LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

METODES SPĒJĀS IZSTRĀDES PAMATPRINCIPU IEVĒROŠANAI DALĪTO PROJEKTU KVALITĀTES UZLABOŠANAI

MAĢISTRA DARBS

Autors: **Lauris Ulmanis,** Stud.apl.Nr. lu12003
Darba vadītāja: profesore, Dr. comp.sc., Darja Šmite

RĪGA 2014

Anotācija

Darba ietvaros tika veikts pētījums, lai labāk izprastu veidus kā ieviest spējās programmatūras izstrādes pamatprincipus dalīto projektu izstrādē. Pētījuma ietvaros tika analizēts eksistējošs projekts Latvijā ar dalītu projektu izstrādes modeli, kurā divu gadu laikā tika mainīts programmatūras izstrādes modelis, pārejot no tradicionālā ūdenskrituma uz spējās izstrādes modeli. Šajā laikā tika veikti vairāki praktiski mēģinājumi ieviest spējās izstrādes idejas, kuras pēcāk tika vērtētas vai ir nesušas uzlabojumus vai gluži pretēji – radījušas papildus problēmas izstrādē. Lielākā daļa no realizētajām idejām ar nelielām modifikācijām tika iestrādātas projekta izstrādes procesos. Pateicoties šiem praktiskajiem mēģinājumiem, darba autoram radās ideja izpētīt, kā spējā izstrāde notiek citos projektos un aprakstīt ieteikumus, kuras informāciju tehnoloģiju kompānijas varētu izmantot kā pamatu, lai pārietu uz spējās programmatūras izstrādes modeli dalītajā projektu izstrādē, tos pielāgojot savam izstrādes modelim, lai projekta adaptācija spējās izstrādes modelim notiktu raitāk un nebūtu jāveic eksperimentālas darbības, lai saprastu, vai izmaiņa nesīs uzlabojumus izstrādes procesos.

Darba pamatā tika izmantotas 2 pieejas problēmas pētīšanai:

1) Pieejamās literatūras izskatīšana, kā ietvaros tika meklēti pieejamie materiāli jau projektos zināmu problēmu apzināšanai un risināšanai;
2) Praktiskais darbs, kā ietvaros tika analizēti un aprakstīti veiktie eksperimentālie soļi Projektā, un balstoties pieejamās literatūras izpēti, izstrādātas nepieciešamās izmaiņas, lai projekts izmantotu spējās izstrādes īpašības.

Dalītajos projektos, kur izstrāde notiek pēc tradicionālā ūdenskrituma modeļa, bieži grūti ir noteikt kopējo projekta izpildes statusu, jo saistītās kompānijas izstrādātā komponente tiek piegādāta tikai tad, kad tā tiek uzskatīta par pilnībā gatavu, izejot cauri visiem ūdenskrituma modeļa soļiem. Ja saistītā kompānija nepiegādā laicīgi komponenti, tā var radīt draudus uz kopējo produkta izstrādi. Tādēļ daudzos projektos visā pasaulē tiek veikti mēģinājumi ieviest Agile izstrādes idejas, lai projektu varētu attīstīt iteratīvi un nodrošinot kopēju, pakāpenisku un kvalitatīvu projekta attīstību.

Abstract

The aim of the work is to carry out the research to understand better the ways how to introduce the basic principles of Agile software development into Globally distributed projects. In the research there is analyzed the existing project in Latvia with the model of Global distribution where the model of Agile software development was gradually changed during 2 years moving from traditional waterfall model to Agile software development. That time there were done several practical attempts to introduce the ideas of Agile software development, later they were assessed if they had brought any improvements or on the contrary – had created extra problems in development. Most of the ideas that had been carried out with small modifications were turned in processes of the project. Due to these practical attempts, the author of the work had the idea to explore Agile software development in other projects and write the suggestions which IT companies could use it as the basics to move to the model of Agile software development in Global distributed projects and then adapt them to their development, so the adaption of the model of Agile software development in the project would be quicker and it wouldn`t be necessary to do any experimental actions to understand if changes brought any improvements in processes of development.

On the base of the work there are used two ways to explore the problem:

1) searching for available materials which recognize and solve the problems that are known in the projects;
2) Practical work, doing the analyses and writing experimental steps done in the research project and basing on found materials, there is worked out the most necessary changes, so the project could use the qualities of Agile software development.

In the Global distributed projects where the development follows the traditional waterfall model, it is often difficult to establish the total status of the project development because the component, made by cohesive company, is delivered only when it is considered completely ready and has been got through all the steps of waterfall model. If the cohesive company does not deliver the component on time, it can cause the threat to the total development of the product. Therefore there are carried out the attempts to introduce the ideas of Agile development in many projects in the whole world to develop this project frequentative and provide the common, qualitative growth of the project.

Autoreferāts

Šī maģistra darba izstrādes laikā autors ir veicis Latvijā esoša dalītā projekta izstrādes procesu analīzi, konstatējis projekta sākotnējās problēmas, un balstoties uz literatūras izpēti, izstrādājis nosacījumus esošo izstrādes procesu uzlabošanai projekta turpinājuma fāzei. Darbības rezultātā ir veikts pētījums, kādā veidā ieviest spējo programmatūras izstrādi projektos ar dalīto izstrādes modeli. Veicamie uzlabojumi projektā ir bāzēti uz citu autoru rekomendācijām, veicot papildus literatūras apskatu un analīzi.

Autors ir iedziļinājies projekta izstrādes procesos, aprakstot kādas problēmas projektu skāra, sākotnēji strādājot ar ūdenskrituma izstrādes modeli un kādā veidā projektā pakāpeniski tika ieviesta spējā izstrāde. Tā kā spējās izstrādes ieviešana notika projektā samērā lēni un nekontrolēti, dēļ zināšanu trūkuma par to, kādā veidā strādāt ar jauno izstrādes metodoloģiju, darbs ir uzrakstīts ar domu, lai arī citi projekti varētu turpmāk izmantot tajā aprakstītās metodes un priekšrakstus, kā strādāt iteratīvi projektos, kuros izstrāde tiek attīstīta vairāku saistītu kompāniju ietvaros.

Saturs

APZĪMĒJUMU SARAKSTS

Apzīmējums	Skaidrojums
Agile	Izstrādes metodika jeb metodoloģiju grupa, kas balstās uz iteratīvu un inkrementālu izstrādes modeli.
COCOMO	Izmaksu novērtēšanas modelis, kas izmanto noteiktu regresa formulu darba vienumam nepieciešamo izmaksu apēķināšanai.
Problēmu raksturojumu shēma	Cēloņu raksturojumu shēma (*Fishbone diagram*), kas ļauj identificēt projekta riskus un problēmas
IP	Izmaiņu pieprasījums – klienta pieteikums, kas definē izmaiņas sistēmā
IT	Informācijas tehnoloģijas
Izstrādātāja kompānija	Kompānija, kas veic darbu vienumus, ko nosaka vadošā kompānija, Ja izstrādātāja kompānija nodot darba vienumus citai kompānijai, tad šīm kompānijām tā skaitās vadošā
Scrum	Agile izstrādes metodoloģija
Vadošā kompānija	Kompānija, kas noteiktus darbus nodod izstrādei izstrādātāja kompānijai. Vadošā kompānija var būt kā izstrādātāja kompānija, ja tā pakļauta citai vadošai kompānijai.

1 Ievads

Attīstoties informācijas tehnoloģiju nozarei, pasaulē arvien biežāk projektu realizācijā ir vērojama tendence kompānijām apvienot spēkus, lai nodrošinātu nepieciešamā darba apjoma veikšanu klienta izvirzītajos piegādes termiņos. Šāda tendence izriet no vairākiem aspektiem – no darba resursu ierobežojumiem vienas kompānijas ietvaros, no darbinieku kompetences trūkuma, no ierobežoto finanšu resursu sadales viedokļa, kā arī no klienta prasību viedokļa. Dalīto projektu nozīmīgums mūsdienās strauji pieaug un daudzas kompānijas sāk to eksperimentālu ieviešanu. Ar dalītā projekta jēdzienu saprot vienai vadošo kompānijai saistītās vairākas citas kompānijas, kurām savukārt var būt pakārtotas vēl citas saistītas kompānijas. Dalīto projektu nozīmīgumu mūsdienās uzsver arī autore Suprika Vasudeva Šrivastava rakstā [1] . Autore raksta, ka bieži dalītie projekti noved pie papildus finanšu izdevumiem, nevarot kompānijām efektīvi virzīt kopēju saskaņotu izstrādes procesu vai vienoties par risinājumiem, termiņiem un izstrādes metodēm. Kompānijas cenšas ierobežot finanšu resursus IT izstrādē un pie reizes nodrošināt IT projektu kvalitātes nepasliktināšanos. Tika izvirzīts jautājums, vai ir iespējams savlaicīgi konstatēt problēmas dalīto projektu izstrādē un tās novērst.

Veicot projekta darba izpildes dalījumu pa kompānijām, parādās dažādi riski, kas var ietekmēt projekta izpildes gaitu, rezultātu un tālāk arī radīt ietekmi uz kompānijas kopējiem ieņēmumiem. Tā kā pasaulē aizvien lielāku popularitāti gūst spējā programmatūras izstrāde [17] , šī darba ietvaros tiks analizēts, kā spējās programmatūras izstrādes modelis ietekmē dalītos projektus. Darba pamatā kā mērķis tika izvirzīts gūt atbildes uz sekojošiem jautājumiem:

1) kā spējās izstrādes metodes ir pielietojamas dalīto projektu izstrādē?
2) kādos veidos ir iespējams uzlabot iteratīvās programmatūras izstrādes procesus dalīto projektu izstrādē?

Lai rastu atbildes uz izvirzītajiem jautājumiem, darba ietvaros tika analizēti vairāku citu autoru veikto pētījumu raksti, konstatēti riski izstrādes procesos, metodes to novēršanai. Papildus izmantotai literatūrai tika analizēts Latvijas valsts mēroga projekts, kura izstrāde uzsākta 2011.gadā un kura izstrādē uz darba rakstīšanas brīdi piedalās 3 kompānijas. Vēsturiski projektā ir piedalījušās vēl 4 kompānijas, 2 no tām ārpus Latvijas robežām. Analizētajos materiālos, balstoties uz izvirzītajiem mērķiem, tika meklēta un analizēta informācija, kas apraksta procesus projektos, kuros ir implementēti pilnvērtīgi vai pietuvināti spējās izstrādes risinājumi.

Lai varētu atbildēt uz pirmo izvirzīto jautājumu, ir jāsaprot spējās izstrādes pamati, kā tie tiek izmantoti projektos ar standarta izstrādes modeli. Tādēļ nākamā nodaļa tiek veltīta šo spējās

izstrādes pamatprincipu izskaidrošanai, lai tālāk uz to bāzes veiktu literatūras apskatu un atbilstoši arī projekta analīzi.

2 Spējās izstrādes pamati

Spējā izstrāde notiek cikliski jeb iteratīvi. Projekta izstrādes vienumi tiek dalīti pa darba pakām ar piegādi iterācijās, kuru ilgums parasti ilgst no vienas nedēļas līdz mēnesim, kas tiek noteikts kā maksimālais iterācijas ilgums. Iterāciju garums projektos tiek fiksēts un dalīts vienmērīgās piegādēs.

Autore Suprika Vasudeva Šrivastava rakstā [1] uzsver, ka spējā izstrāde pamatā balstās un četriem principiem. Pirmkārt, ir jārada programmatūra, kas apmierina klienta prasības, iteratīvi veicot programmatūras piegādes tā, lai klients varētu to novērtēt un sekot līdzi projekta attīstībai. Otrkārt, izstrādājot programmatūru iteratīvi, ir jābūt iespējai mainīt sākotnējās prasības un pielāgot to jebkurā no projekta izstrādes posmiem. Treškārt, ik dienu ir jānotiek komunikācijai starp programmētājiem un biznesa / sistēmu analītiķiem. Ceturtkārt, programmatūras testēšana ir primārāka par pašu izstrādi. Respektīvi, ir jāvar izstrādāt testa scenāriju pirms pašas izstrādes, lai gūtu pārliecību par izstrādātās komponentes kvalitāti, jo pieejot iteratīvi izstrādei, ir jāvar pietiekami īsā laikā realizēt kvalitatīvu sistēmas komponenti.

Pamatelementi, kas nosaka spējo programmatūras izstrādi ir sekojoši [2] :

1) Lietotāju stāsti (*User Stories*) – apraksti, kas top no lietotāju vēlmēm pret topošo produktu un tā atbilstību biznesa prasībām. Lietotāju stāsti pamatā apraksta biznesa procesus, kas ir jāveic jauni veidojamai sistēmai, kā arī dod iezīmes sistēmas arhitektūrai un datu modelim;
2) Darbu saraksts (*Product backlog*) – no lietotāju stāstiem apkopots darbu saraksts, kas ir jāveic, lai realizētu funkcionējošu sistēmu. Darbu sarakstā tiek iekļautas jau izvērtētas nepieciešamās darbietilpības un vēlamā, bet ne cieši noteiktā izstrādes secība;
3) Iterāciju plānošana (*Selected Product Backlog*) – noteiktu darbu apkopošana un sadalīšana iterācijās, turklāt lielākos darbus, dalot mazākās iterācijās. Šeit svarīgi ir ņemt vērā, ka iterāciju garums visas projekta dzīves periodā ir fiksēts, un tajos esošie darba vienumi ir jādala tik mazos darbos, lai tos varētu uzspēt izstrādāt līdz iterācijas nodošanai testēšanai;
4) Dienas darbu pārrunāšana (*Daily Scrum*) – ja iterācijas var ilgst no 1 nedēļas līdz par mēnesim, tad ir svarīgi pārrunāt un saplānot darbus dienas griezumā. Dienas plānošana

parasti ilgst maksimāli 15 minūtes un tās laikā katrs projekta dalībnieks, kas piedalās iterācijas izstrādē, informē komandu par paveiktiem darbiem un darbiem, kas tiks paveikti tuvākās dienas laikā. Svarīgi ir nediskutēt par darbiem un neiegrimt niansēs. Diskusijas starp dalībniekiem ir jāorganizē atsevišķā laikā;

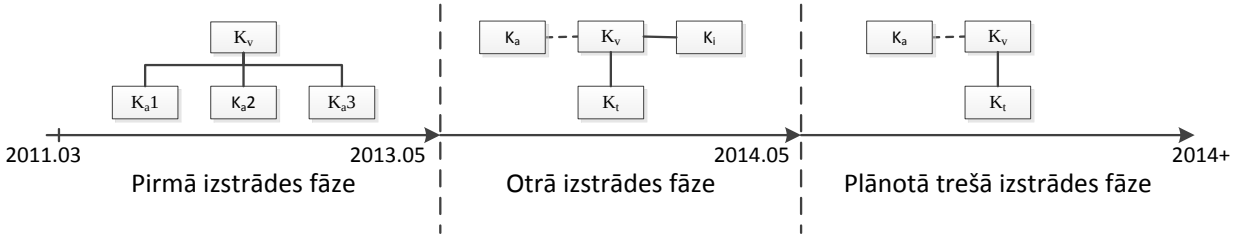
5) Faktiskā darbu izpilde (*Burn-Down charts*) – ilustratīvi atainots iterācijā plānoto darba vienumu izpildes grafiks pa dienām, kas attēlo faktiski atlikušo plānoto laiku, kāds bija paredzēts iterācijas plānošanas procesā. Šāds grafiks ļauj efektīvāk pārskatīt darbu izpildes statusu un izdarīt secinājumus, vai darba vienumam ir radušās kādas problēmas ar izpildes plānu vai viss rit pēc plānotā grafika;
6) Retrospekcijas sanāksmes (*Retrospective*) – iekšējas sanāksmes starp projekta komandu dalībniekiem, kas tiek rīkotas pēc iterācijas slēgšanas un darba vienumu piegādes klientam testēšanai. Šajās sanāksmēs tiek atspoguļota trasējamība starp topošās sistēmas darba sarakstu un jau paveikto. Parasti katram darba vienumiem tiek noteikts statuss un pabeigtības pakāpe. Papildus retrospekcijas sanāksmē tiek analizētas problēmas un nepieciešamības gadījumā tiek domāti jauni risinājumi nākamajām iterācijām. Ja ir darbi, kuri nav izdarīti, sanāksmē lemj par to pārcelšanu un pārdali uz nākamajām plānotajām iterācijām, izrunājot cēloņus.
7) Iterāciju pārskata sanāksmes (*Sprint Review*) – ārējas sanāksmes starp vadošās kompānijas klienta pārstāvi, projekta vadītāju un klientu, kā ietvaros tiek pārrunāts kopējais projekta statuss. Sanāksmē tiek vērtēta sagatavotās iterācijas kvalitāte, balstoties uz akcepttestēšanas atskaiti, kā ietvaros ar klientu tiek pārrunāts, vai ir realizēts viss iterācijā plānotais un realizēts pieņemamā kvalitātē. Ja kāds no programmatūras vienumiem nav izstrādāts pietiekamā kvalitātē, var tikt pieņemts lēmums to pārnest uz nākamo iterāciju.

Spējās izstrādes pamatā ir svarīgs princips, lai atdalītu no programmatūras izstrādes visu mazsvarīgo, kas ir tradicionālajā izstrādes modelī un kas apgrūtina kopējo izstrādes procesu. Mērķis ir ātri reaģēt uz prasību izmaiņām, nodrošinot operatīvu izmaiņu sistēmā, tās arhitektūrā vai sistēmas biznesa procesos, kā arī mēģināt izstrādāt programmatūru pirms nodevuma datuma.

Darba turpinājumā tiks apskatīts Latvijā esošs projekts, kurš ir pamats šī maģistra darba praktiskai daļai, aprakstītas šajā projektā konstatētās problēmas, ieviešot pirmo fāzi, kā arī aprakstot, kādas izmaiņas skāra otro fāzi, mainot projekta izstrādes metodes no ūdenskrituma modeļa uz spējās izstrādes modeli.

3 Projekta uzbūve

Lai labāk izprastu projekta uzbūvi, ir nepieciešams iepazīties ar projekta trim izstrādes fāzēm un saistītajām kompānijām, kas tajās ir bijušas iesaistītas. Projekts, kurā maģistra darba autors piedalās kā sistēmu analītiķis vadošajā kompānijā K_v, tika uzsākts 2011.gada martā, un šobrīd ir otrās fāzes slēgšanas procesā, jo 2014.gada maijā tika piegādāta un uzstādīta produkcijas vidē pēdējā iteratīvi izstrādāta komponente. Attēlā 1 ir atspoguļotas iesaistītās kompānijas sadalījumā pa projekta fāzēm. Pirmā fāzes izstrādē projekta vadošai kompānijai K_v tika piesaistīti papildus resursi programmatūras vienumu izstrādei no trim saistītajām kompānijām (attēlā 1 attēlotas kā K_a1, K_a2 un K_a3). Savukārt otrajā fāzē papildus resursi tika izmantoti testēšanas un automatizēto testu izstrādei no kompānijas K_t, savukārt analīzes procesiem tika piesaistīta kompānija K_i. Kompānija K_a šai fāzē veica savu neatkarīgu komponentes izstrādi, piegādi nodrošinot caur vadošo kompāniju K_v.



Attēls 1. Projekta uzbūve, saistīto kompāniju sadalījums pa projekta fāzēm

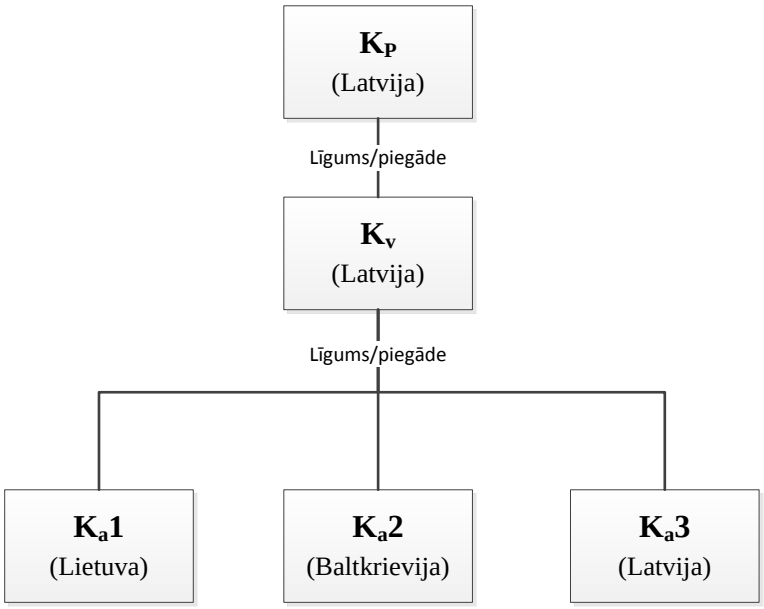
Kā redzams attēlā 1, trešā fāze tiek plānota uzsākt, stipri nemainot esošo kompāniju sadalījumu projektā, jo darba apjomi ir pietiekami, lai esošais sadalījums netiktu mainīts. Vienīgā izmaiņa, kas skar trešo fāzi, ir atteikšanās no kompānijas K_i, kas palīdzēja ar analīzes darbiem projektā. To turpmāk veiks pati vadošā kompānija K_v. Projekta kompāniju atšifrējumi atrodami projekta pirmās un otras fāzes aprakstos, attiecīgi darba 4. un 5. nodaļā. Pirmajā projekta fāzē tika izmantoti ārvalstu kompāniju resursi, bet otrajā fāzē dēļ 4. nodaļā aprakstītajām problēmām vadošās kompānijas vadība no tām atteicās un tālāku projekta attīstību balstīja uz iekšējiem resursiem, papildus piesaistot citu Latvijā esošo kompāniju resursus. Otrā fāzē tika veikti vairāki iteratīvās izstrādes uzlabojumi, par kuriem tiks aprakstīts darba 5.nodaļā.

4 Projekta pirmā fāze, konstatētās problēmas un ieviestie risinājumi

Projekta pirmajā fāzē bija paredzēts strādāt pēc tradicionālā ūdenskrituma modeļa, kas nosaka standarta izstrādes modeli – prasību specificēšanu, projektējumu aprakstu, izstrādi,

testēšanu un piegādi klienta testa/produkcijas videi. Pirmajā fāzē tika konstatētas dažādas nepilnības, kas vēlāk radīja problēmas projekta attīstībai un draudus noslēgtajam līgumam ar klientu. Sākotnēji projektu attīstīja tikai viena kompānija - vadošā, ar kuru klients bija slēdzis līgumu par izstrādi noteiktā laika posmā. Sistēma tika pārņemta no iepriekšējā izstrādātāja, līdz ar to kompānijas darbiniekiem nebija pieredzes par sistēmas arhitektūru un izstrādes principiem. Tas ļoti apgrūtināja jaunu komponenšu izstrādi, jo grūti bija novērtēt nepieciešamās darbietilpības un kā arī zināšanu trūkuma dēļ grūti bija klientam piedāvāt efektīvus risinājumus jauni izstrādājamajām sistēmas komponentēm. Zināšanu trūkuma dēļ par sistēmu, grūti bija izstrādāt prototipus, lai saprastu komponenšu sarežģītību un līdz ar to nevarēja precīzi novērtēt nepieciešamo darbietilpību, lai rezervētu cilvēkresursus to izstrādei. Dēļ šīm problēmām, kompānijas saskārās ar grūtībām laicīgi izstrādāt piegādājamās komponentes un nācās meklēt papildus resursus izstrādei citās ārējās kompānijās. Tas bija mēģinājums īsākā laika periodā tomēr paveikt nepieciešamos izstrādes darbus.

