LATVIJAS UNIVERSITĀTE

Baiba Apine

Programmatūras izstrādes procesa diagnosticēšana un attīstīšana

Promocijas darbs

Rīga - 2003

LATVIJAS UNIVERSITĀTE
RĪGAS INFORMĀCIJAS TEHNOLOĢIJAS INSTITŪTS

Baiba Apine

Programmatūras izstrādes procesa diagnosticēšana un attīstīšana

Promocijas darbs

datorzinātņu doktora (Dr.sc.comp.) zinātniskā grāda iegūšanai

Nozare: datorzinātnes

Apakšnozare: datoru un sistēmu programmēšana

Zinātniskais vadītājs:
Juris Borzovs
asoc.prof., Dr.hab.comp.sc.,
Rīgas Informācijas tehnoloģijas institūta direktors

Zinātniskais konsultants:
Uldis Sukovskis
asoc.prof., Dr.sc.ing.,
Rīgas Informācijas tehnoloģijas institūta
Sistēmu modelēšanas laboratorijas vadītājs

Rīga - 2003

Saturs

Ievads

Programmatūras izstrāde ir viens no riskantākajiem ražošanas procesiem gan pasūtītājam, gan arī programmatūras izstrādātājiem. Kvalitatīvas un pasūtītāja vēlmēm atbilstošas programmatūras iegūšanai svarīgs ir kvalitatīvs, saprotams un prognozējams izstrādes process.

Analizējot pasaules praksi programmatūras izstrādes procesa jomā, ir redzams, ka pēdējo gadu laikā notikušas un turpina notikt ievērojamas pārmaiņas. Tiek pārstrādāti programminženierijas standarti, kas attiecas uz programmatūras izstrādes procesa un produkta kvalitāti. Piemēram, ISO/IEC saimes standarti, turpmāk pievēršot vairāk uzmanības pieredzes uzkrāšanai par programmatūras izstrādes procesu. Proti, programmatūras izstrādes procesa uzlabošanā vērojama nostājas maiņa: tas, kā ražojam programmatūru, ir labi, taču savu darba gaitā radušos pieredzi jāpieraksta, jāuzkrāj un jādara pieejamu citiem izstrādes procesa dalībniekiem. Ideja šķiet vienkārša, tomēr praksē grūti realizējama.

Programminženierijā nemainīgi aktuāla ir izstrādes procesa darbietilpības un izstrādei nepieciešamā laika prognozēšana, kas ir vitāli nepieciešama plānošanas procesā. Desmitiem gadu zinātnieki visā pasaulē nodarbojušies ar jaunu metožu izstrādi, taču šī joma vēl joprojām turpina attīstīties, saglabājoties nemainīgi augstam pieprasījumam praksē. Iemesls ir tāds ka, lai arī metodes piedāvā formalizēt prognozēšanu, to lietošana praksē tomēr liela mērā ir māksla.

Programmatūras izstrādes procesam kļūstot sarežģītākam, arvien pieaugot sadalītās izstrādes popularitātei, būtiska ir izstrādātāju savstarpējā komunikācija un spēja objektīvi paraudzīties uz izstrādes procesu.

Darba mērķis bija izstrādāt rekomendācijas programmatūras izstrādes procesa uzlabošanai, lai tas būtu saprotams un prognozējams visiem procesa dalībniekiem katrā laika momentā izstrādes procesa gaitā.

Darba mērķa sasniegšanai tika risināti šādi **uzdevumi**.

1. Pētīt programmatūras izstrādes risku analīzes procesu un programmatūras izstrādes procesu ietekmējošos riskus.
2. Pētīt programmatūras izstrādes mērīšanas procesu un problēmas, kas saistītas ar mērīšanas procesa ieviešanu programmatūras izstrādes uzņēmumos.

3. Izstrādāt metodoloģiju, kas, integrējot divus programmatūras izstrādes procesa atbalsta procesus - izstrādes procesa mērīšanu un risku pārvaldību - radītu prognozējamu vidi izstrādes procesam, sniedzot informāciju par izstrādes procesa stāvokli un vidi, kurā izstrāde notiek.

Promocijas darbs sastāv no ievada, divām nodaļām, nobeiguma, tabulu, attēlu un literatūras saraksta un trim pielikumiem.

Pirmajā nodaļā "1. Programmatūras izstrādes process un to ietekmējoši faktori" pētīti programmatūras izstrādes procesu ietekmējošie riski. Aplūkoti gan standarti, gan metodes risku pārvaldībai. Darba gaitā veikta projektu pārvaldnieku aptauja par programmatūras izstrādes procesu ietekmējošiem riskiem un to apkarošanas preventīvām un korektīvām darbībām ar mērķi atrast riskus, kas aktuāli programmatūras izstrādātājiem Latvijā. Tāpat pētīta izstrādes procesa mērīšana, aplūkojot gan standartus, gan dažādas metodoloģijas izstrādes procesa mērīšanai. Darbā apkopota pieredze, kas iegūta praktiski strādājot pie mērījumu programmu ieviešanas, analizētas ar izstrādes procesa mērīšanu saistītās problēmas un piedāvāti šo problēmu risinājumi.

Pirmajā nodaļā arī aplūkotas formālas programmatūras izstrādes darbietilpības prognozēšanas metodes, analizējot, kādus izstrādes procesu ietekmējošus faktorus ņem vērā katra no tām. Metodes pielietotas dažādiem reāliem programmatūras izstrādes projektiem ar mērķi demonstrēt un analizēt aplūkoto metožu izmantošanas īpatnības un izstrādāt rekomendācijas to lietošanai praksē.

Otrajā nodaļā "2. Mērījumu un risku bāzētas projektu plānošanas atbalsta metode" piedāvāta programmatūras izstrādes procesa plānošanas metode MERIME (Mērījumu un risku bāzētas projektu plānošanas atbalsta metode - MERIME), kas integrē mērīšanas un risku pārvaldības procesus. Izstrādātas vadlīnijas un sagataves MERIME metodes ieviešanai un izmantošanai programmatūras izstrādes uzņēmumā. Metodes izmantošana izmēģināta praksē un darbā piedāvāta metodes izmantošanas gaitā iegūto izstrādes procesa uzlabojumu analīze.

Pirmajā pielikumā "I Pielikums. Funkcijpunktu skaita iegūšanas metodika" dota šī darba autores izveidotā programmatūras apjoma prognozēšanas metodika,

kuru sekmīgi izmanto programmatūras izstrādes uzņēmumā un komercbankās Latvijā. Šāda metodika nepieciešama, jo izstrādājamā produkta apjoms ir viens no svarīgākajiem izstrādājamā produkta raksturlielumiem.

Otrajā un trešajā pielikumos "II Pielikums. Mērījumu datu savākšanas formas" un "III Pielikums. Uzņēmuma ALFA Mērījumu programmas plāns" ir apkopota pieredze Mērījumu programmas ieviešanai uzņēmumā.

Zinātniskā darba galvenais rezultāts: izstrādāta metodoloģija programmatūras izstrādes procesa uzlabošanai, kas visa programmatūras izstrādes procesa dzīves cikla laikā nodrošina objektīvas informācijas uzkrāšanu un komunikāciju visām izstrādes procesā iesaistītajām pusēm. Šo informāciju izmanto izstrādes procesa operatīvajai vadībai, radot caurspīdīgu un prognozējamu vidi visa izstrādes procesa laikā.

Zinātniskā novitāte: līdz šim programmatūras izstrādes procesa risku pārvaldība un izstrādes procesa mērīšana attīstījās nesaistīti. Ir bijuši mēģinājumi risku pārvaldību saistīt ar mērīšanu, mērījumu rezultātus izmantojot risku identificēšanai. Šeit abi procesi integrēti no cita skatu punkta – autore ir integrējusi programmatūras izstrādes procesa risku pārvaldību un izstrādes procesa mērīšanu, lai abu procesu radītos rezultātus izmantotu izstrādes procesa plānošanai un informācijas par izstrādes procesu komunikācijai visām procesā iesaistītajām pusēm.

Laika periodā no 1996. līdz 2002. gadam **zinātniskā darba rezultāti apspriesti**:

- 5 starptautiskās konferencēs – starptautiskajā operāciju pētīšanas metožu un simulācijas izmantošanas ražošanas un apkalpošanas sfērās konferencē (Rīgā, Latvijā, 1996. gadā), starptautiskajā datu bāzu un informācijas sistēmu konferencē DB&IS (Rīgā, Latvijā, 1998. gadā), starptautiskajā datu bāzu un informācijas sistēmu konferencē DB&IS (Viļņā, Lietuvā, 2000. gadā), starptautiskajā datu bāzu un informācijas sistēmu konferencē DB&IS (Tallinā, Igaunijā, 2002. gadā), starptautiskajā informācijas sistēmu attīstības konferencē ISD (Rīgā, Latvijā, 2002. gadā);

- Rīgas Informācijas tehnoloģijas institūta semināros (Rīgā, 2000. – 2002. gadā);
- LU Zinātniskajā konferencē (Rīgā, 2002. un 2003. gadā);
- Baltic Software Metrics Association (BaSMA) ikgadējos semināros Rīgā (2000. gadā) un Viļņā (2002. gadā);
- Zinātniskā seminārā Software Technology Transfer (Espoo, Somijā, 2002. gadā).

Promocijas darbā ietvertā darba rezultāti apkopoti 8 autores publikācijās.

1. Programmatūras izstrādes process un to ietekmējoši faktori

Lai izstrādātu rekomendācijas programmatūras izstrādes procesa uzlabošanai, svarīgi izprast, kā novērtēt izstrādes procesu, pēc kādiem kritērijiem salīdzināt procesus un procesu rezultātā radītos produktus. Šādu informāciju procesa un produkta novērtēšanai sniedz mērījumi, bet pats mērīšanas process ir būtisks izstrādes procesa atbalsta process.

Tomēr prakse rāda, ka izstrādes procesa mērīšana, lai arī faktiski vienīgais objektīvais informācijas avots par izstrādes procesu, netiek bieži izmantota. Šajā nodaļā pētīta izstrādes procesa mērīšana kā atbalsts izstrādes procesam ar mērķi identificēt mērīšanas procesa veiksmes faktorus, problēmas mērīšanas procesā un piedāvāt šo problēmu risinājumus.

Par programmatūras izstrādes procesu nevar spriest atrauti no vides, kurā izstrāde notiek. Ir dažādi faktori, kas ietekmē izstrādes procesu. Lai apzinātu un pārvaldīto šos faktorus, jāiedibina un jārealizē risku pārvaldības process. Šajā nodaļā pētīts risku pārvaldības process ar mērķi noskaidrot tā vietu programmatūras izstrādes procesā, identificēt problēmas risku pārvaldības procesa īstenošanā un piedāvāt šo problēmu risinājumus.

Jau vairāk nekā desmit gadus programmatūras izstrādes procesa plānošanas atbalstam izmanto dažādas programmatūras izstrādes procesa raksturlielumu prognozēšanas metodes. Šajā nodaļā aplūkotas dažādas metodes ar mērķi izanalizēt, uz kādiem izstrādes procesa raksturlielumiem koncentrējas šīs metodes un kādus izstrādes procesa riskus tās ņem verā.

1.1. Programmatūras izstrādi ietekmējošie riski

Nodaļas mērķi ir identificēt riskus, kas aktuāli programmatūras projektu izstrādātājiem un analizēt metodes un standartus risku pārvaldībai ar mērķi identificēt problēmas risku pārvaldības procesa īstenošanā.

1.1.1. Riska definīcija

Risku definē kā varbūtisku zaudējumu, traumu vai bojājumu [GOV81]. Programminženierijā risku definē kā iespējamību, ka programmatūras izstrāde būs nesekmīga. Proti, iespējamība, ka izvēlētā teorija, principi vai tehnoloģija nespēs radīt programmproduktu tādu, kāds tas bija gaidāms [SEI92].

Risku pārvaldībā izmanto vairākus jēdzienus, kuri tiks turpmāk paskaidroti [CIS02].

1. Drauds – potenciāls notikums, kas izraisītu organizācijai, projektam u.c. nevēlamas sekas. Piemēram, drauds programmatūras izstrādes projektam ir vadošā programmētāja vēlme projekta izstrādes kulminācijas brīdī pāriet darbā citā organizācijā.
2. Riska iespējamība – varbūtība, ar kādu drauds realizēsies. Iespējamību var novērtēt gan skaitliski (piemēram, 10%), gan kvalitatīvi (piemēram, liela, maza utml.). Piemēram, iespējamība, ka vadošais programmētājs pāries darbā citā organizācijā ir 50%, jo viņš jau vairākkārt saspringtās situācijās ir brīdinājis par šādu iespējamību, bet nekur nav aizgājis.
3. Riska ietekme – drauda realizācijas seku novērtējums. Riska ietekmi var novērtēt gan naudā (Ls 1000), gan arī kvalitatīvi (piemēram, liels, mazs utml.). Ir draudi, kurus ārkārtīgi grūti novērtēt naudā. Piemēram, sabojāts uzņēmuma tēls pēc nomelnojošas publikācijas presē.
4. Risks – varbūtiskais zaudējums drauda realizācijas gadījumā. Piemēram, ja riska varbūtība ir 10%, bet riska ietekme Ls 1000, tad risks ir 10% * 1000 = 100 Ls. Gadījumā, ja riska varbūtība un riska lielums ir vērtēti kvalitatīvi, tad risku nosaka pēc varbūtības un ietekmes matricas. Rūtiņās parādīts riska novērtējums, ko izsaka varbūtības reizinājums ar ietekmes vērtību.

Risku novērtējumus izmanto risku pārvaldībā, lai piešķirtu prioritātes un atbilstoši plānotu aktivitātes risku apkarošanai. Piemēram, aktivitātes riska, kura iestāšanās ir ļoti iespējama un ietekme ir liela, apkarošanai jāplāno un jārealizē vispirms, tikai pēc tam pievēršoties tiem riskiem, kuru iestāšanās ir maz iespējama un ietekme – zema.

1.1.2. Standarti risku pārvaldībai programmatūras izstrādes procesā

Risku vadība ir viens no programmatūras izstrādes atbalsta procesiem, kuru rekomendē veikt programminženierijas standarti [CMM99], [IEE99].

1.1.2.1. CMMI –SE/SW ieteiktā risku vadība

Saskaņā ar CMMI –SE/SW ieteikumiem [CMM99], risku pārvaldība jārealizē programmatūras izstrādes projektos, ja tas ir nepieciešams. Risku vadībā ietilpst turpmāk nosauktās un aprakstītās aktivitātes.

1. Sagatavošanās risku pārvaldībai, kuras laikā nosaka risku avotus un kategorijas, nodefinē risku parametrus un risku vadības stratēģiju.

Risku analīzes mērķis ir apzināt tos draudus, kas liedz projektam sasniegt tam izvirzītos mērķus. Riski var būt gan iekšēji, gan ārēji attiecībā pret projektu. Riskus dala kategorijās, ievērojot to ietekmi uz projekta izmaksām, izpildes laiku un produkta kvalitāti. Risku kategoriju noteikšanas rezultāts ir gan iekšējo, gan ārējo risku saraksts un risku kategoriju saraksts.

Riska parametrus izmanto risku analīzei, klasifikācijai un kontrolei. Risku parametru definēšanas rezultātā rodas risku novērtēšanas, klasifikācijas un prioritāšu piešķiršanas kritēriji un risku pārvaldības prasības (risku novērtēšanas intervāli u.c.).

Risku vadības stratēģija aplūko metodes un rīkus, kādus izmantos risku identificēšanai, analīzei un apkarošanai, projektam specifiskus risku avotus. Nosaka, kā riskiem tiks piešķirtas prioritātes, globāli sliekšņi, parametri un kritēriji, lai uzsāktu aktivitātes risku apkarošanai, atbildība, risku mērījumu definīcija, lai sekotu risku statusam, laika intervāli risku novērošanai un novērtēšanai. Risku vadības stratēģijas izstrādes rezultātā rodas projekta risku vadības plāns.

2. Risku identificēšanas un analīzes laikā identificē un analizē riskus ar mērķi noteikt to relatīvo svarīgumu. Risku identificēšanas rezultātā rodas identificēto risku, to kontekstu, nosacījumu un seku saraksts.

Risku analīzes laikā notiek risku izvērtēšana, klasificēšana un prioritāšu piešķiršana. Parasti riski tiek sakārtoti pēc iespējamības un ietekmes, bet var būt arī citi parametri.

3. Risku apkarošana nozīmē, kur vien iespējams mazināt to ietekmi uz projekta mērķiem. Risku apkarošanas ietvaros notiek risku apkarošanas plānu sastādīšana, šo plānu realizācija.

Risku apkarošanas plānu sastādīšanas mērķis ir projekta svarīgākajiem riskiem sastādīt risku apkarošanas plānus saskaņā ar risku vadības stratēģiju. Risku apkarošanas plāni nepieciešami tikai svarīgākajiem riskiem, mazāk svarīgajiem ir pietiekami nodefinēt tikai to kontroli. Svarīgākajiem riskiem jāizstrādā vairākas apkarošanas alternatīvas. Risku apkarošanas plānu sastādīšanas rezultāts ir katra identificētā riska vadības metodes, risku apkarošanas plāni un atbildīgie par katra riska vadību.

4. Risku apkarošanas plānu realizācijas laikā seko līdzi risku statusam, periodiski tos pārbaudot un realizē risku vadības plānus, kā arī papildina risku statusu sarakstus, risku varbūtības, sekas, darbību risku apkarošanai sarakstus.

1.1.2.2. Standarta IEEE P1540 ieteikumi risku pārvaldības procesam

Risku vadības process ir nepārtraukts process visā programmatūras izstrādes procesa dzīves ciklā, kurā ietilpstošās aktivitātes un to secība parādītas 1. attēlā.

Risku vadības plānošanas un realizācijas mērķis ir iedibināt risku vadības procesu, nosaucot, kurš būs atbildīgs par risku vadību, nodefinējot risku vadības procesu un piešķirot resursus. Risku vadības plānošana notiek projekta sākumā un visa informācija tiek iekļauta projekta risku vadības plānā.

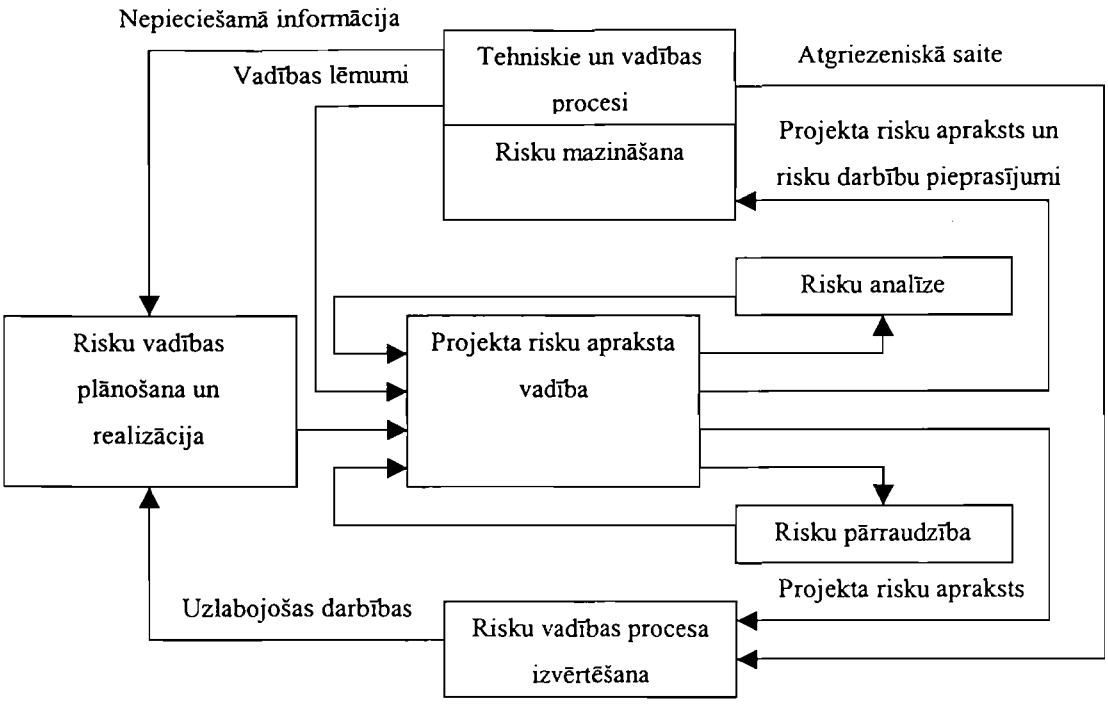
Projekta risku apraksta vadības mērķis ir radīt mehānismu, kā uzkrāt informāciju par riskiem, to apkarošanas vēsturi tā, lai šī informācija būtu pieejama visiem procesa dalībniekiem un būtu ērti lietojama.

Risku analīzes mērķis ir identificēt riskus, to avotus, noteikt risku sliekšņus, patreizējos risku statusus un atrast alternatīvas darbības risku mazināšanai.

Risku pārraudzības laikā seko līdzi risku statusiem, regulāri uzturot projekta risku aprakstu.

Risku mazināšanas laikā realizē darbības risku mazināšanai un uzrauga šo darbību efektivitāti.

Risku vadības procesa izvērtēšanas laikā analizē un izvērtē pašu risku pārvaldības procesu ar mērķi uzlabot šo procesu.



1. attēls. Risku pārvaldības procesa modelis saskaņā ar standartu IEEE P1540

1.1.3. Risku pārvaldības metožu piemēri

1.1.3.1. Condition-Transition-Consequence (CTC) metode

CTC metode risku uzlūko kā trīsdaļīgu riska entītiju, kura sastāv no turpmāk nosauktiem komponentiem [GLU94].

1. Nosacījums, kas apraksta patreiz radošos bažu apstākļus. Nosacījums šeit ir drauds nodaļā "1.1.1 Riska definīcija" dotajos terminos.
2. Pāreja laikā no nosacījuma uz sekām.
3. Sekas, kas apraksta potenciālo iznākumu gadījumā, ja drauds realizējas.

Riska entītiju apraksta šādi: "Pie <Nosacījuma> pastāv bažas, ka (varbūt) iestāsies <Sekas>".

CTC riska entītijai ir atribūti, kas to raksturo precīzāk. Piemēram, iespējamība, ka sekas iestāsies, to ietekme, laika informācija, metrikas riska kontrolei. Viens no svarīgākajiem CTC riska atribūtiem ir riska konteksts. Riska konteksts apraksta notikumus, apstākļus un to savstarpējās saites, kas var ietekmēt

seku iestāšanos. CTC riska entītija faktiski ir riska apraksts, kuru izmanto, lai plānotu aktivitātes risku apkarošanai.

Piemēram, riska entītijas apraksts saskaņā ar CTC metodi ir: pie nosacījuma, ka lietotāja saskarnei jābūt realizētai X Windows un mums nav pieredzes X Windows saskarnes realizācijā, pastāv bažas, ka (varbūt) lietotāja saskarnes daļa netiks pabeigta laikā.

Konteksts: Lietotāja saskarne ir svarīga sistēmas sastāvdaļa un mums nav neviena darbinieka, kas būtu apmācīts darbam X Windows. Mēs visi esam pašmācības ceļā mēģinājuši apgūt šo vidi, bet tā ir pārāk sarežģīta.

Galvenais uzsvars CTC risku pārvaldības metodē ir uz risku identificēšanu, nevis uz seku noteikšanu.

Arī darbus, kas veicami, lai apkarotu riskus, CTC metode apraksta analogi riskiem: "Pie <Nosacījuma> jāizpilda <Uzdevums>, tad iestāsies <Sekas>".

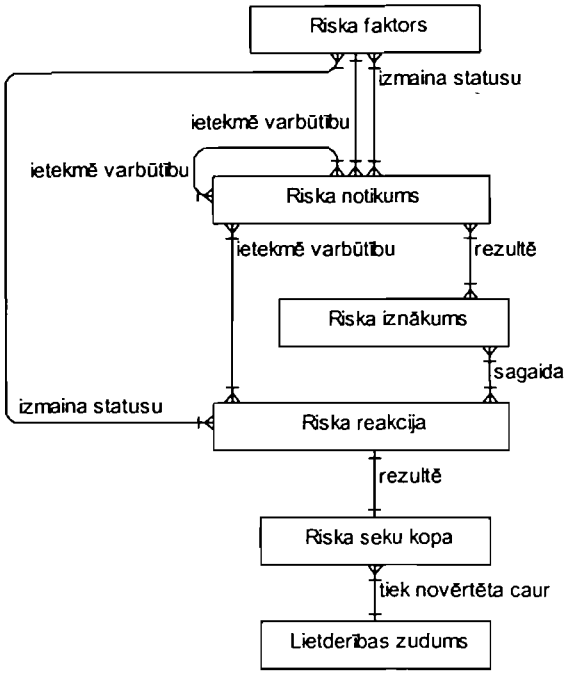
Plānošana, izmantojot CTC metodi, atšķiras no plānošanas tradicionālā veidā. Tradicionāli plānotājs vispirms izvirza mērķi, tad noskaidro esošo situāciju, lai paredzētu, kas notiks nākotnē. CTC pieeja ir: novērtējot esošo situāciju, paredzot iespējamos notikumus, izvērtēt, ko būtu lietderīgi darīt.

Reāli gan darbi, gan riski ir sarežģītāki nekā tos piedāvā uzlūkot šī metode, tādēļ metodes autori iesaka šo izmantot drīzāk kā vadlīnijas un domāšanas veidu, saprotot, ka riska identifikācija un darbu plānošana lielā mērā atkarīga no plānotāja zināšanām un pieredzes.

1.1.3.2. Riskit metode

Riskit metode risku definē kā zaudējuma varbūtību, zaudējumu vai jebkuras pazīmes, objekti vai darbības, kas asociējās ar šo zaudējumu [BAS98].

Varbūtība un zaudējums ir divi galvenie riska atribūti. Taču zaudējums ir atkarīgs no tā, kādas bijušas cerības, bet tas, savukārt, atkarīgs no tā, kas ir ar risku saistītās personas. Riska radītie zaudējumi var būt dažādi katrai ar to saistītajai personai. Piemēram, programmatūras izstrādātāji ir rēķinājušies ar to, ka nebūs pietiekošu resursu programmatūras testēšanai (risks). Tādēļ, visticamāk, arī produkta akcepttests būs neveiksmīgs. Izstrādātāji var savlaicīgi plānot aktivitātes šādas situācijas labošanai.



2. attēls. Riskit metodes komponentu formālā shēma

Aplūkosim 2. attēlā doto riska analīzes formālo shēmu.

Riska faktors izsauc riska notikumu. Riska faktoru piemēri ir: jaunu metožu vai rīku izmantošana programmatūras izstrādei, nepieredzējis personāls u.c.

Riska notikums atspoguļo negatīvo notikumu. Par riska notikumu nekad nevar skaidri zināt, vai tas notiks vai ne. Tādēļ riska notikumam piekārto varbūtību, ar kādu negatīvais notikums iestāsies. Riska notikumu piemēri ir: programmatūras nespēja strādāt, projekta vadošo darbinieku aiziešana, papildus laiks, kas pavadīts apgūstot jaunās metodes.

Riska iznākums raksturo situāciju projektā, kad ir noticis riska notikums, bet vēl nav veiktas nekādas darbības, lai vājinātu riska notikuma sekas. Šajā stāvoklī noteikti jādokumentē riska notikuma sekas, lai uzkrātu šo pieredzi turpmākajam darbam. Pamatojoties uz riska iznākuma aprakstu, bieži vien turpmāk var pieņemt efektīvākus lēmumus nekā pamatojoties uz pašu riska notikumu.

Riska notikuma sekas parasti netiek pieņemtas kā nenovēršams ļaunums. Organizācijas mēdz kaut ko darīt lietas labā, tādējādi vienam riska notikumam var tikt piekārtotas viena vai vairākas riska reakcijas. Ja riska notikumam ir tikai viena iespējamā reakcija, situācija ir determinēta, ja vairākas, - tad katra iespējamā riska

reakcija ir sava alternatīva. Riska reakcija ietekmē riska notikuma varbūtību. Ja ietekme ir varbūtiska, tad ietekme ir analoga kā riska faktoram uz riska notikumu, t.i. riska reakcija ietekmē riska notikuma varbūtību. Riska reakcijas piemēri ir: pēc pārtraukuma sistēma atkal strādā, pazaudētie dati ir atjaunoti no rezerves kopijām u.c.

Riska seku kopa parāda, kādu ietekmi uz projektu ir atstājis riska notikums. Respektīvi, riska seku kopa parāda, kāda situācija ir projektā, ņemot vērā attiecīgo izmantoto riska reakciju. Riska seku piemēri ir: divu mēnešu izpildes kalendārā laika nobīde, projekta izmaksu palielinājums par 10000 latu.

Riskit metodes risku vadības cikls ir analogs CMM ieteiktajam risku vadības ciklam, tomēr Riskit metodei ir vairākas priekšrocības.

1. Paredzēti speciāli soļi, lai aprakstītu riska vadības uzdevumu.
2. Specifiski soļi, lai aprakstītu projekta mērķus, un līdzekļi, lai šo aprakstu uzturētu.

1.1.3.3. Boehm metode

Boehm piedāvātajā risku pārvaldības metodē [BOE91] ir divi primāri soļi, kuriem katram seko vēl trīs soļi (skat. 3. attēls).

Riska identifikācijas rezultāts ir risku saraksts, kuri ir attiecināmi uz projektu. Metodes, kuras izmanto risku identifikācijai, ir dažādi kontroljautājumu saraksti, lēmumu pieņemšanas procesa izvērtēšana, salīdzinot ar iepriekšējo pieredzi un sadalot riskus mazākos komponentos.

Riska analīzes rezultātā ir novērtētas draudu iestāšanās varbūtības un iespējamie zaudējumi.

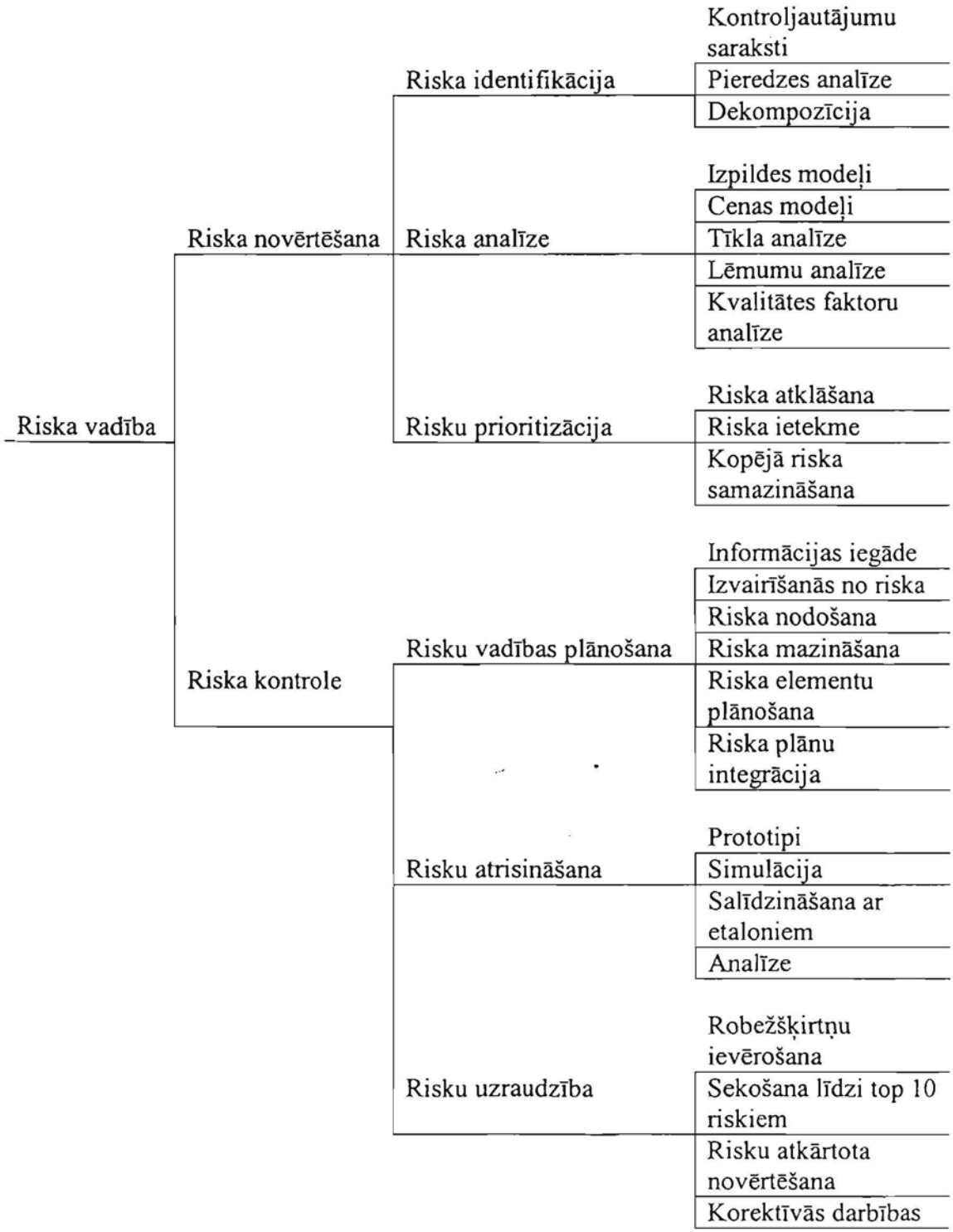
Risku prioritizācijas rezultātā riskus sakārto, ievērojot secību, kādā tos apkaros.

Risku pārvaldības plānošanas rezultāts ir plāns, kurā parādīts, kādas aktivitātes izmantos risku pārvaldībā, piemēram, papildus informācijas iegāde padziļinātai riska analīzei, riska "nodošana tālāk" (piemēram, apdrošināšana) utml.

Risku apkarošanas rezultātā atrod situāciju, kurā risks ir mazināts, vai nav vairs aktuāls. Šeit var izmantot prototipēšanu, simulāciju u.c. metodes.

Risku uzraudzības laikā seko līdzi riska izmaiņām attiecībā pret plāniem.

Boehm risku pārvaldības metodes cikls ir analogs CMM ieteiktajam, parādot praktiskas metodes, kuras izmantojamas attiecīgajā risku pārvaldības cikla posmā.



3. attēls. Boehm risku pārvaldības metodes shēma

1.1.3.4. Secinājumi par risku pārvaldības procesu

Standartu CMM un IEEE P1540 ieteikumi risku pārvaldības organizēšanai ir līdzīgi, būtisku atšķirību nav. Galvenie principi, kuri jāievēro risku pārvaldībā, saskaņā ar standartu ieteikumiem, ir uzskaitīti turpmāk.

1. Risku pārvaldības process ir nepārtraukts process visa izstrādes dzīves cikla laikā.
2. Galvenais dokuments risku pārvaldības procesā ir risku pārvaldības plāns, kurā identificēti riski, novērtēta to iestāšanās varbūtība un ietekme, piešķirtas prioritātes risku apkarošanai.
3. Plānu pārskata regulāri visa izstrādes dzīves cikla laikā, sekojot riskiem, apkarošanas aktivitāšu efektivitātei utt.

Risku pārvaldības metodes ievēro standartos (galvenokārt CMM) noteiktos principus, galveno uzmanību pievēršot riska identificēšanai. Riska identificēšana ir būtiskākais solis, konkrētie pasākumi riska apkarošanai ir risku pārvaldītāju ziņā.

Plānošana risku analīzes procesā atšķiras no plānošanas tradicionālā veidā. Tradicionāli plānotājs vispirms izvirza mērķi, tad noskaidro esošo situāciju un plāno, kā sasniegt mērķi. Taču risku pārvaldības procesā novērtē esošo situāciju un, paredzot iespējamos notikumus, izvērtē, kā jārīkojas.

Risku pārvaldības metožu izmantošanai nepieciešams šīs metodes adaptēt izmantošanai uzņēmumā. Organizācijām, kurām risku pārvaldība ir būtiska, ir individuāli adaptētas risku vadības metodes, piemēram, militārajām organizācijām [ROS98].

Praktiski izmantojot formālas risku pārvaldības metodes, tās uzskata par reālās dzīves pārlieku vienkāršošanu un cilvēki neuzticas pašu veiktās risku analīzes rezultātiem. Pēc dažādu autoru domām [BAS98] galvenie faktori, kāpēc netiek veikta sistemātiska risku vadība ir nosaukti turpmāk.

1. Risks ir pārāk abstrakta un izplūdusi lieta un trūkst rīki un metodes, lai riskus formāli definētu un detalizēti analizētu.
2. Risku analīze ir pārāk sarežģīta, jo katrai programmatūras izstrādē iesaistītajai pusei ir atšķirīgs skata punkts, tādēļ arī riski ir atšķirīgi. Šo aspektu īpaši ievēro Riskit metode (skat. "1.1.3.2 Riskit metode").

3. Risku novērtēšanas kvantitatīvās metodes prasa pārāk daudz darba. Piemēram, pārāk darbietilpīgi ir novērtēt kāda riska zaudējumu naudā. Novērtējums ir tik izplūdis, ka vērtētājs pats vairs netic savam rezultātam.
4. Risku identificēšana ir subjektīva un atkarīga no konkrētā vērtētāja spējas to ieraudzīt.

1.1.4. Programmatūras izstrādes procesa riski

Daudzos avotos ir minēti riski, kas negatīvi ietekmē programmatūras izstrādes projektus. Turpmāk uzskaitīti vairāki.

1. Nepietiekama laika, naudas un cilvēku resursu piesaiste projekta izstrādei [LOC96], [JON98], [HET93], [CMM99].
2. Jaunu tehnoloģiju izmantošana darbu izpildes gaitā [LOC96], [JON98], [HET93], [CMM99].
3. Darbu izpildē iesaistīto cilvēku nepietiekama sagatavotība darbu izpildei, nespēja skaidri definēt veicamos uzdevumus un nespēja pilnvērtīgi komunicēt [LOC96], [JON98], [HET93], [CMM99].
4. Nestabilas, nekvalitatīvi definētas programmatūras prasības [LOC96], [JON98], [HET93], [CMM99].

CMMI augstāk minētajiem pievieno vēl turpmāk nosauktos.

5. Programmatūras projektējuma slikta kvalitāte.
6. Programmatūras izstrādes pamatprocesa aktivitāšu pārklāšanās laikā.

Programmatūras izstrādes procesa pētnieks B.Boehm ir izveidojis programmatūras izstrādes procesa risku top 10 (skat. 1. tabula) [BOE91], kuru pārklāj arī jaunāks, [KEI02] pētījums.

1. tabula. B. Boehm programmatūras izstrādes risku top 10

Nr.	Riska nosaukums	Skaidrojums
1.	Izstrādātāju personāla trūkums	Kvalificēta personāla trūkums un tā maiņa
2.	Pārāk optimistisks projekta plānojums	Izstrādei nepieciešamais laiks un budžets nekorekti novērtēts (par zemu)
3.	Programmatūras funkcionalitāte ir nepareiza	Izstrādātas programmatūras funkcijas, kuras nav nepieciešamas, vai ir nepareizi specificētas
4.	Programmatūras interfeiss ir nepareizs	Neatbilstošs vai grūti uztverams lietotāja interfeiss
5.	"Apzeltīšana"	Tiek pievienoti dažādi "labumi", piemēram, attēli vai skaņas, jo tā gribējis lietotājs vai arī izstrādātājiem bijusi profesionāla interese
6.	Nepārtraukta izmaiņu straume	Nekontrolējamas un neparedzamas sistēmas funkciju izmaiņas
7.	Kļūdas "ārējos" komponentos	Izstrādē izmantoto gatavo komponentu zema kvalitāte
8.	Kļūdas "ārējos" procesos	Izstrādē izmantoto ārējo pakalpojumu zema kvalitāte
9.	Sistēmas lēna darbība	Izstrādātā sistēma strādā lēni
10.	Nepietiekami tehniskie resursi	Nespēja radīt programmatūru, jo ir nepietiekams tehniskais nodrošinājums vai risinājumi

1.1.4.1. Ekspertu aptauja par riskiem

Ievērojot to, ka risku pārvaldībā svarīgākā un sarežģītākā lieta ir risku identificēšana, un, lai noskaidrotu, kādi riski ir svarīgi programmatūras izstrādātājiem programmatūras izstrādes uzņēmumos Latvijā, tika veikta ekspertu aptauja.

1.1.4.1.1. Aptaujai izmantotā metode

Lai noteiktu riskus, kuri ir aktuāli programmatūras izstrādātājiem Latvijā, izmantota ekspertu aptaujas metode [CAP77]. Viena no izplatītākām metodēm ir ekspertu mijiedarbības procedūra - "Delfi" metode [NIC86]. "Delfi" metode sastāv no procedūru virknes, kas ļauj formēt ekspertu grupas viedokli par apspriešanai izvirzītajiem jautājumiem.

1.1.4.1.2. EKSPERTU GRUPAS FORMĒŠANA

Izvēlētajā ekspertu grupā jābūt 10-20 cilvēkiem un šai grupai jābūt formētai atbilstoši noteiktai [CAP77] procedūrai. Viens no nepieciešamajiem noteikumiem ir vienāda ekspertu kompetence. Lai parādītu metodes darbību, par ekspertu grupu izvēlēti triju programmatūras izstrādes uzņēmumu programmatūras izstrādes projektu pārvaldnieki, kuri atbilst turpmāk nosauktajiem kritērijiem.

1. Eksperts pašlaik strādā par projekta pārvaldnieku.
2. Eksperts strādā par projekta pārvaldnieku divus līdz astoņus gadus.
3. Eksperts ir vadījis vismaz divus programmatūras izstrādes projektus.

1.1.4.1.3. ANKETAS SASTĀDĪŠANA

Eksperti sākotnēji piedāvāja programmatūras izstrādes procesam raksturīgo risku sarakstu, kas atbilst viņa pieredzei. Pēc aptaujas pirmās kārtas sākotnējais rādītāju saraksta variants ir pārstrādāts, ievērojot ekspertu viedokļus. Ekspertiem piedāvātie riski doti 2. tabulā. Katram ekspertam sarakstā minētie riski bija jāsakārto biežuma dilšanas secībā un ietekmes dilšanas secībā.

2. tabula. Ekspertu sastādītais risku saraksts

Nr.	Risks
	Ar pasūtītāju saistītie riski
1.	Pasūtītāja nepietiekams tehniskais nodrošinājums
2.	Apgrūtināta sadarbība ar pasūtītāju
	Ar prasībām saistītie riski
3.	Nekvalitatīvi definētas programmatūras prasības
4.	Programmatūras prasību nestabilitāte
	Ar projekta vadību saistītie riski
5.	Neatbilstošs projekta plānojums (nepietiekams laika periods, nepietiekami resursi)
6.	Vāja projekta vadība
	Ar izstrādātājiem saistītie riski
7.	Izstrādes vides kļūdas
8.	Izstrādātāju motivācijas trūkums
9.	Izstrādātāju nepietiekams tehniskais nodrošinājums
10.	Izstrādātāju kadru mainība
11.	Izstrādātāju nepietiekamas zināšanas par izstrādes tehnoloģijām un vidi
12.	Apgrūtināta izstrādātāju sadarbība

1.1.4.1.4. REZULTĀTU APSTRĀDE

Tiek sastādīta ekspertu viedokļu matrica, kurā tiek norādītas eksperta dotās katra riska biežuma (skat. 3. tabula) un ietekmes (skat. 4. tabula) vērtība.

3. tabula. Risku biežuma vērtību matrica (12 – visbiežāk, 1 – visretāk)

Risks	1	2	3	4	5	6	7	8	9	10	11
Apgrūtināta izstrādātāju sadarbība	8	3	5	6	5	1	7	8	3	6	3
Apgrūtināta sadarbība ar pasūtītāju	6	12	10	10	10	9	10	11	2	6	11
Izstrādātāju kadru mainība	4	5	4	9	4	6	4	12	1	8	1
Izstrādātāju motivācijas trūkums	3	6	3	2	3	2	5	12	3	4	5
Izstrādātāju nepietiekamas zināšanas par izstrādes tehnoloģijām un vidi	9	8	9	4	7	3	8	8	5	9	7
Izstrādātāju nepietiekams tehniskais nodrošinājums	5	1	1	1	1	11	1	12	3	5	6
Izstrādes vides kļūdas	1	4	7	11	8	5	3	12	1	7	4
Neatbilstošs projekta plānojums (nepietiekams laika periods, nepietiekami resursi)	10	11	11	8	9	12	9	11	9	10	10
Nekvalitatīvi definētas programmatūras prasības	12	7	12	3	12	7	12	3	6	9	9
Pasūtītāja nepietiekams tehniskais nodrošinājums	7	10	2	7	2	4	2	12	7	5	2
Programmatūras prasību nestabilitāte	11	9	8	12	11	10	11	3	6	9	12
Vāja projekta vadība	2	2	6	5	6	8	6	3	6	10	8

4. tabula. Risku ietekmes vērtību matrica (12 – vissmagāk, 1 – visvieglāk)

Risks	1	2	3	4	5	6	7	8	9	10	11
Apgrūtināta izstrādātāju sadarbība	1	2	9	4	8	2	7	10	3	5	3
Apgrūtināta sadarbība ar pasūtītāju	10	12	12	9	10	12	10	3	1	4	7
Izstrādātāju kadru mainība	9	6	3	3	7	5	4	12	3	10	6
Izstrādātāju motivācijas trūkums	5	3	1	6	1	3	5	12	3	1	5
Izstrādātāju nepietiekamas zināšanas par izstrādes tehnoloģijām un vidi	4	5	7	8	6	9	8	8	3	7	8
Izstrādātāju nepietiekams tehniskais nodrošinājums	3	4	2	1	2	4	1	12	3	3	4
Izstrādes vides kļūdas	7	8	5	7	5	1	3	12	11	6	1
Neatbilstošs projekta plānojums (nepietiekams laika periods, nepietiekami resursi)	12	7	11	10	9	10	9	8	5	12	9
Nekvalitatīvi definētas programmatūras prasības	8	11	10	12	12	7	12	8	10	9	11
Pasūtītāja nepietiekams tehniskais nodrošinājums	2	1	8	2	3	8	2	11	5	2	2
Programmatūras prasību nestabilitāte	6	10	6	11	11	11	11	8	10	8	12
Vāja projekta vadība	11	9	4	5	4	6	6	3	7	11	10

Katra eksperta dotās risku biežuma un ietekmes vērtības apvieno, izmantojot risku biežuma un ietekmes matricu, kurā parādīti risku svarīguma kvadranti (skat. riska definīciju 5. tabulā). Proti, ja attiecīgais risks ir reti sastopams un tā ietekme ir viegla, tad attiecīgā riska svarīgums ir 1 u.t.t. Iegūtie risku svarīgumi doti 6. tabulā.

