielums;
- komandas darba organizēšana.

Komandas veids tiešā veidā ietekmē komandas spējību un to kādas prakses ir jāizmanto konkrētās komandas gadījumā. Piemēram, dalītā komanda nedarbojas identiski komandai, kura atrodas vienā birojā.

3.21. tabula

Komandas veida atribūti

Nosaukums	Apraksts
Dalīta komanda	Komanda atrodas dažādās vietās, piemēram, 1 darbinieks Rīgā, 1 Liepājā, 2 Cēsīs
Vienā birojā esoša komanda	Komanda atrodas vienā birojā, bet dažādās telpās vai stāvos
Vienā telpā esoša komanda	Komanda atrodas vienā birojā un vienā telpā
Jaukta komanda	Daļa no komandas atrodas vienā vietā, bet otra daļa citā. Atšķiras no dalītās komandas ar to, ka liela daļa, nevis katrs indivīds atrodas dažādās vietās

3.1.2.6. Lielums

Komandas lielums ietekmē komandas spējību. Dažos avotos gan ir minēts atšķirīgs rekomendējamais komandas lielums. Viens no rekomendējamiem lielumiem ir nepārsniegt 7 cilvēkus izstrādes komandā [96].

3.22. tabula

Komandas lielums

Vērtība
X < 3
3 < X < 5
5 < X < 7
7 < X < 9
X > 9

3.1.2.7. Vispārīgie komandas atribūti

Vispārīgie komandas atribūti raksturo kopējās iezīmes, kādām ir jāatbilst komandai.

3.23. tabula

Komandas organizēšanās atribūti

Nosaukums
Komanda ir pašorganizējoša
Komanda nestrādā virsstundas

3.1.2.8. Sadarbība

Dažreiz cilvēkiem ir kļūdains priekšstats par sadarbību un tā tiek jaukta ar komunikāciju. Sadarbība nav komunikācija. Kvalitatīva komunikācija var novest pie kvalitatīvas sadarbības, bet tas nenozīmē, ka tā notiek vienmēr. Komunikācija ir dot, vai mainīties ar informāciju mutiski, rakstiski, vai izmantojot citus saziņas līdzekļus [97]. Sadarbība ir strādāt kopā ar kādu, lai kaut ko izveidotu [98]. Sadarbības spējība tiek noteikta procentos, kā soli izmantojot 20%.

3.24. tabula

Sadarbības atribūti

Vērtība	Apraksts
X < 20	Darbības rezultātā neizdodas realizēt iecerēto
20 < X < 40	Dažreiz darbības rezultātā izdodas sasniegt iecerēto, bet ir jūtamas izteiktas sadarbības problēmas
40 < X < 60	Darbības rezultātā ir jūtamas sadarbības problēmas, bet izdodas sasniegt iecerēto
60 < X < 80	Pārsvarā vēlamo rezultātu izdodas sasniegt
80 < X < 100	Vēlamo rezultātu izdodas sasniegt regulāri

3.1.2.9. Vide

Darba vides var atšķirties, un darba videi ir būtiska ietekme uz cilvēkiem, kas tajā strādā. Darba vide ietekmē darbinieku pašsajūtu un produktivitāti. Piemēram, darba vidē, kurā valda nepārtraukts stress, neuzticība vienam pret otru vai ļoti augstas ambīcijas, kā arī citi nelabvēlīgi faktori, produktivitāte būs zemāka, nekā vidē ar labvēlīgiem faktoriem, tādiem kā cieša draudzība, uzticība utt.. Vidi ir iespējams vērtēt pēc vairākiem atribūtiem, tādiem kā, apgaismojums, gaisa kvalitāte un līdzīgiem. Šā darba kontekstā tik detalizēts iedalījums nav nepieciešams un darbs aprobežojas ar kopējās apmierinātības vērtēšanu.

3.25. tabula

Vides vērtējums

Vērtība	Apraksts
X < 20	Vide ir nelabvēlīga un tādā neizdodas veiksmīgi veikt darba pienākumus
20 < X < 40	Vidē ir iespējams veikt darba pienākumus, bet ir būtiskas problēmas
40 < X < 60	Vide ir apmierinoša, un tajā ir iespējams strādāt, bet ir zināmas problēmas
60 < X < 80	Vide ir labvēlīga un pārsvarā nav problēmu, kas traucē veiksmīgi pildīt savus pienākumus
80 < X < 100	Vidē nav traucējošu faktoru

3.1.2.10. Prakšu izmaiņu vadība

Prakšu izmaiņu vadība ir saistīta ar procesiem, kas nepieciešami, lai ieviestu vai nomainītu kādu no izmantotajām praksēm. To var nosacīti iedalīt vairākos apakšdomēnos:

- pieejamība;
- atbildīgais;
- vērtēšana;
- izmaiņu ieviešana;
- prakses.

3.26. tabula

Procesu pieejamības atribūti

Nosaukums
Procesi uzņēmumā eksistē
Procesa dokumentācija ir pieejama visiem
Procesi ir skaidri un dokumentēti visiem saprotamā veidā

3.27. tabula

Atbildīgā atribūti

Nosaukums
Ir noteikts atbildīgais, kurš rūpējas par procesa dokumentāciju
Visiem ir zināms, kurš ir atbildīgais
Atbildīgajam ir visas tiesības veikt izmaiņas

3.28. tabula

Vērtēšanas atribūti

Nosaukums
Ir definēti prakšu vērtēšanas kritēriji
Ir definēta prakšu vērtēšanas procedūra
Informācija par praksēm un to darbības rezultātiem tiek saglabāta
Prakses tiek vērtētas periodiski
Prakšu vērtēšanas rezultāti ir pieejami visiem

3.29. tabula

Izmaiņu ieviešanas atribūti

Nosaukums
Ir definētas izmaiņu ieviešanas procedūras
Informācija par veiktām izmaiņām tiek piereģistrēta
Ir definēts atbildīgais par konkrēto izmaiņu ieviešanu

Parasti komandas izmanto vairākas spējās prakses, kuras palīdz nodrošināt spējos projektus ar kvalitāti. Problēmas var rasties gadījumos, ja tiek izmantots pārāk liels vai pārāk neliels prakšu skaits, kas traucē sasniegt vēlamo rezultātu. Atkarībā no komandu veida, izmēra vai organizācijas ir jāizvēlas dažādas prakses un tas, kādas prakses tiks izvēlētas, ir pilnībā atkarīgs no konkrētās komandas.

3.30. tabula

Prakšu kopas izmērs

Vērtība
$X < 3$
$3 < X < 6$
$6 < X < 9$
$9 < X < 12$
$X > 12$

Katra prakse ir jāizvērtē atbilstoši noteiktajiem un izvēlētajiem vērtēšanas kritērijiem. Ja tiek fiksēts, ka prakse nenes vēlamo rezultātu vai traucē komandas darbībai, to ir nepieciešams aizvietot.

3.31. tabula

Prakses lietošanas ilgums

Vērtība (mēnešos)
X < 1
1 < X < 6
6 < X < 12
12 < X < 24
X > 24

3.1.2.11. Motivācija

Viens no 12 spējības pamatprincipiem ir komandas augstā motivācija [7]. Komandas motivācijas līmeni ir nepieciešams periodiski novērtēt, lai laicīgi ievērotu, ka nepieciešamas kādas izmaiņas.

3.32. tabula

Motivācija

Nosaukums
Komandas dalībnieki ir labi motivēti
Komandas motivācija tiek periodiski vērtēta

3.1.3. Kvalitātes domēns

Kvalitātes domēna mērķis ir nodrošināt kvalitatīva produkta piegādi pasūtītājam. Kvalitātes domēns sastāv no 6 apakšdomēniem (3.5. att.):

3.5. att. Kvalitātes domēns.

- rīki - komandas izmanto dažādus rīkus spējības līmeņa uzlabošanai;
- standarti – apkopo un izvērtē informāciju par ieviestiem standartiem, piemēram, dokumentācijas standartiem, programmēšanas standartiem, kvalitātes standartiem utt.;
- vienošanās – apkopo un izvērtē informāciju par mutiskām vai rakstiskām vienošanām, kuras ir veikuši komandas dalībnieki;
- vadlīnijas – apakšdomēna ietvaros tiek izvērtētas uzņēmuma vai komandas vadlīnijas, kuras saistītas ar dažādiem procesiem, piemēram, ko darīt, ja kodā tiek atrasta kļūda;
- prakšu izmaiņu vadība - kvalitātes domēna kontekstā apkopo un izvērtē kvalitātes nodrošināšanas prakses un palīdz izvēlēties labāko prakšu kopu, konkrētā uzņēmuma un komandas kontekstā;

- kompetence - izvērtē kvalitātes nodrošināšanā iesaistītās kompetences. Lielākās organizācijās un komandās ir vairāk konkrēta apgabala speciālistu, kuru piesaiste atbilstošā projekta stadijā var uzlabot piegādājamā produkta kvalitāti. Tipisks piemērs ir projekta arhitektūra. Gadījumā, ja sarežģīta projekta sākumā netiek pieaicināts arhitekts, tad risinājums var izrādīties nepareizi strukturēts. Līdzīga situācija var rasties arī gadījumos, ja nav kvalitatīva testēšanas speciālista vai datubāzes arhitekta;
- fokusēšanās uz izcilību – izvērtē informāciju par komandas fokusu.

3.1.3.1. Vadlīnijas un zināšanas

Vadlīnijas ir labs veids, kā panākt, ka komandas dalībniekiem ir skaidrs, ko darīt konkrētās situācijās. Vadlīnijas palīdz veidot vienotu arhitektūru vai koda stilu. Vadlīnijas ir iespējams iedalīt apakšdomēnos:

- pieejamība;
- atbildīgais;
- vērtēšana;
- izmaiņu ieviešana.

3.33. tabula

Vadlīniju pieejamības atribūti

Nosaukums
Vadlīnijas eksistē
Vadlīniju dokumentācija ir pieejama visiem
Vadlīnijas ir skaidras un dokumentētas visiem saprotamā veidā

3.34. tabula

Atbildīgā atribūti

Nosaukums
Ir noteikts atbildīgais, kurš rūpējas par vadlīniju dokumentāciju
Visiem ir zināms, kurš ir atbildīgais
Atbildīgajam ir visas tiesības veikt izmaiņas

3.35. tabula

Vērtēšanas atribūti

Nosaukums
Ir definēti vadlīniju vērtēšanas kritēriji
Ir definēta vadlīniju vērtēšanas procedūra
Informācija par vadlīnijām un to darbības rezultātiem tiek saglabāta, lai būtu pieejama nākošā vērtēšanas iterācijā

Nosaukums
Vadlīnijas tiek periodiski vērtētas
Vadlīniju vērtēšanas rezultāti ir pieejami visiem

3.36. tabula

Izmaiņu ieviešanas atribūti

Nosaukums
Ir definētas izmaiņu ieviešanas procedūras
Informācija par veiktām izmaiņām tiek piereģistrēta
Ir definēts atbildīgais par konkrēto izmaiņu ieviešanu

3.1.3.2. Vienošanās

Vienošanās ir sava veida mutiski vai rakstiski līgumi, par kuriem vienojas komandas dalībnieki. Vienošanās palīdz ieturēt konkrētu darba stilu. Piemēram, komanda vienojas, ka sapulču laikā nedrīkst izmantot mobilos tālruņus. Komanda pati izvēlas kurā brīdī ieviest jaunas vienošanās un kurā tās ir jānomaina. Vienošanās domēnu var iedalīt sīkāk:

- pieejamība;
- vērtēšana.

3.37. tabula

Vienošanās pieejamības atribūti

Nosaukums
Vienošanās eksistē
Vienošanās ir pieejama un zināma visiem
Vienošanās ir skaidri definētas un visiem saprotamas
Ir definēta vienošanās izveidošanas un maiņas procedūra
Ir pieejama informācija par vēsturiskajām vienošanām

3.38. tabula

Vērtēšanas atribūti

Nosaukums
Ir definēti vienošanās vērtēšanas kritēriji
Ir definēta vienošanās vērtēšanas procedūra
Informācija par komandas vienošanām un to darbības rezultātiem tiek saglabāta
Vienošanās tiek periodiski vērtētas
Vienošanās vērtēšanas rezultāti ir pieejami visiem

3.1.3.3. Standarti

Kvalitātes nodrošināšanai var izmantot standartus, kuri nosaka, kādam ir jābūt produktam un procesiem. Izmantojamie standarti var būt visiem zināmi starptautiski

vai pašas organizācijas, vai komandas izveidoti. Standartu apakšdomēnu var iedalīt sīkāk:

- pieejamība;
- atbildīgais;
- vērtēšana;
- izmaiņu ieviešana.

3.39. tabula

Standartu pieejamības atribūti

Nosaukums
Standarti eksistē
Standartu dokumentācija ir pieejama visiem
Standarti ir skaidri un dokumentēti visiem saprotamā veidā

3.40. tabula

Atbildīgā atribūti

Nosaukums
Ir noteikts atbildīgais, kurš rūpējas par standartu dokumentāciju
Visiem ir zināms, kurš ir atbildīgais
Atbildīgajam ir visas tiesības veikt izmaiņas

3.41. tabula

Vērtēšanas atribūti

Nosaukums
Ir definēti standartu vērtēšanas kritēriji
Ir definēta standartu vērtēšanas procedūra
Informācija par standartiem un to darbības rezultātiem tiek saglabāta, lai būtu pieejama nākošā vērtēšanas iterācijā
Standarti tiek periodiski vērtēti
Standartu vērtēšanas rezultāti ir pieejami visiem

3.42. tabula

Izmaiņu ieviešanas atribūti

Nosaukums
Ir definētas izmaiņu ieviešanas procedūras
Informācija par veiktām izmaiņām tiek piereģistrēta
Ir definēts atbildīgais par konkrēto izmaiņu ieviešanu

3.1.3.4. Rīki

Kvalitātes un produktivitātes nodrošināšanai komandas izmanto dažādus rīkus. Rīku izvēle ir pilnībā atkarīga no komandas un uzņēmuma finansiālajām iespējām.

Finansiālie aspekti ir jāņem vērā, jo daļa no rīkiem nav pieejama bezmaksas. Rīku apakšdomēns sastāv no vairākiem apakšdomēniem:

- vērtēšana;
- finansējums;
- apmācība;
- pielietojums.

3.43. tabula

Rīku vērtēšanas atribūti

Nosaukums
Ir definēti instrumentu vērtēšanas kritēriji
Ir definēta instrumentu vērtēšanas procedūra
Ir noteikts atbildīgais vērtēšanas veikšanai
Vērtēšanas rezultāti ir pieejami visiem

3.44. tabula

Finansējums vienam izstrādātājam (EUR/gadā)

Vērtība
X < 100
100 < X < 300
300 < X < 600
600 < X 1000
X > 1000

3.45. tabula

Finansējuma apmērs gadā organizācijas kontekstā (EUR/gadā)

Vērtība
X < 1000
1000 < X < 3000
3000 < X < 6000
6000 < X 10000
10000 < X < 20000
X > 20000

Katru rīku, kuru izvēlas organizācija ir nepieciešams apgūt, un tam ir nepieciešams atvēlēt laiku. Parasti, sākumā ar rīku iepazīstas viens vai vairāki cilvēki no organizācijas. Nākamajā solī rīks tiek izrādīts pārējiem. Atkarībā no organizācijas apmācībām atvēlētais laiks atšķiras (sk. 3.46. tabula).

3.46. tabula

Rīka apmācībai atvēlētais laiks (dienas/gadā)

Vērtība
X < 1
1 < X < 3
3 < X < 7
7 < X 14
X > 14

Katrs izmantojamais rīks ir jāizvērtē atbilstoši atribūtiem, jo pastāv iespēja, ka izmantošanas gaitā rīks ir izrādījies nelietderīgs vai tā darbināšana rada vairāk problēmu nekā risinājumu.

3.47. tabula

Rīka ieviešanas atribūti

Nosaukums
Procedūra, kā ieviest rīku, ir definēta
Rīka ieviešanas atbildīgais ir definēts
Informācija par veiktajām izmaiņām tiek saglabāta

3.1.3.5. Prakšu izmaiņu vadība

Prakšu izmaiņu vadība kvalitātes domēna kontekstā ir līdzvērtīga prakšu pārvaldīšanai izstrādes kontekstā. Kvalitātes domēna kontekstā, par praksēm uzskata prakses, kuras izmanto kvalitatīva produkta izveidošanā. To var nosacīti iedalīt vairākos apakšdomēnos:

- pieejamība;
- atbildīgais;
- vērtēšana;
- izmaiņu ieviešana;
- prakšu kopas izmērs;
- prakses lietošanas ilgums mēnešos.

3.48. tabula

Procesu pieejamības atribūti

Nosaukums
Procesi uzņēmumā eksistē
Procesa dokumentācija ir pieejama visiem
Procesi ir skaidri un dokumentēti visiem saprotamā veidā

3.49. tabula

Atbildīgā atribūti

Nosaukums
Ir noteikts atbildīgais, kurš rūpējas par procesa dokumentāciju
Visiem ir zināms, kurš ir atbildīgais
Atbildīgajam ir visas tiesības veikt izmaiņas

3.50. tabula

Vērtēšanas atribūti

Nosaukums
Ir definēti prakšu vērtēšanas kritēriji
Ir definēta prakšu vērtēšanas procedūra
Informācija par praksēm un to darbības rezultātiem tiek saglabāta, lai būtu pieejama nākošā vērtēšanas iterācijā
Prakses tiek periodiski vērtētas
Prakšu vērtēšanas rezultāti ir pieejami visiem

3.51. tabula

Izmaiņu ieviešanas atribūti

Nosaukums
Ir definētas izmaiņu ieviešanas procedūras
Informācija par veiktām izmaiņām tiek piereģistrēta
Ir definēts atbildīgais par konkrēto izmaiņu ieviešanu

Parasti komandas izmanto vairākas spējās prakses, kuras palīdz nodrošināt spējos projektus ar kvalitāti. Problēmas rodas gadījumos, ja tiek izmantots pārāk liels vai pārāk neliels prakšu skaits, kas traucē sasniegt vēlamo rezultātu. Atkarībā no komandu veida, izmēra vai organizācijas ir jāizvēlas dažādas prakses, un un prakšu izvēle ir atkarīga no konkrētās komandas.

3.52. tabula

Prakšu kopas izmērs

Vērtība
X < 3
3 < X < 6
6 < X < 9
9 < X < 12
X > 12

3.53. tabula

Prakses lietošanas ilgums

Vērtība (mēnešos)
X < 1

Vērtība (mēnešos)
1 < X < 6
6 < X < 12
12 < X < 24
X > 24

3.1.3.6. Kompetences

Kvalitātes nodrošināšanā piedalās speciālisti ar dažādām kompetencēm, katrs pilda savas īpašas funkcijas. Šo kompetenču esamība vai neesamība var palielināt vai samazināt kvalitātes domēna spējību. Kompetenču apakšdomēns sastāv no vairākām kompetencēm:

- datubāzes arhitekts;
- ietvaru izstrādātājs;
- UI izstrādātājs;
- infrastruktūras izstrādātājs;
- arhitekts.

3.54. tabula

Kompetenču sadalījuma atribūti

Nosaukums
Vienam dalībniekam ir vairākas kompetences
Viena kompetence tiek dalīta starp vairākiem dalībniekiem
Katram dalībniekam ir viena kompetence

3.55. tabula

Datubāzes arhitekta atribūti

Nosaukums
Kompetence eksistē
Datubāzes arhitektūra
Datubāzes ātrdarbība
Datubāzes dokumentācijas pieejamība
Datubāzes arhitekta pieejamība

3.56. tabula

Ietvaru izstrādātāja atribūti

Nosaukums
Kompetence eksistē
Ietvara izmantojamība
Ietvara kvalitāte
Ietvara stabilitāte
Ietvara dokumentācijas pieejamība
Ietvara izstrādātāja pieejamība

3.57. tabula

Arhitekta atribūti

Nosaukums
Kompetence eksistē
Sistēmas arhitektūra
Arhitektūras dokumentācijas pieejamība
Arhitektūras izmantojamība
Arhitekta pieejamība

3.58. tabula

UI Izstrādātāja atribūti

Nosaukums
Kompetence eksistē
UI lietojamība
UI izstrādātāja pieejamība

3.59. tabula

Infrastruktūras izstrādātāju atribūti

Nosaukums
Kompetence eksistē
Infrastruktūras stabilitāte
Infrastruktūras dokumentācijas pieejamība
Izstrādātāja pieejamība
Infrastruktūras monitoringa pieejamība

3.1.3.7. Fokusēšanās uz izcilību

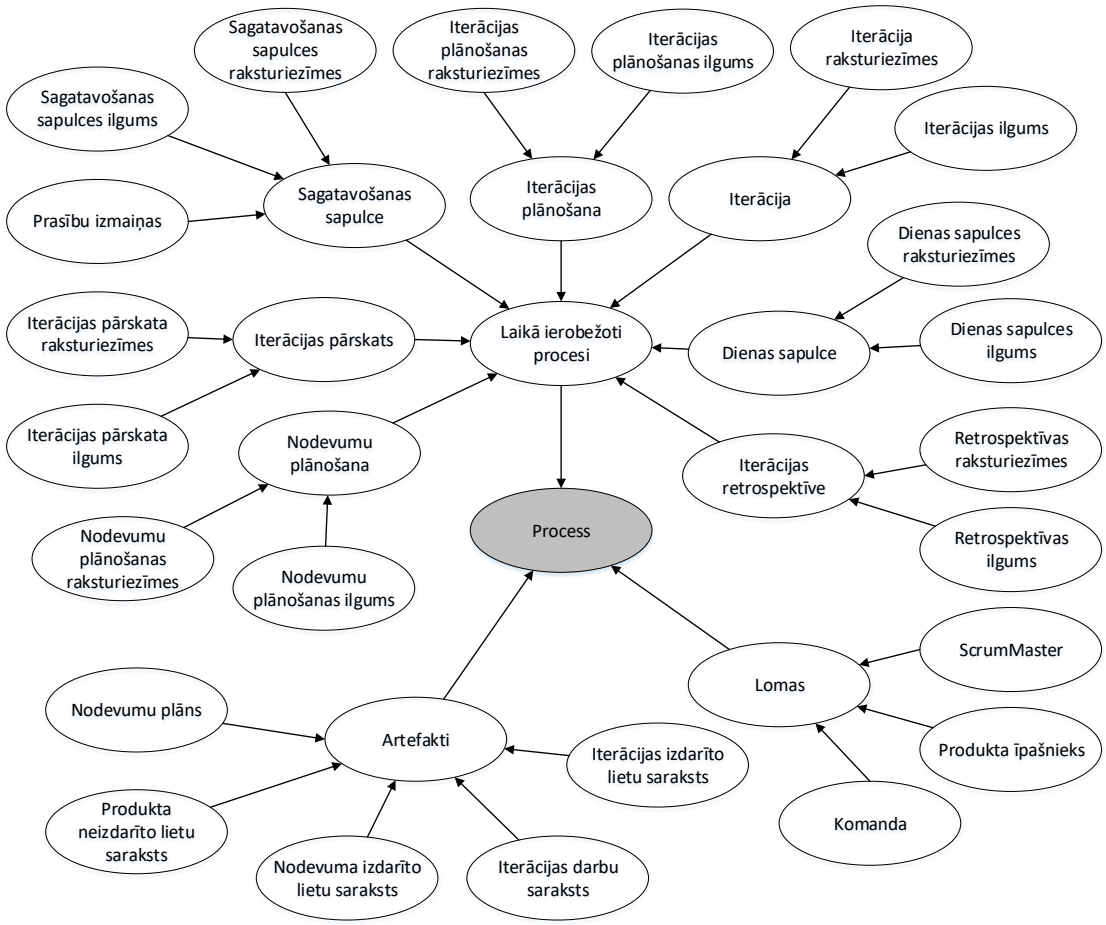
3.60. tabula

Fokusēšanās uz izcilību

Nosaukums
Tiek pievērsta nepārtraukta uzmanība tehniskai izcilībai
Tiek pievērsta nepārtraukta uzmanība labam dizainam (tai skaitā arhitektūrai un struktūrai)
Vienkāršība ir svarīga

3.1.4. Procesu domēns

Procesu domēns izvērtē procesus, kas saistīti ar *Scrum* vai līdzīgu metodi. Darba ietvaros gan tiek apskatīti *Scrum* procesi un atribūti. Procesu domēns sastāv no trīs apakšdomēniem (3.6. att.):



3.6. att. Procesu domēns.

- laikā ierobežoti procesi (angl. *TimeBoxed activities*) komponente atbild par *Scrum* procesa izvērtēšanu [85][86]:
    - sagatavošanas sapulce (angl. *Grooming*) ir sapulce, kuras laikā pirmo reizi tiek pārrunāti jaunie lietotāju stāsti un veikta to izvērtēšana;
    - laidienu plānošana (angl. *Release planning*) ir process, kas saistīts ar jauna laidienu plāna izveidošanu, kas nepieciešams ilgtermiņa izstrādei;
    - iterācijas plānošana (angl. *Sprint planning*) ir process, kura laikā tiek izlemts, pie kādiem lietotāja stāstiem tiks strādāts konkrētās iterācijas laikā;
    - iterācija (angl. *Sprint*) ir laiks, kad tiek implementēti konkrētie lietotāja stāsti. Iterācijas garumi var būt dažādi, un, lai noteiktu pareizu iterācijas garumu konkrētā projekta vai komandas vajadzībām, tad ir nepieciešams veikt iterāciju vērtēšanu;

    - dienas sapulce (angl. *Daily Scrum*) ir īss laiks, kuru komanda pavada kopā noteiktā laikā, lai informētu viens otru par padarītajiem un paredzamajiem darbiem;
    - iterācijas pārskats (angl. *Sprint Review*) ir paredzēts, lai komanda var nodemonstrēt savu sniegumu ieinteresētajām pusēm;
    - iterācijas retrospektīva (angl. *Sprint Retrospective*) ir laiks, kuru komanda velta, lai izrunātos par labajām un sliktajām lietām, kuras notikušas iterācijas laikā.
- artefakti ir objekti, kurus izmanto *Scrum* procesa organizēšanai:
    - produkta neizdarīto lietu saraksts PB ir saraksts, kurā atrodas tie lietotāju stāsti, kurus vēlamies realizēt konkrētā produkta ietvaros un kuri vēl nav paņemti kādā no iterācijām;
    - laidiena izdarīto darbu saraksts (angl. *Release Burndown*) sastāv no implementētajiem lietotāju stāstiem;
    - iterācijas darbu saraksts (angl. *Sprint Backlog*) ir saraksts, kurā atrodas tie lietotāju stāsti, kuri tiek implementēti konkrētajā iterācijā;
    - iterācijas izdarīto darbu saraksts (angl. *Sprint Burndown*) ir saraksts, kurā atrodas izdarītie lietotāju stāsti. Saraksts palīdz Komandai, PO vai *ScrumMaster* sekot līdzi izstrādes gaitai.
- spējās metodēs, līdzīgi kā tas ir arī tradicionālajās metodēs, ir noteikts lomu sadalījums. *Scrum* eksistē trīs pamata lomas:
    - produkta īpašnieks PO ir persona, kurš darbojas uzņēmuma vārdā un ir atbildīgs par izstrādājamo produktu. Produkta īpašniekam ir tiesības pieņemt lēmumus, kuri attiecas uz produktu. Lomas izvērtēšana sniedz iespēju noteikt, vai Produkta īpašniekam ir pietiekoši daudz pilnvaru, vai viņš ir pietiekoši ieinteresēts produktā utt.;
    - *ScrumMaster* ir persona, kura palīdz komandai veikt savu darbu, organizējot komandai nepieciešamo vidi, tai skaitā pasargājot komandu no ārējas iejaukšanās;
    - komanda ir reālā darba veicēja un sastāv no dažādiem speciālistiem (izstrādātāji, testētāji, arhitekti utt.).

Procesa domēna izvērtēšana sniedz iespēju noteikt *Scrum* pielietošanas vājākās vietas un nepieciešamības gadījumā tās uzlabot. Procesu domēna atribūti sniedz iespēju

izvērtēt procesu domēna spējību. Procesu domēns iekļauj sevī procesus, kas saistīti ar *Scrum*, bet nepieciešamības gadījumā domēns var tikt pārveidots, lai izmantotu un izvērtētu kādas citas metodes procesus. Šajā gadījumā gan jāņem vērā, ka metodei būs jāizstrādā savi atribūti un jāveic to vērtējumi, līdzīgi kā tas izdarīts ar *Scrum*.

3.1.4.1. Laikā ierobežotas aktivitātes

Laikā ierobežotu aktivitāšu saraksts sastāv no 3.1.4 apakšnodaļā minētajiem procesiem, tādiem kā "Iterācijas plānošana", "Dienas sapulce", "Iterācijas pārskats" u. c. "Sagatavošanas sapulce", "Iterācijas plānošanas sapulce", "Dienas sapulce" un pārējās sapulces tiek vērtētas pēc vairākām atribūtu grupām:

- vispārīgie atribūti;
- ilguma atribūti.

3.61. tabula

Sagatavošanas sapulces vispārīgie atribūti

Nosaukums
Sapulce eksistē
Sapulce tiek veltīta lietotāju stāstu apspriešanai
Sapulce tiek veltīta sākotnējo vērtējumu noteikšanai
Tiek definēti sākotnējie pieņemšanas kritēriji
Sapulces tiek organizētas regulāri un norisinās paralēli iterācijām
Noteiktie sapulces ierobežojumi tiek ievēroti (piemēram, laiks)
Komandai ir skaidrs, kādam nolūkam sapulce eksistē
Komandai ir skaidrs, kas tai jādara sapulcē
PO ir skaidrs, kādam nolūkam sapulce eksistē
PO ir skaidri viņa pienākumi sapulces laikā
ScrumMaster ir skaidrs, kādam nolūkam sapulce eksistē
ScrumMaster ir skaidri viņa pienākumi sapulces laikā

3.62. tabula

Sagatavošanas sapulces ilgums

Vērtība (Stundas)
X < 1
1 < X < 2
X > 2

3.63. tabula

Prasību atribūti

Nosaukums
Izmaiņas tiek pieņemtas un realizētas arī vēlu izstrādē
Prasības tiek uzglabātas

Nosaukums
Prasību dokumenti ir sabalansēti (Dokumenti nav lieli un iekļauj minimālo nepieciešamo informāciju)

3.64. tabula

Iterācijas plānošanas sapulces vispārīgie atribūti

Nosaukums
Sapulce eksistē
Sapulce tiek veltīta gala vērtējumu noteikšanai
Tiek modificēti pieņemšanas kritēriji
Nepieciešamības gadījumā ir iespējams apspriest neskaidrības, ja tādas eksistē
Tiek izveidoti darba uzdevumi katram US
Komanda izvēlas darba uzdevumus
Noteiktie sapulces ierobežojumi tiek ievēroti (piemēram, laiks)
Komandai ir skaidrs, kādam nolūkam sapulce eksistē
Komandai ir skaidrs, kas tai jādara sapulcē
PO ir skaidrs, kādam nolūkam sapulce eksistē
PO ir skaidri viņa pienākumi sapulces laikā
ScrumMaster ir skaidrs, kādam nolūkam sapulce eksistē
ScrumMaster ir skaidri viņa piemākumi sapulces laikā

3.65. tabula

Iterācijas plānošanas sapulces ilgums

Vērtība (Stundas)
X < 1
1 < X < 2
2 < X < 4
4 < X < 6
X > 6

3.66. tabula

Iterācijas vispārīgie atribūti

Nosaukums
Process eksistē
Komanda ir pasargāta no ārējas iedarbības iterācijas laikā
Definēts noteikts iterācijas garums
Pastāv vienošanās kā izņemt US no iterācijas, ja rodas problēmas
Pastāv vienošanās kā paņemt jaunus US iterācijā, ja rodas iespēja
Sākotnējie darba uzdevumi katram US ir definēti
Komanda ir vienojusies, kādā veidā tiek veidoti un modificēti darba uzdevumi
Iterācijas laikā ir pieejams kāds no klienta pārstāvjiem
Komandai ir skaidra “Izdarīts” definīcija – definīcija nosaka kādiem atribūtiem ir jāatbilst US, lai to varētu uzskatīt par izdarītu
Komandai ir skaidrs, kādam nolūkam iterācija eksistē
Komandai ir skaidrs, kas tai jādara iterācijas laikā
PO ir skaidrs, kādam nolūkam iterācija eksistē
PO ir skaidri viņa pienākumi iterācijas laikā
ScrumMaster ir skaidrs, kādam nolūkam iterācija eksistē
ScrumMaster ir skaidri viņa piemākumi iterācijas laikā
Izstrāde ir ilgtspējīga
Izstrādes temps tiek uzturēts konstants

3.67. tabula

Iterācijas ilgums (nedēļas)

Vērtība
X = 1
X = 2
X = 3
X = 4
X > 4

3.68. tabula

Dienas sapulces vispārīgie atribūti

Nosaukums
Sapulce eksistē
Sapulcei ir noteikts konkrēts laiks katru dienu
Sapulcei ir noteikts garums
Sapulces laikā visi dalībnieki atbild uz 3 jautājumiem (Ko darīji vakar? Ko darīsi šodien? Vai ir kādi traucējumi?)
Sapulce netiek izmantota detalizētai problēmu apspriešanai
Dienas sapulcē piedalās visa komanda
Komandai ir skaidrs, kādam nolūkam sapulce eksistē
Komandai ir skaidrs, kas tai jādara sapulcē
ScrumMaster ir skaidrs, kādam nolūkam sapulce eksistē
ScrumMaster ir skaidri viņa piemākumi sapulces laikā

3.69. tabula

Dienas sapulces ilgums

Vērtība (minūtes)
X < 10
10 < X < 15
X > 15

3.70. tabula

Iterācijas pārskata vispārīgie atribūti

Nosaukums
Sapulce eksistē
Sapulce ir regulāra
Sapulcei ir noteikts garums
Sapulcē ir demonstrējams programmatūras papildinājums
Papildinājumi tiek demonstrēti demo veidā
Iterācijas rezultāts atbilst sprinta plānošanā izvirzītajiem mērķiem
Komanda ir pabeigusi visus iterācijā paņemtos US
Komandai ir skaidrs, kādam nolūkam sapulce eksistē
Komandai ir skaidrs, kas tai jādara sapulcē
PO ir skaidrs, kādam nolūkam sapulce eksistē
PO ir skaidri viņa pienākumi sapulces laikā
ScrumMaster ir skaidrs, kādam nolūkam sapulce eksistē
ScrumMaster ir skaidri viņa piemākumi sapulces laikā

3.71. tabula

Iterācijas pārskata ilgums

Vērtība (stundas)
X < 1
1 < X < 2
2 < X < 5
5 < X < 8
X > 8

3.72. tabula

Iterācijas retrospektīvas sapulces vispārīgie atribūti

Nosaukums
Sapulce eksistē
Sapulce ir regulāra
Sapulcei ir noteikts garums
Sapulcē piedalās visa komanda
Sapulcē iesaistās visa komanda
Sapulcei ir vadītājs
Komanda izmanto retrospektīvu, lai mācītos
Komandai ir skaidrs, kādam nolūkam sapulce eksistē
Komandai ir skaidrs, kas tai jādara sapulcē
ScrumMaster ir skaidrs, kādam nolūkam sapulce eksistē
ScrumMaster ir skaidri viņa piemākumi sapulces laikā

3.73. tabula

Iterācijas retrospektīvas sapulces ilgums

Vērtība (stundas)
X < 1
1 < X < 2
2 < X < 5
5 < X < 8
X > 8

3.74. tabula

Laidienu plānošanas vispārīgie atribūti

Nosaukums
Laidienu plānošana eksistē
Komandai ir zināms laidienu plāns
Komanda pieturas pie laidienu plāna
Komanda spēj realizēt laidienu plānu
Pastāv iespēja modificēt laidienu plānu
Komanda saprot kādam nolūkam eksistē laidienu plāns
Komandai ir skaidrs, kādam nolūkam sapulce eksistē
Komandai ir skaidrs, kas tai jādara sapulcē
PO ir skaidrs, kādam nolūkam sapulce eksistē
PO ir skaidri viņa pienākumi sapulces laikā
ScrumMaster ir skaidrs, kādam nolūkam sapulce eksistē
ScrumMaster ir skaidri viņa piemākumi sapulces laikā

3.75. tabula

Laidienu plānošanas ilgums

Vērtība (stundas)
X < 1
1 < X < 2
X > 2

3.1.4.2. Lomas

Scrum procesu izpildei ir nepieciešama komanda, katrs no komandas pārstāv kādu lomu. Procesu domēna spējības noteikšanai ir nepieciešams izvērtēt lomu spējību.

3.76. tabula

ScrumMaster lomas atribūti

Nosaukums
Loma eksistē
ScrumMaster iesaistās "Sākotnējās plānošanas" sapulcē
ScrumMaster iesaistās "Sprinta plānošanas" sapulcē
ScrumMaster iesaistās "Dienas" sapulcē
ScrumMaster iesaistās "Iterācijas pārskata" sapulcē
ScrumMaster iesaistās "Iterācijas retrospektīvas" sapulcē
ScrumMaster iesaistās "Laidienu plānošanas" sapulcē
ScrumMaster iterācijas laikā sargā komandu no ārējās ietekmes
Komanda ir informēta par *ScrumMaster* pienākumiem
ScrumMaster ir skaidri viņa pienākumi
ScrumMaster loma tiek periodiski vērtēta

3.77. tabula

PO lomas atribūti

Nosaukums
Loma eksistē
Komandai ir skaidri PO pienākumi
PO ir skaidri lomas pienākumi
PO piedalās "Sākotnējās plānošanas" sapulcē
PO piedalās "Iterācijas plānošanas" sapulcē
PO piedalās "Iterācijas pārskata" sapulcē
PO piedalās "Laidienu plānošanas" sapulcē
PO loma tiek periodiski vērtēta
PO ir spējīgs skaidri paskaidrot US būtību

3.78. tabula

Komandas lomas atribūti

Nosaukums
Komandai ir zināmi procesa domēna procesi
Visa komanda piedalās "Sākotnējās plānošanas" sapulcē
Visa komanda piedalās "Iterācijas plānošanas" sapulcē
Visa komanda piedalās "Iterācijā"
Visa komanda piedalās "Iterācijas pārskata" sapulcē

Nosaukums
Visa komanda piedalās “Iterācijas retrospektīvas” sapulcē
Visa komanda piedalās “Dienas” sapulcē
Visa komanda piedalās “Laidienu plānošanas” sapulcē
Komandas darbs tiek periodiski vērtēts

3.1.4.3. Artefakti

Artefakti ir objekti, kuri tiek izmantoti dažādu procesu atbalstam. Atkarībā no izvēlētajām praksēm artefaktu saraksts var mainīties, bet neatkarīgi no izvēlētajām praksēm, komandas rīcībā ir jābūt pieciem artefaktiem:

- produkta neizdarīto lietu saraksts (PB);
- iterācijas darbu saraksts (SB);
- iterācijas izdarīto lietu saraksts (BD);
- laidiena plāns (RP);
- laidiena izdarīto lietu saraksts (RBD).

3.79. tabula

PB artefakta atribūti

Nosaukums
PB eksistē
Komandai ir skaidrs kādam nolūkam kalpo PB
PO ir skaidrs kādam nolūkam kalpo PB
ScrumMaster ir skaidrs kādam nolūkam kalpo PB
PB tiek uzturēts aktuāls
Visiem ir skaidrs, kurš ir atbildīgs par PB uzturēšanu
PB ir viegli pieejams
PB kvalitāte tiek periodiski vērtēta

3.80. tabula

SB artefakta atribūti

Nosaukums
SB eksistē
Komandai ir skaidrs kādam nolūkam kalpo SB
PO ir skaidrs kādam nolūkam kalpo SB
ScrumMaster ir skaidrs kādam nolūkam kalpo SB
SB tiek uzturēts aktuāls
Visiem ir skaidrs, kurš ir atbildīgs par SB uzturēšanu
SB ir viegli pieejams
SB kvalitāte tiek periodiski vērtēta

3.81. tabula

BD artefakta atribūti

Nosaukums
BD eksistē
Komandai ir skaidrs kādam nolūkam kalpo BD
PO ir skaidrs kādam nolūkam kalpo BD
ScrumMaster ir skaidrs kādam nolūkam kalpo BD
BD ir viegli pieejams
BD atspoguļo patieso situāciju

3.82. tabula

RP artefakta atribūti

Nosaukums
RP eksistē
Komandai ir skaidrs kādam nolūkam kalpo RP
PO ir skaidrs kādam nolūkam kalpo RP
ScrumMaster ir skaidrs kādam nolūkam kalpo SB
RP tiek uzturēts aktuāls
Visiem ir skaidrs, kurš ir atbildīgs par RP uzturēšanu
RP ir viegli pieejams
RP kvalitāte tiek periodiski vērtēta

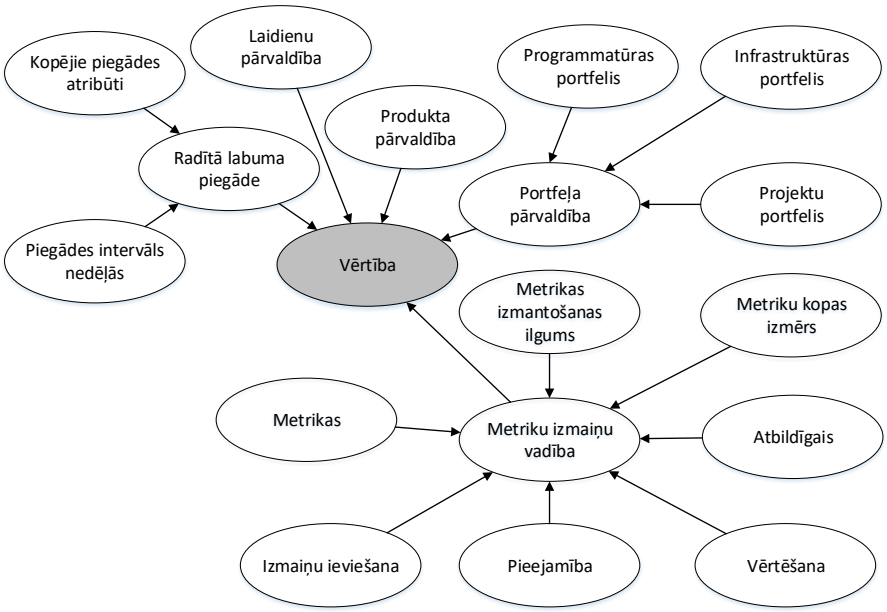
3.83. tabula

RBD artefakta atribūti

Nosaukums
RBD eksistē
Komandai ir skaidrs kādam nolūkam kalpo RBD
PO ir skaidrs kādam nolūkam kalpo RBD
ScrumMaster ir skaidrs kādam nolūkam kalpo RBD
RBD ir viegli pieejams
RBD atspoguļo patieso situāciju

3.1.5. Vērtību domēns

Vērtību domēnā ir apkopots un analizēts organizācijas apgabals, kurā notiek izlaistā produkta biznesa ieguvuma palielināšana. Vērtību domēns sastāv no pieciem apakšdomēniem (3.7. att.):



3.7. att. Vērtību domēns.

- laidienu pārvaldība (angl. *Release management*) ir programmatūras būvējuma vadīšanas, plānošanas un kontroles process dažādos posmos un vidēs [87];
- produkta pārvaldība (angl. *Product management*) ir programmatūras pārvaldība no idejas piedzimšanas līdz programmatūras nodošanai, tai skaitā produkta marketingu [87];
- portfeļa pārvaldība (angl. *Portfolio management*) ir atbildīga par uzņēmuma programmatūras pārvaldību. Portfelis satur informāciju par plānotajām vai esošajām iniciatīvām. Portfeļa pārvaldības vērtēšana sniedz iespēju izvērtēt iepriekšējos un esošos projektus:
    - programmatūras portfelis - administrē informāciju par organizācijā esošo programmatūru, tās cenu un uzturēšanas izmaksām;
    - infrastruktūras portfelis - administrē informāciju par organizācijas infrastruktūras elementiem. Daži infrastruktūras elementi tiek pievienoti projekta vai programmatūras portfelim, ja tie ir tiešā veidā saistīti ar tiem;
    - projektu portfelis - administrē informāciju par organizācijā esošajiem projektiem, plānotajiem projektiem, pagātnes projektiem.
- metriku izmaiņu vadība – atbild par metriku vērtēšanas, ieviešanas un izmaiņu procesiem:

    - atbildīgais – organizē metriku vērtēšanu un jaunu metriku ieviešanu vai maiņu;
    - vērtēšana – process, kura laikā tiek vērtēts, vai izmantojamās metrikas sniedz nepieciešamo rezultātu;
    - pieejamība – organizācijai un komandai ir jābūt pieejamai informācijai par izmantojamajām metrikām un to rezultātiem;
    - izmaiņu ieviešana – process, kura laikā, vadoties pēc vērtēšanas rezultātiem, tiek ieviestas jaunas vai samainītas esošās metrikas;
    - metrikas - sniedz iespēju izmērīt vērtību izmaiņas. Vērtību noteikšanai izmanto iepriekš noteiktu metriku kopu. Organizācija un komanda var izvēlēties konkrētu metriku kopu pirms katras vērtēšanas iterācijas. Rezultātu salīdzināšanai būtu ieteicams izmantot izvēlēto kopu vairākas reizes.
- radītā labuma piegāde – tiek izvērtēta radītā labuma piegāde pasūtītājam:
    - kopējie piegādes atribūti – piegādes atribūti, tādi kā, piegāde ir vienkārša, piegāde ir automatizēta utt.;
    - piegādes intervāls nedēļās – laiks, kurā tiek piegādāts produkta kārtējās papildinājums.