Ūdenskrituma modelis stingri ierobežo programmatūras izstrādes fāzes [2] un līdz ar to līgumiski noteikto nodevumu datumus, kurus ir grūti mainīt, jo līgums bieži paredz soda sankcijas pret izstrādātāju, ja produkts tiek piegādāts ar nokavētu iesniegšanas termiņu. Tas ne tikai ietekmē kompānijas ieņēmumus, bet arī pašas kompānijas reputācija tiek zaudēta IT izstrādes tirgū, radot problēmas šai kompānijai piedalīties turpmākos izsludinātos konkursos. Un tas savukārt noved pie tā, ka turpmākie darbi var netikt pasūtīti un jauni līgumi var netikt slēgti. Mēģinot piegādāt izstrādātos moduļus noteiktajā termiņā, tika projekta izstrādes sākumā lemts par papildresursu izmantošanu. Tādējādi atsevišķi darbi, kurus varēja atdalīt kā atsevišķas komponentes, tika atdoti izstrādei pārējā trim arējām kompānijām.

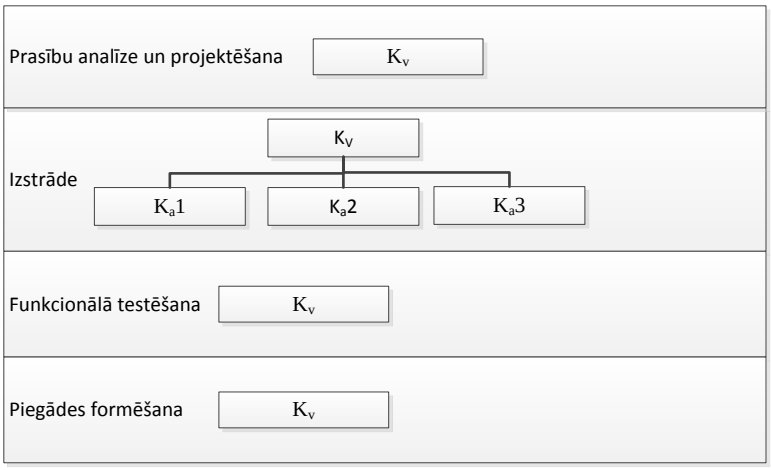


Attēls 2. Kompāniju sadalījuma shēma pirmajā fāzē

Kompāniju sadales shēmu, kas attēlota attēlā 2, strādājot pēc ūdenskrituma izstrādes principiem, raksturo:

- K_p – Kompānija, kas pasūta programmatūru;
- K_v – Vadošā izstrādes kompānija, kas slēdz tiešu līgumu ar pasūtītāju;
- K_a1 – Saistītā kompānija Lietuvā, kas izstrādā komponentes kompānijai K_v (darbu apakšuzņēmējs kompānijai K_v);
- K_a2 – Saistītā kompānija Baltkrievijā, kas izstrādā komponentes kompānijai K_v (darbu apakšuzņēmējs kompānijai K_v);
- K_a3 – Saistītā kompānija Latvijā, kas izstrādā komponentes kompānijai K_v (darbu apakšuzņēmējs kompānijai K_v).

Projekta pirmās fāzes darbu sadale attēlota attēlā 3. Kā redzams, pirmajā fāzē saistītās kompānijas veic tikai izstrādes darbus, pārējos darbus veic vadošā kompānija, kas ir līgumiski atbildīga par programmatūras piegādi programmatūras pasūtītāja kompānijai.



Attēls 3. Projekta darbu sadalījuma shēma pirmajā fāzē

Projekta pirmajā fāzē, tika veikti vairāki eksperimentāli uzlabojumi, ar mērķi panākt nodevumu kvalitātes pieaugumu. Kompānijas vadība, izskatot problēmas, ieviesa vairākas izmaiņas projektā, pārejot projektam no vienas fāzes uz otru.

Lai uzlabotu dalītā projekta procesus un nodevumu kvalitāti, kompānijas ietvaros tika analizētas projektā konstatētās problēmas un tika domāts, kādas izmaiņas būtu jāievieš, lai nodrošinātu pilnvērtīgu izstrādi. Tādēļ, tika panākti vairāki uzlabojumi, balstoties uz konstatētajām problēmām pabeigtajā izstrādes fāzē:

Pirmajā fāzē izstrāde notika, balstoties uz ūdenskrituma modeli. Tā kā programmatūras izstrāde tika veikta dalīti un daļu no programmatūras piegādāja Lietuvā esošā kompānija un daļu piegādāja Baltkrievijā esošā kompānija, tad balstoties uz ūdenskrituma izstrādes modeli, izstrādātās programmatūras komponentes tika piegādātas tikai uz noteikto piegādes termiņu, kas bija atrunāts līgumā. Tas radīja problēmas caurskatīt kvalitāti jau ātrāk izstrādātām komponentēm un novērst laikus pamanāmās nepilnības izstrādē. Tas noveda pie tā, ka 1.fāze ieilga un nācās veikt papildus izmaiņas programmatūrā, lai novērstu savlaicīgi neatklātās kļūdas (*problēmas apzināšana*). Tādēļ kompānija lēma par to, ka ir nepieciešams pāriet uz iteratīvu izstrādes modeli 2.fāzē, lai varētu laicīgi kontrolēt piegādāto komponenšu kvalitāti un savlaicīgi novērst kļūdas (*izmaiņu plānošana*). Tā ar nākamo vienošanās protokolu slēgšanu ar klientu tika atrunāts, ka programmatūra tiks piegādāta daļēji iterācijās un savukārt saistītajām kompānijām tika dota piekļuve vadošās kompānijas programmatūras repozitorijam un tām bija jāsagatavo plānotās iterācijas nodevumi uz iterācijas piegādes slēgšanu (*izmaiņu ieviešana*).

Pēc iteratīvās izstrādes modeļa ieviešanas, tika panākta ātrāka komponenšu izstrāde, lai gan joprojām visā izstrādes fāzes gaitā saglabājās daudzas izstrādes problēmas, kas tieši ietekmēja

iterāciju nodevumu kvalitāti un visas fāzes laikā šīs problēmas tika risinātas pakāpeniski, ieviešot izmaiņas vienu pēc otra. Kā galveno nepilnību konstatēja komunikācijā ar Lietuvas un Baltkrievijas kompānijām (*problēmas apzināšana*), komunicējot ar vadošo kompāniju, kuras atrašanās vieta ir Latvija. Lai to novērstu, vadības līmenī tika pieņemts risinājums turpināt darbu, izmantojot pašu izstrādātāju spēkus, bet testēšanu atstāt citai Latvijā esošai kompānijai (*izmaiņu plānošana*). Tādējādi līgumiski saistošie darbi ar Lietuvas un Baltkrievijas kompānijām tika pabeigti un nākamie darbi tika uzsākti ar struktūru, kur vadošā kompānija veica programmatūras izstrādi un saistītā kompānija nodrošināja izstrādāto komponenšu testēšanu (*izmaiņu ieviešana*).

Lēmums atteikties no ārvalstu kompānijām tika pieņemts tādēļ, ka vadošai kompānijai trūka pieredzes kā veikt dalītu iteratīvu izstrādes procesu. Tā kā pirmā fāze nebija veiksmīga, kompānija tādējādi steidzami centās mazināt riskus uz nākamo fāzi, pārņemot visu izstrādi pie sevis.

Rezultātā 3.fāzes plānošanā kvalitātes grupas ietvaros tika ieplānotas vairākas eksperimentālas idejas projekta procesu uzlabošanai un kompānijas kvalitātes vietnē tika plānots aprakstīt priekšnosacījumus un ieteikumus, kā nodrošināt iteratīvu izstrādes modeli projektos ar dalīto izstrādi. Mērķis ir uzsākt 3.fāzi, izmantojot šos priekšnosacījumus, kā arī noteikt citiem kompānijas projektiem ieteikumus kā veikt iteratīvu izstrādi un iteratīvu izstrādi dalītajos projektos.

Pirmā fāze pēc pārejas uz iteratīvu izstrādi, joprojām strādāja, neievērojot daudzus, *Agile* metodikā noteiktos pamatprincipus. Vienīgais nosacījums, kas pirmajā fāzē darbojās bija iteratīva piegāde saistītajām kompānijām, kuras katru nedēļu piegādāja izstrādātos darba vienumus priekš kopējās sistēmas. Kā arī plānoja, kuri darba vienumi tiks izstrādāti nākamajā nedēļas iterācijā. Iesaistot citas kompānijas, radās arī grūtības ar dalīto izstrādes darbu. Vadošā kompānija K_v joprojām uz līgumiskās bāzes pamata piegādāja izstrādātās komponentes līgumā noteiktajos datumos. Pirmās fāzes konstatētās problēmas un pieņemtie lēmumi ir attēloti tabulā 1. Šīs problēmas risinājumi tika pētīti un ieviesti pēc sarunām starp projekta vadītāju un analītiķiem, ne visu projekta komandu.

Tabula 1. Pirmās fāzes cikli

Konstatētās problēmas	Pieņemtie risinājumi	Rezultāts
Resursu trūkums	Piesaistīt citu kompāniju resursus	Neveiksmīgs
Komunikācijas problēmas ar ārvalstu kompāniju komandām	Uz laiku piesaistīt vienu darbinieku vadošās kompānijas komandai	Neveiksmīgs
Komunikācijas problēmas ar Latvijā esošās saistītas kompānijas komandu	Rīkot tikšanos klātienē	Daļēji veiksmīgs
Daļēja iteratīva izstrāde	Iteratīvai izstrādei ir jābūt arī līgumiski atrunātai starp kompāniju K_v un K_p	Veiksmīgs

Darba turpinājumā sekos detalizēts apraksts par pirmās fāzes konstatētājām problēmām un risinājumiem, lai lasītāju iepazīstinātu ar projekta pirmsākumiem pirms iteratīvās izstrādes pakāpeniskas ieviešanas tajā.

4.1 Ārējo resursu piesaiste

Mēģinot piegādāt izstrādātos moduļus noteiktajā termiņā, tika projekta izstrādes sākumā lemts par papildresursu izmantošanu. Tādējādi atsevišķi darbi, kurus varēja atdalīt kā atsevišķas komponentes, tika atdoti izstrādei 3 arējām kompānijām, kur 2 no tām atradās ārvalstīs un tikai viena Latvijā. Tā kā piegādes termiņi mainīti netika, bet saistītajām kompānijām vajadzēja saprast sistēmas uzbūves kopējo koncepciju, izstrādes process neritēja pietiekami raiti un produkts netika piegādāts termiņā un pietiekami labā kvalitātē, kas savukārt paildzināja akcepttestēšanas procesu un ieviešanu produkcijā atliekot par vairākiem mēnešiem. Dēļ tā, ka izstrādājamās komponentes piegādes laiks strauji tuvojās, līgums par citas valsts kompāniju darbinieku nomu vadošās kompānijas projektam tika slēgts ar samaksu nevis par padarīto darbu, bet gan par patērētām stundām projektā, jo tā visvieglāk varēja piesaistīt papildus darba spēku tik īsā laikā. Tas savukārt iezīmēja riskus un papildu finanšu resursu nepieciešamību, jo saistītajām kompānijām nebija interese nodot laicīgi kvalitatīvi izstrādātu komponenti.

4.2 Komunikācijas problēmas ar ārvalstu kompāniju komandām

Tā kā projekti tika izstrādāti attālināti, radās saziņas problēmas ar prasību izskaidrošanu izstrādātājiem. Turklāt 2 no kompānijām bija ārvalstu un līdz ar to apgrūtinoši bija lasīt kompānijas K_v izstrādāto prasību specifikāciju un projektējumu latviešu valodā, jo bija katra prasība jātulko un jāizskaidro mutiski. Tulkošana bieži vien tika izmantoti mašīntulkošanas līdzekļi, kas bieži vien veica kļūdainu prasību pārtulkošanu un tālāk tas radīja kļūdas programmatūras izstrādē. Rādās papildus jauni neplānoti darbi, saistībā ar prasību tulkošanu un izskaidrošanu, kas savukārt radīja jaunus riskus projekta izstrādes fāzei. Komunikācijas problēmu mēģināja risināt ar ārējo kompāniju pārstāvju uzaicināšanu pastrādāt pāris nedēļas klātienē kopā ar vadošās kompānijas komandu, lai labāk izprastu ne tikai prasības, bet arī kopējo sistēmas arhitektūru. Atgriežoties savā kompānijā, no saistītās kompānijas atsūtītā pārstāvja mērķis bija nodot uzkrātās zināšanas pārējiem projektā iesaistītajiem darbiniekiem.

Vēlāk tika organizētas vairākas attālinātas sanāksmes, izmantojot telekonferences zvanu starp Latvijas kompāniju un Baltkrievijas kompāniju. Šādā telekonferencē tika sapulcināti visi atbildīgie vadošās un saistītās kompānijas komandu dalībnieki un tika pārrunāti darba vienumu statusi, lai saprastu, kas ir izdarīts un vēl jāuzspēj izdarīt līdz komponentes piegādei klienta akcepttesta videi. Parasti arī uzreiz pārrunāja noteiktu darba vienumu problēmas un pat risinājumus, ieteikumus kā pareizās veikt izstrādes darbus. Šāda pieeja tika organizēta tikai ar Baltkrievijā esošo izstrādes komandu. Ar Lietuvas komandu saziņa notika, izmantojot tiešsaistes saziņas līdzekli *Skype,* vai sazvanoties telefoniski par darba vienumu atbildīgajam analītiķim ar konkrētu programmētāju. Saziņa un saprašanās ritēja ļoti lēni un apgrūtinoši, jo šajā projekta fāzē vadošai kompānijai nebija ne pieredzes, kā organizēt dalīto projektu darbus tā, lai darbi tiktu paveikti laikā un nevajadzētu tos pārstrādāt.

4.3 Komunikācijas problēmas ar Latvijā esošās saistītas kompānijas komandu

Trešā kompānija, kura atradās Latvijā, biežāk varēja rīkot klātienes sanāksmes, izrunājot neskaidrās lietas. Tādējādi ar šo kompāniju sadarboties bija vieglāk, bet tomēr arī ne pietiekami efektīvi. Problēmas radīja nesakārtotie jauno komponenšu aprakstošie dokumenti, līdz ar to vadošai kompānijai vajadzēja daudz veltīt laiku izskaidrojošiem darbiem un attālināti tos bija veikt grūti. Radās pat situācijas, kad prasība tika pārprasta un piegādātā komponente strādāja ne atbilstoši klienta izvirzītajām prasībās un biznesa plūsmai. Tādēļ nācās komponenti pārstrādāt, novēršot funkcionālas nepilnības.

4.4 Daļējas iteratīvās izstrādes ieviešana

Tā kā sākotnēji iteratīva izstrāde bija noteikta tikai starp vadošo kompāniju K_v un saistītajām kompānijām K_a1, K_a2 un K_a3 tika pieņemts lēmums, ka nepieciešams nākamo projekta fāzi izstrādāt iteratīvi arī starp vadošo kompāniju un programmatūras pasūtītāja kompāniju K_p. Tas nozīmēja, ka bija nepieciešams veikt sarunas ar pasūtītāju un slēgt jaunu līgumu, kurā tika atrunāta iteratīvā programmatūras piegāde. Iteratīvai izstrādei sākumā tika noteikts ievērot sekojošus principus no spējās programmatūras izstrādes:

1) Uz vienošanās protokola bāzes atrunāt darba vienumus, kurus bija jāizstrādā vienošanas protokolā noteiktajos termiņos;
2) Iterācijas formē, pārvalda un piegādā vadošā kompānija K_v;
3) Darba uzdevumi tiek dalīti vienošanās protokola ietvaros pa iterācijām;
4) Saistītām kompānijām katrā iterācijā tiek izdalīti noteikti izstrādājami darba uzdevumi, kuri ir aprakstīti vienošanās protokolā;

Praktiski šī risinājuma ieviešana noteica projekta iteratīvās izstrādes sākumu visā projektā, ne tikai starp vadošo kompāniju un saistītām, bet arī izstrādātu komponenšu piegādē programmatūras pasūtītāja kompānijai. Lai gan šajā projekta attīstības posmā joprojām nebija projektā realizētas tādas iteratīvās izstrādes principi kā *Scrum* sanāksmes, iterāciju plānošanas sanāksmes, pāra programmēšanas un citi. Daudzi no iteratīvās izstrādes pamatprincipiem tika ieviesti tikai otrajā projekta fāzē, analizējot problēmas projektā un attīstot idejas, kā uzlabot darba procesus.

4.5 Pirmās fāzes slēgšana

Projekta pirmā fāze tika ieviesta produkcijā ar 15 mēnešu nokavēšanos, programmatūras pasūtītāja izstādīto rēķinu izstrādātājam ar soda naudu par termiņa kavēšanu, balstoties uz līgumā noteiktajiem kritērijiem programmatūras piegādes termiņa neizpildes gadījumā, kā arī klienta skaidrošanos augstākstāvošām varas iestādēm par Eiropas naudas neapgūšanu paredzētajos termiņos.

Tā kā produkts nebija pietiekami kvalitatīvs dēļ ārējo kompāniju atšķirīgas izpratnes par arhitektūru un prasībām, daudzas kļūdas tika konstatētas arī produkcijā esošajā versijā, kuras kompānija K_v vēl risināja nepilnu gadu, bieži vien pārstrādājot nepareizi izstrādātās komponentes.

Kā redzams, vadošai kompānijai K_v nebija skaidrs, kā vadīt dalīto projektu izstrādi, kā organizēt darbu plānošanu, realizācijas aprakstu veidošanu, izstrādes un programmatūras piegādes formēšanu. Līdzīgi kā autores Šmites aprakstītajā projekta pētījumā [24] arī šajā projektā tika konstatētas problēmas ar darbietilpību nepareizu novērtēšanu, it īpaši, ja darbi bija jādod izstrādāt citai kompānijai, kurai nebija ne mazākās pieredzes par topošo sistēmu. Līdz ar to arī sākotnēji izvirzītie nodevumu termiņi nebija adekvāti novērtēti veicamajam darbam.

Pārskatot projekta problēmas, kompānijā K_v tika pārdomāti projektu izstrādes principi un uzsākts domāt par to, kā novērst šādas problēmas turpmāk gan esošās projekta turpmākajās fāzēs, gan arī citos projektos. Kompānijā sāka domāt par variantiem, kā pilnveidot darba procesus, lai uzlabotu nodevumu kvalitāti un samazinātu finanšu izdevumus. Šī uzdevuma ietvaros tika izveidota procesu uzlabošanas darba grupa (kurā iesaistīts arī šī maģistra darba autors), kura veica projektos konstatēto problēmu analīzi. Viens no identificētajiem riskiem procesu uzlabošanas darba grupā bija potenciālas problēmas, kas var veidoties, ieviešot spējās programmatūras izstrādi projektos, tai skaitā arī projektos ar dalītu izstrādes modeli, jo ne tikai projekts tika izstrādāts, balstoties uz vairāku kompāniju darbinieku resursiem. Līdz ar šī riska analīzi radās jautājumi, kā attīstīt spējās programmatūras izstrādes pieeju gan parastajos projektos, gan arī dalītajos projektos, lai nodrošinātu gala nodevumu pietiekamu kvalitāti un naudas apguvi līgumā noteiktajos termiņos, nepārsniedzot līgumā pieļaujamo budžetu. Tādēļ otrā fāzē tika ieviestas vēl vairākas izmaiņas izstrādes procesos, par kurām tiks aprakstīts nākamā darba nodaļā.

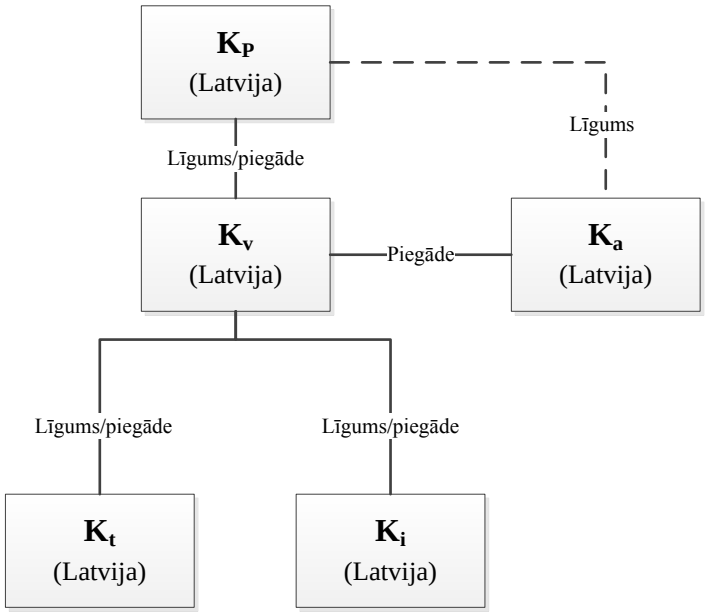
5 Projekta otrā fāze un iteratīvās izstrādes ieviešana

Iteratīvās izstrādes idejas Projektā tika attīstītas pakāpeniski un šobrīd projekts, kurš atrodas otrās fāzes noslēgumā, rāda labus rezultātus, jo 2.fāzē izstrādātās komponentes tika palaistas produkcijas vidē tikai ar 4 mēnešu kavējumu, kur, salīdzinot ar 1.fāzes produkcijā iešanas datumu, kavējums bija mērām 15 mēnešos. 4 mēnešu kavējuma iemesls bija fakts, ka maģistra darbā izmantotais projekts šobrīd izmanto *Agile* principus tikai daļēji jeb pašu ieviesto *Agile* hibrīda (daļēju) programmatūras izstrādes modeli. Tādēļ kompānija, kas attīsta maģistra darbā ietverto projektu, nolēma veikt otrās fāzes ieviesto iteratīvās izstrādes procesu analīzi un uz projekta trešo fāzi sastādīt priekšrakstus, kādas izmaiņas darba procesos ir jāmaina, lai uzlabotu programmatūras izstrādes efektivitāti un celtu izstrādāto programmatūras vienumu kvalitāti. Tas tika darīts ar mērķi, lai arī nākamie projekti varētu izmantot šos ieteikumus un spētu uzsākt

izstrādi iteratīvi, pielāgojot risinājumus saviem projektiem. Lai saprastu kādā veidā iteratīvā izstrāde attīstījās otrajā fāzē, darba turpinājumā sekos izklāsts par risinājumiem, kuri tika izmēģināti un ieviesti projektā.

Iesākot otro fāzi, pateicoties pirmās fāzes izmaiņām projektā, nedaudz uzlabojās izstrādes procesi, jo izstrāde tika veikta, balstoties uz vadošās kompānijas pieejamiem resursiem un tika uzsākta darbu vienumu izstrāde iteratīvi. Paralēli prasību vākšanai, apstiprinātās prasības tika aprakstītas un jau nekavējoties uzsākta vienumu izstrāde, kas vienas nedēļas noteiktajās iterācijās tika piegādāta klientam testēšanai akcepttestēšanas vidē. Tiklīdz izstrādātais darba vienums bija pabeigts, tam tika veikta zaļā ceļa testēšana un pēc tās, tas tika nodots saistītajai kompānijai, kura veica funkcionālos testus. Bet joprojām iterāciju nodevumi bija nekvalitatīvi un klients tajās konstatēja daudz kļūdu – gan kritiskās kļūdas programmatūrā, gan arī funkcionālās kļūdas noteiktu biznesa scenāriju izpildē (*problēmas apzināšana*). Lai saprastu cēloņus un uzlabotu nodevumu kvalitāti, kompānijas vadība nolēma veikt projekta procesu analīzi, balstoties uz kvalitātes uzlabošanas sanāksmēm un projekta ietvaros veikt nepieciešamās izmaiņas, kuras tika saskaņotas visas projekta komandas ietvaros (*izmaiņu plānošana*).

Otrajā fāzē tika nolemts, ka 4 sistēmas moduļus piegādās kompānija K_v, bet atsevišķas komponentes piegādās kompānija K_a. Testēšanas process tika vadīts no kompānijas K_v, bet pati testēšana un automatizētie regresa testi tika izstrādāti saistītajā kompānijā K_t. Turklāt, saistītā kompānija K_i veiks prasību analīzi izmaiņām kas skar jaunās valūtas ieviešanu sistēmā, bet izstrādes darbus šīm izmaiņām nodrošinās vadošā kompānija K_V. Otrajā fāzē visas kompānijas, kuras piedalījās dalītajā projekta izstrādē bija no Latvijas, lai mazinātu risku ar prasību specifikāciju nepareizu izpratni un liekiem tulkošanas darbiem. Visi projekta dokumenti, tai skaitā arī prasību specifikācija, projektā tika veikta latviešu valodā, jo to noteica līgumiska atruna ar programmatūras pasūtītāju.