5. tabula. Risku svarīguma kvadranti

		Reti								Bieži			
	Eksperta viedoklis	1	2	3	4	5	6	7	8	9	10	11	12
Viegla ietekme	1	1	1	1	1	2	2	2	2	3	3	3	3
	2	1	1	1	1	2	2	2	2	3	3	3	3
	3	1	1	1	1	2	2	2	2	3	3	3	3
	4	1	1	1	1	2	2	2	2	3	3	3	3
	5	4	4	4	4	5	5	5	5	6	6	6	6
	6	4	4	4	4	5	5	5	5	6	6	6	6
	7	4	4	4	4	5	5	5	5	6	6	6	6
	8	4	4	4	4	5	5	5	5	6	6	6	6
	9	7	7	7	7	8	8	8	8	9	9	9	9
	10	7	7	7	7	8	8	8	8	9	9	9	9
	11	7	7	7	7	8	8	8	8	9	9	9	9
Smaga ietekme	12	7	7	7	7	8	8	8	8	9	9	9	9

6. tabula. Risku svarīgums

Risks	1	2	3	4	5	6	7	8	9	10	11
Apgrūtināta izstrādātāju sadarbība	2	1	8	2	5	1	5	8	1	5	1
Apgrūtināta sadarbība ar pasūtītāju	8	9	9	9	9	9	9	3	1	2	6
Izstrādātāju kadru mainība	7	5	1	3	4	5	1	9	1	8	4
Izstrādātāju motivācijas trūkums	4	2	1	4	1	1	5	9	1	1	5
Izstrādātāju nepietiekamas zināšanas par izstrādes tehnoloģijām un vidi	3	5	6	4	5	7	5	5	2	6	5
Izstrādātāju nepietiekams tehniskais nodrošinājums	2	1	1	1	1	3	1	9	1	2	2
Izstrādes vides kļūdas	4	4	5	6	5	2	1	9	7	5	1
Neatbilstošs projekta plānojums (nepietiekams laika periods, nepietiekami resursi)	9	6	9	8	9	9	9	6	6	9	9
Nekvalitatīvi definētas programmatūras prasības	6	8	9	7	9	5	9	4	8	9	9
Pasūtītāja nepietiekams tehniskais nodrošinājums	2	3	4	2	1	4	1	9	5	2	1
Programmatūras prasību nestabilitāte	6	9	5	9	9	9	9	4	8	6	9
Vāja projekta vadība	7	7	2	5	2	5	5	1	5	9	8

Katram rādītājam tiek skaitīts tā svarīgums G_j (skat. (F 1)) un vidējā svarīgumu vērtība $\overline{G}$ (skat. (F 2)).

$$G_j = \sum_{i=1}^{m} R_{ij} \text{ , kur} \qquad \text{(F 1)}$$

m - īpašību (rādītāju) skaits

$$\overline{G} = \frac{1}{n} \sum_{j=1}^{n} G_j \text{ , kur} \qquad \text{(F 2)}$$

n - ekspertu skaits

Katram radītājam aprēķina tā svarīguma novirzi no vidējā d_j (skat. (F 3)), noviržu kvadrātu d_j^2 un kvadrātu summu G (skat. (F 4)).

$$d_j = | G_j - \overline{G} | \qquad \text{(F 3)}$$

$$G = \sum_{j=1}^{n} d_j^2 \qquad \text{(F 4)}$$

Katram ekspertam aprēķina T_i (skat. (F 5)).

$$T_i = (t_i^3 - t_i) \text{, kur} \qquad \text{(F 5)}$$

t_i - maksimālais vienādu rangu skaits i-tam ekspertam.

Eksperimentam rēķina konkordācijas koeficientu K (skat. (F 6)), kurš parāda ekspertu viedokļu saskaņotību par apspriežamajiem jautājumiem.

$$K = \frac{12G}{m^2 (n^3 - n) - m \sum_{i=1}^{m} T_i} \text{, kur} \qquad \text{(F 6)}$$

n – īpašību skaits,

m – ekspertu skaits.

Konkordācijas koeficienta vērtība ir robežās no 0 līdz 1. Ja konkordācijas koeficients ir ievērojami mazāks par 1, tad ir nepieciešams veikt kārtējo aptaujas iterāciju. Pirms attiecīgas iterācijas ir jāaprunājas ar ekspertiem, lai noskaidrotu kāpēc novirzās ekspertu viedokļi. Jo koeficients ir tuvāk 1, jo saskaņotāki ir ekspertu viedokļi. Šajā eksperimentā veiktas divas aptaujas kārtas, un pēc otrās kārtas konkordācijas koeficients ir 0.72.

Turpmāk aplūkosim svarīgākos riskus, dažādu autoru un aptaujāto ekspertu piedāvātās preventīvās un korektīvās aktivitātes šo risku apkarošanai.

1.1.4.2. Nestabilas programmatūras prasības

Programmatūras prasību nestabilitāte kā programmatūras izstrādes procesa risks bieži minēts literatūrā (skat. nodaļu "1.1.4. Programmatūras izstrādes procesa riski"). Tomēr mērenas prasību izmaiņas projekta izstrādes gaitā ir normāla parādība programmatūras ražošanā. Plānojot programmatūras izstrādes darbietilpību, jāievēro, ka prasību izmaiņas ir no 1% līdz 3% mēnesī kopš prasību specificēšanas fāzes sākuma [JON98].

Lai apkarotu programmatūras prasību nestabilitāti, izmaiņu ienešana programmatūrā pēc sistēmanalīzes fāzes jāpadara finansiāli neizdevīga [JON98]. [BOE91] iesaka ieviest augstu izmaiņu slieksni, izvēloties programmatūras izstrādi sadalīt posmos, uzkrātās izmaiņas realizējot pakāpeniski katrā posmā. Šādu organizāciju iespējams panākt, izmantojot iteratīvo dzīves ciklu programmatūras izstrādei [BOE88], [HOL00]. Iteratīvo programmatūras dzīves ciklu kā iespējamo preventīvo aktivitāti min arī aptaujātie eksperti.

Otra ekspertu piedāvātā preventīvā aktivitāte, kas var būt arī korektīva, ir projekta izmaiņu vadības padomes nodibināšana. Projekta izmaiņu vadības padomē ietilpst gan pasūtītāja, gan izpildītāja pušu pārstāvji. Projekta izmaiņu vadības padome regulāri satiekas, apspriežot izmaiņu pieprasījumus. Tikai tās izmaiņas, kuras ir apstiprinātas Projekta izmaiņu vadības padomē tiek realizētas.

1.1.4.3. Neatbilstošs projekta plānojums

Neatbilstošs projekta plānojums, parasti nepietiekams izstrādei nepieciešamais laiks vai/un resursi, ir bieži minēts programmatūras izstrādes

procesu ietekmējošs risks (skat. nodaļu "1.1.4 Programmatūras izstrādes procesa riski").

[BOE91] kā preventīvas aktivitātes iesaka programmatūras izstrādei nepieciešamo resursu novērtēšanai izmantot dažādas novērtēšanas metodes vienlaicīgi – gan ekspertu novērtēšanu, gan formālās metodes. Formālās metodes ir, piemēram, Darbu strukturēšanas metode [BOE81], SLIM (Software Lifecycle Model) [SLI95], Estimacs [RUB83], COCOMO un COCOMO II [BOE81].

Aptaujātie eksperti piedāvā četras preventīvas aktivitātes.

1. Izmantot formālas metodes programmatūras izstrādei nepieciešamo resursu un laika prognozēšanai. Šo kā iespējamu preventīvo darbību iesaka arī [JON98].
2. Izmantot tehnoloģijas piedāvātās iespējas – dažādus automatizētus rīkus konfigurācijas pārvaldībai, projekta pārvaldībai u.c., tādējādi samazinot atbalsta procesiem nepieciešamos resursus un laiku.
3. Veidot programmatūras arhitektūru tā, lai iepriekš izveidoti komponenti būtu izmantojami atkārtoti.
4. Regulāri veikt rūpīgu programmatūras izstrādes plānu apskati.

Eksperti kā iespējamās korektīvās darbības piedāvā turpmāk nosauktās.

1. Pārskatīt projekta plānu kopā ar pasūtītāju, lai saliktu darbiem prioritātes vai pagarinātu izpildes termiņus.
2. Palielināt slodzi tiem projekta darbiniekiem, kuri spēj piedāvāt oriģinālus risinājumus. Šo kā iespējamu korektīvu darbību piedāvā arī [YOU97], piebilstot, ka šāds risinājums nevar būt ilglaicīgs.
3. Nomainīt projektā iesaistītos nepieredzējušos darbiniekus ar pieredzējušiem, kuri strādā līdzīgā problēmu apgabalā. Šo risinājumu eksperti piedāvā kā alternatīvu papildus darbinieku piesaistei, kas nedod gaidīto efektu [PRE91].

1.1.4.4. Apgrūtināta sadarbība ar pasūtītāju

Eksperti ieteica divējas preventīvas darbības.

1. Regulāras tikšanās projekta vadības līmenī. Vēlams, lai šīs tikšanās notiktu pasūtītāja telpās. Ja tas nav iespējams, tad pasūtītājs regulāri jāinformē par projekta attīstības gaitu telefoniski vai izmantojot e-pastu. Eksperti vienprātīgi atzina, ka šī ir visefektīvākā preventīvā darbība.

2. Vairāk kā puse ekspertu atzina, ka komunikācijas problēmas galvenais cēlonis ir pasūtītāja neziņa par programmatūras izstrādes tehnoloģisko procesu (dzīves ciklu). Šai gadījumā vienīgā iespēja ir pasūtītāja izglītošana šajā jomā. Šāds viedoklis ir sastopams arī literatūrā [CAR00], kur minēts ieteikums uz izstrādes procesu lūkoties kā uz abpusēju izstrādātāja-pasūtītāja mācību procesu.

Vienīgā ekspertu piedāvātā korektīvā darbība bija pasūtītāja puses kontaktpersonas maiņa, nozīmējot tādu, kurš labi pārzin uzņēmējdarbības vidi un ir ar pietiekamām pilnvarām pasūtītāja organizācijā.

1.1.4.5. Nekvalitatīvi definētas programmatūras prasības

Eksperti piedāvāja divas preventīvas darbības.

1. Netaupīt laiku programmatūras prasību izstrādei. Programmatūras prasību specificēšanai ir jābeidzas ar pasūtītāja un izstrādātāja savstarpēji parakstītu dokumentu – programmatūras prasību specifikāciju.
2. Veidot sistēmas prototipu.

Vienīgā ekspertu piedāvātā korektīvā darbība bija – mēģināt prasības noskaidrot neformāli, proti, atrodot jaunveidojamās sistēmas entuziastus gan pasūtītāja, gan izstrādātāja pusē.

1.1.4.6. Apgrūtināta izstrādātāju sadarbība

Ekspertu visbiežāk minētā preventīvā un reizē arī korektīvā darbība ir izstrādātāju grupas regulāras sanāksmes vismaz reizi nedēļā (bet ne biežāk kā divas reizes nedēļā), kuras jārīko pat tad, ja sākotnēji nav skaidrs, kas tiks apspriests. Šīs sanāksmes var būt gan formālas, gan neformālas.

Eksperti piedāvāja arī turpmāk nosauktās preventīvās darbības.

1. Veidojot izstrādātāju grupu, ņemt vērā izstrādātāju savstarpējās attiecības. Šajā gadījumā informācija par veiksmīgām/neveiksmīgām projektu grupām ir jāuzkrāj, un uzkrātā pieredze jāizmanto jaunu projekta grupu veidošanā.
2. Projekta grupu izvietošana "dzirdamības zonā". Jāizvairās no situācijas, kad kāds no projekta grupas darbiniekiem regulāri nav sameklējams.

1.1.4.7. Izstrādātāju kadru mainība

Eksperti piedāvā turpmāk nosauktās preventīvās darbības.

1. Jebkuras aktivitātes dublēšana, nepieļaujot situāciju, ka par kādu konfigurācijas vienumu zina tikai viens cilvēks.
2. Izstrādes gaitas un izstrādājamās programmatūras dokumentēšana pat tad, ja pasūtītājs to tieši neprasa.

Kā vienīgā korektīvā darbība tiek minēta izstrādātāju motivācija (gan finansiāla, gan sociāla).

1.1.4.8. Izstrādātāju motivācijas trūkums

Eksperti piedāvā turpmāk nosauktās preventīvās darbības.

1. Motivēt materiāli, mēģinot panākt atalgojuma un darba rezultātu tiešu proporcionalitāti.
2. Regulāri informēt projekta grupas dalībniekus par darba rezultātiem, mēģinot radīt apziņu, ka katrs ir tieši atbildīgs par sekmīga rezultāta radīšanu.

1.1.4.9. Izstrādes vides kļūdas

Eksperti piedāvā turpmāk nosauktās preventīvās darbības.

1. Jau iepriekš izpētīt informāciju par izvēlētās vides kļūdām, kas pieejama internetā.
2. Strukturēt izstrādājamo programmatūru tā, lai līdzīgas funkcijas būtu jāprogrammē tikai vienu reizi.
3. Neizvēlēties jaunākās izstrādes vides, kuras tikko parādās tirgū.
4. Organizēt uzņēmuma līmenī pieredzes apmaiņas forumus, kurā izstrādātājiem dalīties pieredzē.

1.1.4.10. Vāja projekta vadība

Eksperti piedāvā turpmāk nosauktās korektīvās darbības.

1. Projekta vadītāja maiņa. Tomēr šajā ieteikumā vērojama arī piesardzība – eksperti saka, ka šāda darbība 50% gadījumu var uzlabot situāciju, bet 50% - vēl vairāk pasliktināt.

2. Konstatējot konkrētas nepilnības projekta vadītāja darbā, piesaistīt pieredzējušāku projektu vadītāju, kurš konsultētu nepilnību novēršanai.

Šāda riska pārvaldība nav plaši aplūkota literatūrā, piemēram, *Boehm* atzīst, ka vāja projektu pārvaldība var ļoti spēcīgi ietekmēt izstrādes procesu, bet projekta plānošanas metodē COCOMO II, kura izstrādāta *Boehm* vadībā, šāds ietekmējošs faktors netiek ņemts vērā. *Boehm* viedoklis ir, ka vāja projekta vadība padara izstrādes procesu tik neprognozējamu, ka COCOMO II pieņem, ka projekts tiks labi vadīts, pretējā gadījumā nav jēgas šādu projektu plānot [BOE91].

1.1.5. Secinājumi

Risku vadība ir svarīga aktivitāte programmatūras izstrādes procesā. Lai arī nav daudz tādu uzņēmumu, kuros izmantotu formālas metodes risku vadībai [BAS98], neformāli šo procesu realizē. Piemēram, veicot regulāras sapulces vai apskates projekta izpildes gaitā.

Ekspertu aptaujā minētas gan preventīvas, gan korektīvas darbības risku mazināšanai, kas saistītas ar dažāda veida komunikāciju uzlabojumiem. Piemēram, regulāras izstrādātāju tikšanās, regulāra pasūtītāja informēšana par projekta izpildes gaitu. Tādējādi var secināt, ka dažāda veida komunikācija ir svarīga sekmīgai projektu izpildes gaitai.

Vairākas ekspertu minētās darbības risku mazināšanai ietver informācijas uzkrāšanu par izstrādes procesu un šīs informācijas izmantošanu dažādu risku apkarošanas plānošanai. Informācijas uzkrāšana ir vēl viens komunikācijas veids.

Šajā nodaļā aplūkotās risku pārvaldības metodes un standarti risku pārvaldību uzlūko kā atsevišķu procesu programmatūras izstrādē. Autore ir pārliecināta, ka risku pārvaldību nav jāuzlūko kā atsevišķu procesu, jo risku pārvaldība ir iedzimti integrēta ar citiem programmatūras izstrādes procesiem, piemēram, plānošanu, projektēšanu u.c. Tādēļ nepieciešams risku pārvaldību integrēt citos programmatūras izstrādes procesos, sevišķi, projekta pārvaldības procesā.

1.2. Informācijas uzkrāšana par programmatūras izstrādes procesu

Šīs nodaļas mērķis ir atrast metodes, kā uzkrāt informāciju par programmatūras izstrādes procesu un šo informāciju izmantot programmatūras izstrādes procesa uzlabošanai, aplūkot mērīšanas pamatjēdzienus, standartus un metodes mērīšanas procesa organizēšanai.

Programmatūras izstrādes procesa mērīšana ir gan projekta pārvaldības, gan kvalitātes nodrošināšanas mehānisms, kas darbojas programmatūras ražošanas uzņēmumā, veidojot uzņēmuma Mērījumu programmu. Mērījumu programma ir pasākumu komplekss uzņēmuma līmenī ar mērķi uzkrāt informāciju par izstrādātajiem projektiem, lai to izmantotu projektu vadībai, jaunu projektu resursu novērtēšanai un stratēģisku lēmumu pieņemšanai uzņēmuma vadības līmenī.

1.2.1. Mērījumu pamatjēdzieni

Programmatūras izstrādes procesa mērīšanā izmanto vairākus jēdzienus, kurus bieži vien lieto kā sinonīmus, taču starp tiem pastāv noteikta atšķirība.

Mērs parāda kāda izstrādes procesa atribūta kvantitatīvu apjomu, izmēru vai kapacitāti [PRE00]. Piemēram, programmrindiņu skaits ir programmatūras apjoma mērs.

Mērījums ir fakts, ka tiek noteikts kāda lielumā mērs [PRE00]. Piemēram, mērījums ir fakts, ka katram programmatūras izstrādes projektam tiek uzkrāts izstrādājamās programmatūras apjoms programmrindiņās.

Metrika ir kvantitatīvs mērs, kas parāda, cik ļoti kāds atribūts ir spēkā sistēmai vai tās komponentam [IEE93]. Piemēram, metrika ir iepriekš aprēķināta darba produktivitātes rādītāja salīdzināšana ar mērāmā izstrādes procesa darba produktivitāti.

1.2.2. Mēri

1.2.2.1. Programmatūras apjoma mēri

Izstrādājamās programmatūras apjoma mēri nosaka programmatūras funkcionalitātes apjomu. Šos mērus izmanto, lai salīdzinātu dažādu programmatūru funkcionalitātes apjumu un izdarītu secinājumus par izstrādes procesa efektivitāti, programmatūras kvalitāti u.c.

Turpmāk aplūkoti vairāki programmatūras apjoma mēri.

1.2.2.1.1. FUNKCIJPUNKTI

Funkcijpunkts ir programmsistēmas funkcionalitātes mērvienība. Funkcijpunktu skaits programmsistēmai nav atkarīgs no programmatūras realizācijas vides. Nosakot sistēmas funkcijpunktu skaitu, sistēmu aplūko no lietotāja viedokļa. Tehniskās detaļas un sarežģījumi, kas rodas tehnisku iemeslu dēļ realizācijas laikā, netiek ņemti vērā.

Funkcijpunktus kā programmatūras funkcionalitātes apjoma mēru sāka izmantot *J.Albrecht* 1971. gadā. Kopš tā laika ir zināmas un tiek izmantotas dažādas metodikas funkcijpunktu skaita iegūšanai [IFP98], [GAR96]. Šīs metodikas turpina attīstīties.

Funkcijpunktu skaita iegūšanai programmatūrai jāsaskaita turpmāk nosauktais (skat. 4. attēls).

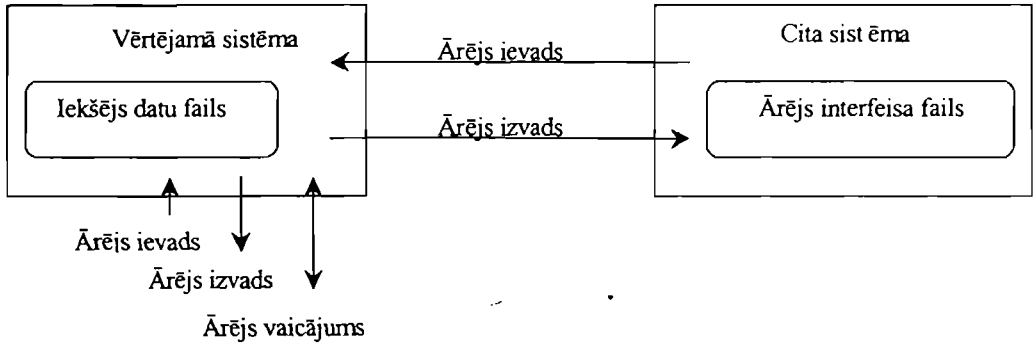
1. **Sistēmas iekšējos datu failus**. Sistēmas iekšējais datu fails ir savstarpēji loģiski saistītu datu kopa, kuras elementus rada, labo un iznīcina sistēmas iekšienē. Sistēmas iekšējais datu fails ir, piemēram, informācija par sistēmas lietotājiem (vārds, parole utml.).
2. **Vaicājumus**. Vaicājums ir ievada iniciēts sistēmas process, kura rezultātā atlasa datus no sistēmas iekšējiem datu failiem. Vaicājums neizmaina sistēmas iekšējos datu failus. Vaicājums ir, piemēram, visu to rēķinu atlase, kas izrakstīti nedēļas laikā.
3. **Ievadus**. Ievads ir sistēmas process, kas apstrādā datus vai vadības informāciju, kas ienāk sistēmā no ārpuses. Saņemtie dati modificē sistēmas

iekšējos datu failus. Ievads ir, piemēram, sistēmā apstrādājamo datu ievadforma.

4. **Izvadus**. Izvads ir sistēmas process, kas ģenerē datus vai vadības informāciju, kas iziet ārpus sistēmas robežām. Izvada piemērs ir vienas atskaites drukāšana.
5. **Ārējos interfeisu failus**. Ārējā interfeisa fails ir savstarpēji loģiski saistītu datu kopa vai vadības informācija, no kuras vērtējamā sistēma nolasa datus, bet tos modificē cita sistēma. Ārējais interfeisa fails ir, piemēram, citas sistēmas uzturēta datu bāze, kuras datiem piekļūst no vērtējamās sistēmas.

Vērtējot jāatceras, ka jāuzskaita tikai tie vaicājumi, ievadi vai izvadi, kuri saskatāmi lietotājam. Arī iekšējiem datu failiem un ārējo interfeisu failiem jāņem vērā, ka šīm datu kopām jābūt saskatāmām no lietotāja viedokļa. Katru lielumu novērtē kā sarežģītu, vidēju vai vienkāršu.

Aizpildot tabulu (skat. 7. tabula) un veicot tabulā norādītos aprēķinus, iegūst vērtējamās sistēmas nepieskaņotu funkcijpunktu skaitu.



4. attēls. Sistēmas datu un funkciju vērtējums

7. tabula Funkcijpunktu skaita aprēķināšana

	Sarežģīti	Vidēji	Vienkārši	Kopā
Ievadi	6 * skaits +	4 * skaits +	3 * skaits	=
Izvadi	7 * skaits +	5 * skaits +	4 * skaits	=
Iekšēji datu faili	15 * skaits +	10 * skaits +	7 * skaits	=
Ārēji interfeisa faili	10 * skaits +	7 * skaits +	5 * skaits	=
Vaicājumi	6 * skaits +	4 * skaits +	3 * skaits	=
			Nepieskaņotu funkcijpunktu skaits:	=

Informācijas avots funkcijpunktu skaitīšanai ir jebkuri sistēmas datus, interfeisus un funkcionalitāti aprakstoši dokumenti.

Funkcijpunkts kā programmsistēmas apjoma mērs izmantojams tikai informācijas sistēmām, kurām nav sarežģītu algoritmu aprēķinu veikšanai. Funkcijpunkti kā programmatūras apjoma mērs nav izmantojams funkciju bibliotēkām, kompilatoriem, reālā laika sistēmām u.c.

Būtiskākais trūkums funkcijpunktu izmantošanai programmatūras apjoma mērīšanai ir tas, ka funkcijpunktu skaita aprēķināšana prasa īpašas zināšanas un iemaņas. Ir vairākas metodikas, pēc kurām skaitīt funkcijpunktus IFPUG 4.0 [IFP98], Mark II [MAR98], Experience 2.0 [EXP02]. Pielikumā "I Pielikums. Funkcijpunktu skaita iegūšanas metodika" dota šī darba autores sastādīta funkcijpunktu skaita iegūšanas metodika, kas izstrādāta, pamatojoties uz [GAR96] un [IFP98]. Šo funkcijpunktu skaitīšanas metodiku sekmīgi izmanto vairākos programmatūras izstrādes uzņēmumos un bankās Latvijā informācijas sistēmu funkcionalitātes apjoma vērtēšanai.

Tomēr funkcijpunkti ir ļoti populārs funkcionalitātes apjoma mērs turpmāk nosaukto īpašību dēļ.

1. Funkcijpunktu skaits programmsistēmai raksturo funkcionalitātes apjomu no lietotāja viedokļa, tādēļ šo mēru ērti izmantot sarunās ar programmatūras pasūtītāju.
2. Funkcijpunktu skaits programmsistēmai nav atkarīgs no programmatūras realizācijas vides.

1.2.2.1.2. PROGRAMMRINDIŅAS

Programmrindiņas kā programmatūras apjoma mēru izmanto jau sen. Tā galvenā priekšrocība ir vienkāršība un iegūšanas relatīvs lētums. Proti, programmrindiņa ir intuitīvi saprotams mērs, ko iespējams iegūt izmantojot automatizētus rīkus.

Analogi kā funkcijpunktu iegūšanas gadījumā, ir vairākas metodikas, kā skaitīt programmrindiņas [COC2], [SLO02], [PAR92].

Taču daudzi praktiķi neatzīst programmrindiņu par funkcionalitātes mēru, jo nevar īsti nodefinēt, ko skaitīt un ko neskaitīt [HET93], kamēr citi to uzskata par

lietderīgāko apjoma mēru [HUM02]. Problēmas, kas saistītas ar programmrindiņām kā apjoma mēru, nosauktas turpmāk.

1. Atšķirīgās programmrindiņu skaitīšanas metodikās ir dažādas programmrindiņas definīcijas. Piemēram, ir metodikas, kas datu deklarācijas neuzskata par programmrindiņu, bet ir arī tādas, kas uzskata [HET93].
2. Grūti salīdzināt programmatūru, kas realizēta dažādās izstrādes vidēs.

Lai risināto problēmu par programmrindiņu skaitu dažādās vidēs, ieviests jēdziens – loģiskā programmrindiņa [PAR92]. Skaitot loģiskās programmrindiņas, ņem vērā katras programmrindiņas tipu, veidu, kā programmrindiņa radusies un programmrindiņas izcelsmi (skat. 8. tabula).

8. tabula. Programmrindiņu skaitīšanas metodikas ieteikumi [COC2]

	Iekļauts programmrindiņu skaitā
Programmrindiņas tips	
Izpildāma komanda	Jā
Neizpildāma komanda	Jā
Deklarācija	Nē
Kompilatora direktīva	Nē
Komentārs	Nē
Kā programmrindiņa radīta	
Programmēta	Jā
Ģenerēta ar koda ģeneratoru	Nē
Konvertēta ar automātisku konvertoru	Jā
Kopēta vai atkārtoti izmantota bez izmaiņām	Jā
Modificēta	Jā
Dzēsta	Nē
Programmrindiņas izcelsme	
Radīta no jauna, iepriekš nav bijusi	Jā
Pārņemta vai adaptēta no iepriekš rakstīta koda	Jā
No tās pašas programmatūras iepriekšējās versijas	Nē
No komerciāla produkta	Nē
No ārējas funkciju bibliotēkas (nemainīta)	Nē

Praksē izmanto saistības, kā no programmatūras apjoma funkcijpunktos iegūt programmatūras apjomu programmrindiņās [COC2] un otrādi [JON98], [EXP02]. Šāda konvertācija nepieciešama, piemēram, izmantojot dažādas

plānošanas metodes, kuras par ieejas parametru izmanto vai nu funkcijpunktus vai programmrindiņas.

No funkcijpunktu skaita iegūstot programmatūras programmrindiņu skaitu, izmanto vidējo programmrindiņu skaitu viena funkcijpunkta realizācijai izvēlētajā realizācijas vidē (F 7). Aprēķinot programmrindiņu skaitu no funkcijpunktiem, jāņem vērā, ka daļa programmas koda var būt neizmantojama, piemēram, neprecīzās specifikācijas dēļ (F 8) [COC2].

$$SLOC = FP \times LPFP, \text{ kur} \qquad \textbf{(F 7)}$$

SLOC – prognozējamais sistēmas programmrindiņu skaits,
FP – funkcijpunktu skaits,
LPFP – programmrindiņu skaits viena funkcijpunkta realizācijai realizācijas vidē, kas svārstās no 300 programmrindiņām viena funkcijpunkta realizācijai zema līmeņa valodās (Assembly) līdz 25 rindiņām ceturtās paaudzes valodās (Visual C++).

$$SLOC = SLOC \times (1 + \frac{BRAK}{100}) \text{ , kur} \qquad \textbf{(F 8)}$$

SLOC – prognozētais programmrindiņu skaits ((F 7),
BRAK – cik procentu programmprodukta koda būs neizmantojami neprecīzās specifikācijas dēļ.

Programmrindiņa kā programmatūras funkcionalitātes apjoma mērs ir praktiski izmantojams un populārs šādu iemeslu dēļ.

1. Programmrindiņa ir intuitīvi viegli saprotams jēdziens.
2. Programmrindiņu skaits ir viegli iegūstams, iespējams izmantot automatizētus rīkus.
3. Programmrindiņa kā mērs izmantojams visām programmsistēmām, neatkarīgi no tipa.

1.2.2.1.3. CITI PROGRAMMATŪRAS APJOMA MĒRI

1977. gadā *Halstead* ieviesa programmatūras apjoma Halsteda mēru – garumu (F 9) [HAL77]. Garums ir vienkārši iegūstams programmatūras apjoma mērs, kuru var izmantot analogi kā funkcijpunktus un programmrindiņas.

$N = N_1 + N_2$, kur (F 9)

N_1 – operatoru skaits programmā (+, -, IF, WHILE u.c.),
N_2 – operandu skaits programmatūrā.

Pazīstami vēl vairāki *Halstead* mēri, kuru praktiskā nozīme pašlaik ir zudusi [HET93]. Piemēram, vārdnīcas apjoms. Tomēr vairums programmatūras mērīšanas programmatūru šos mērus parāda tāpēc, ka tie ir viegli iegūstami [HET93].

Iezīmjpunkti (*feature points*) ir funkcijpunktu atvasinājums, kas aprēķina programmatūras funkcionalitātes apjomu, izmantojot iekšējos un ārējos datu failus, ievadus, izvadus un vaicājumus (analogi, kā funkcijpunkti), bet arī algoritmus. Tādējādi iezīmjpunkti izmantojami arī tādu programmatūru apjoma vērtēšanai, kurās realizēti sarežģīti algoritmi. Piemēram, reālā laika sistēmām, kompilatoriem utml.

McCabe ciklomātiskā sarežģītība ņem vērā programmatūras loģisko struktūru (F 10), ko neņem vērā programmrindiņu mērs un tikai daļēji ņem vērā funkcijpunkti. Šis mērs sākotnēji radies, pārveidojot programmatūras kodu par grafu un pielietojot grafu teorijas elementus, lai atrastu visus programmatūras loģiskos ceļus un izveidotu testus. McCabe ciklomātisko sarežģītību rēķina lielākā daļa programmatūras mērīšanas automatizēto rīku [HET93].

$C = C_1 - C_2 + 2$, kur (F 10)

C_1 – šķautņu skaits programmatūras grafā,
C_2 – loģisko operatoru skaits programmatūrā (IF, CASE).

McCabe ciklomātiskā sarežģītība kļuva populāra un nostabilizējās kā mērs pēc tam, kad tika pierādīta ciklomātiskās sarežģītības korelācija ar kļūdu blīvumu programmatūrā [WAR89].

3D funkcijpunkti ir analogi funkcijpunktiem, bet tie ņem vērā arī funkciju un kontroles dimensijas un ir īpaši piemēroti reālā laika sistēmu funkcionalitātes apjoma mērīšanai [WHI95]. Funkciju dimensiju ņem vērā tādējādi, ka skaita

soļus, cik nepieciešami, lai no ieejas datiem iegūtu izejas informāciju, bet kontroles dimensija – skaitot reālā laika sistēmas stāvokļus. Summāro 3D funkcijpunktu skaitu sistēmai iegūst pēc formulas (F 11).

$$IND = I + O + Q + F + E + T + R \text{ , kur} \qquad \textbf{(F 11)}$$

I – svarots ievadu skaits,
O – svarots izvadu skaits,
Q – svarots vaicājumu skaits,
F – svarots iekšējo datu failu skaits,
E – svarots ārējo interfeisu skaits,
T – svarots funkciju skaits,
R – svarots stāvokļu pāreju skaits.

$$sv = N_{il}\, W_{il} + N_{ia}\, W_{ia} + N_{ih}\, W_{ih} \text{ , kur} \qquad \textbf{(F 12)}$$

N_{il}, N_{ia}, N_{ih} – i-tā elementa skaits (I, O, Q u.c. skat. (F 11),
W_{il}, W_{ia}, W_{ih} – ir atbilstošie svari

1.2.2.2. Produktivitātes mēri

Produktivitātes mēri mēra izstrādes procesa produktivitāti, parādot, cik funkcijpunktus, programmrindiņas u.c. var izstrādāt vienā cilvēkstundā, cilvēkdienā, cilvēkmēnesī u.c. Piemēram, skat. PROD (F 13).

$$PROD = S / PM \text{ , kur} \qquad \textbf{(F 13)}$$

S – izstrādātās programmatūras apjoms funkcijpunktos,
PM – izstrādes darbietilpība.

Produkta piegādes līmenis skat. PDR (F 14). Produkta piegādes līmeni izmanto vairākas programmatūras izstrādes procesa darbietilpības un kalendārā laika prognozēšanas metodes [EXP02], [ISB01].

$$PDR = H / S \text{ , kur} \qquad \textbf{(F 14)}$$

S – izstrādātās programmatūras apjoms funkcijpunktos,
H – stundas projekta izstrādei.

Galvenā problēma, izmantojot produktivitātes mērus, ir tā, ka šo mērījumu vērtības ir grūti salīdzināt dažādiem projektiem, jo izstrādes procesa produktivitāti stipri ietekmē dažādi faktori. Piemēram, izstrādes procesa darbietilpības prognozēšanas metode ExperiencePro (skat. nodaļu "1.3.2.2 ExperiencePro") izmanto 21 produktivitātes faktoru, kas ietekmē izstrādes procesa produktivitāti. Tādēļ nevar vienkārši apgalvot, ka, ja viena izstrādes procesa darba produktivitāte bija 10 funkcijpunkti cilvēkdienā, bet otra – 3 funkcijpunkti cilvēkdienā, tad pirmais process bija produktīvāks par otru. Varbūt otrais process noritēja saspringtos termiņos, projekta izstrādātāju komandai tas bija pirmais projekts u.c., tad šāda produktivitāte ir ļoti laba pie attiecīgajiem izstrādes apstākļiem [FOR98]. Tāpat produktivitāte mainās, attīstoties izstrādes vidēm un tehnoloģijām. Produktivitāte vidēji uzlabojas par 10% gadā [EXP02].

Produktivitātes mēri daudz ko pasaka par izstrādes procesu, tomēr tos nevar analizēt atsevišķi, neanalizējot vidi, kurā izstrādes process notiek.

1.2.2.3. Kvalitātes mēri

Programmatūras kvalitātes mērījumu mērķis ir sekot līdzi izstrādājamā produkta kvalitātei visā tā dzīves cikla laikā, diagnosticējot iespējamās problēmas ar izstrādājamā produkta kvalitāti pēc iespējas agri. Lai arī ir iespējami daudz dažādu mēru, pamatā izmanto datus par kļūdām programmatūras kodā [PRE00].

Ļoti populārs mērs ir kļūdu novēršanas efektivitāte (F 15) [PRE00]. Izsakot šo mēru procentos, tam jābūt ~95% [RUB98], tad piegādāto programmatūras produktu var uzskatīt par kvalitatīvu.

$$DRE = E / (E + D), \text{ kur} \qquad \textbf{(F 15)}$$

E – pirms produkta piegādes atklāto kļūdu skaits,
D – pēc nodošanas atklāto kļūdu skaits

No pasūtītāja ienākušo problēmziņojumu skaits uz vienu programmatūras apjoma vienību ir būtisks izstrādājamās programmatūras kvalitātes rādītājs. Eiropas valstīs ražotajai programmatūrai no pasūtītāja ienākušo problēmziņojumu skaits uz vienu funkcijpunktu ir 0.15, bet ASV ražotajai 0.05 [RUB99].

1.2.2.4. Secinājumi par mēriem

Pieejami daudz dažādi programmatūras funkcionalitātes apjoma mēri. Katrs mērs ērtāk lietojams kāda noteikta tipa programmatūras apjoma vērtēšanai.

Mēru popularitāte ir tieši atkarīga no to saprotamības un iegūšanas ērtuma. Praksē populārākie programmatūras apjoma mēri ir programmrindiņas un funkcijpunkti.

Programmatūras apjoma mērus izmanto dažādas programmatūras izstrādes plānošanas un darbietilpības un izstrādei nepieciešamā laika prognozēšanas metodes. Šīm metodēm jābūt elastīgām un jāatbalsta dažādi mēri, citādi samazināsies to lietotāju loks.

Programmrindiņas ir intuitīvi saprotams un viegli iegūstams mērs. Programmrindiņu skaitīšanai var izmantot automatizētus rīkus. Tomēr programmrindiņas kā programmatūras apjoma mēru vairs nevar uzskatīt par objektīvu mēru, jo attīstoties izstrādes tehnoloģijām, rodoties integrētām, visu dzīves ciklu atbalstošām vidēm, programmrindiņa ir grūti definējama. Tādēļ programmrindiņa pamazām zaudē popularitāti kā programmatūras apjoma mērs.

Funkcijpunkti ir intuitīvi viegli saprotams mērs, kas objektīvi raksturo programmatūras apjomu no lietotāja viedokļa un nav atkarīgs no programmatūras izstrādes vides. Taču funkcijpunkti kā apjoma mērs ir izmantojams interaktīvām datu apstrādes sistēmām, bet nav izmantojams, piemēram, kompilatoriem.

Turpmāk, izvēloties programmatūras apjoma mērus, izvēlēsimies funkcijpunktus, kur tie ir pielietojami. Ja funkcijpunkti nav pielietojami, tad izmantosim programmrindiņas. Abi šie mēri ir savstarpēji apmaināmi (skat. nodaļu "1.2.2.1.1. Funkcijpunkti").

1.2.3. Standarti izstrādes procesa mērīšanai

Programmatūras izstrādes procesa mērīšana ir aplūkota programminženierijas standartos. Piemēram, ISO/IEC 15939:2001 un IEEE P1061 u.c. Šajā nodaļā mērīšanas procesa ilustrācijai detalizētāk aplūkots standarts ISO/IEC 15939:2001. Šis standarts izvēlēts tādēļ, ka tajā detalizēti parādīta mērīšanas procesa struktūra, akcentējot to, kas svarīgs veiksmīgas mērījumu programmas darbībai.

1.2.3.1. ISO/IEC 15939:2001

Programmatūras mērīšanas procesa mērķis ir savākt, analizēt un ziņot par mērījumu datiem, kas raksturo izstrādātā produkta kvalitāti un palīdz vadīt izstrādes procesu. Veiksmīga mērīšanas procesa rezultātā:

- organizācija uzticas mērījumu rezultātiem;
- identificē tehnisko un vadības procesu informācijas vajadzības;
- attīsta un uztur mērījumu kopu, kura atbalsta informācijas vajadzības;
- identificē un plāno mērījumu aktivitātes;
- nepieciešamie dati tiek savākti, saglabāti un interpretēti;
- informācijas produktus izmanto lēmumu pieņemšanai un kā objektīvus argumentus komunikācijai;
- mērīšanas procesu izvērtē un uzlabošanas priekšlikumus iesniedz procesa īpašniekam.

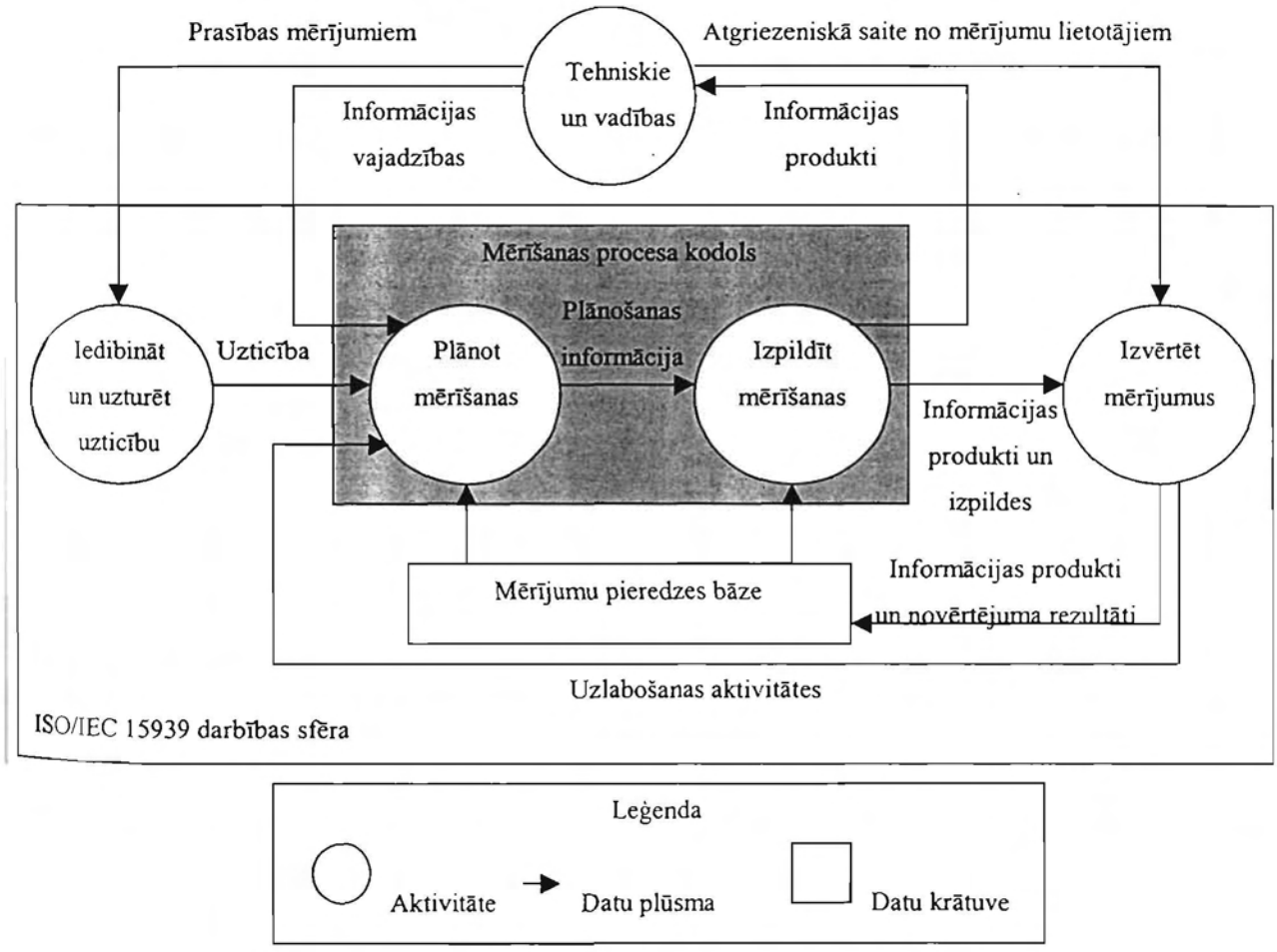
Standarta rekomendētajā mērījumu procesa modelī ir 5 aktivitātes (skat. 5. attēls).

Aktivitātes "Iedibināt un uzturēt uzticību mērījumiem" laikā nodefinē mērījumu darbības sfēru, izskaidro mērīšanas procesu un tā mērķus iesaistītajam personālam, panāk uzņēmuma vadības atbalstu. Tāpat ieplāno resursus mērījumu vākšanai un apstrādei.

Mērīšanas procesa plānošanas laikā identificē informācijas vajadzības un izvēlas atbilstošus mērus. Tad definē datu vākšanas, analīzes un kopsavilkumu veidošanas procedūras. Šīs aktivitātes ietvaros nodefinē mērījumu datu izvērtēšanas kritērijus, iegādājas nepieciešamos tehnoloģiskos risinājumus.

Mērīšanas procesa izpildes laikā datu vākšanas procedūras integrē programmatūras izstrādes procesā, vāc datus, tos analizē, veidojot informācijas produktus. Iegūtie rezultāti jādokumentē un jākomunicē ar mērījumu izmantotājiem.

Mērījumu izvērtēšanas laikā izvērtē gan informācijas produktus, gan arī mērīšanas procesu. Atziņas par to, kā mērīšanas procesu uzlabot, jānoglabā Mērījumu pieredzes bāzē.



5. attēls. Programmatūras izstrādes mērīšanas procesa modelis saskaņā ar ISO/IEC 15939

Šis starptautiskais standarts identificē, definē, izvēlas un pielieto mērījumus projektam un organizācijai. Tāpat šis standarts parāda terminus, kādi jālieto mērījumu kontekstā. Šis standarts neuzskaita, kādi mērījumi būtu jāizmanto programmatūras izstrādes projektu mērīšanai, bet gan identificē procesu, kā mērīšana notiek.

1.2.4. Mērījumu programmu ieviešanas principi

1.2.4.1. GQM metode

Populāra metode Mērījumu programmas ieviešanai ir GQM (*Goal Question Metric*) metode [SOL99]. Izmantojot šo metodi Mērījumu programmas ieviešanai Programmatūras izstrādes uzņēmumā, ir divi etapi.