3.1.5.1. Portfeļa pārvaldība

Portfeļa pārvaldību var iedalīt trijās grupās. Par katru no portfeļu grupām tiek ievākti dažādi raksturlielumi. Darba kontekstā netiek aprakstīts kāds konkrēts portfeļa pārvaldības rīks un tā izvēle tiek pilnībā atstāta organizācijas un komandas ziņā.

- programmatūras portfelis:
    - ražotājs;
    - cena;
    - licenzēšanas nosacījumi;
    - iegādes datums;
    - citi izvēlēti parametri.
- infrastruktūras portfelis:
    - servera parametri;
    - servera atrašanās vieta;
    - cena vai nomas maksa;

- citi izvēlēti parametri.

- projektu portfelis:
  - projekta izmērs;
  - tehnoloģijas;
  - iesaistīto cilvēku saraksts;
  - izmaksas;
  - uzsākšanas datums;
  - beigu datums;
  - citi izvēlēti parametri.

3.84. tabula

Programmatūras portfeļa pārvaldības atribūti

Nosaukums
Programmatūras portfeļa pārvaldība uzņēmumā eksistē
Portfeļa uzturēšanai tiek izmantots kāds rīks vai datubāze
Ir definētas programmatūras portfeļa atbildīgais
Organizācijai un darbiniekiem ir skaidra programmatūras portfeļa nozīme
Darbiniekiem ir atbilstoša līmeņa pieeja programmatūras portfeļa rīkam

3.85. tabula

Infrastruktūras portfeļa pārvaldības atribūti

Nosaukums
Infrastruktūras portfeļa pārvaldība uzņēmumā eksistē
Portfeļa uzturēšanai tiek izmantots kāds rīks vai datubāze
Ir definēts infrastruktūras portfeļa atbildīgais
Organizācijai un darbiniekiem ir skaidra infrastruktūras portfeļa nozīme
Darbiniekiem ir atbilstoša līmeņa pieeja infrastruktūras portfeļa rīkam

3.86. tabula

Projektu portfeļa atribūti

Nosaukums
Projektu portfeļa pārvaldība uzņēmumā eksistē
Portfeļa uzturēšanai tiek izmantots kāds rīks vai datubāze
Ir definētu projektu portfeļa atbildīgais
Organizācijai un darbiniekiem ir skaidra projektu portfeļa nozīme
Darbiniekiem ir atbilstoša līmeņa pieeja projektu portfeļa rīkam

3.1.5.2. Produktu pārvaldība

Produkta pārvaldības atribūti ir saistīti ar produkta pārvaldības pieejamību organizācijā. Darba kontekstā netiek apskatīts kāds konkrēts produktu pārvaldības rīks.

3.87. tabula

Produktu pārvaldības atribūti

Nosaukums
Produktu pārvaldība uzņēmumā eksistē
Produktu pārvaldības informācijas uzturēšanai tiek izmantots kāds rīks vai datubāze
Ir definēts produktu pārvaldības atbildīgais
Organizācijai un darbiniekiem ir skaidra produktu pārvaldības nozīme
Darbiniekiem ir atbilstoša līmeņa pieeja produktu pārvaldības rīkam

3.1.5.3. Laidienu pārvaldība

Laidienu pārvaldība ir process, kura laikā tiek plānots, kādi laidieni tiks realizēti un kāds būs to saturs. Darba kontekstā netiek apskatīts kāds konkrēts laidienu plānošanas rīks.

3.88. tabula

Laidienu pārvaldības atribūti

Nosaukums
Laidienu pārvaldība uzņēmumā eksistē
Laidienu informācijas uzturēšanai tiek izmantots kāds rīks vai datubāze
Ir definēts laidienu pārvaldības atbildīgais
Organizācijai un darbiniekiem ir skaidra laidienu pārvaldības nozīme
Darbiniekiem ir atbilstoša līmeņa pieeja laidienu plānošanas rīkam

3.1.5.4. Metriku izmaiņu vadība

Metriku izmaiņu pārvaldību var iedalīt vairākos apakšdomēnos:

- pieejamība;
- atbildīgais;
- vērtēšana;
- izmaiņu ieviešana;
- metriku kopas izmērs;
- metrikas izmantošanas periods mēnešos.

3.89. tabula

Procesa pieejamības atribūti

Nosaukums
Procesi uzņēmumā eksistē
Procesa dokumentācija ir pieejama visiem
Procesi ir skaidri un dokumentēti visiem saprotamā veidā

3.90. tabula

Atbildīgā atribūti

Nosaukums
Ir noteikts atbildīgais, kurš rūpējas par procesa dokumentāciju
Visiem ir zināms, kurš ir atbildīgais
Atbildīgajam ir visas tiesības veikt izmaiņas

3.91. tabula

Vērtēšanas atribūti

Nosaukums
Ir definēti metriku vērtēšanas kritēriji
Ir definēta metriku vērtēšanas procedūra
Informācija par praksēm un to darbības rezultātiem tiek saglabāta, lai būtu pieejama nākošā vērtēšanas iterācijā
Metrikas tiek periodiski vērtētas
Metriku vērtēšanas rezultāti ir pieejami visiem

3.92. tabula

Izmaiņu ieviešanas atribūti

Nosaukums
Ir definētas izmaiņu ieviešanas procedūras
Informācija par veiktajām izmaiņām tiek piereģistrēta
Ir definēts atbildīgais par konkrēto izmaiņu ieviešanu

Parasti komandas izmanto vairākas metrikas, kuras palīdz noteikt esošo progresu. Problēmas rodas gadījumos, ja tiek izmantots pārāk liels vai pārāk neliels metriku skaits, kas traucē precīzi veikt mērījumus. Atkarībā no komandas veida, izmēra vai organizācijas ir jāizvēlas dažādas metrikas.

3.93. tabula

Metriku kopas izmērs

Vērtība
$X < 3$
$3 < X < 6$
$6 < X < 9$
$9 < X < 12$
$X > 12$

3.94. tabula

Metrikas lietošanas ilgums (mēneši)

Vērtība
$X < 1$
$1 < X < 6$
$6 < X < 12$

Vērtība
12 < X < 24
X > 24

3.1.5.5. Radītā labuma piegāde

Radītā labuma piegāde ir viens no svarīgākajiem spējo metožu pamatnosacījumiem [7]. Radītā labuma piegādes apakšdomēnu sīkāk iedala vairākos apakšdomēnos:

- kopējie piegādes atribūti;
- piegādes intervāls nedēļās.

3.95. tabula

Kopējie piegādes atribūti

Nosaukums
Piegāde ir vienkārša
Piegāde ir automatizēta
Pastāv iespēja atgriezt atpakaļ iepriekšējo versiju
Atbildīgais par piegādi ir definēts
Visi zina, kurš ir atbildīgais
Strādājošs un testēts papildinājums tiek piegādāts

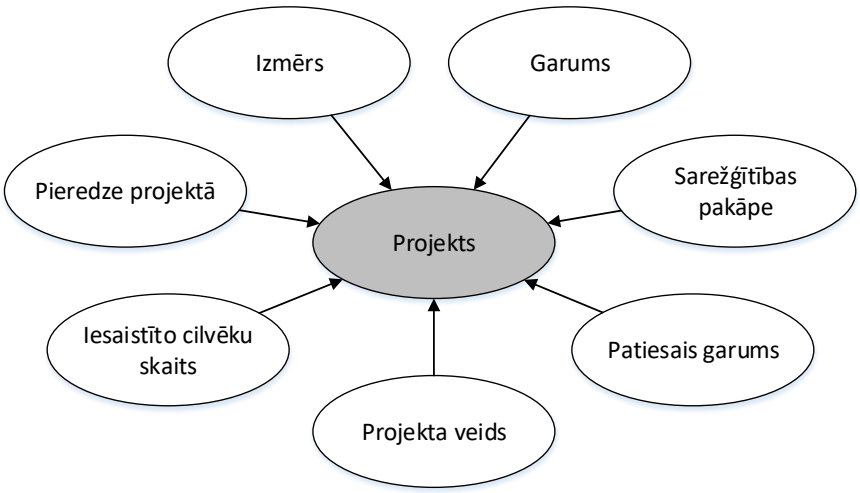
3.96. tabula

Piegādes intervāla (nedēļās) atribūti

Vērtība
X < 1
X = 1
X = 2
2 < X < 4
4 < X < 8
X > 8

3.1.6. Projektu domēns

Projektu domēna atbildība ir analizēt projektus, pie kuriem strādā organizācija. Katrs projekts, pie kura darbojas organizācija var atrasties citā spējības līmenī, un katram projektam ir nepieciešama sava pieeja (3.8. att.).



3.8. att. Projektu domēns.

Projektu domēns sastāv no 7 apakšdomēniem. Katrs apakšdomēns raksturo kādu no projekta raksturlielumiem:

- projekta veids – var identificēt 3 projektu veidus:
  - ūdenskrituma – klasiskā pieeja programmizstrādei;
  - iteratīvais un papildinošais – tiek izmantota kāda no iteratīvām pielāgotām pieejām;
  - spējīgs – pieeja balstās uz *Scrum* vai līdzvērtīgu metodi.
- iesaistīto cilvēku skaits – spējās metodes labāk darbojas ar nelielu cilvēku skaitu un lielas komandas traucē sasniegt augstu spējības līmeni. Lai sasniegtu augstu spējības līmeni lielās komandās, ir nepieciešams izmantot kādu no komandu dalīšanas praksēm;
- pieredze projektā – viens no spējo metožu stūrakmeņiem ir pieredzējusi komanda. Pieredzi var iedalīt vairākās apakšgrupās, piemēram, pieredze projektā, pieredze ar tehnoloģiju utt.. Darbinieks var būt pieredzējis konkrētā tehnoloģijā, bet viņam var nebūt pieredzes konkrētajā projektā;
- izmērs – projekta izmērs tiešā veidā ietekmē projekta spējību. Mazos projektus ir vieglāk realizēt spējā veidā nekā lielus projektus. Lieliem projektiem ir nepieciešama lielāka uzraudzība;
- garums – laiks, cik ilgi tiks, izstrādāts konkrētais projekts;
- sarežģītības līmenis – definē, cik sarežģīts ir izstrādājamais projekts. Projektos ar augstu sarežģītības līmeni ir nepieciešams pieaicināt darbiniekus ar lielāku pieredzi;

- patiesais garums – laiks, cik ilgi konkrētais projekts eksistē un tiek uzturēts. Organizācijas var pieņemt lēmumu, kādu no jau esošajiem projektiem sākt pārvaldīt, izmantojot spējās metodes. Šajos gadījumos ir svarīgi saprast, cik ilgi šis projekts tika uzturēts, izmantojot tradicionālās programmizstrādes metodes. Šis aspekts ir svarīgs, jo ilgi darbojošos projektos iesaistītājiem cilvēkiem ir tendence būt ļoti rezistentiem pret pārmaiņām un ir liela nosliece darīt visu ierastā veidā, tādā veidā apdraudot projekta spējību.

3.1.6.1. Projekta veids

Programmizstrādes projektus var realizēt, izmantojot dažādas pieejas. Atkarībā no realizācijas veida projektus ir izlemts iedalīt 3 projektu veidos (sk. 3.97. tabula).

3.97. tabula

Projekta veids

Nosaukums
Ūdenskrituma
Iteratīvs un papildinošs
Scrum bāzēts

3.1.6.2. Projektā iesaistīto personu skaits

Iesaistīto personu skaits var ietekmēt projekta spējību. Jo vairāk cilvēku ir iesaistīti projektā, jo lielāks ir risks, ka projekts nebūs spējīgs un būs nepieciešamas papildu darbības spējības līmeņa uzturēšanai (sk. 3.98. tabula).

3.98. tabula

Iesaistīto personu skaits

Vērtība
X < 10
10 < X < 20
20 < X < 40
X > 40

3.1.6.3. Pieredze projektā

Pieredze projektā raksturo to, cik ilgi darbinieks ir strādājis ar konkrēto projektu. Izmantojot šo atribūtu tiek iegūta informācija par visiem projektā iesaistītajiem darbiniekiem (sk. 3.99. tabula). Ideālā gadījumā visi projektā iesaistītie cilvēki tajā strādā no paša sākuma līdz beigām un ir pilnībā informēti par izmantotajām tehnoloģijām, arhitektūru un biznesa niansēm.

3.99. tabula

Pieredze projektā (% no kopējā projekta laika)

Vērtība
X < 20
20 < X < 40
40 < X < 60
60 < X < 80
80 < X < 100

3.1.6.4. Projekta izmērs

Izvērtējot projekta spējību ir jāņem vērā projekta izmērs. Mazos projektos spējību uzturēt ir vieglāk nekā lielos. Lieliem projektiem ir nepieciešama ciešāka uzraudzība. Dažādos avotos tiek minēti dažādi projekta izmēra klasifikācijas veidi [116][117][118]. Apkopojot iegūtos datus ir izlemts palikt pie 5 projekta izmēra veidiem (sk. 3.100. tabula).

3.100. tabula

Projekta izmērs

Nosaukums	Investīciju apjoms (USD)	Apraksts
Papildinājumi	X < 250,000	Nelieli papildinājumi projektam, kuriem nav nepieciešama nopietna uzraudzība vai dokumentācija
Mazs	250 < X < 1M	Neliels projekts ar minimālu dokumentāciju un uzraudzību
Vidējs	1M < X < 3M	Vidēji liels projekts ar nelielu dokumentācijas apjomu un uzraudzību. Ir nepieciešama vairāk formāla pieeja.
Liels	3M < X < 10M	Liels un apjomīgs projekts. Ir nepieciešams vairāk dokumentācijas un vairāk formālas pieejas.
Ļoti liels	X > 10M	Projekts ir ļoti liels ar daudziem moduļiem un sarežģītām integrācijām. Ir nepieciešama detalizētāka dokumentācija un vairāk formalizēta pieeja.

3.1.6.5. Projekta garums

Programmizstrādes projektiem ir definēts sākums un beigas, un tos ir iespējams iedalīt sīkākās apakšgrupās vadoties pēc to garuma (sk. 3.101. tabula). Projekta garums bieži, bet ne vienmēr ir saistīts ar tā izmēru un sarežģītības līmeni. Projekta garumu ieteicams aplūkot konkrētās organizācijas un komandas kontekstā. Piemēram, ja organizācija ir strādājusi ar projektiem, kuru garums nepārsniedz 3 mēnešus un tagad vēlas realizēt vairākus gadus garu projektu, tad jaunais projekts organizācijai ir uzskatāms par lielu.

3.101. tabula

Projekta garums

Nosaukums
Īss
Vidēji garš
Garš

3.1.6.6. Projekta sarežģītības līmenis

Projektiem mēdz būt dažādi sarežģītības līmeņi. Sarežģītības līmeni ietekmē izmantojamā tehnoloģija, izstrādājamo komponenšu skaits, integrējamo sistēmas komponenšu skaits utt.. Darba ietvaros ir izlemts izmantot 3 sarežģītības līmeņus izstrādājamām sistēmām.

3.102. tabula

Projekta sarežģītības līmenis

Nosaukums
Zems
Vidējs
Augsts

3.1.6.7. Projekta patiesais garums

Projekta patiesais garums ir laiks, kurā eksistē projekts. Piemēram, kādam no esošajiem projektiem tiek veikti papildinājumi, kuru realizācijai būs nepieciešams 1 gads. Esošais projekts ir bijis izstrādē jau 5 gadus. Šajā gadījumā projekta patiesais garums ir 6 gadi (sk. 3.103. tabula).

3.103. tabula

Projekta patiesais garums (mēneši)

Vērtība
X < 6
6 < X < 1 2
12 < X < 24
24 < X < 36
36 < X < 60
X > 60

3.2. OSM modeļa paplašināšanas iespējas

Organizācijām ir iespēja modificēt OSM modeli, lai tas vairāk atbilstu viņu prasībām. Gadījumā, ja ir veiktas OSM modeļa izmaiņas, organizācijai ir nepieciešams

veikt modificēto elementu vērtību noteikšanu. Darbā ir izmantotas vairākas atribūtu vērtēšanas skalas:

- **Skala ar atribūta nosaukumu un aprakstu** – skala tiek izmantota aprakstot atribūtus, kuriem nav konkrēta vērtības diapazona, piemēram 3.13. tabula un 3.18. tabula. Šāda skala tiek izmantota, lai novērtētu vienu konkrētu elementu. Piemēram, lai pārliecinātos, ka organizācijā eksistē, noteikts process;
- **Skala ar vērtību diapazonu** – skala tiek izmantota atribūtiem, kuriem vērtēšanas rezultātā izvēlas tikai vienu vērtību no piedāvātā diapazona, piemēram 3.2. tabula un 3.22. tabula. Noder tādos vērtējumos kā pieredze gados vai finansējuma apmērs.

Izmantojot norādītos skalu veidus, organizācijas un komandas var papildināt vērtējamo atribūtu kopu, ja rodas tāda nepieciešamība.

3.3. Prakšu ietekme uz OSM

Darba 2. nodaļā ir apskatītas spējās prakses ar augstāko AI. Katra no šīm praksēm ietekmē vienu vai vairākus OSM elementus, un šo prakšu izmantošana var uzlabot organizācijas spējību atbilstošajā OSM elementā. Prakšu atbilstība domēniem ir veidota, vadoties no prakšu izmantošanas aprakstiem, autora personīgās pieredzes un diskusiju laikā ar ekspertiem. Domēnu nosaukumu saīsināšanas nolūkos 3.104. tabula tiek izmanti domēnu nosaukumu saīsinājumui:

- OD – Organizācijas domēns;
- PD – Izstrādes domēns;
- KD – Kvalitātes domēns;
- PRD – Procesu domēns;
- VD – Vērtību domēns;
- PRJD – Projektu domēns.

3.104. tabula

Spējās prakses un to ietekmes apgabali

Prakse	OD	PD	KD	PRD	VD	PRJD
Test Driven Development		x	x	x		
Retrospective	x	x		x		
Review		x		x		

Prakse	OD	PD	KD	PRD	VD	PRJD
INVEST		x			x	
Code Refactoring		x	x			
User Stories		x			x	
Continuous Deployment		x		x	x	
Backlog Usage		x			x	
Pair Programming		x	x			
Measurements	x	x	x	x	x	x
Automated Testing			x			
Automated Unit Testing			x			
Continuous Integration		x			x	
Acceptance Testing			x			
Release Planning					x	x
Iteration Planning		x		x		
Estimation		x		x	x	
Daily Stand Up Meeting		x		x		
Behavior Driven Development		x		x		
Planning Game		x		x		
Working Software		x	x		x	
Source Control	x	x		x		x
Active Stakeholder Participation	x	x	x	x	x	
Definition of Done		x				
Emergent Design		x		x		
Exploratory Testing			x			
Facilitation				x		
Cross-Functional Team		x				
Usability Testing			x			
Code Review		x	x			
Story Mapping		x		x		
Sustainable Pace		x	x		x	
Kanban Board		x	x	x		
Acceptance Test Driven Development			x			
Automated Build		x	x	x		

3.4. Atslēgas vārdu sadalījums pa OSM elementiem

Promocijas darba 1.4.3 nodaļā ir apkopoti rezultāti par konferencēs minētajiem atslēgas vārdiem. Šajā apakšnodaļā ir apkopota informācija par atslēgas vārdu sadalījumu pa OSM augšējā līmeņa elementiem. Šāds apkopojums sniedz priekšstatu par organizācijas domēniem, par kuriem ir pastiprināta interese. Augšējā līmeņa domēnu apzīmējumiem tiek izmantoti saīsinājumi, kuri definēti 3.3 nodaļā.

3.105. tabula

Atslēgas vārdu sadalījums pa OSM elementiem

Atslēgas vārds	OD	PD	KD	PRD	VD	PRJD
Agile adoption	x	x	x	x	x	x
Experience report	x	x	x	x	x	x
Agile team		x	x	x	x	
Practices		x	x			
Testing		x	x			
Leadership		x				
Organizational culture	x	x				
Tools	x	x	x	x	x	x

Atslēgas vārds	OD	PD	KD	PRD	VD	PRJD
Business value		x	x		x	
Customer	x	x				
Distributed agile	x	x				
Development		x	x			
Learning	x	x				
Large scale agile	x	x		x		x
Transition	x	x	x	x	x	x
Quality		x	x			
Collaboration	x	x				
Communication	x	x				
Organization	x					
Coaching	x	x	x			
Enterprise	x					x
User experience		x	x			
Environment	x	x				
Planning	x			x		x
Product management	x			x		x
Requirements		x	x		x	
Teambuilding	x	x				
Project management	x					x
Problems	x	x	x	x	x	x
Research	x	x	x	x	x	x
Hands on labs	x	x	x	x	x	x
Business	x				x	x
Mentoring		x				
Innovation	x	x	x	x	x	x
Principles	x	x	x	x	x	x
Scaling agile	x	x		x	x	x

Daži atslēgas vārdi attiecas tikai uz vienu domēnu, bet ir arī tādi, kuri attiecas uz visiem domēniem. Piemēram, “Innovation”, “Research” un “Problem”. Lielākā daļa atslēgas vārdu ir attiecināmi uz Izstrādes domēnu (30), Organizācijas domēnu (26) un Kvalitātes domēnu (18), kas savā ziņā saskan arī ar personīgo pieredzi, jo tieši šajos domēnos uzņēmumiem ir būtiskas problēmas.

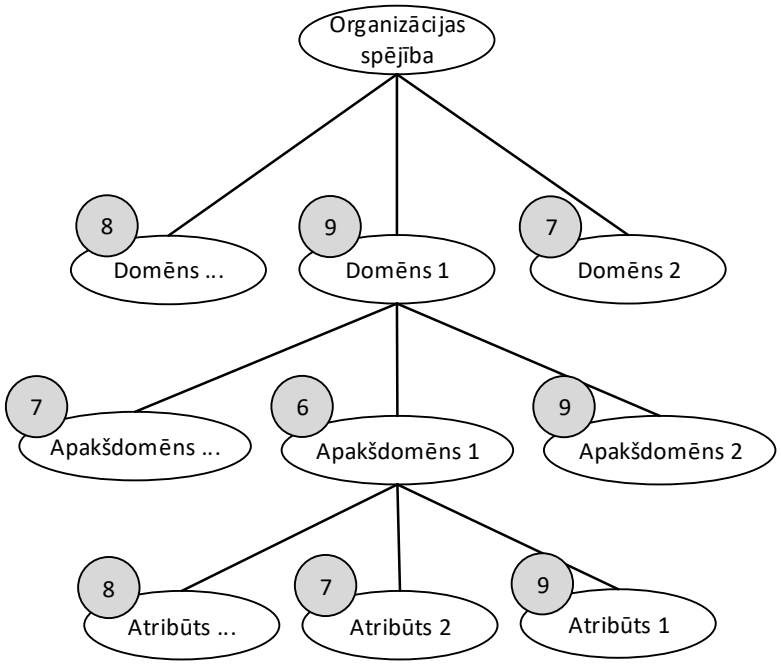
3.5. Domēnu, apakšdomēnu un atribūtu SII noteikšana

Pēc domēnu, apakšdomēnu un to atribūtu (DAA) noteikšanas, var noteikt to spējības ietekmes indeksu (SII).

3.5.1. Spējības ietekmes indekss

SII noteikšana ir nozīmīga darba sastāvdaļa, jo visu domēnu, apakšdomēnu un atribūtu SII nav vienādi. SII nosaka, cik lielā mērā konkrētais DAA elements ietekmē organizācijas spējību. Darbā tiek izmantota skala no 0 līdz 10, kas ir modificēta Likerta skala [122], kur 5 nozīmē, ka elements nedz uzlabo, nedz pasliktina spējības līmeni. Vērtības virs 5 uzlabo spējību un vērtības zem 5 pasliktina spējību. Modificētā skala ar

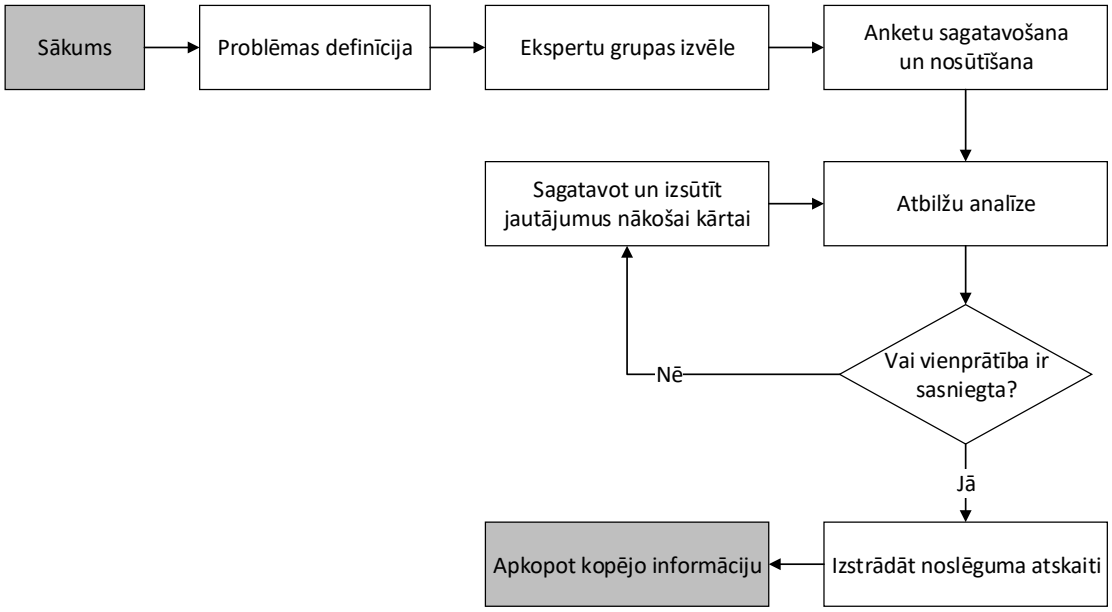
0 tiek izmantota balstoties uz organizācijas *Primary Intelligance* pētījumu [123], kurā ir noskaidrots, ka "0" palīdz intervējamajam ātrāk saprast skalas virzienu. Kā arī, 11 punktu skalai piemīt "patiesais viduspunkts", un ekspertiem ir vairāk iespēju izvēlēties vērtību nekā izmantojot 5 punktu skalu. Piemēram, izmantojot 5 punktu skalu, eksperta vērtējums "ietekmē spējību" tiktu novērtēts ar 4 vai 5, bet izmantojot 11 punktu skalu vērtējums varētu būt 6 vai 8, kur 6 norādītu, ka spējība tiek ietekmēta nedaudz, bet 8 vai 9, ka spējība tiek ietekmēta ievērojami. DAA tiek atveidots kokveida struktūrā (3.9. att.), kur katram zaram ir sava SII vērtība.



3.9. att. DAA kokveida struktūra ar parauga SII vērtībām.

3.5.2. SII noteikšanas metode

Domēnu, apakšdomēnu un atribūtu svara noteikšanai ir iespējams izmantot ekspertu aptaujas metodes [99]. Viena no šādām metodēm ir DELPHI, kurā tiek izmantota ekspertu grupas mijiedarbība vienota rezultāta iegūšanai [100]. Metode sastāv no procedūru virknes un ir organizēta iteratīvā veidā, un atkārtota līdz brīdim, kamēr tiek sasniegta zināma līmeņa vienprātība (sk. 3.10. att.) [101].



3.10. att. DELPHI metodes procesu plūsma.

Metode DELPHI ir vienkārš, bet darbietilpīgs veids, kā iegūt informāciju no ekspertiem. Lielākās problēmas rodas ar pareizu jautājumu sagatavošanu un nepieciešamās ekspertu grupas atrašanu. Spējo metožu ekspertus var atrast, bet pārsvarā to darba grafiki ir ļoti noslogoti un ne vienmēr var paļauties uz laicīgi sniegtu atbildi.

3.5.3. Ekspertu grupas formēšana

Ekspertu grupas formēšana ir sarežģīts uzdevums, jo bija nepieciešams izveidot grupu no 10 līdz 20 ekspertiem [99], kuriem ir līdzvērtīga kompetence. Ekspertu grupas izveidi apgrūtina fakts, ka ekspertiem ir jābūt pieejamiem pētījuma veikšanas laikā. Ņemot vērā spējo ekspertu aizņemtību, intervēšanas process tika organizēts interneta vidē.

Ekspertu grupas sastāvu ir iespējams definēt ar izteiksmi

$$EG = \{S, K, M\}, \tag{2}$$

kur *EG* – ekspertu grupa;

S – eksperti, kuri pilda, vai ir pildījuši *ScrumMaster* lomu;

K – eksperti, kuri piedalījušies spējo projektu realizēšanā (parasti spējās komandas sastāvā);

M – eksperti ar plašām zināšanām par spējām metodēm. Spējās metodes izmanto ikdienā dažādu projektu realizēšanai. Ir pieredzējuši vairāku organizāciju un komandu pārveidošanos uz spējo metožu izmantošanu programmatūras izstrādē.

Ekspertu atrašanai un piesaistīšanai ir izmantoti autora profesionālās darbības laikā satiktie spējie eksperti, kā arī tiem zināmie spējie eksperti. Spējo ekspertu loka paplašināšanai izmantoti cilvēki, kuri saistīti ar dalību konferencēs, kurās tiek apspriestas spējās metodes un to praktiskā lietošana.

Piesaistīto ekspertu pieredze darbā ar spējām metodēm ir no 5 līdz 15 gadiem. Detalizētāki dati ir apskatāmi 3.106. tabula.

3.106. tabula

Ekspertu sadalījums

Eksperti	**Pieredze (gadi)**	**Grupa**
Eksperts 1	7	*K, M*
Eksperts 2	15	*S, K, M*
Eksperts 3	5	*K*
Eksperts 4	7	*S, K*
Eksperts 5	7	*K*
Eksperts 6	7	*S, K*
Eksperts 7	7	*K*
Eksperts 8	7	*K*
Eksperts 9	7	*K*
Eksperts 10	7	*K*
Eksperts 11	10	*S, K*
Eksperts 12	6	*S*
Eksperts 13	7	*K*
Eksperts 14	8	*S, M*
Eksperts 15	9	*S, M*

3.5.4. Anketas sastādīšana

Parasti ekspertu grupa izvēlas, kāda būs rādītāju un atribūtu kopa pirms to vērtēšanas. Darba kontekstā ekspertu grupa vērtē anketēšanas organizatora sastādīto DAA, un nosaka katra DAA elementa SII vērtību. Ņemot vērā ekspertu novērtējumu, DAA elementi tiek pārskatīti un nepieciešamības gadījumā tiek veiktas izmaiņas DAA struktūrā.

Ekspertu vērtēšanai ir nodots 141 domēns/apakšdomēns un 578 atribūti. Pilns vērtējamo elementu saraksts ir atrodams 5. pielikumā.

3.5.5. Anketas aizpildīšana

Katram ekspertam tika izsūtīts savs lietotāja vārds un parole. Eksperts sev izdevīgā laikā varēja veikt anketas aizpildīšanu, kā arī pievienot savus komentārus.

Katrs DAA elements tiek vērtēts skalā no 0 līdz 10:

- 10, 9 – elements būtiski ietekmē spējību;
- 8, 7 – elements vērā ņemami ietekmē spējību;
- 6 – elements nedaudz uzlabo spējību;
- 5 – elements ir neitrāls (nepasliktina un ne uzlabo spējību);
- 4 - elements nelielā mērā pasliktina spējību;
- 3, 2 – elements vērā ņemami pasliktina spējību;
- 1, 0 – elements būtiski pasliktina spējību.

Katras iterācijas rezultāti tiek apstrādāti un paziņoti ekspertiem, izmantojot anketēšanas vietni. Anketēšanas iterācijas tiek atkārtotas līdz brīdim, kad tiek sasniegts uzstādītais ekspertu saskaņotības līmenis. DAA elementu SII vērtību novērtēšanai bija nepieciešamas divas iterācijas. Par saskaņotības līmeni tiek uzskatīts brīdis, kad aprēķinātais saskaņotības koeficients *Sk* ir mazāks par viens. Saskaņotības koeficientu aprēķina ar vienādojumu

$$Sk = \max\{E_i\} - \text{avg}\{E_i\}, \tag{14}$$

kur *Sk* – Saskaņotības koeficients;
E_i – Eksperta vērtējums elementam i.

Otrās iterācijas beigās tika sasniegts nepieciešamais ekspertu saskaņotības līmenis, ka visiem OSM elementiem nosacījums $Sk < 1$ ir patiess.

3.5.6. Rezultāti

Darba pamattekstā ir attēlota apkopotā informācija par DAA elementu SII vērtībām (sk. 3.107. tabula). Detalizēta informācija ar starprezultātiem un vērtēšanas iterāciju gaitu ir atrodama 5**Error! Reference source not found.**. pielikumā.

3.107. tabula

DAA elementu ekspertu noteiktās SII vērtības

Kods	Nosaukums	SII
1	Org. domēns	7

Kods	Nosaukums	SII
1.1	Izmērs	6
1.1.1	Liels	4
1.1.2	Vidēja izmēra	5
1.1.3	Mazs	7
1.1.4	Mikro	6
1.2	Mācīšanās	8
1.2.1	Darbiniekiem atvēlētais laiks mācīties (Stundas/gadā)	7
1.2.1.1	x = 0	1
1.2.1.2	1 < x < 8	3
1.2.1.3	9 < x < 16	4
1.2.1.4	16 < x < 40	5
1.2.1.5	40 < x < 80	7
1.2.1.6	x > 80	7
1.2.1.7	Nezināms	3
1.2.2	Apmācību finansējums uz vienu darbinieku (EUR/gadā)	6
1.2.2.1	x = 0	2
1.2.2.2	1 < x < 100	4
1.2.2.3	100 < x < 200	5
1.2.2.4	200 < x < 300	6
1.2.2.5	300 < x < 400	7
1.2.2.6	400 < x < 500	7
1.2.2.7	500 < x 1000	8
1.2.2.8	1000 < x < 2000	7
1.2.2.9	2000 < x < 3000	8
1.2.2.10	x > 3000	8
1.2.2.11	Nezināms	1
1.2.3	Sertifikācijas eksāmenu finansējuma apmērs (EUR/gadā)	6
1.2.3.1	x = 0	0
1.2.3.2	1 < x < 100	4
1.2.3.3	100 < x < 200	5
1.2.3.4	200 < x < 300	6
1.2.3.5	300 < x < 400	6
1.2.3.6	x > 400	6
1.2.3.7	Nezināms	2
1.3	Pieredze	8
1.3.1	Pieredzes laiks gados par metodi (Y)	7
1.3.1.1	x = 0	3
1.3.1.2	x < 1	4
1.3.1.3	1 < x < 2	5
1.3.1.4	2 < x < 5	7
1.3.1.5	x > 5	7
1.3.1.6	Nezināms	1
1.4	Komunikācija	8
1.4.1	Komunikācijas veids	8
1.4.1.1	Aci pret aci	8
1.4.1.2	Pa telefonu	5
1.4.1.3	Rakstiskā veidā	5
1.4.1.4	Skype vai alternatīva	5
1.4.1.5	Videokonferences	6
1.4.2	Komunikācijas vērtēšanas atribūti	6
1.4.2.1	Vērtēšanas kritēriji ir definēti	4

Kods	Nosaukums	SII
1.4.2.2	Komunikācijas vērtēšana tiek veikta periodiski	6
1.4.2.3	Vērtēšanas rezultāti ir pieejami visiem	6
1.4.2.4	Vērtēšanas rezultāti tiek saglabāti	4
1.4.3	Komunikācijas biežuma atribūti	7
1.4.3.1	Vairākas reizes dienā	7
1.4.3.2	Vienreiz dienā	5
1.4.3.3	Vairākas reizes nedēļā	3
1.4.3.4	Vienreiz nedēļā	2
1.4.3.5	Retāk nekā reizi nedēļā	1
1.4.3.6	Dažas reizes mēnesī	0
1.4.3.7	Vienreiz mēnesī	0
1.4.3.8	Retāk nekā reizi mēnesī	0
1.5	Procesu izmaiņu vadība	5
1.5.1	Procesu pieejamības atribūti	4
1.5.1.1	Process eksistē	5
1.5.1.2	Procesa dokumentācija ir pieejama visiem	4
1.5.1.3	Procesi ir skaidri un dokumentēti visiem saprotamā veidā	5
1.5.2	Atbildīgā atribūti	5
1.5.2.1	Atbildīgais par procesu dokumentāciju ir noteikts	6
1.5.2.2	Visi zin kurš ir atbildīgais	7
1.5.2.3	Atbildīgajam ir visas tiesības veikt izmaiņas	5
1.5.3	Vērtēšanas atribūti	4
1.5.3.1	Ir definēti procesu vērtēšanas kritēriji	4
1.5.3.2	Ir definēta procesu vērtēšanas procedūra	4
1.5.3.3	Informācija par procesiem un to darbības rezultātiem tiek saglabāta, lai būtu pieejama nākošā vērtēšanas iterācijā	4
1.5.3.4	Procesi tiek periodiski vērtēti	6
1.5.3.5	Procesu vērtēšanas rezultāti ir pieejami visiem	5
1.5.4	Izmaiņu ieviešanas atribūti	5
1.5.4.1	Ir definētas izmaiņu ieviešanas procedūras	4
1.5.4.2	Informācija par veiktām izmaiņām tiek piereģistrēta	4
1.5.4.3	Ir definēts atbildīgais par konkrēto izmaiņu ieviešanu	7
1.6	Komandas veidošanas process	8
1.6.1	Komandu organizēšanai ir piesaistīts atbildīgais	7
1.6.2	Piesaistītajam atbildīgajam ir pieredze komandu veidošanā	7
1.6.3	Informācija par darbinieku zināšanām tiek uzglabāta	6
1.6.4	Informācija par darbinieku zināšanām tiek regulāri atjaunota	7
1.6.5	Tiek ņemta vērā darbinieka pieredze	7
1.6.6	Tiek ņemtas vērā darbinieka zināšanas	6
1.6.7	Tiek ņemta vērā iepriekšējā dalībnieku sadarbošanās	5
1.6.8	Iepriekšējās sadarbības rezultāti tiek uzglabāti	5
1.6.9	Tiek veidotas komandas ar augstu motivācijas līmeni	9
1.6.10	Tiek veidota pašorganizējoša komanda	9
1.7	Mērķi	7
1.7.1	Viena no augstākajām prioritātēm ir apmierināt klientu	8
1.7.2	Būtiska ir agra un nepārtraukta piegāde	8
1.7.3	Piegādāt vērtīgu programmatūru pēc iespējas biežāk	8
2	Prod. domēns	8
2.1	Komunikācija	9
2.1.1	Komunikāciju veids	8
2.1.1.1	Aci pret aci	8

Kods	Nosaukums	SII
2.1.1.2	Telefoniski	6
2.1.1.3	Rakstiski	5
2.1.1.4	Skype vai alternatīva sarunu noorganizēšana	6
2.1.1.5	Videokonferences	5
2.1.2	Komunikācijas vērtēšanas atribūti	7
2.1.2.1	Vērtēšanas kritēriji ir definēti	4
2.1.2.2	Komunikācijas vērtēšanas tiek veikta regulāri	7
2.1.2.3	Vērtēšanas kritēriji ir pieejami visiem	6
2.1.2.4	Vērtēšanas kritēriji ir saglabāti	4
2.1.3	Komunikācijas biežuma atribūti	7
2.1.3.1	Vairākas reizes dienā	8
2.1.3.2	Reizi dienā	6
2.1.3.3	Vairākas reizes nedēļā	4
2.1.3.4	Reizi nedēļā	3
2.1.3.5	Retāk nekā reizi nedēļā	2
2.1.3.6	Dažas reizes mēnesī	0
2.1.3.7	Reizi mēnesī	1
2.1.3.8	Retāk nekā reizi mēnesī	0
2.2	Zināšanu pārvaldība	6
2.2.1	Zināšanu uzglabāšanas atribūti	6
2.2.1.1	Zināšanas tiek uzglabātas	5
2.2.1.2	Zināšanas tiek regulāri atjaunotas	7
2.2.1.3	Zināšanas var saglabāt jebkurš no komandas	5
2.2.1.4	Zināšanas ir viegli atrast	6
2.2.1.5	Zināšanas ir labi strukturētas	5
2.2.1.6	Zināšanu saglabāšana ir vienkārša	5
2.2.1.7	Visai komandai ir pieeja saglabātajām zināšanām	6
2.2.1.8	Atbildīgais par zināšanu uzglabāšanas sistēmu ir definēts	6
2.2.1.9	Komanda dalās ar zināšanām	8
2.2.2	Zināšanu dalīšanās atribūti	7
2.2.2.1	Zināšanu dalīšanās veids	7
2.2.2.1.1	Blakus sēdošā programmētāja konsultēšana	7
2.2.2.1.2	Pārējo programmētāju konsultēšana	5
2.2.2.1.3	Kopēju lekciju organizēšana	6
2.2.2.1.4	Koda pārskata organizēšana	7
2.2.2.1.5	Kopēja koda rakstīšanaz pasākuma organizēšana	6
2.2.2.2	Zināšanu dalīšanās biežuma atribūti	6
2.2.2.2.1	Reizi stundā	6
2.2.2.2.2	Vairākas reizes stundā	6
2.2.2.2.3	Vairākas reizes dienā	6
2.2.2.2.4	Reizi nedēļā	5
2.2.2.2.5	Vairākas reizes nedēļā	4
2.2.2.2.6	Reizi mēnesī	4
2.2.2.2.7	Vairākas reizes mēnesī	4
2.2.2.2.8	Retāk par reizi mēnesī	4
2.3	Mācīšanās	8
2.3.1	Komanda runā par problēmām	7
2.3.2	Komanda runā par pozitīvām lietām	6
2.3.3	Problemātiskās lietas un to risinājumi tiek piereģistrēti	5
2.3.4	Pozitīvās lietas tiek piereģistrētas	5
2.3.5	Pozitīvās lietas tiek ņemtas vērā nākamajās iterācijās	7

Kods	Nosaukums	SII
2.3.6	Problēmas un to risinājumi tiek ņemti vērā nākamajās iterācijās	7
2.3.7	Piereģistrētā informācija ir ērti pieejama	5
2.4	Pieredze	7
2.4.1	Kopējā pieredze gados	6
2.4.1.1	X < 0,5	3
2.4.1.2	0,5> X >1	4
2.4.1.3	1 > X > 3	6
2.4.1.4	3 > X > 5	6
2.4.1.5	5 > X > 10	5
2.4.1.6	X > 10	6
2.4.2	Pieredze ar izmantojamo tehnoloģiju	7
2.4.2.1	X < 0,5	3
2.4.2.2	0,5> X >1	5
2.4.2.3	1 > X > 3	6
2.4.2.4	3 > X > 5	6
2.4.2.5	5 > X > 10	7
2.4.2.6	X > 10	7
2.5	Vide	7
2.5.1	X < 20	1
2.5.2	20 < X < 40	1
2.5.3	40 < X < 60	3
2.5.4	60 < X < 80	5
2.5.5	80 < 100	7
2.6	Prakšu izmaiņu pārvaldība	5
2.6.1	Procesa pieejamības atribūti	5
2.6.1.1	Procesi uzņēmumā eksistē	5
2.6.1.2	Procesa dokumentācija ir pieejama visiem	3
2.6.1.3	Procesi ir skaidri un dokumentēti visiem saprotamā veidā	4
2.6.2	Atbildīgā atribūti	6
2.6.2.1	Ir noteikts atbildīgais, kurš rūpējas par procesa dokumentāciju	4
2.6.2.2	Visiem ir zināms, kurš ir atbildīgais	7
2.6.2.3	Atbildīgajam ir visas tiesības veikt izmaiņas	4
2.6.3	Vērtēšanas atribūti	6
2.6.3.1	Ir definēti prakšu vērtēšanas kritēriji	5
2.6.3.2	Ir definēta prakšu vērtēšanas procedūra	4
2.6.3.3	Informācija par praksēm un to darbības rezultātiem tiek saglabāta	4
2.6.3.4	Prakses tiek vērtētas periodiski	7
2.6.3.5	Prakšu vērtēšanas rezultāti ir pieejami visiem	7
2.6.4	Izmaiņu ieviešanas atribūti	5
2.6.4.1	Ir definētas izmaiņu ieviešanas procedūras	4
2.6.4.2	Informācija par veiktām izmaiņām tiek piereģistrēta	5
2.6.4.3	Ir definēts atbildīgais par konkrēto izmaiņu ieviešanu	8
2.6.5	Prakšu kopas izmērs	5
2.6.5.1	X < 3	5
2.6.5.2	3 < X < 6	5
2.6.5.3	6 < X < 9	5
2.6.5.4	9 < X < 12	5
2.6.5.5	X > 12	5
2.6.6	Prakses izmantošana mēnešos	5
2.6.6.1	X < 1	4
2.6.6.2	1 < X < 6	5