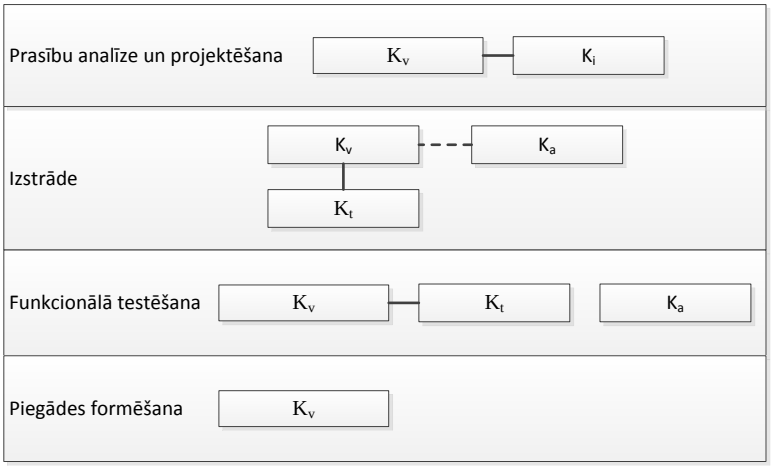
Attēls 4. Kompāniju sadalījums 2.fāzē

Kompāniju sadalījuma shēma otrai izstrādes fāzei attēlota attēlā 4, pārejot uz spējās izstrādes modeli:

- K_p – Kompānija, kas veic programmatūras pasūtījumu;
- K_v – Kompānija, kas slēdz tiešu līgumu ar pasūtītāju;
- K_a – Kompānija, kas līguma ietvaros ir kā saistītā kompānija K_v, bet ir noslēgusi atsevišķu līgumu (Attēlā 4 tiek ilustrēta ar raustītu līniju, lai identificētu sadarbību, kas pastāv ārpus maģistra darbā ietvertā projekta) ar programmatūras pasūtītāju. Šī kompānija veic atsevišķa moduļa izstrādi, to integrējot kopējā sistēmā, un veic piegādes caur kompāniju K_v;
- K_t – Saistītā kompānija kompānija K_v – nodarbojas ar programmatūras testēšanu atbilstoši K_v kompānijas izstrādātiem testa scenārijiem, regresa un automatizēto testu izstrādi kompānijai K_v, kura savukārt pie katras iterācijas piegādes veic to automātisku izpildi, lai izanalizētu nodevumus un konstatētu nevēlas izmaiņas jau piegādātajos programmatūras vienumos;
- K_i – Kompānijai, kurai tika izdalīts atsevišķs darba vienums – prasību aprakstīšanai papildus izmaiņai jau piegādātajiem moduļiem sistēmā. Izmaiņas izstrādi un piegādi kompānija neveic, jo to nodrošina K_v.

Darbu sadalījums projekta otrajā fāzē ir attēlots attēlā 5. Prasību analīzi un projektēšanas darbus veic divas kompānijas, savukārt pamata izstrādi veic 3 kompānijas. Interesanta iezīme ir,

ka kompānija K_t, kas veic testēšanu, veic arī noteiktus izstrādes darbus. Tas tādēļ, ka šī kompānija palīdz izstrādāt lielāko daļu automatizētos testus atbilstoši vadošās kompānijas K_v izstrādātajiem darba vienumu realizācijas aprakstiem.



Attēls 5. Projekta darbu sadalījuma shēma otrajā fāzē

Pēc līguma slēgšanas ar programmatūras pasūtītāju, tika izdalīti vairāki vienošanās protokoli, kuri savukārt iezīmēja nepieciešamās darbu pakas – atsevišķus darba vienumus izstrādei. Šādi sadalot darbus, tos ir vieglāk pārraudzīt un sekot līdzi darba vienumu izpildes trasējamībai projekta dzīves cikla laikā. Savukārt pašus darba vienumus var dalīt vēl mazākos darbos, izdalot noteiktus darbus saistītajām kompānijām.

Pēc šādas sadales metodes un iteratīvās izstrādes pieejas radās jautājumi, kā uzlabot izstrādes procesu, lai gatavā programmatūras piegāde notiktu nepārtraukti un bez liekām aizturēm, kuras rodas dēļ kompāniju attāluma vienai no otra, kas apgrūtina neskaidrību izrunāšanu un realizējamā darba vienuma pareizu izprašanu. Šādas aiztures programmatūras izstrāde radīja bieža darba vienumu atgriešana analīzē dēļ nepietiekama apraksta, jo programmētājs, kurš atradās saistītajā kompānijā nevarēja tik viegli vērsties pie analītiķa, kurš ir atbildīgs par doto darba vienumu pēc padoma. Arī saziņa telefoniski bieži nedeva vēlamus rezultātus, jo nevarēja šādi izskaidrot kā pareizāk realizēt aprakstīto darba vienumu.

5.1 Išikavas diagrammas izmantošana problēmu analīzē

Lai saprastu kādas problēmas pastāv projektā un lai varētu veikt to analīzi un pieņemt lēmumus par nepieciešamajām izmaiņām darba plūsmā, tika sasaukta īpaši izveidota darba grupa no vadošās kompānijas projekta komandas dalībniekiem. Darba grupa organizēja 5 sanāksmes,

sadalot projekta problēmas sekojošās daļās – Dokumentācija, Analīze, Testēšana, izstrāde un projekta kopējās problēmas.

Pielikumā 1 atainota cēloņu raksturojuma shēma projektā, kurā var skatīt identificētos riskus, kuri var radīt problēmas projektā, tādējādi ietekmējot projekta gala produkta kvalitāti. Kvalitātes grupas sanāksmes tika veiktas 4 iterācijās ar atsevišķu tematiku katrā tikšanās reizē:

1) Vispārīgie jautājumi, kā projekta pārvaldība, līgumu slēgšana;
2) Analīzes darbi, priekšizpēte, prasību apkopošana;
3) Izstrāde;
4) Testēšana.

Katrā no atzariem, balstoties uz Išikavas diagrammas zīmēšanas principiem [15] , ir iespējams pierakstīt riskus (projekta iespējamās problēmas), kas var radīt draudus projekta attīstībai ieviešot spējās programmatūras izstrādes metodes projektā, ja tajā iesaistītas vairākas kompānijas. Balstoties uz maģistra darbā ietvertā projekta otrās fāzes problēmām, ieviešot spējās programmatūras izstrādes modeli, projekta ietvaros tika paveikts darbs un izrunātas problēmas, kuras apkopoja, papildinot raksturojumu shēmu ar jau zināmiem riskiem.

Katra sanāksme sastāvēja no sekojošiem posmiem, balstoties uz Išikavas metodes aprakstiem materiālā [15] , pēc kuras tika zīmēta cēloņu raksturojuma shēma:

1) problēmu identificēšana, kuras ietvaros katram komandas dalībniekam tika izdalītas 10 piezīmju lapiņas, uz kā katrs sanāksmes dalībnieks sarakstīja sev svarīgos saskatītos dalīto projektu riskus;
2) problēmu apkopošana, kurā piezīmju lapiņas tika salīmētas uz tāfeles, dublikātus atmetot – pielīmējot lapiņas pie tāfeles, tika komandas ietvaros diskutēts, vai tā tiešām ir problēma visam projektam vai tikai kāda komandas dalībnieka individuāla vēlme;
3) problēmu analīze un raksturojumu shēmas papildināšana, kuras ietvaros katru piezīmju lapiņā minēto risku no tāfeles pārrunā komandas ietvaros un atzīmē, kā spēkā esošu. Rezultātā tika lemts par problēmas nozīmīgumu dalītajā projektā un tika papildināta kopējā raksturojumu shēma.

Veicot šādu pētījumu, tika panākta dalīto projektu problēmu apkopošana, kā arī to detaļu un nozīmīguma pakāpes projektā pārrunāšana. Rezultātā projektā darba grupas ietvaros tika izveidots piedāvājums, kādi uzlabojumi projektā ir nepieciešami, lai mazinātu risku iestāšanos.

Pēc darba grupas izskatīšanas un veiktajām korekcijām tika izstrādāts risinājums, kurš tika ieviests projektā.

Implementētais risinājums. Baltoties uz Išikavas diagrammā iniciētajām problēmām, *Action Research* iterācijas beigās projektā ir ieviesti sekojoši uzlabojumi darba procesā:

1) Līguma un vienošanās protokolu plānošana – līgums tiek slēgts uz ilgstošu laika periodu, kura ietvaros tiek sastādīti vienošanas protokoli ar nepieciešamo vienumu sarakstiem, kas savukārt tālāk identificē konkrētus darba vienumus. Šāda pieeja palīdz kopējo projektu dalīt vienmērīgās daļās, ieviešot noteiktas komponentes pakāpeniski. Savukārt darba vienumus vienošanās protokolos ir ērti dalīt pa projektam saistītajām kompānijām kā atsevišķas darbu pakas ar individuāliem darba uzdevumiem;
2) Darba vienumi kā pieteikumi tiek reģistrēti vienotā pieteikumu sistēmā, kura ir pieejama visiem projekta dalībniekiem, neatkarīgi vai tie atrodas vienā kompānijā vai strādā citā – saistītajā kompānijā. Darba vienumiem tiek noteikts vienots dzīves cikls, izejot cauri visām izstrādes stadijām, sākot no klienta pieteikuma, ievietošanu vienošanās protokolā līdz par uzstādīšanai produkcijas vidē;
3) Uz testiem balstīta izstrāde – veicot risinājuma aprakstu, tas tiek papildināts ar zaļā testēšanas ceļa piemērus, aprakstot pamata darbības, kā sistēmai ir jāuzvedas pēc risinājuma izstrādes;
4) Programmēšana pa pāriem – izstrādi veic viens programmētājs, bet pēc vienībtestu veikšanas, tiek lūgts citam programmētājam veikt koda caurskati (*Code Review*). Šāda pieeja ir efektīva, kad cita kompānija iesniedz savus labojumus un tos izskata kāds no vadošās kompānijas programmētājiem. Ar šādu pieeju bieži tiek izķertas gan programmēšanas kļūdas, gan programmēšanas stila kļūdas;
5) Mazi laidieni un dalījums par iterācijām – formējot iknedēļas piegādes ar mazākiem apjomiem, ir vieglāk veidot piegādes, apvienojot gatavos darba vienumus no dažādām ārējās kompānijām. Tas tiek nodrošināts ar nodevuma sagatavotāju (*Build Master*), kas uz noteiktām versijām sagatavo notestētās izmaiņas;
6) Kopējs repozitorijs programmatūras kodam – izveidots kopējs repozitorijs ar piekļuvi gan iekšējiem, gan ārējiem lietotājiem. Repozitorijs nodrošina izmaiņu atsekošanu un precīzu versionēšanu. Svarīga ir trasējamība uz darba vienumiem, lai nodevuma sagatavotājs (*Build Master*) spētu no repozitorija izveidot nodevuma atzaru;

7) Kopējs repozitorijs dokumentācijai – vienota pieeja aktuālo un iepriekš nodoto dokumentu versiju glabāšanai. Pieeja repozitorijam nodrošināta gan iekšējiem, gan ārējiem lietotājiem.

5.2 Darba uzdevumu aprakstu veidošana

Programmētāji otrās fāzēs sākumā bieži sūdzējās par nepilnvērtīgiem darba vienumu realizācijas aprakstiem. Bieži tika konstatēti gadījumi, kad sistēmu analītiķis iedeva programmētājiem darba vienumu izstrādei bez apraksta. Vadošās kompānijas ietvaros tas lielas problēmas neradīja, jo programmētājs varēja griezties pie analītiķa un izrunāt nepieciešamo implementāciju. Savukārt, ja darba vienums tika nodots izstrādei saistītajām kompānijām, bija problemātiski izrunāt risinājumu un tas atspoguļojās kļūdās nodevumā. Kļūdas bija funkcionālas un atspoguļoja pārprastu prasību realizācijai. Tādējādi projektā tika meklēti veidi kā uzlabot darba vienumu aprakstus, lai jebkurš programmētājs varētu uzsākt pie tā darbus, pēc iespējas mazāk vēršoties pie analītiķa, lai precizētu prasības.

Sanāksmē starp analītiķiem, testētājiem un programmētājiem tika izvirzīti vairāki iespējamie risinājumi:

1) Kopēja realizācijas apraksta šablona izveide, kas nodrošinātu vienādu pieeju realizācijas aprakstiem, ietverot sevī visus punktus, kas ir nepieciešams realizēt kā arī nepieciešamo izstrādājamo formu skices vai izmaiņas;
2) Izmaiņas veikt projektējumā un pie darba vienuma pieteikumu sistēmā kopēt līdz projektējumu.

Izrunājot visus šos risinājumus, šablona izveide, ko obligāti būtu jāpiemēro visiem realizējamiem vienumiem netika implementēta, jo analītiķi saskatīja, ka tādējādi būtu jāveic dubults darbs, aprakstot realizācijas aprakstu un pēcāk veikt izmaiņu iestrādi projektējumu dokumentos. Savukārt otrā ideja tika noraidīta, jo projektējumu dokumenti pētāmajā projektā ir samērā lieli un lielākā daļā sasniedz līdz par tūkstotim lappušu un ja izmaiņas tiek aprakstītas tikai nelielā modulī, tad sanāk uzturēt līdz visu projektējumu. Tas arī apgrūtinātu programmētājam ātri atrast vietu, kur ir aprakstīta realizējamā izmaiņa sistēmā. Kā arī šāda pieeja neatvieglotu darbu testētājiem, jo izmaiņas bieži vien bija „*izmētātas*" pa visu projektējuma dokumentu. Papildus, ja darba vienums būtu jādod citai kompānijai izstrādāt, tās programmētāji saņemtu lieku un noteiktai izmaiņas izstrādei nevajadzīgu informāciju. Kā arī līgumiski nedrīkst

sūtīt visu projektējumu, jo tam ir ierobežota pieeja starp vadošo kompāniju K_T un klienta kompāniju K_K.

Implementētais risinājums. Balstoties uz abiem risinājumiem, projektā, sasaucot otro sanāksmi par šo problēmu, kā risinājums tika pieņemta jauna ideja. Lai uzlabotu darba vienumu aprakstus un šos darba vienumus varētu nodot nepieciešamības gadījumā izstrādei arī citai saistītai kompānijai, kā arī uzlabot testēšanas procesus, tika noteikts katram darba vienumam veikt realizācijas aprakstus vai minēt norādes uz projektējumu. Ātrāk darba vienumu nedrīkst novirzīt no analīzes fāzes uz izstrāde, kamēr nebija darba vienumam pievienots:

1) Realizācijas apraksts, ja darba vienums ir izmaiņa;
2) Izvilkums no projektējuma, kas ietver funkcionālos apgabalus ar nepieciešamajās izmaiņām komponentē. Izvilkumam no projektējuma obligāti ir jāsatur indikatori uz mainītajām vietām;
3) Atsauce uz projektējumu, sadaļu, kurā ir aprakstīta izmaiņa.

Šādi izstrādei nodoti darba vienumi, palīdzēja programmētājam efektīvāk saprast programmējamo apgabalu, kā arī noteikt kā precīzi ir jāizstrādā jaunās komponentes un kā precīzāk ir jāveic izmaiņas esošajās. Savukārt testētājam, kas pētāmajā projektā atradās saistītajā kompānijā K_t uzreiz varēja izlasīt veiktās izmaiņas komponentē un uzsākt testēšanu bez analītiķa līdzdarbības un papildus skaidrojumiem.

Kā vēlāk atklājās, šāds risinājums nesa vēl papildus ieguvumus izstrādes procesā. Analītiķis, kurš bija aprakstījis risinājumu kā projektējuma izmaiņas, viegli tās varēja pārnest atpakaļ projektējumā un daudz ātrāk sagatavot dokumentu nodevumu, jo nebija otrreiz jāapraksta izstrādātais modulis sistēmā.

5.3 Nepareizs darba vienumu dzīves cikls

Testēšana otrajā projekta fāzē tika organizēta caur saistīto kompāniju K_t, kura bieži vien iterācijas izstrādes vidū atradās dīkstāvē vai zem ļoti mazas darbu noslodzes. Savukārt tuvojoties iterācijas piegādes laikam – darba apjoms līdz ar to palielinājās ar tādu apmēru, ka saistītā kompānija K_t vairs netika galā ar testēšanas pienākumiem un apmēram puse no piegādātajiem darba vienumiem tika testēti pēc iterācijas piegādes klientam akcepttestēšanas vidē. Tas ietekmēja nodevumu kvalitāti, jo apmēram puse no piegādē ietvertajiem darbiem nebija pilnvērtīgi notestēti. Analītiķis veica darba vienuma zaļo testu un tikai tad virzīja pieteikumu uz funkcionālo testēšanu. Šāda plūsma radīja problēmas uz operatīvu darba vienumu testēšanu, jo

bieži vien analītiķis nespēj uzreiz blakus citiem darbiem veikt operatīvu testēšanu pēc izstrādes. Šāda nepareiza pieeja sāka kavēt izstrādes procesus, jo darba vienumi bieži ilgstoši palika vienā statusā pie analītiķa uz testēšanu un pie testētājiem nonāca tikai pēdējā iterācijas nedēļā, kad tika formēta piegāde.

Lai uzlabotu testēšanas procesus, tika sasaukta kopā kvalitātes grupas sanāksme, kurā pieaicināja gan sistēmu analītiķus, gan programmētājus un arī testētājus. Sākotnēji tika diskutēts par to, kādēļ pēc programmatūras vienuma izstrādes veikšanas, tas ilgstoši paliek pie analītiķa uz zaļā veļa testa veikšanu. Atklājās, ka dēļ tā, ka analītiķim ir daudz citu darbu darāmu paralēli – gan esošo darba vienumu izstrāde, gan priekšizpēte jauniem darbiem gan jauno darbu projektēšana un realizācijas aprakstu veikšana, tad, vienkārši, sistēmu analītiķim pietrūkst laika, lai operatīvi veiktu zaļā ceļa testēšanu atrisinātajam darba vienumam. Zaļā ceļa testēšanas analītiķiem projektā tika sākotnēji noteikta, jo testēšanas saistītā kompānija K_T nevarēja uzreiz saprast funkcionālo biznesa procesu un nevarēja pilnvērtīgi veikt testēšanu.

Implementētais risinājums. Zemāk attēlotajā tabulā 2 ilustratīvi var redzēt iepriekšējo darba vienumu dzīves ciklu un dzīves ciklu pēc izmaiņu veikšanas projektā.

Tabula 2. Darba vienumu dzīves cikla izmaiņas Projektā

Darba vienumu dzīves cikls pirms izmaiņu ieviešanas					
Analīzē	Izstrādē	Atrisināts	Testēšanā	Gatavs piegādei	Piegādāts
Darba vienumu dzīves cikls pēc izmaiņu ieviešanas					
Analīzē	Izstrādē	Testēšanā		Gatavs piegādei	Piegādāts

Pateicoties ieviestajām izmaiņām projektā par realizācijas aprakstu veidošanu katram darba vienumam vai atsauces veidošanu uz projektējumu, testētāji saņēma pilnvērtīgus realizācijas aprakstus izmaiņām funkcionalitātē, uz kuras varēja veikt pilnvērtīgu testēšanu un testa scenāriju izveidi. Ja ir gadījumi, ko testētāji nevar notestēt, ir iespēja lūgt vadošās kompānijas sistēmu analītiķu palīdzību un ar viņa atbildību virzīt darba vienumu sagatavošanu piegādei.

Pēc veiktajām izmaiņām, no sistēmu analītiķa tika noņemta atbildība par testēšanas procesu, līdz ar to darba vienumus varēja operatīvi pēc izstrādes veikšanas atdot saistītai kompānijai K_t, kas nodrošināja projekta testēšanas soļus. Analītiķim joprojām palika tiesības

veikt zaļā ceļa testēšanu, bet no viņa tika noņemta atbildība par testēšanas rezultātu. Testētājs atbilstoši darba vienuma realizācijas aprakstam vai projektējumam, veic pilnu funkcionālo testu un sastāda scenārijus integritātes testiem. Testētājam tika noteikts pienākums veikt testēšanu uzreiz, tiklīdz tas nodots testēšanai, izvēloties darba vienumus testiem, grupējot tos pēc pieteikuma prioritātes pakāpes (Kritisks, Augsts, Vidējs, Zems) un sakārtojot tos pēc pieteikuma statusa maiņas datuma, laika, kad pieteikums ir nodots testēšanai. Vēl kā papildus nosacījums tika prasīts, ka darba vienumam ir jābūt testēšanā ne vēlāk kā 2 dienas pirms iterācijas piegādes klienta akcepttesta videi. Lai izmaiņas stātos spēkā, bija nepieciešams mainīt pieteikumu kontroles sistēmā, kuru izmanto visas saistītās kompānijas, darba plūsmu. Kā arī analītiķim tika dota pieeja kontrolēt pieteikumu dzīves ciklu, mainot statusus uz sev vēlamo. Tas tika nodrošināts, jo tika konstatēti darba vienumi, kurus sistēmu analītiķis varēja atrisināt pats, nevis to aprakstot un dodot izstrādei programmētājiem.

5.4 Pāra programmēšanas principu izmantošana

Pirms šī cikla bija vērojamas problēmas ar programmatūras piegādes kvalitāti. Tā kā programmatūras izstrādi veica dažādi programmētāji un programmētāji arī no citām saistītajām kompānijām, radās problēmas ar izstrādātā koda atbilstību izstrādājamās sistēmas standartiem, kā arī bieži tika pieļautas gan neuzmanības kļūdas, gan arī funkcionālās kļūdas, radot tālāk problēmas testēšanas procesiem, jo nācās bieži atgriezt darba vienumus atpakaļ izstrādē.

Viens no spējas izstrādes pamatprincipiem nosaka programmēšanu pāros, veidojot programmatūras kodu kopīgi. Bet dalītajos projektos, kur programmētāji sēž dažādās kompānijās attālināti viens no otra, šādu pieeju būtu grūti īstenot. Tādēļ pētāmajā projektā tika domāts cits risinājums, kā panākt programmatūras koda kvalitātes uzlabojumus.

Lai izdomātu risinājumu, tika sasaukta sanāksmes starp analītiķiem, izstrādes grupas vadītāju un izstrādes grupā esošajiem programmētājiem. Sanāksmē tika izklāstīta problēma un tad domāti iespējamie risinājumi, kā panākt kvalitāti darba vienumu piegādēs. Kā iespējamie risinājumi tika izteikti sekojošie varianti:

1) Izstrādāt projektā programmatūras izstrādes standarta aprakstošo dokumentu, kas ietvertu programmatūras izstrādes vadlīnijas un pieraksta stilu;
2) Izstrādātajiem programmatūras vienumiem veikt un aprakstīt vienībtestu, lai pats programmētājs varētu novērtēt kvalitāti savam izstrādātajam programmatūras kodam;

3) Vadošajam programmētajam pārskatīt programmatūras kodu, pirms iterācijas piegādes nodevuma veidošanas.

Sanāksmē, izvērtējot pirmos 2 variantus, tos konstatēja kā neefektīvus, jo standarta izstrāde prasītu papildus resursus, kas projektā jau tā bija ierobežoti, kā arī standarta apraksts būtu vienpusējs skatījums kā labāk rakstīt programmatūras kodu un tas jebkurā gadījumā būtu jāpārskata citiem un jāpapildina. Šāds risinājums arī nenesīs uzreiz nekādu ieguvumu, bet gan pēc zināma laika posma, kad dokuments būs pabeigts un to būs izskatījuši pārējiem izstrādes grupas dalībnieki. Tā ietvaros tika aprakstīts neliels dokuments, kas saturēja instrukcijas – priekšrakstus kā rakstīt programmatūras kodu aplikācijai un datu bāzei – ar domu, lai jaunie darbinieki vai saistīto kompāniju darbinieki to varētu izmantot kā sākuma avotu, uzsākot izstrādes darbus projektā.