1. Mērījumu programmas definīcija.
2. Mērījumu programmas izpilde.

Mērījumu programmas definīcijas laikā definē mērķus. Mērķim jābūt definētam izmērāmos terminos.

Piemēram, Mērījumu programmas mērķi var būt turpmāk nosauktie [GRA92], [SOL99].

1. Uzlabot projektu izpildes gaitas izsekojamību, spējot jebkurā brīdī noteikt projekta gatavības pakāpi.
2. Minimizēt projektu izstrādes resursus un kalendāro laiku par 20%.
3. Uzlabot izstrādājamās programmatūras kvalitāti, samazinot no pasūtītāja ienākušo problēmziņojumu skaitu par 10%.
4. Uzlabot programmatūras izstrādes darbietilpības prognožu precizitāti līdz 85% (prognozētā darbietilpība pret reāli patērēto procentos).

Pamatojoties uz mērķiem, izvirza jautājumus, piemēram, cik precīzi vērtējam programmatūras izstrādei nepieciešamos resursus pašlaik? Vai ir kādas aktivitātes programmatūras izstrādes procesā, kuras neatbilst industrijas standartiem utml.?

Nākamais solis Mērījumu programmas definīcijas laikā ir projekta izstrādes gaitā vācamo datu kopas un datu vākšanas procedūras definēšana, kas ietver turpmāk nosaukto.

1. Vācamo datu kopas definīcija.

Šeit nosaka, kas jāvāc un kādās mērvienībās. Piemēram, cik latu izmaksāts algās projekta realizācijas gaitā. Izvēloties vācamo datu kopu, jāievēro, ka datus vāc, lai sniegtu atbildes uz iepriekš izvirzītajiem jautājumiem izvirzīto mērķu sasniegšanai.

2. Datu vākšanas procedūra.

Šeit nosaka, kad mērījumi jāvāc un kam tas jādara. Piemēram, prognozētās projekta izmaksas jāpieraksta, plānojot projekta izpildes gaitu, kā arī veicot atkārtotu pārplānošanu. Reālās projekta izmaksas – programmatūru ieviešot. Datus pieraksta projekta pārvaldnieks.

3. Atgriezeniskās saites nodrošināšana.

Šeit nosaka, kādi datu kopsavilkumi jāsagatavo un cik bieži, lai nodrošinātu savākto datu efektīvu izmantošanu lēmumu pieņemšanai izvirzīto mērķu sasniegšanai.

Lai mērījumu programma būtu veiksmīga, svarīgi savāktos datus regulāri apkopot, interpretēt un apspriest. Gan izvēloties vācamo datu kopu, gan interpretējot datus un veidojot dažādus kopsavilkumus, svarīgi maksimāli ievērot katra projekta individuālās vajadzības, problēmas un riskus.

1.2.4.2. MQG metode

MQG (*Metric Question Goal*) metode [HET93] iesaka lūkoties uz programmatūras izstrādes procesa mērīšanu iteratīvi, augšupejošā virzienā. Saskaņā ar MQG metodi, mērījumi dabiski rodas programmatūras izstrādes procesā. Pētot mērījumus, rodas jautājumi par dažādām aktivitātēm procesa gaitā, tādējādi vairojot zināšanas par pašu procesu. Piemēram, kāpēc neatbilstoši maza darbietilpība veltīta testēšanai? No šīm zināšanām rodas spēja definēt izmērāmus mērķus un plānot darbības, lai šos mērķus sasniegtu, uzlabojot izstrādes procesu. Procesa uzlabošanas gaitā rodas atkal jauni mērījumu, kas izraisa jaunus jautājumus utt., veidojot iteratīvu procesu. MQG metodei ir vairāki soļi.

1. Vācamo datu kopas un datu vākšanas procedūras definīcija.

Šeit nosaka, kādi mērījumi jāvāc un kādās mērvienībās. Izvēloties vācamo datu kopu, jāievēro, ka datus vāc, lai atbalstītu izstrādes projektu ikdienas vajadzības.

2. Mērījumu datu vākšana.

Šeit notiek datu vākšanas aktivitātes saskaņā ar iepriekšējā soļa definīcijām.

3. Rezultātu prezentācija un izmantošana.

Šeit apkopo un analizē mērījumu datus, rezultātus apspriež ar projekta izstrādātājiem un tālāk izmanto projekta plānošanai.

1.2.4.3. Secinājumi par Mērījumu programmas ieviešanas metodēm

Veiksmīgi realizēta mērījumu programma nodrošina savlaicīgu ar programmatūras izstrādes procesu saistītu problēmu diagnosticēšanu. Tas, savukārt, ļauj savlaicīgi plānot papildus resursu piesaisti radušos problēmu risināšanai. Lai mērījumu programmu uzskatītu par veiksmīgu, tai:

- jāsasniedz vismaz viens no izvirzītajiem mērķiem;
- jāsasniedz vismaz 18 mēneši [GRA92].

Lai arī kādu metodi izvēlētos Mērījumu programmas ieviešanai, mērījumu dati vienalga būs jāvāc un jāpieraksta. Tas ir papildus darbs programmatūras izstrādātājiem, tādēļ ir sagaidāma pretestība no izpildītāju puses. Ļoti svarīgs ir cilvēciskais faktors, izstrādātāji jāpārliecina, ka savāktie dati netiks izmantoti, lai vērtētu izstrādātājus, bet gan tikai izstrādes procesu.

Galvenā MQG metodes priekšrocība attiecībā pret GQM metodi ir tā, ka vairāk tiek izmantoti tie mērījumu dati, kas dabiski rodas izstrādes procesa gaitā, tādējādi tiek prasīts mazāks papildus darba ieguldījums no projekta izstrādātāju puses.

Galvenā GQM metodes priekšrocība attiecībā pret MQG metodi ir tā, ka vieglāk panākt uzņēmuma vadības atbalstu, jo izvirzītie mērķi šai interešu grupai labāk saprotami. [HET93] parāda, ka vidējos un lielos programmatūras izstrādes uzņēmumos ir grūti vienoties par mērķiem.

Mērījumu programmas programmatūras izstrādes projektos veiksmīgi ieviestas daudzos uzņēmumos. Piemēram, Hewlett Packard [GRA92], Bell Atlantic Corp. [TOT98] u.c. Pasaules pieredze rāda, ka uzņēmumu lēmuma pieņemšana par mērījumu programmas ieviešanu nebūt nav viegls solis, turklāt iemesli ir atšķirīgi katram uzņēmumam.

1.2.5. Procesa mērīšana risku pārvaldībai

Ievērojot to, ka mērīšanas procesam ir jābūt nodefinētiem mērķiem (GQM metode, ISO/IEC 15939 standarts), ir izstrādāti ieteikumi ko un kā mērīt, lai atbalstītu kādu izstrādes tehnisko vai pārvaldības procesu. Piemēram, šajā nodaļā aprakstītā risku pārvaldības metode ilustrē to risku vadības metožu saimi, kuras mēģina formalizēt risku pārvaldību, piedāvājot uzkrāt un analizēt datus par

programmatūras izstrādes procesu. Risku pārvaldības procesa formalizācijas mērķis ir izveidot izstrādes procesa mērījumu kopu [HYA96], [SED01], un nodefinēt formālus risku sliekšņus, pamatojoties uz mērījumu rezultātiem [KAS00]. Šīs saimes metodes nepiedāvā vadlīnijas risku apkarošanai un analīzei, kam var izmantot standartos ieteiktās shēmas (skat. nodaļas "1.1.2.1. CMMI – SE/SW ieteiktā risku vadība" un "1.1.2.2. Standarta IEEE P1540 ieteikumi risku pārvaldības procesam").

Informāciju par izstrādes procesa mērījumiem izmanto, lai izvēlētos dažādus risku atribūtus un metrikas risku novērtēšanai prasību definēšanas, projektēšanas, implementēšanas un testēšanas fāzēs programmatūras izstrādes procesā [HYA96].

Risku vadība uzliek divas turpmāk nosauktās prasības mērījumu programmai.

1. Metrikai jāatbalsta riska kvantitatīvais novērtējums. Piemēram, jābūt nodefinētai skaitliskai vērtībai kļūdu skaitam un svarīgumam. Kad sasniegts attiecīgais skaits, jāpiešķir papildus resursi uzticamības riska apkarošanai.
2. Metrikām jāatbalsta vispusīga risku novērtēšana. Jāvar izveidot mērījumu kopsavilkumus, veidojot vienotu, projekta situāciju raksturojošu mērījumu saimi projektam. Šī prasība ir tradicionāla risku pārvaldības metodēm, tās balstās uz situācijas novērtējumu.

Izstrādes procesa mērīšanas mērķis ir novērtēt riskus katrā programmatūras izstrādes posmā un paredzēt, kādi riski sagaida projektu nākotnē, atbildot, piemēram uz jautājumu: "Kuras programmatūras daļas ir ar lielākiem uzticamības un uzturamības riskiem?".

Metode izmanto atšķirīgas metrikas katrā programmatūras izstrādes dzīves cikla fāzē. Piemēram, prasību definēšanas fāzes uzdevums ir nodefinēt nepretrunīgas, pilnas prasības un tās nostabilizēt pēc iespējas agrāk, kā arī nodrošināt prasību trasējamību visā izstrādes procesā. Galvenie riski, kas parādās šajā fāzē, ir iespēja, ka tiks radīta prasībām neatbilstoša programmatūra, tā netiks pabeigta laikā un prasības būs netestējamas. Metode piedāvā analizēt prasību definēšanas fāzi, mērot prasību definīcijas dokumentu. Atribūti šo risku analīzei doti 9. tabulā. Jo lielāka kāda no šo atribūtu vērtībām, jo lielāks ir kāds no identificētajiem projekta riskiem.

9. tabula. Prasību atribūti un metrikas

Atribūts	Metrika	Komentārs
Pārprotamība	Vāju frāžu skaits. Piemēram, "kur vien iespējams", "viegli", "normāli", "..., bet ne tikai" utml.	Izteicienam iespējamas vairākas interpretācijas
Pilnība	Nedefinēto prasību skaits. Piemēram, cik prasību definīcijās ir frāzes "tiks specificēts turpmāk" utml.	Cik vienumi palikuši turpmākai specificēšanai
Mainība	Mainīto prasību skaits / Kopējais prasību skaits. Prasību skaitu iegūst prasību dokumentā skaitot vārdus un frāzes pavēles izteiksmē un uzskaitījumus. Piemēram, "jānodrošina", "jāspēj", "ir prasīts, lai ...", "turpmāk uzskaitītie: ...", "konkrēti ..."	Pakāpe un laiks, cik bieži prasības ir mainītas visā dzīves cikla laikā
Trasējamība	Cik procenti prasību ir trasējamas virzienā uz augšu un uz leju.	Prasību trasējamība gan virzienā no testa un koda uz prasību specifikāciju, gan otrādi

Analogi rīkojas projektēšanas, implementēšanas un testēšanas fāzēs. Projektēšanas fāzē prasības pārveido par programmatūras projektējumu, kura kvalitāti jānovērtē pirms uzsāk implementēšanu, lai novērtētu projektējuma kvalitātes risku.

Implementēšanas fāzes galvenais mērķis ir radīt programmatūru, kas atbilst projekta prasībām. Implementēšanas fāzes galvenie riski ir, ka izstrādātā programmatūra nebūs uzturama un atkārtoti izmantojama. Viegli saprotams, ka tie moduļi, kuriem ir lielāka sarežģītība, ir grūtāk saprotami un uzturami un ir lielāka varbūtība, ka šajos moduļos būs kļūdas. Tādējādi sarežģītība tieši ietekmē programmatūras kvalitāti, uzturamību un atkārtotās pielietojamības iespēju.

Testēšanas fāzes mērķis ir atrast pēc iespējas vairāk kļūdas. Galvenie testēšanas fāzes riski ir nespēja iekļauties termiņā un programmatūras uzticamība.

Mērījumu izmantošana risku pārvaldībā palīdz sekot līdzi situācijai projektā, mērījumu vērtības var norādīt uz risku esamību. Faktiski mērījumus izmanto kā risku indikatorus programmatūras izstrādes projektos. Tomēr nedrīkst akli paļauties uz šādiem formāli identificētiem riskiem.

Lai arī šādu pieeju var uzskatīt par situācijas vienkāršošanu, tomēr šeit tiek izmantota mērījumu pamatideja – mērījumi ir vienīgais objektīvais informācijas avots par izstrādes procesu.

1.2.6. Mērījumu programmas piemērs

Šajā nodaļā parādīta Mērījumu programmas ieviešanas gaita programmatūras izstrādes uzņēmumā ALFA ar mērķi uzlabot informācijas apmaiņu starp projektā iesaistītajām pusēm un uzlabot izstrādes procesa darbietilpības prognozēšanas precizitāti. Piemēra mērķis ir parādīt informācijas plūsmu mērīšanas procesa ietvaros un atrast mērīšanas procesa veiksmes faktorus. Tāpat šeit parādītā Mērījumu programma praksē izmanto šajā nodaļā aplūkotos dažādos mērus un mērījumu vākšanas ideoloģijas.

1.2.6.1. Uzņēmuma ALFA apraksts

Programmatūras izstrādes uzņēmums ALFA ražo programmatūru pēc pasūtījuma. Ražošanas gaitā izstrādes process tiek plānots, tiek uzkrāta informācija par problēmziņojumiem, tiek testēts u.c. Izstrādes procesu var raksturot ar CMM 2. līmeni [CMM99]. Uzņēmuma ALFA klienti ir arī citi programmatūras izstrādes uzņēmumi, kuriem ir katram sava datu uzkrāšanas kārtība. Uzņēmumā ir 200 – 250 darbinieku, kas tieši nodarbojas ar programmatūras izstrādi. Uzņēmumā paralēli notiek ~30 programmatūras izstrādes projekti.

Uzņēmumam praksē ļoti bieži nepieciešams novērtēt programmatūras izstrādei nepieciešamos resursus, lai noslēgtu atbilstošu līgumu starp pasūtītāju un izpildītāju. Mēģinājumi to darīt pēc intuīcijas vai izvēloties "vidējo starp grīdu un griestiem" vairumā gadījumu beidzas ar pārlieku zemu novērtējumu (trīskārt par maz ir parasta lieta). Turklāt pasūtītāja un izpildītāja intuīcijas parasti jūt diezgan atšķirīgi, tādēļ intuitīvu spriedumu grūti izmantot kā argumentu sarunās ar pasūtītāju. Šeit parādās divējas problēmas saistībā ar programmatūras izstrādes procesu.

1. Neprecīza programmatūras izstrādes procesam nepieciešamo resursu novērtēšana.

2. Nepietiekama informācija par programmatūras izstrādes procesu tajā iesaistītajām pusēm.

1.2.6.2. Mērījumu programmas mērķi

Šādā situācijā izmantojām gan GQM, gan MQG metodes. Sākotnēji izvirzījām Mērījumu programmas mērķus, lai panāktu uzņēmuma vadības atbalstu. Galvenais Mērījumu programmas mērķis bija uzlabot projektu pārraudzību, tādējādi uzlabotos priekšstats par izstrādes procesu, un būtu pieejama objektīvāka informācija izstrādei nepieciešamo resursu novērtēšanai. Izvirzītais mērķis netika izteikts mērāmās kategorijās, jo esošajā situācijā nevar noteikt stāvokli uzņēmumā.

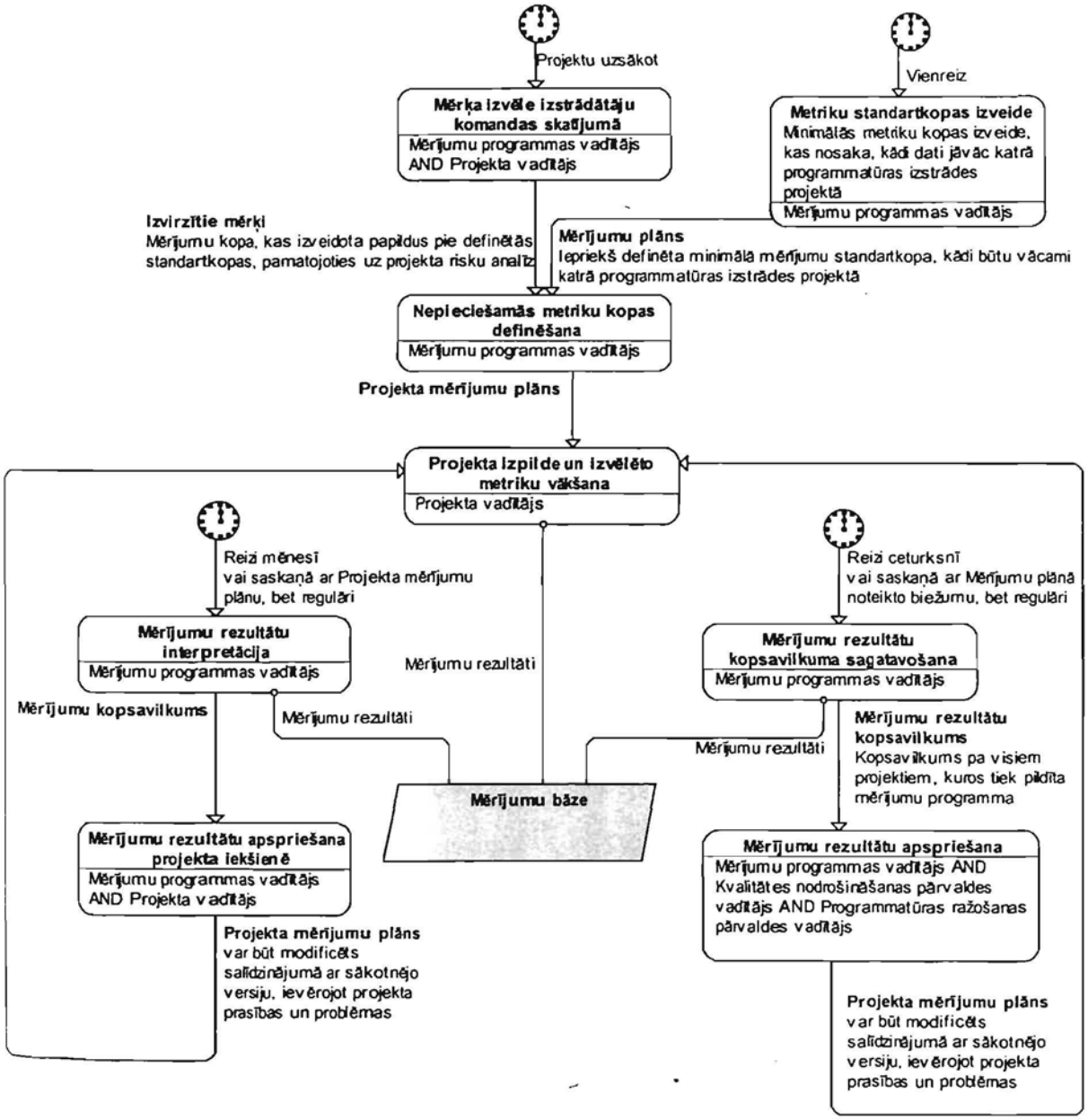
Lai Mērījumu programmas ieviešanu nebremzētu izstrādātāju un operatīvo vadītāju pretestība, maksimāli izmantojām tos datus par izstrādes procesu, kurus uzkrāj programmatūras izstrādes uzņēmumā [API00a], [API02b].

1.2.6.3. Mērījumu programmas plāns

Programmatūras izstrādes uzņēmumā ALFA Mērījumu programmu ieviesām kā pasākumu kompleksu uzņēmuma līmenī, paredzot vienu pilnas slodzes darbinieku Mērījumu programmas koordinēšanai.

Mērījumu programma ir projekts visa uzņēmuma līmenī, tādēļ īpaša uzmanība jāpievērš Mērījumu programmas vadītāja pakļautībai. Tai jābūt neatkarīgai no izstrādes procesa, bet komunikācijai jābūt regulārai [MAY00].

Mērījumu programmas ietvaros paredzēts veikt vairākas aktivitātes (skat. 6. attēls). Turpmāk katra aktivitāte aplūkota atsevišķā apakšnodaļā.



6. attēls. Uzņēmuma ALFA Mērījumu programmas principiālā shēma

1.2.6.3.1. METRIKU STANDARTKOPAS IZVEIDE

Metriku standartkopas izveidi veic vienu reizi Mērījumu programmas ieviešanas sākumā. Metriku kopas izvēles laikā ņem vērā Mērījumu programmai izvirzītos mērķus un projektos pieejamo informāciju. Metriku standartkopu izveido Mērījumu programmas vadītājs un darba rezultāts tiek iekļauts dokumentā "Mērījumu plāns", kurš tālāk tiks izmantots katra individuālā projekta mērījumu plāna sastādīšanai. Mērījumu plāna piemērs dots pielikumā (skat. "III Pielikums. Uzņēmuma ALFA Mērījumu programmas plāns"). Mērījumu plāns ir svarīga

veiksmīgas Mērījumu programmas sastāvdaļa, ja tajā skaidri tiek nodefinēta atbildība par dažādu metriku savākšanu un rezultātu iegūšanu [MAY00].

1.2.6.3.2. MĒRĶA IZVĒLE IZSTRĀDĀTĀJU KOMANDAS SKATĪJUMĀ

Papildus metriku standartkopai katra projekta izstrādes sākumā analizē, kādi faktori īpaši ietekmē izstrādes procesu. Piemēram, tiek veikta izstrādes procesa risku analīze.

Izstrādes procesa sākotnējo analīzi parasti veic projekta vadītājs kopā ar Mērījumu programmas vadītāju. Rezultātā papildus Mērījumu plānā nodefinētajām metrikām var tikt nodefinētas jaunas. Piemēram, ja izstrādājamais produkts ir sarežģīts, tad nodefinē sarežģītības mērus, lai varētu kontrolēt produkta sarežģītību.

Aktivitātes rezultāts ir "Projekta mērījumu plāns", kurā tiek nodefinēts precīzi projekta terminos, kurš ir atbildīgs par kuru metriku vākšanu, cik bieži tas notiek.

1.2.6.3.3. PROJEKTA IZPILDE UN IZVĒLĒTO METRIKU VĀKŠANA

Projekta izpildes laikā veic plānā paredzētos mērījumus. Sākoties katram programmatūras izstrādes projektam, projekta izstrādātājiem piedāvā mērījumu vākšanas formas (skat. "II Pielikums. Mērījumu datu savākšanas formas"). Taču gadījumos, kad uzņēmums ALFA projekta izstrādei strādā kā cita programmatūras ražošanas uzņēmuma apakšuzņēmējs, projektā mērījumus jāuzkrāj saskaņā ar virsuzņēmēja noteikumiem. Šādā situācijā Mērījumu programmai uzkrātie dati jāpārveido datu analīzei piemērotā formā, lai neradītu situāciju, ka mērījumi jāpieraksta divreiz.

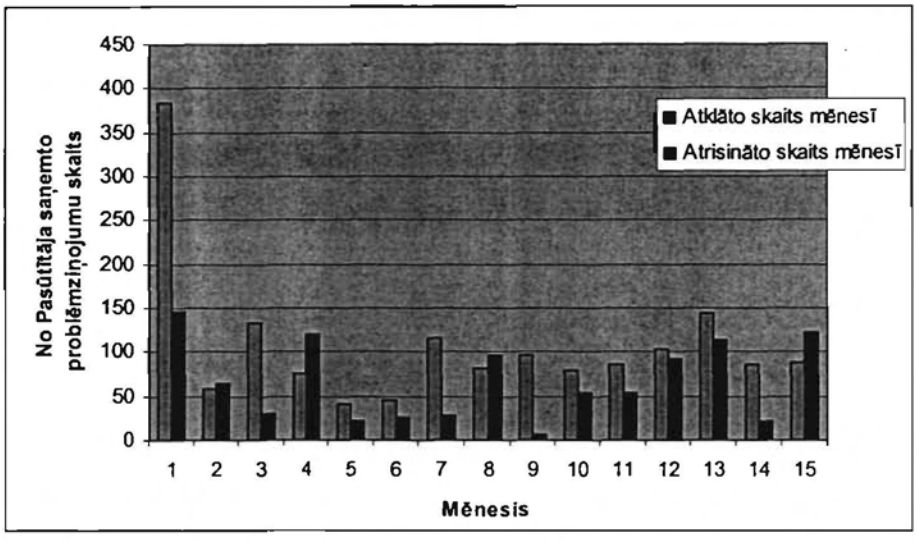
1.2.6.3.4. MĒRĪJUMU REZULTĀTU INTERPRETĀCIJA

Mērījumu rezultātu interpretācija ir būtisks posms Mērījumu programmas veiksmīgai realizācijai. Šim procesam jānotiek regulāri, parādot, ka savāktos mērījumus analizē un iegūto informāciju izmanto izstrādes procesam aktuālu problēmu risināšanai [API02], [API02b], [API02c], [HET93].

Interpretējot mērījumu rezultātus, jāņem vērā Mērījumu programmas mērķis – uzlabot projektu pārraudzību. Tādēļ mērījumu rezultātus interpretēsim tā, lai

objektīvi atspoguļotu situāciju programmatūras izstrādes projektā, tādējādi uzlabojot projektu pārraudzību.

Mērījumu rezultātu interpretācijas piemēri parādīti 7. attēlā un 8. attēlā.

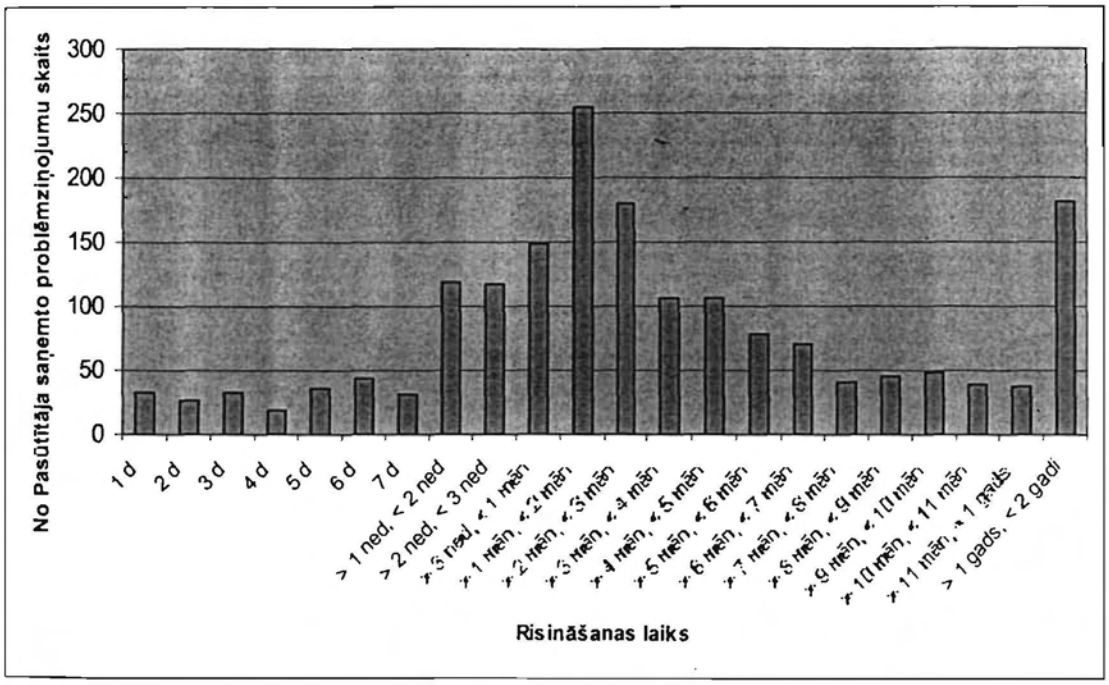


7. attēls. No Pasūtītāja saņemto un atrisināto problēmziņojumu skaita mērījumu interpretācijas piemērs

7. attēlā redzamā diagramma parāda krīzi programmatūras izstrādes projektā pirmajos divos mēnešos, kad reģistrēto problēmu skaits ievērojami pārsniedz atrisināto problēmu skaitu. Šāda situācija projektā prasa nekavējošu "ugunsgrēka dzēšanu", piemēram, palielinot izstrādātāju slodzi uz vienu mēnesi (skat. 4. mēnesi diagrammā). Tādējādi situācija uzlabojās turpmākajos mēnešos.

8. attēlā redzamā diagramma parāda, ka pārsvarā gadījumu no pasūtītāja ienākušos problēmziņojumus novērš mēneša līdz divu mēnešu laikā. Šī informācija palīdz plānot jaunu versiju piegādes termiņus.

Mērījumu rezultātu interpretācija ir radošs process, kuru nav lietderīgi uzticēt automatizētam rīkam, kurš ražo mērījumu "standartatskaites". Atskaišu parasti ir daudz, tādēļ tās netiek rūpīgi analizētas, līdz ar to pazūd Mērījumu programmas atgriezeniskā saite [API00].



8. attēls. Problēmziņojumu novēršanas laika mērījuma interpretācijas piemērs

1.2.6.3.5. MĒRĪJUMU REZULTĀTU APSPRIEŠANA PROJEKTA IEKŠIENĒ

Mērījumu rezultātu apspriešana projekta iekšienē ir ļoti svarīga aktivitāte, jo parāda mērījumu vācējiem, ka savāktie dati tiek izmantoti. Tāpat jāparāda, ka mērījumu rezultāti netiek interpretēti tā, lai tos izmantotu pašu vācēju vērtēšanai – tiek vērtēts un analizēts tikai izstrādes process.

1.2.6.3.6. MĒRĪJUMU REZULTĀTU KOPSAVILKUMA SAGATAVOŠANA

Mērījumu rezultātu kopsavilkumus gatavo ievērojot to, kas nepieciešams, lai sniegtu papildus informāciju par programmatūras izstrādes projektiem, kas nepieciešama uzņēmuma vadītājiem.

Šos kopsavilkumus veido Mērījumu programmas vadītājs un to sagatavošanas biežums ir retāks nekā projektu operatīvās plānošanas vajadzību apmierināšanai. Šie kopsavilkumi ietver informāciju par visiem programmatūras izstrādes projektiem.

1.2.6.3.7. MĒRĪJUMU REZULTĀTU APSPRIEŠANA

Izveidotie mērījumu rezultātu kopsavilkumi tiek regulāri apspriesti, analizējot, cik tālu ir tikts izvirzīto mērķu sasniegšanā.

Mērījumu rezultātu apspriešanā piedalās Mērījumu programmas vadītājs, Kvalitātes nodrošināšanas vadītājs un Programmatūras ražošanas pārvaldes vadītājs.

1.2.6.4. Iesaistītais personāls un informācijas plūsma

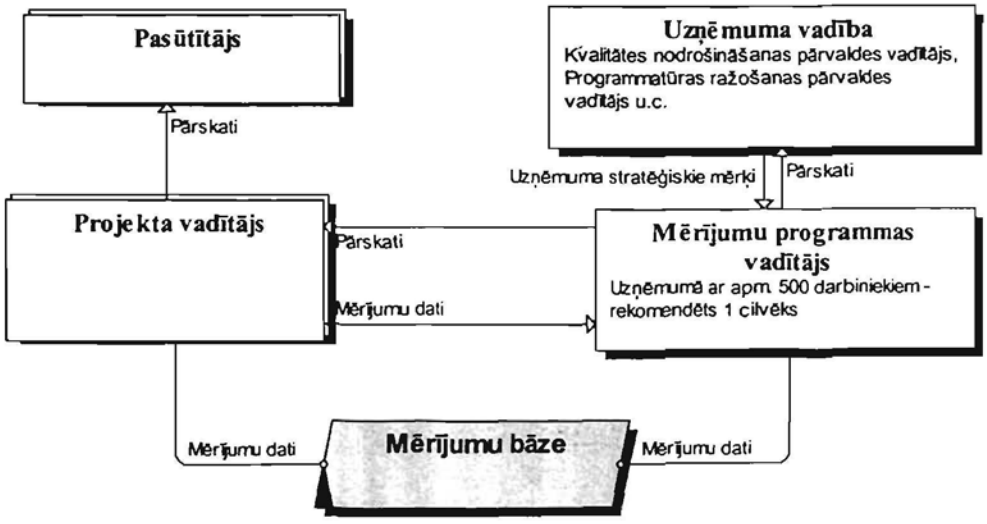
Mērījumu programmas realizācijā piedalās turpmāk nosauktie uzņēmuma darbinieki (skat. 9. attēls).

Projekta vadītājs. No projekta vadītāja tiek saņemti mērījumu dati. Projekta vadītājam iesniedz rezultātu kopsavilkumus, kas paredzēti izmantošanai projekta iekšienē. Tāpat projekta vadītājs pats sagatavo vai izmanto saņemtos kopsavilkumus, lai vajadzības gadījumā sniegtu informāciju projekta pasūtītājam.

Mērījumu programmas vadītājs. Darbinieks, kas koordinē mērījumu programmas ieviešanas gaitu, sagatavo pārskatus, konsultē datu interpretācijā.

Uzņēmuma vadība mērījumu programmas ietvaros ir kopsavilkumu saņēmējs.

Mērījumu programmas ieviešanas gaitā svarīga loma ir regulārai informācijas apmaiņai starp programmas realizācijā iesaistītajām pusēm.



9. attēls. Mērījumu programmas realizācijā iesaistītie darbinieki un informācijas apmaiņa

1.2.6.5. Mērījumu programmas rezultāts

Sākotnēji Mērījumu programmas ieviešanai tika izvirzīti divi mērķi (skat. "1.2.6.2. Mērījumu programmas mērķi").

1. Uzlabot neprecīzo programmatūras izstrādes procesam nepieciešamo resursu novērtēšanu.
2. Sniegt objektīvu informāciju par programmatūras izstrādes procesu.

Mērījumu programma uzņēmumā ALFA ir sasniegusi izvirzītos mērķus. Mērījumu programmas ietvaros nostabilizētā informācijas plūsma nodrošina visu izstrādes procesā iesaistīto pušu informāciju par izstrādes procesa mērījumiem, uzlabojot izstrādes procesa caurspīdīgumu.

Pirmais solis programmatūras izstrādes procesam nepieciešamo resursu novērtēšanas precizitātes uzlabošanai bija esošās situācijas novērtējums, vidējā prognozētās darbietilpības attiecība pret reālo bija tuvu 1. Tas nozīmē, ka vidēji darbietilpība tika prognozēta precīzi. Tomēr neapmierinātību radīja tie atsevišķie, nereti īpatnējie projekti, kuru darbietilpība tika novērtēta divkārt un vairāk par zemu. Mērījumu rezultātiem koncentrējoties pie Mērījumu programmas vadītāja un mērījumu bāzē, radās vienots informācijas avots darbietilpības novērtēšanai.

Mērījumu rezultātu ikmēneša apkopošana un apspriešana projekta iekšienē sniedza papildus informāciju par projekta gaitu un nodrošinot mērīšanas procesam tik nepieciešamo atgriezenisko saiti.

1.2.6.6. Secinājumi par Mērījumu programmu uzņēmumā ALFA

Būtiska Mērījumu programmas sastāvdaļa ir mērījumu plāns, kurā jābūt skaidri nodefinētai atbildībai par mērījumu vākšanu, kvalitātes kontroli, apkopošanu un analizēšanu.

Mērījumu programmas ieviešanā svarīga ir atgriezeniskā saite, parādot, ka mērījumi tiek izmantoti projekta izstrādes procesa uzlabošanai un projekta izstrādātājiem piedāvātās metrikas ir saprātīgas un noderīgas.

Uzņēmuma izvēlētā uzņēmējdarbības niša nosaka projektu, izstrādes tehnoloģiju un vides dažādību. Tādēļ arī uzkrātie mērījumi ir dažādās vidēs. Šī ir situācija, ar kuru jārēķinās, neuzskatot, ka tā ir mērīšanas procesā iesaistīto darbinieku nespēja pildīt mērījumu plānu.

Mērījumu analīze un rezultātu intepretācija ir radošs process, jāizvairās no automatizētas regulāras atskaišu kaudzes ražošanas, ja uzkrātos datus neizmanto izstrādes procesa uzlabošanai.

1.2.7. Secinājumi

Standarti un metodes programmatūras izstrādes procesa mērīšanai nedod konkrētus ieteikumus, kādus programmatūras izstrādes procesa vai produkta atribūtus vajadzētu mērīt. Izstrādes procesam nepieciešamie mērījumi ir atkarīgi no konkrētā projekta vajadzībām.

Standarti un metodes mērīšanas procesa pārvaldībai identificē un definē mērīšanas procesu un sniedz ieteikumus mērīšanas procesa organizēšanai. Būtisks mērīšanas procesa veiksmes faktors ir atgriezeniskā saite, parādot, ka savāktos mērījumus izmanto izstrādes procesa uzlabošanai.

Sekojot līdzi izstrādes procesa mērījumiem visā izstrādes procesa gaitā, izveidojas skaidrs priekšstats par izstrādes procesu. Tomēr mērījumu informāciju nevar pētīt un analizēt atrauti no informācijas par vidi, kurā realizācija notiek. Vidi, savukārt, raksturo risku analīzes rezultāti (identificētie riski, to novērtējums). Tādējādi, izmantojot risku analīzes un mērījumu rezultātus, kas apkopoti vienos un tajos pašos laika momentos, iespējams raksturot situāciju izstrādes procesā un situācijas attīstību laikā, lai to komunicētu visiem procesa dalībniekiem un izmantotu izstrādes procesa uzlabošanai. Šo principu izmanto formālas metodes programmatūras izstrādes procesa raksturlielumu (pārsvarā darbieitlpības un izstrādei nepieciešamā laika) prognozēšanai, mērījumu un risku analīzes rezultātus apkopojot projektam beidzoties.

1.3. Formālu metožu izmantošana izstrādes procesa prognozēšanai

Pagājušā gadsimta pēdējo divu dekāžu laikā ir radītas daudzas formālas metodes programmatūras izstrādes projektu darbietilpības un izstrādei nepieciešamā laika prognozēšanai. Daudzas metodes izmanto patentētus prognozēšanas algoritmus, tādēļ nav publiski pieejama informācija, kas ļautu spriest par šo metožu algoritmiem. Tomēr šajā nodaļā aplūkosim vairākas formālas programmatūras izstrādes darbietilpības un laika prognozēšanas metodes ar mērķi noskaidrot, kādus faktorus attiecīgās metodes izstrādātāji uzskata par izstrādes procesu ietekmējošiem un kādus principus katra formālā metode izmanto un vai ir pietiekami projekta novērtēšanai izmantot tikai šos faktorus.

Uzlūkosim programmatūras izstrādes procesa darbietilpības un izstrādes laika prognozēšanas metodes kā "melno kasti". Kā ieejas informāciju šīs metodes izmanto turpmāk nosauktos parametrus.

1. Izstrādājamās programmatūras apjoms. Programmatūras apjoms ir izmērāms, piemēram, programmrindiņās, funkcijpunktos, iezīmjpunktos u.c.
2. Izstrādes procesu ietekmējoši faktori – dažādi izstrādes procesu ietekmējoši apkārtējās vides, izstrādes platformas u.c. faktori, kas ietekmē izstrādes procesu un tādējādi arī palielina vai samazina izstrādes darbietilpību vai tai nepieciešamo laiku. Šie faktori tiek izvērtēti pēc attiecīgās formālās metodes noteiktās skalas izstrādes procesa sākumā, kā arī atkārtoti, veicot pārplānošanu izstrādes procesa gaitā. Ir tādi faktori, kurus nevar nomērīt objektīvi, plānošanai izmanto plānotāja(u) attiecīgā faktora subjektīvus vērtējumus. Piemēram, programmētāja spējas.

Formālo darbietilpības un izstrādes laika prognozēšanas metožu izmantošanas rezultāts ir izstrādes procesam nepieciešamā darbietilpība. Dažas metodes prognozē arī izstrādei nepieciešamo laiku. Tas, kādas izstrādes pamatprocesa aktivitātes ietver attiecīgā prognoze, ir atkarīgs no metodes.

Metodes iedalās divās lielās grupās: analītiskās un analoģiju bāzētās. Analoģiju bāzētās metodes programmatūras izstrādes darbietilpības prognozēšanai balstās uz negatīvu eksponenciālu līkni (Rayleigh-Norden līkni), kas aproksimē

darbietilpības sadalījumu programmatūras izstrādes procesā. Darbietilpības prognozēšanai kā ieejas parametrus izmanto izstrādājamās programmatūras apjomu, rezultātā iegūto darbietilpību pieskaņo izmantojot dažādus koeficientus, kurus izvēlas atkarībā no apstākļiem, kādos izstrādes process notiek.

Analoģiju bāzētās metodes balstās uz informāciju, kas uzkrāta par programmatūras izstrādes projektiem. Izstrādes procesa raksturlielumu prognozēšana faktiski reducējas uz līdzīgu projektu meklēšanu projektu datu bāzē.

1.3.1. Analītiskās metodes

1.3.1.1. SLIM

SLIM (*Software Lifecycle Model*) ir kvantitatīvs matemātisks programmatūras izstrādes procesa prognozēšanas modelis, kurš izstrādāts pagājušā gadsimta septiņdesmito gadu beigās [KEM87]. Šī metode izmanto negatīvu eksponenciālu līkni (Rayleigh-Norden līkni), kas aproksimē darbietilpības sadalījumu programmatūras izstrādes procesā. Patlaban metodi turpina attīstīt uzņēmums QSM (Quantitative Software Management), kurš izstrādājis rīku programmatūras izstrādes procesa plānošanai, kas balstīts uz šo metodi [QSM02].

(F 16), (F 17), (F 18) ir formulas darbietilpības aprēķinam pēc SLIM metodes. SLIM metode atšķiras no citām ar to, ka izstrādei nepieciešamais laiks jāzina pirms izrēķina izstrādei nepieciešamo darbietilpību. Šo īpatnību nevar uzskatīt par trūkumu, jo praksē bieži gadās, kad izstrādei atvēlētais laiks ir administratīvi noteikts.

$$PM = 12^5 B(\frac{SLOC}{P})^3 \frac{1}{SCED^4} \quad \text{(F 16)}$$

$$PM = 56.4B(\frac{SLOC}{P})^{\frac{9}{7}} \quad \text{(F 17)}$$

$$PM = 8.14B(\frac{SLOC}{P})^{\frac{3}{7}} \quad \text{(F 18)}$$

PM – programmatūras izstrādei nepieciešamā darbietilpība (cilvēkgadi);

P – produktivitātes koeficients (naturāls skaitlis robežās no 754 līdz 57314);

B – speciāls zināšanu faktors (naturāls skaitlis robežās no 1 (slikti) līdz 36 (labi));

SLOC – programmrindiņu skaits, kas raksturo programmatūras apjomu;

SCED – darbu veikšanai nepieciešamais laiks gados.

Formula (F 16) izstrādāta, izmantojot datus par 5000 programmatūras izstrādes projektiem. Formula rāda, ka pastāv sakarība starp programmatūras izstrādei atvēlēto laiku un darbietilpību. Proti, darbietilpība samazinās, palielinoties izstrādei atvēlētajam laikam.

Formula (F 17) aprēķina darbietilpību tiem projektiem, kuriem nepieciešams minimāls programmatūras izstrādes laiks. Šo formulu izmanto projektiem, kuru darbietilpība pārsniedz 20 personmēnešus, bet kuriem jāiekļaujas minimālajā iespējamajā laikā.

Formula (F 18) aprēķina darbietilpību tiem projektiem, kuru minimālais izpildes laiks pārsniedz 6 mēnešus.

Produktivitātes parametrs P jānosaka pirms izstrādes darbietilpības aprēķināšanas. *SLIM* metode piedāvā produktivitātes parametra vērtības robežās no 754 līdz 57314, kas grupējas ap noteiktiem diskrētiem skaitļiem – produktivitātes indeksiem [PUT92]. Ja *SLIM* metodes datu bāzē nav informācijas ne par vienu līdzīgu projektu, tad tiek izvēlētas produktivitātes nominālās vērtības, pamatojoties uz informāciju par izstrādes vidi, tehnisko nodrošinājumu, personāla pieredzi u.c. [BOE84].

Produktivitātes indekss ir augsta līmeņa indekss robežās no 1 (slikti) līdz 36 (labi), kurš mēra visus iepriekš minētos faktorus. Piemēram, ja produktivitātes indekss ir 1, tas nozīmē, ka projekts tiek izstrādāts ar primitīviem izstrādes līdzekļiem, izstrādātāju zināšanas ir vājas utt.

Patlaban pieejams rīks *SLIM Estimate*, kurš atbalsta SLIM modeli, tāpat turpinās darbs pie šīs metodes, to pielāgojot jaunākām izstrādes tehnoloģijām un vidēm.

Metode ir ļoti jutīga pret produktivitātes parametra P izmaiņām. Produktivitātes parametru ir sarežģīti noteikt projekta sākotnējās plānošanas stadijā, kad vēl maz ir zināms par plānojamo projektu.

Izmēģinot SLIM metodi astoņiem izvēlētiem pabeigtiem projektiem (skat. nodaļu "1.3.3. Aplūkoto modeļu "), iegūts, ka SLIM vidējā kļūda ir 288% (SLIM prognozētā darbietilpība pret reāli patērēto). Eksperiments apstiprinājis literatūrā minēto, ka SLIM metodes dotie rezultāti ir neprecīzi maziem programmatūras izstrādes projektiem [KEM93].

1.3.1.2. Jensena modelis

Jensena modelis ir programmatūras izstrādes darbietilpības un izstrādei nepieciešamā laika prognozēšanas modelis, kurš ievēro dažādu apkārtējās vides faktoru ietekmi uz izstrādes procesa darbietilpību un laiku [JEN84]. Jensena modelis, tāpat kā *SLIM*, ir analītisks modelis, kas balstās uz Rayleigh-Norden darbietilpības sadalījuma līkni. Jensena modelis piedāvā darbietilpību aprēķināt atkarībā no sistēmas izmēra un izstrādes tehnoloģijas (F 19).

$$E = 0.4(\frac{S}{C})^2 \frac{1}{t} \text{, kur} \qquad \textbf{(F 19)}$$

$$C = \frac{CTB}{\prod_{i=1}^{11} f_i} \qquad \textbf{(F 20)}$$

E – izstrādes darbietilpība cilvēkmēnešos;

S – programmatūras apjoms programmrindiņās;

C – izstrādes tehnoloģijas konstante;

t – izstrādei nepieciešamais laiks mēnešos;

CTB – izstrādes tehnoloģijas pamatkonstante (naturāls skaitlis no 2000 līdz 20000);

f_i – vides faktors (skat. 10. tabula). Jo attiecīgais vides faktors ir labvēlīgāks, jo lielāka vērtība jāizvēlas.