Kods	Nosaukums	SII
2.6.6.3	6 < X < 12	6
2.6.6.4	12 < X < 24	6
2.6.6.5	X > 24	6
2.7	Sadarbība	7
2.7.1	X < 20	1
2.7.2	20 < X < 40	2
2.7.3	40 < X < 60	4
2.7.4	60 < X < 80	6
2.7.5	80 < X < 100	7
2.8	Motivācija	8
2.8.1	Komandas dalībnieki ir labi motivēti	8
2.8.2	Komandas motivācija tiek periodiski vērtēta	7
2.9	Komanda	8
2.9.1	Komandas veids	7
2.9.1.1	Dalīta komanda	3
2.9.1.2	Komanda atrodas vienā birojā	5
2.9.1.3	Komanda atrodas vienā telpā	6
2.9.1.4	Jaukta veida	3
2.9.2	Komandas izmērs	6
2.9.2.1	X < 3	4
2.9.2.2	3 < X < 5	6
2.9.2.3	5 < X < 7	6
2.9.2.4	7 < X < 9	7
2.9.2.5	X > 9	4
2.9.3	Komandas darba organizēšana	8
2.9.3.1	Komanda ir pašorganizējoša	8
2.9.3.2	Komanda nestrādā virsstundas	8
3	Kval. domēns	8
3.1	Vadlīnijas	6
3.1.1	Vadlīniju pieejamības atribūti	7
3.1.1.1	Vadlīnijas eksistē	6
3.1.1.2	Vadlīniju dokumentācija ir pieejama visiem	5
3.1.1.3	Vadlīnijas ir skaidras un dokumentētas visiem saprotamā veidā	5
3.1.2	Atbildīgā atribūti	5
3.1.2.1	Ir noteikts atbildīgais, kurš rūpējas par vadlīniju dokumentāciju	4
3.1.2.2	Visiem ir zināms, kurš ir atbildīgais	7
3.1.2.3	Atbildīgajam ir visas tiesības veikt izmaiņas	4
3.1.3	Vērtēšanas atribūti	6
3.1.3.1	Ir definēti vadlīniju vērtēšanas kritēriji	4
3.1.3.2	Ir definēta vadlīniju vērtēšanas procedūra	5
3.1.3.3	Informācija par vadlīnijām un to darbības rezultātiem tiek saglabāta, lai būtu pieejama nākošā vērtēšanas iterācijā	5
3.1.3.4	Vadlīnijas tiek periodiski vērtētas	7
3.1.3.5	Vadlīniju vērtēšanas rezultāti ir pieejami visiem	6
3.1.4	Izmaiņu ieviešanas atribūti	5
3.1.4.1	Ir definētas izmaiņu ieviešanas procedūras	5
3.1.4.2	Informācija par veiktām izmaiņām tiek piereģistrēta	5
3.1.4.3	Ir definēts atbildīgais par konkrēto izmaiņu ieviešanu	7
3.2	Vienošanās	7
3.2.1	Vienošanās pieejamības atribūti	6
3.2.1.1	Vienošanās eksistē	6

Kods	Nosaukums	SII
3.2.1.2	Vienošanās ir pieejama un zināma visiem	5
3.2.1.3	Vienošanās ir skaidri definētas un visiem saprotamas	5
3.2.1.4	Ir definēta vienošanās izveidošanas un maiņas procedūra	4
3.2.1.5	Ir pieejama informācija par vēsturiskajām vienošanām	4
3.2.2	Vērtēšanas atribūti	6
3.2.2.1	Ir definēti vienošanās vērtēšanas kritēriji	5
3.2.2.2	Ir definēta vienošanos vērtēšanas procedūra	4
3.2.2.3	Informācija par vienošanām un to darbības rezultātiem tiek saglabāta, lai būtu pieejama nākošā vērtēšanas iterācijā	4
3.2.2.4	Vienošanās tiek periodiski vērtētas	7
3.2.2.5	Vienošanās vērtēšanas rezultāti ir pieejami visiem	5
3.3	Standarti	6
3.3.1	Standartu pieejamības atribūti	6
3.3.1.1	Standarti eksistē	6
3.3.1.2	Standartu dokumentācija ir pieejama visiem	5
3.3.1.3	Standarti ir skaidri un dokumentēti visiem saprotamā veidā	4
3.3.2	Atbildīgā atribūti	5
3.3.2.1	Ir noteikts atbildīgais, kurš rūpējas par standartu dokumentāciju	4
3.3.2.2	Visiem ir zināms, kurš ir atbildīgais	6
3.3.2.3	Atbildīgajam ir visas tiesības veikt izmaiņas	5
3.3.3	Vērtēšanas atribūti	6
3.3.3.1	Ir definēti standartu vērtēšanas kritēriji	5
3.3.3.2	Ir definēta standartu vērtēšanas procedūra	5
3.3.3.3	Informācija par standartiem un to darbības rezultātiem tiek saglabāta, lai būtu pieejama nākošā vērtēšanas iterācijā	4
3.3.3.4	Standarti tiek periodiski vērtēti	7
3.3.3.5	Standartu vērtēšanas rezultāti ir pieejami visiem	7
3.3.4	Izmaiņu ieviešanas atribūti	5
3.3.4.1	Ir definētas izmaiņu ieviešanas procedūras	4
3.3.4.2	Informācija par veiktām izmaiņām tiek piereģistrēta	4
3.3.4.3	Ir definēts atbildīgais par konkrēto izmaiņu ieviešanu	7
3.4	Rīki	7
3.4.1	Rīku vērtēšanas atribūti	6
3.4.1.1	Ir definēti instrumentu vērtēšanas kritēriji	4
3.4.1.2	Ir definēta instrumentu vērtēšanas procedūra	4
3.4.1.3	Ir noteikts atbildīgais vērtēšanas veikšanai	6
3.4.1.4	Vērtēšanas rezultāti ir pieejami visiem	6
3.4.2	Finansējuma uz vienu izstrādātāju atribūti (EUR/gadā)	6
3.4.2.1	X < 100	2
3.4.2.2	100 < X < 300	5
3.4.2.3	300 < X < 600	6
3.4.2.4	600 < X 1000	6
3.4.2.5	X > 1000	6
3.4.3	Finansējuma apmērs gadā organizācijas kontekstā (EUR/gadā)	6
3.4.3.1	X < 1000	0
3.4.3.2	1000 < X < 3000	3
3.4.3.3	3000 < X < 6000	5
3.4.3.4	6000 < X 10000	6
3.4.3.5	10000 < X < 20000	7
3.4.3.6	X > 20000	6
3.4.4	Instrumentu apmācībai atvēlētais laiks (Dienas/gadā)	6

Kods	Nosaukums	SII
3.4.4.1	X < 1	3
3.4.4.2	1 < X < 3	5
3.4.4.3	3 < X < 7	5
3.4.4.4	7 < X 14	6
3.4.4.5	X > 14	6
3.4.5	Rīku ieviešanas atribūti	5
3.4.5.1	Procedūra, kā ieviest rīku, ir definēta	4
3.4.5.2	Rīka ieviešanas atbildīgais ir definēts	7
3.4.5.3	Informācija par veiktajām izmaiņām tiek saglabāta	4
3.5	Prakšu izmaiņu pārvaldība	5
3.5.1	Procesu pieejamības atribūti	5
3.5.1.1	Procesi uzņēmumā eksistē	5
3.5.1.2	Procesa dokumentācija ir pieejama visiem	4
3.5.1.3	Procesi ir skaidri un dokumentēti visiem saprotamā veidā	4
3.5.2	Atbildīgā atribūti	5
3.5.2.1	Ir noteikts atbildīgais, kurš rūpējas par procesa dokumentāciju	5
3.5.2.2	Visiem ir zināms, kurš ir atbildīgais	6
3.5.2.3	Atbildīgajam ir visas tiesības veikt izmaiņas	4
3.5.3	Vērtēšanas atribūti	5
3.5.3.1	Ir definēti prakšu vērtēšanas kritēriji	4
3.5.3.2	Ir definēta prakšu vērtēšanas procedūra	4
3.5.3.3	Informācija par praksēm un to darbības rezultātiem tiek saglabāta, lai būtu pieejama nākošā vērtēšanas iterācijā	5
3.5.3.4	Prakses tiek periodiski vērtētas	8
3.5.3.5	Prakšu vērtēšanas rezultāti ir pieejami visiem	5
3.5.4	Izmaiņu ieviešanas atribūti	5
3.5.4.1	Ir definētas izmaiņu ieviešanas procedūras	4
3.5.4.2	Informācija par veiktām izmaiņām tiek piereģistrēta	4
3.5.4.3	Ir definēts atbildīgais par konkrēto izmaiņu ieviešanu	7
3.5.5	Prakšu kopas izmērs	5
3.5.5.1	X < 3	4
3.5.5.2	3 < X < 6	4
3.5.5.3	6 < X < 9	4
3.5.5.4	9 < X < 12	4
3.5.5.5	X > 12	4
3.5.6	Prakses lietošanas ilgums	6
3.5.6.1	X < 1	3
3.5.6.2	1 < X < 6	5
3.5.6.3	6 < X < 12	5
3.5.6.4	12 < X < 24	6
3.5.6.5	X > 24	7
3.6	Kompetences	7
3.6.1	Kompetenču sadalījuma atribūti	6
3.6.1.1	Vienam dalībniekam ir vairākas kompetences	3
3.6.1.2	Viena kompetence tiek dalīta starp vairākiem izpildītājiem	4
3.6.1.3	Katram dalībniekam ir sava kompetence	5
3.6.2	Datubāzes arhitekta atribūti	6
3.6.2.1	Kompetence eksistē	7
3.6.2.2	Datubāzes arhitektūra	7
3.6.2.3	Datubāzes ātrdarbība	7
3.6.2.4	Datubāzes dokumentācijas pieejamība	5

Kods	Nosaukums	SII
3.6.2.5	Datubāzes arhitekta pieejamība	5
3.6.3	Ietvaru izstrādātāja atribūti	6
3.6.3.1	Kompetence eksistē	6
3.6.3.2	Ietvara izmantojamība	6
3.6.3.3	Ietvara kvalitāte	6
3.6.3.4	Ietvara stabilitāte	7
3.6.3.5	Ietvara dokumentācijas pieejamība	5
3.6.3.6	Ietvara izstrādātāja pieejamība	5
3.6.4	Arhitekta atribūti	7
3.6.4.1	Kompetence eksistē	8
3.6.4.2	Sistēmas arhitektūra	7
3.6.4.3	Arhitektūras dokumentācijas pieejamība	5
3.6.4.4	Arhitektūras izmantojamība	7
3.6.4.5	Arhitekta pieejamība	7
3.6.5	UI Izstrādātāja atribūti	6
3.6.5.1	Kompetence eksistē	6
3.6.5.2	UI lietojamība	7
3.6.5.3	UI izstrādātāja pieejamība	6
3.6.6	Infrastruktūras izstrādātāju atribūti	7
3.6.6.1	Kompetence eksistē	8
3.6.6.2	Infrastruktūras stabilitāte	8
3.6.6.3	Infrastruktūras dokumentācijas pieejamība	5
3.6.6.4	Izstrādātāja pieejamība	7
3.6.6.5	Infrastruktūras monitoringa pieejamība	7
3.7	Fokusēšanās uz izcilību	8
3.7.1	Tiek pievērsta nepārtraukta uzmanība tehniskai izcilībai	8
3.7.2	Tiek pievērsta nepārtraukta uzmanība labam dizainam (tai skaitā arhitektūrai un struktūrai)	8
3.7.3	Vienkāršība ir svarīga	8
4	Proc. domēns	8
4.1	Laikā ierobežotas aktivitātes	8
4.1.1	Sagatavošanas sapulce	7
4.1.1.1	Sagatavošanas sapulces vispārīgie atribūti	7
4.1.1.1.1	Sapulce eksistē	6
4.1.1.1.2	Sapulce tiek veltīta lietotāju stāstu apspriešanai	7
4.1.1.1.3	Sapulce tiek veltīta sākotnējo vērtējumu noteikšanai	6
4.1.1.1.4	Tiek definēti sākotnējie pieņemšanas kritēriji	6
4.1.1.1.5	Sapulces tiek organizētas regulāri un norisinās paralēli iterācijām	6
4.1.1.1.6	Noteiktie sapulces ierobežojumi tiek ievēroti (piemēram, laiks)	6
4.1.1.1.7	Komandai ir skaidrs, kādam nolūkam sapulce eksistē	6
4.1.1.1.8	Komandai ir skaidrs, kas tai jādara sapulcē	6
4.1.1.1.9	PO ir skaidrs, kādam nolūkam sapulce eksistē	6
4.1.1.1.10	PO ir skaidri viņa pienākumi sapulces laikā	6
4.1.1.1.11	*ScrumMaster* ir skaidrs, kādam nolūkam sapulce eksistē	7
4.1.1.1.12	*ScrumMaster* ir skaidri viņa pienākumi sapulces laikā	7
4.1.1.2	Sagatavošanas sapulces ilguma atribūti (stundas)	5
4.1.1.2.1	X < 1	3
4.1.1.2.2	1 < X < 2	6
4.1.1.2.3	X > 2	4
4.1.1.3	Prasību atribūti	6
4.1.1.3.1	Izmaiņas tiek pieņemtas un realizētas arī vēlu izstrādē	8

Kods	Nosaukums	SII
4.1.1.3.2	Prasības tiek uzglabātas	6
4.1.1.3.3	Prasību dokumenti ir sabalansēti (Dokumenti nav lieli un iekļauj minimālo nepieciešamo informāciju)	6
4.1.2	Iterācijas plānošanas sapulce	8
4.1.2.1	Iterācijas plānošanas sapulces vispārīgie atribūti	8
4.1.2.1.1	Sapulce eksistē	8
4.1.2.1.2	Sapulce tiek veltīta gala vērtējumu noteikšanai	7
4.1.2.1.3	Tiek modificēti pieņemšanas kritēriji	7
4.1.2.1.4	Nepieciešamības gadījumā ir iespējams apspriest neskaidrības, ja tādas eksistē	7
4.1.2.1.5	Tiek izveidoti darba uzdevumi katram US	7
4.1.2.1.6	Komanda izvēlas darba uzdevumus	6
4.1.2.1.7	Noteiktie sapulces ierobežojumi tiek ievēroti (piemēram, laiks)	7
4.1.2.1.8	Komandai ir skaidrs, kādam nolūkam sapulce eksistē	7
4.1.2.1.9	Komandai ir skaidrs, kas tai jādara sapulcē	7
4.1.2.1.10	PO ir skaidrs, kādam nolūkam sapulce eksistē	7
4.1.2.1.11	PO ir skaidri viņa pienākumi sapulces laikā	7
4.1.2.1.12	*ScrumMaster* ir skaidrs, kādam nolūkam sapulce eksistē	7
4.1.2.1.13	*ScrumMaster* ir skaidri viņa piemākumi sapulces laikā	7
4.1.2.2	Iterācijas plānošanas sapulces ilguma atribūti	6
4.1.2.2.1	X < 1	3
4.1.2.2.2	1 < X < 2	5
4.1.2.2.3	2 < X < 4	5
4.1.2.2.4	4 < X < 6	5
4.1.2.2.5	X > 6	3
4.1.3	Iterācija	8
4.1.3.1	Iterācijas vispārīgie atribūti	8
4.1.3.1.1	Process eksistē	8
4.1.3.1.2	Komanda ir pasargāta no ārējas iedarbības iterācijas laikā	8
4.1.3.1.3	Definēts noteikts iterācijas garums	7
4.1.3.1.4	Pastāv vienošanās kā izņemt US no iterācijas, ja rodas problēmas	6
4.1.3.1.5	Pastāv vienošanās kā paņemt jaunus US iterācijā, ja rodas iespēja	6
4.1.3.1.6	Sākotnējie darba uzdevumi katram US ir definēti	7
4.1.3.1.7	Komanda ir vienojusies, kādā veidā tiek veidoti un modificēti darba uzdevumi	6
4.1.3.1.8	Iterācijas laikā ir pieejams kāds no klienta pārstāvjiem	8
4.1.3.1.9	Komandai ir skaidra “Izdarīts” definīcija – definīcija nosaka kādiem atribūtiem ir jāatbilst US, lai to varētu uzskatīt par izdarītu	6
4.1.3.1.10	Komandai ir skaidrs, kādam nolūkam iterācija eksistē	8
4.1.3.1.11	Komandai ir skaidrs, kas tai jādara iterācijas laikā	8
4.1.3.1.12	PO ir skaidrs, kādam nolūkam iterācija eksistē	8
4.1.3.1.13	PO ir skaidri viņa pienākumi iterācijas laikā	8
4.1.3.1.14	*ScrumMaster* ir skaidrs, kādam nolūkam iterācija eksistē	8
4.1.3.1.15	*ScrumMaster* ir skaidri viņa piemākumi iterācijas laikā	8
4.1.3.1.16	Izstrāde ir ilgtspējīga	9
4.1.3.1.17	Izstrādes temps tiek uzturēts konstants	8
4.1.3.2	Iterācijas ilguma atribūti (nedēļas)	5
4.1.3.2.1	X = 1	3
4.1.3.2.2	X = 2	5
4.1.3.2.3	X = 3	5
4.1.3.2.4	X = 4	5

Kods	Nosaukums	SII
4.1.3.2.5	X > 4	3
4.1.4	Dienas sapulce	7
4.1.4.1	Dienas sapulces vispārīgie atribūti	7
4.1.4.1.1	Sapulce eksistē	7
4.1.4.1.2	Sapulcei ir noteikts konkrēts laiks katru dienu	8
4.1.4.1.3	Sapulcei ir noteikts garums	6
4.1.4.1.4	Sapulces laikā visi dalībnieki atbild uz 3 jautājumiem (Ko darīji vakar? Ko darīsi šodien? Vai ir kādi traucējumi?)	7
4.1.4.1.5	Sapulce netiek izmantota detalizētai problēmu apspriešanai	8
4.1.4.1.6	Dienas sapulcē piedalās visa komanda	6
4.1.4.1.7	Komandai ir skaidrs, kādam nolūkam sapulce eksistē	7
4.1.4.1.8	Komandai ir skaidrs, kas tai jādara sapulcē	7
4.1.4.1.9	*ScrumMaster* ir skaidrs, kādam nolūkam sapulce eksistē	8
4.1.4.1.10	*ScrumMaster* ir skaidri viņa piemākumi sapulces laikā	8
4.1.4.2	Dienas sapulces ilguma atribūti	7
4.1.4.2.1	X < 10	5
4.1.4.2.2	10 < X < 15	8
4.1.4.2.3	X > 15	2
4.1.5	Iterācijas pārskata sapulce	8
4.1.5.1	Iterācijas pārskata vispārīgie atribūti	8
4.1.5.1.1	Sapulce eksistē	8
4.1.5.1.2	Sapulce ir regulāra	8
4.1.5.1.3	Sapulcei ir noteikts garums	7
4.1.5.1.4	Sapulcē ir demonstrējams programmatūras papildinājums	8
4.1.5.1.5	Papildinājumi tiek demonstrēti demo veidā	9
4.1.5.1.6	Iterācijas rezultāts atbilst sprinta plānošanā izvirzītajiem mērķiem	7
4.1.5.1.7	Komanda ir pabeigusi visus iterācijā paņemtos US	7
4.1.5.1.8	Komandai ir skaidrs, kādam nolūkam sapulce eksistē	8
4.1.5.1.9	Komandai ir skaidrs, kas tai jādara sapulcē	8
4.1.5.1.10	PO ir skaidrs, kādam nolūkam sapulce eksistē	8
4.1.5.1.11	PO ir skaidri viņa pienākumi sapulces laikā	8
4.1.5.1.12	*ScrumMaster* ir skaidrs, kādam nolūkam sapulce eksistē	8
4.1.5.1.13	*ScrumMaster* ir skaidri viņa piemākumi sapulces laikā	8
4.1.5.2	Iterācijas pārskata ilguma atribūti	6
4.1.5.2.1	X < 1	3
4.1.5.2.2	1 < X < 2	4
4.1.5.2.3	2 < X < 5	4
4.1.5.2.4	5 < X < 8	2
4.1.5.2.5	X > 8	2
4.1.6	Retrospektīva	7
4.1.6.1	Iterācijas retrospektīvas vispārīgie atribūti	6
4.1.6.1.1	Sapulce eksistē	7
4.1.6.1.2	Sapulce ir regulāra	6
4.1.6.1.3	Sapulcei ir noteikts garums	6
4.1.6.1.4	Sapulcē piedalās visa komanda	7
4.1.6.1.5	Sapulcē iesaistās visa komanda	7
4.1.6.1.6	Sapulcei ir vadītājs	8
4.1.6.1.7	Komanda izmanto retrospektīvu, lai mācītos	8
4.1.6.1.8	Komandai ir skaidrs, kādam nolūkam sapulce eksistē	8
4.1.6.1.9	Komandai ir skaidrs, kas tai jādara sapulcē	8
4.1.6.1.10	*ScrumMaster* ir skaidrs, kādam nolūkam sapulce eksistē	9

Kods	Nosaukums	SII
4.1.6.1.11	*ScrumMaster* ir skaidri viņa piemākumi sapulces laikā	9
4.1.6.2	Iterācijas retrospektīvas ilguma atribūti	4
4.1.6.2.1	X < 1	3
4.1.6.2.2	1 < X < 2	7
4.1.6.2.3	2 < X < 5	4
4.1.6.2.4	5 < X < 8	3
4.1.6.2.5	X > 8	2
4.1.7	Laidienu plānošanas sapulce	7
4.1.7.1	Laidienu plānošanas vispārīgie atribūti	7
4.1.7.1.1	Laidienu plānošana eksistē	7
4.1.7.1.2	Komandai ir zināms laidienu plāns	8
4.1.7.1.3	Komanda pieturas pie laidienu plāna	8
4.1.7.1.4	Komanda spēj realizēt laidienu plānu	7
4.1.7.1.5	Pastāv iespēja modificēt laidiena plānu	5
4.1.7.1.6	Komanda saprot kādam nolūkam eksistē laidiena plāns	7
4.1.7.1.7	Komandai ir skaidrs, kādam nolūkam sapulce eksistē	7
4.1.7.1.8	Komandai ir skaidrs, kas tai jādara sapulcē	8
4.1.7.1.9	PO ir skaidrs, kādam nolūkam sapulce eksistē	8
4.1.7.1.10	PO ir skaidri viņa pienākumi sapulces laikā	8
4.1.7.1.11	*ScrumMaster* ir skaidrs, kādam nolūkam sapulce eksistē	8
4.1.7.1.12	*ScrumMaster* ir skaidri viņa piemākumi sapulces laikā	8
4.1.7.2	Laidiena plānošanas ilguma atribūti	5
4.1.7.2.1	X < 1	3
4.1.7.2.2	1 < X < 2	5
4.1.7.2.3	X > 2	5
4.2	Lomas	8
4.2.1	*ScrumMaster* lomas atribūti	9
4.2.1.1	Loma eksistē	9
4.2.1.2	*ScrumMaster* iesaistās “Sākotnējās plānošanas” sapulcē	7
4.2.1.3	*ScrumMaster* iesaistās “Sprinta plānošanas” sapulcē	7
4.2.1.4	*ScrumMaster* iesaistās “Dienas” sapulcē	7
4.2.1.5	*ScrumMaster* iesaistās “Iterācijas pārskata” sapulcē	7
4.2.1.6	*ScrumMaster* iesaistās “Iterācijas retrospektīvas” sapulcē	7
4.2.1.7	*ScrumMaster* iesaistās “Laidiena plānošanas” sapulcē	7
4.2.1.8	*ScrumMaster* iterācijas laikā sargā komandu no ārējās ietekmes	8
4.2.1.9	Komanda ir informēta par *ScrumMaster* pienākumiem	7
4.2.1.10	*ScrumMaster* ir skaidri viņa pienākumi	8
4.2.1.11	*ScrumMaster* loma tiek periodiski vērtēta	8
4.2.2	PO lomas atribūti	8
4.2.2.1	Loma eksistē	8
4.2.2.2	Komandai ir skaidri PO pienākumi	7
4.2.2.3	PO ir skaidri lomas pienākumi	7
4.2.2.4	PO piedalās “Sākotnējās plānošanas” sapulcē	6
4.2.2.5	PO piedalās “Iterācijas plānošanas” sapulcē	8
4.2.2.6	PO piedalās “Iterācijas pārskata” sapulcē	9
4.2.2.7	PO piedalās “Laidienu plānošanas” sapulcē	9
4.2.2.8	PO loma tiek periodiski vērtēta	7
4.2.2.9	PO ir spējīgs skaidri paskaidrot US būtību	9
4.2.3	Komandas lomas atribūti	9
4.2.3.1	Komandai ir zināmi procesa domēna procesi	8
4.2.3.2	Visa komanda piedalās “Sākotnējās plānošanas” sapulcē	7

Kods	Nosaukums	SII
4.2.3.3	Visa komanda piedalās “Iterācijas plānošanas” sapulcē	9
4.2.3.4	Visa komanda piedalās “Iterācijā”	9
4.2.3.5	Visa komanda piedalās “Iterācijas pārskata” sapulcē	7
4.2.3.6	Visa komanda piedalās “Iterācijas retrospektīvas” sapulcē	7
4.2.3.7	Visa komanda piedalās “Dienas” sapulcē	8
4.2.3.8	Visa komanda piedalās “Laidiena plānošanas” sapulcē	6
4.2.3.9	Komandas darbs tiek periodiski vērtēts	9
4.3	Artefakti	7
4.3.1	PB artefakta atribūti	8
4.3.1.1	PB eksistē	9
4.3.1.2	Komandai ir skaidrs kādam nolūkam kalpo PB	7
4.3.1.3	PO ir skaidrs kādam nolūkam kalpo PB	8
4.3.1.4	*ScrumMaster* ir skaidrs kādam nolūkam kalpo PB	7
4.3.1.5	PB tiek uzturēts aktuāls	9
4.3.1.6	Visiem ir skaidrs, kurš ir atbildīgs par PB uzturēšanu	6
4.3.1.7	PB ir viegli pieejams	8
4.3.1.8	PB kvalitāte tiek periodiski vērtēta	7
4.3.2	SB artefakta atribūti	9
4.3.2.1	SB eksistē	8
4.3.2.2	Komandai ir skaidrs kādam nolūkam kalpo SB	8
4.3.2.3	PO ir skaidrs kādam nolūkam kalpo SB	8
4.3.2.4	*ScrumMaster* ir skaidrs kādam nolūkam kalpo SB	8
4.3.2.5	SB tiek uzturēts aktuāls	9
4.3.2.6	Visiem ir skaidrs, kurš ir atbildīgs par SB uzturēšanu	7
4.3.2.7	SB ir viegli pieejams	8
4.3.2.8	SB kvalitāte tiek periodiski vērtēta	7
4.3.3	BD artefakta atribūti	6
4.3.3.1	BD eksistē	6
4.3.3.2	Komandai ir skaidrs kādam nolūkam kalpo BD	5
4.3.3.3	PO ir skaidrs kādam nolūkam kalpo BD	6
4.3.3.4	*ScrumMaster* ir skaidrs kādam nolūkam kalpo BD	7
4.3.3.5	BD ir viegli pieejams	6
4.3.3.6	BD atspoguļo patieso situāciju	7
4.3.4	RP artefakta atribūti	6
4.3.4.1	RP eksistē	6
4.3.4.2	Komandai ir skaidrs kādam nolūkam kalpo RP	5
4.3.4.3	PO ir skaidrs kādam nolūkam kalpo RP	5
4.3.4.4	*ScrumMaster* ir skaidrs kādam nolūkam kalpo SB	7
4.3.4.5	RP tiek uzturēts aktuāls	7
4.3.4.6	Visiem ir skaidrs, kurš ir atbildīgs par RP uzturēšanu	7
4.3.4.7	RP ir viegli pieejams	6
4.3.5	RBD artefakta atribūti	5
4.3.5.1	RBD eksistē	5
4.3.5.2	Komandai ir skaidrs kādam nolūkam kalpo RBD	5
4.3.5.3	PO ir skaidrs kādam nolūkam kalpo RBD	6
4.3.5.4	*ScrumMaster* ir skaidrs kādam nolūkam kalpo RBD	6
4.3.5.5	RBD ir viegli pieejams	7
4.3.5.6	RBD atspoguļo patieso situāciju	6
5	Vērtību domēns	7
5.1	Portfeļa pārvaldība	6
5.1.1	Programmatūras portfeļa pārvaldības atribūti	6

Kods	Nosaukums	SII
5.1.1.1	Programmatūras portfeļa pārvaldība uzņēmumā eksistē	5
5.1.1.2	Portfeļa uzturēšanai tiek izmantots kāds rīks vai datubāze	4
5.1.1.3	Ir definētas programmatūras portfeļa atbildīgais	6
5.1.1.4	Organizācijai un darbiniekiem ir skaidra programmatūras portfeļa nozīme	5
5.1.1.5	Darbiniekiem ir atbilstoša līmeņa pieeja programmatūras portfeļa rīkam	4
5.1.2	Infrastruktūras portfeļa pārvaldības atribūti	6
5.1.2.1	Infrastruktūras portfeļa pārvaldība uzņēmumā eksistē	5
5.1.2.2	Portfeļa uzturēšanai tiek izmantots kāds rīks vai datubāze	4
5.1.2.3	Ir definētas infrastruktūras portfeļa atbildīgais	5
5.1.2.4	Organizācijai un darbiniekiem ir skaidra infrastruktūras portfeļa nozīme	4
5.1.2.5	Darbiniekiem ir atbilstoša līmeņa pieeja infrastruktūras portfeļa rīkam	4
5.1.3	Projektu portfeļa atribūti	7
5.1.3.1	Projektu portfeļa pārvaldība uzņēmumā eksistē	7
5.1.3.2	Portfeļa uzturēšanai tiek izmantots kāds rīks vai datubāze	6
5.1.3.3	Ir definētu projektu portfeļa atbildīgais	6
5.1.3.4	Organizācijai un darbiniekiem ir skaidra projektu portfeļa nozīme	5
5.1.3.5	Darbiniekiem ir atbilstoša līmeņa pieeja projektu portfeļa rīkam	5
5.2	Produkta pārvaldība	5
5.2.1	Produktu pārvaldība uzņēmumā eksistē	6
5.2.2	Produktu pārvaldības informācijas uzturēšanai tiek izmantots kāds rīks vai datubāze	4
5.2.3	Ir definēts produktu pārvaldības atbildīgais	6
5.2.4	Organizācijai un darbiniekiem ir skaidra produktu pārvaldības nozīme	5
5.2.5	Darbiniekiem ir atbilstoša līmeņa pieeja produktu pārvaldības rīkam	5
5.3	Laidienu pārvaldība	7
5.3.1	Laidienu pārvaldība uzņēmumā eksistē	7
5.3.2	Laidienu informācijas uzturēšanai tiek izmantots kāds rīks vai datubāze	5
5.3.3	Ir definēts laidiena pārvaldības atbildīgais	7
5.3.4	Organizācijai un darbiniekiem ir skaidra laidienu pārvaldības nozīme	6
5.3.5	Darbiniekiem ir atbilstoša līmeņa pieeja laidienu plānošanas rīkam	5
5.4	Metriku izmaiņu pārvaldība	7
5.4.1	Procesa pieejamības atribūti	6
5.4.1.1	Procesi uzņēmumā eksistē	7
5.4.1.2	Procesa dokumentācija ir pieejama visiem	5
5.4.1.3	Procesi ir skaidri un dokumentēti visiem saprotamā veidā	5
5.4.2	Atbildīgā atribūti	6
5.4.2.1	Ir noteikts atbildīgais, kurš rūpējas par procesa dokumentāciju	4
5.4.2.2	Visiem ir zināms, kurš ir atbildīgais	7
5.4.2.3	Atbildīgajam ir visas tiesības veikt izmaiņas	5
5.4.3	Vērtēšanas atribūti	7
5.4.3.1	Ir definēti metriku vērtēšanas kritēriji	5
5.4.3.2	Ir definēta metriku vērtēšanas procedūra	4
5.4.3.3	Informācija par praksēm un to darbības rezultātiem tiek saglabāta, lai būtu pieejama nākošā vērtēšanas iterācijā	4
5.4.3.4	Metrikas tiek periodiski vērtētas	8
5.4.3.5	Metriku vērtēšanas rezultāti ir pieejami visiem	6
5.4.4	Izmaiņu ieviešanas atribūti	5
5.4.4.1	Ir definētas izmaiņu ieviešanas procedūras	5
5.4.4.2	Informācija par veiktajām izmaiņām tiek piereģistrēta	4

Kods	Nosaukums	SII
5.4.4.3	Ir definēts atbildīgais par konkrēto izmaiņu ieviešanu	7
5.4.5	Metriku kopas izmērs	4
5.4.5.1	X < 3	4
5.4.5.2	3 < X < 6	5
5.4.5.3	6 < X < 9	4
5.4.5.4	9 < X < 12	5
5.4.5.5	X > 12	4
5.4.6	Metrikas lietošanas ilgums (mēneši)	6
5.4.6.1	X < 1	3
5.4.6.2	1 < X < 6	4
5.4.6.3	6 < X < 12	5
5.4.6.4	12 < X < 24	5
5.4.6.5	X > 24	6
5.5	Vērtības piegāde	8
5.5.1	Piegādes kopējie atribūti	8
5.5.1.1	Piegāde ir vienkārša	8
5.5.1.2	Piegāde ir automatizēta	8
5.5.1.3	Pastāv iespēja atgriezt atpakaļ iepriekšējo versiju	7
5.5.1.4	Atbildīgais par piegāde ir definēts	8
5.5.1.5	Visi zina, kurš ir atbildīgais	8
5.5.1.6	Strādājošs un testēts papildinājums tiek piegādāts	8
5.5.2	Piegādes intervāls nedēļās	6
5.5.2.1	X < 1	3
5.5.2.2	X = 1	4
5.5.2.3	X = 2	5
5.5.2.4	2 < X < 4	7
5.5.2.5	4 < X < 8	3
5.5.2.6	X > 8	2
6	Projekts	7
6.1	Projekta veids	7
6.1.1	Ūdenskrituma	0
6.1.2	Iteratīva un papildinoša	8
6.1.3	*Scrum* bāzēts	8
6.2	Iesaistīto personu skaits	7
6.2.1	X < 10	7
6.2.2	10 < X < 20	3
6.2.3	20 < X < 40	3
6.2.4	X > 40	2
6.3	Pieredze ar projektu % no kopējā projekta laika	8
6.3.1	X < 20	2
6.3.2	20 < X < 40	3
6.3.3	40 < X < 60	5
6.3.4	60 < X < 80	6
6.3.5	80 < X < 100	7
6.4	Izmērs	7
6.4.1	Papildinājums	8
6.4.2	Mazs	7
6.4.3	Vidējs	5
6.4.4	Liels	3
6.4.5	Ļoti liels	3
6.5	Garums	6

Kods	Nosaukums	SII
6.5.1	Īss	8
6.5.2	Vidējs	6
6.5.3	Garš	3
6.6	Sarežģītība	7
6.6.1	Zems	7
6.6.2	Vidējs	5
6.6.3	Augsts	4
6.7	Patiesais garums	6
6.7.1	X < 6	6
6.7.2	6 < X < 12	7
6.7.3	12 < X < 24	6
6.7.4	24 < X < 36	5
6.7.5	36 < X < 60	4
6.7.6	X > 60	3

3.6. Secinājumi

Organizācijas spējības definēšana izmantojot DAA sniedz iespēju izvērtēt atsevišķas organizācijas sastāvdaļas un ērtākā veidā veikt iegūto datu analīzi. Ekspertu grupas noteiktās SII vērtības sniedz iespēju precīzāk noteikt iespējamo problēmas iemeslu.

Darbā izveidotā DAA struktūra tika vairākkārt pielāgota un modificēta, tai skaitā modifikācijas ir veiktas, sākot ekspertu anketēšanu. Notikušās modifikācijas gan neizslēdz iespēju, ka DAA struktūra var tikt modificēta atkārtoti. Piemēram, definētā DAA struktūra ir paredzēta organizācijām, kuras izmanto *Scrum* metodi izstrādes procesa organizēšanai. Ņemot vērā, ka *Scrum* nav vienīgā spējā metode, tad organizācijām, kuras izmanto citu pieeju ir nepieciešams veikt izmaiņas Procesu domēnā. Līdzīgā veidā var tikt ieviestas izmaiņas arī citos DAA elementos. Gadījumos, ja ir veiktas izmaiņas DAA struktūrā, ir nepieciešams veikt atkārtotu SII noteikšanu mainītajiem elementiem, kas savukārt nozīmē, ka ir nepieciešams atkārtoti izveidot ekspertu grupu un veikt anketēšanu.

DAA struktūras izveidošana un ekspertu anketēšana izvērsās ievērojami sarežģītāka un darbietilpīgāka, nekā sākumā plānots un nostabilizēt DAA struktūru izdevās 5. piegājienā.

Promocijas darba ietvaros izstrādātā spējības noteikšanas metode ODSN (Organizācijas domēnu spējības noteikšana) izmanto DAA struktūru un SII vērtības organizācijas spējības noteikšanai.

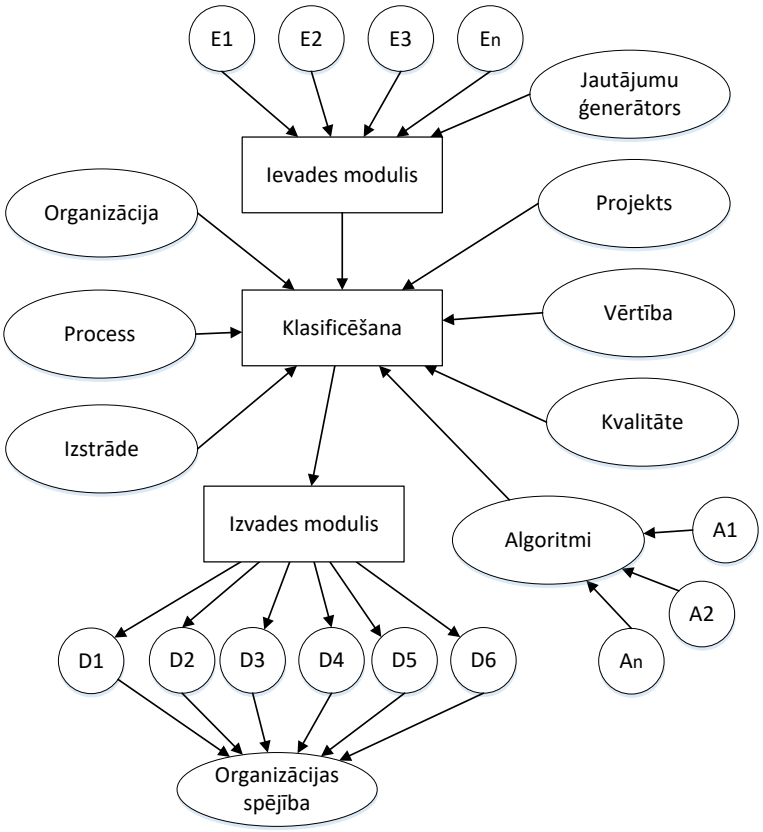
4. Spējības noteikšana

Spējības noteikšanai ir nepieciešama sistemātiska pieeja. Promocijas darbā uzstādīto mērķi var sasniegt, izmantojot iteratīvu pieeju spējības noteikšanai.

4.1. Spējības noteikšanas konceptuālais modelis

Spējības noteikšanai darbā ir izstrādāta metode ODSN (Organizācijas domēnu spējības noteikšana). Konceptuālā moduļa galvenās sastāvdaļas ir (sk. 4.1. att.):

- ievades modulis – darbinieki E1…E_n atbild uz jautājumiem, par domēniem, sniedzot nepieciešamo informāciju Klasificēšanas modulim. Ievades modulis izmanto Jautājumu ģeneratoru, lai iegūtu jautājumu sarakstu, uz kuriem atbildēs darbinieki;
- klasificēšanas komponente – izmanto domēnu informāciju un klasificē domēnus atbilstoši spējības līmenim, izmantojot vienu vai vairākus klasificēšanas algoritmus A_1…A_n;
- izvades modulis – ataino domēnu klasificēšanas rezultātus dažādos griezumos un sniedz lietotājiem informāciju par domēniem D1, D2, D3, D4, D5, D6.



4.1. att. ODSN konceptuālais modelis.

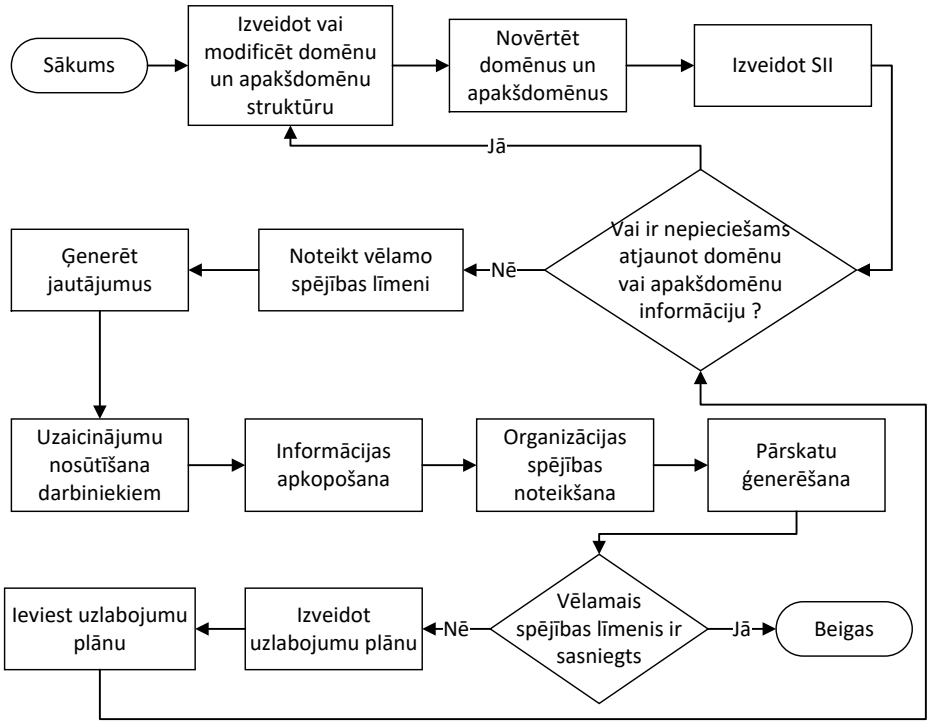
Izvades modulis sniedz detalizētu informāciju par katra domēna spējības līmeni. Detalizētā informācija sastāv no informācijas par domēniem, apakšdomēniem un to atribūtiem, kas sniedz lietotājiem priekšstatu par spējības problēmām. Metode ODSN sastāv no pieciem lietošanas stāstiem.

4.2. Metode ODSN

ODSN metode ir paredzēta regulārai organizācijas, projekta vai komandas spējības noteikšanai. Vēlamā spējības līmeņa sasniegšanai ir nepieciešams nodrošināt regulāru metodes darbību. Metode sniedz iespēju identificēt problēmas apgabalus, tādā veidā palīdzot veikt uzlabojumus organizācijas darbībā. Metodi var izmantot organizācijas, kuras vēlas veikt programmizstrādi spējā veidā. Promocijas darbā izstrādātā ODSN metode paredzēta *Scrum* metodes ieviešanai, bet to iespējams adaptēt arī citām spējām metodēm.

4.2.1. Process

ODSN metodes process sastāv no vairākiem apakšprocesiem, būtiskākie no kuriem ir OSM elementu novērtēšana un jautājumu ģenerēšana (sk. 4.2. att.).



4.2. att. ODSN metodes process.

Metode sastāv no 11 pamatprocesiem:

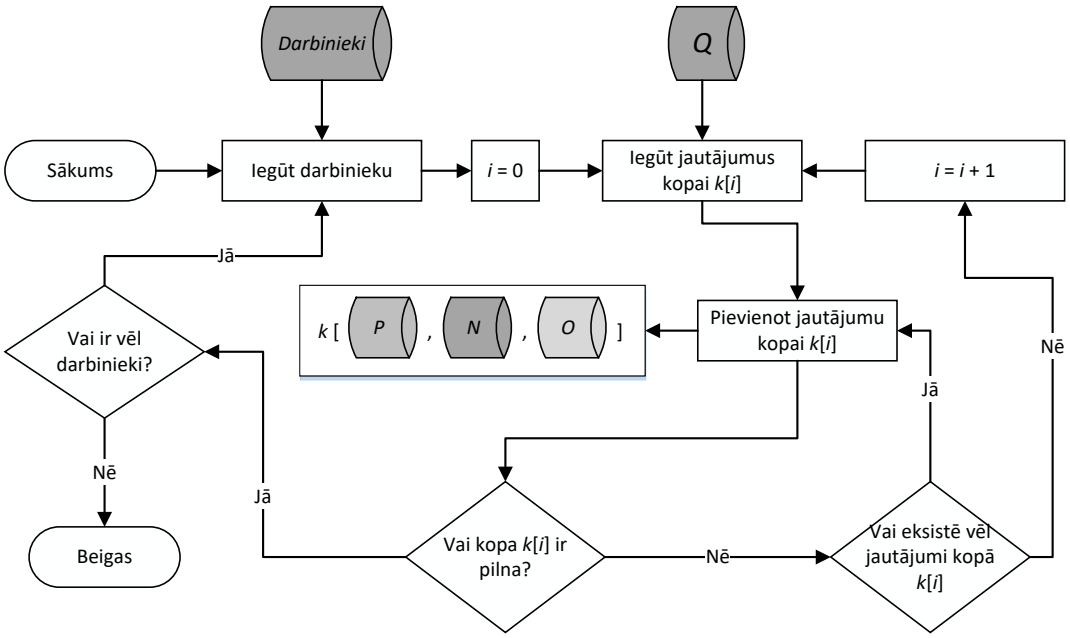
- izveidot vai modificēt DAA struktūru - Sākotnējā DAA struktūra ir jau izveidota. Šis solis ir nepieciešams gadījumos, ja nepieciešamas DAA struktūras izmaiņas;
- novērtēt domēnus un apakšdomēnus – DAA struktūras elementi tiek novērtēti, izmantojot ekspertu novērtēšanas metodi, piemēram, DELPHI;
- izveidot SII – ekspertu grupas anketēšanas rezultātā (parasti vairākās iterācijās) tiek iegūtas DAA elementu SII vērtības, kuras tiek izmantotas spējības līmeņa noteikšanai;
- vēlamā spējības līmeņa noteikšana – organizācija vai komanda definē vēlamo spējības līmeni;
- jautājumu ģenerēšana – SII vērtības tiek izmantotas, lai ģenerētu konkrētam darbiniekam paredzētu jautājumu kopu;
- uzaicinājumu nosūtīšana darbiniekiem – atbilstošajiem darbiniekiem tiek nosūtīts uzaicinājums ar saiti uz darbiniekam ģenerētajiem jautājumiem;
- informācijas apkopošana – informācija no internetā aizpildītājām anketām tiek saglabāta un apkopota spējības noteikšanai;
- organizācijas spējības noteikšana – izmantojot iegūto informāciju un definētās SII vērtības, tiek noteikts organizācijas, projekta vai komandas spējības līmenis;
- pārskatu ģenerēšana – pamatojoties uz noteikto spējības līmeni un vēlamo spējības līmeni, tiek veidoti dažāda līmeņa pārskati, lai sniegtu lietotājam iespēju analizēt spējības līmeni un tā izmaiņu dinamiku;
- izveidot uzlabojumu plānu – gadījumā, ja vēlamais līmenis nav sasniegts, organizācija vai komanda, balstoties uz pārskatiem, var veidot uzlabojumu plānu spējības līmeņa uzlabošanai;
- uzlabojumu plāna ieviešana – balstoties uz izveidoto uzlabojumu plānu, tiek veiktas darbības spējības līmeņa uzlabošanai.