Savukārt otrais variants, kurā tika apspriesta ideja par vienībtestu veidošanu pēc divu nedēļu implementācijas tika izņemts, jo prasīja pārāk daudz laika programmētājam, lai veidotu izstrādāto metožu vienībtestu scenārijus un papildus veiktu testēšanu, meklējot dažādus variantus kā notestēt izstrādāto programmatūras kodu. Bieži vien vienībtestēšana nelieliem risinājumiem prasīja vairāk laika nekā pati izstrāde un tas stipri ietekmēja darba vienumu darbietilpību novērtējumus.

Implementējot projektā trešo potenciālo risinājumu – programmatūras nodevumu kvalitāte pieauga, jo vadošais programmētājs bieži konstatēja nepilnības koda uzbūvē un deva atpakaļ programmētājiem izstrādē. Bet pēc dažu nedēļu šādas darbības tika konstatēts, ka vadošais programmētājs uz iterāciju slēgšanas brīdi vairs nespēj tikt galā ar piegādāto programmatūras kodu izskatīšanu. Tādēļ risinājums tika mainīts, lai atslogotu vadošā programmētāja darbu.

Implementētais risinājums. Mainot trešo risinājumu, tika nolemts, ka pēc programmatūras koda izstrādes, programmētājs pieteikumu sistēmā izziņo par pabeigtību un piefiksē repozitorija ievietošanas numuru. Darba vienums pēc tam tika nodots citam programmētājam, kuram tai brīdī ir mazāka noslodze un var veikt programmatūras koda izskatīšanu. Tādējādi jebkurš repozitorijā ievietotais programmatūras kods tika izskatīts un neradīja papildus slodzi uz projektā pieejamiem resursiem. Kā arī, šāda pieeja efektīvi strādāja arī citu saistīto kompāniju piegādātajiem programmatūras vienumiem, jo nebija svarīgi projektā, kurš izskatīja sarakstīto kodu, bet lai koda apskats noteiktu tiktu fiksēts pirms vienuma iekļaušanas iterācijas nodevumā. Pie reizes varēja vēlot vēl vienu labu iezīmi – šāda pieeja veicināja programmatūras koda izstrādes pieredzes

apmaiņu starp programmētājiem, kas savukārt veicināja labāku programmatūras kodu rakstīšanu, pozitīvi atsaucoties uz nodevumu kvalitāti.

5.5 Ikdienas sanāksmju ieviešana iterācijā

Izstrādājot iterācijas pētāmajā projektā, katras iterācijas sākumā tika izdalīti darba vienumi un tika uzsākta to realizācijas apraksta veidošana, izstrāde un tālāk jau testēšana. Process bija nekontrolēts un bieži noveda pie tā, ka pēdējā iterācijas nedēļā sakrājās daudzi vēl nepadarīti darbi, it īpaši tie, kam bija jāsāk jau testēšanas process.

Implementētais risinājums. Lai novērstu nekontrolētu izstrādes procesu un katru dienu jebkurš projekta dalībnieks saprastu, ko viņš jau ir paveicis un kas vēl ir jāizdara līdz iterācijas slēgšanai, tika uzsākts mēģinājums rīkot *Scrum* sanāksmes. Projektā *Scrum* sanāksme tika organizēta vienuviet sapulcējoties vadošās kompānijas izstrādes komandas dalībniekiem pie tāfeles, uz kuras bija sarakstīti visi uz iterāciju ieplānotie veicamie darbi. Tā kā atsevišķus darbus izstrādāja saistītā kompānija, tad arī šie darbi tika uzskaitīti pie tāfeles. Saistīto kompāniju darbus pārrauga par iterāciju atbildīgais sistēmu analītiķis, kas arī katru dienu *Scrum* sanāksmes laikā pasaka šī darba vienuma statusu un plānoto izdarīt pašreizējajā dienā. Saistītajā kompānijā izstrāde arī notiek, katru dienu rīkojot *Scrum* sanāksmes un projekta statuss tika nodots vadošās kompānijas par iterāciju atbildīgajam sistēmu analītiķim, īstenojot *Scrum* no *Scrum* pieeju iterācijas statusa nodošanai dalīto projektu izstrādē.

Scrum sanāksmes ilgums projektā ir mainīgs un svārstās robežās no 10 minūtēm līdz pat pusstundai, jo papildus statusa došanai šajā momentā tika izrunātas arī problēmas, ja tādas pastāv un risinājumi, kā tās novērst. Lai padarītu *Scrum* sanāksmes interesantākas, par katru neizpildi atbilstoši gada sezonas tematikai tika zīmēti atbildīgajam par darba izpildi zīmējumi. Kad zīmējums tika pabeigts, tas nozīmēja, ka atbildīgais darbinieks dēļ savas neizdarības ir savācis pilnu punktu skaitu un nākamajā projekta saliedēšanas pasākumā viņam ir jāuzsauc komandai pica. Parasti iterācijā no 30 cilvēku komandu tika savāktas kādas 5 picas.

5.6 Otrās fāzes slēgšana

Darba tapšanas brīdī, projekta otrā fāze ir slēgšanas stadijā, kad piegādās komponentes tika uzstādītas produkcijas vidē, tādējādi ļaujot veikt pārskatu un salīdzināt piegādāto darba vienumu un kļūdu daudzumu pret pirmo fāzi, kad izstrāde projektā tika veikta pēc ūdenskrituma izstrādes principiem. Projektā ieviestās spējās izstrādes metodes ļāva izstrādi veikt daudz efektīvāk, samazinot uz projekta izstrādi nepieciešamās izmaksas.

Pirmajā gadā, kad projekts strādāja pēc ūdenskrituma modeļa, darāmo darbu apjoms bija mazāks, jo piegādes netika veiktas iteratīvi un jauni papildus darbi līguma ietvaros netika pasūtīti no klienta puses. Savukārt pēc izstrādes, piegādājot izstrādāto programmatūru uz šo nelielo darba vienumu daudzumu, tika reģistrēts ļoti daudz kļūdu. Zināmā mērā par iemeslu šādam kļūdu daudzumam bija arī neprecīzi definētas prasības, līdz ar to izstrādātā programmatūra nenodrošināja klientam pareizu biznesa darbību. Tādēļ lielākā daļa no kļūdām tika identificētas kā analīzes kļūdas un nācās prasības pārstrādāt. Tikai daļa no pieteiktajām kļūdām tika identificēti kā jauni izmaiņu pieprasījumi, bet tas kopējo kļūdu statistiku neuzlaboja, un, tā kā izstrādes darbi netika veikti iteratīvi, tad, lai realizētu šīs izmaiņas, kompānijai bija jāslēdz jauns līgums ar programmatūras pasūtītāju un uz tā realizāciju klientam bija jāgaida. Jauns līgums ar jauniem darbiem nozīmēja jaunu ciklu ūdenskrituma modelī, sākot ar prasību analīzi, specificēšanu, projektēšanu.

Otrajā gadā, kad ar klientu tika atrunāta iteratīvā piegāde, situācija krasi uzlabojās, jo pateicoties biežām piegādēm, klients jau savlaicīgi varēja testēt piegādātās izmaiņas akcepttesta vidē un veikt nepieciešamās korekcijas. Lai gan, ieviešot globālās izmaiņas, projekta kompānija atteicās no abu saistīto ārvalstu kompāniju darbiem, atstājot izstrādi tikai vienai saistītai kompānijai Latvijā un testēšanu otrai saistītai kompānijai Latvijā. Līdz ar to, kad tuvojās jauno darbu nodošanas termiņš, lielākā daļa pamata problēmu bija novērstas un kļūdas vairāk atspoguļojās nevis analizē, bet jau pašā programmatūrā. Protams, kļūdas arī šajā gadā bija daudz, bet tas tikai tādēļ, ka joprojām nebija sakārtoti iekšējie izstrādes procesi projektā. Trešajā projekta attīstības gadā (2013.gads), turpinoties otrai fāzei, kļūdu skaits samazinājās, jo pateicoties spējās izstrādes ieviestajām metodēm, izstrādes process kļuva daudz kvalitatīvāks, kas atspoguļojās arī iterāciju nodevumos.

Kompānija ievēroja atšķirības starp iteratīvām piegādēm un ierasto izstrādes modeli. Tādēļ, izmantojot projektu kā pamatu, kompānijā tika izstrādāts un prezentēts kopējs spējās izstrādes modelis, kura mērķis bija attēlot, kā visvieglāk standarta ūdenskrituma izstrādes modeli pārbūvēt tā, lai izstrādes darbi tiktu veikti iteratīvi, lai varētu esošos projektus vieglāk adaptēt jaunajām izmaiņās izstrādes procesā.

Tā kā projekts bija paradījis pozitīvus rezultātus, tad, lai iekļautu kvalitātes aprakstos iteratīvo pārejas modeli, tika rīkota izstrādes vadītāju sanāksme, kurā tika prezentētas iteratīvās izstrādes iespējas, kuras varēja pielietot arī dalīto projektu izstrādē. Sanāksmē piedalījās sekojoši dalībnieki: 1 darbības izcilības eksperts, 2 daļas direktori, programmatūras risinājuma realizācijas

dienesta direktors, 2 nodaļas vadītāji, 5 izstrādes grupas vadītāji, vadošais sistēmu arhitekts, 1 projektu vadītājs, 2 vecākie sistēmu analītiķi un kvalitātes uzlabošanas grupas darbības izcilības eksperts. Darba autoram izklāstot pielikumā 2 uzzīmēto izstrādes modeli un to, kā tas adaptējas esošajā izstrādes modelī, rezultātā tika apstiprināts jauns izstrādes modelis, kas attēlots pielikumā 3. Pielikuma 3 attēlā redzams, ka esošais izstrādes modelis tika papildināts ar iteratīvu piegādes ciklu, kas nodrošināja uz iepriekš saskaņotām prasībām iteratīvu programmatūras dizainu jeb projektēšanu, izstrādi, testēšanu un piegādi klienta akcepttestēšanas videi.

Maģistra darba ietvaros pētītais projekts darba tapšanas brīdī ir nonācis otrās fāzes slēgšanā, kas ir kā pārejas posms starp otro fāzi un trešo fāzi. Līdz ar iteratīvās izstrādes ieviešanu, otrās fāzes laikā ir bijuši vairāki iepriekš aprakstīti uzlabojumi, bet ar to ir nepietiekami, jo fāzē izstrādātā komponente tika nodota un ieviesta produkcijā tomēr ar kavējumu. Tādēļ, darba turpinājumā, tiks prezentēts literatūras apskats, kurš veikts ar mērķi labāk izprastu procesus dalīto projektu izstrādē un saplānotu nepieciešamās izmaiņas trešajai projekta fāzei.

6 Literatūras analīze par spējo izstrādi dalītajos projektos

Spējās izstrādes pieejas izmantošana dalīto projektu izstrādē strauji gūst popularitāti, jo aizvien vairāk kompāniju izmanto iespēju izmantot tā sniegtās priekšrocības [23] , kā piemēram, samazinot izstrādes izmaksas, izmantojot darba resursus no atšķirīgām laika joslām, iznomājot resursus no citām kompānijām vai arī pārdodot savus resursus uz laiku citas kompānijas projektiem [16] . 2008.gada pētījumā [1] , ko veicis uzņēmums VersionOne, 57% no aptaujātiem uzņēmumiem norādīja, ka viņu projekti ir dalīti un 41% no tiem ir domājuši vai plānojuši par spējās programmatūras izstrādes ieviešanu šādos projektos. Šis pētījums uzrāda, ka lielākā daļa no dalītajiem projektiem vēl plāno, kā ieviest spējās programmatūras pieeju un attīstīt izstrādes procesus.

Lai saprastu, kādas spējās izstrādes metodes visbiežāk tiek izmantotas dalīto projektu izstrādē, tika analizēts autoru Džalali un Volina veiktais literatūras apskats [3] . Literatūras apskatā autori ir veikuši literatūras analīzi ar dalīto projektu izstrādi saistītiem 77 pētnieciskajiem darbiem. Analīzes rezultātā, balstoties uz metožu aprakstu biežumu materiālos, tika apkopotas sekojošas visbiežāk dalīto projektu izstrādē izmantotās spējās izstrādes metodes [3] :

1) Iteratīvā izstrāde (18 materiāli);
2) *Scrum* sanāksmes (16 materiāli);

3) Programmēšana pāros (14 materiāli);
4) Retrospektīvās sanāksmes (11 materiāli);
5) *Scrum* sanāksmes no *Scrum* (9 materiāli);
6) Uz testēšanu balstīta izstrāde (9 materiāli);
7) Sprinta pārskata sanāksmes (8 materiāli);
8) *Backlog* darbu saraksta izmantošana (7 materiāli);
9) Automatizētā testēšana (7 materiāli);
10) Plānošanas sanāksmes (7 materiāli);

Problēmas spējā programmatūras izstrādē rada iteratīvās izstrādes pieejas nodrošināšana, projekta pārvaldes procesu uzturēšana [1] . Visbiežāk problēmas dalīto projektu izstrādē rada apgrūtināta komunikācija starp vairākām kompānijām, plānojot sprintus, pārskatot sprinta piegādes [1] . Apgrūtinoša ir arī informācijas apmaiņa, gadījumos, kad iterācijas izstrādes vidū ir nepieciešams veikt papildus korekcijas izstrādājamai komponentei, novēršot neparedzētus riskus, labojot kļūdas vai izstrādājot nepieciešamos papilinājumus. [1] . Tādēļ, lai izprastu, kādā veidā ir nepieciešams organizēt darbus dalītajā projektā, turpmākajās nodaļās, balstoties uz literatūras avotiem, tiks aprakstīti spējās izstrādes posmi, sākot ar lietojumu gadījumu veidošanu, turpinot ar darba vienumu identificēšanu, komandas izveidi, iterāciju organizēšanu, programmatūras testēšanu un piegādes veidošanu.

6.1 Lietojumu gadījumi (*Use Stories*)

Lietojumu gadījumi ir nepieciešami prasību analīzei un sākuma dokumentācijas izveidei. Ūdenskrituma modelī parasti tiek aprakstīti biznesa procesi funkcionālajām prasībām. Tie tiek saskaņoti, un uz to bāzes tiek veikta programmatūras arhitektūras izveide, projektēšana un izstrāde. Spējās programmatūras izstrādē tie tiek aizstāti ar vienkāršāku, bet elastīgāku pieeju. Lietojumu gadījumi parasti ir nelieli biznesa darbību apraksti, ko klients apraksta ar vēlmi, lai tos varētu realizēt sistēmā. Lietojumu gadījumi nepavisam nedefinē gala prasības, bet nosaka pirmās iezīmes kā sistēmai būtu jāstrādā.

Lietojumu gadījumiem projekta ietvaros ir jābūt pieejamiem visiem dalītā projekta dalībniekiem. Tas nodrošinās labāku izpratni par topošo sistēmas biznesa procesiem. Ir dažādi rīki, kā uzturēt prasības, piemēram, kompānijas Attlassian rīks *Confluence* [4] , kas nodrošina dalītā projekta komandai pieeju vienuviet aprakstošai sistēmas dokumentācijai. Kā arī nodrošina

komunikāciju starp visiem projekta dalībniekiem, neatkarīgi no to atrašanās vietas, kā arī atbalsta darba vienumu komentēšanu un pat savstarpēju diskusiju grupu veidošanu.

Vadot dalīto projektu, svarīgi ir ievērot katra lietojumu gadījuma trasējamību un, lai no lietojumu gadījumiem tiem veidotos darbu sadalījums, tālāk ir jāveido veicamo darbu saraksts, veidojot *Product backlog*. Tas ir nepieciešams, lai novērtētu projekta potenciālās darbietilpības un izmaksas. Ar tādiem rīkiem kā Jira, var nodrošināt projekta darba vienumu uzskati un trasējamību cauri visam projekta dzīves ciklam.

Lai nodrošinātu prasību efektīvu nodošanu un apmaiņu starp dalītā projekta izstrādes grupām, var izmantot arī citus saziņas līdzekļus, ar kuriem var apmainīties ar informāciju – e-pastu, tiešās saziņas programmas (*Skype [18]* , *Messenger [19]*), versiju kontroles rīkus (piemēram *Subversion[20]* , *Git*). Ja izstrādes līgums ar pasūtītāju nepieļaut izņēmuma gadījumus, nevajadzētu rīkus, ar kuru ir viegli publicēt prasības un pieļaut prasību nokļūšanu nepareizajās rokās. Tādēļ labāk neizmantot sociālos tīklus un citas koplietošanas interneta vietnes kā zināšanu apmaiņas vietas starp projekta dalībniekiem.

Lietojumu gadījums parasti sastāv no diagrammas (*Use Case diagram*) un tās detalizēta apraksta. Projektā no lietojumu gadījumiem tika veidota kopējā prasību specifikācija, kurā lietotāju gadījuma diagrammas apraksta jau noteiktus sistēmas biznesa scenārijus, kā sistēmai ir jāfunkcionē pie dažādām gadījuma situācijām. Projektā veidotā sistēma sastāv no vairākām komponentēm, tādēļ lietotāju stāsti tika veidoti atsevišķi katrai no tām. Jebkurš lietojumu gadījums tika identificēts ar noteiktu kodu un sastāvēja no sekojošām sadaļām. Svarīgi ir katru darba vienumu identificēt, jo tālāk nodot darba vienumu citai kompānijai, vieglāk ir veidot references uz darāmiem darbiem. Katram lietojumu gadījumam tiek noteikta sava prioritāte projektā, kas atvieglo projekta darbu sadalījumu, nosakot, kurš darbs ir darāms primāri, kurš sekundāri. Lietotāju gadījumi tiek saistīti ar priekšnosacījumiem un beigu situāciju aprakstiem. Tas nodrošina caurskatāmību projektā un atvieglo biznesa plūsmas shēmas zīmēšanu.

Lietojumu gadījumu veidošanas ir sākums projekta izstrādē, un to apkopošanu var veikt vadošā kompānija, kā arī notika maģistra darba ietvertajā projektā. Lietojumu gadījumi ir pirmsākums tam, lai iniciētu darbu pakas, darbu vienumus un nodrošinātu prasību trasējamību cauri visam projekta dzīves ciklam.

6.2 Darbu pakas, darbu vienumi un to sadalījums

Balstoties uz klienta akceptētiem lietotāju stāstiem, tiek veidoti nepieciešamo darbu saraksti, kuri iedalās darbu pakās un darbu vienumos:

1) Darbu paka – noteiktas biznesa plūsmas apkopojums;
2) Darba vienumi, kas ir jāveic, lai nodrošinātu procesus biznesa plūsmā.

Biznesa un sistēmu analītiķu darbs ir veikt darbu apkopojumu, balstoties uz biznesa plūsmu diagrammām un iniciēt jau konkrētus darba vienumus, uz kuru bāzes tālāk var izdalīt darāmos darbus. Dalīto projektu izstrādē svarīgi ir panākt tādu darba dalījumu, lai tos varētu viegli nošķirt vienu no otra un vienmērīgi spēt dalīt pa kompānijām, kas veic projekta izstrādi. Viena no iespējamām metodēm, kā veikt pilnvērtīgu darba sadalījumu pa projektā iekļautajām kompānijām, ir veikt sistemātisku kompānijā esošu resursu izpēti.

Lamerdorfs un Munčš [5] sava pētījuma ietvaros apraksta, ka var izmantot 3 dažādas pieejas darbu paku un vienumu dalīšanai dalītajos projektos, integrējot tās projektā:

1) Riska identifikācijas modeli, kas darba vienumu sadali balsta uz riska pārvaldību projektā;
2) Uzdevumu piešķiršanas modeli, bāzēts uz priekšlikumiem, kas darba vienumu sadali balsta uz projekta dalībnieku izvirzītiem priekšlikumiem;
3) Nepieciešamo izmaksu modeli, kas darba vienumu sadali balsta uz paveicamā darba vienuma potenciālajām izmaksām.

Efektīvu rezultātu var panākt, ja izmanto projektā katru no modeļiem. Darba ietvaros izvēlētā projekta ietvaros pamatā tika izmantots nepieciešamo izmaksu modelis un maz tika pētīti papildus riski, kā arī netika veicināta priekšlikumu došana no projekta dalībniekiem. Rezultātā tas noveda pie papildu izmaksu nepieciešamības, jo ārējo izstrādātie darba vienumi bija daļēji jāpārstrādā, lai tos varētu integrēt kopējā sistēmā. Dalīto projektu izstrādē ir svarīgi izmantot katru no šiem modeļiem integrēti, lai darba vienumus vai darba pakas varētu dalīt pa ārējiem projektiem ar pietiekami zemu riska pakāpi, nepārsniedzot projekta kopējās izmaksas. Darba turpinājumā sekos ieskats katrā no šiem trim modeļiem, balstoties uz [5] pētījumu.

6.2.1 Risku identifikācijas modelis

Darba [5] autori apraksta, ka katram modelim ir nepieciešama individuāla pieeja risku pārvaldības procesiem, risku identificēšanai, bet visi var izmantot iepriekšējos projektos gūto pieredzi un bāzēties uz to, novērtējot riskus jaunajiem darba vienumiem. Novērtējot riskus, ir

jāapkopo iepriekš gūtā pieredze, ir jānovērtē ietekmējošie faktori un riski un tādējādi jāizstrādā riska modelis, kā rezultātā var definēt nosacījumu, kāda problēma var iestāties, nostrādājot kādam no faktoriem vai riskiem. Tabulā 3 tiek novērtēts risks darba Projektā, izstrādājot riska modeli turpmākajiem projektiem.

Tabula 3. Riska modeļa izstrādes piemērs projektā

Savāktā pieredze	• *Scrum* ikdienas sanāksmju organizēšana, dēļ saziņas grūtībām, izslēdzot ārējos projektus; • Apgrūtinoša informācijas apmaiņa ar ārējiem projektiem citās valstīs; • Prasību definēšana nacionālajā valodā.
Ietekmējošie faktori un riski	Faktori: • Saziņas valodu zināšanu trūkums; • specifikāciju un projektējumu apraksti nacionālajā valodā • neprecīzi veikti tulkojumi. Riski: • nekvalitatīva izstrāde (nav atbilstoša izvirzītajām prasībām); • darba vienuma darbietilpības pārsniegšana.
Nosacījums, izteikts teksta formā	Ja darba vienumu apraksti ir definēti nacionālajā valodā, ārējiem projektiem, kas atrodas ārpus projekta izstrādes valsts, ir grūtības tās pilnvērtīgi izprast un sistēmu analītiķis ir jāveic papildus izskaidrojošs darbs, lai panāktu vēlamo darba rezultātu.
Loģiskais gala nosacījums	Ja prasības ir definētas nacionālajā valodā, tad projekta atrašanās citā valstī noved pie darba vienumu izpratnes problēmām, kas savukārt veicina papildus izmaksas.

Ar šādu pieeju ir nepieciešams projekta plānošanas fāzē identificēt pēc iespējas vairāk riskus, tādējādi jau priekšlaicīgi nosakot riskus un to veicinošos faktorus projektā. Tas ne tikai atvieglo projekta darba plānošanu, nosakot, kurus darbus var atdot izstrādei citai kompānijai, bet arī jau priekšlaicīgi nosakot potenciālos draudus projektam, kā arī ļauj izstrādāt alternatīvos scenārijos, kad risks stājas spēkā [5] .