10. tabula. Izstrādes vides faktoru ietekme uz darbietilpību Jensena modelim

f_i	Parametrs	Vērtība
f_1	Sistēmanalītiķa spējas	no 1.73 līdz 2.06
f_2	Programmētāja spējas	no 1.75 līdz 2.03
f_3	Modernu metožu izmantošana	no 1.38 līdz 1.46
f_4	Izstrādē iesaistītas vairākas organizācijas	no 1 līdz 1.25
f_5	Izstrāde notiek vairākās vietās	no 1 līdz 1.25
f_6	Resursu un atbalsta pieejamība	no 1 līdz 1.96
f_7	Pieredze problēmapgabalā	no 1 līdz 1.57
f_8	Pieredze izstrādes platformā	no 1 līdz 1.21
f_9	Pieredze izstrādes vidē	no 1 līdz 1.53
f_{10}	Pieredze programmatūras izstrādē	no 1 līdz 1.87
f_{11}	Pieredze līdzīgu sistēmu izveidē	no 1 līdz 1.21

Izstrādes tehnoloģijas konstante CTB izsaka dažādus izstrādes vides faktorus, izstrādes stratēģijas, rīkus un metodes, personāla zināšanas izstrāde tehnoloģijās un problēmapgabalā. Konstantes vērtība ir robežās no 2000 līdz 20000, kur lielākas konstantes CTB vērtības raksturo efektīvāku izstrādes tehnoloģiju, labāk organizētu izstrādes procesu utt.

Patlaban pieejams uzlabots Jensena modelis, kuru izmanto rīkā *SAGE* [JEN95].

Jensena metodes doto rezultātu ietekmē tehnoloģijas konstante, kuru sarežģīti izvēlēties projekta sākotnējā plānošanas stadijā (analogi SLIM modelim). Tomēr šī metode ņem vērā arī 11 faktorus, kas ietekmē izstrādes procesu.

1.3.1.3. COCOMO II jaunas programmatūras izstrādei

COCOMO II metode kā programmatūras apjoma mēru izmanto programmrindiņas (skat. "1.2.2.1. Programmatūras apjoma mēri"). Projekta realizācijai nepieciešamo personmēnešu skaitu iegūst, izmantojot empīriskas sakarības. Šīs sakarības ieguvis COCOMO II metodes izstrādātājs B.Boehm [COC2], apstrādājot informāciju par dažādiem programmatūras izstrādes projektiem. Tajās viens personmēnesis ir 152 cilvēkstundas, kas atbilst vidējam darba ilgumam mēnesī ASV, ņemot vērā darbinieku atvaļinājumus, svētkus un brīvdienas, kā arī kavējumus slimības dēļ. Personmēnešu skaitu aprēķina pēc formulas (F 21)

$$PM = 2.5 \times (0.001 \times SLOC)^{1.01 + 0.01 \times \sum_{j=1}^{5} SF_j} \times \prod_{i=1}^{6} EM_i \qquad \text{(F 21)}$$

SLOC – programmrindiņu skaits;
SF_j - projekta mērogošanas koeficienti. SF_j vērtības ir konstantes, kas atkarīgas no tā, cik lielā mērā attiecīgais apgalvojums ir spēkā vērtējamai sistēmai.
Konstanšu vērtības dotas [COC2]:

SF_1 – izstrādātāju komandai ir iepriekšēja pieredze šādu uzdevumu risināšanā;
SF_2 – darbu ietekmē spēcīgi aparatūras, programmatūras, laika u.c. ierobežojumi;
SF_3 – visas iespējamās atkāpes no standartrisinājumiem ir paredzētas, skaidri definētas un iekļautas izstrādes plānos;
SF_4 – sistēmu izstrādā labi sastrādājies kolektīvs;
SF_5 – sistēmas izstrādes process ir labi organizēts;

EM_i – pieskaņošanas vērtības. EM_i vērtības ir konstantes, kas atkarīgas no tā, cik lielā mērā attiecīgais faktors ietekmē vērtējamās sistēmas izstrādes procesu.
Konstanšu vērtības dotas [COC2]:

EM_1 – sistēmas izstrādātāju spējas;
EM_2 – izstrādājamās sistēmas drošuma prasības, kas ietver prasības uz sistēmas drošumu un dokumentētību, kā arī datu bāzes izmēru;
EM_3 – sistēmas vai tās komponentu atkārtotās pielietojamības prasības;
EM_4 – produkta apjoma ierobežojumi un ātrdarbība;
EM_5 – programmrīku atbalsts sistēmas izstrādes procesā, kas ietver izstrādes vides stabilitāti, integrētu vidi programmprodukta izstrādei, vides atbalstu programmatūras izstrādei komandā;
EM_6 – sistēmas izstrādes kalendārais plānojums.

Projekta realizācijai nepieciešamo kalendāro mēnešu skaitu iegūst no iepriekš aprēķinātā personmēnešu skaita ((F 22).

$$CM = 3 \times PM^{0.33 + 0.002 \times \sum_{j=1}^{5} SF_j} \times \frac{EM_6}{100} \qquad \text{(F 22)}$$

CM – projekta realizācijai nepieciešamo kalendāro mēnešu skaits;
PM – projekta realizācijai nepieciešamo personmēnešu skaits;
SF_j – projekta mērogošanas koeficienti;
EM_6 – pieskaņošanas vērtība, kas nosaka sistēmas izstrādes kalendāro plānojumu.

COCOMO II metode ir ērti izmantojama un viegli saprotama, informācija par COCOMO II metodi ir atklāti pieejama.

1.3.1.4. COCOMO II programmatūras uzturēšanai

COCOMO II programmatūras uzturēšanai ir virsbūve COCOMO II modelim jaunas programmatūras izstrādei, kur par uzturēšanas darbu uzskata programmatūras funkcionalitātes izmaiņu, ja netiek likta klāt jauna funkcionalitāte.

COCOMO II programmatūras uzturēšanai kā ieejas parametru izmanto uzturamās programmatūras apjoma mēru - programmrindiņu skaitu. Izmantojot šo metodi, prognozē izmaināmā programmatūras koda apjomu programmrindiņās – koda pieaugumu (skat. (F 23)). Iegūto koda apjomu izmanto kā ieejas parametru COCOMO II modelim jaunas programmatūras izstrādei, prognozējot nepieciešamo darbietilpību un izmaiņu veikšanai nepieciešamo laiku (skat. nodaļu "1.3.1.3. COCOMO II jaunas programmatūras izstrādei").

$$MSLOC = \frac{SLOC}{100} * (KK_1 + AA + \frac{KK_2 + KK_3 + KK_4}{3} * KK_5) \quad \text{, ja AA} > 50 \quad \textbf{(F 23)}$$

$$MSLOC = \frac{SLOC}{100} * (KK_1 + AA * (1 + 0.02 * \frac{KK_2 + KK_3 + KK_4}{3} * KK_5)) \quad \text{, ja AA} < 50$$

MSLOC - programmrindiņu skaits koda pieaugumam, kurš radīsies uzturēšanas gaitā;

SLOC - programmrindiņu skaits koda gabalam, ar kuru būs jāstrādā uzdevuma izpildes gaitā;

$KK_{1,2,3,4,5}$ - labojamā koda kvalitātes rādītāji (skat. 11);

AA - formulas (F 24) rezultāts – izmaiņu apjoms procentos.

$$AA = 0{,}4 * AA_1 + 0{,}3 * (AA_2 + AA_3) \quad \textbf{(F 24)}$$

AA_1 - nepieciešamās izmaiņas projektējumā (procentos);

AA_2 - nepieciešamās izmaiņas kodā (procentos);

AA_3 - izmaiņas, kas nepieciešamas, lai izmainīto koda gabalu integrētu ekspluatācijā esošajā programmatūrā (procentos).

11. tabula. Uzturamās programmatūras koda kvalitātes rādītāji saskaņā ar COCOMO II metodi programmatūras uzturēšanai

Nosaukums	Apzīmējums	Vērtības					
Uzturamā koda kvalitāte	KK_1	8 - Nekomentēts un nedokumentēts	6 - Vāji komentēts un dokumentēts	4 - Apmierinoši komentēts un dokumentēts	2 - Labi komentēts un dokumentēts	0 - Strādājošs, testēts un dokumentēts kods	
Programmatūras struktūra	KK_2	50 - Nestrukturēts, "spagetti" kods	40 - Vāji strukturēts	30 - Visumā labi strukturēts, ir dažas vājas vietas	20 - Labi strukturēts	10 – Teicami strukturēts un skaidra modularitāte	
Funkcionalitātes skaidrība	KK_3	50 - Funkcionalitātes un programmatūras struktūras nesaskan	40 - Neliela korelācija starp funkcionalitāti un programmatūru	30 - Vidēja korelācija starp funkcionalitāti un programmatūru	20 - Laba korelācija starp funkcionalitāti un programmatūru	10 - Pilnīga skaidrība par funkcionalitātes saistību ar programmatūru	
Programmatūras pašaprakstošās īpašības	KK_4	50 - Nesaprotams, nedokumentēts kods	40 - Daži komentāri kodā, funkciju galvas, mazliet dokumentācijas	30 - Vidēji labi komentāri kodā, funkciju galvas, dokumentācija	20 - Labi komentāri kodā, funkciju galvas, dokumentācija, ir dažas vājas vietas	10 - Pašaprakstošs, uzturēts, labi organizēts kods	
Koda pazīstamība	KK_5	1 - Pilnīgi svešs	0.8 - Pārsvarā nepazīstams	0.6 - Daļēji nepazīstams	0.4 - Daļēji pazīstams	0.2 - Pārsvarā pazīstams	0 - Labi pazīstams

1.3.1.5. Citas metodes

PRICE-S ir empīrisks, parametrisks, atklāts programmatūras izstrādes procesa darbietilpības un izstrādei nepieciešamā laika novērtēšanas modelis, kas izstrādāts *Price Systems* [PAR88], [PAR95]. Sākotnēji radīts militārajām vajadzībām, tagad tas ir pieejams arī komerciālas programmatūras izstrādes darbietilpības novērtēšanai. *PRICE-S* izmantošanā ir vairāki secīgi soļi. Vispirms aprēķina sistēmas funkcionalitātes apjomu īpašās, šim modelim raksturīgās, vienībās, kuru aprēķina principi ir līdzīgi kā funkcijpunktiem. Pēc tam novērtē dažādiem sistēmas izstrādes etapiem (projektēšanai, programmēšanai u.c.) nepieciešamo vidējo laiku. Nākamie soļi saistīti ar izstrādei nepieciešamā vidējā laika pieskaņošanu, ievērojot dažādus faktorus. Piemēram, izstrādājamā produkta sarežģītību, izstrādātāju pieredzi u.c.

SoftCost ir matemātisks izstrādei nepieciešamo resursu prognozēšanas modelis, kas radīts NASA vajadzībām [TAU81]. Tas izmanto citu modeļu, piemēram, *PRICE-S*, *SLIM*, labās idejas. *SoftCost* izmanto 68 parametrus, kas saistīti ar darba produktivitāti, izstrādes laiku un dokumentēšanu. *SoftCost* izmanto arī izstrādes procesa darbu strukturizācijas ideju, katram strukturētajam darbam nepieciešamo darbietilpību pieskaņojot, izmantojot dažādus modeļa parametrus. Modeļa izmantošanas rezultātā iegūst programmatūras izstrādei nepieciešamo darbietilpību, laiku, prognozējamo izstrādātāju produktivitāti un dokumentācijas apjomu.

1.3.2. Analoģiju bāzētas metodes

Analoģiju bāzētās metodes balstās uz iepriekšējās pieredzes analīzi. Tiek uzkrāta informācija par programmatūras izstrādes projektiem. Darbietilpības un izstrādei nepieciešamā laika prognozēšana projektam faktiski notiek, meklējot analogus projektus projektu datu bāzē. Šo metožu izmantošanas veiksmīgums atkarīgs no tā, cik veiksmīgi atrasts meklēšanas kritērijs [ISB00].

Analoģiju bāzēto metožu galvenie trūkumi ir turpmāk nosauktie.

1. Rezultāts ir balstīts uz pieredzi, kas vairs nav derīga izstrādes tehnoloģiju attīstības dēļ.

2. Grūti izvēlēties līdzīgo projektu meklēšanas kritērijus, dažādi kritēriji var dot dažādus rezultātus.
3. Lai iegūtu labus rezultātus, uzkrātajiem datiem jābūt no tā paša uzņēmuma, lai būtu saistīti ar organizācijas kultūru.

Tomēr analoģiju bāzētās metožu dotie rezultāti ir objektīvi un viegli pamatojami, izmantojot citu projektu datus. [ISB00] veiktajā pētījumā secināts, ka, ja analoģiju bāzētās metodēs izmanto paša uzņēmuma uzkrātos datus, iegūtās darbietilpības prognozes ir precīzākas nekā izmantojot analītiskās metodes vai analoģiju bāzētās metodes, izmantojot citu uzņēmumu uzkrātos datus.

1.3.2.1. Checkpoint

Chekpoint ir analoģiju bāzēta programmatūras izstrādes darbietilpības novērtēšanas metode. To atbalsta rīks, kuru izstrādājusi SPR (*Software Productivity Research*) [JON96]. Šī metode izmanto aptaujas anketu, kurā ietverti jautājumi par programmatūras izstrādes vidi, tehnoloģijām, izstrādātāju zināšanām u.c. Katram jautājumam ir vairākas iespējamās atbildes, kur katrai atbildei ir svars no 1 līdz 5. Atbilde ar svaru 3 raksturo vidējo stāvokli ASV programmatūras ražošanas industrijā. Ja atbildes svars ir mazāks par 3, tad situācija ir labāka par vidējo. Gan jautājumi, gan atbilžu svari tiek pārskatīti katru gadu un ir SPR īpašums. *Checkpoint* metode balstīta uz informāciju par aptuveni 8000 programmatūras izstrādes projektiem.

Checkpoint metode izmanto funkcijpunktus kā programmsistēmas apjoma mēru.

Metodes autors iesaka izmantot makro un mikro novērtēšanu [JON98]. Makro novērtēšanai visus izstrādes procesu ietekmējošos faktorus ietver vienā pieskaņošanas vērtībā. Piemēram, 12. tabulā parādīts piemērs, kā saskaņā ar *Checkpoint* metodi, darba produktivitāti ietekmē izstrādātāju komanda un izstrādes rīki.

12. tabula. *Checkpoint* metodes makro novērtēšanai izmantojamie koeficientu piemērs

Ietekmējošo faktoru kombinācija	Reizinātājs produktivitātei
Izcila komanda izmanto izcilus rīkus	2
Izcila komanda izmanto vidējus rīkus	1.75
Vidēja komanda izmanto izcilus rīkus	1.5
Izcila komanda izmanto primitīvus rīkus	1.25
Vidēja komanda izmanto vidējus rīkus	1
Vidēja komanda izmanto primitīvus rīkus	0.9
Iesācēju komanda izmanto izcilus rīkus	0.75
Iesācēju komanda izmanto vidējus rīkus	0.65
Iesācēju komanda izmanto primitīvus rīkus	0.5

Piemēram, ja darba produktivitāte vidējai komandai izmantojot vidējus rīkus ir 12 funkcijpunkti personmēnesī, tad iesācēju komandai, izmantojot primitīvus rīkus, darba produktivitāte būs tikai 6 funkcijpunkti personmēnesī.

Mikro novērtēšanas gadījumā katru produktivitātes faktoru aplūko atsevišķi, tādējādi iegūstot precīzāku novērtējumu. Piemēram, izmantojot 13. tabulā dotos produktivitātes reizinātājus, iegūstam, ka iesācēju komanda (izstrādātāju komandas spējas ir vājas), izmantojot primitīvus rīkus (izmantoto rīku iespējas ir vājas), spēj saražot 12 * 0.5 * 0.8 = 4.8 funkcijpunktus cilvēkmēnesī.

13. tabula. *Checkpoint* metodes mikro novērtēšanai izmantojamo koeficientu piemērs

Novērtējums	Izstrādātāju komandas spējas	Izmantoto rīku iespējas
Izcilas	1.5	1.2
Labas	1.25	1.1
Vidējas	1	1
Zem vidējā	0.75	0.9
Vājas	0.5	0.8

Checkpoint metode ir ērti izmantojama un viegli saprotama. Metode viegli pielāgojama izmantošanai uzņēmumā, ja uzņēmumā ir pieejami darba produktivitātes rādītāji.

1.3.2.2. ExperiencePro

ExperiencePro ir analoģiju bāzēta programmatūras izstrādes darbietilpības novērtēšanas metode. To atbalsta rīks, kuru izstrādājusi *Software Technology Transfer* [EXP02]. Šī metode izmanto programmatūras ražošanas produktivitāti, to mērot funkcijpunktos stundā. Programmatūras ražošanas produktivitāti attiecīgajam projektam nosaka, izmantojot informāciju par ~500 projektiem, kas atrodas rīka *ExperiencePro* datu bāzē. Metode izmanto 21 produktivitātes faktoru, kuri pieskaņo programmatūras izstrādes produktivitāti, to uzlabojot vai pasliktinot atkarībā no situācijas projektā. Produktivitātes faktori sagrupēti četrās grupās.

1. Projekta faktori.

Projekta faktori ir, piemēram, pasūtītāja iesaistīšanās izstrādes procesā. Vislabākajā gadījumā pasūtītājs vai potenciālie lietotāji iesaistās ļoti aktīvi, bet sliktākajā – pasūtītājs nav ieinteresēts produkta izstrādē, līdzdalībai izstrādes procesā nav ieplānots laiks un resursi. Vēl projekta faktori ir izstrādes vides atbalsts, galveno izstrādātāju pieejamība, cik dažādas struktūrvienības vai projekti ir iesaistīti izstrādes procesā un izstrādes procesam atvēlētais laiks.

2. Procesa faktori.

Procesa faktori ir standartu un procedūru izmantošana izstrādes procesā, dažādu metožu un rīku izmantošana izstrādes procesā, prasību stabilitāte un kontrolējamība, izstrādes procesa organizētība.

3. Produkta faktori.

Produkta faktori ir programmatūras sarežģītība, datu bāzes apjoms, interfeisu skaits ar citām programmatūrām, prasības produkta kvalitātei un ātrdarbībai.

4. Cilvēka faktori.

Cilvēka faktori ir sistēmanalītiķu spējas, izstrādātāju zināšanas problēmapgabalā un izstrādes vidē, projekta vadītāja pieredze un izstrādātāju pieredze un spējas strādāt komandā.

Ja attiecīgais faktora novērtējums pēc situācijas analīzes projektā ir labāks par vidējo, tad uzlabojas produktivitāte, proti, produktu var izstrādāt

īsākā laikā, izmantojot mazāk resursus. Konkrētās produktivitātes faktoru vērtības ir *Software Technology Transfer* īpašums un tās nav pieejamas.

ExperiencePro metode ir vienkārša, tā ir balstīta uz informāciju par iepriekš izstrādātiem projektiem. Aprēķinot nominālo produktivitāti, tiek analizēta informācija par līdzīgiem projektiem no projektu bāzes. Bīstami ir tas, ka šādu līdzīgu projektu var būt maz, tad iegūtā produktivitāte nav uzticama.

Produktivitātes faktori izvēlēti, balstoties uz projektu datu analīzi. Nav teikts, ka šie ir visi faktori, kas ietekmē izstrādes produktivitāti.

1.3.3. Aplūkoto modeļu ilustrācija

Aplūkoto modeļu salīdzināšanai izmantosim 8 programmatūras izstrādes projektu darbietilpības datus. Situācija projektu izstrādes gaitā attēlota 14. tabulā. Izvēlēti divu programmatūras izstrādes uzņēmumu projekti, ievērojot turpmāk nosauktos kritērijus.

1. Programmatūras izstrāde pabeigta laikā no 1999 līdz 2002. gadam.

   Šādi projekti izvēlēti tādēļ, ka tehnoloģijas, kuras izmanto projektu izstrādei, mainās ļoti strauji. Dati par projektiem, kuri ir vecāki, nebūs salīdzināmi ar jaunākiem.

2. Ir pieejama informācija par programmatūras izstrādes procesu raksturojošiem riskiem (skat. 2. tabulu), kas novērtēti saskaņā ar 17. tabulā dotajiem ieteikumiem.

Projektu izstrādes darbietilpību un izstrādei nepieciešamo laiku ietekmē dažādi faktori (riski). Piemēram, vāja projekta vadība var pagarināt izstrādei nepieciešamo laiku un palielināt darbietilpību pat divkārt [BOE84]. Šeit projektu raksturošanai izvēlēti tie riski, kurus par svarīgiem uzskata aptaujātie programmatūras izstrādes procesa eksperti (skat. nodaļu "1.1.4.1. Ekspertu aptauja par riskiem").

14. tabula. Metožu salīdzināšanai izvēlētie programmatūras izstrādes projekti

Projekta raksturlielums	PRJ_1	PRJ_2	PRJ_3	PRJ_4	PRJ_5	PRJ_6	PRJ_7	PRJ_8
Izstrādes vide	InterBase, PHP, Apache	ORACLE	Java	ANSI C	Perl	Perl	ORACLE	ORACLE
Uzņēmējdarbības sfēra	Valsts nozīmes IS	Vadības IS	Iekārtu testēšanas programmatūra	Funkciju bibliotēka	Funkciju bibliotēka	Funkciju bibliotēka	Valsts nozīmes IS	Valsts nozīmes IS
Programmatūras apjoms funkcijpunktos	131	1680	321.2	104	640	156	1113	971
Programmatūras apjoms koda rindiņās	3275	42000	8030	2600	16000	3900	27825	24275
Pasūtītāja faktori								
Pasūtītāja nepietiekams tehniskais nodrošinājums	2	1	1	1	1	1	3	2
Apgrūtināta sadarbība ar pasūtītāju	2	2	1	1	1	1	2	3
Prasību riski								
Nekvalitatīvi definētas programmatūras prasības	2	4	4	2	2	2	4	2
Programmatūras prasību nestabilitāte	3	3	2	1	1	1	3	2
Projekta vadības riski								
Neatbilstošs projekta plānojums (nepietiekams laika periods, nepietiekami resursi)	4	3	1	2	2	2	2	2
Vāja projekta vadība	2	2	2	1	1	1	2	3
Izstrādātāju riski								
Izstrādes vides kļūdas	3	1	1	1	1	1	3	2
Izstrādātāju motivācijas trūkums	1	2	3	1	1	1	2	3

Projekta raksturlielums	PRJ_1	PRJ_2	PRJ_3	PRJ_4	PRJ_5	PRJ_6	PRJ_7	PRJ_8
Izstrādātāju nepietiekams tehniskais nodrošinājums	2	1	1	1	1	1	2	2
Izstrādātāju kadru mainība	1	1	3	1	1	1	3	2
Izstrādātāju nepietiekamas zināšanas par izstrādes tehnoloģijām un vidi	4	1	4	1	1	1	2	2
Apgrūtināta izstrādātāju sadarbība	1	1	1	1	1	1	3	2

Ievērojot attiecīgo projektu raksturlielumus, kas parādīti 14. tabulā, tika prognozēta projekta izstrādes darbietilpība saskaņā ar attiecīgo metodi. Rezultāti parādīti 15. tabulā, kur CD – cilvēkdienas, bet P/R – attiecīgās metodes darbietilpības prognozes attiecība pret reālo projekta izstrādes darbietilpību (100% - precīzs novērtējums, <100% - metode darbietilpību ir novērtējusi par zemu, >100% - metode ir pārvērtējusi reālo darbietilpību).

15. tabula. Dažādu darbietilpības prognozēšanas metožu rezultāti izvēlētajiem projektiem

Projekta ID	SLIM		Jensen		Checkpoint		COCOMO II		Reālā darbietilpība
	CD	P/R	CD	P/R	CD	P/R	CD	P/R	CD
PRJ_1	104	45.02%	346	149.78%	182	78.76%	144	62.34%	231
PRJ_2	18715	864.04%	22762	1050.87%	2857	131.91%	2490	114.96%	2166
PRJ_3	436	218.00%	260	129.82%	446	223.06%	394	197.00%	200
PRJ_4	82	57.16%	30	20.65%	144	100.31%	112	77.78%	144
PRJ_5	506	351.72%	687	477.22%	533	370.37%	826	573.61%	144
PRJ_6	157	163.99%	105	109.40%	260	270.83%	174	181.25%	96
PRJ_7	4750	118.16%	1273	31.66%	1855	46.14%	1522	37.86%	4020
PRJ_8	8739	491.78%	1695	95.41%	1337	75.27%	1310	73.72%	1777
Vidēji		288.73%		258.10%		162.08%		164.81%	

Aprēķinot dažādu metožu dotās darbietilpības prognozes attiecību pret reālo darbietilpību (skat. 15. tabula), var izdarīt turpmākos secinājumus.

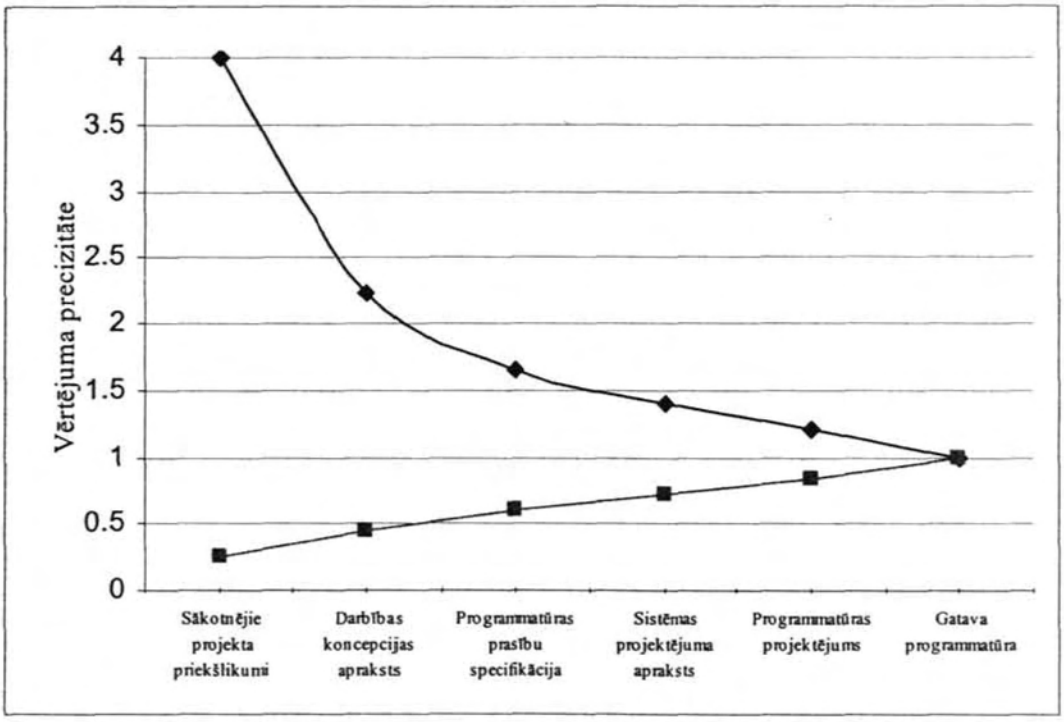
1. Formālās darbietilpības novērtēšanas metodes vidēji pārvērtē izstrādes darbietilpību. Iespējamie iemesli varētu būt tādi, ka metodes domātas lielu projektu darbietilpības novērtēšanai, kā arī tās izveidotas, pamatojoties uz ASV programmatūras izstrādes industrijas datiem.
2. Visprecīzākos darbietilpības novērtējumus izvēlētajiem projektiem vidēji iegūst, izmantojot *Checkpoint* un *COCOMO II* metodes.
3. Ir projekti, kuru darbietilpību pārvērtē visas metodes (skat. PRJ_5, PRJ_2). No šādas situācijas nav iespējams izvairīties projektiem, kuri ir "īpatnēji".

Piemēram, PRJ_5 izstrādē izmantoti koda fragmenti no cita projekta nevis kā atkārtoti izmantojami moduļi, bet tieši kā koda gabali. Tādējādi

mākslīgi palielinot izstrādājamā koda apjomu, kas ir viens no metožu ieejas parametriem. Lai arī šāda rīcība ir riskanta gadījumā, ja izmantotie koda gabali ir kļūdaini, PRJ_5 gadījumā šis risks nerealizējās un rezultāts bija veiksmīgs.

1.3.4. Secinājumi

Formālās programmatūras izstrādes darbietilpības novērtēšanas metodes kā ieejas parametru izmanto izstrādājamās programmatūras apjomu, kuru mēra funkcijpunktos vai programmrindiņās (skat. nodaļu "1.2.2.1. Programmatūras apjoma mēri"). Ļoti svarīgi ir pareizi novērtēt izstrādājamās programmatūras apjomu (funkcijpunktos vai programmrindiņās). Tas nebūt nav viegli, sevišķi, ja funkcionalitātes apjoms jāvērtē projekta sākumā, kad ļoti maz zināms par programmatūras prasībām. 10. attēlā parādīta līkne, kas raksturo programmatūras apjoma prognozes precizitāti, prognozētājam izmantojot attiecīgus dokumentus [COC2]. Piemēram, vadoties no sākotnējiem projekta priekšlikumiem, funkcionalitātes apjomu var pārvērtēt četrkārt. Tālāk, izmantojot nepareizu funkcionalitātes apjoma prognozi formālā modelī, iegūst neadekvātu rezultātu. 2002. gadā veiktais pētījums [BEN03] vairāk nekā 100 programmatūras izstrādes uzņēmumos parādīja, ka, piemēram, COCOMO izmanto 9% uzņēmumu, vēl 27% izmanto citas analītiskās metodes darbietilpības prognozēšanai. Galvenais iemesls, kāpēc tik maz uzņēmumos izmanto šīs metodes, pētījumā minēts tas, ka sarežģīti novērtēt tieši izstrādājamā produkta apjomu, kas ir viens no analītisko metožu ieejas paramatriem. Ar šo problēmu jārēķinās, izmantojot metodes, kas balstās uz Rayleigh-Norden darbietilpības aproksimācijas līkni vai izstrādes procesa produktivitāti.



10. attēls. Programmatūras izstrādes darbietilpības prognozes precizitāte. Uz horizontālās ass norādīti dokumenti, kas izmantoti apjoma vērtēšanai

No problēmām, kas saistītas ar izstrādājamā produkta apjoma novērtēšanu, iespējams izvairīties izmantojot analoģiju bāzētās metodes, par ieejas parametru izmantojot kādu citu izstrādes procesa vai produkta raksturlielumu. Piemēram, izstrādei nepieciešamo laiku, specifikācijas apjomu lappusēs u.c. Šādu pieeju piedāvā [ISB00], [EXP02], piedāvājot izmantot savas projektu datu bāzes. Tomēr jaunākie pētījumi [BEN03] rāda, ka uzticamus rezultātus dod dati par projektiem, kas uzkrāti pašā uzņēmumā.

Ir pieejami pētījumi par vienas vai otras metodes piemērotību vai nepiemērotību noteikta tipa projektiem. Piemēram, par SLIM metodes nepiemērotību maziem projektiem [KEM93], bez tam formālo modeļu dotie rezultāti ir stipri atkarīgi no lietotāja prasmes tos izmantot [API98]. Problēmu ar formālo modeļu doto prognožu precizitāti nevar atrisināt, izveidojot modeli, kas derētu visiem projektiem, visām izstrādes vidēm un apstākļiem. Katrai metodei varēs atrast piemērus, uz kuriem tās dotais rezultāts nebūs pietiekami

precīzs. Šeit risinājums ir formāli iegūtos rezultātus izvērtēt un pieskaņot, izmantojot vai nu paša plānotāja pieredzi vai nu uzņēmumā uzkrātos datus par izstrādes procesu.

Formālie programmatūras izstrādes procesa darbietilpības un izstrādes laika prognozēšanas modeļi ievēro dažādus riskus un to ietekmi uz izstrādes procesu, kas izpaužas kā darbietilpības palielināšanās (samazināšanās) un izpildes laika pagarināšanās (saīsināšanās). To, kādus riskus ņem vērā katrs modelis, atkarīgs no modeļu autoru pētījumiem par izstrādes procesu. Lai arī modeļus sauc par formāliem, tomēr nav iespējams izvairīties no subjektīva vērtējuma izmantošanas. Proti, izvērtējot izstrādes procesu ietekmējošos riskus, rezultātu tieši ietekmē plānotāja viedoklis par tiem. Ir riski, kurus ņem vērā visi aplūkotie formālie darbietilpības un izstrādei nepieciešamā laika prognozēšanas modeļi. Piemēram, izstrādātāju nepietiekamas zināšanas par izstrādes tehnoloģijām un vidi un neatbilstošs projekta plānojums. 16. tabulā parādīti ekspertu nosauktie riski, parādot, kuri formālie modeļi ņem vērā šos riskus un kuri nē. Ir ekspertu izvēlētie riski (skat. nodaļu "1.1.4.1 Ekspertu aptauja par riskiem"), kurus neņem vērā neviena no aplūkotajām formālajām metodēm, bet kurus par būtiskiem atzinuši aptaujātie eksperti. Piemēram, izstrādes vides kļūdas un izstrādātāju kadru mainība.

Tādējādi, lai prognozētu programmatūras izstrādei nepieciešamos resursus un laiku, iesākumā būtu jāveic risku analīze, pēc tam izvēloties piemērotāko formālo modeli. Tomēr šāda pieeja nav iespējama vairāku iemeslu dēļ.

1. Nav pieejamas zināšanas par vairāku modeļu izmantošanu. Pieredze rāda, ka pārsvarā uzņēmumos ir zināšanas par vienu vai divu formālu modeļu izmantošanu.
2. Modeļu izmantošana bieži vien saistīta ar attiecīga programmrīka iegādi, kuri mēdz būt dārgi (vairāki tūkstoši dolāru).

Tādēļ lietderīgi izvēlēties vienu formālu modeli, iemācīties to pareizi izmantot. Apzinoties to, ka formāls modelis nav ideāls, iegūto rezultātu nepieciešams pieskaņot. Pieskaņošanai jāizmanto informāciju, kas uzņēmumā uzkrāta par programmatūras izstrādes procesu: izstrādes procesa mērījumus un

risku analīzes rezultātus. Laika gaitā, informācijas bāzei kļūstot lielākai un iedibinoties uzticībai uzkrātajai informācijai, iespējams atteikties no formālu metožu izmantošanas, prognozes balstot tikai uz paša uzņēmuma datiem.

16. tabula. Ekspertu izvēlēto risku izmantošana dažādos formālajos modeļos (+ - metode ņem vērā attiecīgā riska ietekmi uz izstrādes procesu)

Izstrādes procesu ietekmējoši faktori [API02a]	*SLIM*	*Jensen*	*Checkpoint*	*ExperiencePro*	*COCOMO II*
Pasūtītāja nepietiekams tehniskais nodrošinājums		+			
Apgrūtināta sadarbība ar pasūtītāju	+			+	
Nekvalitatīvi definētas programmatūras prasības	+		+	+	+
Programmatūras prasību nestabilitāte	+		+	+	
Neatbilstošs projekta plānojums	+	+	+	+	+
Vāja projekta vadība			+	+	
Izstrādes vides kļūdas					
Izstrādātāju motivācijas trūkums				+	
Izstrādātāju nepietiekams tehniskais nodrošinājums	+	+	+		+
Izstrādātāju kadru mainība					
Izstrādātāju nepietiekamas zināšanas par izstrādes tehnoloģijām un vidi	+	+	+	+	+
Apgrūtināta izstrādātāju sadarbība	+		+	+	+

Vairākām analītiskajām metodēm, kas balstās uz Rayleigh-Norden līkni, ir izveidoti paplašinājumi, kas izmantojami kā atbalsts darbietilpības operatīvajai plānošanai, piemēram, SLIM. Analoģiju bāzētām metodēm šādu paplašinājumu nav. Taču, uzņēmumā regulāri uzkrājot izstrādes procesa mērījumu informāciju un risku analīzes rezultātus, tādu iespējams izveidot.

1.4. Secinājumi par pirmo nodaļu

Programmatūras izstrādes process ir sarežģīts un grūti pārskatāms, tajā iesaistītas gan tehnoloģijas, gan cilvēki. Vairāki nesen veikti pētījumi rāda, ka galvenā problēma izstrādes procesā ir izstrādātāju savstarpējā komunikācija gan par izmantotajiem risinājumiem, gan izstrādes procesa gaitu [CAR00], [KEI02]. Lai spriestu par izstrādes procesu, nepieciešama objektīva informācija par tā gaitu. Objektīvu informāciju sniedz izstrādes procesa mērīšana. Ir izstrādāti standarti un metodes, kas dod ieteikumus mērīšanas procesa realizācijai. Standarti CMMI, ISO/IEC 15939:2001, IEEE P1061 un mērīšanas procesa ieviešanas metodoloģijas [SOL99], [HET93] dod rekomendācijas mērīšanas procesa organizēšanai un pārvaldīšanai. Tomēr tie nesniedz ieteikumus, ko tieši un kādos gadījumos vajadzētu mērīt, atzīstot, ka šis jautājums ir ļoti specifisks katram programmatūras ražošanas uzņēmumam un pat katram atsevišķam projektam. Tādēļ mērīšanas procesa plānošanas augstākais iespējamais līmenis ir uzņēmuma līmenis. Lai gan izstrādes procesa mērīšanai tiek pievērsta arvien lielāka un lielāka uzmanība, par ko liecina kaut vai tas, ka tiek pārstrādāti programminženierijas standarti ISO 9000-3, ISO/IEC 12207 vairāk uzmanības veltot tieši mērījumiem kā vienīgajam informācijas avotam par izstrādes procesu, mērīšanas procesa ieviešana uzņēmumos bieži vien ir neveiksmīga. [API00A], [API02B] kā galvenais mērīšanas procesa veiksmes faktors minēta efektīva savākto mērījumu izmantošana. Svarīgi ir regulāri demonstrēt projektu izstrādātājiem, ka viņu savāktie mērījumus regulāri izmanto ar izstrādes procesu saistītu problēmu risināšanai. Otrs būtisks veiksmes faktors mērījumu programmas ieviešanā ir mēģināt maksimāli izmantot to informāciju par izstrādes procesu, kas dabiski rodas izstrādes procesa gaitā. Tādējādi maksimāli tiek ietaupīts izstrādātāju darbs šim atbalsta procesam, samazinot izstrādātāju pretestību mērījumu vākšanai.

Ir pieejamas metodes, kuras, izmantojot standartu dotās rekomendācijas mērīšanas procesa organizēšanai, iesaka, ko un kā mērīt konkrētu mērķu sasniegšanai. Piemēram, metodes, kas iesaka risku identificēšanai izmantot izstrādes procesa mērījumus [HAY96], [KAS00], [SED01]. Tomēr autori uzsver, ka mērījumi ir tikai risku indikatori, pēc būtības risku identificēšana ir radošs process. Šo pašu atzinumu var attiecināt arī uz mērījumu izmantošanu izstrādes procesa plānošanai: lai arī mērījumi parāda objektīvo situāciju izstrādes projektā, tie izmantojami kā papildus informācija plānotājam, atzīstot, ka izstrādes procesa plānošana ir radošs process.

Pirmajā nodaļā aplūkoti trīs dažādi mēri, kas raksturo izstrādes procesu un rezultātā iegūstamo produktu: programmatūras apjoma mēri, produktivitātes mēri un kvalitātes mēri. Izmantojot šos mērus dažādās kombinācijās, iespējams iegūt vispusīgu informāciju par izstrādes procesu un izstrādājamo produktu. Katra konkrētā mēra popularitāte ir tieši atkarīga no tā saprotamības un iegūšanas ērtuma. Mēri mainās, mainoties programmatūras izstrādes tehnoloģijām. Līdz ar mēriem mainās metrikas, kas balstītas uz šiem mēriem. Piemēram, [EXP02] pētījumi rāda, ka izstrādes procesa produktivitāte palielinās par 10% gadā.

Ja izstrādes procesa mērījumi sniedz priekšstatu par izstrādes procesu visām iesaistītajām pusēm, tie nav interpretējami atrauti no vides, kurā izstrāde notiek. Šo informāciju, savukārt, nodrošina risķu pārvaldības gaitā uzkrātā informācija. Pirmajā nodaļā aplūkoti standarti CMMI, IEEE P1540 un metodes risku pārvaldībai, kas risku pārvaldības procesa organizāciju balsta uz šiem standartiem. Risku pārvaldības metodes [GLU94], [BAS98], [BOE91] koncentrējas uz riska identificēšanu un definēšanu, uzskatot to par būtiskāko un sarežģītāko risku pārvaldības daļu, kas ir subjektīva un atkarīga no konkrētā vērtētāja spējas ieraudzīt attiecīgo risku. Veikta ekspertu aptauja par izstrādes procesa riskiem un to apkarošanas preventīvajām un korektīvajām darbībām, izmantojot DELPHI metodi. Šīs aptaujas mērķis bija noskaidrot tos riskus, preventīvās un korektīvās darbības šo risku apkarošanai, kas aktuāli programmatūras izstrādātājiem Latvijā, salīdzinot iegūtos rezultātus ar pasaules pieredzi. Iegūtie rezultāti apstiprināja dažādos avotos atrodamos riskus,

piemēram, jaunu tehnoloģiju izmantošana, nekvalitatīvas izstrādājamā produkta prasības [LOC96], [JON98], [HET93], [CMM99], [BOE91], [KEI02], bet parādīja arī jaunus, piemēram, izstrādes vides kļūdas. Ekspertu aptaujā minētas gan preventīvas, gan korektīvas darbības risku mazināšanai, kas saistītas ar dažāda veida izstrādātāju savstarpējo komunikāciju un komunikācijas ar pasūtītāju uzlabojumiem. Tāpat vairākas aptaujā minētās darbības risku mazināšanai ietver informācijas uzkrāšanu par izstrādes procesu un šīs informācijas izmantošanu dažādu risku apkarošanas plānošanai, tādējādi parādās praktiska nepieciešamība šos procesus integrēt, izmantojot to dotos rezultātus.

Risku pārvaldības un izstrādes procesa mērīšanas rezultātus izmanto programmatūras izstrādes procesa raksturlielumu prognozēšanas modeļi. Pirmajā nodaļā aplūkoti gan analītiski, gan analoģiju bāzēti programmatūras izstrādes procesa darbietilpības un izstrādei nepieciešamā laika prognozēšanas modeļi un demonstrēti šo metožu dotie rezultāti dažādiem reāliem, pabeigtiem programmatūras izstrādes projektiem. Aplūkotie modeļi izmanto datus par programmatūras izstrādes projektiem (mērījumus), ievērojot dažādus riskus un to ietekmi uz izstrādes procesu, kas izpaužas kā darbietilpības palielināšanās (samazināšanās) un izpildes laika pagarināšanās (saīsināšanās). To, kādus riskus ņem vērā katrs modelis, atkarīgs no modeļu autoru pētījumiem par izstrādes procesu.

Aplūkotie modeļi koncentrējas uz darbietilpības un izstrādei nepieciešamā laika prognozēšanu jaunas programmatūras izstrādes projektiem [PUT92], [JEN84], [JON96], [COC2], [EXP02], [PAR95], [TAU81]. Atsevišķiem modeļiem ir izstrādāti papildinājumi specifiskākiem projektiem, piemēram, programmatūras uzturēšanas darbietilpības prognozēšanai [COC2], [EXP02]. Ievērojot to, ka modeļi balstās uz izstrādes procesa mērījumiem, tie nepārtraukti noveco, attīstoties izstrādes tehnoloģijām. Piemēram, ievērojot to, ka izstrādes procesa produktivitāte palielinās par 10% gadā, izmantojot gadu vecu darbietilpības prognozēšanas modeli, iegūsim prognozi, kas par 10% pārvērtē izstrādei nepieciešamo darbietilpību pat tajā gadījumā, ja ideāli pratīsim izmantot attiecīgo modeli. Lai risinātu šo problēmu, modeļu autori ik

gadus atjaunina (uztur) attiecīgo modeli [QSM02], [JON96], nepārtraukti vācot datus par izstrādes projektiem. Principā šo "novecošanas" problēmu nevar atrisināt analītiskajiem modeļiem, kamēr analoģiju bāzētajiem šī problēma nav tik aktuāla, ja projektu bāzi regulāri papildina ar datiem par jauniem projektiem.

Modeļiem, kas balstīti uz Rayleigh-Norden darbietlpības aproksimācijas līkni, ir izstrādāti papildinājumi darbietilpības operatīvajai plānošanai [PUT92]. Analoģiju bāzētajiem modeļiem šādu papildinājumu nav, jo to izstrādei nepieciešamo datu iegūšana par izstrādes procesu ir ievērojami sarežģītāka nekā izstrādes procesa aproksimēšana, izmantojot analītiskas metodes. Šāda papildinājuma izveide prasītu integrēt mērīšanas un risku pārvaldības procesus, lai šo procesu rezultātus izmantotu plānošanai. Šādu papildinājumu pieejamība nodrošinātu turpmāk nosaukto.

1. Radītu iespēju prognozēt ne tikai izstrādes procesa darbietilpību un izstrādei nepieciešamo laiku, bet arī citus izstrādes procesa raksturlielumus.
2. Efektīvi izmantotu objektīvu informāciju par izstrādes procesu, ko sniedz izstrādes procesa mērīšana.
3. Parādītu mērīšanas procesa dalībniekiem, ka savāktā mērījumu informācija tiek nepārtraukti izmantota.
4. Padarītu risku pārvaldību par nepieciešamību, integrējot to izstrādes procesa plānošanā.

Nākamajā nodaļā piedāvāta autores izveidota mērījumu un risku bāzēta projekta plānošanas metode, kura nodrošina iepriekš minēto.