Spējības uzlabošanas process tiek iteratīvi atkārtots. Atkārtošanas biežums ir atkarīgs no organizācijas un komandas, piemēram, dažas komandas var veikt novērtēšanu katras iterācijas beigās, bet citas komandas reizi divās iterācijās. Jautājumu skaitu katrā intervēšanas reizē nosaka organizācija. Organizācijai un komandai ir jāatrod viduscelš starp daudziem jautājumiem katru reizi, lai varētu ātri iegūt pilnīgu informāciju, vai nelielas jautājumu kopas izmantošanu un ilgāku laiku kopskata iegūšanai. Ir jāņem vērā, ka liela jautājumu kopa parasti izraisa nepatiku darbiniekos

un atbilžu kvalitāte krītas. Ieteicamais jautājumu kopas izmērs ir 10 jautājumi, un anketēšanai nevajadzētu ilgt vairāk par 5 līdz 10 minūtēm [102].

4.2.2. Jautājumu kopas ģenerators

Jautājumu ģenerēšanas komponente ir svarīga metodes ODSN sastāvdaļa un ir nepieciešama, lai ģenerētu tikai nelielu jautājumu kopu katram darbiniekam katrā intervēšanas iterācijā. Kopumā eksistē vairāk nekā 300 jautājumi un nav lietderīgi uzdod katram darbiniekam vienā reizē visus jautājumus. Jautājumu ģenerators ņem vērā šo faktu un ģenerē jautājumus, balstoties uz SII vērtību (4.3. att.).



4.3. att. Jautājumu kopas ģenerēšanas process.

Jautājumu ģenerators ģenerē jautājumu apakškopas katram darbiniekam, izmantojot kopējo jautājumu apakškopu, kura izteikta kā

$$Q = \{q_1, q_2, q_3, \dots, q_m\}, \tag{3}$$

kur Q – visu jautājumu kopa;

$q_{1,\dots,}\, q_m$ – jautājumi, kur m ir kopējais jautājumu skaits.

Darbinieku jautājumu kopu var definēt kā visu jautājumu apakškopu

$$A_{1\dots n} \in Q, \tag{4}$$

kur Q – visu jautājumu kopa;

$A_{1...n}$ – darbinieka jautājumu apakškopa, kur n ir kopējais darbinieku skaits, kuri piedalās anketēšanā.

Katram darbiniekam uzģenerētā jautājumu kopa sastāv no trīs veidu jautājumiem

$$A_{1...n} = \{P, N_n, O_n\}, \quad (5)$$

kur $A_{1...n}$ – darbinieka jautājumu apakškopa, kur n ir kopējais darbinieku skaits, kuri piedalās anketēšanā;

P – prioritāro jautājumu kopa – intervēšanas iniciators pirms intervēšanas atzīmē ļoti svarīgus jautājumus, uz kuriem vēlas iegūt atbildi pēc iespējas ātrāk. Prioritārie jautājumi tiek pievienoti visām darbinieku jautājumu kopām un veido 20 % no uzģenerētās jautājumu kopas;

N_n – neatbildētie jautājumi, kas sakārtoti pēc SII, kur n ir konkrētais darbinieks. Pēc prioritāro jautājumu pievienošanas kopai, tai tiek pievienoti jautājumi ar augstu SII vērtību, uz kuriem darbinieks vēl nav atbildējis. Šie jautājumi procentuāli aizņem visvairāk un veido 60 % no jautājumu kopas;

O_n – iepriekš atbildētie jautājumi, kas sakārtoti pēc SII, kur n ir konkrētais darbinieks. Eksistē būtiski jautājumi, uz kuriem ir nepieciešams atbildēt biežāk nekā citiem, tādēļ jautājumu kopai visu laiku tiek pievienoti būtiski jautājumi, uz kuriem jau ir atbildēts iepriekš. Šie jautājumi veido atlikušos 20 % no jautājumu kopas.

Piemēram, ja mūsu jautājumu kopa sastāv no 17 jautājumiem

$$Q = \{q_1 = 9, q_2 = 8, q_3 = 7, q_4 = 6, q_5 = 9, q_6 = 8, q_7 = 7, q_8 = 7, q_9 = 8, q_{10} = 9, q_{11} = 9, q_{12} = 8, q_{13} = 7, q_{14} = 7, q_{15} = 6, q_{16} = 7, q_{17} = 8\}, \quad (6)$$

un mums ir 4 jautājumi prioritāro jautājumu kopā P_q,

$$P_q = \{q_{10} = 9, q_{17} = 8, q_3 = 7, q_{15} = 6\}, \quad (7)$$

un 4 jautājumi iepriekš atbildēto jautājumu kopā O,

$$O = \{q_{12} = 8, q_7 = 7, q_{16} = 7, q_4 = 6\}, \quad (8)$$

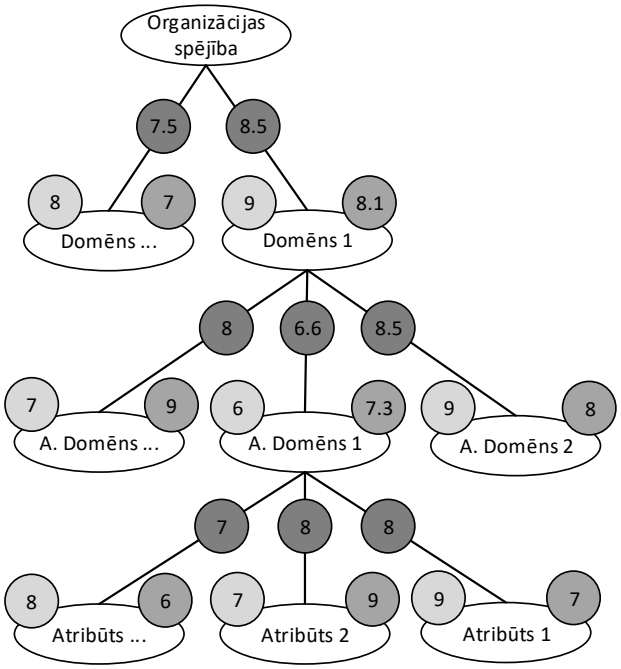
tad pie nosacījuma, ka mums ir viens darbinieks un jautājumu kopā ir 10 jautājumi iegūsim jautājumu kopu A_1.

$$A_1 = \{P\{q_{10}, q_{17}\}, N\{q_1, q_5, q_{11}, q_2, q_6, q_9\}, O\{q_{12}, q_7\}\} \quad (9)$$

Pēc jautājumu kopu ģenerēšanas, kopas tiek nosūtītas atbilstošajiem darbiniekiem anketu aizpildīšanai. Anketēšanas rezultātā tiek izveidots DAA vērtību koks.

4.2.3. DAA vērtību koks

DAA vērtību koks ir ērts veids, kā attēlot iegūto informāciju un viegli atrast problēmu apgabalu, jo ekspertu vērtētās SII vērtības tiek attēlotas blakus anketā iegūtajai un apkopotajai informācijai (sk. 4.4. att.).



4.4. att. DAA vērtību koks.

DAA vērtību kokā katram elementam tiek pievienotas divas vērtības. Kreisajā pusē ir ekspertu noteiktās SII vērtības, bet labajā atrodas anketēšanas rezultātā iegūtās darbinieku vērtēšanas vērtības (DVV). Katra atribūta DVV vērtība DAA vērtību kokā tiek aprēķināta kā vidējā vērtība par atribūtu un ir izteikta ar vienādojumu

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}, \tag{10}$$

kur $\overline{x}$ – atribūta DVV vērtība;

x_i – atribūta vērtējums, ko sniedzis darbinieks i;

n – darbinieku skaits, kuri snieguši atbildi par konkrēto atribūtu.

Katra apakšdomēna DVV vērtība tiek aprēķināta, izmantojot vidējo svērto vērtību no iekļautajām un aprēķinātajām atribūtu DVV vērtībām. Katra domēna DVV vērtība tiek aprēķināta, izmantojot vidējo svērto vērtību no iekļautajiem apakšdomēniem un ir iztaikta ar vienādojumu

$$\overline{y} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}, \tag{11}$$

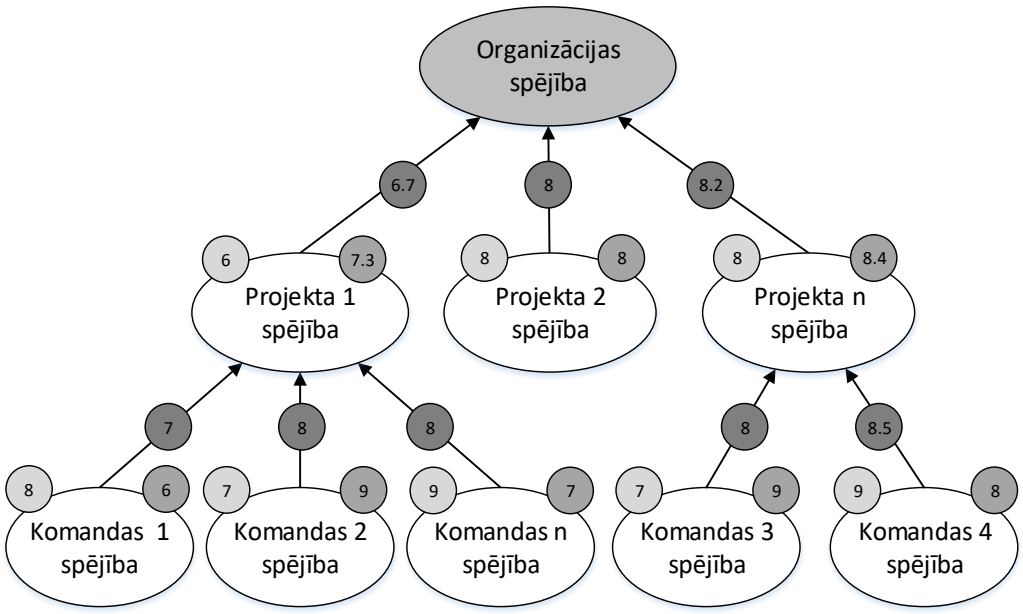
kur $\overline{y}$ – domēna, apakšdomēna DVV vērtība;

x_i – atribūta vai apakšdomēna DVV vērtība;

w_i – atribūta vai apakšdomēna SII vērtība (svars).

4.2.4. Vairāku komandu un projektu spējība

Reti ir gadījumi, kad ir tikai viena komanda un viens projekts, it īpaši, ja runājam par vidēja un liela izmēra organizācijām. Ja organizācija strādā pie vairākiem projektiem un kādā no projektiem ir vairākas komandas, tad ir nepieciešams izmantot līmeņus. Spējību ir iespējams noteikt dažādos līmeņos: organizācijas, projekta un komandas līmenī (sk. 4.5. att.).



4.5. att. Spējības grupēšanas līmeņi.

Vairāku projektu un vairāku komandu gadījumā ir nepieciešami papildu soļi spējības noteikšanai. Viens no soļiem ir komandas ietekmes indeksa (KII) noteikšana, un otrs solis ir projekta ietekmes indeksa (PII) noteikšana.

4.2.4.1. Komandas ietekmes indekss

Komandas ietekmes indekss (KII) ir veids, kā izteikt, cik svarīga ir vērtējamā komanda konkrētā projekta kontekstā. Piemēram, komanda A tiek vairākus mēnešus nodarbināta projektā P, un ir izstrādājusi arhitektūru un sagatavojusi pamata ietvaru. Kādā brīdī tiek pieslēgta komanda B, kurā ir vairāk nepieredzējušu studentu un kuras uzdevums ir risināt vienkāršākas problēmas. Šajā gadījumā komandas B ietekmes indekss uz kopējo projektu būs mazāks nekā komandas A ietekmes indekss uz projektu P. KII tiek izteikts vērtībās no 0 līdz 10, kur 0 komandai nav ietekmes uz projektu, un 10 komandai ir būtiska ietekme uz projektu.

Veids, kā noteikt komandas KII vērtību tiek atstāts projekta vadības ziņā. Kā ērtāko varētu minēt iespēju projekta vadībai un komandu vadītājiem veikt nelielu ekspertu aptauju. Aptaujā kā eksperti figurē komandas vadītājs un *ScrumMaster*. Definētā KII vērtība tiek fiksēta uz noteiktu laiku, kamēr nerodas nepieciešamība veikt tās pārvērtēšanu.

4.2.4.2. Projekta ietekmes indekss

Projekta ietekmes indekss (PII) ir veids, kā izteikt, cik svarīga ir vērtējamā projekta ietekme organizācijas kontekstā. Piemēram, projektā A tiek nodarbināta lielākā daļa organizācijas izstrādātāju, un projekts A ienes organizācijai lielāko peļņas daļu, bet projekts B ir neliels iekšēja rakstura projekts, kurā iesaistīta neliela komanda. Abu šo projektu ietekme uz organizācijas spējību ir atšķirīga, tādēļ projektu A un B PII vērtība būs atšķirīga. PII tiek izteikts vērtībās no 0 līdz 10, kur 0 nozīmē, ka projektam nav ietekmes uz organizāciju, bet 10 ka projektam ir būtiska ietekme uz organizāciju.

Veids, kā noteikt projekta PII vērtību tiek atstāts organizācijas ziņā. Kā vienu no veidiem var izmantot iesaistīto cilvēku skaitu vai projekta izmaksas. PII var tikt noteikta līdzīgā veidā kā KII, izmantojot ekspertu grupu. Ekspertu grupā var piedalīties atbilstošo projektu *ScrumMaster* un organizācijas vadība. PII vērtības tiek definētas uz noteiktu laiku un var tikt mainītas, ja rodas šāda nepieciešamība. Nevar apgalvot, ka projekta A PII vērtība būs lielāka par projekta B PII vērtību visu laiku.

4.2.5. *FOIL* (angl. *First-Order Inductive Learner*) algoritma izmantošana spējības noteikšanai

FOIL ir pirmā līmeņa likumu iegūšanas algoritms, kuru var izmantot klasifikācijas uzdevumu atrisināšanai [89].

4.2.5.1. Terminoloģija

Algoritmu apraksti dažreiz ir sarežģīti un pārpildīti ar dažādu terminoloģiju, tādēļ šajā apakšnodaļā ir apkopota informācija par algoritma aprakstā izmantoto terminoloģiju. Pirmā līmeņa loģikas (angl. *first role logic*) pamatjēdzieni [89]:

- katra labi veidota izteiksme (angl. *expression*), kas veidota no **konstantēm** (piem., Testētājs, 24 vai Izstrādātājs), **mainīgajiem** (piem. X), **predikātiem** (piem., Izstrādātājs, kā Izstrādātājs(Jānis)), un **funkcijām** (piem., Pieredze, kā Pieredze(Jānis));
- **termins** (term) ir jebkura konstante, mainīgais vai funkcija, kas izmantota terminam (piem., Jānis, X, Pieredze(Jānis), Pieredze(X));
- **litera** (literal) ir jebkurš predikāts, vai tā noliegums (negation), kas pielietota jebkurai terminu kopai. Piemērs: Izstrādātājs(Jānis), ¬ Izstrādātājs(X), Lielāks kā (Pieredze(Jānis), 20);
- **pamatlitera** ir litera, kas nesatur mainīgos (piem., Pieredze(Jānis));
- **negatīva litera** ir litera, kas satur noliedzošu predikātu (piem. ¬ Testētājs(Jānis));
- **pozitīva litera** ir litera bez nolieguma zīmes (piem., Izstrādātājs(Jānis));
- **klauzula** (clause) ir jebkura literu disjunkcija (disjunction) $M_1 \vee .... M_n$, kuras mainīgie ir vispārīgi kvantitatīvi;
- **Horna klauzula** ir izteiksme formā $H \leftarrow (L_1 \wedge ... \wedge L_n)$, kur H, $L_1 ... L_n$ ir pozitīvas literas. H tiek saukts par Horna klauzulas **galvu** (head). Literu $L_1 \wedge ... \wedge L_n$ konjukcija (conjuction) tiek saukta par Horna klauzulas **ķermeni** (body);
- Jebkurām literām A un B, izteiksme $(A \leftarrow B)$ ir vienāda izteiksmei $(A \wedge \neg B)$ un vienāda izteiksmei $\neg (A \wedge B)$, kas vienāda izteiksmei $(\neg A \wedge \neg B)$. Sakarā ar to Horna klauzulas ekvivalenti var tikt rakstīti kā disjunkcija: $H \vee \neg L_1 \vee ... \vee \neg L_n$;
- **substitūcija** (substitution) vai aizstāšana ir jebkura funkcija, kas aizvieto mainīgos ar terminiem. Piemēram, aizvietojums {x/3, y/z} aizvieto mainīgo x ar terminu 3 un mainīgo y ar terminu z. Ja dota substitūcija Θ un litera L, tad mēs rakstam ΘL, lai apzīmētu substitūcijas Θ pielietošanu pret L;
- **apvienojošā substitūcija** (unified substitution) divām literām L_1 un L_2 ir jebkura substitūcija Θ, kas apmierina nosacījumu $L_1\Theta = L_2\Theta$.

4.2.5.2. Vispārīga informācija par *FOIL* algoritmu

Formāli hipotēzēs, kuras iegūtas izmantojot *FOIL*, ir pirmā līmeņa likumi (angl. *first-order rules*), kur katrs likums līdzinās Horna klauzulai ar diviem izņēmumiem [1]:

- likumi, kas iegūti izmantojot *FOIL*, ir vairāk ierobežoti nekā pārējās Horna klauzulas, jo literām nav atļauts saturēt funkcijas, kas savukārt samazina hipotēzes telpas meklēšanas sarežģītību;
- *FOIL* likumi ir izteiksmīgāki nekā Horna klauzulas, jo literas, kuras parādās ķermenī, var būt noliedzošas (angl. *negated*).

```
FOIL(Target_predicate, Predicates, Examples)
• Pos ← those Examples for which the Target_predicate is True
• Neg ← those Examples for which the Target_predicate is False
• Learned_rules ← {}
• while Pos, do
      Learn a NewRule
      • NewRule ← the rule that predicates Targer_predicate with preconditions
      • NewRuleNeg ← Neg
      • while NewRuleNeg, do
            Add a new literal to specialize NewRule
            • Candidate_literals ← generate candidate new literals for NewRule, based on Predicates
            • Best_literal ← argmax Foil_Gain(L, NewRule)
            • add Best_literal to preconditions of NewRule
            • NewRuleNeg ← subset of NewRuleNeg that satisfies NewRule preconditions
      • Learned_rules ← Learned_rules + NewRule
      • Pos ← Pos - {members of Pos covered by NewRule}
• Return Learned_rules
```

4.6. att. *FOIL* algoritms.

Kā redzams attēlā, ārējais cikls apstrādā pa vienam likumam vienlaicīgi, izņemot no apskatāmajiem datiem pozitīvos piemērus, un turpina ar nākamo likumu. *FOIL* gadījumā iekšējais cikls meklē tikai likumus, kuri nosaka, ka mērķa litera ir pozitīva. Kā arī *FOIL* izmanto „hillclimbing" meklēšanu, nevis „beam" meklēšanu (var teikt, tā ir beam meklēšana ar plašumu 1).

Hipotēzes telpas pārmeklēšana, kuru veic *FOIL* algoritms, ir vieglāk saprotama, ja uz to skatās hierarhiski. Katra iterācija pa ārējo ciklu pievieno jaunu likumu disjunktīvai hipotēzei, *Iemācītie_likumi*. Katra nākamā likuma būtība ir vispārināt tekošo disjunktīvo hipotēzi (tādā veidā palielinot pozitīvi klasificēto elementu skaitu), pievienojot jaunu disjunkciju. Šajā gadījumā meklēšana ir specifiska-uz-vispārināto, sākot ar visspecifiskāko tukšo disjunkciju un beidzot, kad hipotēze ir pietiekoši vispārināta, lai iekļautu visus pozitīvos testa piemērus. Iekšējais *FOIL* cikls veic sīkāk strukturētu meklēšanu, lai noteiktu precīzu katra likuma definīciju. Iekšējais cikls meklē otru hipotēzes telpu, kas sastāv no literu konjukcijas, lai atrastu konjukcijas, kas

veidos priekšnosacījumus jaunajam likumam. Šajā hipotēzes telpā, tas veido *vispārināto-uz-specifisko*, „hill-climbing" meklēšanu, kura sākas ar vispārinātāko priekšnosacījumu (tukšs priekšnosacījums), tad pievieno literas pa vienai, tik ilgi, kamēr tai neatbilst neviens negatīvs piemērs.

Divas būtiskas atšķirības starp *FOIL* un citiem „SEQUENTIAL-COVERING" un „LEARN-ONE-RULE" algoritmiem ir sekojošas:

- izmantojot vispārināto-uz-specifisko meklēšanu, lai iemācītos katru nākamo likumu, *FOIL* izmanto vairākus detalizētus soļus, lai ģenerētu likuma kandidātus. Šī starpība izriet no nepieciešamības izmantot **mainīgos** likumu priekšnosacījumos;
- *FOIL* izmanto veiktspējas mērījumu *Foil_Gain*, kas atšķiras no entropijas mērījuma, kas tiek izmantots citos „LEARN-ONE-RULE" algoritmos, kā arī ir jāņem vērā, ka *FOIL* meklē tikai likumus, kas attiecas uz pozitīviem piemēriem.

Ideāls rezultāts priekš *FOIL* ir atrast likumu kopu, kas nosedz visas pozitīvās kombinācijas un nevienu negatīvo.

4.2.5.3. *FOIL* kandidātu ģenerēšana

Lai uzģenerētu kandidātus konkrētam likumam, *FOIL* ģenerē dažādas jaunas literas, katra no kurām var tikt neatkarīgi pievienota priekšnosacījumiem. Piemēram, konkrētajam apskatāmajam likumam ir sekojoša forma [89]

$$P(x_1, x_2, ..., x_k) \leftarrow L_1 \dots L_n\,, \qquad (12)$$

kur $L_1 \dots L_n$ - literas, kuras veido konkrētā likuma priekšnosacījumus;
$P(x_1, x_2, \dots, x_k)$ - litera kas veido likuma galvu.

FOIL ģenerē kandidātus konkrētajam likumam, ņemot vērā literas $L_{n+1,}$ kuras atbilst vienai no konkrētajām formām $Q(v_1 \dots v_r)$, kur Q ir jebkurš predikāta nosaukums, kurš parādās Predikātos (angl. *Predicates*) un kur v_i ir jauni mainīgie, kuri jau ir likumā. Vismaz vienam no mainīgajiem v_i konkrētajā literā, ko veidojam, ir jau jābūt likumā. $Equal(x_j, x_k)$, kur x_j un x_k ir mainīgie, kas jau eksistē likumā.

Ilustrēšanai apskatīsim sekojošu piemēru, mēs gribam iemācīties likumus, kas spēj noteikt mērķa literu *ScrumCoatch(x,y)*, kur pārējie predikāti piemēru aprakstīšanai ir *Coatch* un *Developer*. Vispārināt-uz-specifisko *FOIL* meklēšana sākas ar vispārinātāko likumu:

$$ScrumCoatch(x, y) \leftarrow$$

Likums apgalvo, ka *ScrumCoatch(x, y)* ir patiess visiem *x* un *y*. Lai sašaurinātu esošo likumu procedūra uzģenerē sekojošas literas kā papildinājumu nosacījumiem *Equal(x,y), Developer(x), Developer(y), Coatch(x,y), Coatch(y,x), Coatch(x,z), Coatch(z,x), Coatch(y, z), Coatch(z,y)* un katras šīs literas negācijas (piem. ¬*Equal(x,y)*). Pievērsiet uzmanību, ka *z* šajā kontekstā ir jauns mainīgais, tai pašā laikā x un y jau eksistē tekošajā likumā.

Tagad pieņemam, ka *FOIL* izvēlas *Coatch(y,z)* kā vienu no daudzsološākajām literām, kas mūs noved pie specificējoša likuma.

$$ScrumCoatch(x, y) \leftarrow Coatch(y,z)$$

Ģenerējot pārējās literas, lai nodrošinātu pilnīgāku likumu kopu, *FOIL* tagad ņems vērā visas literas, kas minētas iepriekšējā solī, kā arī papildu literas *Developer(z), Equal(z,x), Equal(z,y), Coatch(z,w), Coatch(w,z)* un to negācijas. Šīs jaunās literas tiek ņemtas vērā šajā posmā, jo tika pievienots jauns mainīgais un *FOIL* tagad ir jāpievieno vēl viens jauns mainīgais *w*.

Ja *FOIL* algoritms šajā brīdī izvēlētos literu *Coatch(z,x)* un nākamajā iterācijā literu *Developer(y)*, tad tas novestu pie sekojoša likuma, kurš nosedz tikai pozitīvos piemērus no testa datiem. Tādēļ pašreizējā likuma ģenerēšana ir jāpārtrauc, un galā iegūsim sekojošu likumu.

$$ScrumCoatch(x, y) \leftarrow Coatch(y,z) \wedge Coatch(z,x) \wedge Developer(y)$$

Kad likums ir gatavs, visi pozitīvie piemēri no testa datiem tiek izņemti. Ja pāri paliek kāds pozitīvs piemērs, vēl viens vispārināts-uz-specifisko likums tiek izveidots, lai pārklātu pozitīvos piemērus.

4.2.5.4. *FOIL* algoritma meklēšanas vadīšana

Lai izvēlētos vislabāko literu, ko katrā solī pievienot likumam *FOIL*, algoritms mēra katras literas efektivitāti pret testa datiem. Procesa ilustrēšanai apskatīsim sekojošu piemēru, kur literas galva ir *ScrumCoatch(x,y)* un mēs meklējam atbilstošās priekšnosacījumu literas. Pieņemsim, ka apmācības datos ir sekojoši pieņēmumi, kur mēs izmantojam pieņēmumu, ka *P(x,y)* lasās kā „P no x ir y".

ScrumCoatch(Victor, Sharon), Coatch(Sharon, Bob), Coatch(Tom, Bob), Developer(Sharon), Coatch(Bob, Victor)

Mums ir jāpieņem, ka katrai no minētajām literām un mainīgajiem vērtība varbūt false, kā rezultātā mēs varam apgalvot, ka:

¬ *ScrumCoatch(Tom, Bob),* ¬ *ScrumCoatch(Victor, Victor) utt.*

Lai izvēlētos labākās literas pašreizējam likumam, *FOIL* uzskata katru atšķirīgo veidu, kurā likuma mainīgie var saistīties ar konstantēm testa piemēros. Piemēram, sākotnējā solī, kad likums ir *ScrumCoatch(x, y)* ← likuma mainīgie x un y nav piesaistīti, ar kādu no priekšnosacījumiem, tādēļ var saistīties ar jebkuru no 4 konstantēm *Victor, Sharon, Bob* un *Tom*, izmantojot pierakstu {x/Bob, y/Sharon}, lai norādītu uz konkrēto mainīgo sasaisti. Ņemot vērā, ka mums ir 4 konstantes, tad pastāv 16 iespējamās mainīgo sasaistes sākotnējam likumam. Sasaiste {x/Victor, y/Sharon} atbilst pozitīvam piemēram, jo testa datos ir pieņēmums *ScrumCoatch(Victor, Sharon)*. Atlikušajām 15 sasaistēm, kuras pieļauj likums (piemēram, {x/Bon, y/Tom}), atrod noliedzošu pierādījumu, jo tādu pieņēmumu testa datos atrast nevar.

Katrā solī likumi tiek novērtēti, balstoties uz pozitīvo un negatīvo mainīgo sasaisti. Priekšroka tiek dota likumiem, kuru rezultātā tiek atrasti vairāki pozitīvi paraugi un mazāks skaits negatīvu paraugu. Pievienojot jaunas literas likumam, mainās arī sasaistes. Jāņem vērā, ja likumam pievieno jaunu literu, kurā ir jauns mainīgais, likums palielināsies garumā (piemēram, ja tiek pievienota litera *Coatch(y, z)*, tad oriģinālajai sasaistei {x/Victor, y/Sharon} būs jāpaliek par {x/Victor, y/Sharon, z/Bob}). Ja pievieno jaunu mainīgo, kas var sasaistīties ar dažādām konstantēm, tad sasaišu skaits var pieaugt.

FOIL novērtēšanas funkcija ir atkarīga no pozitīvo un negatīvo sasaišu skaita pirms un pēc jaunas literas pievienošanas. Pieņemsim mums ir likums *R* un kandidāt litera *L*,

kura varētu tikt pievienota likuma *R* ķermenim. Likums R^1 ir likums, kāds izveidojas pēc literas *L* pievienošanas likumam *R*. *Foil_Gain(L, R)* vērtība, pievienojot *L* pie *R*, tiek definēta ar vienādojumu [89]

$$Foil\_Gain(L,R) \equiv t\left(\log_2 \frac{p_1}{p_1+n_1} - \log_2 \frac{p_0}{p_0+n_0}\right), \quad (13)$$

kur p_0 - pozitīvo sasaišu skaits likumā *R*;

n_0 - negatīvo sasaišu skaits likumā *R*;

p_1 - pozitīvo sasaišu skaits likumā R^1;

n_1 - negatīvo sasaišu skaits likumā R^1;

t - pozitīvo sasaišu skaits likumā *R*, kuras ir nosegtas pēc literas *L* pievienošanas likumam *R*.

Pamatojoties uz informācijas teoriju, $-log_2 \frac{p_0}{p_0+n_0}$ ir minimālais bitu skaits, kas nepieciešams, lai šifrētu klasifikāciju patvaļīgai pozitīvai sasaistei starp sasaistēm, kuras nosedz likums *R*. Līdzīgi $-log_2 \frac{p_1}{p_1+n_1}$ ir nepieciešamo bitu skaits, ja sasaiste ir iekļauta likumā R^1. Sakarā ar to, ka *t* ir tikai pozitīvo sasaišu skaits, ko nosedz likums *R*, un kuras paliek arī nosegtas ar likumu R^1, *Foil_Gain(L, R)* var tikt uzskatīta kā samazinājums nepieciešamo bitu skaitā, lai šifrētu visu pozitīvo sasaišu skaitu, ko nosedz likums *R*.

4.2.5.5. Rekursīvu likumu mācīšanās

Iepriekš minētajos piemēros netika apskatīti tādi varianti, kur litera, ko pievienojam, ir rekursīva, respektīvi, predikāts norāda uz likuma galvu. Tomēr, ja iekļausim mērķa predikātu predikātu ievades sarakstā, *FOIL* to arī ņems vērā, kad ģenerēs predikātus. Tas dod iespēju izveidot rekursīvus likumus, kad likuma galvā un ķermenī ir viens un tas pats predikāts. Piemēram, apskatīsim šādu gadījumu, kad likums nodrošina vecāka atrašanu (*Ancestor*).

IF *Parent(x, y)* THEN *Ancestor(x, y)*

IF *Parent(x, z)* ∧ *Ancestor(z, y)* THEN *Ancestor(x, y)*

Šādos gadījumos ir jāizmanto dažādi nosacījumi, kas garantētu, lai neveidotos bezgalīgas rekursijas, jo pastāv iespēju, ka šāda veida likums visu laiku izsauks pats sevi [89].

4.2.5.6. Apmācības dati

FOIL algoritms likumu ģenerēšanai izmanto apmācības datus. Apmācības datos ir pārbaudīta informācija (ekspertu vērtējums), kādas domēnu vērtības noved pie atbilstošas spējības klases. Jo lielāka ir apmācības datu kopa, jo precīzāki likumi var tikt uzģenerēti. Darbā izmantotie dati ir testa dati un paredzēti tikai algoritma darbības demonstrēšanai. Reālu apmācības datu izveidošana ir darbietilpīgs process, un nav paredzēts to izveidošanu iekļaut promocijas darbā. Lai izveidotu augstas kvalitātes apmācības datus, ir nepieciešami aptuveni 20 uzņēmumi, kuri piekristu spējo ekspertu vērtēšanai, kā arī ir nepieciešams ekspertu tīkls, kurš būtu ar mieru veikt šo 20 uzņēmumu vērtēšanu. Testa apmācības datos organizācijas spējības klasifikācijai ir trīs klases C1 = *Nav spējīga*, C2 = *Daļēji spējīga*, C3 = *Spējīga* un tas ir nepieciešams, lai vienkāršotu algoritma demonstrāciju.

4.1. tabula

FOIL algoritma apmācības dati

D1	**D2**	**D3**	**D4**	**Spējības klase**
3	2	1	1	C1
3	2	1	2	C2
3	2	2	1	C1
3	2	2	2	C3
3	1	1	1	C1
3	1	1	2	C2
3	1	2	1	C1
3	1	2	2	C3
2	2	1	1	C1
2	2	1	2	C2
2	2	2	1	C1
2	2	2	2	C3
2	1	1	1	C1
2	1	1	2	C2
2	1	2	1	C1
2	1	2	2	C1
1	2	1	1	C1
1	2	1	2	C1
1	2	2	1	C1
1	2	2	2	C3
1	1	1	1	C1
1	1	1	2	C2
1	1	2	1	C1
1	1	2	2	C1

4.2.5.7. Atrisinājums

Likumus var aprakstīt dažādos veidos, viens no tiem ir izmantot formu IF ... THEN ..., vai Horna klauzulu formā, *Head* ← *Body*, ko klasifikācijas uzdevuma gadījumā var norādīt kā Klase ← Nosacījums.

Vienkāršotā organizācijas novērtēšanas gadījumā, mēs varam izšķirt trīs spējības klases:

- C1 = Nav spējīga;
- C2 = Daļēji spējīga;
- C3 = Spējīga.

Likumus ir iespējams veidot, vadoties pēc apmācamo datu kopas. Cipars kvadrātiekavās norāda uz atrasto eksemplāru skaitu katrā klasē, pielietojot konkrēto likumu.

Katrā klasē ir sekojošs elementu skaits C1 = Nav spējīga = 15, C2 = Daļēji spējīga = 5, C3 = Spējīga = 4.

Piemēram,

IF D4 = 1 THEN Klase = C1 [#C2=0 #C3=0 #C1=12]

Konkrētais likums nosedz lielāko daļu klases C1, un nevienu no klasēm C2 un C3. Šajā gadījumā mums nenosegtas paliek 3 klases C1 instances, kas nozīmē, ka mums šīs klases atpazīšanai ir nepieciešams paplašināt likumu kopu, lai pārklātu arī pārējos pozitīvos šīs klases piemērus. Lai atrastu nākamo literu no mācīšanās datiem, jāizņem visi dati, ko nosedza pirmā litera. Ja mēs to izdarām, tad pie pozitīvajiem piemēriem paliek 4.2. tabula. redzamie dati.

4.2. tabula

Pāri palikušie neatlasītie C1 klases rezultāti

D1	D2	D3	D4	Spējības klase
3	2	1	2	C2
3	2	2	2	C3
3	1	1	2	C2
3	1	2	2	C3
2	2	1	2	C2
2	2	2	2	C3
2	1	1	2	C2

D1	D2	D3	D4	Spējības klase
2	1	2	2	C1
1	2	1	2	C1
1	2	2	2	C3
1	1	1	2	C2
1	1	2	2	C1

Tumšā krāsā ir iezīmēti visi atlikušie pozitīvie gadījumi. Tagad varam pamēģināt likumu kolekcijai izveidot papildu likumu. Pieņemsim, ka uzrakstām likumu šādā formā:

IF D4 = 2 THEN ... [#C2=5 #C3=4 #C1=3]

Kā redzams no informācijas kvadrātiekavās, šāds likums nostrādās uz visiem testa datiem, jo gala rezultātā ir paraugi no visām klasēm. Mums ir jāliek klāt papildu nosacījumi, lai iegūtu tikai vajadzīgās klases paraugus. Pieliksim klāt D3 literu, kā rezultātā iegūsim likumu:

IF D4 = 2 AND D3 = 2 THEN ... [#C2=0 #C3=4 #C1=2]

Pieliekot šādu nosacījumu, esam atsijājuši visus C2 gadījumus, pazaudējuši vienu pozitīvu C1 klases elementu, bet esam ieguvuši vienu ļoti vērtīgu atzinumu, kas attiecas uz klasi C2.

IF D4 = 2 AND D3 = 1 THEN ... [#C2=5 #C3=0 #C1=1]

Ar šādu nosacījumu kopu esam ieguvuši visus pozitīvos variantus klasei C3, bet tanī pašā laikā, esam atlasījuši vienu negatīvu piemēru no klases C2. Varam šim likumam pievienot klāt vēl kādu nosacījumu, lai atfiltrētu C1 klases elementus. Ja pieliksim atribūtu D2.

IF D4 = 2 AND D3 = 1 AND D2 = 2 THEN ... [#C2=2 #C3=0 #C1=1]

Pievienojot atribūtu, rezultāts ir tikai pasliktinājies. Likumam ir nepieciešams pievienot citus nosacījumus.

IF D4 = 2 AND D3 = 1 AND D2 = 1 THEN ... [#C2=3 #C3=0 #C1=0]

Pamainot likumu šādi, esam ieguvuši nosacījumus, kas daļēji pārklāj klasi C2. Esam atmetuši visus negatīvos piemērus, bet pazaudējuši arī 2 pozitīvos, kuru atrašanai būtu līdzīgi kā C1 klases gadījumā, jāmeklē papildu likums likumu kopai, kas nodrošinātu pārējo pozitīvo gadījumu atrašanu. Bet ko darīt, ja nav tādas likumu kopas,

kura pilnībā pārklātu mūs interesējošo klasi bez negatīviem piemēriem no citas? Šajā gadījumā, mums ir jāsamierinās ar to, ka pastāv dati, kurus neizdosies pilnībā klasificēt. Mums būtu svarīgi, lai šādu datu, kuri nepadodas normālai klasificēšanai būtu pēc iespējas mazāk, un ar to vienkārši ir jāsamierinās, jo reālās dzīves situācijās visdrīzāk būs izņēmumu gadījumi.

Dotais aprakstītais piemērs, bija vienkārša likumu veidošana balstoties uz nelieliem apmācības datiem, ja mums apmācāmo datu ir daudz, tad šāda pārmeklēšana katram gadījumam ir ļoti nogurdinoša un laikietilpīga. Tādēļ ir ērtāk izmantot tādus algoritmus kā *FOIL* un automātiski apstrādāt datus, un veidot likumu kopas. *FOIL* izmantošanas gadījumā, kad nosaka kuru literu labāk pievienot, tiek izmantota *FOIL_GAIN* funkcija, kura nosaka, kāds ir guvums, pievienojot konkrēto literu.

Likumu ģenerēšana klasei C1 sākas ar mācīšanās piemēru sadalīšanu pozitīvos un negatīvos [90][91].

4.3. tabula

(+) C1 klases apmācības dati

D1	**D2**	**D3**	**D4**	**Spējības klase**
3	2	1	1	C1
3	2	2	1	C1
3	1	1	1	C1
3	1	2	1	C1
2	2	1	1	C1
2	2	2	1	C1
2	1	1	1	C1
2	1	2	1	C1
2	1	2	2	C1
1	2	1	1	C1
1	2	1	2	C1
1	2	2	1	C1
1	1	1	1	C1
1	1	2	1	C1
1	1	2	2	C1

4.4. tabula

(-) C1 klases apmācības dati

D1	**D2**	**D3**	**D4**	**Spējības klase**
3	2	1	2	C2
3	2	2	2	C3
3	1	1	2	C2
3	1	2	2	C3
2	2	1	2	C2
2	2	2	2	C3

D1	D2	D3	D4	Spējības klase
2	1	1	2	C2
1	2	2	2	C3
1	1	1	2	C2

Sekojot algoritmam, ir jāizvēlas, kāda nākamā litera ir jāpievieno likumu kopai, tādēļ veicam *FOIL_GAIN* aprēķinu [89][90][91].

$I(T_1) = -\log_2(15/(15+9)) = -\log_2(0,625) = 0,678$

Izvērtējam visus domēnus un to vērtības izmantojot *FOIL_GAIN*. Rezultātā iegūstam, ka vislielākā jēga ir no literas ar D4(X,1). Šo literu arī pievienojam likumam. C1 ← D4(X, 1).

Litera	Aprēķins	*Foil_Gain*
D4(X, 1)	$I(T_2) = -\log_2(12/(12+0)) = -\log_2(1)=0$	12*(0,678-0) = 8,136
D4(X, 2)	$I(T_2) = -\log_2(3/(3+9)) = -\log_2(0,25)=2$	3*(0,678-2) = -3,966
D3(X, 2)	$I(T_2) = -\log_2(8/(8+4)) = -\log_2(0,666)= 0,586$	8*(0,678-0,586) = 0,736
D3(X, 1)	$I(T_2) = -\log_2(7/(7+5)) = -\log_2(0,583)= 0,788$	7*(0,678-0,788)=- 0,77
D2(X, 2)	$I(T_2) = -\log_2(7/(7+5)) = -\log_2(0,583)= 0,788$	7*(0,678-0,788)=- 0,77
D2(X, 1)	$I(T_2) = -\log_2(8/(8+4)) = -\log_2(0,666)= 0,586$	8*(0,678-0,586) = 0,736
D1(X,3)	$I(T_2) = -\log_2(4/(4+4)) = -\log_2(0,5)= 1$	4*(0,678-1) = -1,288
D1(X,2)	$I(T_2) = -\log_2(5/(5+3)) = -\log_2(0,625)= 0,678$	5*(0,678-0,678) = 0
D1(X,1)	$I(T_2) = -\log_2(6/(6+2)) = -\log_2(0,75)= 0,415$	2*(0,678-0,415) = 0,263

Ņemot vērā, ka konkrētais likums neatgriež nevienu negatīvu piemēru, varam beigt konkrētā likuma apstrādi, bet šis likums nenosedz visus pozitīvos piemērus. Tādēļ mums likumu kopā ir jāpieģenerē vēl papildu likums, kas varētu nosegt atlikušos pozitīvos piemērus. Lai to izdarītu, mēs no testa datiem izņemam ārā visus pozitīvos datus, kurus nosedz uzģenerētais likums.

(+)

D1	D2	D3	D4	Spējības klase
2	1	2	2	C1
1	2	1	2	C1
1	1	2	2	C1

Un atkal ir jāsāk apskatīt atribūtus, vai ir iespējams iegūt nosacījumus, kas palīdz klasificēt atlikušos pozitīvos datus.

$I(T_1) = -\log_2(3/(3+9)) = -\log_2(0,25) = 2$

Litera	**Aprēķins**	***Foil_Gain***
D4(X, 2)	$I(T_2) = -\log_2(3/(3+9)) = -\log_2(0,25)=2$	3*(2-2) = 0
D3(X, 2)	$I(T_2) = -\log_2(2/(2+4)) = -\log_2(0,333)= 1,586$	2*(2-1,586) = 0,414
D3(X, 1)	$I(T_2) = -\log_2(1/(1+5)) = -\log_2(0,166)= 2,59$	1*(2-2,59) = -0,59
D2(X, 2)	$I(T_2) = -\log_2(1/(1+5)) = -\log_2(0,166)= 2,59$	1*(2-2,59) = -0,59
D2(X, 1)	$I(T_2) = -\log_2(2/(2+4)) = -\log_2(0,333)= 1,586$	2*(2-1,586) = 0,414
D1(X,3)	$I(T_2) = -\log_2(0/(0+4)) = -\log_2(0)= \infty$	-
D1(X,2)	$I(T_2) = -\log_2(1/(1+3)) = -\log_2(0,25)= 2$	1*(2-2) = 0
D1(X,1)	$I(T_2) = -\log_2(2/(2+2)) = -\log_2(0,5)= 1$	2*(2-1) = 2

Kā redzams lielākais ieguvums, ir no literas D1(X,1), ko pievienojam klāt mūsu likumam. Ņemot vērā, ka, izpildot likumu D1(X,1), mums ir arī negatīvi paraugi, mums ir jāturpina likuma attīstīšana pievienojot tam klāt vēl vienu literu.