Projektā darba vienumu izpēti un darbietilpību novērtējumu veic vadošie sistēmu analītiķi, tādējādi precīzāk ļaujot noteikts nepieciešamās darbietilpības konkrētu darbu veikšanai. Šāds risku novērtēšanas modelis projektā ļauj efektīvāk novērtēt, kuru darbu vienumu izstrādi novirzīt citu kompāniju izstrādei un kuru izstrādi veikt pašiem. Kā jau projekta attīstībā tika identificēts, lielas problēmas radīja tieši prasību un projektējumu apraksti nacionālajā valodā, kas noveda pie papildus skaidrojošiem darbiem no sistēmu analītiķu puses. Paaugstinoties neparedzētām darbietilpībām projekta pirmajā fāzē, palielinājās projekta izmaksas un samazinājās atlikušais laiks citu darbu veikšanai līdz projekta piegādei klientam.

6.2.2 Uzdevumu piešķiršanas modelis, bāzēts uz priekšlikumiem

Balstoties uz projekta riskiem un to veicinošos faktoriem, modelis nosaka iespējamo risinājumu definēšanu uz priekšlikumu bāzes. Kā ieejas datus darba [5] autori iesaka izmantot iepriekšējā modelī noteiktos riskus un faktorus. Uzdevumu piešķiršanas modelis ir elastīgs un piemērojams katram modelim individuāli. Modelis nosaka ietekmes un projekta pamata vērtībām – kvalitāti, nepieciešamām izmaksām, laiku. Katrs darba vienums tiek vērtēts atsevišķi, un tiek meklēti iespējamie risinājumi, samazinot riskus, kas var ietekmēt minētās projekta pamata vērtības.

Projektā, dēļ nepareizu darbietilpību novērtēšanas, tika lemts par atsevišķu darbu izdali arējai kompānijai, tādējādi mēģinot atslogot vadošās kompānijas programmētājus no papildus darbiem. Darba paka saturēja daudzus darba vienumus ar atsevišķu darbietilpību novērtējumu. Bet projekta vadītājs diemžēl nenovērtēja riskus un faktorus, kas tos varēja veicināt, tādējādi paildzinot projekta nodošanu apmēram divas reizes, kā būtu to paveikusi projekta vadošās kompānijas darba grupa.

Šī modeļa ietvaros lietderīgi būtu organizēt projekta iekšējās sanāksmes vadošās kompānijas projekta vadītājam, sistēmu analītiķiem un vadošajiem programmētājiem. Tas nodrošinātu katras darba pakas un darba vienuma izvērtēšanu un risinājumu priekšlikumu veidošanu, pēc kuras varētu arī lemt par darba vienumu sadali citām kompānijām dalītā projekta ietvaros.

6.2.3 Nepieciešamo izmaksu modelis

Darba vienumu sadalē ir jāņem vērā arī nepieciešamās izmaksas un jāvērtē, kādas būs nepieciešamās izmaksas darba vienumu realizēšanai. Lai novērtētu darba vienumu izmaksas, var tikt izmantoti tādus izmaksu novērtēšanas modeļus kā COCOMO, bet dalīto projektu izstrādē ir

jāņem vērā arī potenciālie riski pārsniegt novērtētās izmaksas. Riskus un to ietekmējošos faktorus nosaka risku identifikācijas modelī.

Darba [5] autori iesaka salīdzināt novērtētās darbu izmaksas ar darba spēju izmaksām konkrētos dalītā projektu kompānijās, ņemot vērā izstrādātāju izmaksas, kompetenci veikt izstrādi darba vienumam. Tādējādi ir iespējams novērtēt, kuru darba paku vai darba vienumu izdalīt saistītai kompānijai dalītajā projektā un kuru nē. Kā arī, ja projektā ir iesaistītas vairākas kompānijas, var lemt par darbu paku sadali starp tām, izejot no saistītās kompānijas darbaspēku kompetences un nepieciešamajām izmaksām. Ir jāņem vērā arī kompānijas kompetence strādāt iteratīvi, izmantot spējās izstrādes metodes programmatūras realizācijā. Tāpat autori min projektu Francijā, kurā ar šādu pieeju tika noteikti līdz 80% savlaicīgi paredzētu risku. Šī projekta vadītāji šādu risku vērtēšanas pieeju ir novērtējuši kā ļoti efektīvu projekta pārvaldībā globāli dalītu projektu izstrādē.

6.3 Komandu veidošana

Ne mazāk svarīga ir pareiza komandu veidošana, kas veiks piešķirtos darbus. Balstoties uz *Agile* metodiku, pētījumu par komandu sadali ir veicis Džeisons un Šerija, tos aprakstot publikācijā [6] . Pētījumā tiek izmantoti principi no trīs dimensijām:

1) Spējās izstrādes vērtības, metodes un principi;
2) Komandas struktūra, kā ietvaros tika pētīti komandu atšķirīgie raksturlielumi;
3) Virtualitāte, kas dalīto projektu izstrādē apraksta komandu kā virtuālu, jo tā atrodas attālināti.

Autoru Džeisona un Šerijas kopdarba veiktajā pētījumā tiek izmantotas 3 ASV esošas kompānijas, kur pirmajā kompānijā ir 135 tūkstoši darbinieku, otrajā kompānija ir 40 tūkstoši darbinieku un trešajā kompānijā ir 9 tūkstoši darbinieku. Visas trīs kompānijas strādā ar dalītiem projektiem un darbinieki šo kompāniju veidotajos projektos atrodas daudzās citās valstīs ārpus ASV teritorijas. Pēc pētījuma veikšanas, autori izvirza sekojošus ieteikumus, kā uzlabot darbu ar komandām, kuru dalībnieki strādā dalīti vairākās valstīs pasaulē:

1) Precizēt uzdevumu aprakstus un veidot īsas iterācijas ar nelielām piegādēm – pētījuma autori uzskata, ka sadalot projekta darbus pietiekami mazās iterācijās, tas atvieglo uzdevumu veikšanu un vieglāk komandām ir sasniegt īstermiņa rezultātus, kas galu galā projektam palīdz ātrāk un stabilāk tikt pie gala rezultāta, saliekot projektu kopā no maziem nodevumiem, kā arī tas atvieglo projekta pārvaldības procesus;

2) Nodrošināt saistītājām kompānijām autonomu pārvaldi, lai saistītās kompānijas pašas savas komandas dalībniekiem dotu uzdevumus, ko risināt un tādējādi veidotos autonoma komanda. Savukārt saistītās kompānijas komanda uzdevumus saņem no vadošās kompānijas, kura nosaka to, kādi darba vienumi tiks uzticēti saistītās kompānijas komandai. Ja šī hierarhija tiek pārrauta, komandas dalībnieki vairs neuzticēties savas kompānijas komandas vadītājiem un gaidīs uzdevumus no vadošās kompānijas *Scrum master*. Tas savukārt var novest pie tā, ka vadošā kompānija nespēs masveidā pārvaldīt visus saistīto kompāniju dalībniekus un viņu veikumu iterācijās;
3) Nepieciešama nepārtraukta atgriezeniskā saite no saistīto kompāniju komandu darbiem, piemēram no saistīto kompāniju komandu *Scrum* sanāksmēm, autonomo iterāciju plānošanas un retrospekcijas sanāksmēm, piegāžu plānošanu;
4) Vadošās kompānijas projekta komandai ir jādefinē kopējie projekta nosacījumi, kurus ir jāievēro visām projekta saistīto kompāniju komandām, bet nedrīkst saistīto kompāniju komandām definēt nosacījumus, kā strādāt saistītās kompānijas iekšienē, lai panāktu vadošās komandas izvirzītos nosacījumus. Pētījuma autori iesaka uzticēties saistīto kompāniju komandām un ļaut viņiem sev zināmiem līdzekļiem sasniegt izvirzītos vadošās kompānijas mērķus;
5) Veidojot saistīto kompāniju komandas, ievērot mērenību – jo mazāka būs komanda, jo vieglāk būs to pārvaldīt. Ja kompānijas piedāvātā komanda ir liela, vienmēr pastāv iespēja tos pārdalīt un izmantot kompāniju tā, lai tā strādātu kā divas atsevišķas saistītās komandas ar saviem dalībniekiem;
6) Izvēlēties autonomām komandām saistītās kompānijas ar iespēju, lai mazāk atšķirtos laika zonas, lai pēc iespējas vairāk pārklātos darba stundas. Tas atvieglos spējās programmatūras izstrādes darbu, *Scrum* ikdienas sanāksmju organizēšanu. Ja projekta komandu dalībniekiem ir jārisina jautājumi, tos viņi varēs izdarīt tiešsaistē, negaidot nākamo dienu. E-pasts, protams, ir risinājums, bet uz to nāksies gaidīt, kamēr citas laika zonas projekta dalībnieks izlasīs un iedziļināsies. Visticamāk viņš to paveiks savā darba laikā nevis ārpus tās. Tādi saziņas līdzekļu kā tiešs zvans personai vai tiešsaites komunikācija, atvieglos problēmas izpēti un uzlabos gan izstrādājamās komponentes kvalitāti, gan arī nodrošinās piegādi paredzētajos iterācijas termiņos;
7) Izmantot pēc iespējas vairāk saziņā tādas saziņas tehnoloģijas kā darbvirsmas koplietošanu, darba grupas programmatūru (*MS Office, Google Documents,* u.tml.),

video konferences un telekonferences, kā arī tiešsaistes saziņu rīkus (*Skype, Instant messaging*). Šādu tehnoloģiju izmantošana stipri atvieglos darba ikdienu, uzlabos savstarpējo saziņu un atvieglos problēmu izpēti. Piemēram, iterācijas plānošanu var veikt attālināti, izmantojot video konferences iespēju. Telekonference var tikt izmantota retrospekcijas sanāksmē, kurā tiek analizēts paveiktais, iepriekš visiem sanāksmes dalībniekiem nosūtot retrospekcijas atskaiti.

Kā piemēru maģistra darba autors var minēt telekonferences zvanus Projektā, kā ietvaros pēc noteiktas komponentes ieviešanas produkcijā tiek atsevišķas pamanītās kļūdas pārrunātas ar kopējo pārvaldes organizācijai Briselē, jo projekta veidotā sistēma sadarbojas ar citu Eiropas valstu līdzvērtīgām sistēmām, nodrošinot datu apmaiņu starp tām. Tika veidota kopēja telekonference visām Eiropas Savienības dalībvalstīm, kurā katra valsts iezvanījās un informēja pārvaldes organizāciju par veiktajiem uzlabojumiem sistēmā un apsprieda problēmas, kā arī izrunāja nepieciešamos risinājumus. Ikdienā starp projekta dalībniekiem visbiežāk tiek izmantoti tādi saziņas līdzekļi kā e-pasts, tiešsaistes saziņa, zvani, kā arī informācijas apmaiņa ar pieteikumu uzskaites sistēmu, kas kalpo par pamatu darba uzdevumu pārvaldei;

8) Ir jābūt izpratnei, ka ģeogrāfiskai komandas atrašanās vieta nevar ietekmēt komandas darbu un veiktspēju. Pētījuma autori ierosina neņemt vērā visas „iedomātās" robežas starp komandām – ģeogrāfiskās robežas, kultūras robežas, laika robežas un uztvert komandu kā virtuālu.

Vēl interesanta piezīme ir autoram Hossainam [6] rakstā, kurā viņš uzsver 4. ieteikuma aprakstu, 4.secinājumā pasakot, ka katras saistītās kompānijas komanda var izmantot savu individuālu un pielāgotu *Scrum* programmatūras izstrādes modeli. Tas nozīmē, ka vadošai komandai nevajadzētu iejaukties citas kompānijas komandas darbā, bet ir jāseko līdz tās darbu izpildei un rezultātiem. Autors arī secina, kas komandas ir tiešām jāveido mazas, kā minēts 5. ieteikumā, un to, ka vadīt lielas komandas ir izaicinājums projekta pārvaldīšanai, jo ir jāspēj tās kontrolēt. Kā jau ieteikumos minēts, šādos gadījumos ieteicams lielas komandas dalīt mazākās, to vieglākai pārvaldīšanai. Šādu pieeju, apraksta autors publikācijā, iesakot, ka katrai kompānijai ir jāveido sava atsevišķa *Scrum* komanda un jāstrādā autonomi ar piešķirtajiem darba vienumiem, savukārt vadošai kompānija jānodrošina *Scrum* vadošā komanda no lokālajām *Scrum* komandām saistītajās kompānijās, reizi dienā apmainoties ar informāciju, lai iegūtu informāciju par projekta

gaitu un statusu. Autors publikācijā uzsver 3 veidu *Scrum* komandu sadalīšanas modeļus globāli dalīto projektu izstrādē:

1) Katra *Scrum* komanda strādā autonomi ar savu atsevišķu komandas *Scrum master*, kurš neatskaitās katru dienu vadošai kompānijai. Komandas dalībnieki strādā vienuviet – atbildība par darbu izpildi tiek noteikta katrai komandai iekšēji un iterāciju piegādes tiek formētas neatkarīgi;
2) Katrai *Scrum* komandai ir savs *Scrum master*, kur ir vadošās kompānijas *Scrum* dalībnieks. *Scrum master* komandai katru dienu atskaitās vadošai kompānijai par iekšējo *Scrum* sanāksmes norisi un iterācijas attīstības gaidu, kā arī darbu gatavību iterācijas piegādes formēšanai. Komandas dalībnieki arī šai gadījumā atrodas vienuviet;
3) Katrai *Scrum* komandai ir savs *Scrum master*, kas atskaitās vadošās kompānijas *Scrum* komandai, bet nav svarīgi *Scrum* komandas dalībnieku atrašanās vienuviet – komandas daļinieki var atrasties dažādās kompānijās un pat dažādās valstīs. Šādu *Scrum* komandu var uzskatīt kā virtuālu, jo nav svarīga dalībnieku atrašanās vieta, bet gan komandu organizatoriskā struktūra.

Savukārt, autori Ošri, Korlarski un Villkos savā darbā [25] apraksta 4 iespējamos sadales modeļus:

1) Produkta modelis, kurā darbi tiek sadalīti pa komandām pēc produkta struktūras, nosakot to, ka, izejot no sistēmas arhitektūras, noteiktas komponentes izstrādā atsevišķā saistītās kompānijas komanda;
2) Projekta fāzes/soļa modelis, kurā darbi starp saistīto kompāniju komandām tiek dalīti pa projekta attīstības soļiem. Piemēram, viena kompānija apraksta specifikāciju un veic projektēšanas darbus, otra veic izstrādi un trešā veic testēšanu.;
3) Labi aprakstītu darba vienumu sadales modelis, kā ietvaros darba vienumi, kuriem ir pilnvērtīgi un skaidri projektējuma apraksti, tiek nodoti izstrādei citai kompānijai;
4) uz produkta konfigurāciju orientēts modelis, kā ietvaros viena kompānija var veikt izstrādi, savukārt citai tiek nodoti konfigurācijas darbi.

Projektā daļēji tiek izmantoti pirmie un otrie autoru minētie modeļi, jo projekta izstrādē testēšanu veic kompānija K_t, savukārt otrajā fāzē noteiktas sistēmas komponentes analīzi veica kompānija K_i. Savukārt atsevišķas komponentes tiek atdotas izstrādei saistītajām kompānijām.

6.4 Iterāciju ieviešana

Nākamā svarīgā sadaļa pēc darba vienumu plānošanas procesiem ir darba vienumu sadale pa iterācijām un nodošana izstrādei. Kā arī ikdienas *Scrum* sanāksmju organizēšana, kas ļauj efektīvi pārvaldīt plānoto darbu virzību iterācijā un savlaicīgi konstatēt izpildes riskus un pieņemt svarīgus lēmumus darba vienuma izstrādē.

6.4.1 Iterāciju plānošana

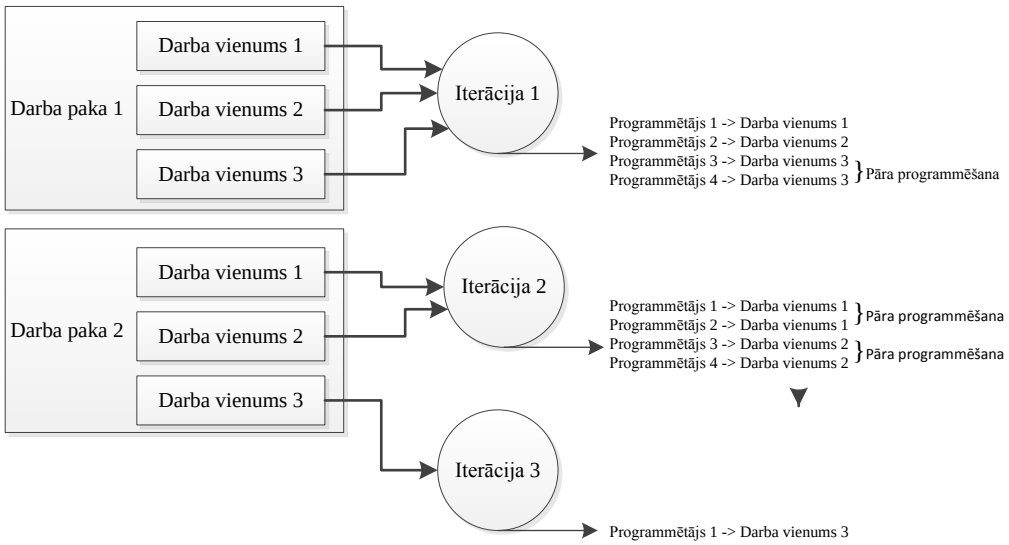
Iteratīvajā darbu plānošanā svarīgi ir zināt, kā sistēmas izstrādājamās komponentes integrējas sistēmā un ir jāsaprot, kuras komponentes ir nepieciešamas izstrādāt vispirms, un kuras var atlikt vēlākai izstrādei. Tādēļ projektā liela lomu ir iterāciju plānošanai. Iterācijas iesaka dalīt no nedēļas līdz mēnesim, atkarībā no veicamo darbu apjoma, sistēmas lieluma un piegādājamo komponenšu daudzuma. Maģistra darbā ietvertā projektā ir atrasts ideāls līdzsvars iteratīvai darba veikšanai – 3 nedēļas. 3 nedēļu ietvaros tiek veikta plānoto darbu izstrāde, prasību precizēšana un paša izstrādājamā vienuma pilnveidošana atbilstoši prasību izmaiņām. Mazākiem projektiem, protams, var iterācijas lielumu samazināt līdz nedēļai, ja izstrādājamie darba vienumi ir dalīti pietiekami mazos vienumos, kas paveicami ātrāk ar mazāku darba ieguldījumu.

Pasivara savā pētījumā [7] apraksta interesantu modeli, kad dalītajā projektā iterācijas var tik dalītas mazākās daļās – mazākās iterācijās. Piemēram, viņas aprakstītajā projektā, tiek izdalītas lielās iterācijas un mazās. Lielās iterācija nosaka 4 nedēļu ilgstošu izstrādi visām projekta komandām, kas izstrādā noteiktu vienotu komponenti vai sistēmas apgabalu. Savukārt mazā iterācija nosaka 2 nedēļu ilgstošu izstrādi atsevišķām dalītā projekta komandām ar izvirzītiem individuāliem mērķiem, lai iteratīvi abās 2 nedēļu iterācijās sasniegtu 4 nedēļu lielās iterācijas izvirzītos mērķus. Visas iterācijas, gan lielās, gan mazās tika uzsāktas un pabeigtas visām projektā saistītajām komandām vienlaicīgi. Projektā 4 nedēļu iterāciju pārvaldīja vadošās kompānijas komanda, kas arī izvirzīja mērķos plānots darbus. Tālāk darbi tika dalīti par saistīto kompāniju komandām un tās savukārt ievēroja 2 nedēļu mazās iterācijas izpildes plānus. Interesanti, ka autores aprakstītajā projektā tiek uzsvērts jēdziens *design-sprint,* kas nosaka iterāciju sadali ne tikai izstrādes darbiem, bet arī prasību analīzes un projektēšanas darbiem. Tādējādi *Scrum* metodes tiek pielietotas arī analītiķu veicamajiem darbiem.

Spējās programmatūras izstrādē iterāciju plānošanai tiek izmantota *Pokera* spēle, kas dalītajos projektos ir ļoti grūti veikt tādā formā, kā to nosaka *Agile* [22] . Lai to nodrošinātu, var izmantot video konferences vai speciāli izstrādātas aplikācijas šim nolūkam. Vairāki mobilo

lietotņu programmatūras izstrādātāji jau šobrīd piedāvā tirgū efektīvus rīkus, kas ļauj īstās kārtis aizvietot ar virtuālajām un veikt individuālu vērtējumu attālināti. Priekš *Android* un *iOS* platformām ir pieejamas tādas programmas kā *Planning Poker*, ko izstrādājusi kompānija *Unboxed Consulted* vai *Centare Planning Poker*, ko izstrādājusi kompānija *Centare Group*. Šādas aplikācijas lieliski aizvieto īstās kārtis un pat piedāvā kāršu kavu izvēli atbilstoši novērtējamajam darba lielumam (cilvēkdienu skaitam). Iteratīvā plānošana noteikt reizi iterācijā, piemēram, reizi 3 nedēļās un parasti ilgst ne ilgāk par 4 stundām. Maģistra darba autors ir novērojis, ka plānošanas spēli labāk ir organizēt rīta stundās, kas sakrīt ar Lefingvela pirmās vadlīnijas aprakstu [8] , jo darbinieki ir atpūtušies un plānošanas procesu neietekmē tādi blakus faktori, kā ikdienas darbu risināšana un nogurums. Ja šādas sanāksmes rīko pēcpusdienās vai vakara stundās, plānošana nenotiek tik pārdomāti kā rīta stundās, tādējādi palielinot risku un nepareizu darbietilpību novērtēšanu iterācijā.

Iteratīvajā darba plānošanā svarīga ir darbu paku un darbu vienumu efektīva sadale. Piemēram, ja darba vienums ir pārāk liels iterācijai, ir jāmeklē iespējas izstrādi veikt dalīti – iterācijā pie darba vienuma strādā vairāki programmētāji (var arī no dažādām kompānijām), sadalot komponentei nepieciešamos darbus atsevišķi un realizācijas aprakstā definējot kopīgās lietas. Attēlā 6 ir attēlots piemērs, kā var veikt darba vienumu sadali iterācijās noteiktiem programmētājiem.



Attēls 6. Piemērs darba vienumu dalīšanai iterācijās

Kā redzams attēlā, darba vienums var tiks piešķirts arī vairākiem programmētājiem, to sadalot 2 mazākos darbos. Problēmas var rasties, ja abi programmētāji nav vienas kompānijas

darbinieki, bet gan strādā divās atsevišķās kompānijās. Tas rada papildus riskus un nododamā darba vienuma kvalitāti un veicot pārstrādi, palielina darbietilpības un izmaksas projektam kopumā. Lai nodrošinātu vienādu izstrādājamās komponentes izstrādes pieeju dalīto projektu izstrādē var palīdzēt sekojoši risinājumi:

1) projekta programmēšanas standarta apraksts, kas nosaka ne tikai stilu kā jāraksta programmatūras kods, bet arī arhitektūras pamatelementus – bibliotēkas, ER modeļa kompozīciju, projekta programmēšanas labo stilu. Tas samazina izstrādāto komponenšu savietojamības risku starp kompānijām, veicot piegādi un darba vienuma integrāciju kopējā sistēmā. Projektā priekš nākamajām projekta fāzēm tiek gatavots dokuments, kas apraksta projekta izstrādes standartus, jo visai bieži ir novērotas nepilnības integrējot izstrādātus vienumus vienā komponentē;
2) darba vienuma dalītais risinājuma apraksts, kurā sistēmu analītiķis ir noteicis programmētājam, kas ir jārealizē un aprakstījis kopējo funkcionalitāti. Sistēmu analītiķis var darba vienumu izdalīt 2 mazākos darba uzdevumos katram programmētājam atsevišķi. Šādā veidā izdalot darbus, ir viegli tos piešķirt saistītai kompānijai, gan apvienojot gan pa vienai vienībai. Ja tomēr realizācija pārklājas, tas tiek pierakstīts pie veicamā darba vai ar programmētājiem tiek komunicēts atsevišķi;
3) izmantot *pair programming* priekšrocības, kas nosaka abu programmētāju viena otra koda caurskatīšanu (*Code review*), uzlabojot programmatūras koda kvalitāti un identificējot iespējamās problēmas jau izstrādes procesā Pāra programmēšanas efektivitāti iteratīvajā izstrādē apraksta arī Holmstroma un Fitzgeralds publikācijā [16] .