2. Mērījumu un risku bāzētas projektu plānošanas atbalsta metode

Mērījumu un risku bāzētas projektu plānošanas atbalsta metode (Mērījumu un risku bāzētas projektu plānošanas atbalsta metode - MERIME) izmantojama dažādu izstrādes procesa raksturlielumu, tai skaitā darbietilpības un izstrādei nepieciešamā laika, prognozēšanai gan operatīvās plānošanas vajadzībām, gan visam izstrādes procesam kopumā.

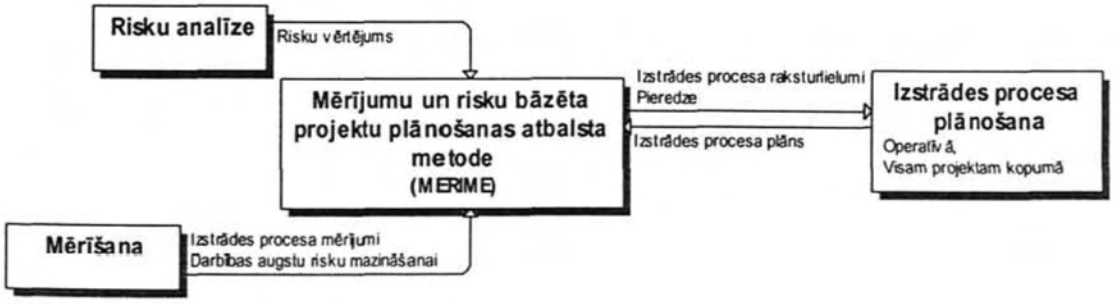
Šādas metodes nepieciešamību nosaka turpmāk nosauktie argumenti.

1. Iepriekš aplūkotās formālās izstrādes prognozēšanas metodes koncentrējas uz izstrādes procesa darbietilpību un izstrādei nepieciešamo laiku visam izstrādes procesam kopumā (skat. nodaļu "1.3. Formālu metožu izmantošana izstrādes procesa prognozēšanai"), tomēr praksē nepieciešams prognozēt arī citus izstrādes procesa raksturlielumus, piemēram, radītā produkta apjomu, dokumentēšanai un testēšanai nepieciešamo darbietilpību u.c.
2. Izstrādes procesa raksturlielumus jāprognozē ne tikai visam izstrādes procesam kopumā, bet praksē nepieciešams prognozēt šos lielumus arī operatīvās plānošanas vajadzībām.
3. Izstrādes procesa mērīšana sniedz objektīvu informāciju par izstrādes procesu, taču kritiski ir prast efektīvi izmantot šo informāciju. Mērīšanas procesa dalībniekiem ir nepārtraukti jārāda, ka savāktā mērījumu informācija tiek nepārtraukti izmantota (skat. nodaļu "1.2. Informācijas uzkrāšana par programmatūras izstrādes procesu"). Tādēļ nepieciešama metode, kas integrē mērīšanas procesu ar citiem izstrādes procesa procesiem. Šeit īpaši noderīgi ir integrēt mērīšanas procesu ar plānošanu.
4. Risku pārvaldība ir būtiska programmatūras izstrādes procesa sastāvdaļa, taču praksē nav populāri izmantot formālas metodes risku pārvaldībai. Neformāli risku pārvaldība tomēr notiek katrā programmatūras izstrādes projektā kā projekta pārvaldības sastāvdaļa. Taču neformāls process ir grūti vadāms, tādēļ nepieciešams risku pārvaldību padarīt risku pārvaldību par nepieciešamību, integrējot to izstrādes procesa plānošanā.

MERIME metode integrē trīs programmatūras izstrādes procesa atbalsta procesus: plānošanu, mērīšanu un risku pārvaldību. Šī metode izmantojama gan programmatūras izstrādes procesa pārvaldīšanai, gan kvalitātes pārvaldīšanai.

Sākotnēji uzlūkosim piedāvāto MERIME metodi kā "melno kasti" (skat. 11. attēls), noskaidrojot turpmāk nosaukto.

1. Metodes izmantošanas gaitā iegūstamos rezultātus (skat. nodaļu "2.2 Risku un mērījumu metodes rezultāts").
2. MERIME saistību ar citiem projekta atbalsta procesiem, radot metodes izmantošanai nepieciešamo ieejas informāciju (skat. nodaļu "2.1 Ieejas datu sagatavošana").



11. attēls. Mērījumu un risku bāzētas projektu plānošanas atbalsta metodes saistība ar citiem procesiem

2.1. Ieejas datu sagatavošana

Metodes ieejas datus rada trīs programmatūras izstrādes procesa atbalsta procesi (skat. 11. attēls).

1. Programmatūras izstrādes procesa risku pārvaldība (skat. nodaļu "1.1 Programmatūras izstrādi ietekmējošie riski").
2. Programmatūras izstrādes procesa mērīšana (skat. nodaļu "1.2 Informācijas uzkrāšana par programmatūras izstrādes procesu").
3. Programmatūras izstrādes procesa plānošana (skat. nodaļu "1.3 Formālu metožu izmantošana izstrādes procesa prognozēšanai").

2.2. Risku un mērījumu metodes rezultāts

MERIME metode nodrošina tās informācijas sistemātisku uzkrāšanu un analīzi, kas nepieciešama izstrādes procesa plānošanai (skat. 11. attēls). MERIME izmantošanas rezultātus var iedalīt divās grupās: tiešas ietekmes un netiešas ietekmes rezultāti. Tiešās ietekmes rezultāti ir tie, kurus iegūst kā tiešu atbalstu plānošanas procesā. Šie rezultāti nosaukti turpmāk.

1. Izstrādes procesa raksturlielumus operatīvai plānošanai un visam izstrādes procesam kopumā. Piemēram, cik procentu no kopējās projekta darbietilpības nākamajā mēnesī (vai nākamajos divos) patērēs projekta pārvaldība vai projekta dokumentēšana, kāda būs visa izstrādes procesa darbietilpība utml.
2. Izstrādātāju pieredzi, kā rīkoties saspringtā situācijā projektā. Piemēram, ja programmatūras izstrādes projektā ir aktuāla programmatūras prasību nestabilitāte, tad MERIME izmantošanas rezultātā iespējams saņemt informāciju, kā līdzīgās situācijās rīkojušies citu projektu izstrādātāji.
3. Vai patreizējā situācija programmatūras izstrādes projektā atbilst plānotajai.

Netiešas ietekmes rezultāti saistīti ar dažādu izstrādes procesa atbalsta procesu integrāciju un ir nosaukti turpmāk.

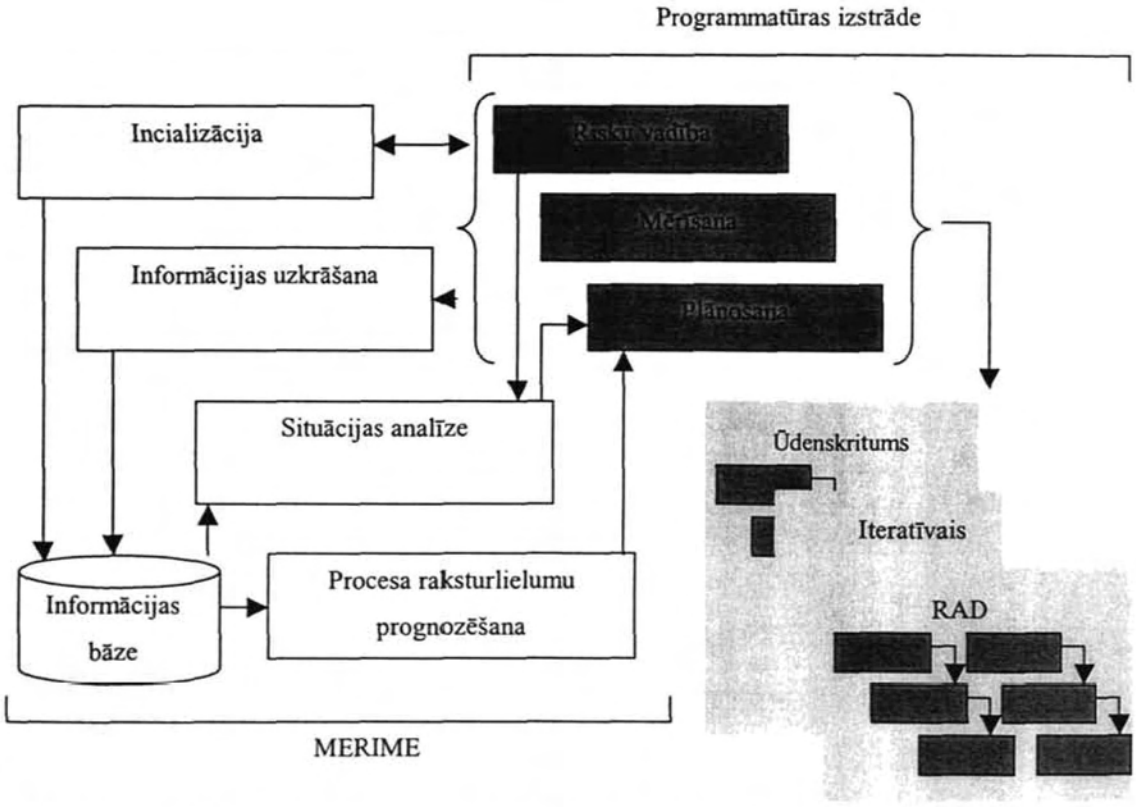
1. Risku analīzes process integrēts ar plānošanas procesu.
2. Izstrādes procesa mērījumu rezultātus izmanto nepārtraukti visā izstrādes procesā.

2.3. MERIME metode

MERIME metodes principiālā shēma dota 12. attēlā. Katrā programmatūras izstrādes projektā notiek izstrādes procesa plānošana, formāla vai neformāla risku vadība un izstrādes procesa mērīšana. MERIME metodes izmantošanā ir četri soļi, kuri turpmāk aprakstīti atsevišķās nodaļās.

1. Inicializācija (skat. nodaļu "2.3.2. Inicializācija").
2. Informācijas uzkrāšana par programmatūras izstrādes projekta riskiem, izstrādes procesa mērījumiem un plāniem (skat. nodaļu "2.3.3. Informācijas uzkrāšana").

3. Situācijas analīze (skat. nodaļu "2.3.4. Situācijas analīze").
4. Procesa raksturlielumu prognozēšana (skat. nodaļu "2.3.5. Mērījumu vērtību prognozēšana plānotajam projektam").



12. attēls. MERIME metodes izmantošanas principiālā shēma

MERIME metode sadarbojas ar trim izstrādes procesa atbalsta procesiem, kuri ir integrēti programmatūras izstrādes procesā. Tādējādi metodes darbība nav atkarīga no programmatūras izstrādei izmantojamā programmatūras izstrādes dzīves cikla modeļa. Metode izveidota tā, lai tā jāintegrē tikai ar trim procesiem: risku pārvaldību, mērīšanu un plānošanu.

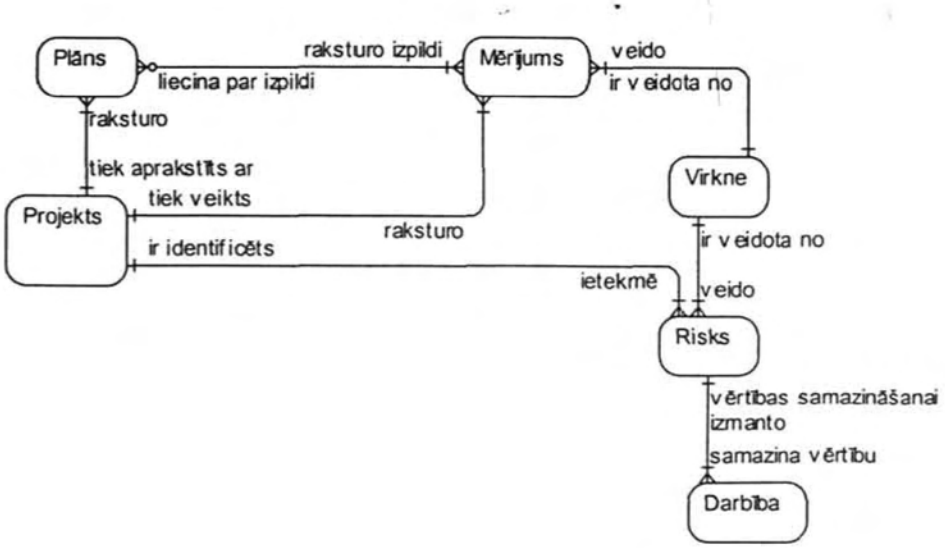
2.3.1. MERIME informācijas bāze

Sākumā aplūkosim MERIME informācijas bāzi, kurā uzkrāj visu informāciju, kuru izmanto MERIME metode.

Izstrādātā MERIME metodes informācijas bāzes struktūra, kuru visērtāk attēlot, izmantojot meta-modeli. Entītiju-relāciju diagramma, kas dota 13.

attēlā, attēlo MERIME metodes informācijas bāzes formālo meta-modeli, kurā ir 6 objektu tipi.

1. "Projekts" ir programmatūras izstrādes projekts, kura risku analīzes un mērīšanas rezultāti tiek iekļauti MERIME informācijas bāzē.
2. "Risks" ir vērtējamais risks. Piemēram, risks "programmatūras prasību nestabilitāte" novērtēts kā augsts.
3. "Mērījums" ir attiecīgā projekta raksturlielums, kas tiek mērīts. Piemēram, "dokumentēšanas darbietilpība procentos" ir 15% utml.
4. "Darbība" ir darbība augstas riska vērtības samazināšanai. Piemēram, "Nodibināt izmaiņu vadības padomi", "Piesaistīt problēmapgabala ekspertus no pasūtītāja puses", "Uzsākt sarunas par projekta turpināšanas lietderīgumu" utml.
5. "Virkne" ir mērījuma vai riska vērtējumu virkne, kurā glabājas riska vērtējumi (mērījuma vērtības) pa izvēlētajiem laika intervāliem. Virknes atribūti ir riska vai mērījuma nosaukums, laika moments, kurā risku vai mērījumu vērtē un riska vai mērījuma vērtība.
6. "Plāns" ir izstrādes procesa plāns, kurš tiks izmantots situācijas analīzei projektā, lai salīdzinātu izstrādes procesa mērījumus un izvērtētu to atbilstību plānam. Plāna atribūti ir datums, kad plāns veidots, plānotā darbietilpība u.c. atribūti, kas paredzēti inicializācijas laikā.



13. attēls. MERIME metodes objektu tipi un relācijas

MERIME metodes inicializācijas gaitā, izmantojot šo meta-modeli, apraksta MERIME informācijas bāzi, definējot konkrētus modeļa objektus ar visiem to atribūtiem. Objektu tipi ir meta-modelī definētie.

MERIME informācijas bāzes definēšanai var izmantot meta-modeļa bāzētu universālu repozitoriju [API96], [API00], kas atvieglo MERIME informācijas bāzes attēlošanu. Tāpat MERIME informācijas bāzes vizualizācijai iespējams izmantot datu apraksta valodu [API95].

2.3.2. Inicializācija

Inicializācijas gaitā notiek MERIME metodes pielāgošana izmantošanai attiecīgajā uzņēmumā, to integrējot ar risku vadības, mērīšanas un plānošanas procesu tradīcijām.

Inicializācijas gaitā izvēlas laika periodus, kad notiks informācijas reģistrēšana MERIME informācijas bāzē. Metodes izmantošanas gaitā laika momentos t_i tiks apkopoti un reģistrēti mērījumu rezultāti, risku vadības rezultāti un plānots turpmākais programmatūras izstrādes process (skat. (F 25)).

$$T = \left\{ t_i \mid i = \overline{1,\infty}, i \in N \right\} \qquad \textbf{(F 25)}$$

Piemēram, inicializācijas gaitā izlemj, ka šie laika momenti būs katra mēneša pirmā nedēļa. Tātad turpmāk katra mēneša pirmajā nedēļā apstrādās un analizēs risku vadības, mērījumu informāciju un informāciju par izstrādes procesa plāniem. Šajos laika momentos informāciju reģistrēs MERIME informācijas bāzē.

MERIME metodes inicializācijas solī faktiski notiek risku vadības, mērīšanas un plānošanas procesu sinhronizācija. Turpmāk nosauktas tās prasības, kas jāievēro katram no minētajiem procesiem.

1. Risku pārvaldība

Inicializācijas gaitā programmatūras izstrādes uzņēmuma līmenī vienojas, kādus riskus analizēs, pēc kādas skalas novērtēs attiecīgos riskus, cik

bieži tie tiks izvērtēti un kurš būs atbildīgs par attiecīgā riska vadību. Tāpat jāparedz risku analīzes procesa kontrole, proti, cik bieži notiks risku analīzes procesa apskates un kurš būs atbildīgs par risku analīzes procesa uzraudzību. Citiem vārdiem sakot, plāno risku pārvaldības procesu.

Lai risku pārvaldības procesu integrētu MERIME, inicializācijas rezultātā tiek izveidota vadāmo risku kopas sagatave (skat. (F 26)). Turpmākajā MERIME izmantošanas gaitā uzkrāj kopas R elementus.

$$R = \left\{ R_j \mid j = \overline{1,m} \right\}, \text{ kur} \qquad \textbf{(F 26)}$$

m - pārvaldāmo risku skaits

Piemēram, R_1 ir risks "nestabilas programmatūras prasības", R_2 ir risks "izstrādātāju nepietiekamas zināšanas izstrādes vidē" utt.
Svarīga risku vadības sastāvdaļa ir akceptējamā risku sliekšņa S noteikšana. MERIME metodes izmantošanas gaitā, kādam riskam R_j pārsniedzot robežu S, ir jāplāno speciālas darbības riska mazināšanai. Parasti šāda risku sliekšņa noteikšana ir risku vadības procesa sastāvdaļa.

2. Mērīšana

Inicializācijas gaitā programmatūras izstrādes uzņēmuma līmenī vienojas, kādu katra projekta izstrādes procesa mērījumu informāciju uzkrās un analizēs, lai noteiktu katra programmatūras izstrādes projekta atbilstību plāniem un analizētu situāciju projektā.

Inicializācijas rezultātā izveido mērījumu kopas sagatavi (skat. (F 27)). Turpmākajā MERIME izmantošanas gaitā aizpilda kopu M attiecīgajiem projektiem.

$$M = \left\{ M_k \mid k = \overline{1,n} \right\}, \text{ kur} \qquad \textbf{(F 27)}$$

n - fiksējamo mērījumu skaits

Piemēram, M_1 ir mērījums "darbietilpība programmatūras kodēšanai mēnesī cilvēkdienās", M_2 ir mērījums "no pasūtītāja saņemto problēmziņojumu skaits" utt.

3. Plānošana

MERIME ietvaros izstrādes procesa plānu P uzskata par risku un mērījumu prognozi noteiktam laika momentam t_i (F 28).

Inicializācijas gaitā izlemj, kādu projekta plānošanas informāciju uzskatīs par būtisku katram programmatūras izstrādes projektam uzņēmumā. Piemēram, plānošanas gaitā tiks prognozēta mērījuma M_1 "izstrādes procesa darbietilpība nākamajam izstrādes mēnesim" vērtība (šādu mērījuma M_1 plānoto vērtību apzīmēsim ar M_1^P) un riska R_1 "izstrādātāju nepietiekamas zināšanas izstrādājamās programmatūras problēmapgabalā" ietekme uz izstrādes procesu (šādu riska R_1 plānoto vērtību apzīmēsim ar R_1^P).

$$P = \left\{ M^P \cup R^P \mid M^P \subseteq M, R^P \subseteq R \right\} \qquad \textbf{(F 28)}$$

Jāievēro, ka gan mērījumu, gan riska novērtējumu vērtības var nesakrist precīzi ar plānotajām vērtībām. Tādēļ katram mērījumam M_k un riskam R_j jānodefinē pieļaujamās robežas, kurās atrodoties riska vai mērījuma vērtībai, izstrādes procesu uzskata par atbilstošu plānam. Turpmāk šīs pieļaujamās robežas apzimēsim ar ε_k^M k-tā mērījuma pieļaujamo robežu, ε_j^R j-tā riska pieļaujamo robežu. MERIME metode nenosaka precīzas robežas, šo robežu noteikšanā jāvadās no praktiskās pieredzes darbā ar izstrādes procesa mērījumiem. Šāda subjektivitāte ir pieļaujama, jo plānošanas procesā tā kā tā nav iespējams izvairīties no subjektivitātes (skat. secinājumus par risku pārvaldību nodaļā "1.1. Programmatūras izstrādi ietekmējošie riski").

Inicializācijas gaitā faktiski notiek turpmāk nosauktās aktivitātes.

1. Informācijas bāzes modeļa definēšana, izmantojot MERIME informācijas bāzes meta-modeli (skat. nodaļu "2.3.1. MERIME informācijas bāze").
2. Risku analīzes, mērīšanas un plānošanas procesu sinhronizācija tā, lai mērījumu un risku vērtības tiktu fiksētas vienādos laika momentos t_i.

2.3.3. Informācijas uzkrāšana

Informācijas uzkrāšanas gaitā uzkrāj informāciju par projektiem tā, kā tas paredzēts inicializācijas solī.

MERIME informācijas bāzē reģistrē plānošanas, risku analīzes un mērījumu rezultātus, kas fiksēti laika momentos t_i. Tādējādi informācijas uzkrāšanas gaitā veidojas risku un mērījumu virknes (F 29).

$$R_j = \langle r_{j,i} \rangle,\ j = \overline{1,m} \qquad \text{(F 29)}$$

$$M_k = \langle m_{k,i} \rangle,\ k = \overline{1,n}$$

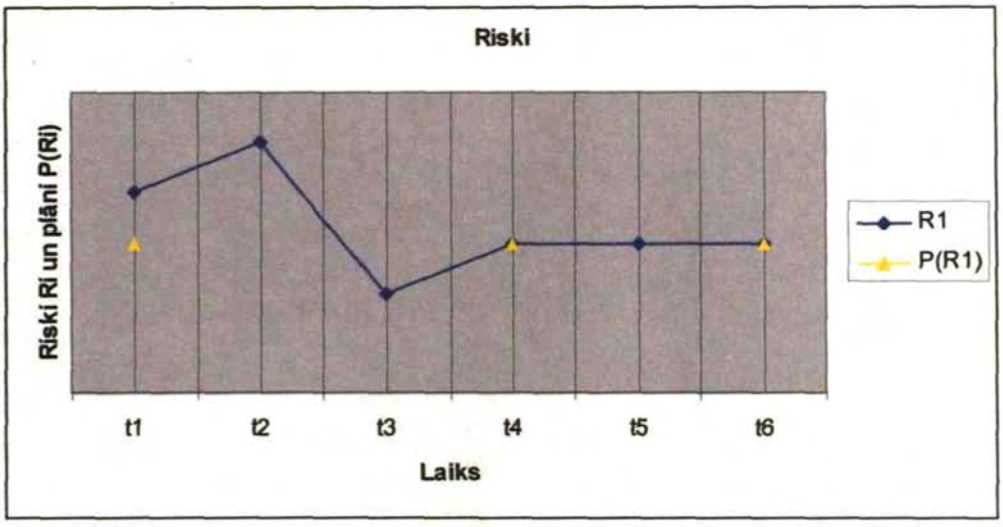
$$P_i = \{M_i^P \cup R_i^P\},\ \text{kur}$$

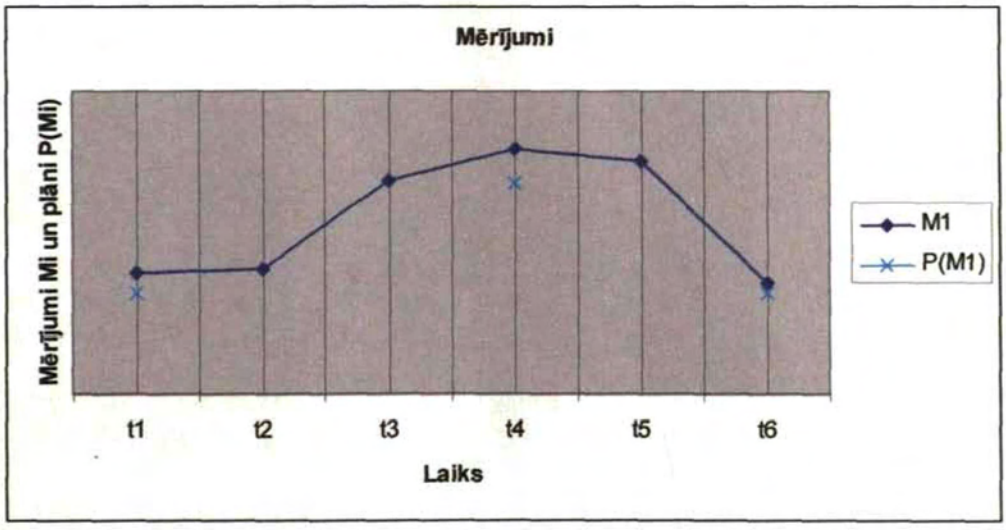
$r_{j,i}$ - j-tā riska novērtējums laika momentā t_i

$m_{k,i}$ - k-tā mērījuma vērtība laika momentā t_i

P_i - izstrādes procesa plāns laika momentam t_i

Risku kopas R_j, mērījumu kopas M_k un plānu kopas P_i var attēlot atbilstības grafikos ar laika kopu T (skat. 14). Laika kopas T elementi ir laika momenti, kuros attiecīgā riska vai mērījuma vērtība fiksēta.





14. attēls. MERIME informācijas bāzes aizpildīšana ar datiem par izstrādes procesa plāniem, risku analīzes rezultātiem un mērījumiem

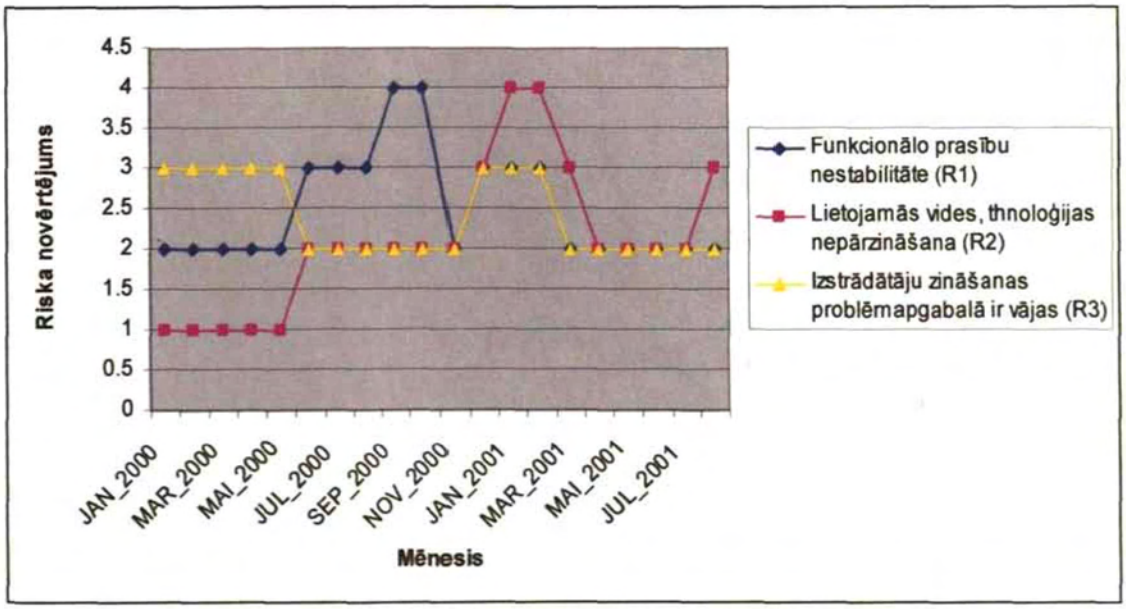
2.3.3.1. Risku analīzes informācijas uzkrāšana

MERIME metode neuzliek nekādus ierobežojumus risku analīzes metodes izvēlei, risku analīzes gaitā var izmantot IEEE P1540, CMMI risku vadības metožu dotos ieteikumus, kas aprakstīti nodaļās "1.1.2.1. CMMI – SE/SW ieteiktā risku vadība" un "1.1.2.2. Standarta IEEE P1540 ieteikumi risku pārvaldības procesam", kā arī jebkuras citas risku analīzes metodes. Tomēr, lai risku analīzes dati būtu izmantojami MERIME metodei, tiem jāatbilst šādiem noteikumiem.

1. Risku analīzes rezultāti jāapkopo regulāri inicializācijas laikā izvēlētajos laika momentos t_i.
2. Visiem riskiem $r_{j,i}$ jābūt izvērtētiem pēc vienādas skalas, par kuru vienojas inicializācijas gaitā.

Šis nav uzskatāms par spēcīgu ierobežojumu. Ja uzņēmumā ir noteikta risku analīzes kārtība, tad šeit iespējams izmantot uzņēmumā pieņemto risku vērtēšanas skalu.

Piemēram, R_1 ir risks "funkcionālo prasību nestabilitāte", bet attiecīgā riska novērtējuma vērtību virkne ir <2, 2, 2, 2, 2, 3, 3, 3, 4, 4>, kur katrs virknes elements atbilst riska R_1 novērtējumam laika momentā t_i. Laika momenti t_i ir, piemēram, katra mēneša pirmā nedēļa, sākot ar projekta uzsākšanas brīdi. Tādējādi risku virknes attēlo projekta situācijas izmaiņas laikā.



15. attēls. Projekta PROJ_1 risku novērtējums no 2000. gada janvāra līdz 2001. gada augustam

15. attēlā redzamajā piemērā attēlota risku virkne projektam PROJ_1 tā izstrādes pusotra gada laikā. Šajā piemērā risku novērtēšanai izmantota kvalitatīvā risku novērtēšanas metode, kur 1 nozīmē, ka attiecīgais risks neietekmē negatīvi izstrādes procesu, bet 5 – risks noteikti ietekmēs izstrādes

procesu negatīvi. Šajā piemērā redzams, ka situācija projektā tā izstrādes laikā var stipri mainīties, piemēram, projekta izpildes gaitā, no 2000. gada novembra sākot izmantot jaunu izstrādes tehnoloģiju (R_2).

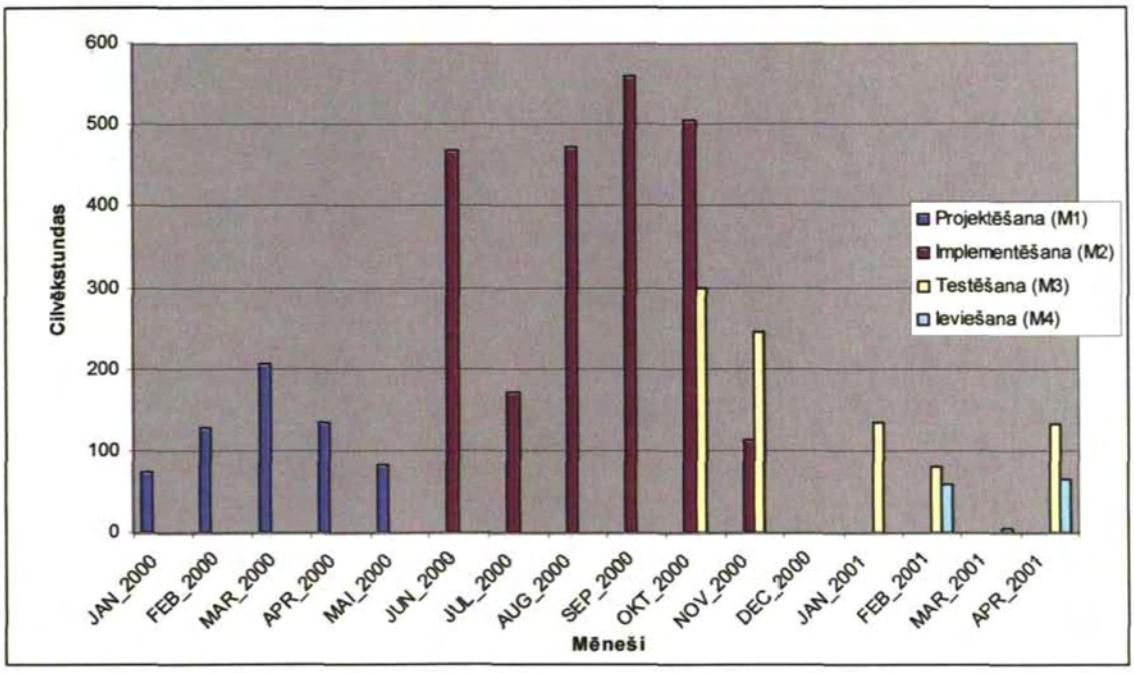
Riska novērtējumam pārsniedzot akceptējamo riska slieksni S, jāplāno aktivitātes attiecīgā riska samazināšanai vai seku mīkstināšanai.

2.3.3.2. Izstrādes procesa mērījumu informācijas uzkrāšana

Izstrādes procesa mērīšanai ir savas tradīcijas katrā programmatūras ražošanas uzņēmumā. Pat tad, ja uzņēmumā nav oficiālas mērījumu programmas, katrā uzņēmumā ir noteiktas izstrādes procesa mērīšanas tradīcijas. Līdzīgi, kā risku analīzes gadījumā, metode nenosaka, kādi tieši mērījumi jāvāc, lai gan jāatceras, ka metodes darbināšanas viens no rezultātiem ir izstrādes procesa raksturlielumi, kas tiek tieši iegūti no izstrādes procesa mērījumiem. Tādēļ jāievēro, lai viena uzņēmuma ietvaros mērījumi tiktu vākti un apkopoti pēc vienotas metodikas.

Lai risku analīzes dati būtu izmantojami MERIME metodei, tiem jāatbilst šādiem noteikumiem.

1. Izstrādes procesa mērījumi jāapkopo regulāri inicializācijas laikā izvēlētajos laika momentos t_i.
2. Visiem MERIME informācijas bāzē esošajiem mērījumiem $m_{k,i}$ jābūt vāktiem un apstrādātiem pēc vienotas metodikas, par tāpat vienojas inicializācijas gaitā. To nodrošina vienota Mērījumu programma programmatūras izstrādes uzņēmumā (skat. nodaļu "1.2.4. Mērījumu programmu ieviešanas principi").



16. attēls. Darbietilpības mērījumu uzkrāšana projektam PROJ_1 no 2000. gada janvāra līdz 2001. gada aprīlim

Piemēram, M_1 ir mērījums "darbietilpība programmatūras projektēšanai mēnesī cilvēkstundās", bet attiecīgā mērījuma vērtību virkne ir <80, 120, 210, 125, 90, 0>, kur katrs virknes elements atbilst mērījuma M_1 vērtībai laika momentā t_i. Laika momenti t_i ir, piemēram, katra mēneša pirmā nedēļa, sākot ar projekta uzsākšanas brīdi. Tādējādi mērījumu virknes attēlo projekta attīstību laikā.

16. attēlā redzamajā piemērā attēlotas četru dažādu mērījumu virknes projektam PROJ_1 tā izstrādes pusotra gada laikā. Šajā piemērā redzams, kā attīstās projekta izstrāde, vispirms ir projektēšana (M_1, 2000. gada janvāris – maijs), tad implementēšana (M_2, 2000. gada jūnijs - novembris), testēšana un ieviešana.

2.3.3.3. Informācijas uzkrāšana par plāniem

Izstrādes procesa plānošanai, analogi kā mērīšanai, ir savas tradīcijas katrā programmatūras ražošanas uzņēmumā. MERIME metode nenosaka, kāda tieši informācija jāuzkrāj, to iespējams pielāgot katra uzņēmuma izstrādes procesa plānošanas tradīcijām atbilstoši inicializējot.

MERIME izstrādes procesa plānu uzskata par dažādu izstrādes procesa raksturlielumu un risku vērtību prognožu kopu. Tādējādi izstrādes procesa plānošana faktiski reducējas uz dažādu izstrādes procesa raksturlielumu vērtību prognozēšanu.

2.3.4. Situācijas analīze

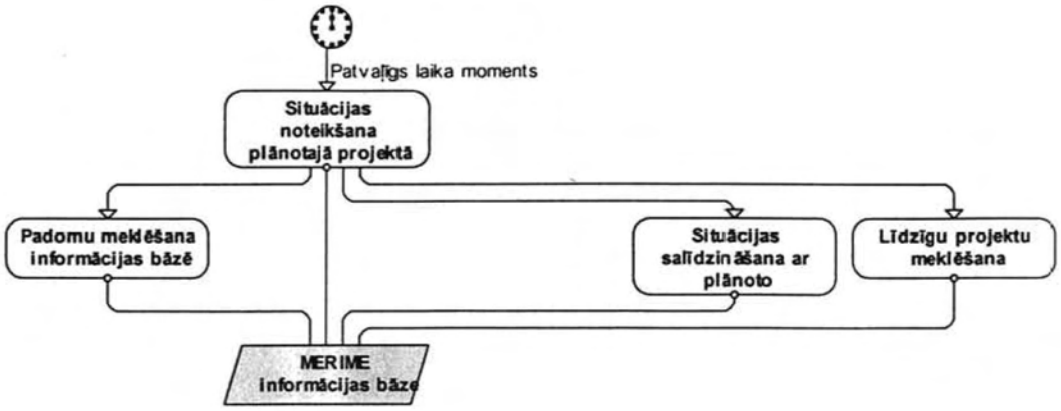
Situācijas analīze ir pirmais solis, kad MERIME metodi sāk izmantot projekta izstrādes procesa plānošanai (turpmāk – Plānotais projekts).

Situācijas analīzi projektā veic, pamatojoties uz informāciju, kas uzkrāta MERIME informācijas bāzē, un atbildot uz jautājumiem: vai esam uztaisījuši tik, cik bijām plānojuši, vai esam iekļāvušies plānotajā darbietilpībā un laikā, kāda ir izstrādātā produkta kvalitāte, kādi sagaidāmi izstrādes procesa raksturlielumi turpmāk?

Situācijas analīzes gaitā izmanto gan to informāciju, kas MERIME informācijas bāzē uzkrāta par citiem projektiem, gan to informāciju, kas uzkrāta par Plānoto projektu un nosaukta turpmāk.

1. Plānotā projekta plānošanas informāciju P^{Pl}, kura reģistrēta MERIME informācijas bāzē.
2. Plānotā projekta risku analīzes informāciju R^{Pl} un mērījumu informāciju M^{Pl}, kas reģistrēta MERIME informācijas bāzē. Šāda informācija var nebūt pieejama, ja MERIME metodi izmanto jauna projekta plānošanai vēl pirms darbu uzsākšanas.
3. Risku analīzes informāciju R un mērījumu informāciju M, kas uzkrāta MERIME informācijas bāzē par citiem projektiem, kuru izstrāde jau pabeigta vai kuri atrodas izstrādes procesā.

Situācijas analīzes gaitā ir vairāki soļi, kuri parādīti 17. attēlā redzamajā biznesprocesā [GRD98]. Turpmāk katrs metodes solis aprakstīts atsevišķā nodaļā.



17. attēls. MERIME metodes situācijas analīzes solis

2.3.4.1. Situācijas noteikšana plānotajā projektā

Šajā solī nosaka, kāda ir situācija plānotajā projektā, izmantojot MERIME informācijas bāzē uzkrāto informāciju par plānoto projektu. Situācijas noteikšanai MERIME metode izmanto izstrādes procesa dažādus raksturlielumus.

1. Izstrādes procesa mērījumi M^{Pl} – raksturo izstrādes procesu "no iekšpuses". Izstrādes procesa mērījumi parāda izstrādes procesa attīstības fāzi.
2. Izstrādes procesa risku analīzes informācija R^{Pl} – raksturo vidi, kādā programmatūra tiek ražota.

Tādējādi situāciju projektā faktiski raksturo risku un mērījumu informācija laika momentā t_i. Faktiski situācija projektā laika momentā t_i ir komponentu kortežs $s(t_i)$ pār kopām M_1, M_2, ..., M_n, R_1, ..., R_m (skat. (F 30)). Šādu kortežu virkne, sākot no laika momenta t_1 līdz laika momentam t_i, raksturo situācijas attīstību projektā.

$$s(t_i) = (m_{g,i}, r_{k,i}), \text{ kur} \qquad \textbf{(F 30)}$$

$$m_{g,i} \in M_g, g = \overline{1,n}$$

$$r_{k,i} \in R_k, k = \overline{1,m}$$

Gadījumā, kad MERIME metodi izmanto jauna programmatūras izstrādes projekta plānošanai, situācija jānovērtē, izmantojot plānošanas informāciju.

2.3.4.2. Situācijas salīdzināšana ar plānoto

Situācijas salīdzināšanai ar plānoto faktiski tiek salīdzinātas risku un mērījumu vērtības ar attiecīgo risku un mērījumu plānotajām vērtībām.

MERIME metode nepiedāvā konkrētus kritērijus, kad uzskatīt, ka projekta izstrādes process atbilst plānotajam un kad nē. Tomēr katram mērījumam M_k un riskam R_j individuāli var nodefinēt robežas, kurās attiecīgo procesa raksturlielumu var uzskatīt par pieņemamu (skat. (F 31)). Šīs robežas jānodefinē jau metodes inicializācijas gaitā.

$$\forall M_k \exists \varepsilon_k^M : / m_{k,i}^{Pl} - m_{k,i} / < \varepsilon_k^M, k = \overline{1,n} \qquad \textbf{(F 31)}$$

$$\forall R_j \exists \varepsilon_j^R : / r_{j,i}^{Pl} - r_{j,i} / < \varepsilon_j^R, j = \overline{1,m} \text{ , kur}$$

$m_{k,i}$ – M_k mērījuma vērtība laika momentā t_i

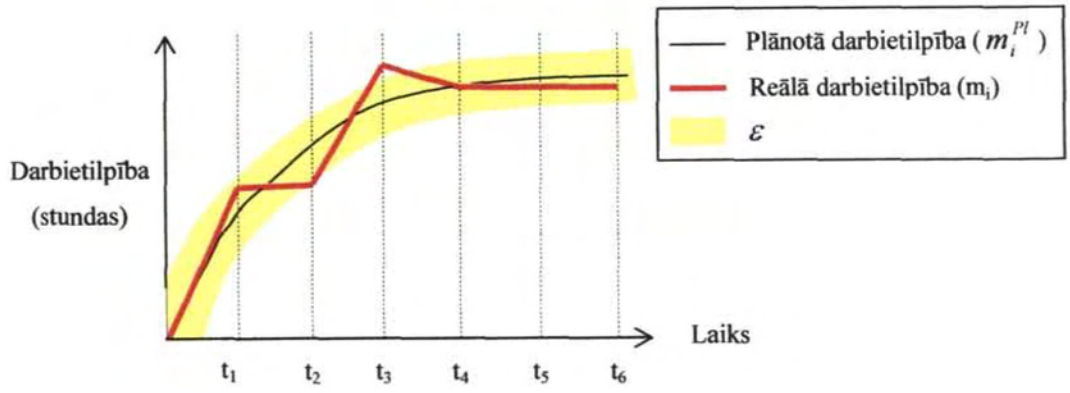
$m_{k,i}^{Pl}$ - plānotā M_k mērījuma vērtība laika momentā t_i

$r_{j,i}$ – R_j riska novērtējums laika momentā t_i

$r_{j,i}^{Pl}$ - plānotais R_j riska novērtējums laika momentā t_i

Piemēram, esošās situācijas salīdzināšanai ar plānoto izmanto mērījuma un laika atbilstības grafiku (skat. 18. attēls). Šajā atbilstības grafikā redzams, ka reālā darbietilpība programmatūras projekta izstrādei pārsvarā saskan ar plānoto, ir pieļaujamajās robežās, bet ir bijusi arī ārpus akceptējamās robežas.

Tomēr vajadzētu atturēties no striktas, automatizētas pieļaujamo robežu noteikšanas un kontroles, katrs gadījums ir jāizskata individuāli.



18. attēls. Plānotās un reālās darbietilpības salīdzinājums programmatūras izstrādes projektam

2.3.4.3. Līdzīgu projektu meklēšana

MERIME metodes izmantošanas gaitā MERIME informācijas bāzē tiek meklēti līdzīgi projekti plānotajam projektam.

Definīcija

MERIME metode programmatūras izstrādes projektus A_1 un A_2 uzskata par līdzīgiem laika momentā t_i, ja šiem projektiem izpildās nosacījums (F 32).

$$/ r_{j,i}^{A_1} - r_{j,i}^{A_2} / \le \varepsilon_j^R , / m_{k,i}^{A_1} - m_{k,i}^{A_2} / \le \varepsilon_k^M \text{, kur} \qquad \textbf{(F 32)}$$

$m_{k,i}^{A}$ - M_k mērījuma vērtība laika momentā t_i

$r_{j,i}^{A}$ - R_j riska novērtējums laika momentā t_i

Citiem vārdiem sakot, MERIME metode programmatūras izstrādes projektus A_1 un A_2 uzskata par līdzīgiem laika momentā t_i, ja šiem projektiem eksistē mērījuma vērtība un riska novērtējums, kas atrodas attiecīgā mērījuma un attiecīgā riska ε robežās. Piemēram, ir divi programmatūras izstrādes projekti A_1 un A_2, kuru izstrāde uzsākta pirms mēneša (laika moments t_1). Abiem projektiem ir aktuāls risks – izstrādei nepieciešamā laika trūkums.

Projekta A_1 funkcionalitātes apjoms (mērījums) ir 250 funkcijpunkti, bet A_2 funkcionalitātes apjoms ir 230 funkcijpunkti. Projekti A_1 un A_2 būs līdzīgi mēnesi pēc projekta uzsākšanas, ja $\varepsilon^{M}_{Funkcionalitāt\_apjoms}$=20%. Proti, riska novērtējumi ir sakrītoši, bet mērījuma vērtības atšķiras mazāk kā par 20%.

MERIME metode programmatūras izstrādes projektus A_1 un A_2 uzskata par līdzīgiem laika posmā $[t_i, t_{i+1}]$, ja šie projekti ir līdzīgi katrā laika momentā no t_i līdz t_{i+1} vienam un tam pašam riskam un mērījumam.

Ja projekti ir līdzīgi laika posmā $[t_i, t_{i+1}]$, tad var teikt, ka šiem projektiem ir līdzīgas attiecīgā riska un mērījuma virknes garumā l.