Litera	**Aprēķins**	***Foil_Gain***
D4(X, 2)	$I(T_2) = -\log_2(2/(2+2)) = -\log_2(0,5)=1$	2*(2-1) = 2
D3(X, 2)	$I(T_2) = -\log_2(1/(1+1)) = -\log_2(0,5)=1$	1*(2-1) = 1
D3(X, 1)	$I(T_2) = -\log_2(1/(1+1)) = -\log_2(0,5)=1$	1*(2-1) = 1
D2(X, 2)	$I(T_2) = -\log_2(1/(1+1)) = -\log_2(0,5)=1$	1*(2-1) = 1
D2(X, 1)	$I(T_2) = -\log_2(1/(1+1)) = -\log_2(0,5)=1$	1*(2-1) = 1

Pievienojam likumam klāt literu D4(X, 2), kā rezultātā iegūstam likumu:

$C1 \leftarrow D1(X,1) \wedge D4(X,2)$

Tagad ir jāpārliecinās, vai šis likums nosedz arī kādu negatīvu piemēru. Ja šis likums nosedz arī kādu negatīvu piemēru, mums ir jātrupina literas pievienošana. Mūsu likums šādā formā nosedz arī vienu negatīvu piemēru, tādēļ mums ir jāpievieno vēl viena litera.

Litera	**Aprēķins**	***Foil_Gain***
D3(X, 2)	$I(T_2) = -\log_2(1/(1+1)) = -\log_2(0,5)=1$	1*(2-1) = 1
D3(X, 1)	$I(T_2) = -\log_2(1/(1+1)) = -\log_2(0,5)=1$	1*(2-1) = 1
D2(X, 2)	$I(T_2) = -\log_2(1/(1+1)) = -\log_2(0,5)=1$	1*(2-1) = 1
D2(X, 1)	$I(T_2) = -\log_2(1/(1+1)) = -\log_2(0,5)=1$	1*(2-1) = 1

Visas no šīm literām ir vienlīdz sliktas, jo katra no tām atgriež vienu negatīvu piemēru. Tas norāda uz to, ka likumam ir nepieciešams pievienot vēl vienu literu, lai iegūtu tikai pozitīvu piemēru.

$C1 \leftarrow D1(X,1) \wedge D4(X,2) \wedge D3(X, 2)$

Litera	**Aprēķins**	***Foil_Gain***
D2(X, 2)	$I(T_2) = -\log_2(0/(0+1)) = -\log_2(0)=\infty$	-

Litera	Aprēķins	*Foil_Gain*
D2(X, 1)	$I(T_2) = -\log_2(1/(1+0)) = -\log_2(1)=1$	1*(2-0) = 2

Esam ieguvuši likumu, ko pievienot likumu kopai, kura definē klasi C1. Pēc likuma izpildes nav negatīvu piemēru. C1 klasi nosakošo likumu kopa tagad izskatās šādi:

- C1 ← D4(X, 1)
- C1 ← D1(X,1) ∧ D4(X,2) ∧ D3(X, 2) ∧ D2(X, 1)

Ņemot vērā, ka visi pozitīvie piemēri vēl nav nosegti, ir nepieciešams ģenerēt jaunu likumu, atlikušajiem pozitīvajiem piemēriem. Izņemam visus pozitīvos rezultātus, kurus nosedz esošais likums.

(+)

D1	D2	D3	D4	Spējības klase
2	1	2	2	C1
1	2	1	2	C1

(-)

D1	D2	D3	D4	Spējības klase
3	2	1	2	C2
3	2	2	2	C3
3	1	1	2	C2
3	1	2	2	C3
2	2	1	2	C2
2	2	2	2	C3
2	1	1	2	C2
1	2	2	2	C3
1	1	1	2	C2

$I(T_1) = -\log_2(2/(2+9)) = -\log_2(0,181) = 2,465$

Litera	Aprēķins	*Foil_Gain*
D4(X, 2)	$I(T_2) = -\log_2(2/(2+9)) = -\log_2(0,181)=2,465$	2*(2,465-2,465) = 0
D3(X, 2)	$I(T_2) = -\log_2(1/(1+4)) = -\log_2(0,2)=2,321$	1*(2,465-2,321) = 0,144
D3(X, 1)	$I(T_2) = -\log_2(1/(1+5)) = -\log_2(0,166)= 2,59$	1*(2,465-2,59)=-0,125
D2(X, 2)	$I(T_2) = -\log_2(1/(1+5)) = -\log_2(0,166)= 2,59$	1*(2,465-2,59)=-0,125
D2(X, 1)	$I(T_2) = -\log_2(1/(1+4)) = -\log_2(0,2)= 2,321$	1*(2,465-2,321) = 0,144
D1(X,3)	$I(T_2) = -\log_2(0/(0+4)) = -\log_2(0)= \infty$	-
D1(X,2)	$I(T_2) = -\log_2(1/(1+3)) = -\log_2(0,25)= 2$	1*(2,465-2) = 0,465
D1(X,1)	$I(T_2) = -\log_2(1/(1+2)) = -\log_2(0,333)= 1,586$	1*(2,465-1,586) = 0,879

C1 ← D1(X, 1)

Likums ir jāpapildina tālāk ar vēl kādu literu, jo eksistē arī negatīvi piemēri.

Litera	**Aprēķins**	***Foil_Gain***
D4(X, 1)	$I(T_2) = -\log_2(0/(0+9)) = -\log_2(0)=\infty$	-
D4(X, 2)	$I(T_2) = -\log_2(2/(2+2)) = -\log_2(0,5)=1$	2*(2,465-1) = 2,93
D3(X, 2)	$I(T_2) = -\log_2(1/(1+1)) = -\log_2(0,5)=1$	1*(2,465-1) =1,465
D3(X, 1)	$I(T_2) = -\log_2(1/(1+1)) = -\log_2(0,5)= 1$	1*(2,465-1)=1,465
D2(X, 2)	$I(T_2) = -\log_2(1/(1+1)) = -\log_2(0,5)= 1$	1*(2,465-1)=1,465
D2(X, 1)	$I(T_2) = -\log_2(1/(1+1)) = -\log_2(0,5)= 1$	1*(2,465-1) = 1,465

C1 ← D1(X, 1) ∧ D4(X,2)

Litera	**Aprēķins**	***Foil_Gain***
D3(X, 2)	$I(T_2) = -\log_2(0/(0+1)) = -\log_2(0)=\infty$	-
D3(X, 1)	$I(T_2) = -\log_2(1/(1+1)) = -\log_2(0,5)= 1$	1*(2,465-1)=1,465
D2(X, 2)	$I(T_2) = -\log_2(1/(1+1)) = -\log_2(0,5)= 1$	1*(2,465-1)=1,465
D2(X, 1)	$I(T_2) = -\log_2(0/(0+1)) = -\log_2(0)= \infty$	-

C1 ← D1(X, 1) ∧ D4(X,2) ∧ D3(X,1)

Litera	**Aprēķins**	***Foil_Gain***
D2(X, 2)	$I(T_2) = -\log_2(1/(1+0)) = -\log_2(1)= 0$	1*(2,465-0)=2,465
D2(X, 1)	$I(T_2) = -\log_2(0/(0+1)) = -\log_2(0)= \infty$	-

C1 ← D1(X, 1) ∧ D4(X,2) ∧ D3(X,1) ∧ D2(X, 2)

Tagad C1 klases noteikšanas likumu kopa izskatās sekojoši:

- C1 ← D4(X, 1)
- C1 ← D1(X,1) ∧ D4(X,2) ∧ D3(X, 2) ∧ D2(X, 1)
- C1 ← D1(X, 1) ∧ D4(X,2) ∧ D3(X,1) ∧ D2(X, 2)

Ņemot vērā, ka ir palicis vēl viens nenoklāts pozitīvs piemērs, mums likumu kopai ir jāpievieno vēl likums.

(+)

D1	**D2**	**D3**	**D4**	**Spējības klase**
2	1	2	2	C1

(-)

D1	D2	D3	D4	Spējības klase
3	2	1	2	C2
3	2	2	2	C3
3	1	1	2	C2
3	1	2	2	C3
2	2	1	2	C2
2	2	2	2	C3
2	1	1	2	C2
1	2	2	2	C3
1	1	1	2	C2

$I(T_1) = -\log_2(1/(1+9)) = -\log_2(0,1) = 3,321$

Litera	Aprēķins	*Foil_Gain*
D4(X, 2)	$I(T_2) = -\log_2(1/(1+9)) = -\log_2(0,1)=3,321$	1*(3,321-3,321) = 0
D3(X, 2)	$I(T_2) = -\log_2(1/(1+4)) = -\log_2(0,2)=2,321$	1*(3,321-2,321) = 1
D3(X, 1)	$I(T_2) = -\log_2(0/(0+5)) = -\log_2(0)= \infty$	-
D2(X, 2)	$I(T_2) = -\log_2(0/(0+5)) = -\log_2(0)= \infty$	-
D2(X, 1)	$I(T_2) = -\log_2(1/(1+4)) = -\log_2(0,2)= 2,321$	1*(3,321-2,321) = 1
D1(X,3)	$I(T_2) = -\log_2(0/(0+4)) = -\log_2(0)= \infty$	-
D1(X,2)	$I(T_2) = -\log_2(1/(1+3)) = -\log_2(0,25)= 2$	1*(3,321-2) = 1,321
D1(X,1)	$I(T_2) = -\log_2(0/(0+2)) = -\log_2(0)= \infty$	-

C1 ← D1(X,2)

Litera	Aprēķins	*Foil_Gain*
D4(X, 2)	$I(T_2) = -\log_2(1/(1+3)) = -\log_2(0,25)=2$	1*(3,321-2) = 1,321
D3(X, 2)	$I(T_2) = -\log_2(1/(1+1)) = -\log_2(0,5)=1$	1*(3,321-1) = 2,321
D3(X, 1)	$I(T_2) = -\log_2(0/(0+5)) = -\log_2(0)= \infty$	-
D2(X, 2)	$I(T_2) = -\log_2(0/(0+5)) = -\log_2(0)= \infty$	-
D2(X, 1)	$I(T_2) = -\log_2(1/(1+1)) = -\log_2(0,5)=1$	1*(3,321-1) = 2,321

C1 ← D1(X,2) ∧ D3(X,2)

Litera	Aprēķins	*Foil_Gain*
D4(X, 2)	$I(T_2) = -\log_2(1/(1+1)) = -\log_2(0,5)=1$	1*(3,321-1) = 2,321
D2(X, 2)	$I(T_2) = -\log_2(0/(0+5)) = -\log_2(0)= \infty$	-
D2(X, 1)	$I(T_2) = -\log_2(1/(1+0)) = -\log_2(1)=0$	1*(3,321-0) = 3,321

C1 ← D1(X,2) ∧ D3(X,2) ∧ D2(X,1)

Tagad C1 klases likumu noteikšanas kopa izskatās sekojoši:

- C1 ← D4(X, 1)
- C1 ← D1(X,1) ∧ D4(X,2) ∧ D3(X, 2) ∧ D2(X, 1)
- C1 ← D1(X, 1) ∧ D4(X,2) ∧ D3(X,1) ∧ D2(X, 2)

- C1 ← D1(X,2) ∧ D3(X,2) ∧ D2(X,1)

Mums nav vairs palicis neviens pozitīvs piemērs, kuru nenosedz likumu kopa. Tādēļ C1 klases likumu noteikšanas kopa ir uzskatāma par gatavu, un varam ķerties klāt ārējam ciklam un nākamās klases likuma ģenerēšanai. Līdzīgā veidā tiek ģenerēti likumi arī pārējām klasēm C2 un C3. C2 un C3 klašu likumu ģenerēšana ir apskatāma 4. pielikumā, jo pamatdarbā tā aizņem pārāk daudz vietas. Zemāk ir redzams likumu kopu apkopojums, kurš iegūts līdz galam, apstrādājot apmācības datus ar *FOIL* algoritmu.

4.5. tabula

FOIL algoritma ģenerēto likumu apkopojums

C1	• C1 ← D4(X, 1) [1] • C1 ← D1(X,1) ∧ D4(X,2) ∧ D3(X, 2) ∧ D2(X, 1) [2] • C1 ← D1(X, 1) ∧ D4(X,2) ∧ D3(X,1) ∧ D2(X, 2) [3] • C1 ← D1(X,2) ∧ D3(X,2) ∧ D2(X,1) [4]
C2	• C2 ← D4(X, 2) ∧ D3(X, 1) ∧ D2(X, 1) [5] • C2 ← D4(X, 2) ∧ D3(X,1) ∧ D2(X, 2) ∧ D1(X, 3) [6] • C2 ← D3(X, 1) ∧ D4(X, 2) ∧ D1(X, 2) [7]
C3	• C3 ← D4(X, 2) ∧ D3(X, 2) ∧ D1(X, 3) [8] • C3 ← D4(X, 2) ∧ D3(X, 2) ∧ D2(X, 2) [9]

Lai pārbaudītu likumu precizitāti, izskatīsim apmācāmos datus, izmantojot izveidotos likumus. Sanumurēsim apmācības datus un norādīsim, kāda ir sagaidāmā klase. Blakus kolonā norādīsim, kāda klase ir iznākusi izmantojot mūsu likumu kopas, un kurš likums ir nostrādājis.

4.6. tabula

FOIL ģenerēto likumu pārbaudes apkopojums

Nr.	D1	D2	D3	D4	Sagaidāmā klase	Rezultāts	Nostrādājušais likums
1	3	2	1	1	C1	C1	1
2	3	2	1	2	C2	C2	6
3	3	2	2	1	C1	C1	1
4	3	2	2	2	C3	C3	8,9
5	3	1	1	1	C1	C1	1
6	3	1	1	2	C2	C2	5
7	3	1	2	1	C1	C1	1
8	3	1	2	2	C3	C3	8
9	2	2	1	1	C1	C1	1
10	2	2	1	2	C2	C2	7
11	2	2	2	1	C1	C1	1
12	2	2	2	2	C3	C3	9
13	2	1	1	1	C1	C1	1
14	2	1	1	2	C2	C2	5,7
15	2	1	2	1	C1	C1	1,4

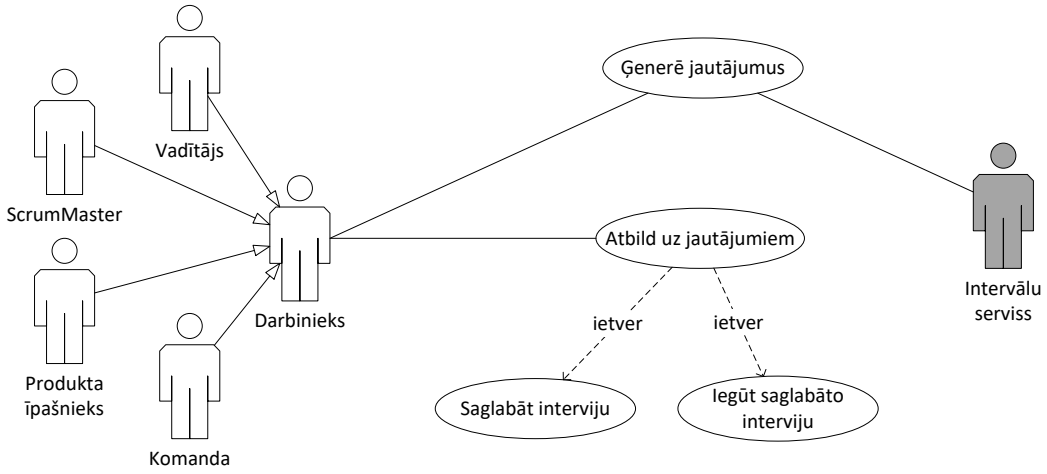
Nr.	D1	D2	D3	D4	Sagaidāmā klase	Rezultāts	Nostrādājušais likums
16	2	1	2	2	C1	C1	4
17	1	2	1	1	C1	C1	1
18	1	2	1	2	C1	C1	3
19	1	2	2	1	C1	C1	1
20	1	2	2	2	C3	C3	9
21	1	1	1	1	C1	C1	1
22	1	1	1	2	C2	C2	5
23	1	1	2	1	C1	C1	1
24	1	1	2	2	C1	C1	2

4.3. Spējības noteikšanas rīks

Spējības noteikšanai ir iteratīvs raksturs, un ir nepieciešams kāds rīks, lai varētu ērtā veidā organizēt ekspertu intervēšanu SII vērtības noteikšanai un darbinieku intervēšanu DVV vērtības noteikšanai. Rīkam ir jānodrošina ērts veids, kā attēlot iegūto informāciju.

4.3.1. ODSN lietotāja stāsti

ODSN metodes lietotāja stāsti nosaka, kādos veidos metode var tikt izmantota. Pirmais lietošanas stāsts ir "Jautājumu atbildēšana", kurš sastāv no divām aktivitātēm. Pirmā aktivitāte ir jautājumu kopas ģenerēšana, otrā aktivitāte ir atbildēšana uz jautājumiem (sk. 4.7. att.).

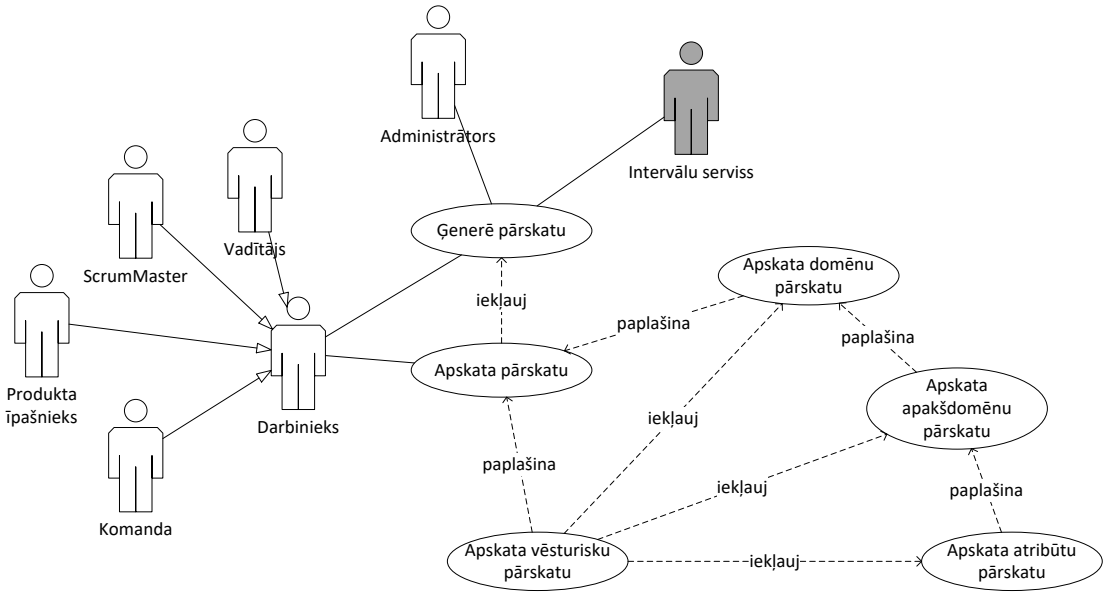


4.7. att. Jautājumu atbildēšanas lietotāju stāsts.

Jautājumu ģenerēšanu var ierosināt kāds no darbiniekiem manuāli vai automātiski, to var izdarīt arī "Intervālu serviss". "Intervālu servisu" var konfigurēt, lai intervēšanu varētu atkārtot konkrētos lietotāja izvēlētos intervālos. Lai iegūtu precīzāku informāciju par organizācijas domēniem, ir vēlams intervēt cilvēkus no dažādiem organizācijas līmeņiem (vadītājs, komanda, produkta īpašnieks, *ScrumMaster*).

Intervēšanas laikā darbiniekam ir iespēja saglabāt un vēlāk turpināt iesākto intervēšanas procesu.

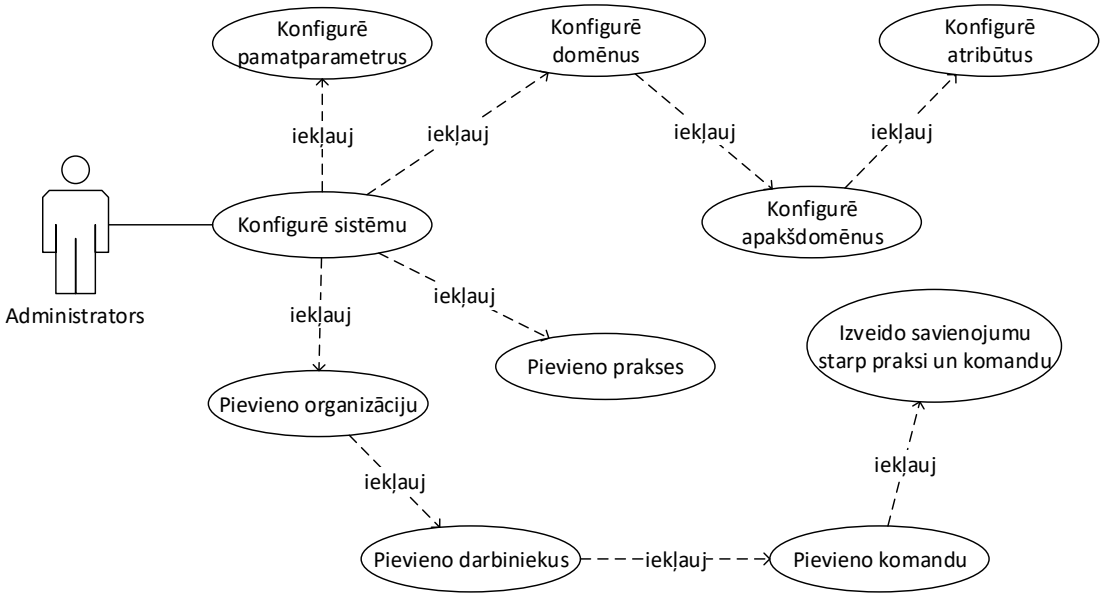
Otrais lietotāja stāsts ir "Pārskatu ģenerēšana un apskatīšana", un tas nosaka, kādā veidā un kādi pārskati ir apskatāmi (sk. 4.8. att.). Gadījumā, ja pārskati vēl nav izveidoti, ir iespējams to ģenerēšanu inicializēt manuāli.



4.8. att. Pārskatu ģenerēšana un apskatīšana.

Sagatavotie pārskati attēlo informāciju par dažādiem organizācijas domēniem, apakšdomēniem un atribūtiem. Nepieciešamības gadījumā ir iespējams apskatīt pārskatu vēsturiskā griezumā.

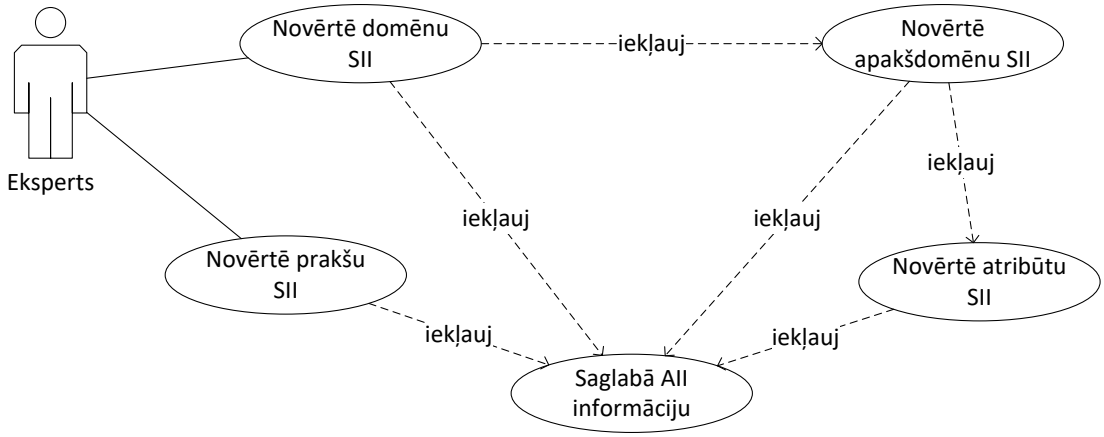
Trešais lietotāja stāsts ir rīka sagatavošana darbam (4.9. att.).



4.9. att. Rīka konfigurācija.

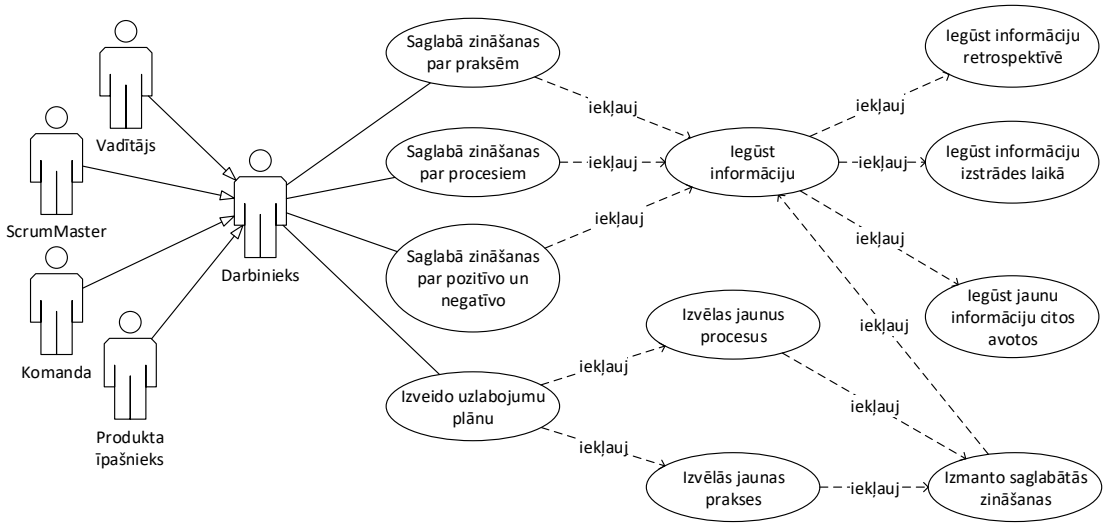
Metodes funkcionēšanas nodrošināšanai ir nepieciešams konfigurēt domēnus, apakšdomēnus un atribūtus, kā arī ir nepieciešams pievienot informāciju par organizāciju, darbiniekiem un komandu. Pēc komandas un prakšu pievienošanas ir nepieciešams norādīt, ar kādām praksēm pārbaudes periodā darbosies komanda. Ir nepieciešams ņemt vērā, ka komanda var izvēlēties jaunas vai jau mēģinātas prakses labāka rezultāta sasniegšanai.

Ceturtais lietotāja stāsts ir ekspertu noteiktā SII saglabāšana. Izmantojot ekspertu tīklu un metodi DELPHI [99].



4.10. att. SII vērtības noteikšana.

Eksperti nosaka SII domēniem, apakšdomēniem, atribūtiem un praksēm.

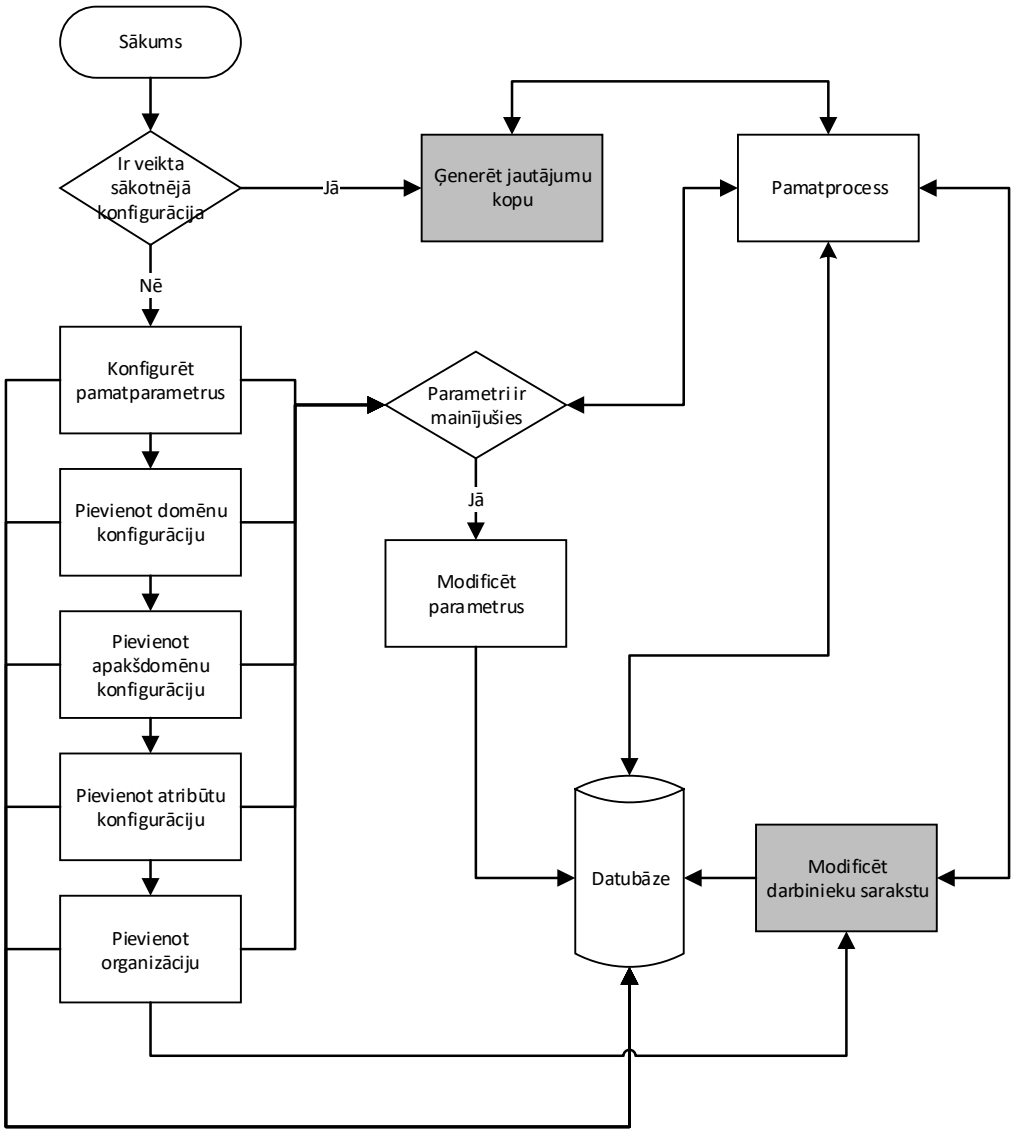


4.11. att. Uzlabojumu plāna veidošana.

Organizācijas spējības noteikšana sniedz iespēju izstrādāt uzlabojumu plānu (sk. 4.11. att.). Uzlabojumu plāna izstrādē liela nozīme ir informācijas iegūšanai. Tā tiek iegūta retrospektīvās izstrādes laikā un citos informācijas avotos. Iegūtā informācija palīdz izvēlēties atbilstošākas prakses un modificēt procesus.

4.3.2. Datu plūsmu diagrammas

Metode ODSN implementē iepriekš aprakstīto konceptuālo modeli. ODSN var sadalīt divās datu plūsmās. Pirmā datu plūsma ir metodes ārējais cikls, kurš definē, kas darāms pirms vērtēšanas uzsākšanas un kas darāms globālā kontekstā (4.12. att.). Otrā datu plūsma ir metodes iekšējais cikls, kurš tiek izmantots regulāri organizācijas spējības noteikšanai un uzlabojumu plānu sastādīšanai.



4.12. att. Metodes ODSN ārējā datu plūsma.

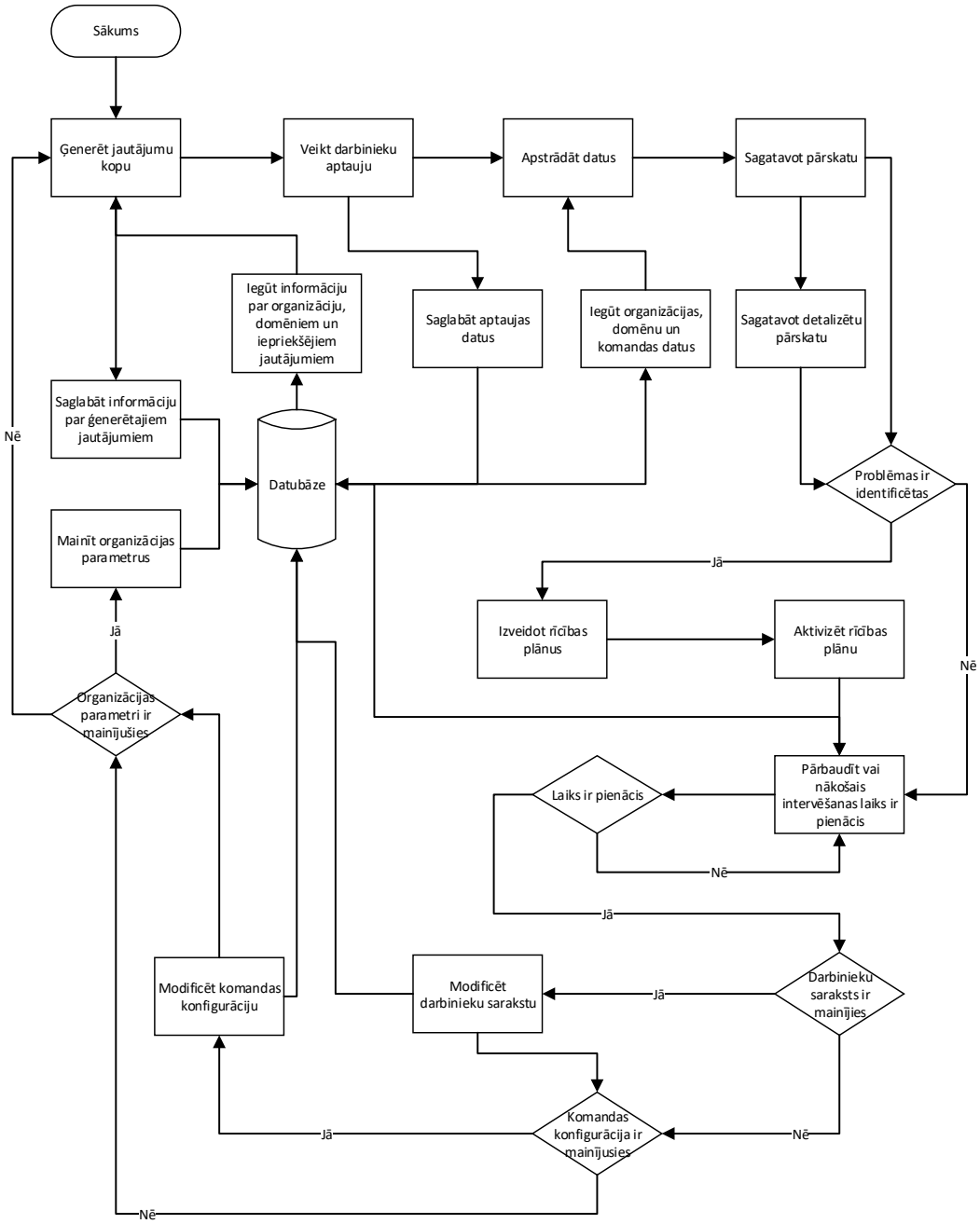
Pirmās datu plūsmas sākumā tiek veikta pārbaude, vai sākotnējā rīka konfigurācija ir veikta. Ja konfigurācija nav veikta, tiek uzsākts konfigurācijas process:

- konfigurēt parametrus – globāla rakstura parametri, tādi kā intervēšanas intervāls vai ģenerētās jautājumu kopas izmērs;
- pievienot domēnu konfigurāciju – datubāzei tiek pievienoti pieci organizācijas spējības domēni. Gadījumos, kad ir nepieciešamība pievienot darbā vēl neaprakstītu domēnu, bet kādā īpašā situācijā tas ir nepieciešams, to var izdarīt šajā aktivitātē;
- pievienot apakšdomēnu konfigurāciju – datubāzei tiek pievienoti darbā aprakstītie spējības apakšdomēni (sk. 3. Apakšnodaļā). Īpašos gadījumos var pievienot vairāk apakšdomēnus nekā aprakstīts darbā, ja tas ir nepieciešams

konkrētajai organizācijai. Šajos gadījumos nedrīkst aizmirst, ja tiek pievienoti neaprakstīti un nenovērtēti apakšdomēni, ir nepieciešams veikt to AII noteikšanu, izmantojot metodi DELPHI [100] vai līdzīgu;

- pievienot atribūtu konfigurāciju – katrs apakšdomēns tiek aprakstīts, izmantojot atribūtu kopu. Darbā iekļautie atribūti un to vērtības tiek ievadītas ārējā cikla laikā. Nepieciešamības gadījumā ir iespējams pievienot vairāk atribūtus tos vispirms izvērtējot [100];
- pievienot organizāciju – metodei ODSN paredzētais rīks neaprobežojas ar vienas organizācijas vērtēšanu un tehniski sniedz iespēju strādāt ar vairākām organizācijām;
- modificēt darbinieku sarakstu – pie pirmās datu ievades šajā aktivitātē notiks darbinieku saraksta pievienošana. Darbinieku saraksts ir nepieciešams, jo bez šīs informācijas nebūs iespējama komandu veidošana un interviju nosūtīšana.

Gadījumos, ja sākotnējā konfigurācija ir veikta, tiek inicializēts jautājumu ģenerēšanas process, kura rezultātā tiek iegūta atbilstoša izmēra jautājumu kopa katram lietotājam. Pirms jautājumu ģenerēšanas vienmēr tiek veikta pārbaude, vai kādi no parametriem nav mainījušies. Iekšējais cikls nodrošina spējības noteikšanu un tiek iteratīvi atkārtots katru reizi, kad nepieciešams vērtējums (4.13. att.).



4.13. att. Metodes ODSN iekšējā datu plūsma.

Iekšējā datu plūsma sastāv no aktivitāšu un nosacījumu kopas, kura sākas ar jautājumu ģenerēšanu:

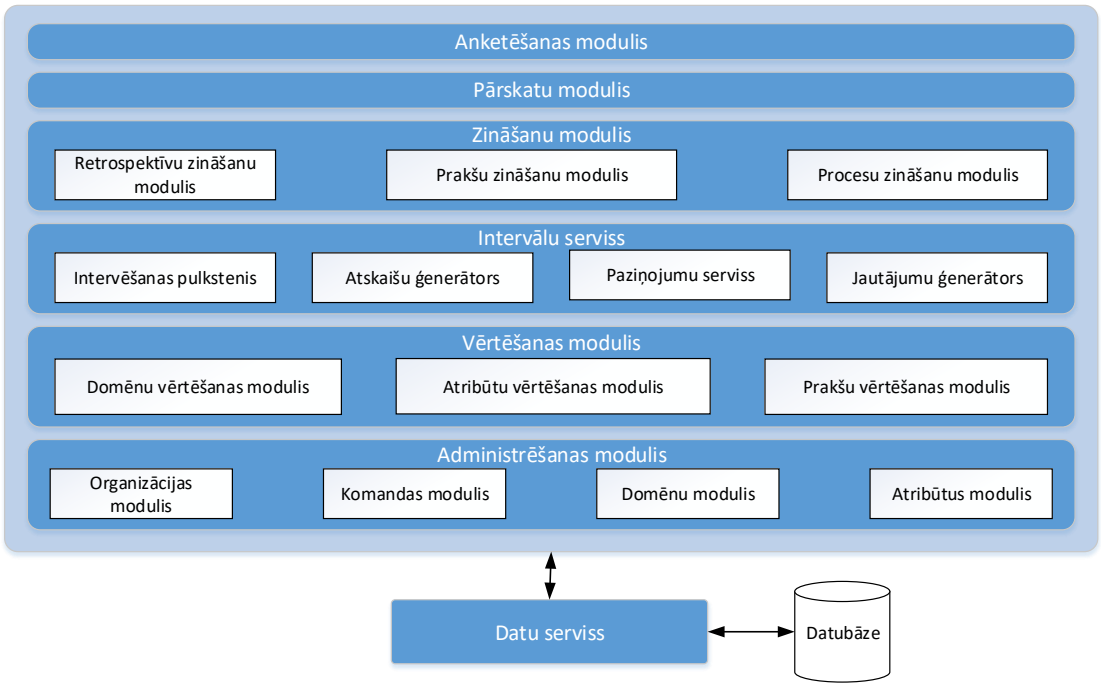
- ģenerēt jautājumu kopu – jautājumu kopas ģenerēšanā ir iesaistīta informācija par iepriekš ģenerētajiem jautājumiem katram darbiniekam, par jautājumu svarīguma pakāpi. Ņemot vērā, ka viena no ODSN pamata nostādnēm ir nenomocīt darbiniekus ar lielu jautājumu kopu, bet tai pašā laikā iegūt pēc iespējas vairāk noderīgas informācijas spējības noteikšanai, tad ģenerētajiem jautājumiem ir jābūt ar augstu pievienoto vērtību;

- saglabāt informāciju par ģenerētajiem jautājumiem – informācija par ģenerēto jautājumu kopu ir jāsaglabā, jo tā tiek izmantota nākamo jautājumu ģenerēšanā;
- iegūt informāciju par organizāciju, domēniem un iepriekšējiem jautājumiem – ģenerēšanas laikā, tiek izmantota informācija par iepriekšējiem jautājumiem un domēniem;
- veikt darbinieku aptauju – darbinieku aptaujas tiek veiktas iteratīvi. Organizācija nosaka, cik garas ir iterācijas un var izlemt, vai ir nepieciešama ārkārtas intervēšana;
- saglabāt aptaujas datus – anketēšanas dati tiek saglabāti, lai tos varētu apstrādāt vēlāk;
- iegūt organizācijas, domēnu, apakšdomēnu, atribūtu un intervēšanas datus, lai varētu apstrādāt un klasificēt organizāciju;
- apstrādāt datus – aktivitātē notiek iegūto datu apstrāde un klasificēšana;
- sagatavot pārskatu – notiek domēnu pārskatu sagatavošana. Tas ir nepieciešams, lai pārskatu apskatīšanas laikā tie jau būtu sagatavoti;
- sagatavot detalizētu pārskatu – notiek apakšdomēnu un atribūtu pārskatu sagatavošana;
- izveidot rīcības plānu – ņemot vērā pārskatos un iterācijās iegūto informāciju, tiek izveidots rīcības plāns;
- aktivizēt rīcības plānu – ja problēmas ir identificētas un eksistē rīcības plāns, tad šajā aktivitātē notiek tā izpilde. Rīcības plāns var sastāvēt no viena vai vairākiem punktiem;
- pārbaudīt, vai nākamais intervēšanas laiks ir pienācis – intervēšanas intervāls tiek saglabāts sistēmā, un intervēšanas serviss aktivizē jautājumu ģenerēšanu un intervēšanas procesu;
- modificēt darbinieku sarakstu – pirms jaunas jautājumu kopas ģenerēšanas ir jāveic pārbaude, vai nav nepieciešams modificēt darbinieku sarakstu. Darbiniekiem, kuri ir pievienojušies organizācijai, nākamajā intervēšanas procesā ir jāģenerē jautājumu kopa;
- modificēt komandas konfigurāciju – gadījumos, ja darbinieku sastāvs ir mainījies ir nepieciešamas izmaiņas arī komandu konfigurācijā;

- mainīt organizācijas parametrus – gadījumos, ja ir mainījušies organizācijas līmeņa parametri, tos ir nepieciešams modificēt pirms jaunas jautājumu kopas ģenerēšanas.

4.3.3. Rīka arhitektūra

Nodaļā 4.1 ir aprakstīts ODSN metodes konceptuālais modelis. Daļa no metodes ir spējības noteikšanas rīks, kura arhitektūra ir apskatāma 4.14. att. . Arhitektūra ir iedalāma 7 moduļos, kur katrs modulis nodrošina nepieciešamo funkcionalitāti:



4.14. att. ODSN metodes rīka arhitektūra.