Iterācijas plānošanas rezultātā tiek sagatavots iterācijas darbu saraksts (*iteration backlog*) ar uzdevumiem, kas ir plānoti paveikt šai iterācijas posmā. Iterācijas darbu saraksts ir katras iterācijas pamata elements, un katrs uzdevums šai sarakstā iterācijas dzīves ciklā iziet cauri sekojošiem posmiem [8] :

1) atbildības nodošana, kā ietvaros komandas vadošās kompānijas *Scrum master* izdala darbus savas kompānijas programmētājiem un citu saistīto kompāniju *Scrum master,* kuri savukārt tālāk nodod darbus savas kompānijas programmētājam, saskaņojot to ar pašu programmētāju atbilstoši programmētāja kompetences līmenim;
2) izstrāde – programmētājs izstrādā uzdevuma realizēšanas plānu, kas pie reizes ir arī vienībtesta plāns un veic izstrādes darbus;

3) darba vienuma piegāde – programmētājs, izpilda vienībtestēšanu atbilstoši iepriekš sastādītajam plānam un veic realizētā vienuma saglabāšanu dalītā projekta kopējā repozitorijā, integrējot realizēto vienumu kopējai sistēmas komponentei;
4) pabeigtības deklarēšana – signāls testētājiem par izstrādātā lietojumu stāsta gatavību, uz kā pamata var sākt veikt funkcionālos testus Ja funkcionālais tests tiek akceptēts, lietojumu stāstam iterāciju darbu sarakstā tiek nomainīts statuss, ka vienums ir sagatavotai piegādei klienta akcepttesta videi.

Pasivaras darba [7] ietvaros pētītajā projektā apraksta, ka iterāciju plānošana ir tikusi organizēta 3 līmeņos – dalīto komandu sanāksmes, lokālā sanāksme Norvēģijas kompānijas komandai un lokālā komandas Malaizijas komandai. Dalīto projektu sanāksmju organizēšanai tika izvēlēts noteikts laiks, kad Norvēģijas un Malaizijas kompānijas komandas vienlaicīgi darba laikā varētu organizēt telekonferences sanāksmi. Telekonferences sanāksmes tika organizētas ar Microsoft izstrādātās aplikācijas *NetMeeting* palīdzību, neizmantojot Web kameras. Šajās sanāksmēs primāri tika izrunāti projekta organizatoriskie jautājumi, par *Scrum* darba organizāciju, iterāciju plānošanu, tika izrunātas pamanītās nepilnības un domāti risinājumi darba stila uzlabošanai komandās, par pamatu ņemot spējās programmatūras izstrādē noteiktos pamatprincipus. Tālāk izstrādājamā produkta klienta pārstāvis, izmantojot iterāciju *backlog*, iet cauri visiem plānotajiem darba vienumiem un uzdot saistīto kompāniju komandām nepieciešamos jautājumus. Svarīgi, lai pirms sanāksmes organizēšanas, klienta pārstāvis jau būtu noteicis prioritātes plānotajiem darbiem nākamai iterācijai un noteicis nepieciešamās darbietilpības.

Tā kā Norvēģijā un Malaizijā laika zonas ir atšķirīgas, tad Pasivaras aprakstītajā [7] pētījumā iterāciju plānošanas sanāksmes tika organizēta rīta pusē pēc Norvēģijas laika un Malaizijā vakarpusē, tieši pirms darba laika beigām. Tā kā Norvēģijā atrodas vadošā kompānija, darba vienumus definē un kopēju plānu sastāda šī kompānija. Nākamajā rītā, ja nepieciešams tiek vēl precizētas iterācijas detaļas un nosūtīts plāns saistītai kompānijai.

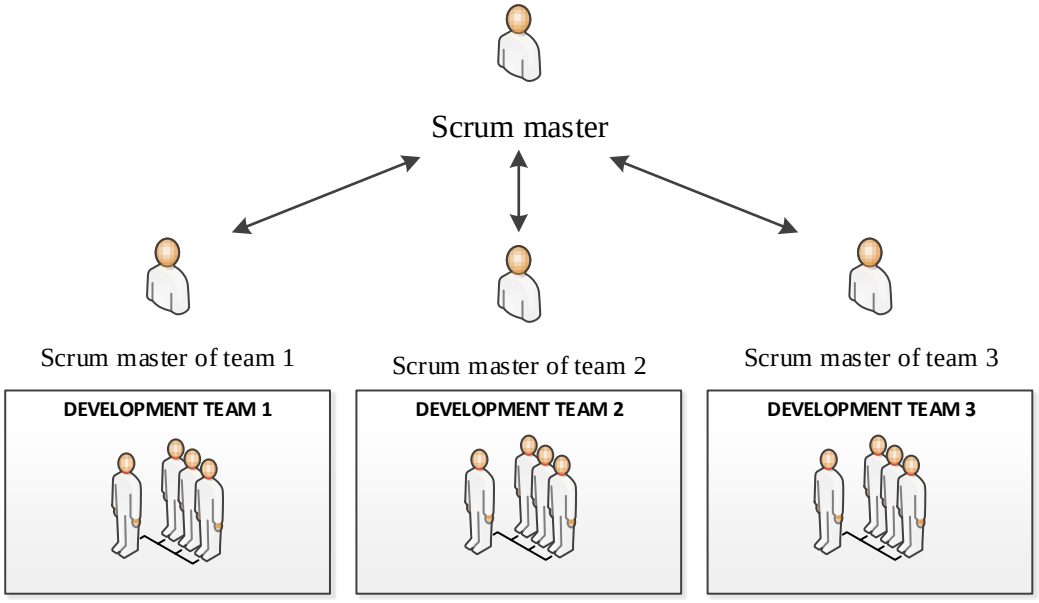
6.4.2 *Scrum* ikdienas sanāksmes

Svarīga ir katra iterācijas dalībnieka atskaites sniegšana par paveikto un nākamo darbu plānošana dienas ietvaros. Tādēļ katru dienu (ne mazāk) ir jārīko *Scrum stand-up* sanāksmes, kā ietvaros 15 minūšu laikā (*vēlams*) katrs iterācijas dalībnieks īsi pastāsta par iepriekšējā dienā paveikto un plānotiem darbiem pašreizējā dienā. *Scrum stand-up* sanāksmes vada *Scrum master*,

kas piefiksē katra dalībnieka teikto, lai pēc tam varētu veikt darba izpildes novērtējumu un iterācijas plāna kopējo virzību, kā arī iterācijā piegādājamo darba vienumu kvalitāti.

Dalīto projektu izstrādē *Scrum* sanāksmes ir grūti iedomājamas tādā formā, kā to nosaka Agile metodika. Pēc Agile metodikas, visiem dalībniekiem ir jābūt klātienē un, kājās stāvot, 15 minūšu laikā jāizrunā izdarītie un plānotie darbi. Tātad, ja iterāciju gatavo vairākas kompānijas ar saviem darbiniekiem, *Scrum* pēc Agile metodikas aprakstītajā formā vairs nav iespējama.

Risinājums ir veikt *Scrum* komandas sadalīšanu mazākās komandās [6] . Dalīto projektu izstrādē katra no kompānijām var veidot savu autonomu *Scrum* komandu un rīkot ikdienas sanāksmes. Tā kā katru *Scrum* komandu vada *Scrum master*, tad pēc šīm sanāksmēm ir vieglāk organizēt papildus *Scrum* sanāksmi, kurā piedalās tikai *Scrum master* no saistīto kompāniju projektiem. Tas nodrošina efektīvu veidu kā apkopot projekta izpildes gaitu un uzzināt statusu. Svarīgi ir kopēju *Scrum* protokolu izsūtīt visiem projekta dalībniekiem.



Attēls 7. *Scrum* komandas dalītajā projektā [6]

Šādam sadalījumam, kas attēlots attēlā 7, ir svarīgi ņemt vērā laiku, kad notiek informācijas apkopošana no saistītajiem projektiem. Tā kā dalītajos projektos bieži vien ir vērojamas tādas situācijas, kad saistīto projektu kompānijas atrodas citās valstīs, tad ļoti liela iespēja ir, ka tām var būt arī cita laika josla. Sadalot *Scrum* komandas, var panākt katrā komandā *Scrum* sanāksmes organizēšanu sev vēlamā laikā, bet *Scrum master* no katra saistītā projekta komandas dienas ietvaros nosūta vadošās kompānijas *Scrum master* apkopojumu par padarīto un plānotiem darāmiem darbiem dienas laikā. Līdz ar to vadošās kompānijas *Scrum master* var dot

ierosinājumus, kā arī sazināties ar noteiktiem saistīto projektu dalībniekiem, lai precizētu darāmos darbus vai dotu padomus, kā tos labāk paveikt.

Plānotajā iterācijā katra darba izpildei ir jāseko līdz, katru dienu atjaunojot plānošanas sanāksmē noteikto kopējo stundu (cilvēkdienu) skaita patēriņu. Šādu grafiku *Agile* metodikā sauc par *Burn-Down Charts*, kā ietvaros grafiski pa dienām var redzēt aktuālo palikušo neiztērēto stundu (dienu) apjumu, novērtējot to, cik vēl atlicis laika izstrādes darbiem. Dalīto projektu gadījumā svarīgi ir apkopot katra iterācijas darba vienuma patērēto stundu (dienu) skaitu. Šī informācija ir jāapkopo katras autonomās komandas *Scrum master* un jānosūta kopā ar atskaiti par darbu izpildi vadošās kompānijas *Scrum master*.

Katram darba vienumam laika patēriņš var tikt uzturēts kopējā pieteikumu sistēmā, kurā grafiski var redzēt kopējo paredzēto izpildes laiku un faktiski patērēto. Šāda pieeja garantē to, ka jebkurš projekta dalībnieks var apskatīt kopējo iterācijas darbu izpildes statusu un attiecīgi plānot darbus atlikušajā vēl neizmantotajā iterācijas laikā.

Pasivara savā pētījumā [7] apraksta *Scrum* sanāksmju procesu dalītajam projektam Somijā un Malaizijā, kur katrā no abām kompānijām projektā ir iesaistīti 20 cilvēki. Šajā projektā *Scrum* sanāksmes tiek organizētas katru dienu to divu stundu laikā, kad abām kompānijām pārklājas aktīvais dienas darba laiks. *Scrum* sanāksmēs piedalās izstrādātāji, sistēmu analītiķi un *Scrum master*. Papildus iezīme ir tāda, ka katrā kompānijā papildus vēl piedalās vēl izstrādājamās sistēmas klienta pārstāvis. Abas komandas uz 15 minūtēm rezervē telekonferences telpu ar interneta pieslēgumu un *Web* kamerām. Arī šai projektā tika ievērota tāda *Agile* metodikas iezīme, ka diskusijas netiek izvērstas *Scrum* sanāksmē, bet tiek organizētas atsevišķas sanāksmes problēmu izrunāšanai un risinājuma rašanai. Kā autore pētījumā [7] apraksta, *Scrum* sanāksmes šai dalītajā projektā ļāva labāk saliedēt abas komandas, gūt kopskatu par projekta kopējo gaitu un palīdzēja savlaicīgi identificēt projekta problēmas, jo katru dienu pārskatot paveiktos darbus un plānojot darbus nākamai dienai, ir grūti ilgstoši noslēpt problēmas. Viens no projekta dalībniekiem pat esot pateicis: „Ka ikdienas *Scrum* sanāksmes ir labākais, kas noticis šai dalītā projekta komandām."

Pasivara arī apraksta, kā šai projektā tie organizēts *Scrum of Scrums* sanāksmju ideja. No katras komandas tiek izvirzīts viens pārstāvis, kas piedalīsies šais sanāksmēs. *Scrum of Scrums* sanāksmes parasti tiek organizētas vienreiz nedēļā, kurā katrs pārstāvis pastāsta, kas ir izdarīts nedēļas laikā un kādi mērķi tiek izvirzīti nākamai nedēļai, līdz nākamai sanāksmei. Pārstāvis ne vienmēr ir viena un tā pati persona, bet var mainīties, atkarībā no izstrādātā vienuma veida.

Visbiežāk tas var būt vai nu vecākais sistēmu analītiķis, vai sistēmu analītiķis, kurš ir veicis projektēšanas darbus izstrādājamajam darba vienumam. Lai uzlabotu projekta nodevumu integritāti, projekta pārstāvis arī iepazīstina izstrādātā un izstrādājamās komponentes ietekmi uz pārējās sistēmas daļām, ko izstrādā citas komandas un kā tiks organizēta nākamās iterācijas piegāde. Autore uzsver, ka šādas sanāksmes uzlabo dalīto komandu komunikāciju un palīdz komandām saprast, ko pārējās komandas dara projektā un kā citu komandu paveiktais darbs ietekmēs viņu veicamos darbus. Ar šādu pieeju tiek mazināts integritātes risks dalītajos projektos.

6.4.3 Retrospekciju sanāksmes

Pēc iepriekš aprakstītajiem soļiem tiek veikta iterācijas darbu saraksta pārskatīšana retrospekcijas sanāksmē, kurā tiek analizēta katra darba vienuma kvalitāte un pabeigtība. Ja darba vienums nav pilnībā pabeigts, var tikt lemts par tās piegādes atlikšanu uz nākamo iterācijas posmu. Svarīgi ir, lai katrai komandai, kas piedalās projektā ir pieeja pieteikumu sistēmai. Papildus tam, vadošās kompānijas *Scrum master* var mainīt atbildību uz kādu no saistītajām kompānijām, tādējādi nododot uzdevumu izpildei komandai, kas strādā citā kompānijā.

Pasivara savā darbā [7] apraksta kā tika organizētas retrospekcijas sanāksmes dalītajā projektā Norvēģijā un Malaizijā. Retrospekcijas sanāksmes šai projektā tika organizētas pēc iterāciju demonstrācijas sanāksmes un tajā piedalījās visi dalīto projektu komandu dalībnieki, *Scrum master*, kā arī projekta vadītājs. Sanāksmes projektā notika ne ilgāk par stundu, kā ietvaros katra no komandām atbildēja uz sekojošiem jautājumiem:

1) Kādas bija iterācijas pozitīvās lietas?
2) Kādas iterācijā bija problēmas un kas neveicās tik labi?
3) Kādus uzlabojumus varētu ieviest, lai uzlabotu izstrādes darba procesus?

Atbildot uz šiem jautājumiem un meklējot vēlamos uzlabojumus, komandas, kuras izstrādā komponentes šim projektam, šādu darba metodi ir novērtējušas kā efektīvu veidu, kā uzlabot darba procesu. Šādā veidā izskatot iterācijā veiktos darbus, dalītā projekta komandas tika vairāk saliedētas un atrisinātas ikdienas problēmas, kas rodas, ja vairākas komandas strādā attālināti viena no otras, bet tomēr vienoti veido vienu sistēmu.

6.4.4 Darbu saraksts (*backlog*)

Maģistra darba autora Projektā darba sarakstu veidošanai un uzturēšanai tiek izmantots rīks Jira, kurā uz noteiktu 3 nedēļu iterāciju tiek noteikts ar klientu saskaņots darba vienumu saraksts un definēti iterācijas nodevumu termiņi, darbietilpības. Darbus no darba vienumu saraksta var

izstrādāt pati vadošās kompānijas komanda vai nodot to ārējiem izstrādātājiem, kuri arī izmanto šo pašu rīku. Tālāk zem katra no darba vienumiem ir iespējams reģistrēt papildus darbus, kas precizē realizācijas gaitu. Nepieciešamības gadījumos šos papildus darbus arī ir iespējams dalīt pa saistītajām kompānijām.

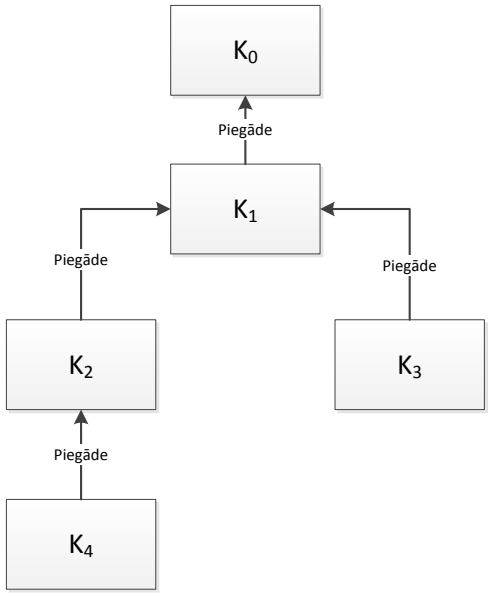
Tā kā projektā ne tikai izstrādes darbi tiek veikti dalīti, bet arī cita kompānijas veic testēšanu, tas ir svarīgi, lai pēc darba vienumu izstrādes ārējā kompānija varētu pieslēgties darba vienumu saraksta rīkam un reģistrēt pamanītās kļūdas aplikācijā. Tiklīdz testēšanas procesā ir konstatēta kļūda, testēšanas kompānijas komanda, informē par to analītiķi, kurš ir veicis atbilstošā darba vienuma projektēšanu un realizācijas apraksta veidošanu. Ja arī analītiķis konstatē kļūdu, darba vienums tiek nomainīts atpakaļ uz izstrādi. Ja tomēr kļūda neeksistē un testētājs ir kļūdījies, analītiķis atgriež darba vienumu testēšanai. Protams, ir jāņem vērā viena nianse, lai atvieglotu darba procesu analītiķim. Testētājam ir jābūt pārliecībai, ka testēšanā nodotais darba vienums tiešām nestrādā atbilstoši plānotajam un aprakstītajam realizācijas aprakstā vai projektējumā. Ja testētājs nav pārliecināts, viņam šai projektā ir jāsazinās vispirms ar analītiķi un jānoskaidro situācija.

Pasivaras pētītajā projektā [7] arī tiek izmantota Jira aplikācija darbu vienumu uzglabāšanai, plānošanai par iterācijām. Darbu sarakstus uz katru iterāciju saplāno projekta vadītāji. Projekta vadošai komandai ir tiesības definēt iterācijā noteiktos primāros darba uzdevumus, bet dalītā projekta saistītās kompānijas zem tiem var veidot papildus uzdevumus, tādējādi precizējot veicamo darba sarakstu uz noteikto iterāciju, kā arī uzlabojot darba izpildes caurskati un plānošanas procesus dienas ietvaros.

6.5 Programmatūras izstrāde, testēšana un piegāde

Jebkura projekta pamats ir programmatūras vienumu izstrāde un izstrādātu darba vienumu piegāde. Dalītajā programmatūras izstrādē problēmas var radīt fakts, ka kopējais programmatūras nodevums tiek formēts no vairākiem mazākiem nodevumiem, kuras ir piegādājušas saistīto kompāniju komandas. Tādēļ jau pie darbu sadales ir svarīgi pārdomāt vai konkrētais darba vienums, kas tiks atdots izstrādei saistītai kompānijai neradīs integrācijas problēmas, mēģinot tos iestrādāt kopējā komponentē, kura savukārt tiks piegādāta klienta testa videi.

Dalīto projektu izstrādē, kā jau iepriekš minēts, vēlams iterācijas ilgumu noteikt 2 vai 3 nedēļas, ņemot vērā, ka piegādes formēšanas brīdī nāksies apkopot realizētos vienumus no saistīto komandu piegādēm.



Attēls 8. Piegāžu shēmas piemērs iterācijai

Attēlā 8 ilustratīvi ir atainota situācija, kurā uz iterāciju ir jāveic piegāde klienta akcepttestēšanas videi. Attēlā ir noteiktas sekojošas kompānijas:

1) K_0 – Klienta kompānija, kurā atrodas akcepttestēšanas vide;
2) K_1 – Vadošā kompānija, kas formē piegādes priekš klienta akcepttesta vides, apkopojot saistīto kompāniju komandu piegādes;
3) K_2 un K_3 – kompānijai K_1 pakārtotās kompānijas;
4) K_4 – kompānijai K_2 pakārtotā kompānija.

Kā redzams attēlā, noslēdzot iterāciju kompānijai K_4 ir jānoformē piegāde kompānijai K_2, savukārt kompānijām K_2 un K_3 ir jānoformē piegādei K_1, kuras savukārt sagatavo kopējo piegādi ar realizētiem datu vienumiem klienta kompānijai K_0. Visas saistītās kompānijas izmanto vienu pieteikumu reģistrēšanas sistēmu, kas nodrošina pieteikumu statusu uzglabāšanu. Papildus problēmas var rasties, ka klientam ir jāpiegādā vairākas versijas sistēmai, piemēram, ir atsevišķa testa videi produkcijas esošām komponentēm un atsevišķa testa vide jauniem realizējamiem vienumiem. Rodas jautājums, kā labāk organizēt šādas piegādes starp saistītajām kompānijām, lai neciestu programmatūras kvalitāte un visi iterācijā realizētie datu vienumi tiktu piegādāti savlaicīgi, neradot integritātes un stabilitātes problēmas visai sistēmai kopumā.

Risinājums ir jāmeklē piegāžu darbu formēšanas organizēšanā. Maģistra darba ietvertajā projektā ir novērots, ka piegādes formēšanai, apkopojot visu saistīto kompāniju realizētos darba vienumus, ir nepieciešama vismaz viena diena. Svarīgi ir, lai kopējais projekta programmatūras

koda repozitorijs šīs dienas sākumā tiktu „iesaldēts" un lai tajā netiktu veiktas papildu izmaiņas. To apstiprina arī autors Lefingvela darbā [8] . Tas nodrošinās piegādes formēšanas stabilitāti un samazinās integritātes riskus, veicot regresa testus pirms piegādes klienta testa videi. Autors arī iesaka vispirms uzrakstīt testa scenāriju un tikai tad veikt izstrādi [8] , līdz ar to maģistra darba autoram radās ideja projektā ieviest uz testēšanu balstītās izstrādes metodi.

Lefingvels arī darbā [8] uzsver faktu, ka, lai saformētu piegādi, tikpat liela nozīme ir pieteikuma statusam pieteikumu reģistrēšanas sistēmā. Tam ir jābūt statusā, lai piegādes veidotājs saprastu, ka darba vienums ir nodots piegādes formēšanas procesam. Ja statuss nav atbilstošs, piegādes formētājs šo darba vienumu var neiekļaut piegādē, par ko arī tiek piefiksēts darbu atskaitē (*iteration backlog*) [8]. Tātad jebkura no saistītajām kompānijām pēc darbu realizēšanas garantē ar šo statusu piegādes gatavību, dodot signālu vadošai kompānijai par darbu vienumu pabeigtību un gatavību piegādei.

Vadošā kompānija atjauno komponenti ar realizētiem un piegādātiem darba vienumiem un uzstāda regresa testa vidē, lai veiktu regresa testus. Regresa testi ir automatizēti un pārklāj jau iepriekš realizēto sistēmas funkcionalitāti. Ja kāds no testiem nav sekmīgi izgājis, par to tiek informēts piegādes formētājs, kurš savukārt veic attiecīgu izpēti problēmai un nepieciešamības gadījumā sazinās ar programmētāju vai atbildīgo analītiķi, tādējādi veicot izmaiņas un novēršot atklāto problēmu. Kad piegādes formēšana ir pabeigta, tā tiek nosūtīta klientam uzstādīšanai testa vidē, kur var noritēt akcepttestēšana no klienta puses.

Tikai pēc piegādes nosūtīšanas klienta testa videi, tiek dota zaļā gaisma iterācijas pārskata sanāksmei un nākamās iterācijas plānošanas darbiem. Programmatūras koda repozitorijs tiek atslēgts nākamajām koda izmaiņām tikai pēc plānošanas procesa pabeigšanas un jaunu darba vienumu piešķiršanas atbildīgajiem projekta dalībniekiem.