Šeit izvēlētais līdzības kritērijs izvēlēts pēc minimālisma principa šādu apsvērumu dēļ.

1. Lai būtu iespējams atrast informāciju arī situācijās, kad ļoti maz ir zināms par plānojamo projektu.
2. Izsmalcinātu un sarežģītu kritēriju izveide neatrisinātu problēmu, ka atrasto līdzīgo projektu raksturlielumi būtu dažādi, jo tā ir analoģiju bāzētu plānošanas metožu iedzimta īpatnība [ISB01], iegūtais rezultāts jāizvērtē plānotājam pašam. Šeit izmantotais princips ir analogs informācijas drošības jomā izmantotajam: neieguldīt līdzekļus informācijas aizsardzībā, bet apdrošināt informāciju pret zudumu [CIS02].

Līdzīgu projektu meklēšanai zināšanu bāzē jāatrod visi tie projekti, kuri ir līdzīgi plānotajam projektam saskaņā ar MERIME metodes projektu līdzības definīciju. Līdzīga projekta meklēšanas zināšanu bāzē rezultāts ir paraugprojektu kopa Q.

2.3.4.4. Padomu meklēšana informācijas bāzē

Šīs aktivitātes mērķis ir atrast tās darbības, kas atrodamas MERIME informācijas bāzē attiecīgo risku mazināšanai, lai pēc tam izvērtētu, vai tās ir piemērotas izmantošanai plānotā projekta patreizējā situācijā.

Padomu meklēšanai zināšanu bāzē izmanto plānojamā projekta riska novērtējumu attiecīgajā situācijā. Ja plānojamā projekta attiecīgā riska novērtējums pārsniedz akceptējamo risku slieksni S, tad MERIME

informācijas bāzē tiek sameklētas visas tās darbības, kas ir tikušas reģistrētas MERIME informācijas bāzē attiecīgā riska samazināšanai.

Rezultāts padomu meklēšanai MERIME informācijas bāzē ir darbības augstu risku samazināšanai. Šis arī ir MERIME metodes viens no izmantošanas rezultātiem.

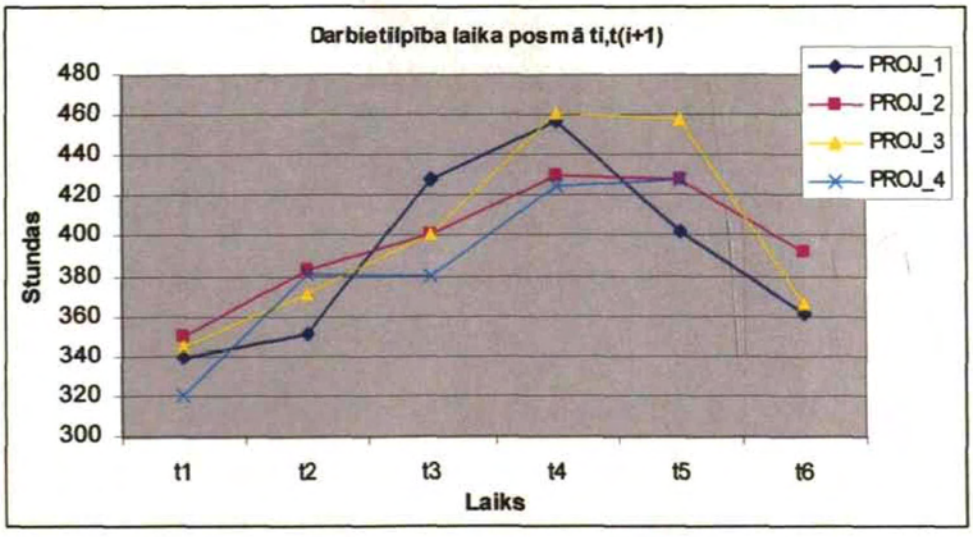
2.3.5. Mērījumu vērtību prognozēšana plānotajam projektam

Mērījumu vērtību prognozēšanas laikā nosaka, kādas varētu būt dažādu mērījumu vērtības attiecīgajā situācijā plānotajam projektam nākamajā laika momentā t_{i+1}.

Mērījumu vērtību prognozēšanai plānotajam projektam izmantojam iepriekšējā solī atrasto paraugprojektu kopā Q ietilpstošo līdzīgo projektu mērījumu datus laika momentam t_{i+1}. Plānotā projekta attiecīgā mērījuma vērtība tiek prognozēta līdzīgo projektu attiecīgā mērījuma pieļaujamajās ϵ robežās.

Piemēram, 19. attēlā redzamajā piemērā prognozēsim plānotajam projektam PROJ_4 izstrādes darbietilpību (mērījums M_2) laika momentā t_6. Izmantosim to, ka projektam PROJ_4 laika posmā $[t_1, t_5]$ ir līdzīgi projekti PROJ_1, PROJ_2 un PROJ_3. Šie projekti ir līdzīgi, jo ir līdzīgas risku R_1 un R_2 virknes laika posmiem $[t_1, t_6]$ un līdzīgas mērījumu M_1 un M_2 virknes laika posmiem $[t_1, t_5]$.

Laika momenti	t_1	t_2	t_3	t_4	t_5	t_6
M_1: Funkcionalitātes apjoms (funkcijpunkti), $\varepsilon_1^M = 20\%$						
Paraugprojekts: PROJ_1	1221	1228	1228	1228	1228	1228
Paraugprojekts: PROJ_2	1178	1178	1178	1178	1178	1178
Paraugprojekts: PROJ_3	1205	1205	1205	1210	1220	1252
Plānotais projekts: PROJ_4	1108	1123	1238	1200	1200	
M_2: Izstrādes darbietilpība laika posmā $t_i - t_{i+1}$ (stundas), $\varepsilon_2^M = 20\%$						
Paraugprojekts: PROJ_1	340	352	428	457	402	362
Paraugprojekts: PROJ_2	351	384	401	430	428	392
Paraugprojekts: PROJ_3	346	372	401	461	458	367
Plānotais projekts: PROJ_4	321	382	381	425	428	?
R_1: funkcionālo prasību nestabilitāte, S = 3, $\varepsilon_1^R = 1$						
Paraugprojekts: PROJ_1	3	2	3	3	3	2
Paraugprojekts: PROJ_2	2	3	2	2	2	3
Paraugprojekts: PROJ_3	2	2	3	4	3	3
Plānotais projekts: PROJ_4	2	2	2	3	2	2
R_2: apgrūtināta sadarbība ar Pasūtītāju, S = 3, $\varepsilon_2^R = 1$						
Paraugprojekts: PROJ_1	2	2	2	1	1	1
Paraugprojekts: PROJ_2	2	1	2	2	2	3
Paraugprojekts: PROJ_3	2	2	1	1	1	1
Plānotais projekts: PROJ_4	1	2	2	1	2	2



19. attēls. Plānotā projekta PROJ_4 mērījumu vērtību prognozēšanas piemērs

2.4. MERIME izmantošanas sagatave

Šajā nodaļā dota MERIME metodes sagatave, proti, ieteikumi MERIME metodes inicializācijai un datu uzkrāšanai, lai šo metodi varētu izmantot jebkurā programmatūras izstrādes uzņēmumā, parādot MERIME metodes inicializācijas soli (skat. 12. attēls) un nosakot mērījumu pieļaujamās robežas ε_i^M.

MERIME metodes inicializācijas gaitā uzņēmumā jāvienojas par risku analīzes rezultātu, mērīšanas rezultātu un plānošanas informācijas savākšanas un apstrādes kārtību un biežumu. Tāpat jāizpēta uzņēmumā eksistējošie risku analīzes, mērīšanas un plānošanas procesi, lai garantētu, ka savāktie informācija savākta pēc vienādas metodikas.

2.4.1.1. Risku, mērījumu un plānošanas informācijas apstrādes biežums

Ieteicamais risku, mērījumu un plānošanas datu savākšanas un izvērtēšanas biežums ir reizi mēnesī. Šāds datu apkopošanas biežums ir optimāls un ir izvēlēts, pamatojoties uz praksi. Datus apkopojot biežāk, katrs jaunais datu punkts maz atšķiras no iepriekšējā, informācijas ir daudz, tādēļ tā grūti pārskatāma. Datus apkopojot retāk kā reizi mēnesī, maziem projektiem (3-6 mēneši [YOU97]) ir pārāk maz datu punktu, lai varētu izmantot MERIME metodi šo projektu plānošanai.

2.4.1.2. Risku analīzes informācija

Risku pārvaldības process tiek tiek organizēts saskaņā ar standartā IEEE P1540 (skat. nodaļu "1.1.2.2. Standarta IEEE P1540 ieteikumi risku pārvaldības procesam") dotajiem ieteikumiem. Lai risku pārvaldības procesa rezultātus izmantotu MERIME metodē izstrādes procesa plānošanai, risku vadības procesa plānā paredzētas turpmāk nosauktās aktivitātes.

1. Risku novērtēšanai izmanto kvalitatīvo risku novērtējuma metodi, izmantojot 17. tabulā doto skalu. Riska novērtējuma vērtībai pārsniedzot 3 (riska robeža S, situācija ir sliktāka par vidējo), jāplāno (rakstiski) aktivitātes attiecīgā riska mazināšanai.

Piemēram, ja projekta izpildes gaitā risks "programmatūras prasību nestabilitāte" pārsniedz 3, iespējamā aktivitāte šī riska samazināšanai ir "Nodibināt projekta izmaiņu vadības padomi, kurā ietilptu gan pasūtītāja, gan izpildītāja puses pārstāvji". Šo mērījumu fiksē atbildīgais par risku analīzi, parasti projekta pārvaldnieks. Sākotnēji MERIME informācijas bāzi aizpilda ar ekspertu dotajām aktivitātēm risku samazināšanai (skat. nodaļu "1.1.4 Programmatūras izstrādes procesa riski").

17. tabula. Risku novērtējuma skala

Riska atzīme	Riska atzīmes izvēles kritēriji
1	Riska ietekme nav paredzama vai risks nav piemērojams
2	Pastāv niecīga iespēja, ka risks var ietekmēt izstrādes procesu negatīvi, bet nav nepieciešams plānot īpašas aktivitātes riska apkarošanai
3	Pastāv iespēja, ka risks var ietekmēt izstrādes procesu negatīvi, jāpārdomā rīcības alternatīvas
4	Pastāv iespēja, ka risks var ietekmēt izstrādes procesu negatīvi: jāplāno konkrētas aktivitātes riska apkarošanai un pastiprināti jāseko riskam
5	Risks gandrīz noteikti negatīvi ietekmēs izstrādes procesu: jāplāno konkrētas aktivitātes riska apkarošanai, pastiprināti jāseko riskam un riska apkarošanai jāpiesaista papildus vadošais personāls

2. Risku analīzes rezultāti jāapkopo un jāievieto MERIME informācijas bāzē regulāri, saskaņā ar izvēlēto informācijas apkopošanas biežumu, tādējādi risku analīzes informācija būtu sinhronizēta ar mērījumu informāciju.
3. Regulāri izvērtē riskus R_1 līdz R_{12} saskaņā ar programmatūras izstrādes procesa risku, kas aktuāli programmatūras izstrādes procesam, sarakstu (skat. 2. tabulu nodaļā "1.1.4. Programmatūras izstrādes procesa riski").
4. Risku pieļaujamās robežas $\varepsilon_i^R = 1$, ja izmanto risku novērtēšanas skalu, kas dota 17. tabulā.
5. Sākotnēji MERIME informācijas bāzē reģistrē tās darbības risku apkarošanai, kuras katram no riskiem R_1 līdz R_{12} dotas nodaļā "1.1.4. Programmatūras izstrādes procesa riski".

Piemēram, programmatūras izstrādes projektam PROJ_1 risku analīzes rezultāti, doti 18. tabulā, kur risku atzīmes izvēlētas saskaņā ar 17 dotajiem kritērijiem.

18. tabula. Risku analīzes rezultātu piemērs

Risks	Jan	Feb	Ma	Apr	Mai	Jun
Programmatūras prasību nestabilitāte	2	3	2	3	4	3
Izstrādātāju nepietiekamas zināšanas izstrādes vidē	1	1	2	4	4	3

2.4.1.3. Mērījumu informācija

Mērījumu informācijas uzkrāšanu organizē, izmantojot Mērījumu programmu programmatūras izstrādes uzņēmumā (skat. nodaļu "1.2. Informācijas uzkrāšana par programmatūras izstrādes procesu"). Šajā nodaļā dota minimālā mērījumu informācija, kas nepieciešama MERIME metodes izmantošanai.

Programmatūras izstrādes procesa mērīšanu paredz projekta plānošanas stadijā, katrā programmatūras izstrādes projektā ieplānojot turpmāk nosaukto mērījumu vākšanu un apstrādes kārtību. Uzkrātos datus jādara pieejamus mērījumu programmas vadītājam saskaņā ar izvēlēto biežumu.

1. Darba laika resursa uzskaiti.
   1.1. Programmatūras izstrādei patērētā darbietilpība attiecīgajā laika periodā cilvēkstundās (M_1). Robeža, kurā programmatūras izstrādei patērēto darbietilpību attiecīgajā laika periodā uzskatīt par atbilstošu plānam ε_1^M =20% (kā iegūt attiecīgās ϵ robežas skat. nodaļā "2.4.1.5. Mērījumu pieļaujamo robežu noteikšana").
   1.2. Radītās programmatūras apjoms attiecīgajā laika periodā funkcijpunktos, ja tie pielietojami vai programmrindiņās (M_2). Robeža, kurā programmatūras apjomu uzskatīt par atbilstošu plānam ε_2^M =20%.
   1.3. Programmatūras izstrādes procesa aktivitātēm patērētā darbietilpība attiecīgajā laika periodā (koncepcijas izstrādei, prototipa izstrādei,

prasību specificēšanai, projektēšanai, implementēšanai, testēšanai, ieviešanai, uzturēšanai, projekta pārvaldībai u.c.) cilvēkstundās (M_3). Robeža, kurā programmatūras izstrādes procesa aktivitātēm patērēto darbietilpību attiecīgajā laika periodā uzskatīt par atbilstošu plānam ε_3^M =20%.

2. No pasūtītāja ienākošo problēmziņojumu uzskaite.

Dati, kurus ieteicams reģistrēt par problēmziņojumu, doti "II Pielikums. Mērījumu datu savākšanas formas" (saskaņā ar standarta IEEE/EIA 12207.1, 6.10 prasībām).

Projekta izstrādes procesa mērījumu datu sākotnējo analīzi veic Mērījumu programmas vadītājs, aizpildot MERIME informācijas bāzi.

2.4.1.4. Plānošanas informācija

Inicializācijas gaitā jāparedz šādas plānošanas informācijas uzkrāšana.

1. Prognozētais izstrādājamās programmatūras apjoms programmrindiņās vai funkcijpunktos (M_1^P). Šo prognozi var iegūt, analizējot to informāciju, kas par izstrādājamo produktu ir pieejama. Piemēram, programmatūras apjomu funkcijpunktos var prognozēt, analizējot programmatūras prasību specifikāciju.

2. Darbietilpība plānotā apjoma izstrādei cilvēkstundās (M_2^P).

Programmatūras izstrādes procesa darbietilpības prognozēšanai ieteicams izmantot kādu formālu darbietilpības novērtēšanas metodi, piemēram, COCOMO II, SLIM u.c. (skat. nodaļu "1.3. Formālu metožu izmantošana izstrādes procesa prognozēšanai").

3. Programmatūras izstrādes procesa aktivitātēm patērētā darbietilpība attiecīgajā laika periodā (koncepcijas izstrādei, prototipa izstrādei, prasību specificēšanai, projektēšanai, implementēšanai, testēšanai, ieviešanai, uzturēšanai, projekta pārvaldībai, dokumentēšanai u.c.) (M_3^P).

Darbietilpības sadalījumam pa aktivitātēm var izmantot industrijas vidējos sadalījumus, kas pieejami dažādos avotos.

Tādējādi ir noteikta informācija, kādu nepieciešams uzkrāt MERIME informācijas bāzē, lai varētu MERIME metodi izmantot programmatūras izstrādes uzņēmumā.

2.4.1.5. Mērījumu pieļaujamo robežu noteikšana

Mērījumu vērtību pieļaujamās robežas ir svarīgs lielums, jo tās izmanto gan izvērtējot projekta izstrādes gaitu atbilstību plānotajai, gan arī, lai prognozētu izstrādes procesa raksturlielumus. Tādēļ šajā nodaļā aplūkota mērījumu ε_i^M robežu noteikšana.

Mērījumu vērtību pieļaujamo robežu noteikšana ir viena no problēmām, kāpēc sarežģīti izmantot analoģiju bāzētas metodes. Piemēram, vai tāda projekta darbietilpību, kura funkcionalitātes apjoms atšķiras par 30% no plānojamā projekta apjoma, var izmantot par pamatu plānojamā projekta darbietilpības prognozēšanai?

Mērījumu vērtību pieļaujamo robežu noteikšanai ir divas iespējas.

1. Izmantot praktisko pieredzi, kas ir uzņēmuma rīcībā, nosakot attiecīgo mērījumu robežas.

Šajā gadījumā vēlams izveidot ekspertu grupu, kurā piedalītos tie cilvēki, kuriem ir pieredze izstrādes procesa plānošanā attiecīgajā uzņēmumā. Šie cilvēki tad arī sākotnēji vienojas par pieļaujamām robežām katram mērījumam.

2. Izmantot mērījumu datu analīzi, analizējot mērījumu datus par pabeigtiem programmatūras izstrādes projektiem un par pabeigtu projektu izstrādes procesiem.

Šeit aplūkosim pieļaujamo mērījumu robežu noteikšanu, izmantojot mērījumu datu analīzi.

Uzdevums ir atrast mērījumu robežas saskaņā ar MERIME metodes līdzības definīciju. Saskaņā ar MERIME metodes definīciju, projektus uzskata par līdzīgiem, ja eksistē kaut viens risks un kaut viens mērījums, kuri atrodas definētajās mērījuma un riska robežās (skat. (F 32)).

Pieļaujamās mērījumu robežas būs atkarīgas no mērījumu bāzes, tādēļ katram uzņēmumam tās būs atšķirīgas. Šeit izvēlēsimies pieļaujamās mērījumu robežas, analizējot *ExperiencePro* [EXP02] datu bāzē esošo informāciju par

167 pabeigtiem programmatūras izstrādes projektiem. Detalizēts šajā datu bāzē pieejamās informācijas apraksts dots nodaļā "2.5.1. Informācijas bāze". Šeit izmantosim to, ka par katru programmatūras izstrādes projektu ir pieejama šāda mērījumu un risku informācija.

1. Izstrādātās programmatūras funkcionalitātes apjoms funkcijpunktos (M_1 saskaņā ar (F 27)).
2. Izstrādes procesa darbietilpība cilvēkstundās (M_2 saskaņā ar (F 27)).
3. Izstrādes procesa ilgums mēnešos (M_3 saskaņā ar (F 27)).
4. Informācija par 21 risku (R_1 līdz R_{21} saskaņā ar (F 26)), kur katrs risks novērtēts robežās no 1 (situācija šajā jomā ir izcila) līdz 5 (situācija ir slikta, izstrādes procesu noteikti ietekmēs negatīvi).

Tādējādi, izmantojot *ExperiencePro* datu bāzi, meklēsim pieļaujamās mērījumu robežas izstrādātās programmatūras funkcionalitātes apjomam (ε^{M_1} saskaņā ar (F 31)), izstrādes procesa darbietilpībai (ε^{M_2} saskaņā ar (F 31)) un izstrādes procesa ilgumam mēnešos (ε^{M_3} saskaņā ar (F 31)).

Lai meklētu pieļaujamās mērījumu robežas saskaņā ar MERIME līdzības definīciju, rīkosimies šādi.

1. Katram *ExperiencePro* datu bāzē esošajam projektam izvēlēsimies visus iespējamos mērījuma un riska pārus $p_{j,r}(m_j, r_i)$, kur $j = \overline{1,3}, i = \overline{1,21}, m_j \in M_j, r_i \in R_i$. Tādējādi katram projektam iegūstam 63 šādus pārus, kurus, saskaņā ar MERIME līdzības definīciju (skat. (F 32)), uzskatām par līdzības kritērijiem.
2. Katram projektam, katram mērījuma un riska pārim $p_{j,r}$ meklēsim līdzīgos projektus *ExperiencePro* datu bāzē. Attiecīgā mērījuma pieļaujamās robežas izvēloties iteratīvi, sākot no 100% līdz 5% ar soli -5%. Skaitīsim, cik projektiem ir atrasti līdzīgi pēc attiecīgā līdzības kritērija attiecīgajās mērījuma robežās.

Piemēram, 19. tabulā redzams, ka, izvēloties programmatūras apjoma mērījuma (M_1) pieļaujamās robežas 5%, pārī ar R_1, *ExperiencePro* datu bāzē vismaz viens līdzīgs projekts tiek atrasts 85.03% projektu.

1. Par pieļaujamo mērījuma robežu uzskatīsim tādu, kur līdzīgus projektus ExperiencePro bāzē saskaņā ar MERIME līdzības definīciju, var atrast 95% projektu.
2. Nav lietderīgi izvirzīt mērķi atrast kaut vienu līdzīgu 100% projektu, problēmu ar atsevišķiem "ārkārtas" projektiem nav iespējams atrisināt analoģiju bāzētām prognozēšanas metodēm [ISB01].

19. tabulā redzams, ka programmatūras apjoma mērījuma pieļaujamās robežas ir 20%, jeb ε^{M_1}=20%, jo šī ir minimālā ϵ robeža, kur pārī ar katru risku, līdzīgi projekti ir atrasti 95% projektu.

19. tabula. Izstrādājamās programmatūras apjoma pieļaujamās robežas meklēšana ExperiencePro datu bāzē

m_1 robežas	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}	r_{21}
5%	85.03%	86.23%	85.63%	83.23%	88.02%	85.63%	86.83%	86.23%	86.83%	86.23%	85.03%	80.24%	80.84%	83.83%	82.63%	83.83%	85.03%	85.63%	84.43%	83.83%	88.02%
10%	93.41%	91.62%	93.41%	91.02%	94.61%	94.01%	94.61%	92.22%	92.81%	95.21%	92.22%	92.22%	90.42%	94.01%	92.81%	94.01%	94.01%	95.21%	91.62%	92.81%	95.21%
15%	94.01%	96.41%	96.41%	95.21%	96.41%	96.41%	97.01%	94.61%	95.21%	97.60%	95.21%	95.21%	95.81%	96.41%	95.81%	97.01%	95.81%	97.01%	94.61%	97.01%	97.60%
20%	**97.01%**	**97.01%**	**97.60%**	**97.01%**	**98.20%**	**97.60%**	**98.20%**	**96.41%**	**97.60%**	**98.20%**	**97.60%**	**96.41%**	**97.60%**	**97.60%**	**97.01%**	**97.60%**	**97.60%**	**98.20%**	**96.41%**	**97.60%**	**98.20%**
25%	98.80%	97.60%	97.60%	97.60%	98.80%	98.20%	98.80%	97.60%	97.60%	98.80%	98.20%	97.01%	98.20%	97.60%	97.60%	98.20%	98.80%	98.20%	98.20%	98.20%	98.80%
30%	99.40%	98.20%	98.80%	98.80%	99.40%	98.80%	99.40%	98.20%	98.80%	99.40%	98.80%	97.60%	98.80%	98.20%	98.20%	98.80%	99.40%	98.80%	98.80%	98.80%	99.40%
35%	99.40%	98.20%	98.80%	98.80%	99.40%	98.80%	99.40%	98.20%	98.80%	99.40%	98.80%	98.20%	98.80%	98.20%	98.80%	99.40%	99.40%	98.80%	98.80%	99.40%	99.40%
40%	99.40%	98.20%	98.80%	98.80%	99.40%	98.80%	99.40%	98.20%	98.80%	99.40%	98.80%	98.20%	98.80%	98.20%	98.80%	99.40%	99.40%	98.80%	99.40%	99.40%	99.40%
45%	99.40%	98.80%	99.40%	98.80%	99.40%	98.80%	99.40%	98.80%	98.80%	99.40%	98.80%	98.20%	98.80%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%
50%	99.40%	98.80%	99.40%	98.80%	99.40%	98.80%	99.40%	98.80%	98.80%	99.40%	98.80%	98.80%	98.80%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%
55%	99.40%	98.80%	99.40%	98.80%	99.40%	98.80%	99.40%	98.80%	98.80%	99.40%	98.80%	98.80%	98.80%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%
60%	99.40%	98.80%	99.40%	98.80%	99.40%	98.80%	99.40%	98.80%	99.40%	99.40%	99.40%	98.80%	98.80%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%
65%	99.40%	98.80%	99.40%	99.40%	99.40%	98.80%	99.40%	98.80%	99.40%	99.40%	99.40%	98.80%	98.80%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%
70%	99.40%	98.80%	99.40%	99.40%	99.40%	99.40%	99.40%	98.80%	99.40%	99.40%	99.40%	98.80%	98.80%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%
75%	99.40%	98.80%	99.40%	99.40%	99.40%	99.40%	99.40%	98.80%	99.40%	99.40%	99.40%	98.80%	98.80%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%
80%	99.40%	98.80%	99.40%	99.40%	99.40%	99.40%	99.40%	98.80%	99.40%	99.40%	99.40%	98.80%	98.80%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%
85%	99.40%	98.80%	99.40%	99.40%	99.40%	99.40%	99.40%	98.80%	99.40%	99.40%	99.40%	98.80%	98.80%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%
90%	99.40%	98.80%	99.40%	99.40%	99.40%	99.40%	99.40%	98.80%	99.40%	99.40%	99.40%	98.80%	98.80%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%
95%	99.40%	98.80%	99.40%	99.40%	99.40%	99.40%	99.40%	98.80%	99.40%	99.40%	99.40%	98.80%	98.80%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%
100%	99.40%	98.80%	99.40%	99.40%	99.40%	99.40%	99.40%	98.80%	99.40%	99.40%	99.40%	98.80%	98.80%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%	99.40%

Lietojot šo pašu principu atlikušajiem diviem ExperiencePro bāzē esošajiem mērījumiem, iegūstam, ka izstrādes procesa darbietilpībai pieļaujamās robežas ir 20% (ε^{M_2}=20%) un izstrādes procesa ilgumam mēnešos pieļaujamās robežas ir 5% (ε^{M_3}=5%).

Pieļaujamās mērījumu robežas jāizvēlas MERIME metodes inicializācijas gaitā, kad informācijas bāzes vēl nav. Tādēļ ieteikums rīkoties šādi.

1. Sākotnēji inicializācijas gaitā izvēlēties pieļaujamās mērījumu robežas, vadoties no pieredzes.
2. Ja nav pieejama pieredze, tad sākotnēji jāizvēlas šajā nodaļā piedāvātās pieļaujamās mērījumu robežas.
3. Vēlāk, kad būs uzkrāta informācija par vairāk projektiem, atkārtoti meklēt attiecīgajiem mērījumiem pieļaujamās robežas, izmantojot šajā nodaļā piedāvāto principu.

20. tabula. Mērījumu vērtību pieļaujamās robežas, ja par MERIME informācijas bāzi izmanto *ExperiencePro* datu bāzi

Nr.	Mērījums	Apzīmējums	Vērtība
1.	Programmatūras apjoms	ε^{Apjoms}	20%
2.	Programmatūras izstrādes darbietilpība	$\varepsilon^{Darbietilpība}$	20%
3.	Programmatūras izstrādes ilgums	ε^{Ilgums}	5%

2.5. MERIME validācija

Šajā nodaļā aplūkota MERIME metodes validācija. MERIME metode validēta pret pabeigtiem programmatūras izstrādes projektiem, lai varētu spriest par MERIME metodes dotajiem rezultātiem. MERIME metodes validācijai ir vairāki soļi, kur katrs solis aplūkots atsevišķā nodaļā.

1. Informācijas bāzes izvēle (skat. "2.5.1. Informācijas bāze").
2. Reprezentatīvas plānojamo programmatūras izstrādes projektu kopas izvēle.
3. MERIME metodes izmantošana visu iespējamo mērījumu prognozēšanai katram projektam no iepriekšējā solī izvēlētās kopas.
4. MERIME metodes doto prognožu analīze un secinājumi par iegūtajiem rezultātiem.

2.5.1. Informācijas bāze

MERIME metodes validācijai izmantosim projektu pārvaldības metodes un rīka ExperiencePro izstrādei izmantotos datus [EXP02] (skat. nodaļu "1.3.2.2 ExperiencePro"). Šajā datu bāzē ir informācija par 800 reāliem, pabeigtiem programmatūras projektiem Somijā. Informācija par projektiem nāk no dažādām uzņēmējdarbības sfērām, kurās ļoti intensīvi izmanto IS. Piemēram, telekomunikācijas, finansu sektors. *ExperiencePro* datu bāzē esošo projektu dati izmantoti tāda paša nosaukuma programmatūras izstrādes projekta plānošanu atbalstoša rīka radīšanai, kurš ir pieejams tirgū. No šīs datu bāzes speciāli šī promocijas darba izstrādei *ExperiencePro* autori atlasīja 170 programmatūras izstrādes projektus, kuri pabeigti pēc 1996. gada.

Šīs informācijas bāzes izmantošana MERIME metodes validācijai ir pieļaujama, jo tajā iekļautā informācija par projektiem atbilst turpmāk nosauktajiem kritērijiem.

1. Informācijas bāzē iekļauta informācija par reāliem programmatūras izstrādes projektiem, kuru izstrāde pabeigta sekmīgi. Šeit sekmīgi nozīmē,

ka izstrādes procesa rezultātā ir radīta programmatūra, kura atbilst pasūtītāja noteiktajām prasībām.

2. Informācijas bāzē pieejamā informācija savākta, izmantojot vienotu mērījumu datu un risku analīzes rezultātu vākšanas metodiku.

Tādējādi varam uzskatīt, ka šī informācijas bāze atbilst MERIME informācijas bāzes nosacījumiem (skat. "2.3.1. MERIME informācijas bāze").

MERIME metodes validācijas pirmajā solī izveidosim MERIME informācijas bāzi saskaņā ar MERIME metodes soļiem, par pamatu izmantojot *ExperiencePro* datu bāzi.

2.5.1.1. MERIME informācijas bāzes modelis

Definēsim MERIME informācijas bāzes modeli saskaņā ar 13. attēlā doto MERIME informācijas bāzes meta-modeli. Izmantosim to informāciju, kāda par katru projektu ir pieejama *ExperiencePro* datu bāzē. Klasificēsim *ExperiencePro* datu bāzē esošo informāciju saskaņā ar MERIME metodes objektu tipiem (skat. 21. tabula).

21. tabula. MERIME informācijas bāzes modelis *ExperiencePro* datiem

Objekta tips	Objekta atribūts	Skaidrojums
Projekts	ID	Projekta identifikators
	Izmantotie CASE rīki	Projekta izstrādes procesā izmantotie CASE rīki
	Izmantotā metodoloģija	Projekta izstrādes procesā izmantotā metodoloģija
	Izmantotie projekta vadības rīki	Projekta izstrādes procesā izmantotie projekta vadības rīki
	Uzņēmējdarbības sfēra	Uzņēmējdarbības sfēra, kurai attiecīgā programmatūra radīta (ražošana, finansu vadība u.c.)
	Projekta tips	Izstrādes projekta tips (uzturēšana, jaunas programmatūras izstrāde u.c.)
	Izstrādes platforma	Programmatūras izstrādes platforma (lieldatori, personālie datori, datortīkls)
	Galvenā izstrādes vide	Izstrādes vide, kurā, galvenokārt, notiek izstrāde
Mērījums	Skat. M_1 līdz M_3 22. tabulā	
Risks	Skat. R_1 līdz R_{21} 22. tabulā	

2.5.1.2. MERIME inicializācija

MERIME metodes inicializācijas laikā definēsim turpmāk nosauktos lielumus.

1. Mērījumu un risku novērtēšanas laika periodu kopu T.
2. Mērījumu kopu M, kura sastāv no trim kopām M_1 līdz M_3 un attiecīgo mērījumu pieļaujamās novirzes. Attiecīgā mērījuma novirzes lielums noteikts saskaņā ar rezultātiem, kas iegūti mērījumu pieļaujamo robežu noteikšanas gaitā (skat. nodaļu "2.4.1.5. Mērījumu pieļaujamo robežu noteikšana").
3. Risku kopu R, kura sastāv no R_1 līdz R_{21}. Risku novērtējumu skala izvēlēta saskaņā ar *ExperiencePro* izmantoto risku novērtēšanas skalu.

Saskaņā ar *ExperiencePro* metodiku katru risku novērtē ar skaitli no 1 līdz 5, kas raksturo attiecīgo draudu. Vērtība 1 nozīmē, ka drauds negatīvi ietekmējis programmatūras izstrādes procesu, bet vērtība 5, ka attiecīgais drauds nav piemērojams šim projektam. Tā kā šie riski katram projektam tiek vērtēti pēc projekta beigām, tad nav jāvērtē varbūtība, ar kādu attiecīgais drauds realizēsies. Ir zināms, ka attiecīgais drauds ir ietekmējis izstrādes procesu (drauda realizācijas varbūtība ir 100%), skaitļi no 1 līdz 5 izsaka drauda pakāpi.

22. tabula. MERIME inicializācija ExperiencePro datu izmantošanai

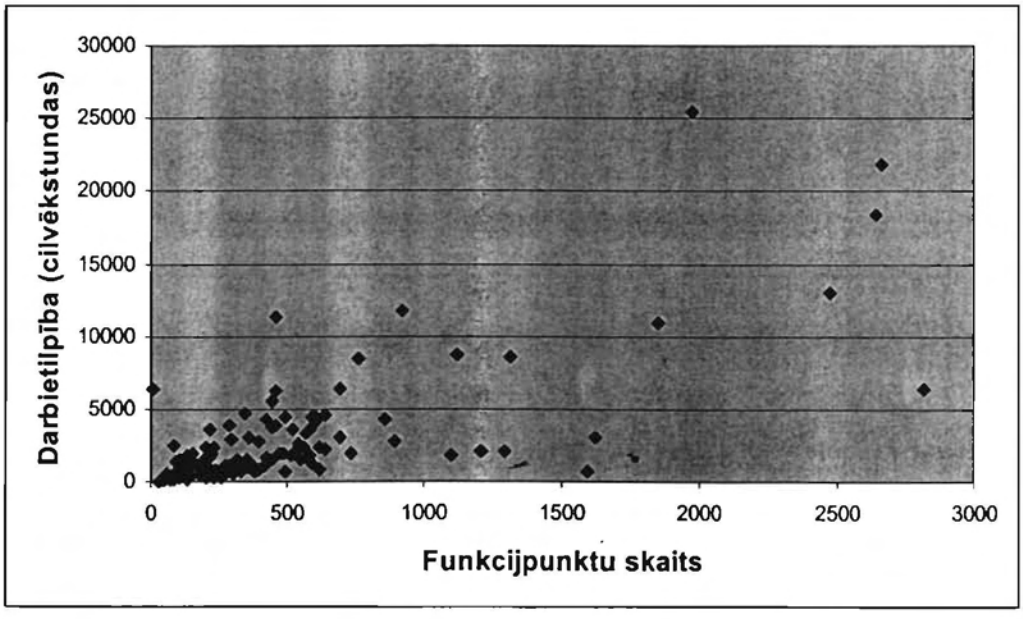
Apzīmējums	Nosaukums	Skaidrojums
T	**Laiks**	
$t_i,\ i = \overline{1,6}, i \in N$	Laika periodi	Laika periodi 1 mēnesis, 2 mēneši utt.
M	**Mērījumi**	
M_1	Ilgums	Projekta ilgums mēnešos
ε_1^M	Ilguma novirze	± 5%
M_2	Izmērs	Projekta izmērs funkcijpunktos
ε_2^M	Izmēra novirze	± 20%
M_3	Darbietilpība	Programmatūras izstrādātāja patērētā darbietilpība cilvēkstundās
ε_3^M	Darbietilpības novirze	± 20%
R	**Riski**	**Šeit norādīta tikai attiecīgā riska zemākā un augstākā robežas, pilna *ExperiencePro* risku pieejama [EXP02]**
R_1	Pasūtītāja pārstāvju iesaistīšana	1 : pasūtītājam nav laika piedalīties prasību specificēšanā, paredzamais tirgus nav izprotams, 5 : pasūtītāja līdzdalība ir aktīva, visas prasības ir definētas un kontrolētas, produkta tirgus ir pilnībā saprasts
R_2	Izstrādes vides atbalsts projekta izstrādei	Vai izstrādei izmantojamie rīki, platformas u.c. atbalsta projekta izstrādi. 1 : bieži sastopama izstrādes vides nepietiekamība, 5 : pilnīgi pietiekama izstrādes vide, tiek uzturēta vienīgi šī projekta vajadzībām
R_3	IT personāla pieejamība	Vai pasūtītāja IT personāls ir bijis pieejams izstrādes procesa laikā ar to saistīto problēmu risināšanai. 1 : IT personāls nav pieejams, ir vairāki paralēli izstrādes projekti, 5 : kvalificētie izstrādātāji pieejami, kad tas nepieciešams, viņi strādā tikai šajā projektā
R_4	Ieinteresēto pušu skaits	Cik organizācijas vai citi paralēli projekti tikuši iesaistīti izstrādes procesa gaitā. 1 : projekta izstrādē iesaistītas vairāk kā trīs struktūrvienības, paralēli tiek izstrādāti vairāk kā 4 projekti, 5 : projekta izstrādē iesaistīta 1 struktūrvienība, kas strādā tikai šī projekta izpildei
R_5	Izstrādei nepieciešamā laika pieejamība	Vai izstrādei atvēlētais laika periods bijis pietiekams. 1 : izstrādei nepieciešamais laiks ir mazāk nekā divkārt, 5 : izstrādei nepieciešamais laiks ir reālistisks, 50% no nepieciešamā
R_6	Izstrādes standartu ietekme	Izstrādes procesā izmantoto standartu pieejamība un to kvalitāte. 1 : standartiem un labajai praksei jāseko stingri, bet tie ir nestabili un mainās projekta izstrādes gaitā, 5 : detalizēti un stabili standarti, kuri pazīstami visiem izstrādātājiem
R_7	Izstrādes metožu ietekme	Izstrādes procesā izmantoto programminženierijas metožu izmantošana, izmantoto metožu kvalitāte. 1 : programminženierijas metodes netiek izmantotas, 5 : izmanto visā izstrādes dzīves cikla laikā, metodes ir speciāli pielāgotas projekta vajadzībām
R_8	Izstrādes rīku ietekme	Izstrādes procesā izmantoto programmrīku izmantošana, šo rīku kvalitāte. 1 : minimāls rīku atbalsts – teksta redaktori, kompilators un vienkārša atkļūdošana, 5 : integrēta vide produkta izstrādei visa izstrādes dzīves cikla uzturēšanai
R_9	Izmaiņu vadības līmenis	Programmatūras prasību stabilitāte, spēja kontrolēt šīs prasības. 1 : nepārtraukta izmaiņu straume, nav izstrādes līguma, ~30% nākušas klāt, salīdzinot ar sākotnējām prasībām, 5 : Nav jaunas prasības
R_{10}	Izstrādes procesa	Izstrādes procesa stabilitāte. 1 : prasības izstrādes procesu atgriež iepriekšējās fāzēs vairākkārt. Nav iespēju situāciju labot,

Apzīmējums	Nosaukums	Skaidrojums
	brieduma līmenis	nesaskan ar kvalitātes sistēmu, 5 : progresējošs izstrādes process, nav sagaidāmi defekti vai integritātes problēmas. Stipra atbilstība kvalitātes modeļiem
R_{11}	Funkcionalitātes prasības	Cik sarežģītas ir izstrādājamā produkta funkcionālās prasības. 1 : funkcionāli un tehnoloģiski sarežģīts risinājums, daudz DBVS, daudz algoritmisku funkciju, 5 : nav lietotāja interfeisu, vienkārša datu apstrāde, funkcionalitātes pareizību grūti novērtēt
R_{12}	Uzticamības prasības	Cik uzticamam jābūt izstrādātajam produktam. 1 : darbības pārtraukumi var ietekmēt cilvēku dzīvības, vai radīt ekoloģiskas katastrofas, datus nedrīkst zaudēt nekādā gadījumā, 5 : nepieciešama periodiska darbināšana, tas, ka sistēma nestrādā, netraucē uzņēmuma darbību
R_{13}	Lietojamības prasības	Cik ērti lietojamam jābūt izstrādātajam produktam. 1 : daudz lietotāju daudzās organizācijās ar dažādiem zināšanu līmeņiem, nepieciešama ievērojama pielāgošana, 5 : daži lietotāji vienā organizācijā, viegli iegūstamas izstrādātāju konsultācijas
R_{14}	Efektivitātes prasības	Kādi ir efektivitātes mērķi, kas izvirzīti produktam. 1 : efektivitāte ir kritiska, nepieciešamas speciālas zināšanas, lai šo efektivitāti nodrošinātu, 5 : nav efektivitātes prasības, kuras būtu īpaši jāplāno
R_{15}	Uzturamības prasības	Cik viegli uzturamam jābūt izstrādātajam produktam. 1 : liela, stratēģiska IS mainīgā uzņēmējdarbības vidē, jācenšas nodrošināt nepārtraukta sistēmas darbība, 5 : palīgprogrammatūra, kura nav saistīta ar izmaiņām likumdošanā vai uzņēmējdarbībā
R_{16}	Pārnesamības prasības	Cik viegli pārnesamam no vienas vides citā jābūt izstrādātajam produktam. 1 : lietotāji atrodas dažādās organizācijās, izmanto dažādas platformas, neregulāra atjaunošana, 5 : programmatūrai jāstrādā uz noteiktas platformas, atjaunošanas process ir vadāms
R_{17}	Izstrādātāju sistēmanalīzes spējas	1 : nav pieredzes prasību analīzē un līdzīgos projektos, 5 : izstrādātāju komanda sastāv no augstas klases profesionāļiem
R_{18}	Izstrādātāju zināšanas problēmapgabalā	1 : izstrādātāju pieredze problēmapgabalā ir mazāka par 6 mēnešiem, 5 : pieredze problēmapgabalā ir laba gan izstrādātāja, gan pasūtītāja pusē, izprotams viss bizness kopumā
R_{19}	Izstrādātāju zināšanas par izstrādes rīkiem	1 : izstrādātājiem nav iepriekšēju zināšanu darbā ar izstrādes rīkiem, vidējā pieredze ir mazāka par 3 mēnešiem, 5 : izstrādātāji labi zina izstrādes rīkus, pieejamas konsultācijas par projektam specifiskām vajadzībām, vidējā pieredze ir vairāk kā 3 gadi
R_{20}	Projekta vadītāja pieredze	1 : projekta vadītājam nav iepriekšējas pieredzes līdzīgos projektos, 5 : projekta vadītājs ir īsts profesionālis dažādos programmatūras izstrādes projektos
R_{21}	Izstrādātāju spēja strādāt komandā	1 : juceklīga komanda, minimālas zināšanas par darbu komandā, 5 : labi attīstīts komandas gars, komanda spēj atrisināt visas problēmas
S		1

2.5.1.3. Informācijas uzkrāšana

MERIME metodes informācijas uzkrāšanas solis šajā gadījumā faktiski ir inicializētās MERIME informācijas bāzes aizpildīšana ar *ExperiencePro* datiem. Šeit raksturosim datus, kādi ir *ExperiencePro* datu bāzē (pēc informācijas aizpildīšanas arī MERIME informācijas bāzē).

ExperiencePro datu bāzē esošo projektu izstrādes darba produktivitāte parādīta 20. attēlā. Redzams, ka pastāv sakarība starp izstrādājamās programmatūras apjomu un realizācijai nepieciešamo darbietilpību (korelācijas koeficients 0.74).



20. attēls. *ExperiencePro* datu bāzē esošo programmatūras izstrādes projektu funkcionalitātes apjoma un izstrādes darbietilpības saistība

ExperiencePro datu bāzē ir informācija gan par ļoti maziem projektiem, kuru funkcionalitātes apjoms ir 6 funkcijpunkti, gan arī lieliem (apjoms 2816 funkcijpunkti). Vidējais projektu funkcionalitātes apjoms ir 492 funkcijpunkti (funkcionalitātes apjomu u.c. mērījumu vērtības dotas 23. tabulā).

23. tabula ExperiencePro datu bāzē esošo jaunas programmatūras izstrādes projektu mērījumi

	Vidējā vērtība	Vidējā kvadrātiskā novirze	Minimālā vērtība	Maksimālā vērtība
Izmērs (funkcijpunkti)	492	550	6	2816
Darbietilpība (cilvēkstundas)	2878	4022	55	25397
Ilgums (mēneši)	11	9	0	55

2.5.2. Plānojamo projektu kopas P^{Pl} izvēle

Nākamais MERIME metodes validācijas solis pēc informācijas bāzes inicializācijas un aizpildīšanas ir reprezentatīvas plānojamo projektu kopas P^{Pl} izvēle.

Par plānojamo projektu kopu P^{Pl} izvēlēsimies visus *ExperiencePro* datu bāzē esošos programmatūras izstrādes projektus. Tādējādi katrs plānojamo projektu kopā P^{Pl} iekļautais projekts atbilst šādām prasībām.

1. Tas ir reāls programmatūras izstrādes projekts, kura izstrāde pabeigta sekmīgi, t.i. izstrādes procesa rezultātā ir radīta programmatūra, kura atbilst pasūtītāja noteiktajām prasībām.
2. Mērījumu un risku pārvaldības informācija savākta, izmantojot noteiktu mērījumu datu un risku analīzes rezultātu vākšanas metodiku, kādu nodrošina rīkā *ExperiencePro* iebūvētā metodika.