- anketēšanas modulis – modulis nodrošina anketēšanas procesu. Anketēšanas modulī darbinieki redz viņiem sagatavotos jautājums un viņi uz tiem var atbildēt;
- pārskatu modulis - nodrošina pārskatu ģenerēšanas inicializēšanu un pārskatu apskatīšanu;
- zināšanu modulis – iegūtās zināšanas ir nepieciešams saglabāt un vēlāk apskatīt. Zināšanu modulis sastāv no trīs apakšmoduļiem:
    - retrospektīvu zināšanu modulis – saglabājam retrospektīvā iegūtās zināšanas par labo un slikto pieredzi iepriekšējā iterācijā;
    - prakšu zināšanu modulis – organizācijas un komandas var izvēlēties dažādas prakses mērķu sasniegšanai. Metode paredz, ka informācija, par

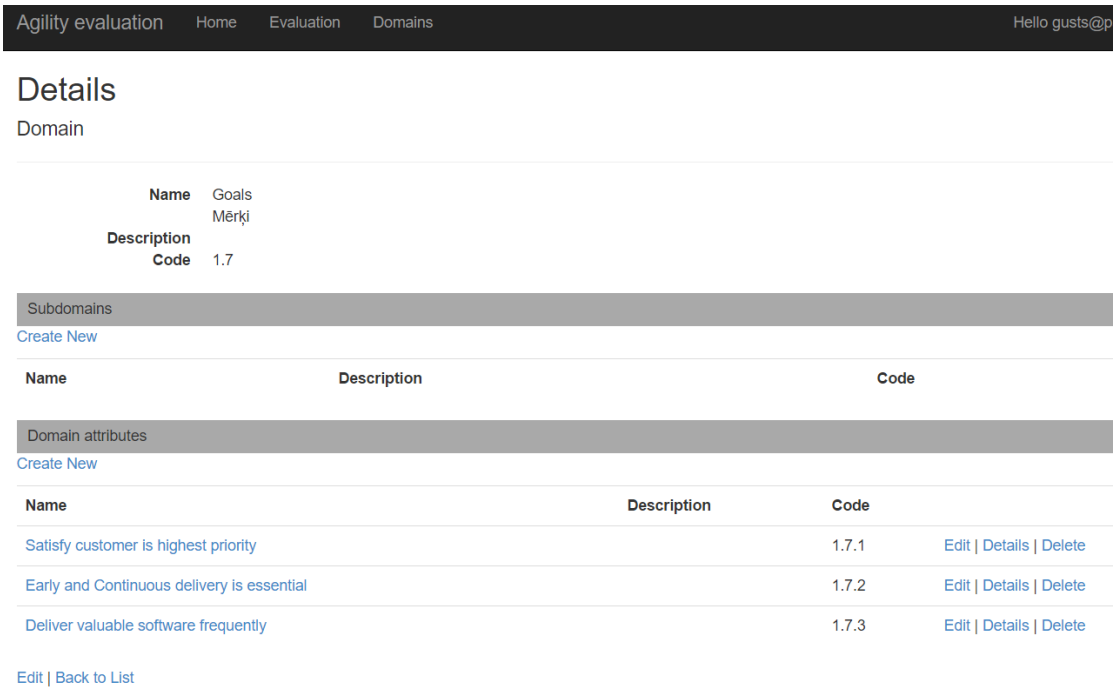
praksi tiek saglabāta. Tādā veidā ļaujot izvēlēties un strādāt ar daudzsološām praksēm, malā atliekot neperspektīvās;
    - procesu zināšanu modulis – līdzīgi kā informācija par praksēm, arī informācija par izmēģinātajiem procesiem ir jāsaglabā, lai varētu izvēlēties savai organizācijai atbilstošus procesus.
- intervālu serviss – apkopo procesus, kuri tiek izpildīti noteiktos laika intervālos un pagaidām sastāv no četriem apakš moduļiem:
    - intervēšanas pulkstenis – uztur intervālus un inicializē procesus;
    - atskaišu ģenerators – nodrošina atskaišu sagatavošanu, lai tās uzreiz būtu pieejamas ieinteresētajiem darbiniekiem;
    - paziņojumu serviss – konkrētos intervālos izsūta darbiniekiem saiti uz intervēšanas lapu;
    - jautājumu ģenerators – sagatavo jautājumu kopu katram darbiniekam un izsauc paziņojumu servisu.
- vērtēšanas modulis – nodrošina ekspertus ar iespēju novērtēt domēnu, atribūtu un prakšu SII. Vērtēšanas modulis sastāv no trīs apakšmoduļiem:
    - domēnu vērtēšanas modulis;
    - domēnu atribūtu vērtēšanas modulis;
    - prakšu vērtēšanas modulis.
- administrēšanas modulis – nodrošina iespēju administrēt metodei nepieciešamo informāciju un sastāv no četriem apakšmoduļiem:
    - organizācijas modulis – administrē organizācijas līmeņa informāciju, tādu kā organizācijas izmērs, pieredze utt.;
    - komandas modulis – administrē komandas līmeņa informāciju, tai skaitā darbinieku sarakstu un komandu sastāvu;
    - domēnu modulis – administrē domēna un apakšdomēnu informāciju;
    - atribūtu modulis – administrē apakšdomēnu atribūtus.
- datu serviss – nodrošina datu iegūšanu un saglabāšanu kādā no datubāzēm. Datubāze var būt gan lokāla, gan izvietota mākonī.

4.3.4. Prototips

Metodes pārbaudei izveidotais rīks ir izstrādāts, izmantojot *Microsoft ASP.NET MVC 5* tehnoloģiju, un ir izvietots *Microsoft* izveidotajā platformā *Microsoft Azure*. Datubāze ir balstīta uz *Microsoft SQL* servera bāzes. Rīka programmatūras kods ir

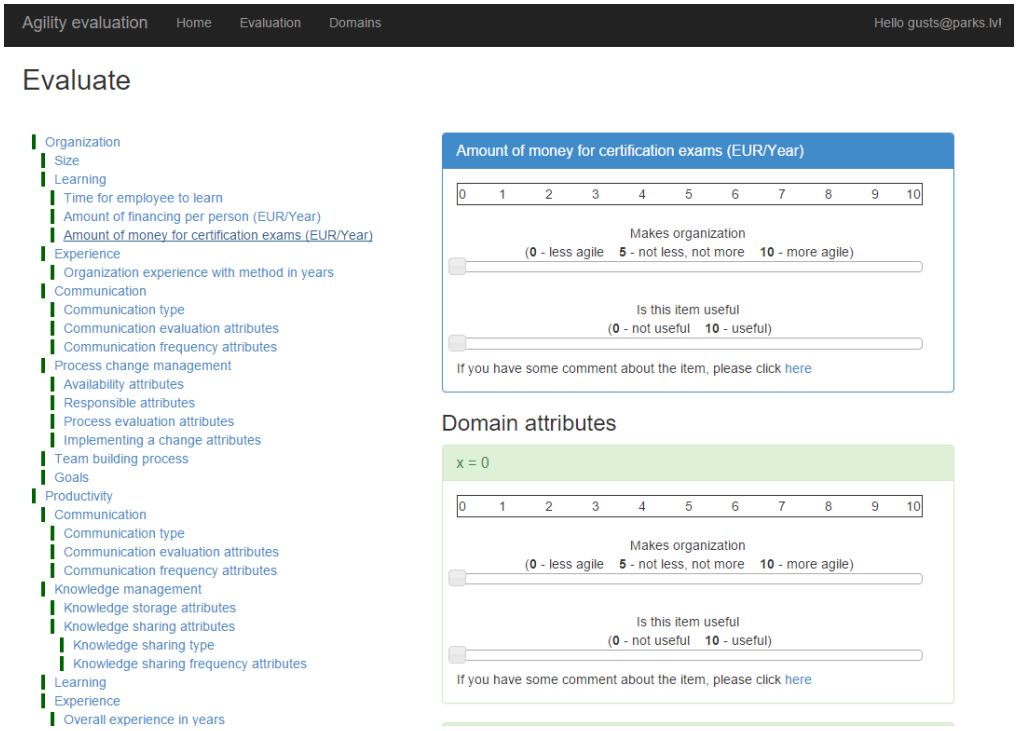
veidots programmēšanas valodā C# un izmanto HTML (angl. *HyperText Markup Language*), CSS (angl. *Cascading Style Sheets*), *JavaScript*, *jQuery* un *jQuery UI* bibliotēku funkcionalitātes nodrošināšanai.

Ērtākai rīka izmantošanai un plašākas funkcionalitātes nodrošināšanai mainīgajā vidē, izstrādāto OSM modeli ir iespējams modificēt rīka administrēšanas sadaļā (sk. 4.15. att.). Modeļa izstrādes laikā, tas vairākkārt ir mainīts un atšķiras no sākotnējās versijas. Ir jāņem vērā, ja tiek veiktas izmaiņas OSM modelī, tad ir nepieciešams veikt atkārtotu OSM elementu vērtēšanu, izmantojot SME tīklu.



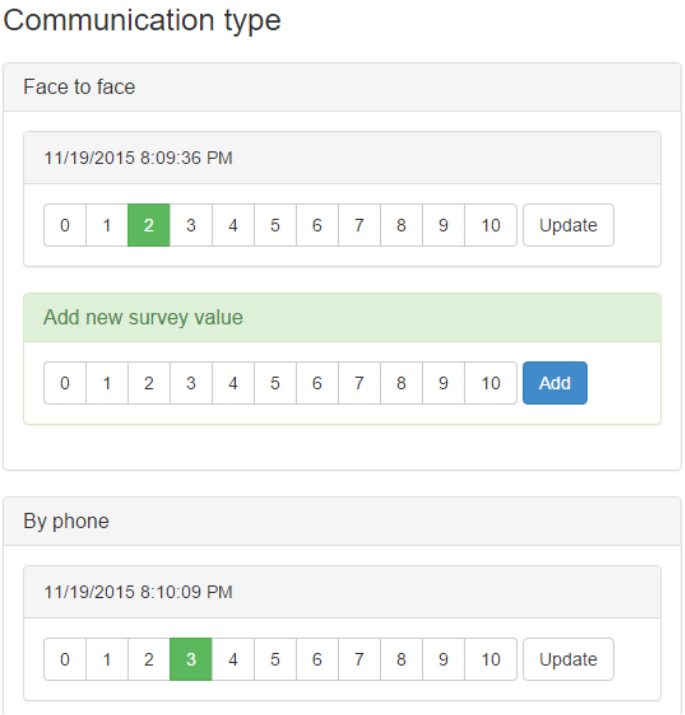
4.15. att. Domēnu un atribūtu administrēšana.

Visiem pieaicinātajiem SME tiek piešķirti lietotājvārdi un paroles rīka izmantošanai. SME var pieslēgties rīkam un veikt OSM elementu vērtēšanu. Pēc katra izpildītā vērtēšanas soļa informācija tiek saglabāta datubāzē, tādā veidā nodrošinot, ka vērtēšana var tikt veikta brīdī, kad SME ir pieejams laiks (sk. 4.16. att.).



4.16. att. OSM elementu vērtēšanas skats.

Uzņēmuma darbiniekiem pieejamajā sadaļā, ir iespējams sniegt atbildi uz ģenerētajiem jautājumiem (sk. 4.17. att.). Gadījumos, kad lietotājs uz šo jautājumu ir atbildējis iepriekš, lietotājs var redzēt iepriekš sniegto atbildi un laiku.



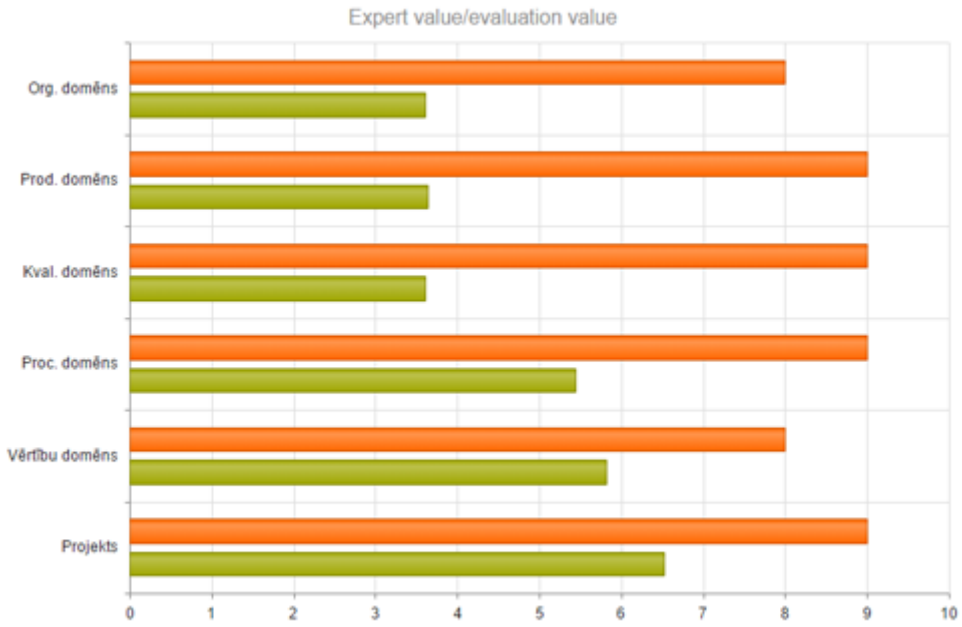
4.17. att. Vērtēšanas skats, ko redz darbinieks.

Darbinieks ar atbilstošajām tiesībām var veidot vērtēšanas pārskatu, kurā ir iespējams redzēt vērtēšanas rezultātu (sk. 4.18. att.). Vērtēšanas pārskatam ir iespējams izvēlēties vērtēšanas periodu, attēlojamo detalizācijas līmeni, kā arī eksportēt pārskatu uz Microsoft Excel.

Tiek ņemtas vērā darbinieka zināšanas	8	7.00
Tiek ņemta vērā iepriekšējā dalībnieku sadarbošanās	6	0.00
Iepriekšējās sadarbības rezultāti tiek uzglabāti	5	0.00
Tiek veidotas komandas ar augstu motivācijas līmeni	9	4.00
Tiek veidota pašorganizējoša komanda	8	1.00
Mērķi	9	2.23
Viena no augstākajām prioritātēm ir apmierināt klientu	8	5.00
Būtiska ir agra un nepārtraukta piegāde	9	1.00
Piegādāt vērtīgu programmatūru pēc iespējas biežāk	9	1.00
Prod. domēns	9	3.64
Komunikācija	9	4.47
Komunikāciju veids	9	5.77
Aci pret aci	9	8.00
Telefoniski	7	5.00
Rakstiski	5	5.00
Skype vai alternatīva sarunu noorganizēšana	7	9.00
Video konferences	7	1.00
Komunikācijas vērtēšanas atribūti	6	0.00
Vērtēšanas kritēriji ir definēti	3	0.00
Komunikācijas vērtēšanas tiek veikta regulāri	6	0.00
Vērtēšanas kritēriji ir pieejami visiem	4	0.00
Vērtēšanas kritēriji ir saglabāti	3	0.00
Komunikācijas biežuma atribūti	6	7.00
Vairākas reizes dienā	7	7.00
Zināšanu pārvaldība	8	0.98
Zināšanu uzglabāšanas atribūti	5	0.98
Zināšanas tiek uzglabātas	7	0.00

4.18. att. Rezultātu atskaites skats.

Pārskatā tiek izmantotas dažādas krāsas vērtību attēlošanai. Zaļā krāsa norāda, ka atbilstošais OSM elements ir pietuvināts spējam novērtējumam, bet sarkans, ka elements nav sasniedzis vēlamo spējības līmeni. Dzelteni/Oranžā krāsa norāda, ka elements nav sasniedzis vēlamo spējības līmeni un ir nepieciešamas izmaiņas, lai tas to varētu sasniegt.



4.19. att. Rezultātu attēlošana grafiskā veidā līmeņu griezumā.

Lietotājam ir iespēja apskatīt rezultātus grafiskā veidā (sk. 4.19. att.). Šāds attēlošanas veids sniedz iespēju ērtākā veidā identificēt problemātiskos apgabalus. Grafikā izmantojot oranžo krāsu tiek attēlots ekspertu viedoklis par elementa svarīgumu, un zaļā krāsā tiek attēlots darbinieku vērtējums konkrētajā uzņēmumā. Šajā gadījumā var redzēt, ka pārbaudes uzņēmumā ir būtiskas nobīdes no vēlamā rezultāta augstākā līmeņa domēnos, un uzņēmumam ir nepieciešams detalizētāk apskatīt katru konkrēto līmeni, lai noskaidrotu zemā vērtējuma iemeslu.

4.4. Secinājumi

Spējības noteikšanai ir izstrādāts konceptuālais modelis, kurš raksturo galvenos elementus, kuri nepieciešami spējības noteikšanai. Konceptuālais modelis sastāv no tādiem elementiem, kā “ievades modulis” (nodrošina datu iegūšanu no lietotājiem), “izvades modulis” (nodrošina aprēķinātās informācijas attainošanu) un “klasificēšanas komponente” (veic nepieciešamos aprēķinus, lai noteiktu OSM elementu spējības līmeni).

Balstoties uz konceptuālo modeli, ir izstrādāta metode ODSN. Metodes ODSN process sastāv no vairākiem posmiem, svarīgākie no kuriem ir “DAA elementu novērtēšana”, “Jautājumu ģenerēšana”, “Orgnizācijas spējības noteikšana” un “Uzlabojumu plāna izstrāde”.

Būtiska metodes sastāvdaļa ir "jautājumu ģenerātors", kurš nodrošina jautājumu sagatavošanu anketēšanas dalībniekiem. "jautājumu ģenerātors" ir nepieciešams, lai vienā reizē, vienam darbiniekam nebūtu jāatbild uz visiem jautājumu kopas *Q* jautājumiem. Neliela jautājumu kopa nepārslogo anketējamo un tā sniegtās atbildes vairāk atbilst patiesajam stāvoklim.

Lai ērtākā veidā attainotu SII (ekspertu vērtējums) un DVV (darbinieku vērtējums) vērtības, ir izstrādāts DAA vērtību koks. DAA kokam līdzīga struktūra tiek izmantota, lai atainotu vairāku komandu un projektu spējību, gadījumos, ja organizācija vienlaikus izstrādā vairākus projektus, un katra projekta izstrādē ir iesaistīta vairāk nekā viena komanda.

Organizācijas spējības noteikšanas pēdējā posmā ir izmantojams *FOIL* algoritms un tā darbība ir pārbaudīta 4.2.5 nodaļā. Šajā gadījumā, pārbaudei tika izmantoti testa apmācības dati. Patieso apmācības datu izveidošana nav iekļauta promocijas darbā, jo darba apjoms un tam nepieciešamie soļi iziet ārpus promocijas darba robežām. Pārbaudot *FOIL* algoritma darbību, ir secināts, ka ir iespējams algoritmu izmantot spējības noteikšanas pēdējā posmā.

Promocijas darba ietvaros ir izstrādāta spējības noteikšanas rīka arhitektūra un uz to balstīts prototips, kurš izmanto interneta videi paredzētās tehnoloģijas *ASP.NET MVC* un *Microsoft Azure*. Rīka prototipu izmanto eksperti, lai noteiktu domēnu SII vērtības, kā arī, rīka prototipu izmanto organizācijā strādājošie darbinieki, lai atbildētu uz sagatavotajiem jautājumiem un sagatavotu spējības analīzes pārskatus. Pārskati tiek izmantoti uzlabojumu plāna izstrādē. Nākamais solis ir metodes ODSN un rīka prototipa pārbaude.

5. Metodes ODSN darbības pārbaude

Metodes darbība tiks pārbaudīta, veicot trīs reālu uzņēmumu, kuri veikuši transformāciju uz spējo metožu izmantošanu, analīzi.

5.1. Pārbaudes organizācijas

Šajā apakšnodaļā ir aprakstīti trīs uzņēmumi, kuru spējības līmeni ir izlemts pārbaudīt. Konkrētie trīs uzņēmumi ir izvēlēti, jo darba autors tajos ir strādājis tieši to transformācijas laikā. Gala rezultāts visos uzņēmumos atšķiras, kas padara šos uzņēmumus par labiem kandidātiem metodes pārbaudei.

5.1.1. Pārbaudes organizācija 1

Pirmā pārbaudes organizācija ir organizācija, kura vairāk par 10 gadiem ir nodarbojusies ar ārpakalpojumu sniegšanu vienam konkrētam uzņēmumam un konkrētā uzņēmuma projektu realizēšana aizņem ~90 % no kopējās izstrādes. Darba organizācija pirms spējās metodes ieviešanas, ir tradicionāla, kad no klienta tiek saņemta programmatūras prasību specifikācija konkrētam projektam, kurš konkrētajā laika posmā ir jārealizē. Projekti parasti ilgst vairākus mēnešus, garākais no kuriem ir 1 gads. Uzņēmums izstrādā projektējumu un vienojas ar pasūtītāju par dažādiem nosacījumiem. Uzņēmums izveido projekta komandu. Projekta vadītājs, iesaistot vadošo izstrādātāju, sadala projektu loģiskās daļās un veic konkrēto daļu novērtēšanu. Vienošanās starp uzņēmumiem ir noslēgta, un notiek izstrāde. Pēc noteikta laika sistēma tiek piegādāta klientam.

Problēmas parasti rodas ar piegādes laiku un kļūdu sakopšanu pēc piegādes. Bieži sākotnējie novērtējumi ir neprecīzi un izstrāde iekavējas. Projekta izstrādes laikā mainās prasības vai arī daļa no prasībām ir interpretētas nepareizi, kā rezultātā ar laiku pieaug pasūtītāja neapmierinātība ar piegādes laiku un kvalitāti.

Kādā no brīžiem klients piedāvā izstrādi veikt, izmantojot *Scrum* metodi. Dotās iniciatīvas vadīta organizācija izlemj sākt programmatūras izstrādi, izmantojot *Scrum* metodi. Organizācijai sākotnēji nav kompetences spējo metožu pielietošanā. Tieši šī iemesla dēļ projekta vadība sāk veikt apmācību. Negatīvais aspekts šādai pieejai ir tāds, ka projekta vadība ir vienīgā, kurai ir izpratne par spējām metodēm. Izstrādes komandas interpretācija bieži atšķiras no projekta vadības izpratnes par to, kas un kā jādara. Nākamajās apakšnodaļās ir apkopota informācija ar intervēšanas rezultātiem.

5.1.2. Pārbaudes organizācija 2

Otrā pārbaudes organizācija ir organizācija, kura agrāk nav nodarbojusies ar programmatūras izstrādi. Organizācija visu nepieciešamo programmatūru ir iepirkusi no citām programmizstrādes kompānijām. Ņemot vērā slikto pieredzi ar piegādātājiem, organizācija ir izlēmusi veidot savu programmizstrādes departamentu. Lēmumam ir gan finansiāli, gan kvalitāti ietekmējoši faktori.

Kā piemērotākais izstrādes modelis ir izvēlēts spējais, izmantojot *Scrum* metodi. Pēc programmizstrādes nodaļas izveidošanas sākas intensīvas darbinieku apmācības, jo zināšanu līmenis par spējām metodēm visiem nav vienāds un nav pietiekams veiksmīgai *Scrum* ieviešanai.

Pēc 4 mēnešiem vēlamā *Scrum* procesa kvalitāte vēl nav sasniegta, un organizācija pieaicina uz vairākām nedēļām spējo metožu ekspertus. Eksperti reģistrē savus novērojumus, lai sniegtu konkrētus ieteikumus procesa uzlabošanai, kā arī papildu tiek veikti komandas gara saliedēšanas pasākumi.

Papildu iepriekš minētajiem pasākumiem, organizācija iegulda līdzekļus darbinieku kvalifikācijas celšanai. Kvalifikācijas celšanā ir iekļauti sertifikācijas pasākumi, vieslekcijas, mācīšanās portālu abonementi un iekšējās lekcijas, kuras sagatavo kāds no darbiniekiem. Nākamajās apakšnodaļās ir apkopota informācija ar intervēšanas rezultātiem.

5.1.3. Pārbaudes organizācija 3

Trešā pārbaudes organizācija ir organizācija, kura vairāk nekā 20 gadus nodarbojas ar programmizstrādi, izmantojot klasisko pieeju. Izstrāde notiek, izmantojot klasiskās pieejas procesus un artefaktus. Organizācija kādu laiku vēlas savam portfelim pievienot spējās izstrādes kompetenci, un šāda iespēja parādās, kad nepieciešams apjomīgu projektu realizēt īsā laikā, bet specifikācija ir pārāk virspusēja un nevar tikt izmantota normālai izstrādes organizēšanai. Ir daudz nianšu, kuras ir jāprecizē, lai varētu ķerties pie darba. Klasisko pieeju nevar izmantot, jo precīzas dokumentācijas izstrādei nav nepieciešamā laika.

Ņemot vērā, ka organizācijai, šobrīd, nav nepieciešamās spējās izstrādes kompetences, organizācija ir izlēmusi piesaistīt trešo pusi spējā projekta vadībai. Trešā puse iesaistās tikai projekta vadīšanai un organizēšanai, izstrādi veic organizācijā jau

esošie izstrādātāji. Nenotiek visas organizācijas pārveidošana uz organizāciju, kura pilnībā izmanto spējās metodes. Spējā veidā tiks izstrādāts viens konkrēts projekts.

Projekta realizēšanai tiek izveidota komanda no esošajiem izstrādātājiem, projekta vadītāja (no organizācijas bez pieredzes spējo projektu vadīšanā) un projekta vadītāja (no trešās puses ar pieredzi spējo projektu vadīšanā, kura ir lielāka par 15 gadiem). Izstrādātājiem tiek nolasīta prezentācija par *Scrum*, pēc kuras sākās intensīva sistēmas izstrāde. Nākamajās apakšnodaļās ir apkopota informācija ar intervēšanas rezultātiem.

5.2. Pārbaudes rezultāti

Trīs uzņēmumu spējības līmeņa pārbaude ir veikta, iegūstot atbildes uz visu jautājumu kopu *Q*. Gadījumos, ja nebija pieejamas atbildes uz visiem jautājumiem, tad spējības noteikšana notieka, izmantojot tikai sniegtās atbildes. Spējības noteikšanas pārskatā procentuāli tiek norādīts, uz cik jautājumiem ir sniegtas atbildes. Lai iegūtu atbildes uz jautājumiem, ir izmantots promocijas darba ietvaros izstrādātais programmatūras prototips. Apakšnodaļā ir apkopoti dati par pirmajiem diviem DAA koka līmeņiem (sk. 5.1. tabula). Informācija ar visiem DAA koka līmeņiem ir pieejama 6. pielikumā.

5.1. tabula

Pārbaudes organizāciju spējības līmeņa rezultāti

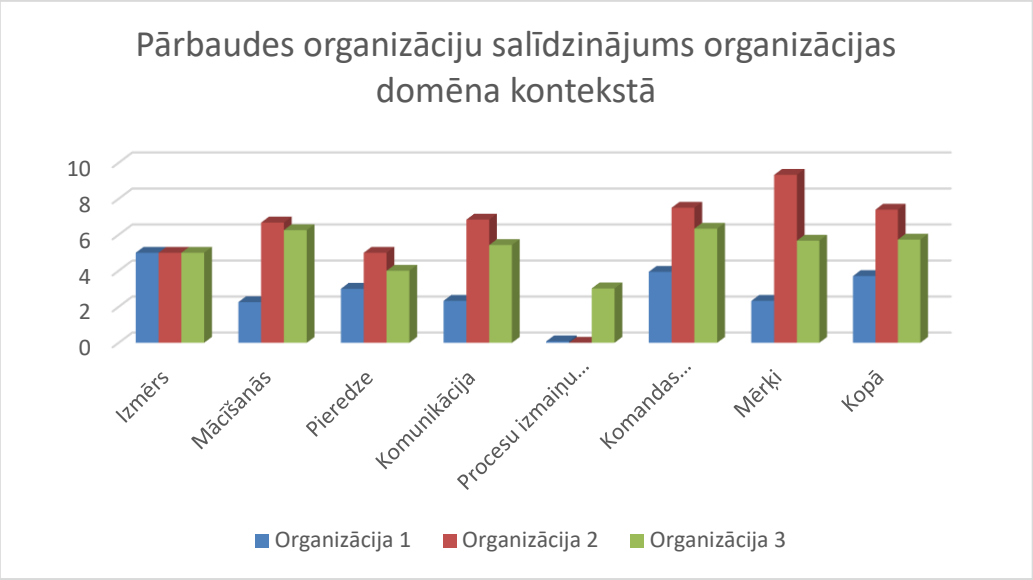
			Organizācija 1	Organizācija 2	Organizācija 3
Kods	**Nosaukums**	**SII**	**DVV**	**DVV**	**DVV**
1	Org. domēns	7	3.71	7.40	5.73
1.1	Izmērs	6	5.00	5.00	5.00
1.2	Mācīšanās	8	2.26	6.68	6.26
1.3	Pieredze	8	3.00	5.00	4.00
1.4	Komunikācija	8	2.33	6.84	5.43
1.5	Procesu izmaiņu vadība	5	0.08	0.00	3.02
1.6	Komandas veidošanas process	8	3.94	7.50	6.34
1.7	Mērķi	7	2.33	9.33	5.67
2	Izstrādes domēns	8	3.84	6.69	3.99
2.1	Komunikācija	9	4.70	7.28	5.54
2.2	Zināšanu pārvaldība	6	1.06	8.68	7.85
2.3	Mācīšanās	8	1.93	7.45	5.12
2.4	Pieredze	7	6.08	6.08	6.08
2.5	Vide	7	3.00	5.00	3.00
2.6	Prakšu izmaiņu pārvaldība	5	1.56	4.98	5.08

			Organizācija 1	Organizācija 2	Organizācija 3
Kods	**Nosaukums**	**SII**	**DVV**	**DVV**	**DVV**
2.7	Sadarbība	7	6.00	6.00	4.00
2.8	Motivācija	8	4.60	8.00	3.73
2.9	Komanda	8	4.33	6.24	4.71
3	Kval. domēns	8	3.67	8.00	5.67
3.1	Vadlīnijas	6	0.91	5.75	4.93
3.2	Vienošanās	7	0.23	5.48	5.25
3.3	Standarti	6	0.22	5.47	4.05
3.4	Rīki	7	3.03	5.48	4.43
3.5	Prakšu izmaiņu pārvaldība	5	1.61	3.21	4.04
3.6	Lomas	7	3.51	5.81	3.95
3.7	Fokusēšanās uz izcilību	8	3.67	8.00	5.67
4	Proc. domēns	8	5.28	8.35	7.08
4.1	Laikā ierobežotas aktivitātes	8	4.53	7.33	5.64
4.2	Lomas	8	5.77	8.69	6.76
4.3	Artefakti	7	4.71	7.96	7.44
5	Vērtību domēns	7	5.58	8.19	6.30
5.1	Portfeļa pārvaldība	6	4.17	7.62	7.59
5.2	Produkta pārvaldība	5	7.42	7.38	7.19
5.3	Laidienu pārvaldība	7	4.27	8.77	5.67
5.4	Metriku izmaiņu pārvaldība	7	1.00	4.29	1.25
5.5	Vērtības piegāde	8	4.38	6.16	7.26
6	Projekts	7	5.60	5.94	6.13
6.1	Projekta veids	7	8.00	8.00	8.00
6.2	Iesaistīto personu skaits	7	7.00	7.00	7.00
6.3	Pieredze ar projektu	8	5.00	7.00	6.00
6.4	Izmērs	7	5.00	5.00	5.00
6.5	Garums	6	6.00	6.00	6.00
6.6	Sarežģītība	7	5.00	5.00	4.00
6.7	Patiesais garums	6	3.00	3.00	7.00

5.3. Secinājumi

Apskatot informāciju 6 domēnu kontekstā, ir ievērojams, ka Organizācijai 2 kopvērtējumā ir augstākais spējības līmenis (sk. 5.1. tabula). Izmantojot izstrādāto prototipu, ir iespējams detalizētāk apskatīt katru domēnu un apakšdomēnu, lai precīzāk noskaidrotu, kādēļ katrs domēns ir ieguvis tam aprēķināto vērtību. Šāda pieeja ļauj precīzāk noteikt problēmas apgabalu un atrast tam atbilstošu risinājumu, izstrādājot uzlabojumu plānu.

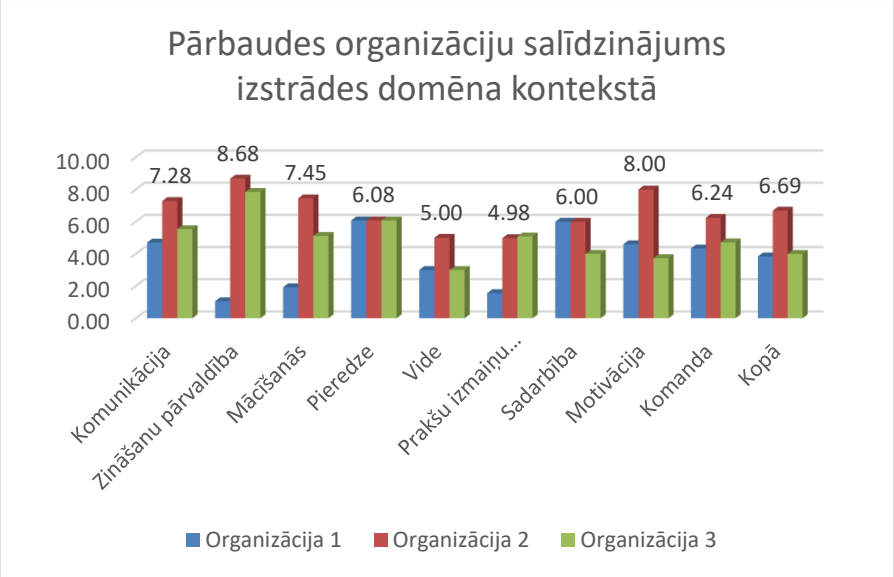
Organizācijas domēna līmeņa dati norāda, ka Organizācijas 1 spējības līmenis organizācijas domēna kontekstā ir zemāks nekā Organizācijai 2 un Organizācijai 3, respektīvi 3.71, 7.40 un 5.73. Organizācijas 1 lielākie klupšanas akmeņi organizācijas domēnā ir nepietiekama uzmanība apakšdomēniem Mācīšanās, Komunikācija un Mērķi (sk. 5.1. att.).



5.1. att. Pārbaudes organizāciju salīdzinājums, organizācijas domēna kontekstā.

Organizācijas domēna kontekstā augstāko rezultātu uzrāda Organizācija 2. Organizācija 2 ir viena no tām pārbaudītajām organizācijām, kurā vadības līmenī tika izlemts, ka visi projekti ir jāizstrādā izmantojot spējās metodes, kā arī papildu tika nolīgti SME, lai veiktu darbinieku apmācību darbam ar spējām metodēm.

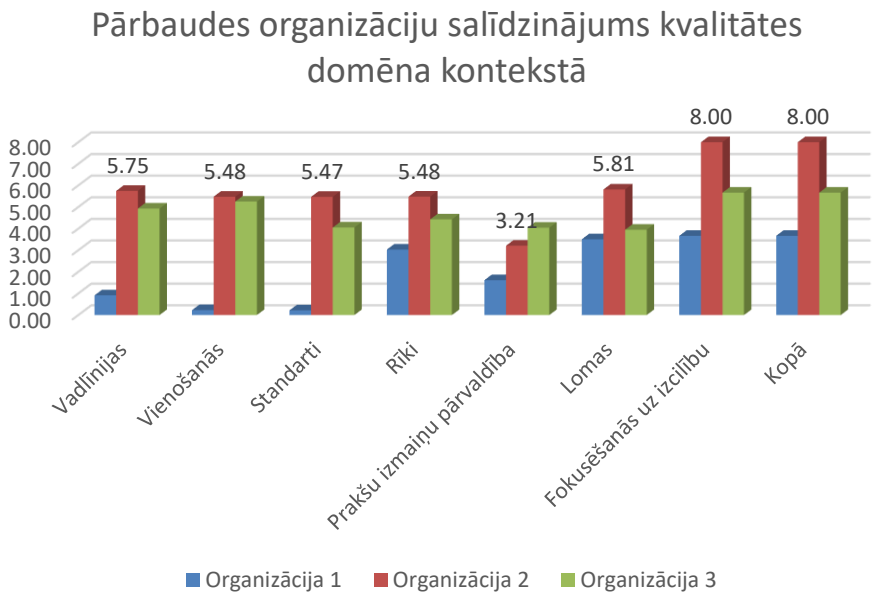
Izstrādes domēnā augstākie rādītāji ir Organizācijai 2, kuras rādītāji ir ievērojami augstāki nekā Organizācijai 1 un Organizācijai 3. Lielākā atšķirība ir atrodama Mācīšanās, Vides, Motivācijas un Komandas apakšdomēnos (sk. 5.2. att.).



5.2. att. Pārbaudes organizāciju salīdzinājums, izstrādes domēna kontekstā.

Viens no ievērības cienīgiem novērojumiem ir tas, ka visu trīs organizāciju pieredze spējo metožu izmantošanā un tehniskajās zināšanās ir līdzvērtīga, vērtēšanas brīdī.

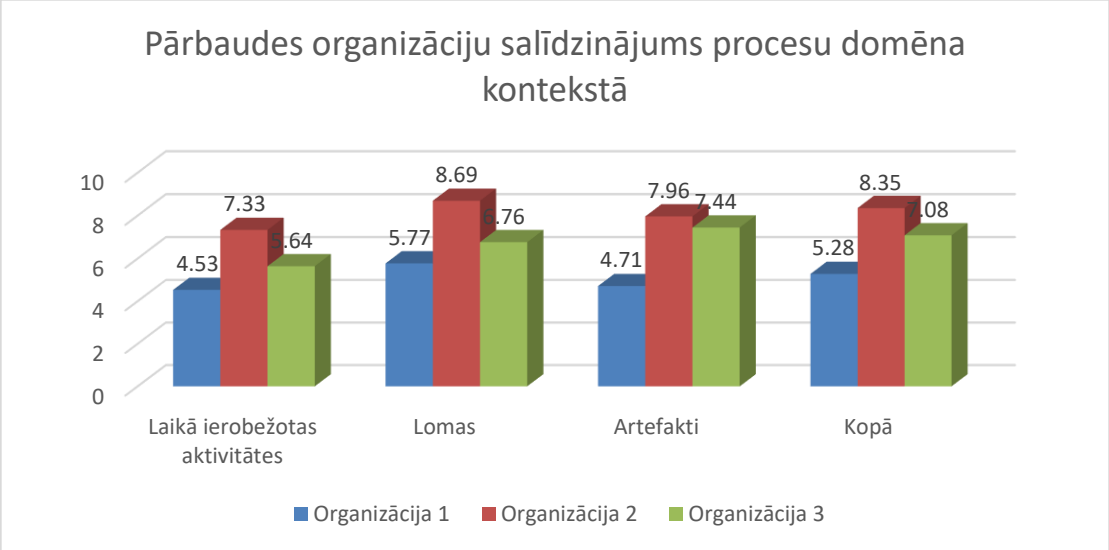
Kvalitātes domēna kontekstā Organizācijai 2 un Organizācijai 3 ir augstāki spējības rādītāji nekā Organizācijai 1. Vadlīnijās, Vienošanās, Standartu un Rīku apakšdomēnos Organizācijai 2 un Organizācijai 3 ir līdzvērtīgi rādītāji un labākus rādītājus Organizācija 2 uzrāda Lomu (Kompetenču) un Fokusēšanās uz izcilību apakšdomēnos (sk. 5.3. att.).



5.3. att. Pārbaudes organizāciju salīdzinājums, kvalitātes domēna kontekstā.

Kvalitātes domēna kontekstā lielu lomu ir nospēlējos Fokusēšanās uz izcilību apakšdomēns, jo domēnā aprakstītie atribūti ir vieni no spējo metožu stūrakmeņiem [7]. Viens no Organizācijas 2 pamatmērķiem, katru gadu, ne tikai programmatūras izstrādē, bet arī citos darbības virzienos ir bijis Fokusēšanās uz izcilību.

Procesu domēna kontekstā Organizācija 2 uzrāda labākus spējības rādītājus nekā Organizācija 1 un Organizācija 3, 5.28, 8.35 un 7.08 respektīvi (sk. 5.4. att.).

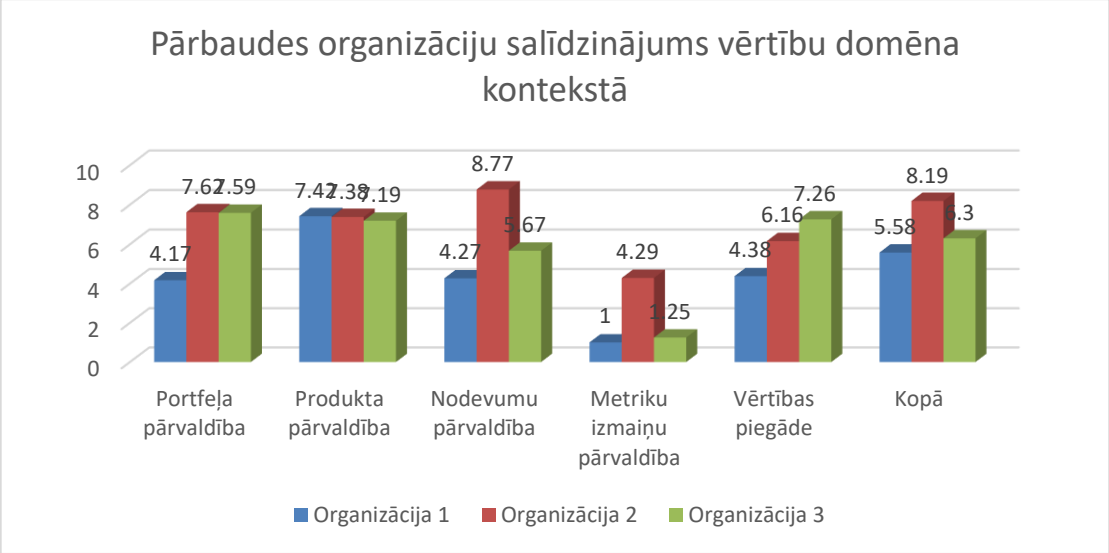


5.4. att. Pārbaudes organizāciju salīdzinājums, procesu domēna kontekstā.

Ievērojamākā starpība ir apakšdomēnos Laikā ierobežotas aktivitātes un Lomas, ko ir iespējams pamatot ar faktu, ka Organizācija 2 transformācijas sākumposmā pieaicināja SME, lai precīzāk izprastu spējo metožu procesus, un kādas darbības ir jāveic konkrētās lomas veicējam. Organizācija 3 šos rādītājus ir uzlabojusi, jo projektā ir iesaistīts ārējs *ScrumMaster* ar vairāk nekā 15 gadu pieredzi. Gadījumā, ja šāds eksperts netiktu pieaicināts, rezultāts būtu ievērojamāki zemāks, jo Organizācijai 3 nav iepriekšējas pieredzes spējo metožu izmantošanā. Organizācijas 1 spējības rezultāti procesu domēnu kontekstā ir zemāki nekā pārējām organizācijām un, apskatot intervēšanas rezultātus, ir ievērojams, ka problēma slēpjas organizācijas nepietiekamā zināšanu līmenī par spējām metodēm un to procesiem, kā arī nevēlēšanās pieaicināt SME darbinieku izglītošanai un spējā procesa uzlabošanai.

Vērtību domēnu kontekstā kopējais spējības līmenis Organizācijai 2 ir augstāks nekā Organizācijai 1 un Organizācijai 3 (sk. 5.5. att.). Šajā gadījumā, Organizācijas 2 spējības līmenis nav augstāks visos vērtējamos apakšdomēnos, piemēram, Vērtības

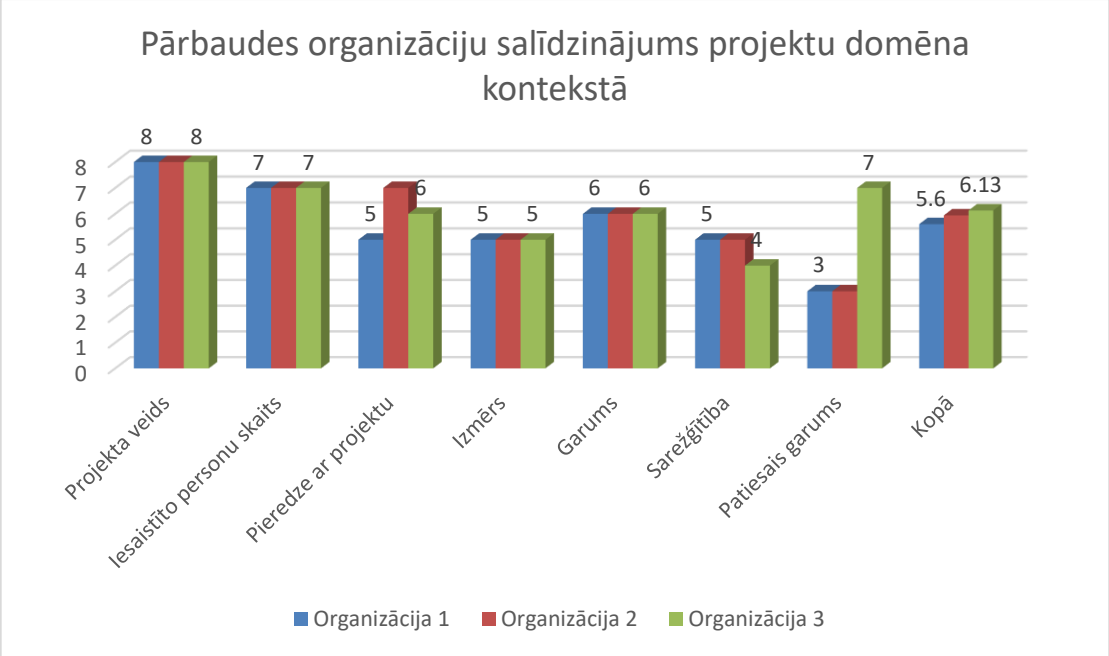
piegādes apakšdomēnā, kurā Organizācijas 3 spējības līmenis ir augstāks par Organizācijas 2 spējības līmeni.



5.5. att. Pārbaudes organizāciju salīdzinājums, vērtību domēna kontekstā.

Portfeļa pārvaldība Organizācijā 2 un Organizācijā 3 ir līdzvērtīgi augstā līmenī un ir izskaidrojama ar konkrētu štata vietu uzņēmumā, kura ir atbildīga par atbilstošā portfeļa pārvaldību. Organizācijā 1 šādas konkrētas štata vietas nav, tādēļ ir grūtāk iegūt informāciju par konkrēto portfeli, vai tā speciāli ir jāsagatavo. Produkta pārvaldība ir līdzvērtīgā līmenī visās trijās organizācijās. Laidienu pārvaldības apakšdomēnā, augstākais vērtējums ir Organizācijai 2, un tas ir saistīts ar ļoti precīzi izstrādātu laidienu plānu vairākiem gadiem uz priekšu dažādu produktu griezumā. Organizācijai 1 un Organizācijai 3 šāds plāns ir izstrādāts tikai daļēji un nav pieejams visiem produktiem. Vērtības piegādes apakšdomēnā augstākais rādītājs ir Organizācijai 3, jo organizācija vērtējamā projekta kontekstā veica ļoti regulāras un automatizētas programmatūras papildinājumu piegādes. Organizācijā 1 un Organizācijā 2 šādas piegādes netika veiktas ar nepieciešamo regularitāti.

Projekta domēna kontekstā visu pārbaudīto organizāciju spējības līmenis ir līdzvērtīgs un nav novērojama vērā ņemama starpība (sk. 5.6. att.).



5.6. att. Pārbaudes organizāciju salīdzinājums, projektu domēna kontekstā.