6.6 Literatūras apskata izdarītie secinājumi

Literatūras apskats par spējās izstrādes izmantošanu dalītajos projektos tika veikts ar domu, uzsākot trešo projekta fāzi ieviest vēl uzlabojumus projekta procesos, lai projekts izmantotu spējās izstrādes noteiktos pamatprincipus pilnvērtīgi. Literatūras apskats labi parādā, kādā veidā plānot iterācijas dalītajos projektos, kā organizēt ikdienas *Scrum* sanāksmes, kā nodrošināt spējās izstrādes ikdienas projekta statusa noskaidrošanu starp vairākām kompānijām, kā pareizi organizēt darba vienumu sadali starp kompānijām, rīkot plānošanas un retrospekcijas sanāksmes, un kā pareizi nodrošināt piegāžu formēšanu.

Pēc veiktā literatūras apskata, darba autors saskatīja, ka projekta trešajā fāzē ir iespējams veikt vairākus uzlabojumus gan plānošanas procesā, kad organizējot darba vienumu dalīšanu starp projekta dalībniekiem un citas kompānijas izstrādātājiem, kā pareizi dalīt atbildību, organizējot *Scrum* sanāksmes un noskaidrojot projekta statusu. Radās ideja uzlabot arī testēšanas procesus, nodrošinot uz testēšanu balstīto izstrādes ieviešanu projektā.

Nākamā darba nodaļa ir veltītai projekta trešajai fāzei, kā ietvaros tiks izskaidroti ieplānotie projekta uzlabojumu šajā fāzē.

7 Plānotie trešās fāzes uzlabojumi projektā

Projektā vairāk nekā 3 gadu laikā tika atklātas vairākas problēmas, kuras tika analizētas, ieviesti risinājumi un pieņemti lēmumi izmēģināt tos projektā. Līdzīgi kā pirmajā un otrajā fāzē, arī trešajā fāzē ir plānots ieviest vairākus uzlabojumus, izmantojot rīcību izpētes (*Action Research)* [11] metodi, kas sastāv no sekojošiem soļiem potenciālā risinājuma ieviešanai projektā:

1) Problēmas identificēšana projektā – projektā tika identificēta kāda problēma vai problēmu virkne projekta procesā vai procesos;
2) Nepieciešamo izmaiņu analīze:
    a. Problēmas izpēte un analīze - uz projekta darba grupā veiktas diskusiju bāzes, tiek izpētīti iemesli un sagatavoti potenciālie risinājumi – nepieciešamo aktivitāšu soļi, lai panāktu situācijas uzlabošanos projektā
    b. Izmaiņu plānošana – pēc problēmu apzināšanas, tika apzinātas nepieciešamās izmaiņas projektā;
3) Izmaiņu ieviešana projektā:
    a. Aktivitāšu soļu ieviešana projektā, kā ietvaros tiek mainīti projekta procesi;
    b. Izmainīto procesu uzraudzīšana, lai noteiktu, vai ieviestie aktivitātes soļi kā risinājums ir palīdzējis novērst sākotnēji iniciēto problēmu.

Rīcību izpētes metodes mērķis ir saskatīt problēmas projektā un rast idejas to novēršanai. Tā kā nepieciešamo izmaiņu ieviešanai piedalījās visa projekta komanda, tad rīcību izpētes procesā tika iesaistīti visi projekta dalībnieki. Metode sevī ietver vairākas iterācijas, kurās tika mainīti darba procesu organizācija projektā, ar mērķi saprast vai plānotā ideja veikt izmaiņas projektā nes ieguvumu vai gluži pretēji ir neefektīva un ir jāatgriežas pie iepriekšējiem izstrādes procesiem. Zemāk aprakstītajā tabulā 4 ir aprakstītas *Agile* metodes, kuras ir ieviestas projektā

pirmajā un otrajā fāzē, un tās metodes, kuras ir paredzēts ieviest projekta trešajā fāzē, balstoties uz iepriekšējā nodaļā veikto literatūras apskatu.

Tabula 4. Agile metožu lietojumu ieviešana Projektā

	1.fāzes (1.posms)	1.fāzes (2.posms)	2.fāze	3.fāze
Lietotāju stāsti (User Stories)	Ir	Ir	Ir	Ir
Darbu saraksts (Product backlog)	Ir	Ir	Ir	Ir
Iterāciju plānošana (Selected Product Backlog)	Nav	Daļēja*	Ir	Ir
Dienas darbu plānošana (Daily Scrum)	Nav	Nav	Ir	Ir
Faktiskā darbu izpilde (Burn-Down charts)	Nav	Nav	Nav	Ir
Retrospekcijas sanāksmes (Retrospective)	Nav	Nav	Nav	Ir
Iterāciju pārskats (Sprint Review)	Nav	Nav	Nav	Ir
Scrum no Scrum sanāksmes (Scrum of scrums)	Nav	Nav	Nav	Ir
Uz testēšanu balstīta izstrāde (Test Driven Development)	Nav	Nav	Nav	Ir

*Iterāciju plānošana nenotiek pēc *Agile* metodikā noteiktiem principiem, bet gan tos plāno viens cilvēks – vai nu projekta vadītājs, vai sistēmu analītiķis, kurš ir atbildīgs par līgumā noteiktajiem vienošanas protokolu darbiem.

Projektā ieviestos uzlabojumu pirmajā un otrajā fāzē, kā arī plānotos uzlabojumus trešajā fāzē vizuāli var apskatīt darba pielikumā 4. Šeit ir attēlotas visas projekta fāzes, konstatētās problēmas un veiktās izmaiņas projektā.

7.1 Iterāciju plānošana

Iterāciju garums Projektā ir mainīgs un var mainīties no 1 līdz 3 nedēļām, atkarībā no piegādājamo darba vienumu kritiskuma biznesa procesam svarīgumam. Projekta otrajā fāzē, piegādājot vairākas iterācijas, tika konstatēta problēma, ka netika veikta iteratīva plānošana atbilstoši *Agile* metodikai. Tika veikti darbi atbilstoši līgumam, kur pildāmo darba sarakstu iterācijā nosaka projektu vadītājs un analītiķis. Pēc *Agile* metodikas, iterāciju plānošanas sanāksmēs ir jāpiedalās arī izstrādātājiem un testētājiem. Bieži vien uz iterāciju tika solīts pārāk daudz izstrādāt, bet no tā cieš kopējā nodevuma kvalitāte, jo solītais modulis bieži vien ir kļūdains un nefunkcionējošs. Kā arī tika konstatēts, ka projektā iztrūkst sistemātiska pieeja darbam – bieži vien tika papildus ieplānoti un darīti ārpus kārtas neplānoti darbi, kas nebija paredzēti uz iterācijas nodevumu. Tas arī ietekmēja laika trūkumu iterācijā.

Nepareiza plānošana it īpaši atsaucās uz testēšanas darbiem. Testēšanas darbi iterācijas plānošanā ir tikuši novērtēti par maz. Tā kā testēšanu Projektā veica saistītā kompānija K_t, radās aizdomas, ka testēt attālināti nav efektīvi. Lai risinātu minēto problēmu, tika sasaukta projekta

komanda, lai apspriestu šīs problēmas iemeslus un pieņemtu lēmumus, kādā veidā izvairīties no plānoto testēšanas darbu darbietilpību pārtērēšanas. Sanāksmē tika pieaicināti arī testēšanas komandas pārstāvji.

Iespējamie problēmas iemesli tika minēti dažādi – saistītās kompānijas zināšanu trūkums par sistēmas kopējo arhitektūru un biznesu, nekvalitatīvi darba vienumu realizācijas apraksti, grūti saprotami projektējumi un prasību specifikācijas, kā arī pārāk īss laiks, kas ir atvēlēts testēšanai. Izvērtējot šos iemeslus un kopīgi izejot cauri iepriekšējo iterācijā piegādātajiem darbiem, tika konstatēts, ka problēma ir meklējama iterācijas plānošanas procesos. Tā kā testēšana notika attālināti citā kompānijā, plānošanā saistībā ar *Agile* noteikto principu nebija pieaicināti testēšanas komandas pārstāvji. Šī iemesla dēļ, iterācijā ieplānoto darbu darbietilpības tika novērtētas kļūdaini. Atbildīgie sistēmu analītiķi par iterāciju bija precīzi novērtējuši projekta pārvaldes, analīzes un programmēšanas darbus, bet testēšanu bija vērtējuši no sava skatījuma un tikai dēļ tā, ka bija neērti uz katru iterācijas plāna sagatavošanu iepazīstināt ar plānotām darbietilpībām arī testētājus. Testēšanai atvēlētās cilvēkstundas tika aprēķinātas vienkārši ar koeficientu 0.3 no programmēšanai paredzēto cilvēkstundu apjoma.

Plānotais risinājums. Balstoties uz *Agile* iteratīvās izstrādes noteikto metodiku un ņemot vērā autores Pasivara pētījumā [7] rekomendācijas, ir jānosaka nemainīgs iterācijas garums visai projekta fāzei. Ņemot vērā projekta apmērus, maģistra darba ietvertajā projektā ir nepieciešams iterācijas garumu ievērot vismaz 3 nedēļas. Šīs nedēļas ir minimums, lai nodrošinātu korektu risinājumu analīzi, izstrādi un veiktu funkcionālos un integrācijas testus. Darbu plānošana jāveic tā, lai plānotie darbi tiktu izdarīti 3 nedēļu laikā. Pie iterāciju plānošanas projektā ir jāpārdomā, vai paredzēto izstrādes darbus var uzspēt izdarīt iterācijas ietvaros vai arī nē. Ja izstrādājamo darba vienumu nevar uzspēt pilnvērtīgi izstrādāt iterācijā, ir jālemj vai nu par darba vienuma izstrādes pārcelšanu uz nākamo iterāciju vai papildus darba resursu nodrošināšanu. Uz nākamo iterāciju plānošanu tika pieņemts lēmums iesaistīt plānošanā arī testētāju pārstāvi, kurš apstiprinās sistēmu analītiķa noteiktās darbietilpības vai arī nepieciešamības gadījumā ieteiks veikt korekcijas. Tā kā plānošanas sanāksmes pētāmajā projektā netika rīkotas, bet gan iterāciju saplāno atbildīgais analītiķis par iterāciju, tad tālāk aprēķinātas cilvēkstundas tika saskaņotas starp vecāko sistēmu analītiķi, projekta vadītāju un testēšanas pārstāvi.

7.2 Darba vienumu sadalījums

Viena no projekta kļūdām bija tā, ka netika pareizi novērtētas nepieciešamās izmaksas darba vienumu veikšanai, atdodot tās citas valsts kompānijai. Turklāt, tika nepareizi slēgts līgums ar saistīto kompāniju par darba izpildi. Līgums noteica darba samaksu nevis par piegādāto darba vienumu, bet gan par patērētām darba stundām veicot programmatūras izstrādes darbus. Ņemot vērā tabulā 1 minētos riskus un to ietekmējošos faktorus, darba vienuma darbietilpības strauji palielinājās vairākas reizes. Tas nozīmē, ka pirms darba pakas atdošanas izstrādei ārējai kompānijai visticamāk netika izvērtēti riski attiecībā uz izmaksu pieaugumu.

Plānotais risinājums. Balstoties uz iepriekš aprakstītajiem uzdevumu piešķiršanas modeļiem, ko aprakstījuši autori Lamerdorfs un Munčš [5] , darba autors izstrādāja piedāvājumu darba sadales procesam, ko varētu piemērot nākamajiem projektiem. Darba sadalījuma procesā ir ņemta vērā arī darba autora līdzšinējā pieredze dalītajos projektos. Lai veiktu darba vienuma pareizu sadalījumu, ir jāveic sekojoši soļi:

1) bāzējoties uz risku identifikācijas modeli, ir jāveic katra darba vienuma potenciālo risku un ietekmējošo faktoru analīze. Šeit var izmantot Išikavas metodi, lai identificētu iespējamos risku projektā. Rezultāts: *Darba vienumu ietekmējošo risku modeļu saraksts*;
2) bāzējoties uz identificētajiem riskiem un ietekmējošiem faktoriem, darba grupā ir jāizrunā katra darba vienuma potenciālā realizācija un ietekmes uz kopējo realizējamo sistēmu vai atsevišķām sistēmas komponentēm. Rezultāts: *iespējamo risinājumu apraksti*;
3) ir jāveic ārējo projektu izmaksu novērtējums un tajā strādājošo programmētāju kompetences atbilstība realizējamiem darba vienumiem. Tas ļaus novērtēt, vai plānotās izmaksas (aprēķinātas, piemēram, ar COCOMO modeli) būs adekvātas kompetencei, kas rezultatīvi tieši ietekmēs darba vienuma gala iznākumu. Novērtējumu vajadzētu izteikts procentuālā formā, nosakot cik liela ir varbūtība, ka kompānijas realizēs kvalitatīvu produktu atbildoši definētajām prasībām. Rezultāts: *Darba vienumu procentuāls novērtējums, ar iespējamību piešķirt to saistītajam projektam*.

Izmantojot iegūtos datus, projekta vadītājam un vadošajiem sistēmu analītiķiem ir jāveic darba paku un darba vienumu sadalījums pa saistītajiem projektiem.

7.3 Piegādes formēšanas process

Formējot iterāciju piegādes, projektā tika konstatēts, ka piegāžu formēšanai tika atvēlēts pārāk maz laika. Piegādes projektā parasti tika uzsāktas formēt pēdējās iterācijas nedēļas piektdienās dienas otrajā pusē. Apkopojot piegādes dalītajā projektā, kur komponentes piegādā vairākas kompānijas, tā ir liela problēma. It īpaši, ja rodas problēmas, ir pārāk maz laika, lai pilnvērtīgi ar attiecīgās darba vienumu saistīto izstrādātāju saskaņotu nepieciešamās izmaiņas problēmas novēršanai. Sākot piegādes formēšanu, tika slēgts repozitorijs izmaiņām. Tā kā integrācijas testi tika laisti iepriekšējā naktī, tad nākamajā rītā vēl veiktie izstrādes darbi netika pilnvērtīgi pārbaudīti. Kā arī konstatēts, ka netika veiktas iterāciju demonstrācijas – netika veikta informācijas apmaiņa pat to, kas ir izstrādāts un līdz ar to pārējiem komandas dalībniekiem nav priekšstatu par to, kā izstrādātā komponente darbojas un integrējas kopējā sistēmas arhitektūrā.

Plānotais risinājums. Balstoties uz Lefingvela rekomendācijām [8], projektā ir jānosaka laiks, kad iterācijai paredzamās izmaiņas repozitorijā vairs nedrīkst tikt ievietotas. Tas ir jāattiecina uz visiem dalītajā projektā saistītajām kompānijām. Projektā laiks tika noteikts – pēdējās iterācijas nedēļas ceturtdiena 20:00. Pa nakti tika veikti projekta integrācijas testi un nākamajā rītā, tika veikti tikai un vienīgi kritiski labojumi, dēļ kuriem nevar tikt veikta piegādes formēšana. Visi nepieciešamie kritiskie labojumi tika saskaņoti ar darba vienuma analītiķi un programmētāju, neatkarīgi no tā, kurā kompānijā darba vienums ir ticis izstrādāts. Svarīgi ir, lai katrs steidzams risinājums tiktu saskaņots ar piegāžu formētāju vai *Scrum master*. Tikai tad var tikt veikti papildus labojumi. Pārējās kļūdas tika nodotas tālākai iterācijas plānošanai.

7.4 *Scrum* no *Scrum* sanāksmju organizēšana

Tā kā projekts sadarbojas ar ārējās kompānijām, tad aktuāli bija pārdomāt plānu, kā organizēt *Scrum* ikdienas sanāksmes ne tikai vadošās kompānijas komandā, bet arī noskaidrot kāds ir kopējais iterācijas statuss arī saistītajās komandās. Maģistra darba ietvaros, darba autors, balstoties uz publikācijām, kas apraksta projektus cituviet pasaulē, ieteica ieviest darba stilu, katru dienu ar saistīto komandu organizēt tiešsaistes sanāksmes, izmantojot, piemēram, Cisco kompānijas izstrādāto aplikāciju *WebEx* [12] , kas nodrošina attālinātu sanāksmju rīkošanu, nodrošinot prezentācijas translēšanu sanāksmē esošajām personām, kā arī balss un sarakstes atbalstu.

Plānotais risinājums. Balstoties uz Hossaina [6] un Pasivaras [7] aprakstītajām rekomendācijām, katra no projektā saistīto kompāniju komandām, katru dienu organizē iekšēju

Scrum ikdienas sanāksmi un saistītās kompānijas *Scrum master* sagatavo īstu prezentāciju (līdz 3-5 slaidi) par iterāciju darbu gatavību un virzību, ietverot sekojošu informāciju:

1) Izdarītie darbi iepriekšējā dienā un plānotie darbi uz nākamo *Scrum* sanāksmi, norādot arī atbildīgo personu darba izpildei;
2) Konstatētās problēmas un pieņemtie iekšējie risinājumi. Ja risinājums nav pieņemts, problēmu ir jārisina kopēji ar vadošās kompānijas komandu, sazinoties atsevišķi, jo svarīgi ir ievērot *Scrum* iteratīvās sanāksmes rīkošanas principus [13] un neizvērst problēmu analīzi pašā *Scrum* sanāksmē;
3) Saistītās kompānijas *Scrum master* viedokli par iterācijas virzību.

WebEx sanāksmes tika ieplānotas *Microsoft Outlook* [13] plānotājā katru dienu noteiktā laikā pēc vadošās kompānijas *Scrum* ik rīta sanāksmes, kas tika organizēta 10.30 pēc vietējā laika. Šajā tiešsaistes sanāksmē var piedalīties, pieslēdzoties servisam, jebkurš projekta dalībnieks, bet obligāts apmeklējums tika noteikts vadošās kompānijas *Scrum master* (vecākais sistēmu analītiķis), saistītās kompānijas *Scrum master* (vecākais sistēmu analītiķis saistītajā kompānijā) un atbildīgajiem analītiķiem par iterācijā izstrādājamo darba vienumu.

7.5 Retrospekcijas sanāksmju ieviešana

Otrajā fāzē tika veikta iterāciju piegāde klienta testa videi, bet netika pārskatīts iterācijā padarītais, tādējādi, daudzas iterācijās konstatētās un risinātās problēmas palika neizrunātas starp visiem projekta komandas dalībniekiem. Līdz ar to bieži vien viena un tā pati problēma atkārtojās dažādās iterācijās un bija jāmeklē risinājumi, kā to novērst. Viens no priekšlikumiem ir ieviest projektā atskata uz iterāciju sanāksmes (*Sprint retrospective*), kas nodrošinātu piegādāto darba vienumu pārrunāšanu projektā un viedokļu apmaiņu.

Plānotais risinājums. Papildus projektā tika noteikts, ka ar jaunās fāzes izstrādes darbu uzsākšanu, ir jāorganizē arī iterāciju pārskata sanāksmes, lai komanda sanāktu kopā un varētu pārskatīt un pārrunāt paveiktos darbus. Ņemot vērā Pasivaras rekomendācijas [7] , šajā sanāksmē, svarīgi ir no saistītajām komandām piedalīties par iterācijas darbiem atbildīgajiem sistēmu analītiķiem. Ja nav iespējams ierasties klātienē, tad vadošās kompānijas komanda rīko telekonferenci, kā ietvaros tika pārrunāti visu iterācijā piegādāto darba vienumu statusi, iniciētās problēmas un to cēloņi. Izmantojot Cisco *WebEx* rīka nodrošinātajā sanāksmē sagatavoto prezentāciju un citus materiālus, tos ir iespējams translēt tiešsaistē, tādējādi saistītās kompānijas darbiniekam ir iespējams sekot līdz sanāksmei.

7.6 Uz testēšanu balstītas izstrādes ieviešana projektā

Papildus, lai uzlabotu izstrādes gaitu, uz nākamo fāzi projektā tika plānota ideja ieviest un izmēģināt uz testēšanu balstītu izstrādi, tādējādi mēģinot panākt izstrādāto darba vienumu vēl augstāku kvalitāti, tos piegādājot uz iterāciju slēgšanas brīdi. Šāda pieeja stipri atvieglotu testēšanas procesus, it īpaši projektā, kur testēšanu veic cita saistītā kompānijā K_t. Šobrīd programmētājs projektā veic izstrādes darbus atbilstoši projektējumam vai realizācijas aprakstam, ko sastāda sistēmu analītiķis, veic vienībtestus un pēc tam atdod darba vienumu testēšanai kompānijai K_t. Kompānijas K_t veic funkcionālo testēšanu un sekmīgas testēšanas rezultātā lika programmētājiem (pārsvarā jebkuram, kurš ir dotajā momentā brīvs no tiešiem citiem izstrādes darbiem) izveidot automatizētā testa scenāriju. Daļu no testa scenārijiem veica kompānijas K_t programmētāji vai paši testētāji, kuriem bija programmēšanas pieredze. Tas īsti nav pareizi, jo darba vienums tiek otrreiz atgriezts programmēšanā. Turklāt, projektā lielākoties tika iedots citam programmētājam izveidot automatizētos testus, kurš nemaz pats nav programmējis notestēto darba vienumu un bieži radās kļūdas automatizētajos testos, kas savukārt atsaucās uz regresa testiem klienta akcepttesta vidē.

Plānotais risinājums. Lai novērstu šo nepilnību, darba autors, baltoties u autora Lefingvela ieteikumiem [8] , ar nākamo izstrādes fāzi projektā ieviesīs uz testēšanu balstītu programmatūras izstrādi. Mēģinājums tiks organizēts jau no pirmās iterācijas vismaz 3 iterācijas un sekmīga risinājuma gadījumā tiks ieviests projektā patstāvīgi, kā arī tiks ieteikts citiem projektiem kā piemērs izstrādes procesu uzlabošanai. Starp iterācijām atskatā uz iterāciju sanāksmēs tiks izrunātas konstatētās problēmas un meklēti risinājumi, lai vēl uzlabotu šo procesu.

Izmēģinājuma cikla sākumā tiks mainīti un ieviesti sekojoši nosacījumi izstrādei:

1) Sistēmu analītiķis sagatavojot realizācijas aprakstu vai projektējumu pie darba vienuma apraksta scenāriju, kādas darbības sistēmā ir jāveic, lai automatizētais tests būtu veiksmīgs un izpildītu lietojumu gadījumā (*use case*) aprakstīto biznesa procesu;
2) Programmētājs iepazīstas ar scenāriju un izpēta realizācijas aprakstu vai projektējumu. Ja rodas jautājumi, vēršanas pie sistēmu analītiķa, kurš ir atbildīgs par darba vienuma realizācija;
3) Programmētājs izstrādā darba vienuma automatizēto testa pirmo scenāriju darba vienuma aprakstītam solim;
4) Veic izstrādi atbilstoši realizācijas aprakstam, ar mērķi izstrādāt programmatūras kodu tā, lai iepriekš izstrādātais scenārijs būtu sekmīgs;

5) Izstrādā nākamos scenārijus iteratīvi izpildot aprakstītos iepriekšējos divus soļus. Pēc katra soļa izstrādes svarīgi ir pārbaudīt vai arī iepriekšējie scenāriji ir veiksmīgi, nepieciešamības gadījumā izlabojot nepilnības izstrādātajā programmatūras kodā;
6) Rezultātā ir jābūt reizē izstrādātam automatizētam testam un darba vienumam;
7) Darba vienums tiek atdots testēšanas komandai.

Ar šādu pieeju projektā tiks risināti jaunie darbi un izmaiņas. Kļūdas programmatūras kodā tiks labotas, nepieciešamības gadījumā papildinot automatizēto testu programmatūras kodu. Šāda pieeja stipri atvieglos testēšanu saistītai kompānijai K_t, jo savlaicīgi tiks novērstas kļūdas programmatūras kodā un testētāji varēs vairāk pievērsties funkcionāliem testiem.