2.5.3. MERIME metodes izmantošanas gaita

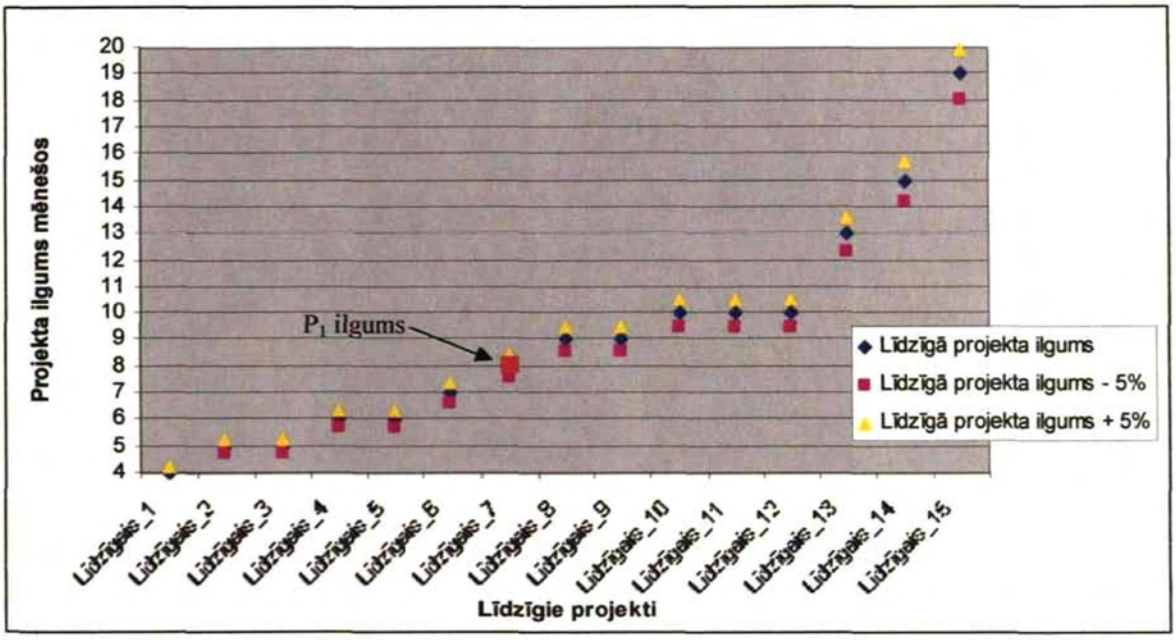
Šajā nodaļā parādīts, kā validēta MERIME metode katram projektam no plānojamo projektu kopas P^{Pl}, par MERIME informācijas bāzi izvēloties *ExperiencePro* datu bāzi.

1. Izvēlas projektu P_i no plānojamo projektu kopas P^{Pl}.
2. Izņem projektu P_i no MERIME informācijas bāzes.
3. Projektam P_i izvēlas visus iespējamos risku un mērījumu pārus.

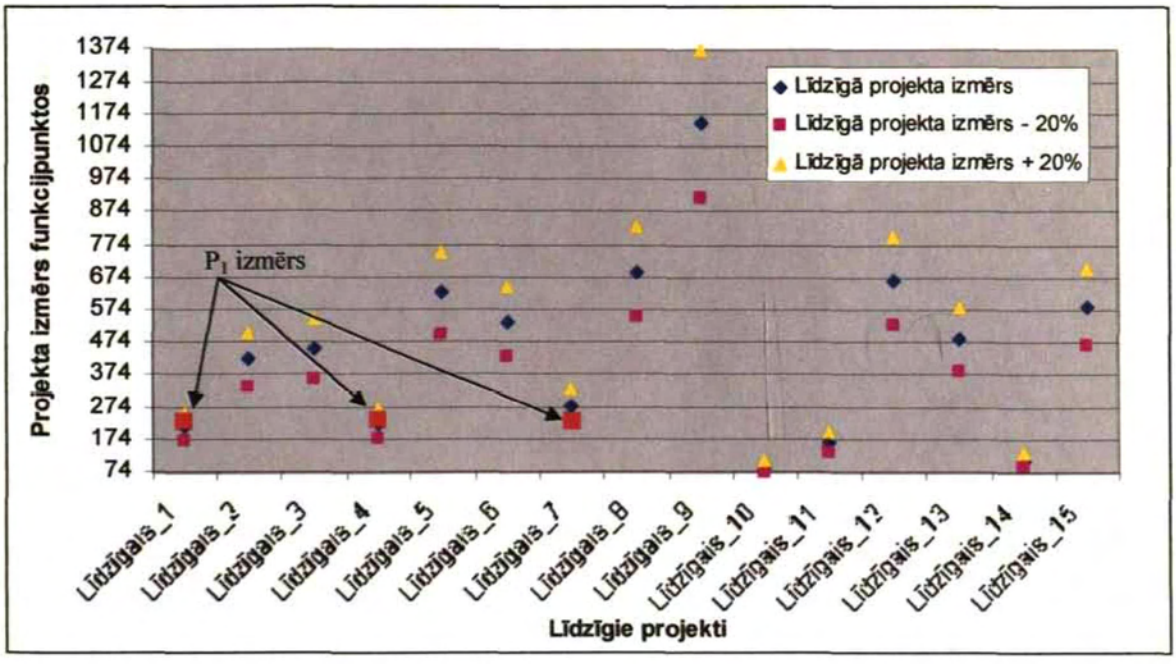
Saskaņā ar MERIME terminoloģiju, tiek veikta situācijas analīze programmatūras izstrādes projektā. Katram projektam iespējamas 63 dažādas situācijas (dažādi mērījuma un riska pāri (3 mērījumi un 21 risks)). Piemēram, plānojamam projektam P_1 mērījuma "Darbietilpība" un riska "Pasūtītāja pārstāvju iesaistīšana" situācija Sit_1 ir (1520,2), kur 1520 ir projekta izstrādes darbietilpība stundās (skat. mērījumu M_3 22. tabulā), bet 2 riska "Pasūtītāju pārstāvju iesaistīšana" vērtība.

4. Katram plānojamā projekta P_i riska un mērījuma pārim meklē līdzīgos projektus MERIME informācijas bāzē saskaņā ar MERIME programmatūras izstrādes projektu līdzības definīciju (skat. nodaļu "2.3.4.3. Līdzīgu projektu meklēšana").
5. Visiem P_i attiecīgajā situācijā līdzīgajiem projektiem, kas atrasti MERIME informācijas bāzē, aplūko atlikušo divu mērījumu vērtības un šo vērtību pieļaujamās ϵ robežas.

Piemēram, projektam P_1 no plānojamo projektu kopas P^{Pl} situācijā Sit_1 MERIME informācijas bāzē ir atrasti 15 līdzīgi projekti. Līdzīgo projektu mērījumi "Ilgums" un "Izmērs" (skat. mērījumus M_1 un M_2 22. tabulā) un to pieļaujamās novirzes attēlotas attiecīgi 21. un 22. attēlos.



21. attēls. Plānojamam projektam P_1 situācijā Sit_1 MERIME informācijas bāzē atrasto līdzīgo projektu ilgumi un to pieļaujamās ε novirzes



22. attēls. Plānojamam projektam P_1 situācijā Sit_1 MERIME informācijas bāzē atrasto līdzīgo projektu izmēri un to pieļaujamās ε novirzes

6. Līdzīgo projektu meklēšanu attiecīgajā situācijā uzskata par veiksmīgu, ja starp plānojamam projektam P_i līdzīgajiem projektiem, eksistē kaut viens projekts, kuram P_i mērījuma vērtība ir kāda no līdzīgo projektu mērījuma pieļaujamām ϵ robežām.

Piemēram, projektam P_1 no plānojamo projektu kopas P^{Pl} situācijā Sit_1 līdzīgo projektu meklēšana ir bijusi veiksmīga, jo starp 15 līdzīgajiem projektiem, P_1 mērījums "Ilgums" ir viena līdzīgā projekta ilguma robežās (skat. 21. attēls) un mērījums "Izmērs" ir triju līdzīgo projektu izmēru robežās (skat. 22. attēls).

2.5.4. MERIME metodes rezultāts

MERIME metodes validācijas rezultāts, ja par MERIME informācijas bāzi izmanto *ExperiencePro* datu bāzi, parādīts 24. tabulā. Šajā tabulā redzams, cik procentiem projektu no plānojamo projektu kopas P^{Pl} ir veiksmīgi atrasti līdzīgie projekti MERIME informācijas bāzē attiecīgajā situācijā.

Piemēram, katram plānotajam projektam no kopas P^{Pl}, fiksējot risku R_1 un mērījumu M_3 (situācija saskaņā ar MERIME metodes definīciju), mērījums "Ilgums" (M_1) ir vismaz viena līdzīgo projektu ilguma ϵ robežās 74,53% plānoto projektu, bet mērījums "Izmērs" (M_2) ir vismaz viena līdzīgo projektu ilguma ϵ robežās 71,43% plānoto projektu.

Fiksējot vienu mērījumu un vienu risku, pārējie iespējamie mērījumi nonāk vismaz viena līdzīgā projekta mērījuma ϵ robežās vidēji 68% plānoto projektu. Ir atrodami avoti [BAI99], kur šāda precizitāte tiek uzskatīta par pietiekamu. Informācija par dažādu programmatūras izstrādes darbietilpības prognozēšanas metožu doto prognožu precizitāti nav publiski pieejama, jo šīs metodes izmanto dažādos komerciālos rīkos, kas ir to izstrādes uzņēmumu īpašums (skat. nodaļu "1.3. Formālu metožu izmantošana izstrādes procesa prognozēšanai").

24. tabula. Plānojamam projektam veiksmīgi atrasto līdzīgo projektu skaits procentos (cik % no plānojamo projektu kopas ir atrasti līdzīgie projekti MERIME informācijas bāzē attiecīgajā situācijā).

Fiksētais risks	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}	R_{13}	R_{14}	R_{15}	R_{16}	R_{17}	R_{18}	R_{19}	R_{20}	R_{21}	Vidēji
Fiksēts mērījums "Darbietilpība" (M_3)																						
Veiksmīgi atrasts ilgums (M_1)	74.53	81.99	78.88	74.21	80.86	80.86	83.85	82.10	79.66	80.30	80.09	80.24	81.00	76.54	80.47	80.05	79.80	79.81	79.75	73.20	78.83	**79.38**
Veiksmīgi atrasts izmērs (M_2)	71.43	74.53	73.29	74.21	75.93	73.46	76.40	75.31	71.43	72.56	73.29	74.21	75.93	78.46	78.40	75.31	74.99	75.05	74.98	75.10	75.00	**74.73**
Veiksmīgi atrasti abi mērījumi (M_1 un M_2)	60.25	64.60	60.87	57.86	63.58	61.73	65.84	64.81	62.44	62.72	60.25	64.60	60.87	57.86	63.58	61.73	65.84	64.81	62.10	62.45	62.46	**62.44**
Fiksēts mērījums "Izmērs" (M_2)																						
Veiksmīgi atrasts ilgums (M_1)	74.07	78.40	74.07	73.46	78.40	75.93	79.91	76.27	77.93	78.13	77.12	79.29	76.76	78.85	77.53	78.01	78.48	77.34	78.76	77.40	78.42	**77.36**
Veiksmīgi atrasta darbietilpība (M_3)	69.75	75.31	70.99	70.37	77.16	75.93	65.78	74.41	80.76	66.63	66.75	86.03	72.64	56.65	83.84	87.80	49.25	69.33	42.63	49.01	67.56	**69.46**
Veiksmīgi atrasti abi mērījumi (M_1 un M_3)	58.02	64.81	59.88	58.02	66.05	63.58	58.25	63.05	63.11	59.74	62.74	63.34	60.10	61.91	63.17	60.93	61.63	62.90	61.29	61.48	62.66	**61.75**
Fiksēts mērījums "Ilgums" (M_1)																						
Veiksmīgi atrasts izmērs (M_2)	50.64	51.25	51.16	51.06	50.32	51.26	51.27	51.53	51.79	51.74	50.64	50.00	46.15	53.85	53.85	54.19	53.80	50.65	51.74	51.84	51.74	**51.45**
Veiksmīgi atrasta darbietilpība (M_3)	49.36	67.60	46.90	54.46	44.57	67.60	67.60	67.60	67.60	67.60	53.85	48.68	48.95	53.21	51.28	52.90	54.49	48.70	51.43	46.90	54.46	**52.90**
Veiksmīgi atrasti abi mērījumi (M_2 un M_3)	30.77	14.56	4.11	24.95	74.86	14.56	4.11	24.95	74.86	74.86	33.33	30.26	27.97	35.26	33.33	32.90	35.26	30.52	14.56	4.11	24.95	**30.72**

2.6. MERIME izmantošanas pieredze

2.6.1. MERIME metodes izmantošana pabeigtiem projektiem

Lai ilustrētu MERIME metodes izmantošanu, aplūkosim astoņus pabeigtus programmatūras izstrādes projektus (PRJ_1 līdz PRJ_8), kuri jau tika izmantoti dažādu formālu darbietilpības metožu doto rezultātu ilustrācijai. Šie astoņi ir reāli programmatūras izstrādes projekti, kuru rezultātā ir radīta programmatūra, kas atbilst pasūtītāja prasībām. Par šiem projektiem pieejamā informācija, tai skaitā informācija par izstrādes procesu ietekmējošiem riskiem dota 14. tabulā. Iemesli, kāpēc izvēlēti šie projekti ir doti nodaļā "1.3. Formālu metožu izmantošana izstrādes procesa prognozēšanai".

Turpmāk izmantosim MERIME metodi, lai prognozētu projektu PRJ_1 līdz PRJ_8 darbietilpību.

2.6.1.1. MERIME metodes inicializācija un informācijas uzkrāšana

Par MERIME informācijas bāzi izvēlēsimies projektu pārvaldības metodes un rīka ExperiencePro izstrādei izmantotos datus [EXP02] (skat. nodaļu "1.3.2.2. ExperiencePro"). Tādējādi MERIME metodes inicializācijas un informācijas uzkrāšanas soļi ir aprakstīti nodaļas "2.5. MERIME validācija" apakšnodaļā "2.5.1. Informācijas bāze".

2.6.1.2. Situācijas analīze

Situācijas analīzes solī analizē katra projekta izstrādes procesa situāciju. Izmantosim to informāciju, kura ir pieejama par katru no 8 programmatūras izstrādes projektiem (skat. 14. tabula).

Lai izvēlētajiem 8 projektiem meklētu līdzīgos projektus MERIME informācijas bāzē, informācija kas pieejama par šiem projektiem, jāizsaka MERIME bāzes terminos. Proti, jāvienādo mērījumu vienības un jāsaskaņo riski un to vērtības.

1. Ir pieejama informācija par radītās programmatūras apjomu (funkcijpunktos). Šis mērījums atbilst mērījumam "Projekta izmērs" (M_2) MERIME informācijas bāzē.
2. Katrā programmatūras projektā pieejamo informāciju par riskiem (skat. 14. tabula) projicē uz MERIME informācijas bāzē izmantotajiem riskiem. Projicēšanā vadās pēc šādiem principiem.
   2.1. Risku skaidrojumi projekta kontekstā un MERIME risku informācijas uzkrāšanas metodikā ir līdzīgi.
   2.2. Riska vērtība jāpārveido atbilstoši MERIME metodes risku novērtēšanas skalai.

25. tabulā parādīts, kā, ievērojot iepriekš minētos principus, projicēta par projektu pieejamā informācija. Vienādi burti rindā "Saistība" parāda attiecīgā riska informāciju projektam un šī riska projekciju MERIME terminos. Tāpat riska vērtēšanas skala ir atšķirīga: MERIME informācijas bāzē izmantotā ir spoguļattēls no sākotnēji izmantotās. To MERIME risku, par kuriem nav informācijas, kolonnas ir baltas.

25. tabula. Situācijas modelēšana saskaņā ar MERIME metodi astoņiem izvēlētiem programmatūras izstrādes projektiem

	Par projektu pieejamā informācija (skat. 14. tabula)													MERIME riski																				
		Pasūtītāja riski		Prasību riski		Projekta vadības riski		Izstrādātāju riski						Projekta riski					Procesa riski					Produkta riski						Izstrādātāju riski				
Projekts	Programmatūras apjoms funkcijpunktos	Pasūtītāja nepietiekams tehniskais nodrošinājums	Apgrūtināta sadarbība ar pasūtītāju	Nekvalitatīvi definētas programmatūras prasības	Programmatūras prasību nestabilitāte	Neatbilstošs projekta plānojums	Vāja projekta vadība	Izstrādes vides kļūdas	Izstrādātāju motivācijas trūkums	Izstrādātāju nepietiekams tehniskais nodrošinājums	Izstrādātāju kadru mainība	Izstrādātāju nepietiekamas zināšanas par izstrādes tehnoloģijām, vidi	Apgrūtināta izstrādātāju sadarbība	Pasūtītāja pārstāvju iesaistīšana	Izstrādes vides atbalsts projekta izstrādei	IT personāla pieejamība	Ieinteresēto pušu skaits	Izstrādei nepieciešamā laika pieejamība	Izstrādes standartu ietekme	Izstrādes metožu ietekme	Izstrādes rīku ietekme	Izmaiņu vadības līmenis	Izstrādes procesa brieduma līmenis	Funkcionalitātes prasības	Uzticamības prasības	Lietojamības prasības	Efektivitātes prasības	Uzturamības prasības	Pārnesamības prasības	Izstrādātāju sistēmanalīzes spējas	Izstrādātāju zināšanas problēmapgabalā	Izstrādātāju zināšanas par izstrādes rīkiem	Projekta vadītāja pieredze	Izstrādātāju spēja strādāt komandā
PRJ_1	131	2	2	2	3	4	2	3	1	2	1	4	1			5	4	4			4	3		4								2	4	5
PRJ_2	1680	1	2	4	3	3	2	1	2	1	1	1	1			5	4	4			5	3		2								5	4	5
PRJ_3	321	1	1	4	2	1	2	1	3	1	3	4	1			3	5	4			5	4		2								2	4	5
PRJ_4	104	1	1	2	1	2	1	1	1	1	1	1	1			5	5	5			5	5		4								5	5	5
PRJ_5	640	1	1	2	1	2	1	1	1	1	1	1	1			5	5	5			5	5		4								5	5	5
PRJ_6	156	1	1	2	1	2	1	1	1	1	1	1	1			5	5	5			5	5		4								5	5	5
PRJ_7	1113	3	2	4	3	2	2	3	2	2	3	2	3			3	4	4			4	3		2								4	4	3
PRJ_8	971	2	3	2	2	2	3	2	3	2	2	2	2			4	3	3			4	4		4								4	3	4
Saistība			A	B	C	D	E			F	G	H	I			G	A	D			F	C		B								H	E	I

2.6.1.3. Mērījumu vērtību prognozēšana

Mērījumu vērtību prognozēšanas solī meklēsim astoņiem programmatūras izstrādes projektiem iepriekš aprakstītajās situācijās (skat. 25. tabula) līdzīgus projektus MERIME informācijas bāzē un aplūkosim atrasto līdzīgo projektu darbietilpību.

Šeit līdzīgos projektus meklēsim pēc mērījuma "projekta izmērs" un visiem tiem riskiem, kuru vērtības mums ir pieejamas (skat. 25. tabula). Vairākus riskus izmantosim tādēļ, lai maksimāli ņemtu vērā to informāciju, kura mums ir pieejama, nevis izvēlēsimies vienu risku, kas būtu pietiekami saskaņā ar MERIME definīciju.

MERIME informācijas bāzē atrasto līdzīgo projektu darbietilpība parādīta 26. tabulā. Piemēram, projektam PRJ_3 ir atrasti septiņi līdzīgi projekti, ir pieejama šo septiņu līdzīgo projektu darbietilpība. Tabulā redzams, ka prognozētā pret reālo darbietilpību vienam no līdzīgajiem projektiem ir 95% (100% ir ideāla sakritība), bet citiem līdzīgajiem projektiem svārstās no 38% līdz 92%. Neviens no līdzīgajiem projektiem darbietilpību nepārvērtē.

Vislielākā prognozes kļūda ir PRJ_7 (vidēji 17%, jeb darbietilpība novērtēta par zemu seškārt). Šāda pārvērtēšana vai novērtēšana par zemu vairākkārt var būt, ja attiecīgie projekti ir netipiski [ISB00].

26. tabula. MERIME metodes prognozētā darbietilpība astoņiem izvēlētiem programmatūras izstrādes projektiem

Projekta ID	Reālā darbietilpība (cilvēkdienas)	Līdzīgā projekta 1 darbietilpība (prognoze, cilvēkdienas)	Līdzīgā projekta 2 darbietilpība (prognoze, cilvēkdienas)	Līdzīgā projekta 3 darbietilpība (prognoze, cilvēkdienas)	Līdzīgā projekta 4 darbietilpība (prognoze, cilvēkdienas)	Līdzīgā projekta 5 darbietilpība (prognoze, cilvēkdienas)	Līdzīgā projekta 6 darbietilpība (prognoze, cilvēkdienas)	Līdzīgā projekta 7 darbietilpība (prognoze, cilvēkdienas)	Prognoze/Reālo darbietilpību līdzīgam projektam 1	Prognoze/Reālo darbietilpību līdzīgam projektam 2	Prognoze/Reālo darbietilpību līdzīgam projektam 3	Prognoze/Reālo darbietilpību līdzīgam projektam 4	Prognoze/Reālo darbietilpību līdzīgam projektam 5	Prognoze/Reālo darbietilpību līdzīgam projektam 6	Prognoze/Reālo darbietilpību līdzīgam projektam 7	Vidēji prognoze/reālo darbietilpību
PRJ_1	231	47	193						20%	84%						52%
PRJ_2	2166	3175	83	1360	382				147%	4%	63%	18%				58%
PRJ_3	200	190	77	117	121	184	174	124	95%	38%	58%	60%	92%	87%	62%	70%
PRJ_4	144	182	84	40	47	138	148		126%	58%	27%	32%	96%	103%		74%
PRJ_5	144	792	558	545	375	227	445		550%	387%	378%	260%	158%	309%		341%
PRJ_6	96	193							201%							201%
PRJ_7	4020	223	1469	349					6%	37%	9%					17%
PRJ_8	1777	349	535	236					20%	30%	13%					21%

2.6.1.4. Secinājumi

Aplūkojot MERIME metodes dotos rezultātus astoņiem izvēlētiem programmatūras izstrādes projektiem, jāsecina sekojošais.

1. Rezultāts ir tieši atkarīgs no tiem datiem, kurus izmanto prognozēšanai. Šī ir analoģiju bāzētu programmatūras izstrādes darbietilpības prognozēšanas metožu īpatnība.

Proti, ja plānojamais projekts ir "nestandarta", tam nebūs iespējams atrast līdzīgus projektus vai arī rezultāti nebūs derīgi šim "nestandarta" projektam. Un otrādi, ja informācijas bāzē pieejamie projekti ir "nestandarta", tad arī iegūtais rezultāts pavisam parastam projektam būs nepiemērots.

Tai pat laikā šī ir arī MERIME metodes priekšrocība, jo iespējams izveidot informācijas bāzi, kurā ir tieši uzņēmumam raksturīgu projektu izstrādes informācija.

2. Dažādu mērījumu vērtību prognozēšanai MERIME metode ļauj izmantot tik daudz informācijas, cik ir pieejams par plānojamo projektu.

Šī iespēja ir svarīga priekšrocība MERIME metodes praktiskai izmantošanai, jo informācija, kāda ir pieejama par plānojamo projektu, ir ļoti dažāda katrā situācijā. MERIME metode uzliek tikai minimālos nosacījumus tam, kādai informācijai jābūt pieejamai (viens risks un viens mērījums). Šajā nodaļā aplūkotajā piemērā bija pieejama informācija par vienu mērījumu (izmēru) un 8 riskiem.

2.6.2. MERIME - ALFA informācijas bāze

Šajā nodaļā parādīta MERIME metodes izmantošana vienā programmatūras izstrādes uzņēmumā.

Uzņēmums ALFA ir vidējs uzņēmums, kas specializējies lielu un sarežģītu specializētās (custom-built) programmatūras projektu izstrādē. Šādos projektos īpaši svarīga ir prasme organizēt darbu, māka koordinēt un saskaņot attiecības starp dažādām projektā iesaistītajām pusēm, izstrādes procesu rūpīgi plānojot. Uzņēmums ALFA piedāvā lielu projektu sistēmanalīzi, programmatūras projektu realizāciju, integrāciju, neinženierijas projektus, pārstrādājot jau

gatavas sistēmas, un programmatūras uzturēšanu lielās informācijas sistēmās. Uzņēmums strādā vairākus gadus.

Nosacījumi, kādēļ uzņēmumā ALFA bija nepieciešama metodoloģija plānošanas atbalstam, risku analīzei un kontrolei, nosaukti turpmāk.

1. Pieredze par projektā izmantotajiem tehniskajiem risinājumiem, projekta pārvaldību u.c. ir koncentrējusies pie atsevišķo projektu izpildītājiem un šo projektu vadītājiem. Informācijas apmaiņa starp dažādiem projektiem ir apgrūtināta, jo projekti ir lieli un ilgstoši (gads un vairāk), projektu izstrādātāji ļoti iedziļinās konkrētā pasūtījuma problēmapgabalā un izstrādes tehnoloģijā, informācijas apmaiņa dabiskā ceļā, mainoties izstrādātāju komandai, ir apgrūtināta. Nepieciešams mehānisms, kā visiem projektiem kopīgo informāciju savākt un darīt pieejamu citu projektu izstrādātājiem.
2. Apgrūtināta projektu salīdzināšana. Nepieciešams katru projektu varēt salīdzināt iesākumā ar citiem projektiem tai pat uzņēmumā, vēlāk ar citiem projektiem attiecīgajā uzņēmējdarbības sfērā. Katrs projekts ir individuāls, tādēļ grūti atrast kritērijus, pēc kādiem izvērtēt, vai projekts ir "labs" (produktīvs, izvirzītajām prasībām atbilstošs u.c.). vai "slikts".
3. Apgrūtināta uzkrātās informācijas izmantošana. Informācijas apmaiņa prasa no izstrādātājiem papildus darbu datu savākšanai un kvalitātes kontrolei, tādēļ viņu motivācijai izstrādes gaitā nemitīgi jārāda, ka savāktie dati tiek pārstrādāti praktiski izmantojamā informācijā, kas noderīga plānošanas, risku pārvaldības u.c. projekta pārvaldības funkciju veikšanai.
4. Sarežģīta projektu darbietilpības un izstrādei nepieciešamā laika prognozēšana. Projekti ir ilgstoši, tādēļ katram projekta pārvaldniekam individuāli neveidojas pieredze darbietilpības un izstrādei nepieciešamā laika prognozēšanai (viņa/viņas praksē ir apmēram 3 – 4 projekti). Jāizmanto dati par citiem projektiem.
5. Individuāla projekta pieredze var nebūt izmantojama nevienā citā projektā. Katrs uzņēmumā ALFA izstrādātais projekts ir atšķirīgs no citiem, ko nosaka uzņēmuma izvēlētā uzņēmējdarbības niša. Tādēļ, uzkrājot projekta pieredzi, jārēķinās, ka pilnībā tā var nebūt derīga izmantošanai, tādēļ jāparedz iespēja izmantot fragmentāru informāciju.

Procesi, kuros izmanto MERIME metodi uzņēmumā ALFA, nosaukti turpmāk.

1. Izstrādei nepieciešamās darbietilpības un laika prognozēšanai pirms projekta uzsākšanas.
2. Programmatūras izstrāde projektu gaitas operatīvajai plānošanai, izmantojot risku analīzes un izstrādes procesa mērīšanas gaitā uzkrāto informāciju.
3. Rīcības plāna sastādīšanai augsta riska situācijās.

Turpmāk aplūkota MERIME metodes izmantošana uzņēmumā ALFA, sekojot MERIME metodes ieviešanas rāmim (skat. 12. attēls). MERIME metodes inicializēšanai un datu uzkrāšanai, izmantota šajā darbā aprakstītā MERIME metodes izmantošanas sagatave (skat. nodaļu "2.4 MERIME izmantošanas sagatave").

2.6.2.1. Inicializācija

Uzņēmums ALFA strādā vairākus gadus, izstrādājot dažādus projektus dažādiem pasūtītājiem dažādās valstīs. Katrā atsevišķā projektā ir iedibināti plānošanas, risku pārvaldības un izstrādes procesa mērīšanas procesi. Šo procesu norises kārtību reglamentē vai nu pasūtītājs, ja pasūtītāja organizācijā ir reglamentēta programmatūras izstrādes projektu kārtība, vai arī nosaka paši projektu izstrādātāji vai projekta pārvaldnieks, vadoties no iepriekšējās pieredzes. Izstrādes procesa plānošanas, risku pārvaldības un mērīšanas procesu īpatnības, ar kādām bija jārēķinās MERIME metodes inicializācijas gaitā uzņēmumā ALFA, ir nosauktas turpmāk.

1. Galvenā interese plānošanas procesā koncentrējas uz darbietilpības (resursu piesaistes izstrādes procesam plāns), laika (kalendārais plāns) un produkta kvalitātes aspektiem.
2. Laika momenti, kad notiek izstrādes procesa plānošana, ir šādi.
   2.1. Projektu uzsākot plāno visam izstrādes procesam nepieciešamo darbietilpību un visam izstrādes procesam nepieciešamo laiku.
   2.2. Katra mēneša pirmajā nedēļā plāno izstrādes procesa darbietilpību nākamajam mēnesim kopumā, kā arī atsevišķiem darbu veidiem, piemēram, dokumentēšanai, testēšanai, testēšanas laikā atklāto problēmu

novēršanai utml. Šāda atsevišķu darbu darbietilpības operatīvā plānošana nepieciešama, jo šiem darbiem var piesaistīt darbiniekus no struktūrvienībām, kuras specializējas testēšanā, dokumentēšanā utml.

3. Risku pārvaldības process nav izdalīts kā atsevišķs process, izņemot tos projektus, kuros risku pārvaldības plāns un plāna izpildes uzraudzība ir pasūtītāja prasība. Faktiski risku pārvaldības process notiek katrā projektā. Par risku pārvaldības procesu ir atbildīgs projekta pārvaldnieks, kurš ņem vērā dažādus, viņaprāt, attiecīgajam projektam aktuālos riskus projekta izstrādes procesa plānošanas laikā.
4. Uzņēmumā ALFA veic šādus izstrādes procesa mērījumus.
   4.1. Projekta izstrādes dažādām aktivitātēm patērēto darba laiku.
   4.2. Informāciju par problēmziņojumiem, lai kontrolētu programmatūras kvalitāti.
   4.3. Informāciju par programmatūras izmaiņu pieprasījumiem, lai kontrolētu programmatūras prasību izmaiņas.
5. Mērījumu datu apstrādes kārtība un biežums ir individuāli katrā projektā un, galvenokārt, tiek veikti, lai mērījumu datus izmantotu operatīvajai plānošanai un atskaišu veidošanai programmatūras pasūtītājam.
6. Mērījumu datu vākšanas metodika un datu apstrādes vide ir katram projektam individuāla, vai nu pasūtītāja noteikta, vai tiek izvēlēta pēc projekta pārvaldnieka ieskatiem.

MERIME metodes inicializācijai izmantosim MERIME metodes izmantošanas sagatavi.

MERIME metodes inicializācijas solī galvenais uzdevums ir maksimāli pielāgoties uzņēmumā eksistējošai kārtībai plānošanā, risku pārvaldībā un izstrādes procesa mērīšanā. Tāpat jāievēro, ka neviendabīgā datu uzkrāšanas vide ir iedzimta izstrādes procesa īpatnība.

2.6.2.1.1. Risku un mērījumu izvērtēšanas biežums

Risku un mērījumu izvērtēšanas biežums uzņēmumā ALFA izvēlēts reizi mēnesī, katra mēneša pirmajā nedēļā, lai šo procesu saskaņotu ar uzņēmumā pieņemto kārtību.

Katra mēneša pirmā nedēļa ir laiks, kad projektu pārvaldnieki apkopo informāciju par iepriekšējā mēneša darbiem un plāno darbus jaunajam mēnesim. Šādi pielāgojot datu vākšanas laika momentus, iespējams samazināt personāla pretestību datu vākšanai.

2.6.2.1.2. RISKU PĀRVALDĪBA

Risku pārvaldības process uzņēmumā ALFA nav izdalīts kā atsevišķs process, ja vien pasūtītājs nepieprasa savādāk. Faktiski risku pārvaldība projektos notiek visu līmeņu projektu pārvaldības līmenī, riskus identificējot, izvērtējot un plānojot konkrētas aktivitātes risku apkarošanai projekta uzraudzības sanāksmju laikā. Identificētie riski, draudu realizācijas varbūtības un plānotās aktivitātes risku mazināšanai dokumentē šo sanāksmju protokolos.

Katra mēneša pirmajā nedēļā, konsultējoties ar attiecīgā projekta vadītāju, risku analīzes informāciju apkopo un ievieto MERIME informācijas bāzē. Apkopošanas gaitā par informācijas avotu izmanto projektu pārvaldības sanāksmju protokolus, identificētos riskus novērtējot pēc kvalitatīvās risku novērtēšanas metodes.

Uzņēmumā ALFA izvēlētā risku novērtēšanas metode ir atšķirīga no MERIME sagatavē ieteiktās. Lieliem projektiem ieteikts izvēlēties kvantitatīvo risku novērtēšanas skalu, savukārt, uzņēmumā ALFA izvēlēta kvalitatīvā, saskaņā ar 17. tabulā doto. Galvenais iemesls, kādēļ netika izvēlēta kvantitatīvā risku analīzes metode, ir par risku pārvaldību atbildīgo darbinieku (parasti projektu pārvaldnieku) grūtības novērtēt naudas izteiksmē katra drauda realizācijas gadījumu (S=3 saskaņā ar 17. tabulā doto risku novērtēšanas skalu).

Par pieļaujamo risku robežu (pieļaujamais riska slieksnis S MERIME metodē) uzskata situāciju, kad pastāv iespēja, ka attiecīgais risks ietekmēs izstrādes procesu negatīvi un ir jāpārdomā rīcības alternatīvas attiecīgā riska mazināšanai, taču nav lietderīgi ieguldīt resursus konkrētu rīcības plānu izstrādei.

Šāda risku analīzes informācijas uzkrāšanas pieeja nodrošina turpmāk nosaukto.

1. Risku analīzes informācijas savākšanai nav nepieciešams papildus darbs no projekta izstrādātāju puses.

2. Risku analīzes informācija MERIME informācijas bāzē nonāk apkopota pēc vienotas metodikas.

2.6.2.1.3. IZSTRĀDES PROCESA MĒRĪŠANA

Izstrādes procesa mērīšana uzņēmumā ALFA nav izdalīta kā atsevišķs process, ja pasūtītājs nepieprasa savādāk. Faktiski izstrādes process tiek mērīts katrā programmatūras izstrādes projektā, uzkrājot trejādus mērījumu datus.

1. Izstrādātā produkta apjoms.

Katrs programmatūras izstrādes projekts seko līdzi radītā produkta apjomam, to mērot funkcijpunktos, kur tas iespējams, vai programmrindiņās.

2. Izstrādes procesa darbietilpība.

Tajos programmatūras izstrādes projektos, kuros projekta pārvaldnieks to uzskata par lietderīgu, mēra izstrādes procesa darbietilpību.

3. Dati par problēmām un izmaiņu pieprasījumiem.

Datu uzkrāšana vienotā problēmu un izmaiņu reģistrā katram projektam ir obligāta kvalitātes sistēmas prasība. Tādējādi katrā programmatūras izstrādes projektā uzkrāj datus par produktu iekšējās testēšanas (vienībtestēšanas, integrācijas testēšanas) laikā atklātajām, kā arī pēc produkta nodošanas pasūtītājam atklātajām problēmām un izmaiņu pieprasījumiem.

Visi iepriekš minētie mērījumu dati tiek uzkrāti dažādās vidēs, pēc dažādām metodoloģijām. Šāda situācija ir objektīva, ievērojot uzņēmuma ALFA izvēlēto uzņēmējdarbības nišu. Lai nodrošinātu uzkrāto mērījumu apkopošanu un izmantošanu ne tikai viena projekta iekšienē, uzņēmumā ALFA tika ieviesta Mērījumu programma [API00A], kura aplūkota šī promocijas darba nodaļā "1.2.6 Mērījumu programmas piemērs".

Tādējādi MERIME inicializācijas gaitā ir ieplānots izmantot tos mērījumu datus, kas tiek savākti un apstrādāti Mērījumu programmas realizācijas gaitā. Apkopotie, MERIME metodei pieejamie mērījumu dati parādīti 27. tabulā.

27. tabula. Uzņēmumā ALFA programmatūras izstrādes projektos uzkrātā izstrādes procesa mērījumu informācija

Mērījums M_i MERIME metodē	Mērījuma pieļaujamā robeža ε^{M_i} MERIME metodē
Izstrādātā produkta apjoms (M_1)	20%
Programmatūras izstrādes procesa aktivitātēm patērētā darbietilpība attiecīgajā laika periodā (koncepcijas izstrādei (M_2), prototipa izstrādei (M_3), prasību specificēšanai (M_4), projektēšanai (M_5), implementēšanai (M_6), testēšanai (M_7), ieviešanai (M_8), uzturēšanai (M_9), dokumentēšanai (M_{10}), projekta pārvaldīšanai (M_{11}), lietotāju apmācībai (M_{12}), izstrādātāju mācībām (M_{13}), izmaiņu vadībai (M_{14}), kvalitātes nodrošināšanai (M_{15}))	20%
Vidējais atbildes laiks uz problēmziņojumu (M_{16})	10%
Kļūdu novēršanas efektivitāte (M_{17})	5%

2.6.2.1.4. IZSTRĀDES PROCESA PLĀNOŠANA

Izstrādes procesa plānošana notiek katrā programmatūras izstrādes projektā projektu uzsākot un katrā projekta robežšķirtnē. Plānošanas procesā tiek prognozētas šādu mērījumu vērtības.

1. Izstrādājamā produkta apjoms. Izstrādājamā produkta apjoma novērtēšanai izmanto funkcijpunktus, kur tas iespējams, vai programmrindiņas.
2. Darbietilpība plānotā apjoma produkta izstrādei. Darbietilpības prognozēšanai izmanto formālas metodes, visbiežāk COCOMO II metodi gan jaunas programmatūras izstrādei, gan programmatūras uzturēšanai.
3. Plānotā apjoma produkta izstrādei nepieciešamais laiks. Produkta izstrādei nepieciešamā laika prognozēšanai izmanto formālas metodes, visbiežāk COCOMO II metodi gan jaunas programmatūras izstrādei, gan programmatūras uzturēšanai.
4. Atsevišķām produkta izstrādes aktivitātēm nepieciešamā darbietilpība attiecīgajā laika periodā (koncepcijas izstrādei, prototipa izstrādei, prasību specificēšanai, projektēšanai, implementēšanai, testēšanai, ieviešanai un uzturēšanai, dokumentēšanai, projekta pārvaldīšanai, lietotāju apmācībai, izstrādātāju mācībām, izmaiņu vadībai, kvalitātes nodrošināšanai u.c.). Atsevišķām aktivitātēm nepieciešamo darbietilpību iegūst, izmantojot

industrijas procentuālos sadalījumus programmatūras izstrādes projektiem [JON98] u.c. avotus, ja ir pieejami.

5. Sagaidāmais problēmziņojumu un izmaiņu pieprasījumu skaits attiecīgajā laika periodā. Šeit parasti vadās no personīgās pieredzes, neizmantojot formālas metodes.

Uzņēmumā ALFA izmanto formālas metodes izstrādes procesa plānošanai, tomēr formāli iegūtie rezultāti tiek salīdzināti ar uzņēmumā pieejamajiem datiem par programmatūras izstrādes procesu, lai radītu lielāku uzticamību formāli iegūtajiem rezultātiem. Lai plānotu to izstrādes procesa raksturlielumu, piemēram, problēmziņojumu skaitu, vērtības, kuru iegūšanai nav pieejamas formālas metodes, nepieciešami uzņēmumā pieejamie dati.

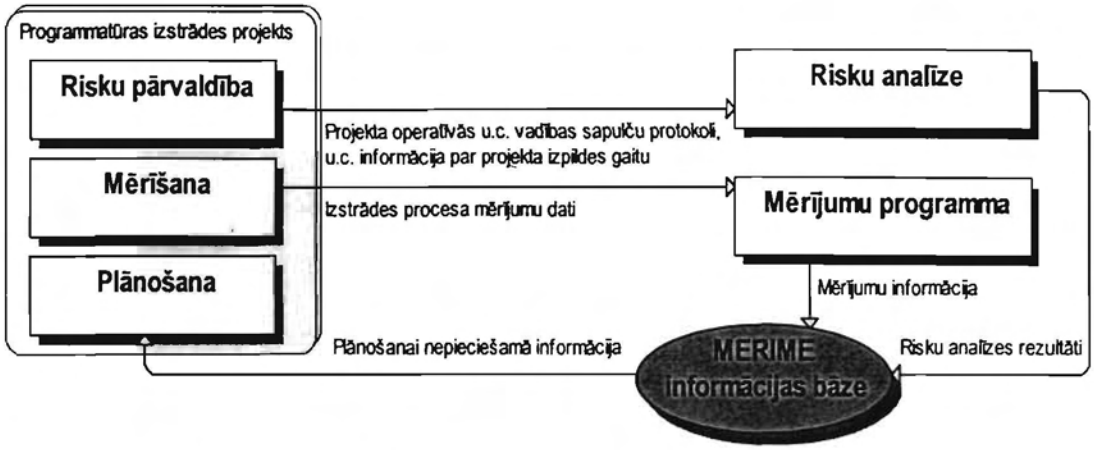
MERIME metodes inicializācijas solī apzināta tā informācija, kas nepieciešama plānošanai. Šī informācija būs prioritāra, vācot mērījumu datus.

MERIME inicializācijas solī tika nolemts neuzkrāt plānošanas informāciju MERIME informācijas bāzē, jo izstrādes procesa operatīvā plānošana ir projekta iekšējs process, tā vai tā daļu publiskošana izsauktu pretestību no izstrādātāju puses.

2.6.2.1.5. KOPSAVILKUMS PAR MERIME METODES INICIALIZĀCIJU UZŅĒMUMĀ ALFA

MERIME inicializācijas solī izdarītais ir nosaukts turpmāk.

1. Apzināta informācija, kāda nepieciešama izstrādes procesa plānošanai.
2. Nodefinēta MERIME metodes izmantošanas informācijas plūsma uzņēmumā ALFA, kas maksimāli integrēta ar katra programmatūras izstrādes projekta izstrādes procesu (skat. 23. attēls).
3. Inicializēta MERIME informācijas bāze.



23. attēls. MERIME inicializācijas solī nodefinētā informācijas plūsma

2.6.2.2. Mērījumu informācijas uzkrāšana

Mērījumu un risku analīzes informācijas uzkrāšana notiek tā, kā tas paredzēts MERIME inicializācijas solī, saskaņā ar definēto informācijas plūsmu.

Šobrīd MERIME-ALFA informācijas bāzē ir informācija par 25 aktīviem programmatūras izstrādes projektiem un 15 pabeigtiem programmatūras izstrādes projektiem. MERIME-ALFA informācijas bāzē ir informācija gan par jaunas programmatūras izstrādes, gan uzturēšanas, sistēmanalīzes un datu migrācijas projektiem.

2.6.2.3. Situācijas analīze

Saskaņā ar MERIME izmantošanas rāmi, situācijas analīzes solis ir pirmais, kad MERIME informācijas bāzē uzkrāto informāciju sāk izmantot programmatūras izstrādes projektu plānošanai.

Situācijas analīzes solī aplūkosim divus dažādus programmatūras izstrādes projektus, kur viens ir jaunas programmatūras izstrādes projekts augsta riska apstākļos, bet otrs – stabils programmatūras uzturēšanas projekts. Abu projektu operatīvajai plānošanai tika izmantota MERIME metode.

2.6.2.3.1. SITUĀCIJAS ANALĪZE JAUNAS PROGRAMMATŪRAS IZSTRĀDES PROJEKTĀ INVEST

INVEST – jaunas programmatūras izstrādes projekts, kuru izstrādā ~10 cilvēku komanda pēc pašvaldības pasūtījuma. Šobrīd projekts ilgst 6 mēnešus.

Projektu uzsākot, tika prognozēta izstrādei nepieciešamā darbietilpība (2300 cilvēkdienas) un izstrādei nepieciešamais laiks (10 mēneši), izmantojot COCOMO II metodi jaunas programmatūras izstrādes projektam. Izstrādes laiks 10 mēneši nebija pieņemams programmatūras pasūtītājam, tas tika administratīvi samazināts uz 5 mēnešiem, nesamazinot piegādājamā produkta funkcionalitātes apjomu. Tādējādi visā projekta izstrādes laikā ir aktuāls izstrādei nepieciešamā laika trūkums. Šādos apstākļos izstrādes procesa darbietilpības prognozēšanai vairs nevar izmantot COCOMO II metodi, jo tā nedod rezultātu, ja izstrādei nepieciešamo laiku samazina vairāk par 25% no nepieciešamā [COC2]. Projektu uzsākot, izstrādātāju komandai nebija zināšanas par izstrādes vidi. Šis risks pakāpeniski mazinājās projekta izpildes gaitā, līdz projekta izstrādes 6. mēnesī tas vairs nebija aktuāls. Formāli, saskaņā ar MERIME metodes situācijas apraksta definīciju, situācija projektā INVEST parādīta 28. tabulā.

Projekta izstrādes 4. mēnesī noskaidrojās, ka projekta izstrādi laikā pabeigt neizdosies, tomēr bija vēlme projektu pabeigt pēc iespējas ātri. Tādēļ bija nepieciešami padomi, ko darīt projekta kritiskā situācijā.

Projekta izstrādātājiem strādājot saspringtā režīmā, sāka kristies izstrādes produktivitāte, līdz trešajā mēnesī tā divkārt nokrītās (skat. 28. tabula). Lai nepieļautu turpmāku izstrādes procesa produktivitātes kritumu (skat. jautājuma zīmes 28. tabulā), kam par iemeslu bija darbietilpības palielināšanās, bija nepieciešams prognozēt implementēšanai un testēšanai nepieciešamo darbietilpību nākamajam mēnesim, izmantojot līdzīgo projektu pieredzi. Prognozētās mērījumu vērtības projektam INVEST parādītas nodaļā "2.6.2.4.1 Mērījumu vērtību prognozēšana projektam INVEST".