Projekts, pie kura ir strādāts, katrā no organizācijām, ir izrādījies ļoti līdzīgs pēc atribūtu vērtībām, tādēļ Projekta domēna līmenī spējības vērtība ievērojami neatšķiras visos trijos projektos (5.6, 5.94, 6.13). Lielākā starpība ir Projekta patiesais garums apakšdomēnā, kur Organizācijas 3 izstrādājamais projekts ir pilnīgi jauns un tādēļ neietver spējību samazinošos faktorus, kad nepieciešams strādāt ar vecu projektu, kuram līdzi nāk dažādi apgrūtinājumi.

Pēc DAA datu salīdzināšanas organizācijām un komandām ir iespēja izdarīt secinājumus par problēmu apgabaliem un sākt veidot uzlabojumu plānu turpmākā darba organizēšanai. Dati norāda, kurā domēnā un apakšdomēnā ir novirzes no ekspertu vērtējumiem, kas dod iespēju organizācijām un komandām izvēlēties kādu no pētītājām praksēm, kuras aprakstītas 2.2 nodaļā. Par papildu atbalstu ir izmantojama 3.104. tabula, kurā ir norādīts, kādas prakses ietekmē konkrētos OSM modeļa elementus. Organizācijas un komandas var patstāvīgi izlemt, kādas prakses no 3.104. tabula tiks izmēģinātas nākamajā uzlabojumu plānā.

Apskatot un pētot rezultātus, autors ir konstatējis, ka nepieciešams turpināt pētījumus un veikt papildu pilnveidojumus OSM modelī, padziļinot detalizācijas līmeni. Autora skatījumā nepieciešamība pēc šādām modifikācijām var rasties, pētot citas organizācijas un identificējot procesus, kurus nav iespējams izprast, izmantojot esošo modeli.

Galvenie rezultāti un secinājumi

Promocijas darbā tika izvirzīts mērķis izstrādāt metodi un rīka koncepciju organizācijas spējības noteikšanai, kā arī izstrādāt spējības noteikšanas rīka arhitektūras risinājumu un rīka prototipu. Papildu mērķis bija veikt izstrādātās metodes un rīka prototipa aprobāciju vairākās organizācijās to spējības līmeņa noteikšanai. Izvirzītā mērķa sasniegšanai tika realizēti šādi uzdevumi:

- izpētīta spējo metožu vēsture, to izmantošanas priekšrocības un riski, kā arī to nozīmīgākās raksturiezīmes;
- izstrādāta spējās metodēs izmantoto terminu karte un definēts AI spējām metodēm, praksēm un atslēgas vārdiem;
- izstrādāta organizācijas spējības definīcija un OSM;
- definēts SII un noteikta tā vērtība OSM elementiem;
- izstrādāta organizācijas spējības noteikšanas metode ODSN un izveidots jautājumu ģenerēšanas algoritms;
- izstrādāts spējības noteikšanas rīka konceptuālais modelis un arhitektūra;
- veikta piedāvātās arhitektūras realizācija;
- veikta izstrādātā prototipa eksperimentālā pārbaude, vērtējot trīs dažādu uzņēmumu spējības līmeni.

Visi promocijas darbā izvirzītie mērķi un uzdevumi ir sasniegti. Promocijas darbā ir sasniegti šādi galvenie teorētiskie un praktiskie rezultāti:

- ir izstrādāta AI definīcija, kura nosaka, cik liela ir lietotāju interese par konkrēto spējo metodi, un ir noteikta AI vērtība spējām metodēm, praksēm un atslēgas vārdiem periodā no 2008. līdz 2012. gadam;
- ir izstrādāta organizācijas spējības līmeņa definīcija un noteikts organizācijas spējības līmenis trijām pārbaudes organizācijām;
- organizācijas spējības līmeņa noteikšanai ir izstrādāts OSM modelis, kurš apraksta organizāciju, izmantojot sešus domēnus un apakšdomēnu atribūtu sistēmu;
- ir izstrādāta SII definīcija, kura nosaka katra OSM modeļa elementa ietekmi uz organizācijas spējību. Izmantojot ekspertu tīklu, ir noteiktas SII vērtības visiem OSM elementiem;

- izmantojot OSM modeli, ir izstrādāta organizācijas spējības noteikšanas metode ODSN. Darbā ir definēts ODSN metodes process un tā posmi;
- darbā ir izstrādāts jautājumu ģenerēšanas algoritms, kurš nodrošina darbinieku intervēšanai nepieciešamos jautājumus no kopējās jautājumu kopas;
- lai pārbaudītu metodes ODSN darbību, ir izstrādāts sistēmas konceptuālais modelis un arhitektūra. Pamatojoties uz izstrādāto arhitektūru, ir izstrādāts organizācijas spējības līmeņa noteikšanas sistēmas prototips, kurš tiek izmantots, lai pārbaudītu trīs dažādu organizāciju spējības līmeni.

Pētījuma rezultātā tika pierādītas izvirzītās tēzes:

- organizāciju neziņa par spējām metodēm rada problēmas spējo metožu izmantošanā – veicot pārbaudi trijās organizācijās, katra no kurām izvēlējās dažādas pieejas spējo metožu izmantošanā, izdevās pierādīt, ka sliktākie spējības rezultāti ir organizācijai, kura nav apmācījusi savus darbiniekus un nav pieaicinājusi ekspertus, lai uzlabotu izstrādes procesu;
- izmantojot metodes un rīkus ir iespējams noteikt organizācijas spējību – izmantojot izstrādāto metodi ODSN un izstrādāto rīka prototipu, ir noteikti trīs organizāciju domēnu spējības līmeņi;
- organizācijas spējības noteikšana palīdz izstrādāt atbilstošu uzlabojumu plānu organizācijas spējības uzlabošanai – izmantojot noteiktos domēnu spējības līmeņus ir iespējams izstrādāt uzlabojumu plānu un, iekļaujot tajā atbilstošas spējās prakses, ir iespējams uzlabot kopējo spējības līmeni.

Pētījumu rezultātā ir izdarīti šādi secinājumi:

- programmizstrādes uzņēmumu interese par spējām metodēm pieaug, jo spējās metodes var atrisināt daļu problēmu, ar kurām uzņēmumi sastopas ikdienā. Spējās metodes ir stingri orientētas uz pasūtītāja prasību apmierināšanu iespējami īsākā laikā un ar iespēju šīs prasības mainīt arī vēlu izstrādes procesā;
- neskatoties uz to, ka spējām metodēm ir vērā ņemamas priekšrocības salīdzinājumā ar tradicionālām izstrādes metodēm, pastāv arī zināmi riski to

izmantošanā. Uzņēmumiem un komandām ir iespēja mazināt šos riskus, izmantojot atbilstošas spējās prakses;

- ir pieejams plašs literatūras klāsts par spējām metodēm un praksēm, šo literatūras klāstu gan jāiemācās pareizi izmantot, jo indivīdiem, kas ar šo metodi vēl nav strādājuši, ir viegli to nepareizi interpretēt un apmaldīties piedāvātajā terminoloģijā un izmantotajos procesos;
- par pamata metodi pēdējā laikā tiek izmantota metode *Scrum* un to apstiprina promocijas darbā definētais un noteiktais AI, gan arī informācija no citiem avotiem. Metode *Scrum* apvienojumā ar spējām praksēm, pareizi to izmantojot, sniedz iespēju sasniegt vēlamo rezultātu. Dažādi uzņēmumi un komandas var izmantot dažādas spējās prakses nepieciešamā rezultāta sasniegšanai. Visām darbā minētajām praksēm ir savas priekšrocības un trūkumi. Pirms izvēlēties kādu no praksēm uzņēmumiem ir jāizvērtē to priekšrocības un trūkumi;
- definētā SII vērtība, kopā ar izstrādāto OSM modeli un spējības noteikšanas metodi ODSN palīdz noteikt organizācijas spējības līmeni izveidoto sešus domēnu kontekstā;
- SII vērtības noteikšanai ir izmantota metode DELPHI un spējo metožu ekspertu tīkls, bet ir jārēķinās ar to, ka eksperti ir ļoti grūti pieejami un nevienmēr ir pretimnākoši, kas ļoti sarežģī SII vērtību noteikšanu;
- pilnvērtīgai organizācijas spējības līmeņa noteikšanai ir izmantots *FOIL* algoritms ar pārbaudes apmācības datiem. Lai iegūtu pilnvērtīgus apmācības datus, ir jārēķinās ar to, ka apmācības datu iegūšana ir sarežģīts un darbietilpīgs process, ko, analizējot tikai Latvijā strādājošos uzņēmumus nav iespējams pilnībā realizēt. Faktiski darbs ir tik apjomīgs, ka tā organizēšanai un aprakstīšanai ir nepieciešams atsevišķs pētījums, kurš nav iekļauts promocijas darbā;
- domēna līmenī iegūtie spējības rādītāji ir pietiekami, lai izstrādātu uzlabojumu plānu un apmācības datu trūkums netraucē noteikt problēmapgabalus;
- izmantojot izstrādāto spējības līmeņa noteikšanas prototipu, ir noteikts domēnu spējības līmenis trīs pārbaudāmajām organizācijām;

- darbā aprakstītais OSM modelis par pamatu izmanto metodi *Scrum*. Veicot izmaiņas modelī, ODSN metodi ir iespējams izmantot ar citām spējām metodēm.

Pētījuma rezultātus var izmantot uzņēmumi, kuri vēlas pāriet uz spējo metožu izmantošanu vai to jau ir izdarījuši un atrodas transformācijas procesā. Izstrādātā metode un rīks palīdzēs tām noteikt problēmu apgabalus un izveidot atbilstošus uzlabojumu plānus. Izveidotā terminoloģijas karte, kā arī metožu un prakšu sadalījums iesācējiem palīdzēs skaidrāk uztvert pieejamo literatūru un veiksmīgāk apgūt spējo metodoloģiju.

Turpmāko pētījumu iespējamie virzieni:

- izpētīt vairāk uzņēmumus, kuri ir ieviesuši spējās metodes, lai varētu izveidot pilnvērtīgākus apmācības datus *FOIL* vai līdzīga algoritma izmantošanai;
- izpētīt iespēju pilnveidot izstrādāto prototipu, lai būtu iespējams spējības noteikšanas rīku izmantot kā pakalpojumu organizācijas spējības noteikšanai;
- paplašinot ekspertu tīklu, pilnveidot OSM modeli.

Bibliogrāfija

[1] Agile Alliance Conference 2008. Agile Alliance. Pieejams: http://agile2008.agilealliance.org/ (Apmeklēts: 17.05.2012).

[2] Agile Alliance Conference 2009. Agile Alliance. Pieejams: http://agile2009.agilealliance.org/ (Apmeklēts: 19.05.2012).

[3] Agile Alliance Conference 2010. Agile Alliance. Pieejams: http://agile2010.agilealliance.org/ (Apmeklēts: 25.05.2012).

[4] Agile Alliance Conference 2011. Agile Alliance. Pieejams: http://agile2011.agilealliance.org/ (Apmeklēts: 30.05.2012).

[5] Agile Alliance Conference 2012. Agile Alliance. Pieejams: http://agile2012.agilealliance.org/ (Apmeklēts: 10.06.2012).

[6] Agile Software Development. Wikipedia, free encyclopedia. Pieejams: http://en.wikipedia.org/wiki/Agile_software_development (Apmeklēts: 11.06.2012).

[7] Manifesto for Agile Software Development. Agile Alliance. Pieejams: http://agilemanifesto.org/ (Apmeklēts: 11.06.2012).

[8] D. Moran, Top 10 Reasons to Use Agile Development, Pieejams: http://www.devx.com/enterprise/Article/44619/0/page/1 (Apmeklēts: 15.06.2012).

[9] R. Levine and M. McDonough. Why Do Agile Projects Fail?, Pieejams: http://www.brighthub.com/office/project-management/articles/55778.aspx (Apmeklēts: 20.06.2012).

[10] R. Coffin and D. Lane. A practical Guide to Seven Agile Methodologies, Part 1, 2006. Pieejams: http://www.devx.com/architect/Article/32761 (Apmeklēts: 26.06.2012).

[11] R. Coffin and D. Lane. A practical Guide to Seven Agile Methodologies, Part 2, 2006. Pieejams: http://www.devx.com/architect/Article/32836 (Apmeklēts: 20.06.2012).

[12] L. Williams. A Survey of Agile Development Methodologies, 2007. Pieejams: http://agile.csc.ncsu.edu/SEMaterials/AgileMethods.pdf (Apmeklēts: 22.06.2012).

[13] Agile Courses. University Of Oxford, 2011. Pieejams: http://www.softeng.ox.ac.uk/subjects/AGM.html (Apmeklēts: 15.07.2012).

[14] Agile Delivery Methods. Agilier. Pieejams: http://www.agilier.com/agile-business-change-techniques/agile-delivery-methods.html (Apmeklēts: 11.08.2012).

[15] Agile Software Development. Enotes.com, 2011. Pieejams: http://www.enotes.com/topic/Agile_software_development#Agile_methods (Apmeklēts: 11.06.2013).

[16] Agile Software Development Site. Seapine Software. Pieejams: http://www.devagile.com/ (Apmeklēts: 11.06.2013).

[17] Extreme Programming. Wikipedia The Free Encyclopedia. Pieejams: http://en.wikipedia.org/wiki/Extreme_Programming (Apmeklēts: 11.06.2013).

[18] G. Lenz and T. Moeller, .NET-A Complete Development Cycle. Addison-Wesley Professional, ISBN-10: 0321168828, ISBN-13: 978-0321168825, 2003.

[19] K. Beck and C. Andres, Extreme Programming Explained: Embrace Change (2nd Edition), ISBN-10: 0321278658, ISBN-13: 978-0321278654, 2004.

[20] S. Warden, Extreme Programming Pocket Guide (1st Edition). O'Reilly Media, ISBN-10: 0596004850, ISBN-13: 978-0596004859, 2003.

[21] M. Cohn, Succeeding with Agile: Software Development Using Scrum. Addison-Wesley Professional, ISBN-10: 0-321-57936-4, ISBN-13: 978-0-321-57936-2, 2009.

[22] Scrum (development). Wikipedia The Free encyclopedia. Pieejams: http://en.wikipedia.org/wiki/Scrum_(development) (Apmeklēts: 15.07.2013).

[23] K. Schwaber, Agile Project Management with Scrum. Microsoft Press, ISBN-10: 073561993X, ISBN-13: 978-0735619937, 2004.

[24] Feature Driven Development. Wikipedia The Free Encyclopedia. Pieejams: http://en.wikipedia.org/wiki/Feature_Driven_Development (Apmeklēts: 14.06.2013).

[25] S. R. Palmer and J. M. Felsing, A Practical Guide to Feature-Driven Development. Prentice Hall, ISBN: 0130676152 / 0-13-067615-2, 2002.

[26] FDD process model. Feature Driven Development. Pieejams: http://www.featuredrivendevelopment.com/files/FDD%20Process%20Model%20Diagram.pdf (Apmeklēts: 11.06.2012).

[27] A. Carmichael and D. Haywood, Better Software Faster. Prentice Hall, ISBN-10: 0130087521, ISBN-13: 978-0130087522, 2002.

[28] Test-driven development. Wikipedia The Free Encyclopedia. Pieejams: http://en.wikipedia.org/wiki/Test-driven_development (Apmeklēts: 19.03.2016).

[29] K. Beck, Test Driven Development: By Example. Addison-Wesley Longman, ISBN-10: 0321146530, ISBN-13: 978-0321146533, 2002.

[30] J. W. Newkirk and A. A. Vorontsov, Test-Driven Development in Microsoft .NET. Microsoft Press, ISBN-10: 0735619484, ISBN-13: 978-0735619487, 2004.

[31] E. Hakan and M.Torchiano, On the Effectiveness of Test-first Approach to Programming. Proceedings of the IEEE Transactions on Software Engineering, 31(1), January 2005.

[32] N. Llopis, Stepping Through the Looking Glass: Test-Driven Game Development (Part 1). Games from Within, 2007. Pieejams: http://www.gamesfromwithin.com/articles/0502/000073.html (Apmeklēts: 15.06.2012).

[33] M. M. Matthias and F. Padberg, About the Return on Investment of Test-Driven Development. Universität Karlsruhe, Germany. pp. 6, 2007.

Pieejams: http://www.ipd.uka.de/mitarbeiter/muellerm/publications/edser03.pdf (Apmeklēts: 05.06.2012).

[34] M. Fowler, K. Beck, J. Brant, et al., *Refactoring:* Improving the Design of Existing Code. Addison-Wesley Professional, ISBN-10: 0201485672, ISBN-13: 978-0201485677, 1999.

[35] Dependency inversion principle. Wikipedia The Free Encyclopedia. Pieejams: http://en.wikipedia.org/wiki/Dependency_inversion_principle (Apmeklēts: 22.06.2012).

[36] Unified Process. Wikipedia The Free Encyclopedia. Pieejams: http://en.wikipedia.org/wiki/Unified_Process (Apmeklēts: 11.01.2016).

[37] S. Kendall, The Unified Process Explained. Addison-Wesley Professional, ISBN-13: 978-0201742046, 2001.

[38] S. Sousa, The Advantages and Disadvantages / Best Practices of RUP Software Development. Pieejams: http://www.my-project-management-expert.com/the-advantages-and-disadvantages-of-rup-software-development.html (Apmeklēts: 04.02.2016).

[39] M. Broomé, Rational Unified Process - an overview. Pieejams: http://www.it.uu.se/edu/course/homepage/acsd/vt09/RUP-slides.pdf (Apmeklēts: 05.09.2013).

[40] Microsoft Solutions Framework. Wikipedia The Free Encyclopedia. Pieejams: http://en.wikipedia.org/wiki/Microsoft_Solutions_Framework (Apmeklēts: 17.10.2013).

[41] M. Keeton, Microsoft Solutions Framework (MSF): A Pocket Guide. Van Haren Publishing, ISBN-10: 9077212167, ISBN-13: 978-9077212165, 2004.

[42] M. S. V. Turner, Microsoft Solutions Framework Essentials: Building Successful Technology Solutions. Microsoft Press; 1 edition, ISBN-10: 0735623538, ISBN-13: 978-0735623538, 2006.

[43] S. Anderson, Collins English Dictionary – Complete and Unabridged. HarperCollins Publishers, New Yourk, NY, ISBN-10: 0007191537, ISBN-13: 978-0007191536, 2003.

[44] Kanban. Wikipedia The Free Encyclopedia. Pieejams: http://en.wikipedia.org/wiki/Kanban (Apmeklēts: 17.10.2013).

[45] Lean Software Development. Wikipedia The Free Encyclopedia. Pieejams: http://en.wikipedia.org/wiki/Lean_software_development (Apmeklēts: 17.10.2013).

[46] Specification by Example. Wikipedia The Free Encyclopedia. Pieejams: http://en.wikipedia.org/wiki/Specification_by_example (Apmeklēts: 17.10.2013).

[47] Kanban (development). Wikipedia The Free Encyclopedia. Pieejams: http://en.wikipedia.org/wiki/Kanban_(development) (Apmeklēts: 17.10.2013).

[48] Guide to Agile Practices. Agile Alliance. Pieejams: http://guide.agilealliance.org/ (Apmeklēts: 19.03.2016).

[49] Methodology. Business Dictionary. Pieejams: http://www.businessdictionary.com/definition/methodology.html (Apmeklēts: 14.07.2013).

[50] Method. The Free Dictionary. Pieejams: http://www.thefreedictionary.com/method (Apmeklēts: 14.07.2013).

[51] The American Heritage Dictionary of the English Language. Houghton Mifflin Harcourt, Boston, MA, ISBN-10: 0395825172, ISBN-13: 978-0395825174, 2000

[52] Methodology. Wikipedia The Free Encyclopedia. Pieejams: http://en.wikipedia.org/wiki/Methodology (Apmeklēts: 14.07.2013).

[53] Atdd. Guide to Agile Practices. Agile Alliance. Pieejams: http://guide.agilealliance.org/guide/atdd.html (Apmeklēts: 14.07.2013).

[54] Acceptance Testing. Guide to Agile Practices. Agile Alliance. Pieejams: http://guide.agilealliance.org/guide/acceptance.html (Apmeklēts: 14.07.2013).

[55] Automated Build. Guide to Agile Practices. Agile Alliance. Pieejams: http://guide.agilealliance.org/guide/autobuild.html (Apmeklēts: 14.07.2013).

[56] A. Kolawa, Automated Defect Prevention: Best Practices in Software Management. Wiley-IEEE Computer. Society Press, ISBN 0-470-04212-5, 2007.

[57] S. Butt and R. Badger, Benefits of Automated Testing. Pieejams: http://red-badger.com/blog/2013/02/01/benefits-of-automated-testing/ (Apmeklēts: 14.07.2013).

[58] S. Vaaraniemi, Benefits of Automated Testing. Codeproject. Pieejams: http://www.codeproject.com/Articles/5404/The-benefits-of-automated-unit-testing (Apmeklēts: 15.07.2013).

[59] BDD. Guide to Agile Practices. Agile Alliance. Pieejams: http://guide.agilealliance.org/guide/bdd.html (Apmeklēts: 15.07.2013).

[60] Pragmatic BDD for .NET. SpecFlow. Pieejams: http://www.specflow.org/ (Apmeklēts: 16.07.2013).

[61] Refactoring. Guide to Agile Practices. Agile Alliance. Pieejams: http://guide.agilealliance.org/guide/refactoring.html (Apmeklēts: 16.07.2013).

[62] Code review. Wikipedia The Free Encyclopedia. Pieejams: http://en.wikipedia.org/wiki/Code_review (Apmeklēts: 20.07.2013).

[63] Continuous Deployment. Guide to Agile Practices. Agile Alliance. Pieejams: http://guide.agilealliance.org/guide/cd.html (Apmeklēts: 20.07.2013).

[64] Cross-functional team. Wikipedia The Free Encyclopedia. Pieejams: http://en.wikipedia.org/wiki/Cross-functional_team (Apmeklēts: 21.07.2013).

[65] Cross-Functional teams. Reference for Business, Encylopedia of Business, 2nd edition. Pieejams: http://www.referenceforbusiness.com/small/Co-Di/Cross-Functional-Teams.html (Apmeklēts: 16.07.2013).

[66] Daily meeting. Guide to Agile Practices. Agile Alliance. Pieejams: http://guide.agilealliance.org/guide/daily.html (Apmeklēts: 16.07.2013).

[67] Definition of done. Guide to Agile Practices. Agile Alliance. Pieejams: http://guide.agilealliance.org/guide/definition-of-done.html (Apmeklēts: 16.07.2013).

[68] Emergent Design. Wikipedia The Free Encyclopedia. Pieejams: http://en.wikipedia.org/wiki/Emergent_Design (Apmeklēts: 16.07.2013).

[69] Estimation. Guide to Agile Practices. Agile Alliance. Pieejams: http://guide.agilealliance.org/guide/estimation.html (Apmeklēts: 17.07.2013).

[70] Facilitation. Guide to Agile Practices. Agile Alliance. Pieejams: http://guide.agilealliance.org/guide/facilitation.html (Apmeklēts: 17.07.2013).

[71] Invest. Guide to Agile Practices. Agile Alliance. Pieejams: http://guide.agilealliance.org/guide/invest.html (Apmeklēts: 17.07.2013).

[72] Iteration Planning. Version One. Pieejams: http://www.versionone.com/Agile101/Agile-Development-Iteration-Planning/ (Apmeklēts: 22.07.2013).

[73] Kanban board. Guide to Agile Practices. Agile Alliance. Pieejams: http://guide.agilealliance.org/guide/kanban.html (Apmeklēts: 22.07.2013).

[74] T. Javdani, H. Zulzalil, A. Ghani, On the Current Measurement Practices in Agile Software Development. University Putra, Malaysia. Pieejams: http://arxiv.org/ftp/arxiv/papers/1301/1301.5964.pdf (Apmeklēts: 23.07.2013).

[75] Pair Programming. Guide to Agile Practices. Agile Alliance. Pieejams: http://guide.agilealliance.org/guide/pairing.html (Apmeklēts: 23.07.2013).

[76] Planning Poker. Guide to Agile Practices. Agile Alliance. Pieejams: http://guide.agilealliance.org/guide/poker.html (Apmeklēts: 25.07.2013).

[77] Story Mapping. Guide to Agile Practices. Agile Alliance. Pieejams: http://guide.agilealliance.org/guide/storymap.html (Apmeklēts: 25.07.2013).

[78] Sustainable Pace. Guide to Agile Practices. Agile Alliance. Pieejams: http://guide.agilealliance.org/guide/sustainable.html (Apmeklēts: 25.07.2013).

[79] Tdd. Guide to Agile Practices. Agile Alliance. Pieejams: http://guide.agilealliance.org/guide/tdd.html (Apmeklēts: 27.07.2013).

[80] Usability Testing. Guide to Agile Practices. Agile Alliance. Pieejams: http://guide.agilealliance.org/guide/usability.html (Apmeklēts: 28.07.2013).

[81] User Stories. Guide to Agile Practices. Agile Alliance. Pieejams: http://guide.agilealliance.org/guide/user-stories.html (Apmeklēts: 29.07.2013).

[82] Dynamic Environment. Artificial Intelligence Lab. Pieejams: http://ai.eecs.umich.edu/cogarch4/toc_defs/defs_env/defs_dyn_env.html (Apmeklēts: 30.07.2013).

[83] Dynamic Environment. Ask.com. Pieejams: http://www.ask.com/question/what-is-a-dynamic-environment-in-an-organization (Apmeklēts: 30.07.2013).

[84] D. Weyns and H. Van Dyke Parunak, F. Michel, Environments for Multi-Agent Systems II. Springer, ISBN-10: 3-540-32614-6, ISBN-13: 978-3-540-32614-4, 2005.

[85] K. S. Rubin, Essential Scrum: A Practical Guide to Most Popular Agile Process. Addision-Wesley Professional, 2012.

[86] J. Beaver, The Agile Team Handbook. CreateSpace Independent Publishing Platform, 2013.

[87] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addision-Wesley Professional, 2010.

[88] R. Pichler, Agile Product Management with Scrum: Creating Products that Customers Love. Addision-Wesley Professional, 2010.

[89] T.M. Mitchell, Machine Learning (10.4 un 10.5 nodaļas). Lpp. 283-291. 1997.

[90] A. Borisovs, Izdales materiāli mācību kursā „Intelektuālās datorsistēmas", 2012.

[91] S. Parshutin, G. Kuleshova, A. Barisov, Application of first order rules to reconstructing link damages in logistics net. 2005.

[92] A. Cockburn, Crystal Clear, A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects. Addision-Wesley Professional, ISBN-10: 0201699478, ISBN-13: 978-0201699470, 2004.

[93] Dynamic Systems Development Method. Wikidot. Pieejams: http://dsdmofagilemethodology.wikidot.com/ (Apmeklēts: 30.03.2014).

[94] M. Poppendieck and T. Poppendieck, Lean Software Development: An Agile Toolkit. Addison-Wesley Professional, ISBN 0-321-15078-3, 2003.

[95] European Commission, The new SME definition. Official Journal of the European Union L 124, May 2003.

[96] H. Rubin, A Metrics View of Software Engineering Performance Across Industries. IT Metrics Strategies V:9:3, September 1999.

[97] Communication. Oxford Dictionaries. Pieejams: http://www.oxforddictionaries.com/definition/english/communication (Apmeklēts: 15.03.2014).

[98] Collaboration. Oxford Dictionaries. Pieejams: http://www.oxforddictionaries.com/definition/english/collaboration (Apmeklēts: 15.03.2014).

[99] М. Саркисян, Теория прогнозирования и принятия решений. Высшая школа, 1977.

[100] Л.В. Ницецкий и Л.П. Новицкий, “Применение методов экспертного опроса для оценки качества диалоговых обучающих систем”. Методы и средства кибернетики в управлении учебным процессом высшей школы. Сборник научных трудов, Рига РПИ, 1986.

[101] N. Slocum, Participatory methods toolkit, A practitioner’s manual. King Baudouin Foundation, ISBN 90-5130-506-0, 2005.

[102] SurveyMonkey, How Much Time are Respondents Willing to Spend on Your Survey? https://www.surveymonkey.com/blog/2011/02/14/survey_completion_times/ (Apmeklēts: 01.08.2015).

[103] Agile Software Development Definition. TechTarget. Pieejams: http://searchsoftwarequality.techtarget.com/definition/agile-software-development (Apmeklēts: 01.05.2016).

[104] What is Agile Software Development. Agile Alliance. Pieejams: https://www.agilealliance.org/agile101/what-is-agile/ (Apmeklēts: 01.05.2016).

[105] Agile Software Development definition. PC Magazine, Encyclopedia. Pieejams: http://www.pcmag.com/encyclopedia/term/37607/agile-software-development (Apmeklēts: 01.05.2016).

[106] What Is Agile? Agile Methodology. Pieejams: http://agilemethodology.org/ (Apmeklēts: 01.05.2016).

[107] Qumer., & Sellers, An evaluation of the degree of agility in six agile methods and its implacability for method engineering. Information and Software Technology, pp. 280-295, 2008.

[108] E. Derby and D. Larsen, Agile Retrospectives, Making Good Teams Great. Pragmatic Bookshelf, 2006.

[109] The CHAOS Manifesto. The Standish Group, 2012. Pieejams: http://blog.standishgroup.com/ (Apmeklēts: 01.10.2016).

[110] S. Ambler. Agile projects success rate. AmbySoft. Pieejams: http://www.ambysoft.com/surveys/ (Apmeklēts: 01.10.2016).

[111] Agile Adoption rates. Forester Research. Pieejams: http://www.forrester.com (Apmeklēts: 01.10.2016).

[112] C. Larman and V. Basili, Iterative and Incremental Development: A Brief History, 2003. Pieejams: http://www.craiglarman.com/wiki/downloads/misc/history-of-iterative-larman-and-basili-ieee-computer.pdf (Apmeklēts: 20.03.2016).

[113] E. A. Edmonds, A Process for the Development of Software for Nontechnical Users as an Adaptive System. General Systems, 1974.

[114] H. Takeuchi and I. Nonaka, New Product Development Game. Harvard Business Review 86116, p. 137-146, 1986.

[115] State of Agile Survey Results. VersionOne Inc, 2011. Pieejams: http://www.versionone.com/ (Apmeklēts: 01.17.2016).

[116] H. C. Maurya, A. Khatoon, N. Chaudhary, Metrics for Software Project Size Estimation. International Journal of Advanced Research in Computer Science and Software Engineering, Volume 5, Issue 1, January, 2015.

[117] Project Sizes. Method 123 Project Management Methodology. Pieejams: http://www.mpmm.com/project-sizes.php (Apmeklēts: 01.24.2016).

[118] Information Technologies, Project Classification. The University of New Mexico. Pieejams: https://it.unm.edu/projects/projectdefined.html (Apmeklēts: 01.24.2016).

[119] Ten Tips for Agile Leaders. The Pragmatic Bookshelf. Pieejams: https://pragprog.com/magazines/2012-02/ten-tips-for-agile-leaders (Apmeklēts: 01.31.2016).

[120] The Role of Leaders on a Self-Organizing Team. Mountain Goat Software. Pieejams: https://www.mountaingoatsoftware.com/blog/the-role-of-leaders-on-a-self-organizing-team (Apmeklēts: 01.31.2016).

[121] Agility Path. Scrum.org. Pieejams: https://www.scrum.org/Portals/0/Documents/Agility-Path/Agility-Path_Executive-Summary.pdf (Apmeklēts: 17.05.2012).

[122] Likert scale. Wikipedia The Free Encyclopedia. Pieejams: https://en.wikipedia.org/wiki/Likert_scale (Apmeklēts: 16.07.2016).

[123] J. Bledsoe. The Magic in a 0-to-10 Rating Scale. Primary Intelligence. Pieejams: https://en.wikipedia.org/wiki/Likert_scale (Apmeklēts: 16.07.2016).

Pielikumi

1. Pielikums

Spējo metožu Aktualitātes Indekss (AII)

Šajā pielikumā ir apvienota detalizēta informācija par spējo metožu AII.

P.1.1. tabula

Interneta vietnēs par populārākajām minētās metodes

Populārākās implementācijas	**Avots**
• Extreme Programming (XP) • Scrum • Lean Software Development • Feature Driven Development (FDD) • Agile Unified Process (UP) • Crystal	[10][11]
• Extreme Programming (XP) • Crystal • Scrum • Feature Driven Development (FDD)	[12]
• Extreme Programming (XP) • Scrum • Crystal • Feature Driven Development (FDD) • Lean Software Development • Dynamic System Development Method (DSDM)	[13]
• Dynamic System Development Method (DSDM) • Extreme Programming (XP) • Rational Unified Process (RUP) • Scrum	[14]
• Agile Modeling • Agile Unified Process (AUP) • Agile Data Method • Dynamic System Development Method (DSDM) • Essential Unified Process (EssUP) • Extreme Programming (XP) • Feature Driven Development (FDD) • Getting Real • Open Unified Process (OpenUP) • Scrum • Lean Software Development	[15]
• Extreme Programming (XP) • Scrum • Test Driven Development (TDD) • Feature Driven Development (FDD) • Dynamic System Development Method (DSDM) • Lean Software Development	[16]

P.1.2. tabula

Spējo metožu vecums

Metode	**Parādīšanās gads**
Kanban	1940
Scrum	1986
Microsoft Solution Framework	1993
Dynamic System Development	1994
Crystal	1996
Acceptance Test Driven Development	1996

Metode	Parādīšanās gads
Feature Driven Development	1997
Agile Unified Process	1999
Extreme Programming	1999
Lean Software Development	2003
Test Driven Development	2003
Behavior Driven Development	2006

P.1.3. tabula

Populārākās spējās metodes pamatojoties uz *Agile Alliance* konferenču materiāliem

Metode	Pieminēta	Punkti
Crystal	0	0
Dynamic System Development	0	0
Microsoft Solution Framework	0	0
Agile Unified Process	2	40
Feature Driven Development	3	50
Acceptance Test Driven Development	9	60
Behavior Driven Development	27	70
Kanban	55	80
Extreme Programming	66	90
Lean	81	100
Test Driven Development	128	110
Scrum	223	120

P.1.4. tabula

Grāmatas par spējām metodēm portālā *Amazon.com* (dati iegūti 20.04.2012)

Metode	Grāmatas	Punkti
Acceptance Test Driven Development	40	10
Crystal	49	20
Dynamic System Development	82	30
Agile Unified Process	128	40
Behavior Driven Development	134	50
Lean Software Development	297	60
Microsoft Solution Framework	330	70
Feature Driven Development	344	80
Scrum	491	90
Kanban	735	100
Test Driven Development	1185	110
Extreme Programming	1703	120

P.1.5. tabula

Meklēšanas rezultāti pēc atslēgas vārdiem meklētājā *google.com* (dati iegūti 02.06.2012)

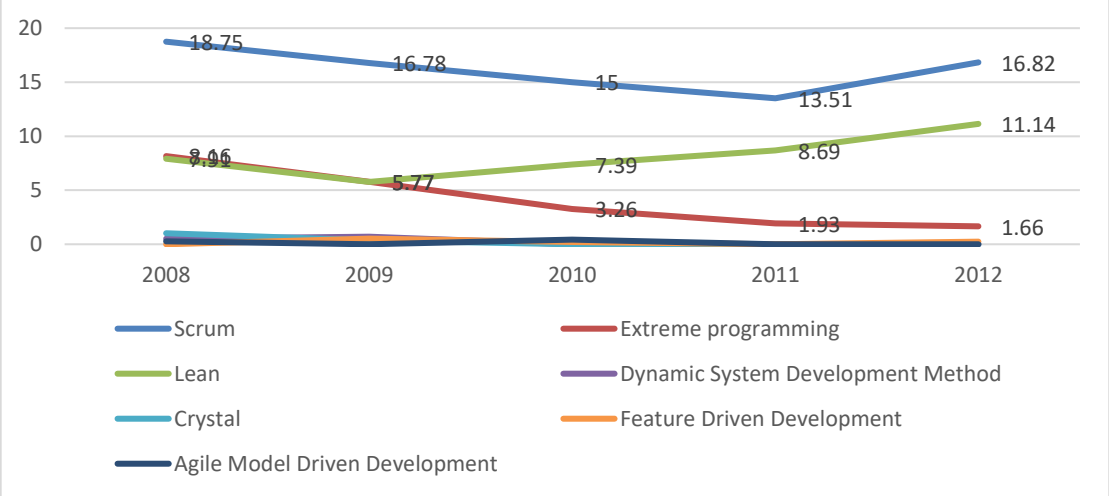
Metode	Google.com	Punkti
Dynamic System Development Method	601 000	10
Agile Unified Process	2 180 000	20
Behavior Driven Development	3 750 000	30
Acceptance Test Driven Development	4 570 000	40
Crystal	5 200 000	50
Kanban	6 020 000	60
Lean Software Development	6 380 000	70
Microsoft Solution Framework	17 100 000	80
Scrum	22 600 000	90

Metode	Google.com	Punkti
Extreme Programming	26 800 000	100
Test Driven Development	33 500 000	110
Feature Driven Development	43 800 000	120

P.1.6. tabula

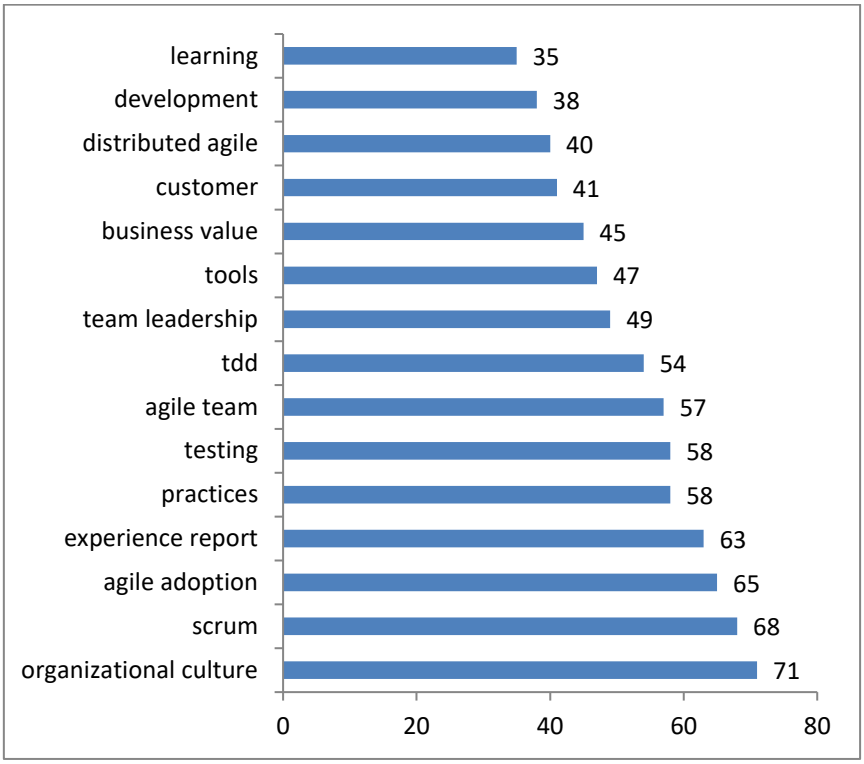
Meklēšanas rezultāti pēc atslēgas vārdiem meklētājā *bing.com* (dati iegūti 02.06.2012)

Metode	bing.com	Punkti
Dynamic System Development	417 000	10
Agile Unified Process	1 330 000	20
Behavior Driven Development	3 320 000	30
Crystal	3 440 000	40
Acceptance Test Driven Development	4 740 000	50
Kanban	5 150 000	60
Scrum	10 300 000	70
Lean Software Development	22 100 000	80
Microsoft Solution Framework	24 800 000	90
Extreme Programming	33 400 000	100
Feature Driven Development	43 400 000	110
Test Driven Development	62 600 000	120

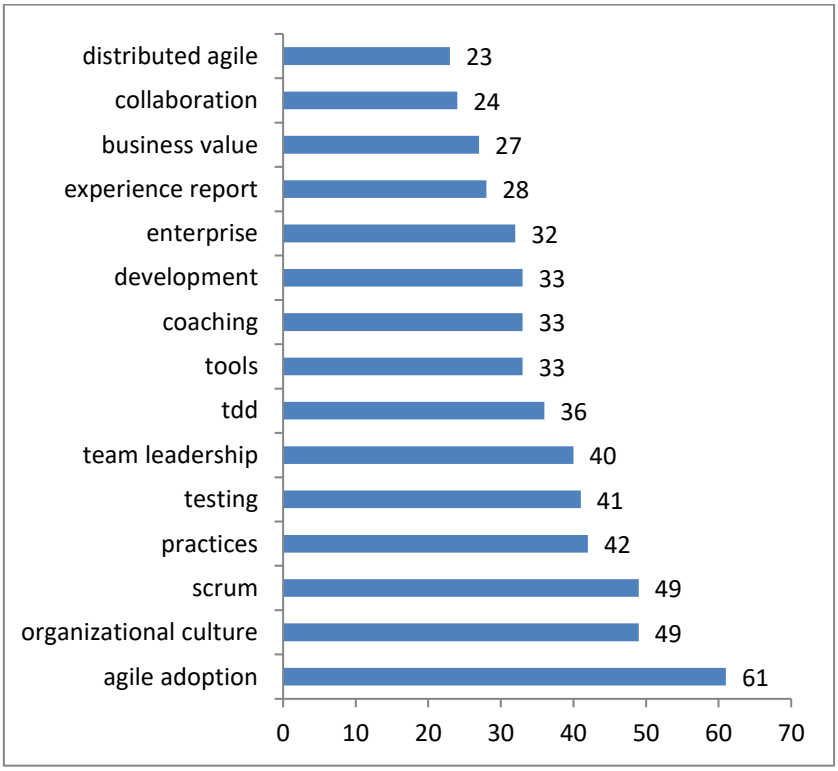


P.1.1. att. Metožu dinamika pa gadiem.

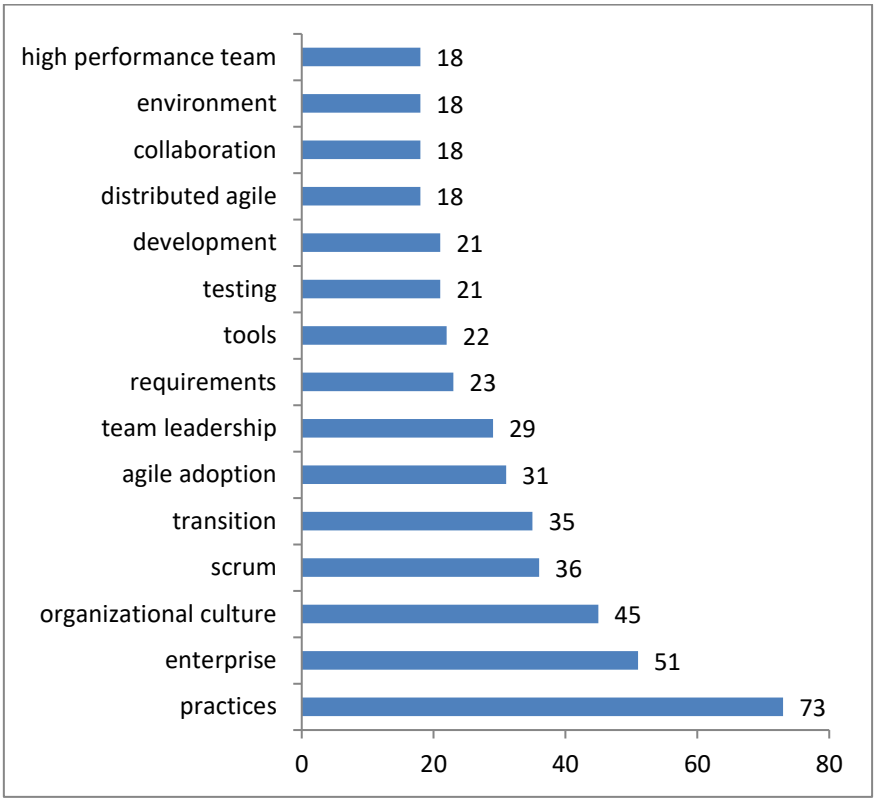
Spējo metožu attīstības tendences



P.2.1. att. 15. Populārākie atslēgas vārdi 2008. gadā.

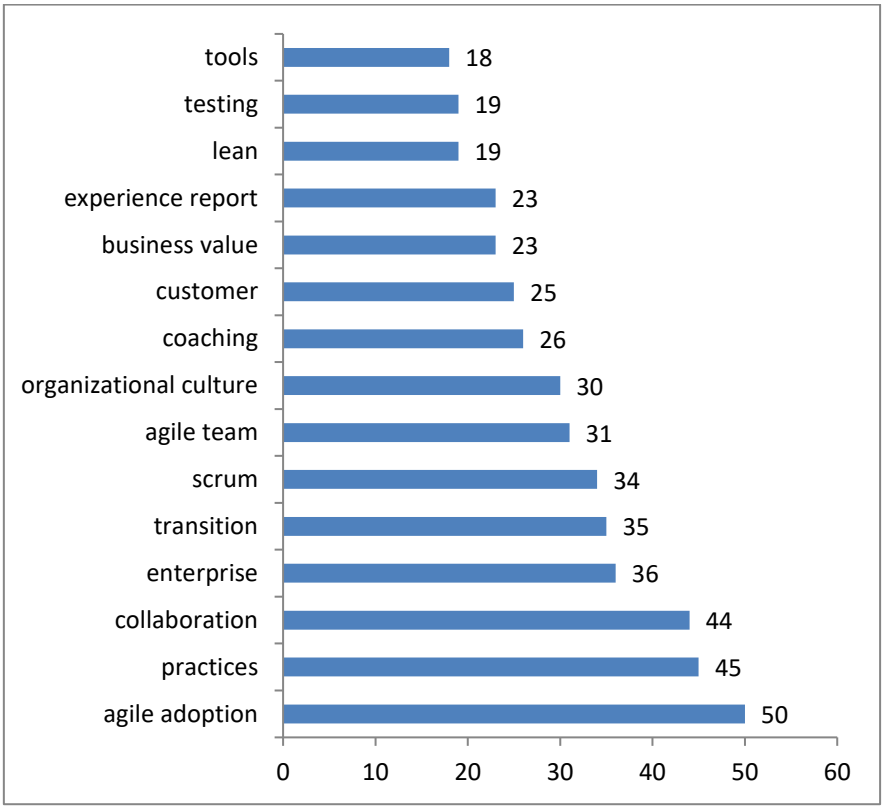


P.2.2. att. 15. Populārākie atslēgas vārdi 2009. gadā.