Veicot funkcionālo testēšanu, testēšanas kompānijas komanda, informē par to analītiķi, kurš ir veicis atbilstošā darba vienuma projektēšanu un realizācijas apraksta veidošanu. Ja arī analītiķis konstatē kļūdu, darba vienums tiek nomainīts atpakaļ uz izstrādi. Ja tomēr kļūda neeksistē un testētājs ir kļūdījies, analītiķis atgriež darba vienumu testēšanai. Protams, ir jāņem vērā viena nianse, lai atvieglotu darba procesu analītiķim. Testētājam ir jābūt pārliecībai, ka testēšanā nodotais darba vienums tiešām nestrādā atbilstoši plānotajam un aprakstītajam realizācijas aprakstā vai projektējumā. Ja testētājs nav pārliecināts, viņam šai projektā ir jāsazinās vispirms ar analītiķi un jānoskaidro situācija.

8 Nobeigums un secinājumi

Maģistra darba ietvaros tika izpētīts Latvijā izstrādāts dalīts projekts, kas 3 gadu laikā pamazām mainīja izstrādes modeli, no ūdenskrituma pakāpeniski pārejot uz spējās izstrādes modeli. Izpētot papildus literatūru, kurā tika aprakstīti citi projekti pasaulē, radās idejas kā vēl papildus uzlabot šo projektu, mēģinot iestrādāt papildus metodes, kuras ir pielāgotas iteratīvai izstrādei. Veiktais literatūras pētījums ļāva labāk izprast, ka ir iespējams strādāt iteratīvi dalītos projektos, kad projektu attīsta vairākas kompānijas, kuras bieži vien neatrodas vienā valstī un pat vienā laika joslā. Papildus veiktā analīze Projektā ļāva konstatēt nepilnības esošajos spējās izstrādes procesos pirmajā un otrajā izstrādes fāzē un rast veidus, kā tos uzlabot, sagatavojot projektu plānotai trešajai fāzei.

Izejot cauri projekta pirmajiem divarpus gadiem, darba autors konstatēja, ka iteratīvā izstrāde projektā tikusi ieviesta lēni un prasījusi daudz resursus no visiem projektā esošajiem dalībniekiem, jo nemitīgi darba procesi bija jāuzlabo un jāpielāgo, kas arī zināmā mērā atsaucās uz darbiem un to kvalitāti. Izpētītā literatūra un veiktā projekta analīze ļāva apskatīt un aprakstīt risinājumus no citiem projektiem pasaulē, ko projekta kompānija varēs turpmāk izmantot, lai nākamos projektus varētu izstrādāt daudz kvalitatīvāk, balstoties uz jau izpētītiem spējās programmatūras izstrādes principiem.

Papildus tam, kompānijā, kurā tiek ieviests projekts, ar darba autora palīdzību tika mainīti esošie kvalitātes apraksti, mainot iepriekš noteikto ūdenskrituma modeli uz jaunu, kurā tiek pielietotas iteratīvās izstrādes mehānismi. Tādēļ, turpinot iesākto darbu, kompānijas kvalitātes vietnē tiks detalizēti aprakstīti visi posmi iteratīvai izstrādei dalītajos projektos, lai tos varētu pielietot jaunie plānotie projekti, neieguldot papildus laiku literatūras izpētē vai no jauna mēģinot ieviest iteratīvu izstrādi projektā patstāvīgi, kā tas notika Projektā. Kā jau iepriekš minēts, tas prasīja papildus laiku, daudzas izmaiņas projekta procesos un, līdz ar to, arī cilvēku un finanšu resursus. Ar rīcību izpētes mehānismu tiek izpētīti un ieviesti jauni spējās izstrādes principi projekta jaunajā fāzē un veiksmes gadījumā aprakstīti un papildināti uzņēmuma kvalitātes sistēmā, aprakstot tos principus jau kā izstrādē pielietojamas metodes.

9 Izmantotā literatūra un avoti

[1] Suprika Vasudeva Shrivastava, Hema Date. Distributed Agile Software Development: A Review. Journal of computer science and engineering. – CoRR abs/1006.1955, 2010.

[2] Laurie A. Williams, Mladen A. Vouk: Agile Software Development. Wiley Encyclopedia of Computer Science and Engineering 2008.

[3] Samireh Jalali, Claes Wohlin: Agile Practices in Global Software Engineering - A Systematic Map. – ICGSE, 2010. – pp. 45-54.

[4] Confluence rīks. [tiešsaiste]. Pieejams: https://www.atlassian.com/software/confluence .

[5] Ansgar Lamersdorf, Jürgen Münch: Model-Based Task Allocation in Distributed Software Development, 2010. – pp. 37-53.

[6] Emam Hossain, Muhammad Ali Babar, Hye-young Paik: Using Scrum in Global Software Development: A Systematic Literature Review. – ICGSE, 2009. – pp. 175-184.

[7] Maria Paasivaara, Sandra Durasiewicz, Casper Lassenius: Using scrum in a globally distributed project: a case study. – Software Process: Improvement and Practice 13(6), 2008. – pp. 527-544.

[8] Dean Leffingwell. Mastering the Iteration: An Agile White Paper. Rally Software Development Corporation 2007.

[9] Jason H. Sharp and Sherry D. Ryan. Best Practices for configuring Globally Distributed Agile teams. A Publication of the Association of Management, Journal of Information Management. – ISSN #1042-1319, 2008.

[10] Jeff Sutherland, Anton Viktorov, Jack Blount, Nikolai Puntikov: Distributed Scrum: Agile Project Management with Outsourced Development Teams. – HICSS, 2007. – pp. 274.

[11] Valsa Koshy. What is Action Research? – SAGE Publications Ltd., 2011.

[12] Cisco Webex rīks. [tiešsaiste]. Pieejams: http://www.webex.com/

[13] Ken Schwaber and Jeff Sutherland. The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game. Scrum.org, 2013.

[14] Microsoft Outlook rīks. [tiešsaiste]. Pieejams: http://office.microsoft.com/en-001/outlook/

[15] Gaston Trauffler**.** Ishikawa Diagram cause and effect diagram. The National Agency for Innovation and Research in Luxembourg, 2008.

[16] Helena Holmström, Brian Fitzgerald, Pär J. Ågerfalk, Eoin Ó Conchúir: Agile Practices Reduce Distance in Global Software Development. – IS Management 23(3), 2006. – pp. 7-18.

[17] Zane Galviņa, Darja Šmite. Software Development Processes in Globally Distributed Environment. Computer Science and Information Technologies. – Scientific Papers, Univercity of Latvia. Vol. 770, 2011.

[18] Skype rīks. [tiešsaiste]. Pieejams: http://www.skype.com/

[19] Microsoft Messenger rīkls [tiešsaiste]. Pieejams: http://windows.microsoft.com/en-us/messenger/

[20] Subversion rīkls [tiešsaiste]. Pieejams: http://subversion.tigris.org/

[21] Git rīks [tiešsaiste]. Pieejams: http://git-scm.com/

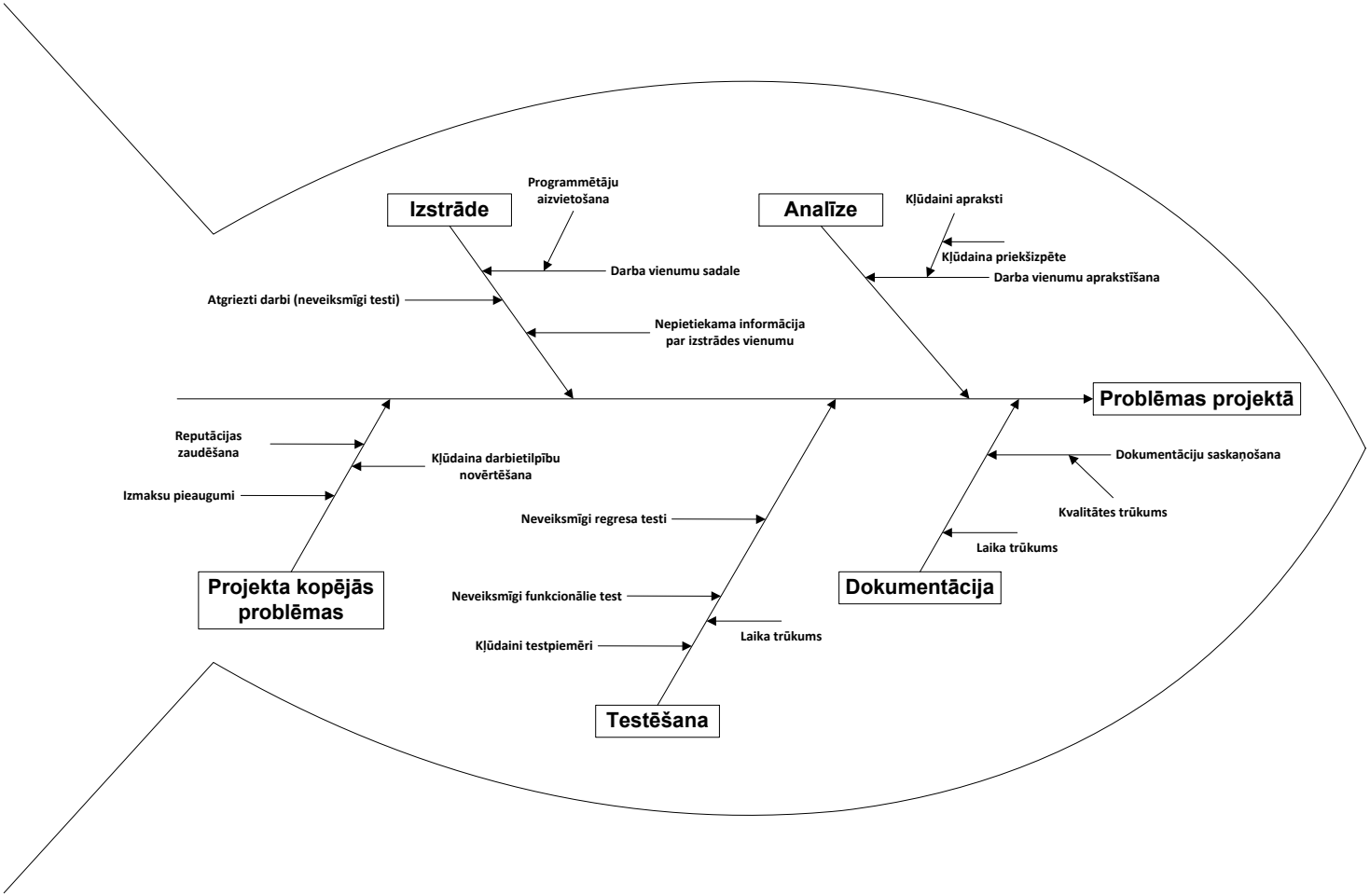
[22] An Introduction to Planning Poker. [tiešsaiste]. Pieejams: http://agile.dzone.com/articles/introduction-planning-poker

[23] Eoin Ó Conchúir, Helena Holmström, Pär J. Ågerfalk, Brian Fitzgerald: Exploring the Assumed Benefits of Global Software Development. – ICGSE, 2006. – pp. 159-168.
[24] Darja Smite, Çigdem Gencel: Why a CMMI Level 5 Company Fails to Meet the Deadlines?. – PROFES, 2009. – pp. 87-95.
[25] Oshri I., Korlarsky J., & Willcocks L.P. The Handbook of Global Outsourcing and Offshoring. – Palgrave Macmillan, 2009. – pp. 129-131.

10 Pielikumi

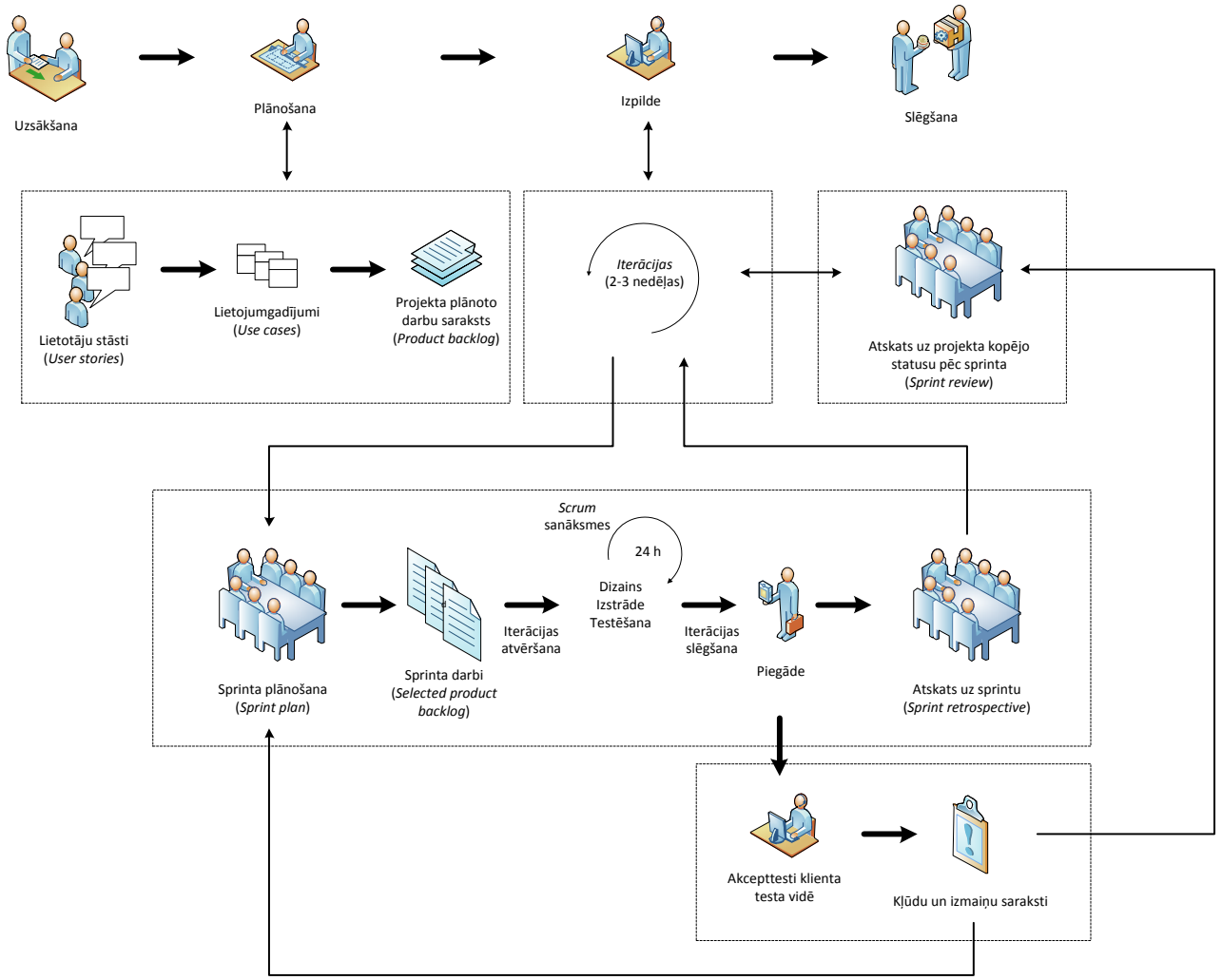
10.1.1.1 Pielikums 1

Cēloņu raksturojumu shēma projektā



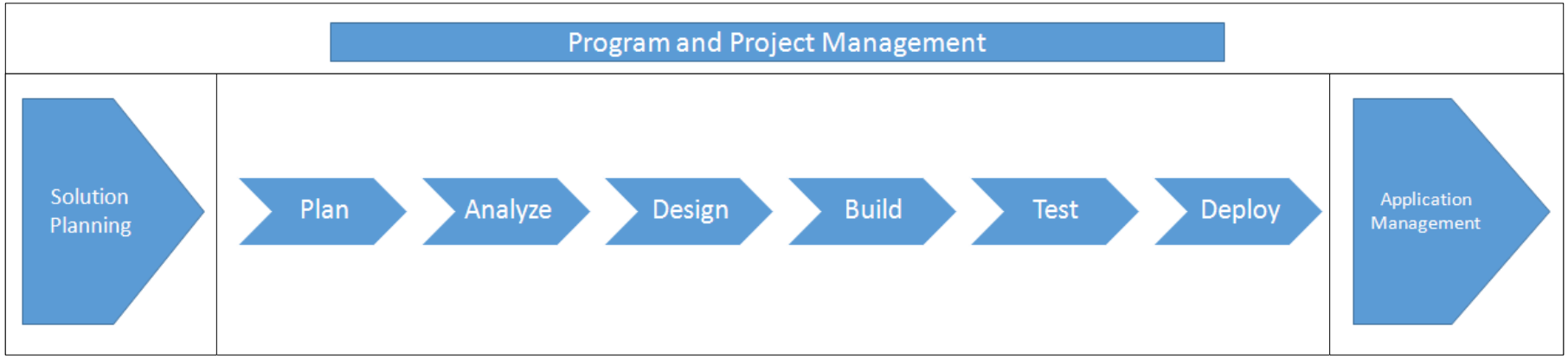
10.1.1.2 Pielikums 2

Izstrādātais spējās izstrādes modelis Projektā

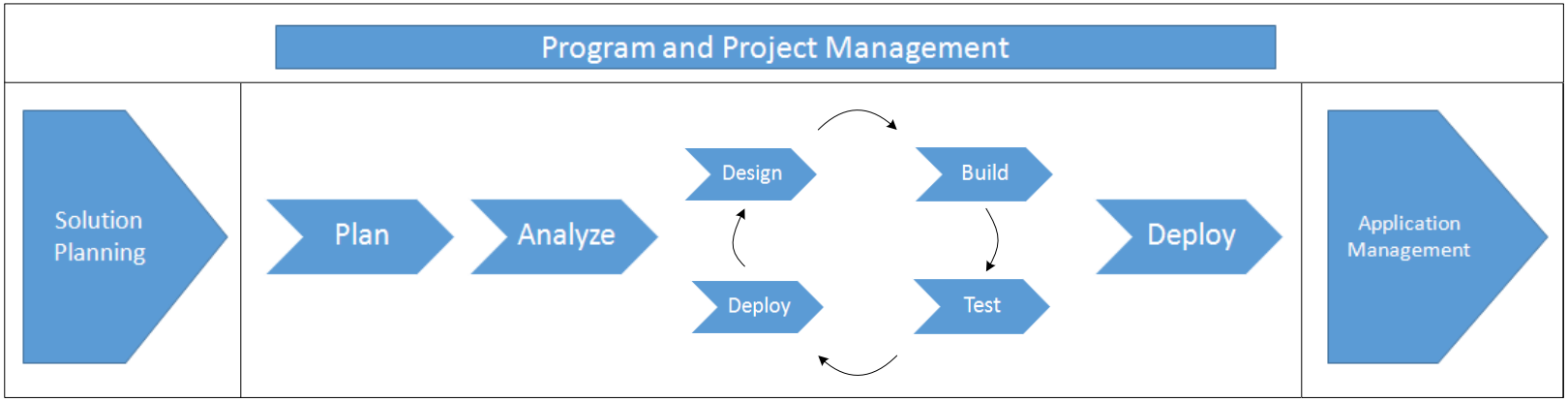


10.1.1.3 Pielikums 3

Kvalitātes aprakstos noteiktais izstrādes modelis projekta kompānijā

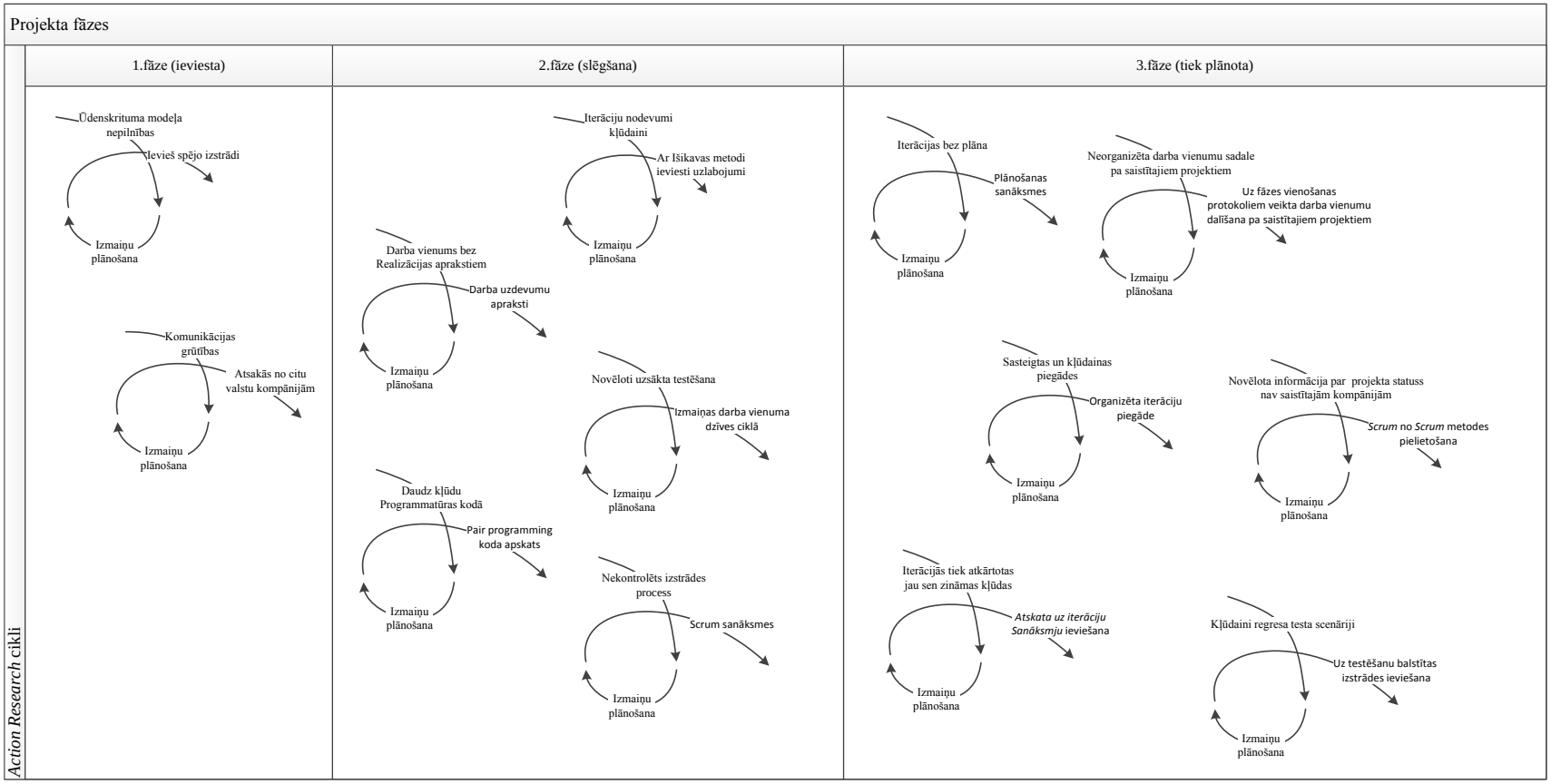


Kvalitātes aprakstos ieviestais iteratīvās izstrādes modelis projekta kompānijā



10.1.1.4 Pielikums 4

Ieviestie risinājumi projektā



Maģistra darbs: **Metodes spējās izstrādes pamatprincipu ievērošanai dalīto projektu kvalitātes uzlabošanai**

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

Autors: _____

(Autora paraksts)

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

Darba vadītājs: _____

(Vadītāja paraksts)

Darbs iesniegts **maģistrantūras sekretariātā** _____.

(Iesniegšanas datums)

Ar šo es apliecinu, ka darba elektroniskā versija ir augšupielādēta LU informatīvajā sistēmā.
Studiju metodiķe: _____.

(Metodiķes paraksts)

Recenzents: _____

(Akad.amats, zin.grāds, vārds, uzvārds)

Darbs aizstāvēts maģistra gala pārbaudījuma komisijas sēdē

_____ prot. Nr. _____

(Darba aizstāvēšanas datums)

Komisijas sekretārs: _____

(Sekretāra paraksts)