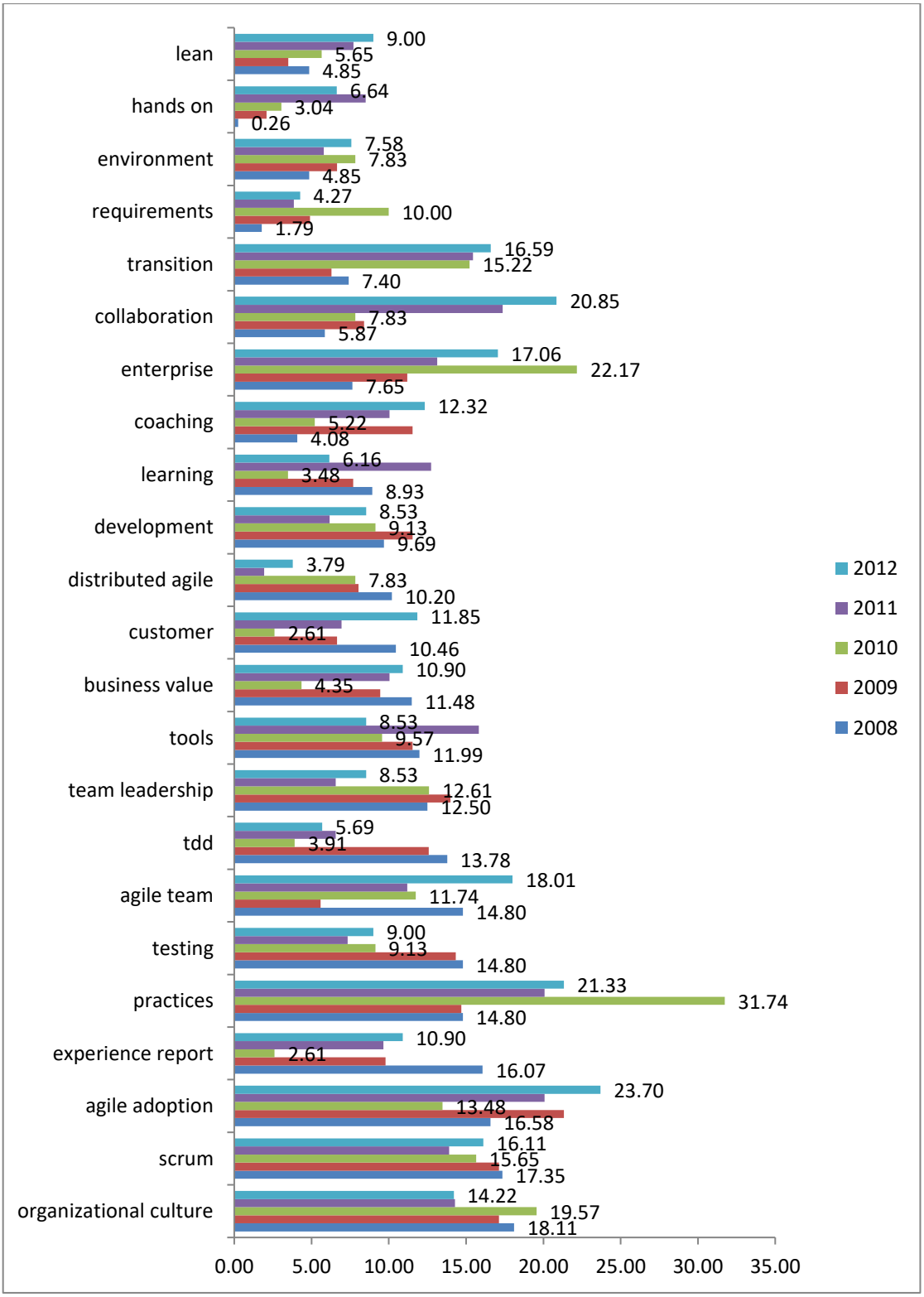


P.2.3. att. 15. Populārākie atslēgas vārdi 2010. gadā.

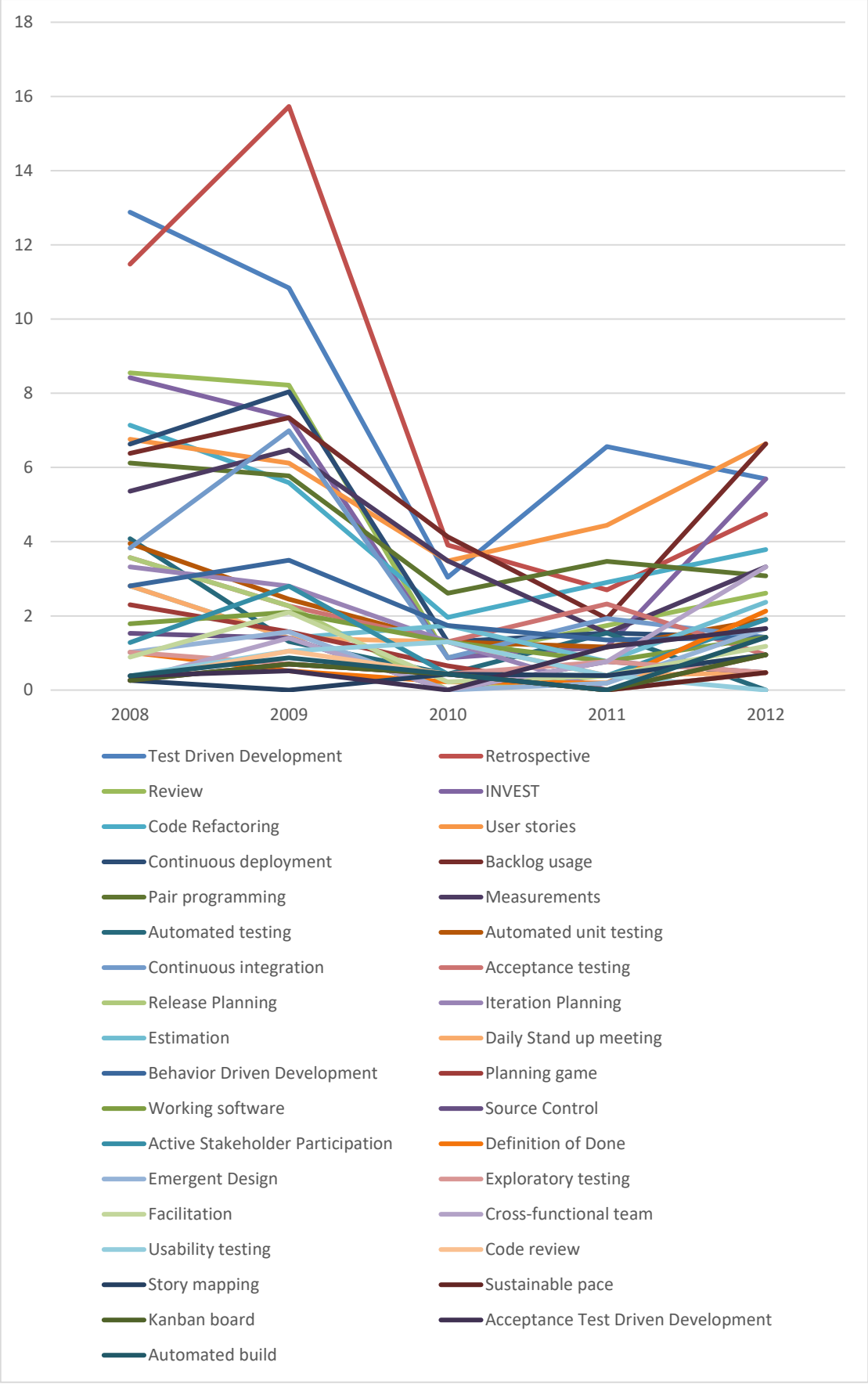
P.2.4. att. 15. Populārākie atslēgas vārdi 2011. gadā.



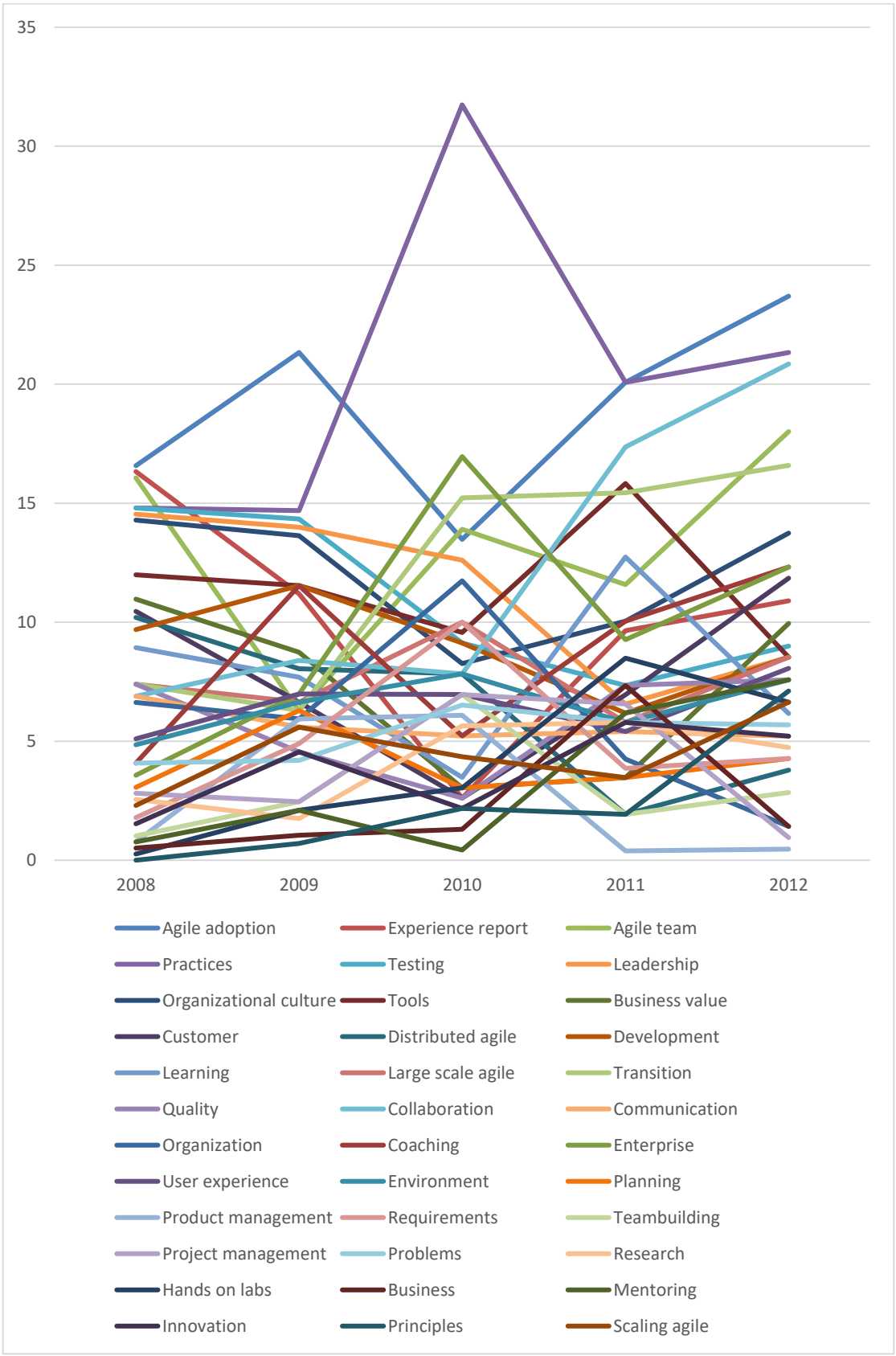
P.2.5. att. 15. Populārākie atslēgas vārdi 2012. gadā.



P.2.6. att. Atslēgas vārdu popularitāte procentuāli pret rakstu skaitu konkrētajā periodā no 2008. līdz 2012.



P.2.7. att. Spējo prakšu dinamika pa gadiem.



P.2.8. att. Atslēgas vārdu dinamika pa gadiem.

3. Pielikums

Spējās prakses

3.1.1. Automatizētais būvējums (angl. *Automated build*)

Būvējums ir process, kura rezultātā no programmas koda un citiem resursiem, par kuriem atbild izstrādātājs, tiek izveidota programma, ar kuru darbojas klients [55]. Būvēšanas process var sastāvēt no izejas dokumentu kompilēšanas, dokumentu arhivēšanas, uzstādīšanas paku izveidošanas, datu bāzes shēmas izveidošanas un citām nepieciešamām darbībām. Par automatizētu būvējumu tiek uzskatīta visu nepieciešamo soļu automatizācija, lai nebūtu nepieciešama cilvēka iejaukšanās un procesu varētu iniciēt nepieciešamajā brīdī bez papildu piepūles.

Pozitīvās īpašības:

Būvējuma automatizācija ir daļa no nepārtrauktas integrācijas (angl. *Continuous integration*), bet sniedz pozitīvu rezultātu arī tad, ja tiek izmantota atsevišķi.

- automatizējot būvēšanas soļus ir iespējams samazināt iespējamo kļūdu skaitu būvēšanas procesā, it īpaši, ja būvēšanas process sastāv no daudziem soļiem, un tos viegli sajaukt;
- ir nepieciešama detalizēta dokumentācija, un tas veicina skaidru izpratni par vidi un trešās puses atkarībām.

Trūkumi:

- būvēšanas procesam vajadzētu būt īsākam par 10 minūtēm, tai skaitā automatizēto testu izpildīšanai, savādāk būs grūti sasniegt nepārtrauktas integrācijas ieviešanu;
- automatizētam būvēšanas procesam ir jābūt palaižamam ārpus izstrādes vides, un tā veikšanai ir nepieciešama kvalitatīva infrastruktūra.

3.1.2. Automatizētā testēšana (angl. *Automated testing*)

Programmatūras testēšanā, testu automatizācija ir speciāla rīka izmantošana testēšanas nodrošināšanai, rezultātu apskatīšanai un salīdzināšanai [56]. Testu automatizācija sniedz iespēju automatizēt darbības testēšanas nodrošināšanai kādā formalizētā testēšanas procesā, kā arī iespēju testēšanai pievienot papildu testus, tādā veidā izvairoties no manuālās testēšanas.

Pozitīvās īpašības:

- palīdz ietaupīt laiku, it īpaši, regresa testēšanu. Automatizēto testēšanu var veikt katru nakti un no rīta pārbaudīt testēšanas rezultātus, tādā veidā pārliecinoties, ka iepriekšējā dienā pievienotā funkcionalitāte nav sabojājusi jau agrāk izveidoto funkcionalitāti [57];
- ņemot vērā, ka automatizētos testus palaiž dažādi rīki, tad šis process ir daudz ātrāks nekā manuālā testēšana;
- testēšanas programmatūras ietvars tiek regulāri atjaunots, kas sniedz papildu iespējas;
- palielināts testu pārklājums.

Trūkumi:

- testēšanas programmatūras ietvars tiek regulāri atjaunots, kas var novest pie situācijas, kad konkrēti testi pārstāj darboties;
- pilnīgi visu automatizēt pagaidām nav iespējams;
- testu sagatavošanai ir nepieciešams papildu laiks;
- komandas cilvēkiem ir nepieciešams augstāks zināšanu līmenis;
- ir nepieciešams izmantot papildu ārējos rīkus.

3.1.3. Automatizētā vienumu testēšana (angl. *Automated unit testing*)

Automatizētā vienumu testēšana savā ziņā ir līdzīga automatizētai testēšanai, starpība šajā gadījumā ir tajā, ka uzsvars tiek likts uz vienumu automatizētu testēšanu.

Pozitīvās īpašības:

- automatizēti vienumu testēšanas rīki palīdz nodrošināt koda kvalitāti tagad un nesalauzt to nākotnē [58];
- izstrādātājiem ir mazāk jāuztraucas par jaunas funkcionalitātes ieviešanu, jo vienumu testēšanas rezultātā, kļūdas tiek atrastas ātrāk;
- mazāk problēmu ar programmatūras ielāpu izvietošanu, jo izmaiņas iespējams, ātrāk notestēt;
- samazina nepieciešamību pēc manuālās testēšanas;
- samazina riska faktoru gadījumos, kad kāds no komandas biedriem ir slims vai ir jāaizvieto [58].

Trūkumi:

- negarantē, ka programmatūra ir bez kļūdām, tādēļ nedrīkst aizmirst, ka ir nepieciešama arī manuālā testēšana;
- nav garantijas, ka pats vienuma testa kods ir korekts;
- ir nepieciešams iemācīties nerakstīt liekus testus. Dažreiz lietotāji pievieno vienuma testu tikai pievienošanas pēc;
- komandas dalībnieki ir papildu jāizglīto vienumu testu rakstīšanā un konkrētā testu palaišanas rīka izmantošanā.

3.1.4. Darāmo lietu saraksts (angl. *Backlog*)

Spējā metodoloģija nav iedomājama bez darāmo lietu saraksta. Atkarībā no spējās metodes saraksta detaļas var mainīties. Galvenā saraksta nozīme ir apkopot darāmos darbus sarakstā, mainīt prioritātes sarakstā esošajiem ierakstiem un nākamajā iterācijā ņemt no saraksta elementus ar augstāko prioritāti.

Pozitīvās īpašības:

- ir zināms, kādi darbi būs jādara;
- darāmie darbi ir sakārtoti pēc prioritātes;
- ātri var atrast un komentēt darāmos darbus;
- viegli pievienot un izņemt plānotos darbus no saraksta. Viegli uzturēt PB aktuālu.

Trūkumi:

- nepieciešams rūpēties par saraksta aktualitāti.

3.1.5. Uzvedības vadīta izstrāde (angl. *Behavior Driven Development - BDD*)

Uzvedības vadītā izstrāde ir veidojusies sintezējot TDD un ATDD, tās apvienojot un papildu pievienojot vairākas iespējas [59]:

- pievienot katram lietotāja stāstam „Five Why`s" principu, lai lietotāja stāsts pilnībā atbilstu biznesa vajadzībām;
- vispirms ir jāpadomā par to, kāds būs biznesa iznākums no US, kurus vēlamies realizēt. Jāievieš tikai tie US, kuri nodrošina atbilstošo biznesa iznākumu;
- uzvedības ir jāapraksta tādā formā, lai tās uzreiz būtu skaidras izstrādātājiem, testētājiem un domēna ekspertiem.

Pozitīvās īpašības:

- BDD piedāvā labāku sadarbību starp izstrādātājiem, testētājiem un domēna ekspertiem;
- apraksti, kas veidoti izmantojot BDD pieeju (izmantojot Given-When-Then formātu) ir tuvāki valodai, ko mēs izmantojam ikdienā, tādēļ saprotamāki visām iesaistītajām pusēm;
- rīki, kurus izmanto BDD var automātiski uzģenerēt tehnisko un lietotāja dokumentāciju.

Trūkumi:

- BDD pieeju ir grūti paskaidrot programmētājiem iesācējiem, jeb cilvēkiem bez priekšzināšanām par TDD;
- BDD ir konceptuāla pieeja, un pagaidām nav pieejami daudzi rīki, bet šo rīku skaits palielinās, piemēram SpecFlow [60].

3.1.6. Koda pārstrukturēšana (angl. *Code Refactoring*)

Koda pārstrukturēšana ir koda iekšējas struktūras mainīšana, nemainot koda ārējo uzvedību. Neviens izstrādātājs nevar garantēt, ka atgriežoties pie izstrādātā koda pēc kāda noteikta laika, nebūs nepieciešamības to pārstrukturēt.

Pozitīvās īpašības:

- koda pārrakstīšana uzlabo koda mērķa atribūtus (garumu, dublēšanos u. c.), kas savukārt atvieglo koda uzturēšanu;
- koda pārstrukturēšana palīdz izprast, ko kods dara;
- koda pārstrukturēšana palīdz izstrādātājiem izprast dizaina lēmumus, it īpaši, ja koda uzturēšanā ir iesaistīti vairāki izstrādātāji.

Trūkumi:

- koda pārstrukturēšanai ir nepieciešams laiks, kurš šim nolūkam ir jāparedz;
- komandas tehniskajām zināšanām ir jābūt ļoti augstā līmenī, kādu vienmēr nav iespējams nodrošināt.

3.1.7. Koda pārskats (angl. *Code review*)

Koda pārskats ir sistemātiska koda pārlūkošana un to izmanto, lai atrastu kļūdas, kuras izveidojušās izstrādes fāzē. Koda pārskata rezultātus var izmantot, lai izlabotu

kļūdas vai, lai veiktu koda pārstrukturēšanu, piemēram, ja kodu ir rakstījis nepieredzējis izstrādātājs [62].

Pozitīvās īpašības:

- visi izstrādātāji komandā zina kādu kodu raksta pārējie;
- kļūdas tiek atrastas ātrāk;
- izstrādātāji ir spiesti rakstīt lasāmāku kodu;
- labās kodēšanas prakses ātrāk tiek nodotas izstrādātājiem ar mazāku pieredzi.

Trūkumi:

- koda pārskata organizēšanai ir nepieciešams laiks un tajā ir jāiesaista visa komanda, kas šo praksi var padarīt dārgu;
- var rasties spriedze komandā, gadījumos, ja sastopas ļoti pretēji viedokļi. Ir reizes, kad konkrēts pieraksta veids ir tikai kāda konkrēta izstrādātāja izvēle.

3.1.8. Nepārtraukta izvietošana (angl. *Continuous deployment*)

Nepārtraukto izvietošanu var uzskatīt par nepārtrauktās integrācijas (angl. *Continuous integration*) paplašinājumu, kura ir orientēta uz sagatavošanas laika samazināšanu [63]. Lai sasniegtu nepārtrauktu izvietošanu, komanda paļaujas uz sagatavotu infrastruktūru, kurā ir automatizēti izvietošanas soļi, kuru rezultātā lietotājs saņem atjauninātu programmatūras versiju. Nepārtrauktas izvietošanas gadījumā, procesam ir jābūt vadītam tādā veidā, lai būtu iespējams, atgriezties pie iepriekšējās versijas.

Pozitīvās īpašības:

- galvenā pozitīvā īpašība ir sagatavošanas laika samazināšana;
- ātrāka investīciju atgriešana pēc katras jaunas versijas izvietošanas;
- ātrāka atgriezeniskā saite no lietotāja.

Trūkumi:

- infrastruktūrai ir jābūt ļoti labi sakārtotai un automatizējamai, kas prasa zināmus finanšu resursus.

3.1.9. Nepārtraukta integrācija (angl. *Continuous integration*)

Nepārtrauktai integrācijai ir divi galvenie mērķi, samazināt laiku un piepūli, kas nepieciešama integrācijas ciklam un spēja piegādāt jebkurā brīdī jaunu strādājošu versiju.

Pozitīvās īpašības:

- samazināts piegādes laiks un piepūle;
- iespēja ātri uzlikt jaunu strādājošu versiju;
- iespēja ātri iegūt atgriezenisko saiti no lietotāja.

Trūkumi:

- ir jāizmanto ārēji rīki, kuri var nebūt lēti;

- komandai ir jābūt augstam zināšanu līmenim, ko nav iespējams vienmēr nodrošināt.

3.1.10. Daudzfunkcionāla komanda (angl. *Cross-functional team*)

Daudzfunkcionāla komanda ir komanda, kuras dalībniekiem ir dažāda veida funkcionālās zināšanas, kuras tiek izmantotas konkrēta mērķa sasniegšanai [64].

Pozitīvās īpašības:

- indivīdi ar dažādiem skatupunktiem palielina kreativitātes līmeni;
- uzlabojas problēmas atrisināšanas iespējas;
- uzlabojas komunikācijas līmenis.

Trūkumi:

- dēļ dažādiem skatupunktiem var rasties konflikti komandas ietvaros;
- komandai ir jābūt, ļoti saskaņotai laba rezultāta iegūšanai;
- problēmas var rasties ar lielizmēra projektiem [65].

3.1.11. Ikdienas īsās sapulces (angl. *Daily Stand up meeting*)

Katru dienu, konkrētā laikā komanda savācas uz 15 minūtēm un dalās informācijā ar pārējiem, atbildot uz trīs jautājumiem:

- Ko es darīju vakar?
- Ko es darīšu šodien?
- Vai manam darbam kaut kas traucē?

Sinonīmi:

- “Daily stand-up”;
- “Daily scrum”;
- „huddle”;
- „roll-call”.

Pozitīvās īpašības:

- sapulce ir ātrāka un efektīvāka nekā parastā sapulce, kurā darbiniekiem ir atļauts sēdēt;
- komandas dalībnieki iegūst vērtīgu informāciju par to, ko dara pārējie komandas biedri.

Trūkumi:

- nepareizi organizējot sapulci, tā pārvēršas par atskaitīšanos, kam tā nav paredzēta;
- komandas sāk ieslīgt pārāk lielā detalizācijas līmenī vai sāk apspriest tehniskas detaļas;
- komanda baidās dalīties ar problēmām, pat ja tādas ir;
- komanda uzskata šo sapulci par lielu formalitāti un nevēlas tajā piedalīties.

3.1.12. Pabeigšanas definīcija (angl. *Definition of Done*)

Komandai ir jāvienojas par kritērijiem, kuriem jāatbilst US, lai to varētu uzskatīt par pabeigtu. Ja kāds US neatbilst kādam no šiem kritērijiem, tad to nevar uzskatīt par pabeigtu iterācijas ietvaros [67].

Sinonīmi:

- „DONE-done";
- „READY-ready";
- "Done List";
- "Done Checklist";
- „Product sashimi".

Pozitīvās īpašības:

- pabeigšanas definīcija nodrošina sarakstu, kurš palīdz implementācijas aktivitātēs;
- definīcija palīdz izvairīties no atkārtotas darīšanas, ja US ir pieņemts, tad tas ir pieņemts;
- konkrēta līguma ievērošana samazina pārpratumus starp klientu, izstrādātāju un produkta īpašnieku.

Trūkumi:

- dažiem US ir nepieciešami īpaši pabeigtības nosacījumi;
- pabeigšanas nosacījumiem ir jābūt precīzi izveidotiem, savādāk, tas nesniegs sagaidāmo rezultātu un var situāciju tikai pasliktināt. Visai komandai ir jāvienojas, par konkrētu pabeigtības nosacījumu izmantošanu.

3.1.13. Veidojošais dizains (angl. *Emergent Design*)

Izmantojot veidojošos dizainu, komanda sāk piegādāt daļu no nepieciešamās funkcionalitātes un ļauj sistēmas dizainam pakāpeniski veidoties. Piemēram, tiks implementēta funkcionalitāte A un pēcāk funkcionalitāte B. Kamēr funkcionalitāte B tiks būvēta, izstrādātāji apskatīsies, kas A un B funkcionalitātei ir kopīgs un kopīgās lietas tiks izdalītas, tādā veidā ļaujot veidoties dizainam [68].

Pozitīvās īpašības:

- mazāk izejas koda;
- zemākas uzturēšanas izmaksas.

Trūkumi:

- veidojošais dizains ir ļoti atkarīgs no koda pārstrādāšanas, tā nodrošināšanai ir nepieciešams atbilstošs vienumu testu skaits.

3.1.14. Vērtēšana (angl. *Estimation*)

Programmatūras izstrāde nav iedomājama bez darāmo darbu vērtēšanas, jo visas iesaistītās puses grib zināt, cik liela piepūle būs nepieciešama, lai paveiktu konkrēto darbu. Parasti, bet ne vienmēr, piepūle tiek izteikta ar laiku, kurš nepieciešams darba

padarīšanai. Saskaitot kopā katra atsevišķā elementa izpildes laiku ir iespējams iegūt novērtējumu, cik daudz laika kopumā nepieciešams, lai ieviestu atbilstošo funkcionalitāti.

Pozitīvās īpašības:

- novērtēšana ir nepieciešama, jo savādāk nav skaidrības, cik daudz darbu var ņemt konkrētajā iterācijā;
- klients ir informēts par aptuveno laiku, kurš nepieciešams izstrādei un testēšanai.

Trūkumi:

- neievērojot nenoteiktības faktoru, novērtējums var būt kļūdains;
- nepareizi rīkojoties ar novērtējumiem ir viegli panākt situāciju, ka daļa lietotāju stāstu tiek novērtēti ar pārāk lielu laika buferi;
- daudzi neizprot, ka sākotnējais novērtējums ir sākotnējais novērtējums un tas var mainīties atkarībā no papildu informācijas, kura ir iegūta iterācijas laikā.

3.1.15. Izpētes testēšana (angl. *Exploratory testing*)

Izpētes testēšana ir testēšanas pieeja, kuras galvenās raksturiezīmes ir:

- uzsvars tiek likts uz testētāja autonomiju, prasmēm un kreativitāti;
- rekomendē veikt dažādas testu orientētas aktivitātes viscaur projektam, nevis tikai konkrētā fāzē;
- uzsvars tiek likts uz savstarpēji atbalstošu tehniku un lielu testu daudzumu, nevis formālu testa plānu.

Pozitīvās īpašības:

- mazāk sagatavošanās darbu, kļūdas tiek atrastas ātrāk, un pieeja ir intelektuāli stimulējoša;
- testētāji var izmantot deduktīvo spriešanu, pamatojoties uz iepriekšējiem rezultātiem, un testēt uzreiz nepieciešamo vietu, nevis gaidīt, kamēr konkrētā vieta tiks notestēta;
- liela daļa kļūdu ir atrastas pēc sākotnējās testēšanas.

Trūkumi:

- testi, kuri izveidoti un izpildīti neplānoti, nevar būt iepriekš pārskatīti un ir apgrūtinoši pateikt, kādi testi bija izpildīti;
- nav konkrētas secības, kādā testi tika palaisti vai arī instrukcijas ir ļoti sarežģītas.

3.1.16. Veicināšana (angl. *Facilitation*)

Veicinātājs ir cilvēks, kuram ir iedotas tiesības vadīt sapulces un viņa galvenais mērķis ir veidot sapulcei nepieciešamo atmosfēru, lai grupas process būtu efektīvs [70].

Pozitīvās īpašības:

- sapulce notiek atbilstošā atmosfērā un tā ir produktīvāka.

Trūkumi:

- kādam ir jāuzņemas veicinātāja loma;
- veicinātajam ir jābūt zinošam.

3.1.17. INVEST

Prakse nosaka, ka US ir jāatbilst 6 kritērijiem [71]:

- „I" ndependent – neatkarīgam no citiem US;
- „N" egotiable – diskutējamam;
- „V" aluable – vērtīgam;
- „E" stimable – novērtējamam;
- „S" mall – nelielam, lai varētu izveidot vienas iterācijas ietvaros;
- „T" estable – testējamam.

Pozitīvās īpašības:

- US ir neatkarīgi, diskutējami, vērtīgi, novērtējami, nelieli un testējami, tādi, ar kādiem var strādāt.

Trūkumi:

- grūti ir pateikt, kad sasniegts vēlamais rezultāts.

3.1.18. Iterāciju plānošana (angl. *Iteration Planning*)

Iterāciju garums parasti ir no 1 līdz 6 nedēļām. Komanda organizē plānošanas sapulci pirms katras iterācijas, kuras laikā US tiek sadalīti darāmajos darbos. Plānošanas sapulces garumam ir jābūt no 2-4 stundām, ja tam tiek veltīts vairāk laika, tad drīzākais tam tiek tērēts pārāk daudz laika [72].

Pozitīvās īpašības:

- sākoties iterācijai, ir zināmi darāmie darbi, kuri nepieciešami konkrētā US realizācijai;
- visa neskaidrā informācija par US tiek noskaidrota.

Trūkumi:

- sapulces var ievilkties, kā rezultātā, tie US, kuri tiek apskatīti beigās ir sadalīti darbos un novērtēti nekvalitatīvi;
- jāprot organizēt iterāciju plānošanas sapulces, lai ieguvums būtu maksimāls.

3.1.19. *Kanban*

Kanban praksi izmanto, lai organizētu darbu, kurā figurē saraksts ar mums svarīgiem darbiem un stāvokļiem, kādiem šiem darbiem ir jāiziet cauri [73]. Vienkāršos pielietošanas gadījumos uz tāfeles tiek izveidotas 3 kolonnas: „To do", „In progress", „Done", bet aktivitāšu skaits var būt lielāks pēc nepieciešamības. Programmatūras izstrādē kolonnas varētu būt: „Backlog", "Ready", "Coding", "Testing", "Approval", "Done", "Live".

Pozitīvās īpašības:

- optimizē sarakstu;
- samazina nevajadzīgo lietu apjomu;
- nodrošina elastīgumu produkcijas vidē;
- palielina produktivitāti;
- samazina izmaksas;
- uzlabo plūsmu;
- samazina sagatavošanas laiku.

Trūkumi:

- mazāk efektīvs gadījumos ar dalītiem resursiem;
- nedrīkst aizmirst par WIP (angl. *Work In Progress*) ierobežojumiem (Vienlaicīgi progresā nevar būt pārāk daudz darba uzdevumu).

3.1.20. Mērījumi (angl. *Measurements*)

Spējām metodēm ir svarīgi mērīt dažādus ar izstrādi saistītos atribūtus, lai varētu saprast, kā notiek izstrādes process. Spējās metodes izmanto konkrētas metrikas [74]:

- programmatūras izmērs;
- produktivitāte;
- statusa grafiki – progresa mērījumiem;
- kumulatīvās plūsmas grafiks – norāda darba apjomu konkrētā stadijā;
- biznesa vērtība;
- piepūles novērtējums.

Pozitīvās īpašības:

- mērījumi palīdz noteikt produktivitāti;
- var novērtēt lietotāja stāstus un programmatūras izmērus;
- noteikt darbu statusu;
- noteikt nepieciešamo piepūli.

Trūkumi:

- mērījumu rezultāti ir jāprot pareizi interpretēt;
- izmantojot pārāk daudz dažādas metrikas, var sanākt, ka daļa no viņām ir lieka un reālajā situācijā nemaz nav nepieciešama.

3.1.21. Pāru programmēšana (angl. *Pair programming*)

Pāru programmēšana ir tad, kad divi izstrādātāji konkrētu laika sprīdi strādā pie viena datora. Viens izstrādātājs raksta kodu, otrs izsaka komentārus, pēc kāda brīža lomas tiek mainītas un darbs tiek turpināts.

Pozitīvās īpašības:

- uzlabojas koda kvalitāte;
- labāka zināšanu izplatīšanās komandas ietvaros;
- jaunie izstrādātāji ātrāk apgūst jaunas zināšanas, jo pāros parasti saliek pieredzējušu izstrādātāju ar mazāk pieredzējušu;

- samazinās koordinēšanas piepūles, jo divi cilvēki dara kopā vienu uzdevumu un ir informēti par niansēm, kas saistītas ar konkrēto US;
- gadījumā, ja kādam no pāra izstrādātājiem ir jābūt kaut kur citur, tad darbs pie konkrētā lietotāja stāsta turpinās.

Trūkumi:

- nepieciešamā rezultāta sasniegšanai ir pareizi jāizveido pāri;
- abiem programmētājiem ir jābūt pilnībā iesaistītiem;
- par 15% paaugstinātas izmaksas [75].

3.1.22. Plānošanas spēle (angl. *Planning game*)

Spēle, ko spēlē spējās komandas, lai veiktu US novērtēšanu. Katram no komandas tiek iedotas novērtējumu kārtis, un visi dalībnieki apsēžas tā, lai viņi viens otru redzētu. PO nolasa informāciju par konkrēto US, tā vērtību un detaļām. Tad vienlaikus tiek paceltas novērtējumu kārtis. Ja novērtējumi nav viens un tas pats skaitlis vai divi skaitļi blakus, tad ir nepieciešams veikt atkārtotu novērtēšanu. Pirms atkārtotas novērtēšanas veikšanas, dalībnieks ar mazāko punktu skaitu un lielāko punktu skaitu sniedz paskaidrojumus, kādēļ viņš domā, ka viņa vērtējums ir pareizs. Ja novērtējuma punkti atrodas blakus, tad tiek piefiksēts lielākais no viņiem [76].

Pozitīvās īpašības:

- izmantojot spēles formātu, novērtēšanas process ir vienkāršs, un komanda neieslīgst sīkās tehniskās detaļās. Vairumā gadījumu novērtējumi ir precīzi;
- sapulces formāts ļauj izteikt savas domas pat klusākajiem komandas dalībniekiem;
- diskutēšanas rezultātā ir iespējams iegūt detalizētu informāciju par US.

Trūkumi:

- visu komandas dalībnieku tikšanās ir dārga;
- moderatoram ir jābūt izveicīgam, lai tiktu ievērotas sapulces robežas;
- dažas lietas var traucēt precīziem novērtējumiem, jo var ietekmēties no valdonīgām personām vai organizācijas politikas.

3.1.23. Laidienu plānošana (angl. *Release Planning*)

Laidiena plānošanas sapulce tiek organizēta, lai izveidotu laidienu plānu, kurš nosaka, kas un kad tiks piegādāts. Laidienu plāns vēlāk tiek izmantots, lai izveidotu iterāciju plānus.

Pozitīvās īpašības:

- noteikti laidienu grafiki un to saturs;
- laidienu saturs ir saskaņots ar pasūtītāju un skaidri definēts.

Trūkumi:

- pastāv iespējamība, ka plāns būs jāmaina, jo spējās metodēs ir grūti saplānot darbus stipri uz priekšu.

3.1.24. Retrospektīva (angl. *Retrospective*)

Retrospektīva ir veids, kā atskatīties atpakaļ uz jau notikušiem notikumiem un tos izvērtēt. Retrospektīva tiek sasaukta iterācijas un projekta beigās, lai varētu spriest par to, kas ir darbojies labi un kas slikti. Ieteicamais retrospektīvas sapulces garums ir 1h iterācijas beigās. Sapulces laikā tiek izveidoti divi saraksti, lietas kuras darbojas labi un lietas, kuras darbojas slikti.

Pozitīvās īpašības:

- process ir vienkāršs un efektīvs;
- komanda regulāri attīstās un mācās no savām kļūdām.

Trūkumi:

- sapulce nedrīkst būt formāla;
- ne visi sapulces dalībnieki vēlas izteikties, ir nepieciešams veids, kā nerunīgos iesaistīt diskusijā.

3.1.25. Izejas datņu kontrole (angl. *Source Control*)

Izejas datņu kontrole ir ļoti svarīga prakse, kuru ir ieteicams izmantot visām nopietnām programmatūras izstrādes kompānijām. Izejas datņu kontrole nodrošina, failu versionēšanu un informācijas par izmaiņām saglabāšanu, kā arī iespēju veidot jaunus koda zarus un tos apvienot, izmantojot konfliktu risināšanas rīkus.

Pozitīvās īpašības:

- iespēja uzturēt dažādas datņu versijas;
- iegūt konkrētu datņu versiju;
- veidot koda zarus;
- apvienot koda zarus;
- datņu apvienošanai izmantot konfliktu risināšanas iespējas;
- rīki ir integrēti izstrādes vidē.

Trūkumi:

- ir nepieciešams iegādāties rīku servera licences, lai gan tagad ir pieejami arī ļoti attīstīti bezmaksas rīki.

3.1.26. Stāstu kartēšana (angl. *Story mapping*)

Stāstu kartēšana sastāv no US sakārtošanas divās neatkarīgās dimensijās. Karte sakārto lietotāja aktivitātes uz horizontālās ass prioritātes secībā un vertikālā ass parāda implementācijas sarežģītību pieaugošā secībā. Tādā veidā sakārtotas kartes nodrošina, ka pirmā horizontālā rinda ir „Ceļa skelets" (angl. *„waliking skeleton"*), tā ir vienkāršota strādājoša produkta versija. Turpinot pārvietošanos pa rindām iegūsim programmai papildu funkcionalitāti [77].

Pozitīvās īpašības:

- nodrošina, ka nākamajā izlaidumā izvietotā biznesam nepieciešamā funkcionalitāte būs strādājoša un nebūs tādas situācijas, ka svarīgā

funkcionalitāte nedarbojas, jo iepriekšējos soļos nav ieviesta kāda zemāk stāvoša funkcionalitāte.

Trūkumi:

- kartēšanas process var būt sarežģīts un darbietilpīgs, ir nepieciešams iemācīties ātri noteikt prioritātes un savstarpējās saiknes starp US.

3.1.27. Noturīgs temps (angl. *Sustainable pace*)

Komandai ir nepieciešams mierīgs darba temps visu laiku. Tas tiešā veidā attiecas uz nepieciešamību strādāt virsstundas, kas ir aizliegts ar komandas un vadības vienošanos. Nav svarīgi, kādas ir šīs virsstundas, komandas dalībnieks pēc virsstundām ir pārguris un viņa darbaspējas krasi krītas [78].

Pozitīvās īpašības:

- komandas darba spējas ir augstā līmenī ilgstošā laika periodā.

Trūkumi:

- kritiskos brīžos darbinieki var negribēt strādāt papildu stundas.

3.1.28. Testu vadāmā izstrāde (angl. *Test Driven Development - TDD*)

Testu vadāmā izstrāde ir programmēšanas pieeja, kad pirms biznesa koda uzrakstīšanas, vispirms tiek izveidots tests, ar kuru jauno funkcionalitāti varēs pārbaudīt. Lai izmantotu šo praksi, ir nepieciešams ievērot konkrētus likumus [79]:

- uzrakstiet vienu parastu vienuma testu, kurš apraksta konkrētu programmas daļu;
- tests ir jāpalaiž, un tam ir jāatgriež kļūda, jo biznesa kods šai funkcionalitātei vēl nav gatavs;
- ir jāuzraksta tik daudz biznesa kodu, lai nodrošinātos, ka tests iziet veiksmīgi;
- pārveidot biznesa kodu tā, lai tas būtu maksimāli vienkāršs;
- darbības atkārto, no jauna pievienojot jaunus testus un funkcionalitāti.

Pozitīvās īpašības:

- samazinās kļūdu skaits;
- projekta beigu fāzē, ir mazāk jācīnās ar kļūdām;
- uzlabojas programmatūras arhitektūra.

Trūkumi:

- ir nepieciešams papildu laiks testu izveidošanai;
- mainoties funkcionalitātei, ir nepieciešams mainīt arī agrāk uzrakstītos testus, kam nepieciešams papildu laiks.

3.1.29. Lietojamības testēšana (angl. *Usability testing*)

Lietojamības testēšana ir nepieciešama, lai varētu saprast, kā lietotājs reaģēs uz konkrētu sistēmas dizainu un vai sistēma ir ērti lietojama. Process sastāv no lietotāja novērošanas, kamēr viņš darbojas ar sistēmu [80].

Pozitīvās īpašības:

- tiek iegūta atgriezeniskā saite ar gala lietotāju;
- lietotāja saskarnes kļūdas un nepilnības tiek atrastas ātrāk;
- palielinās lietotāju apmierinātība ar sistēmu;
- samazinās risks, ka programma netiks lietota;
- lietotājiem ir vieglāk sasniegt savus mērķus.

Trūkumi:

- testēšanas rezultāti var būt neprecīzi, jo dažādiem lietotājiem var būt dažādi skatupunkti.

3.1.30. Lietotāju stāsti (angl. *User stories*)

Pēc sarunām ar klientu un produkta īpašnieku, komanda sadala darbu funkcionālos papildinājumos jeb US. Katram lietotāju stāstam ir jābūt maksimāli neatkarīgam vienam no otra [81].

Pozitīvās īpašības:

- viens no labākajiem veidiem, kā aprakstīt jauno funkcionalitāti skaidri un saprotami;
- iespējams funkcionalitāti sadalīt nelielās daļās, kuras vieglāk realizēt;
- klienti un produkta īpašnieki var mainīt US aprakstus, pirms US ir implementēts;
- izstrādātājiem ir skaidrs, kas viņiem ir jāuztaisa.

Trūkumi:

- var nepareizi sadalīt prasības US, it īpaši, ja komanda US praksi ir sākusi izmantot nesen;
- US sākotnējais apraksts var būt nepietiekami detalizēts;
- izmantojot US praksi ir nepieciešams izmantot arī vienumu testus un pieņemšanas testus, jo savādāk pastāv liela iespēja, ka būs regresa kļūdas.

3.1.31. Strādājoša programmatūra (angl. *Working software*)

Strādājoša programmatūra ir viens no spējo metožu pamatprincipiem un praksēm. Komandas uzdevums ir panākt to, ka ir iespējams iegūt strādājošu programmatūru jebkurā laikā, tādā veidā nodrošinot klientu ar jaunāko versiju pēc nepieciešamības. Dažās komandās tiek piekopts, ka jaunie US tiek veidoti jaunos koda zaros, panākot, ka ir iespējams uzbūvēt un sagatavot versiju ar pēdējiem pabeigtiem US, neriskējot ar nestabilu versiju.

Pozitīvās īpašības:

- jebkurā laikā ir iespējams iegūt programmatūras jaunāko versiju;
- klients ir apmierinātāks, jo var ātrāk pārbaudīt jaunāko programmatūru.

Trūkumi:

- papildu koda zaru izveidošana un uzturēšana ir darbietilpīgs process, jo versiju apvienošana prasa papildu laiku.