28. tabula. Situācijas attīstība projektā INVEST četros mēnešos

Mēnesis	1	2	3	4	5
Risks					
Izstrādātāju nepietiekamas zināšanas par izstrādes tehnoloģijām un vidi (R_5)[1]	5	4	4	3	2
Izstrādei nepieciešamā laika trūkums (R_8)	5	5	5	5	5
Mērījumi					
Implementēšanas produktivitāte	0.0008	0.0008	0.0004	0.0009	
Testēšanas produktivitāte		0.0025	0.0063	0.0125	
Darbietilpība implementēšanai (cilvēkstundas) (M_6)	412	577	781	949	?
Darbietilpība testēšanai (cilvēkstundas) (M_7)	0	20	36	64	?
Implementētais apjoms (%)	33%	47%	55%	70%	
Testētais apjoms (%)	0%	5%	15%	50%	

2.6.2.3.2. SITUĀCIJAS ANALĪZE PROGRAMMATŪRAS UZTURĒŠANAS PROJEKTĀ REGISTR

Projekts REGISTR ir programmatūras uzturēšanas projekts. Projekta gaitā tiek uzturēta valsts nozīmes IS. Projekta ilgums ir divi gadi, tajā strādā 8 cilvēki. Šis ir stabils projekts, kur pastāv niecīgas bažas, ka izstrādātāju nepietiekamas zināšanas par izstrādes tehnoloģijām un vidi, prasību nestabilitāte, izmaiņu veikšanai nepieciešamais laiks u.c. faktori varētu negatīvi ietekmēt izstrādes procesu. Risku analīzes un mērījumu datus šim projektam sāka vākt pirms četriem mēnešiem. Formāli saskaņā ar MERIME metodi, situācija projektā REGISTR parādīta 29. tabulā. Jautājuma zīmes parāda, kuru mērījumu vērtības prognozēt attiecīgajā situācijā, izmantojot līdzīgo projektu datus.

Projektā REGISTR operatīvajai plānošanai nepieciešams zināt darbietilpību implementēšanai un projekta pārvaldībai, lai plānotu attiecīgo speciālistu piesaisti citu projektu izpildei, nekaitējot projekta REGISTR izstrādei. Prognozētās mērījumu vērtības projektam REGISTR parādītas nodaļā "2.6.2.4.2 Mērījumu vērtību prognozēšana projektam REGISTR".

[1] Saskaņā ar MERIME-ALFA informācijas bāzes inicializāciju

29. tabula. Situācijas attīstība projektā REGISTR četros mēnešos

Mēnesis	1	2	3	4	5
Risks					
Izstrādātāju nepietiekamas zināšanas par izstrādes tehnoloģijām un vidi (R_5)	2	2	2	2	2
Mērījumi					
Programmatūras implementēšanai patērētā darbietilpība (cilvēkstundas) (M_6)	400	350	420	420	?
Projekta pārvaldīšanai patērētā darbietilpība (cilvēkstundas) (M_{11})	40	40	30	40	?

2.6.2.4. Mērījumu vērtību prognozēšana plānotajam projektam

Mērījumu vērtību prognozēšanas solī saskaņā ar MERIME metodi, prognozē mērījumu vērtības, izmantojot situācijas analīzes solī iegūto situācijas aprakstu projektā. Šeit parādīts, kā un kāda informācija tika prognozēta projektiem INVEST un REGISTR, kurus aplūkojām situācijas analīzes solī.

2.6.2.4.1. MĒRĪJUMU VĒRTĪBU PROGNOZĒŠANA PROJEKTAM INVEST

Lai risinātu saspringto situāciju INVEST projektā, tika izmantoti padomi, kas MERIME informācijas bāzē uzkrāti risku pārvaldības gaitā. Tika aplūkotas šādas alternatīvas.

1. Riska faktoram "Nepietiekamas zināšanas par vidi un tehnoloģijām".
   1.1. Piesaistīt vienu izstrādes vides ekspertu, kurš var būt ārējs konsultants, kura galvenais uzdevums ir nevis programmēt, bet konsultēt citus izstrādātājus.
   1.2. Izvēlēt vienu pieredzējušu, talantīgu un komunikablu cilvēku, kas būtu "ledlauzis" vidē un konsultētu pārējos.
2. Riska faktoram "Nepietiekams laiks projekta izpildei".
   2.1. Nomainīt tos, kuriem ir maza pieredze, ar tiem, kuriem ir pieredze par izstrādes vidi. Būtu labi, ja darbinieki būtu no tās pašas uzņēmējdarbības sfēras.

2.2. Palielināt slodzi tiem projekta darbiniekiem, kuriem ir pieredze izstrādes vidē (varbūt slīdošais grafiks, papildus motivācija utt.).

Jau projekta sākumā tika izvēlēts viens darbinieks, kura galvenā prioritāte bija nevis programmatūras izstrāde, bet gan pārējo projekta dalībnieku konsultēšana par izstrādes vidi. Šis risinājums izrādījās veiksmīgs, pēc četriem mēnešiem šis risks projektā INVEST vairs nebija aktuāls.

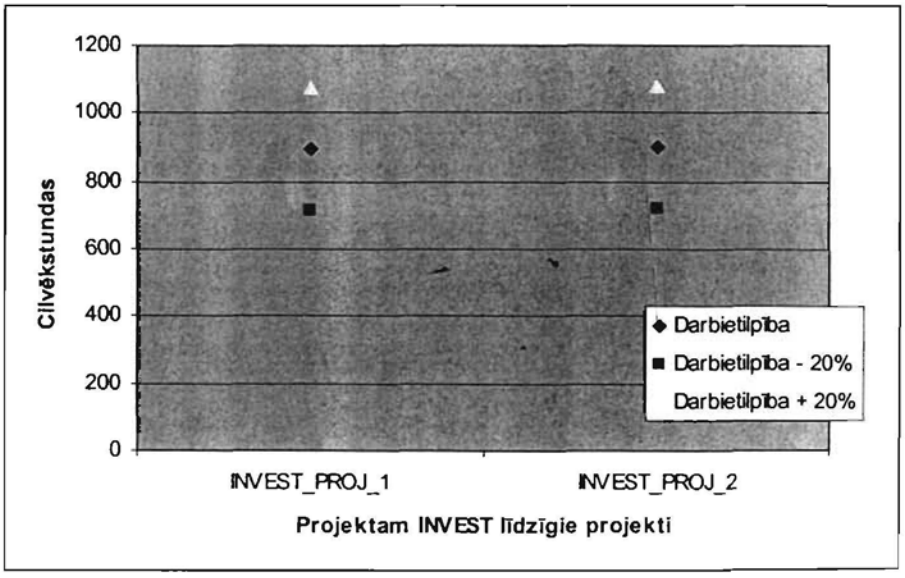
Grūtāk bija ar rīcību situācijā, kad projekta izstrādei nepietika laika. Pēc diviem mēnešiem, kad projekta pārvaldībai vairs nebija šaubu, ka projekta izstrāde noteikti nebeigsies plānotajā laikā, projekta izstrādātāju komandai pievienoja izstrādātājus. Rezultātā radās produktivitātes kritums divkārt (skat. 28. tabula), nedodot gaidīto efektu. Tika pieņemts lēmums saglabāt projekta izstrādātāju skaitu, palielinot slodzi darbiniekiem. Izstrādātāji pēc slīdoša grafika strādāja virsstundas.

Izmantojot MERIME metodi, projekta INVEST operatīvās plānošanas vajadzībām tika prognozēta implementēšanas un testēšanas darbietilpība, no kuras var iegūt produktivitāti.

Projektam INVEST attiecīgajā situācijā MERIME-ALFA informācijas bāzē tika atrasti divi līdzīgi projekti INVEST_PROJ_1 un INVEST_PROJ_2 (skat. 30. tabula). Tādējādi projekta INVEST implementēšanai un testēšanai prognozētā darbietilpība būs INVEST_PROJ_1 vai INVEST_PROJ_2 attiecīgā mērījuma ϵ apkārtnē (±20%). Piemēram, 24. attēlā redzama projekta INVEST implementēšanas darbietilpības prognoze 5. mēnesim. Implementēšanas darbietilpība nākamajam mēnesim prognozēta no 4 līdz 6 cilvēkmēnešiem, bet testēšanai – no 0.2 līdz 0.4 cilvēkmēnešiem. Šeit projektam INVEST būtisks rezultāts ir implementēšanai prognozētā darbietilpība, kas līdzīgos projektos līdzīgā situācijā samazinās, nevis palielinās, kā tas būtu turpinot palielināt programmētāju skaitu.

30. tabula. Projektam INVEST atrastie paraugprojekti

Mēnesis	1	2	3	4	5
INVEST_PROJ_1					
Risks					
Izstrādātāju nepietiekamas zināšanas par izstrādes tehnoloģijām un vidi (R_5)	4	3	4	3	3
Izstrādei nepieciešamā laika trūkums (R_8)	4	5	5	4	4
Mērījumi					
Darbietilpība implementēšanai (cilvēkstundas) (M_6)	402	612	821	1003	894
Darbietilpība testēšanai (cilvēkstundas) (M_7)	0	0	44	55	63
INVEST_PROJ_2					
Risks					
Izstrādātāju nepietiekamas zināšanas par izstrādes tehnoloģijām un vidi (R_5)	5	4	4	3	3
Izstrādei nepieciešamā laika trūkums (R_8)	4	4	5	5	5
Mērījumi					
Darbietilpība implementēšanai (cilvēkstundas) (M_6)	351	592	786	922	901
Darbietilpība testēšanai (cilvēkstundas) (M_7)	0	18	26	46	57



24. attēls. Projektam INVEST implementēšanas darbietilpības prognoze nākamajam mēnesim

Izmantojot MERIME metodi projekta INVEST operatīvajai plānošanai, projektā INVEST bija turpmāk nosauktie izstrādes procesa uzlabojumi.

1. Tika atrasti risinājumi projektam INVEST kritiskā situācijā augstu risku ietekmes mazināšanai.
2. Tika prognozētas projektam nepieciešamo mērījumu vērtības, izmantojot to informāciju, kāda bija pieejama projektā, neveltot papildus darbu mērījumu datu vākšanai un apkopošanai.
3. Implementēšanas un testēšanas darbietilpības mērījumu vērtības netika iegūtas kā atvasinātie mērījumi, proti, prognozējot darbietilpību visam izstrādes procesam kopumā un pēc tam sadalot pēc industrijas sadalījuma, bet gan tieši, kas palielināja uzticību iegūtajam rezultātam.

Izmantojot MERIME metodi projekta INVEST operatīvajai plānošanai, projekta pārvaldnieks to atzina par lietderīgu atbalstu plānošanas procesā. MERIME metodes dotie rezultāti uzskatāmi demonstrēja saspringto situāciju projektā visām iesaistītajām pusēm saprotamā valodā un palīdzēja izvairīties no kļūdaina projekta pārvaldības lēmuma pieņemšanas.

2.6.2.4.2. MĒRĪJUMU VĒRTĪBU PROGNOZĒŠANA PROJEKTAM REGISTR

Lai prognozētu implementēšanai un projekta pārvaldībai nepieciešamo darbietilpību nākamajā mēnesī, izmanto situācijas attīstību projektā REGISTR četru mēnešu garumā (skat. 29. tabula). MERIME informācijas bāzē meklē paraugprojektus, kuru attiecīgo mērījumu un risku vērtības ir līdzīgas (mērījumi atrodas attiecīgo mērījumu ϵ robežās, risku vērtības neatšķiras vairāk par S).

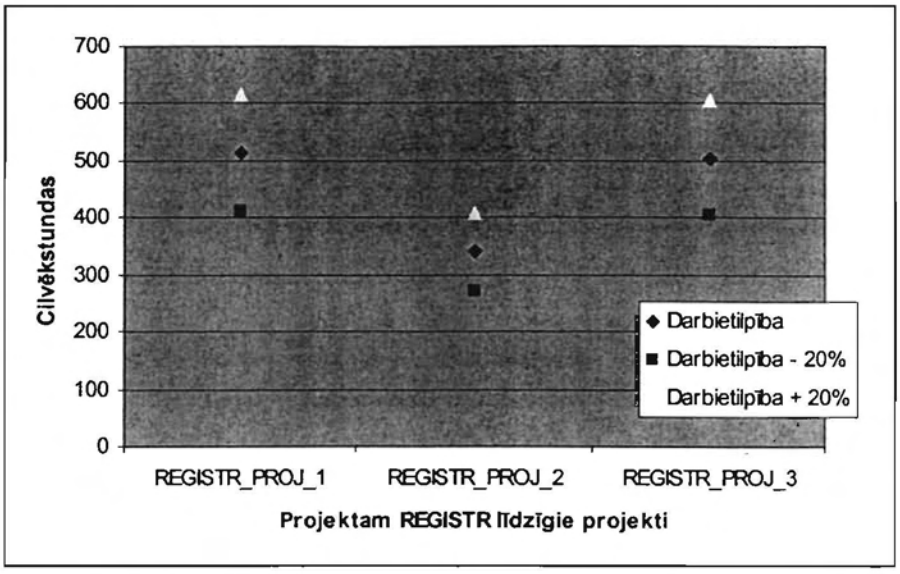
31. tabulā doti MERIME informācijas bāzē atrasto paraugprojektu REGISTR_PROJ_1 līdz REGISTR_PROJ_3 mērījumu vērtības. Šie projekti ir līdzīgi projektam REGISTR attiecīgajā situācijā pēc MERIME metodes projektu līdzības definīcijas.

Tādējādi projekta REGISTR implementēšanai un projekta pārvaldībai prognozētā darbietilpība būs REGISTR_PROJ_1, REGISTR_PROJ_2 vai REGISTR_PROJ_3 attiecīgā mērījuma ϵ apkārtnē (±20%). Piemēram, 25. attēlā redzama projekta REGISTR implementēšanas darbietilpības prognoze 5. mēnesim. Redzams, ka implementēšanas darbietilpība nākamajam mēnesim

saglabāsies 2 līdz 3 cilvēkmēnešu apjomā, jeb tajā pašā līmenī kā patreizējā situācijā (420 cilvēkstundas 29). Projekta pārvaldības darbietilpība nākamajā mēnesī prognozēta no 0.3 līdz 0.5 cilvēkmēnešiem, saglabājoties esošajā līmenī. Tādēļ jāizvairās no implementēšanai un projekta pārvaldībai piesaistīto darbinieku iesaistīšanas citos projektos.

31. tabula. Projektam REGISTR atrastie paraugprojekti

Mēnesis	1	2	3	4	5
REGISTR_PROJ_1					
Risks					
Izstrādātāju nepietiekamas zināšanas par izstrādes tehnoloģijām un vidi (R_5)	1	1	2	1	1
Mērījumi					
Programmatūras implementēšanai patērētā darbietilpība (cilvēkstundas) (M_6)	514.5	535.5	448	591.7	513.5
Projekta pārvaldīšanai patērētā darbietilpība (cilvēkstundas) (M_{11})	20	26.5	11.5	30.5	71.5
REGISTR_PROJ_2					
Risks					
Izstrādātāju nepietiekamas zināšanas par izstrādes tehnoloģijām un vidi (R_5)	3	3	3	3	2
Mērījumi					
Programmatūras implementēšanai patērētā darbietilpība (cilvēkstundas) (M_6)	384	343	425	353	340
Projekta pārvaldīšanai patērētā darbietilpība (cilvēkstundas) (M_{11})	62	45	52	65.5	95.5
REGISTR_PROJ_3					
Risks					
Izstrādātāju nepietiekamas zināšanas par izstrādes tehnoloģijām un vidi (R_5)	2	2	2	2	2
Mērījumi					
Programmatūras implementēšanai patērētā darbietilpība (cilvēkstundas) (M_6)	467	171	472	458	504
Projekta pārvaldīšanai patērētā darbietilpība (cilvēkstundas) (M_{11})	17	11	38	16	14



25. attēls. Projektam REGISTR implementēšanas darbietilpības prognoze nākamajam mēnesim

Izmantojot MERIME metodi projektā stabilā programmatūras uzturēšanas projektā REGISTR, bija turpmāk nosauktie uzlabojumi.

1. Tika iegūta informācija operatīvajai plānošanai, kas ļāva novērst kļūdas projekta operatīvajā plānošanā.
2. Intuitīvās prognozes tika apstiprinātas ar skaitļiem, ļaujot pamatot intuitīvo spriedumu.

2.6.2.5. Secinājumi par MERIME metodes izmantošanu uzņēmumā ALFA

Šajā nodaļā parādīta MERIME metodes izmantošana operatīvajai plānošanai gadījumos, kad ir pieejama informācija par nedaudziem projektiem. Šāda situācija plānošanai uzskatāma par kritisku un jābūt ļoti uzmanīgiem, interpretējot iegūtos rezultātus. Taču arī šādā situācijā MERIME metode ir izmantojama.

Uzlabojumi, kādus MERIME metode deva uzņēmumam ALFA, nosaukti turpmāk.

1. Integrēts risku pārvaldības un izstrādes procesa mērīšanas procesi, to rezultātus izmantojot plānošanai, uzlabojot informācijas apmaiņu starp dažādiem projektiem.

2. Regulāra risku analīzes un mērīšanas procesu rezultātu apstrāde to pieredzi, kura ir koncentrējusies pie atsevišķo projektu izpildītājiem un šo projektu vadītājiem, padara pieejamu citiem projektiem plānošanai.
3. Izmantojot MERIME informācijas bāzi, iespējams salīdzināt dažādus projektus viena uzņēmuma ietvaros, izmantojot to informāciju, kāda ir pieejama projektos.
4. Uzkrātās informācijas apmaiņa neprasa no izstrādātājiem papildus darbu datu savākšanai.
5. Izstrādes procesa gaitā savāktie dati tiek pārstrādāti praktiski izmantojamā informācijā, kas noderīga plānošanas, risku pārvaldības u.c. projekta pārvaldības funkciju veikšanai.
6. Projektu darbietilpības un izstrādei nepieciešamā laika prognozēšanai ir pieejama vairāk informācija, tai skaitā dati par citiem projektiem.
7. Iespējams izmantot fragmentāru mērījumu un risku analīzes informāciju. Šī ir būtiska priekšrocība. Piemēram, projekta izstrādes procesam nonākot saspringtā situācijā, izstrādātāji mēra izstrādes procesu, pieraksta risku analīzes rezultātus u.c., uzkrājot augstas kvalitātes informāciju par izstrādes procesu. Situācijai normalizējoties, datu kvalitāte samazinās vai to pierakstīšana tiek pārtraukta vispār. MERIME metode ļauj izmantot arī šādus datus.

2.7. Secinājumi par MERIME metodes izmantošanu

MERIME metodes izmantošanai ir vairākas priekšrocības. Metodes priekšrocības ir turpmāk nosauktās.

1. MERIME metodi var izmantot dažādu izstrādes procesa raksturlielumu prognozēšanai gan visam projektam kopumā, gan operatīvajai plānošanai. Iespējams iegūt tās mērījumu vērtības, kuras noglabātas MERIME informācijas bāzē. Proti, esošās formālās darbietilpības prognozēšanas metodes koncentrējas uz darbietilpības un izstrādei nepieciešamā laika prognozēm. Šie, nenoliedzami, ir svarīgākie rezultāti, taču MERIME piedāvā iegūt arī citus mērījumus. Piemēram, produktivitāti, produkta piegādes ātrumu, darbietilpību implementēšanai, testēšanai, dokumentēšanai u.c.
2. Izmanto pieredzi no tā programmatūras ražošanas uzņēmuma, kurā informācija par projektiem tiek uzkrāta, nodrošinot mērīšanas procesa galveno veiksmes faktoru – regulāru mērījumu analīzes atgriezenisko saiti mērīšanas procesa dalībniekiem.
3. MERIME metode izmantojama situācijās, kad projektu nav daudz. Šāda situācija ir raksturīga programmatūras izstrādes uzņēmumiem Latvijā. Pat tad, ja projektu būtu ievērojami vairāk, dati par vecākajiem projektiem nav izmantojami straujās tehnoloģijas attīstības dēļ.
4. Izmantojot MERIME metodi projekta gaitas operatīvajai plānošanai, iespējams izmantot fragmentārus datus. Šī priekšrocība ir īpaši svarīga, jo praksē bieži projektos uzkrāj risku analīzes un mērījumu informāciju saspringtās situācijās, tādējādi veidojas teicamas kvalitātes, bet fragmentāri dati par projektiem. Situācijai normalizējoties, datu kvalitāte samazinās vai to pierakstīšana tiek pārtraukta vispār. MERIME metode ļauj izmantot arī šādus datus.
5. Augstu risku vērtību mazināšanai ieteicamās darbības ir piesaistītas uzņēmuma kultūrai un iespējām.

6. MERIME metodes ietvaros risku pārvaldības procesu neuzlūko kā savrupu izstrādes procesa atbalsta procesu, tas ir integrēts ar mērīšanas procesu, rezultātus izmantojot plānošanai.
7. MERIME metodes situācijas analīzes rezultātā izveidoto aprakstu var izmantot projekta situācijas komunikācijai visām izstrādes procesā iesaistītajām pusēm, jo tas atspoguļo gan izstrādes procesa stāvokli, ko raksturo mērījumi, gan vidi, kurā process notiek, ko parāda risku analīzes rezultāti.
8. MERIME metodei nav fiksēta mērījumu un risku komplekta, kurus šī metode izmanto, atšķirībā no citiem prognozēšanas modeļiem. Tādējādi plānošanas gaitā iespējams izmantot tieši tik informācijas, cik ir zināms par plānojamo projektu. Šī priekšrocība ir ļoti būtiska projekta izstrādes sākuma stadijā, kad ļoti maz zināms par uzsākto projektu.

Nobeigums

Nepārtraukti strauji attīstoties informācijas tehnoloģijām, informācijas sistēmu risinājumiem kļūstot arvien sarežģītākiem, arī programmatūras izstrādes process kļūst komplicētāks. Tādēļ nemainīgi augsts ir pieprasījums pēc metodēm, kuras būtu praktiski izmantojamas programmatūras izstrādes procesa plānošanai, uzraudzībai un uzlabošanai. Par to liecina kaut vai tas, ka pēdējos gados izstrādātas vairākas šādas metodes, kā arī pārstrādāti vairāki programminženierijas standarti, lai tos pielāgotu mainīgajam izstrādes procesam.

Galvenie šī darba rezultāti ir turpmāk nosauktie.

1. Izstrādāta un validēta analoģiju bāzēta programmatūras izstrādes procesa raksturlielumu prognozēšanas metodoloģija MERIME.
2. MERIME, integrējot divus programmatūras izstrādes procesa atbalsta procesus: mērīšanu un risku pārvaldību, rada prognozējamu vidi izstrādes procesam un atbalstu izstrādes procesa plānošanai, sniedzot informāciju par izstrādes procesu visām procesa dalībniekiem.
3. Izstrādātā metotodoloģija MERIME ir ieviesta programmatūras izstrādes uzņēmumā, sniedzot reālu atbalstu izstrādātājiem programmatūras izstrādes procesa uzlabošanai.
4. Veikts pētījums par programmatūras izstrādes risku analīzi un apzināti programmatūras izstrādes procesu ietekmējošie riski, preventīvās un korektīvās darbības to apkarošanai, kas aktuāli programmatūras izstrādātājiem Latvijā.
5. Pētīts programmatūras izstrādes mērīšanas process un izstrādāti ieteikumi mērīšanas procesa ieviešanai programmatūras izstrādes uzņēmumos.

Autores veiktie pētījumi prezentēti vairākās starptautiskās zinātniskās konferencēs un to publicētajos materiālos.

Kaut arī darbā izvirzītie mērķi ir sasniegti, savu ieguldījumu šajā jomā autore turpina pētījumus programmatūras izstrādes procesa uzlabošanas jomā Rīgas Informācijas tehnoloģijas institūtā, izstrādājot programmatūras izstrādes procesa metrikas, kas konkrēti raksturotu programmatūras izstrādes procesu Latvijā. Šādas metrikas ir vitāli nepieciešamas lai objektīvi parādītu, ka programmatūras izstrādes process Latvijā ir kvalitatīvs un produktīvs. Tādējādi

Latvijas programminženierijas industrijā strādājošie uzņēmumi ir drošs sadarbības partneris ārvalstu investoriem.

Darba autore ir pateicīga par atbalstu šī promocijas darba izstrādē Rīgas Informācijas tehnoloģijas institūtam, Latvijas Zinātnes padomei par piešķirto doktorantūras grantu, LU K.Morberga stipendijas komisijai par K.Morberga stipendiju 2001./2002. un 2002./2003. ak.g. Darbs daļēji finansēts arī no LZP sadarbības projekta nr. 02.0002.1.3. "Latvijas informātikas tehnoloģijas attīstīšana konkurētspējīgas produkcijas ražošanai tirgus specifiskos sektoros".

Literatūra

1. [BOE88] B. Boehm. *"A Spiral Model of Software Development and Enhancement"*. IEEE Computer, vol.21, #5, May 1988, pp. 61-72.
2. [JON98] T. Capers Jones. *"Estimating Software Costs"*. McGraw-Hill, USA, 1998 724 p.
3. [HET93] B.Hetzel. *"Making Software Measurement Work"*. John Wiley & Sons, Inc., 1993, 290 p.
4. [BOE81] B.W. Boehm. *"Software Engineering Economics"*. Prentice-Hall, NJ, 1981, 767 p.
5. [SLI95] *"SLIM 3.2 User's Manual"*. Quantitative Software Measurement, Inc., McLean, Va., 1995, 263 p.
6. [RUB83] H. Rubin. *"Macroestimation of Software Development Parameters: the Estimacs System"*. SOFTFAIR Conference on Software Development Tools, Techniques and Alternatives, Arlington, IEEE Press, New York, July 1983, pp. 4-16.
7. **[API95] Baiba Apine. *"A Visualisation and Analysis of Experimentally Gathered Results in an User-Friendly Mode"*. Informatica, 1995, Vol.6, No.4., pp. 387-396.**
8. **[API96] Baiba Apine, Arnis Kleins, Ojars Krasts, Uldis Sukovskis, Artis Teilans, Vita Zviedre. *"Modelling methodology and tool for business systems: Registrator"*. Abstracts of the international conference: Simulation, Gaming, Training and Business Process Reengineering in Operations, 1996, Riga, Latvia, pp. 44-45.**
9. **[API98] Baiba Apine. *"Practitioner's approach to software cost estimation"*. Abstracts of the international conference: DB & IS '98, 1998, Riga, Latvia, pp. 134-140.**
10. **[API00] Baiba Apine, Ilgvars Apinis, Ojars Krasts, Uldis Sukovskis. *"Meta-model Based and Component Based Approach for Information Systems Design"*. Proceedings of the 4th IEEE International Baltic Workshop: Baltic DB&IS '2000, 2000, Vilnius, Lithuania, pp. 78-83.**
11. [SOL99] R.Solingen, E.Berghout. *"The Goal/Question/Metric Method"*. McGrawHill, 1999, 195 p.

12. [YOU97] Yourdon, E. *"Death March. The complete Software Developer's Guide to Surviving 'Mission Impossible' Projects"*. Prentice Hall, 1997, 218 p.
13. [GOV81] Gove, Philip Babcock, Editor. *"Webster's Third New International Dictionary: Unabridged"*. Springfield, MA: Merrian-Webster, 1981.
14. [SEI92] Software Engineering Institute. *"The SEI Approach to Managing Software Technical Risks"*. Bridge, October, 1992, pp. 19-21.
15. [GLU94] Gluch, David P. *"A Construct for Describing Software Development Risks"*. Technical report CMU/SEI-94-TR-14 ESC-TR-94-014.
16. [ROS98] Rosenberg, Linda H., Hammer T., Gallo A. *"Continuous Risk Management at NASA"*. [Elektroniskais resurss].-Pieejas veids: tīmeklis WWW. URL: http://satc.gsfc.nasa.gov/support. Resurss aprakstīts: 2001. g. 10.apr.
17. [BAS98] Basili R., Kontio J. *"Riskit: Increasing Confidence in Risk Management"*. [Elektroniskais resurss].-Pieejas veids: tīmeklis WWW. URL: http://satc.gsfc.nasa.gov/support.-Resurss aprakstīts: 2001. g. 10.apr.
18. [HYA96] Hyatt L. E., Rosenberg L. H. *"Software Metrics Program for Risk Assessment"*. [Elektroniskais resurss].-Pieejas veids: tīmeklis WWW. URL: http://satc.gsfc.nasa.gov/support/IAC_OCT96.-Resurss aprakstīts: 2001.g. 10.apr.
19. [CMM99] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber. *"Capability Maturity Model for Software, Version 1.1"*. Software Engineering Institute, CMU/SEI-93-TR-24.
20. [KEI02] Keil M., Tiwana A., Bush A. *"Reconciling user and project manager perceptions of IT project risk: aDelphi study"*. Info Systems J, Vol. 12, 2002, pp. 103-119.
21. [IEE99] IEEE P1540 *"Standard for Software Life Cycle Processes – Risk Management"*, IEEE, December, 2000, 48 p.
22. [SED01] Sedigh-Ali S., Ghafoor A., Paul R. A. *"Software Engineering Metrics for COTS-Based Systems"*. IEEE Software, May, 2001, pp. 38-46.

23. [KAS00] Kasser J., Williams V. R. *"What do you Mean can't Tell me if my Project is in Trouble?"*. [Elektroniskais resurss].-Pieejas veids: tīmeklis WWW. URL: http://satc.gsfc.nasa.gov/support.-Resurss aprakstīts: 2001. g. 12.apr.

24. [LOC96] Lockyer, K.,Gordon, J. *"Project Management and Project Network Techniques"*. Bell and Bain Ltd., 1996, pp. 49-51.

25. [COC2] C.Abts, B.Boehm, B.Clark, S.Devnani-Chulani. *"COCOMO II Model Definition Manual"*. University of Southern California, 2000, 68 p.

26. [GAR96] D. Garmus, D. Herron. *"Measuring the Software Process"*. Prentice-Hall, Inc., USA, 1996.

27. [GRD98] J.Bārzdiņš, J.Tenteris, Ē.Viļums. *"Biznesmodelēšanas valoda GRAPES-BM 4.0 un tās lietošana"*. Poligrāfists, 1998, 150 lpp.

28. [API00A] Baiba Apine, Uldis Smilts, Uldis Sukovskis. *"Software Measurement Practice to Address Customer Satisfaction"*. Scientific Proceedings of Riga Technical University, 2000, Applied Computer Systems. – 1st thematic issue, pp. 11 – 18.

29. [KEM93] Chris F. Kemerer. *"Empirical studies of assumptions that underlie software cost estimation"*. Information and Softw. Technol., Vol.34 #4, 1992, pp. 211-218.

30. [CAP77] *"Теория прогнозирования и принятия решений"*. Под ред. С.А. Саркисяна. М.: Высшая школа, 1977.

31. [NIC86] Ницецкий Л.В., Новицкий Л.П. *"Применение методов экспертного опроса для оценки качества диалоговых обучающих систем"*. Методы и средства кибернетики в управлении учебным процессом высшей школы. Сборник научных трудов, Рига РПИ, 1986.

32. [KEM87] Chris F. Kemerer, *"An Empirical Validation of Software Cost Estimation"*, ACM Communication, May, 1987, pp. 416-429.

33. [PUT92] Lawrence H. Putnam, Ware Myers. *"Measures for Excelence"*. Yourdon Press Computing Series, 1992.

34. [BOE84] Barry W. Boehm. *"Software Engineering Economics"*. IEEE Transactions on Software Engineering, Jan, 1984, pp. 4-21.

35. [JEN84] Randall W.Jensen. *"A Comparision of the Jensen and COCOMO Schedule and Cost Estimation Model"*. Proceedings of the International Society of Parametric Analysis, 1984, pp. 96-106.

36. [JEN95] Randall W.Jensen. *"A New Perspective in Software Schedule and Cost Estimation"*. [Elektroniskais resurss].-Pieejas veids: tīmeklis WWW. URL: http://www.seisage.com/publications.htm .-Resurss aprakstīts: 2003. g. 17.feb.

37. [JON96] Caper Jones. *"Applied Software Metrics"*. McGraw Hill, 1996, 324 p.

38. [PAR88] Robert E. Park. *"The Central Equation of the PRICE Software Cost Model"*. 4th COCOMO User's Group Meeting, November, 1988.

39. [PAR95] Naval Sea Systems Command . *"Parametric Cost Estimating Handbook"*. [Elektroniskais resurss].-Pieejas veids: tīmeklis WWW. URL: http://www.jsc.nasa.gov/bu2/PCEHHTML/pceh.htm.-Resurss aprakstīts: 2003. g. 17.feb.

40. [BOE91] Barry W. Boehm. *"Software Risk Management: Principles and Practices"*. IEEE Software, January, 1991, pp. 32-41.

41. [CIS02] Information Systems Audit and Control Association. *"2002 CISA Review Manual"*, 2002, 742 p.

42. [QSM02] *"SLIM-Estimate 5.0"*. [Elektroniskais resurss].-Pieejas veids: tīmeklis WWW. URL: http://www.qsm.com/slim_estimate.html.-Resurss aprakstīts 2003. g. 3.jan.

43. [EXP02] *"Estimation and Measurement Methodology of Experience Pro 3.0"*. [Elektroniskais resurss].-Pieejas veids: tīmeklis WWW. URL: http://www.sttf.fi/html.-Resurss aprakstīts 2002.g.12.jūn.

44. [TAU81] Robert C. Tausworthe. *"Deep Space Network Software Cost Estimation Model"*. Jet Propulsion Laboratory Publication 81-7, Pasadena, C.A., April, 1981.

45. [SLO02] *"Counting Source Lines of Code (SLOC)"*. [Elektroniskais resurss].-Pieejas veids: tīmeklis WWW. URL: http://www.dwheeler.com/sloc.-Resurss aprakstīts 2002.g.12.jūn.

46. [PAR92] Park R.E. *"Software Size Measurement: A Framework for Counting Source Statements"*. Technical report, CMU/SEI-92-TR-20, ESC-TR-92-020, September, 1992.

47. [HUM02] *"Estimatin With Objects – Part II"*. [Elektroniskais resurss].- Pieejas veids: tīmeklis WWW. URL: http://www.sei.cmu.edu/publications/articles/watts-humphrey/estimate-objects-002.html.-Resurss aprakstīts 2002.g.12.jūn.

48. [IFP98] IFPUG. *"Function Point Counting Practices Manual"*. [Elektroniskais resurss].-Pieejas veids: tīmeklis WWW. URL: http://www.ifpug.org/publications/manual.htm.-Resurss aprakstīts 2003.g.17.feb.

49. [MAR98] Charles Symons. *"Software Sizing and Estimating: MKII Function Point Analysis"*. John Wiley&Sons, 1993, 76 p.

50. [HAL77] Halstead. *"Elements of Software Science"*. New York, Elsevier North-Holland, 1977.

51. [WAR89] Ward W.T. *"Software Defect Prevention Using McCabe's Complexity Metric"*. Hewlet-Packard Journal, April, 1989, pp. 64-68.

52. [PRE00] Pressman, R.S. *"Software Engineering A Practitioner's Approach"*. McGraw-Hill, 2000, pp. 77-108.

53. [IEE93] IEEE Software Engineering Standards. *"Std.61012-1990"*. pp. 47-48.

54. [WHI95] Whitmire S.A. *"An introduction to 3D Function Points"*. Software Development, April, 1995, pp. 43-53.

55. [API02] Baiba Apine. *"Measurements and risks Based Method to Support Software Development Process Planning"*. Proceedings of the Fifth International Baltic Conference, BalticDB&IS 2002, 2002, Tallinn, Estonia, pp. 3-14.

56. [API02c] Baiba Apine. *"Measurements and risks Based Method to Support Software Development Process Planning"*. Databases and Information Systems II, selected papers of BalticDB&IS 2002, 2002, Kluwer Academic Publishers, The Netherlands, pp. 187-199.

57. **[API02a] Baiba Apine. *"Software Development Risk Management Survey"*. Information Systems Development. Advances in Methodologies, Components, and Management, 2002, Kluwer Academic Publishers, USA, pp. 241-251.**

58. **[API02b] Baiba Apine, Martins Gills, Janis Plume. *"Software Development Process Improvement through Measurements and Requirements Traceability"*. Proceedings of the Fifth International Baltic Conference, BalticDB&IS 2002, 2002, Tallinn, Estonia, pp. 65-76.**

59. [MAY00] Mayes J. *"Achieving Business Objectives III: A Real-World Software Process Improvement Implementation"*. IT Metrics Strategies, August, 2000, vol. IV, no. 8, pp. 1-9.

60. [FOR98] P. Forselius. *"Situation Analysis of Software Projects - The Key Input for SPI"*. EuroSPI'99, Pori, October 27th, 1999.

61. [ISB01] International Software Benchmarking Standards Group. *"Practical Project Estimation"*. 2001, 46 p.

62. [BEN03] *"The Guru Method Prevails"*. [Elektroniskais resurss].-Pieejas veids: tīmeklis WWW. URL: http://www.cutter.com/benchmark/index.html.-Resurss aprakstīts 2003.g.07.jan.

63. [RUB99] *"The 1999 Worldwide Benchmark Report: Software Engineering and IT Findings and Projections"*. [Elektroniskais resurss].-Pieejas veids: tīmeklis WWW. URL: http://www.cutter.com/itms/itms9902.html.-Resurss aprakstīts 2002.g.08.jūn.

64. [CAR00] Carlsson, S. *"Iformation Systems Development: Participation and Intersubjectivity – is this just a Matter o Communication?"*. Current Trends in Information Systems Development Methodologies, Gdansk, Poland, 1988.

65. [BAI99] Baik, J. *"The Effects of CASE Tools on Software Development Effort"*. Center for Software Engineering Computer Science Department University of Southern California, 1998, pp. 98.

Tabulas

Attēli

I Pielikums. Funkcijpunktu skaita iegūšanas metodika

Saturs

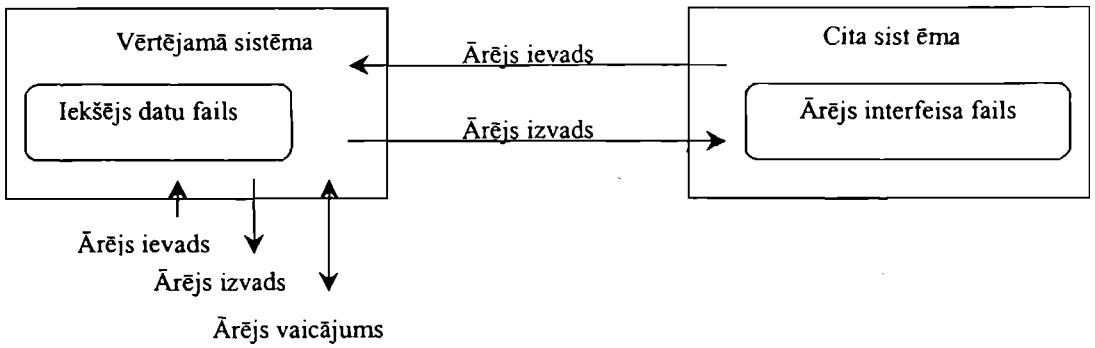
1. Pamatjēdzieni

Funkcijpunkts ir programmsistēmas funkcionalitātes mērvienība, kas raksturo sistēmu no lietotāja viedokļa. Funkcijpunktu skaits programmsistēmai nav atkarīgs no programmatūras realizācijas vides. Nosakot sistēmas funkcijpunktu skaitu, sistēmu aplūko no lietotāja viedokļa. Tehniskās detaļas un sarežģījumi, kas rodas tehnisku iemeslu dēļ realizācijas laikā, netiek ņemti vērā. Funkcijpunktu skaita iegūšanai jāsaskaita (skat. 1.attēls).

1. **Sistēmas iekšējos datu failus**. Sistēmas iekšējais datu fails ir savstarpēji loģiski saistītu datu kopa, kuras elementus rada, labo un iznīcina sistēmas iekšienē. Sistēmas iekšējais datu fails ir, piemēram, informācija par sistēmas lietotājiem (vārds, parole utml.).
2. **Vaicājumus**. Vaicājums ir ievada iniciēts sistēmas process, kura rezultātā atlasa datus no sistēmas iekšējiem datu failiem. Vaicājums neizmaina sistēmas iekšējos datu failus. Vaicājums ir, piemēram, visu to rēķinu atlase, kas izrakstīti nedēļas laikā.
3. **Ievadus**. Ievads ir sistēmas process, kas apstrādā datus vai vadības informāciju, kas ienāk sistēmā no ārpuses. Saņemtie dati modificē sistēmas iekšējos datu failus. Ievads ir, piemēram, sistēmā apstrādājamo datu ievadforma.
4. **Izvadus**. Izvads ir sistēmas process, kas ģenerē datus vai vadības informāciju, kas iziet ārpus sistēmas robežām. Izvada piemērs ir vienas atskaites drukāšana.
5. **Ārējos interfeisu failus**. Ārējā interfeisa fails ir savstarpēji loģiski saistītu datu kopa vai vadības informācija, no kurās vērtējamā sistēma nolasa datus, bet tos modificē cita sistēma. Ārējais interfeisa fails ir, piemēram, citas sistēmas uzturēta datu bāze, kuras datiem piekļūst no vērtējamās sistēmas.

Vērtējot jāatceras, ka jāuzskaita tikai tie vaicājumi, ievadi vai izvadi, kuri saskatāmi lietotājam. Arī iekšējiem datu failiem un ārējo interfeisu failiem jāņem vērā, ka šīm datu kopām jābūt saskatāmām no lietotāja viedokļa. Katru lielumu novērtē kā sarežģītu, vidēju vai vienkāršu.

Praktiski ieteikumi, kā pazīt katru no uzskaitītajiem lielumiem un noteikt tā sarežģītības pakāpi, aplūkoti nodaļās "Sistēmas dati" un "Sistēmas funkcijas". Aizpildot tabulu (1) un veicot tabulā norādītos aprēķinus, iegūst vērtējamās sistēmas funkcionalitātes apjomu funkcijpunktos.



1.attēls. Sistēmas datu un funkciju vērtējums

1.tabula. Funkcijpunktu skaita aprēķināšana

	Sarežģīti	Vidēji	Vienkārši	Kopā
Ievadi	6 * skaits +	4 * skaits +	3 * skaits +	=
Izvadi	7 * skaits +	5 * skaits +	4 * skaits +	=
Iekšēji datu faili	15 * skaits +	10 * skaits +	7 * skaits +	=
Ārēji interfeisa faili	10 * skaits +	7 * skaits +	5 * skaits +	=
Vaicājumi	6 * skaits +	4 * skaits +	3 * skaits +	=
			Funkcijpunktu skaits:	=

2.tabula. Funkcijpunktu skaitīšanai izmantojamie dokumenti

Dokumentu grupa	**Dokumentu nosaukumi un komentārs**
Projekta sagatavošanas dokumenti	Dokumenti: • Sākotnējie projekta priekšlikumi
Projekta specifikācijas	Dokumenti: • Darbības koncepcijas apraksts • Sistēmas/apakšsistēmas (kopā ar aparatūru) prasību specifikācija • Programmatūras prasību specifikācija • Saskarņu prasību specifikācija
Projektējumu dokumentācija	Dokumenti: • Sistēmas/apakšsistēmas (kopā ar aparatūru) projektējuma apraksts • Programmatūras projektējuma apraksts • Saskarņu projektējuma apraksts • Datu bāzu projektējuma apraksts Tāpat var izmantot dažādas projektēšanas procesā izveidotus modeļus un diagrammas: ER diagrammas, sistēmas funkcionalitāti aprakstošas diagrammas (piemēram, GRADE biznesprocesu (BP) vai procesu (PD) diagrammas)
Lietotāja dokumentācija	Dokumenti: • Lietotāja rokasgrāmata • Programmatūras ievadizvades rokasgrāmata

Informācijas avots funkcijpunktu skaitīšanai ir jebkuri sistēmas datus, interfeisus un funkcionalitāti aprakstoši dokumenti (skat. 2.tabula). Jo precīzāk dokuments apraksta sistēmu, jo precīzāks sagaidāms tās funkcionalitātes apjoma novērtējums.

2. Sistēmas robežu noteikšana

Lai sekmīgi identificētu vērtējamās sistēmas iekšējos datu failus, ārējos interfeisa failus, ārējos ievadus, izvadus un vaicājumus, jānosaka vērtējamās sistēmas ārējā robeža. Nosakot sistēmas ārējās robežas, jābalstās uz sistēmas lietotāja skatu.

Ja sistēma atsevišķu uzdevumu veikšanai izmanto citu sistēmu, bet lietotājam šī mijiedarbe nav redzama, tad šī nav uzskatāma par sistēmas robežu. Lai gan no projektējuma viedokļa tā neapšaubāmi tāda ir.

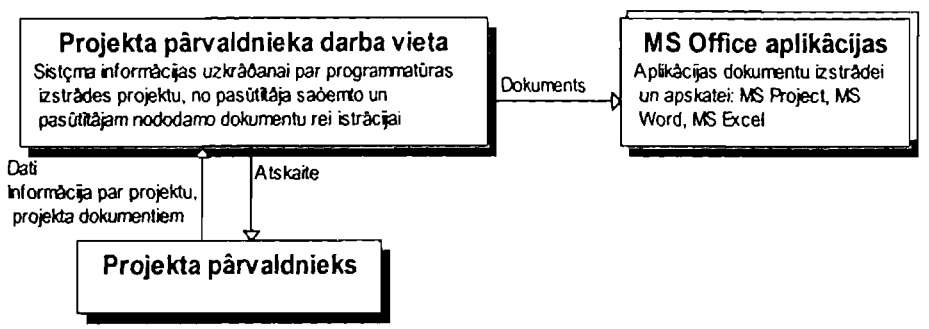
Ja sistēmas projektēšanai izmanto GRADE [GRDP], tad sistēmas ārējās robežas noteikšanai var izmantot augšējā līmeņa komunicējošo objektu diagrammas (CO).

2.1. *Piemērs - Projekta pārvaldnieka darba vieta*

Metodiskajā materiālā aplūkosim aplikācijas piemēru "Projekta pārvaldnieka darba vieta". Sistēmai ir šādas funkcijas:

- Projekta pārvaldnieks ievada informāciju par projektu: projekta nosaukums, realizācijas vide, nodošanas termiņš u.c. un reģistrē projekta apakšprojektus, ja tādi ir.
- Projekta pārvaldnieks reģistrē visus projekta izstrādes gaitā radītos, no pasūtītāja saņemtos un pasūtītājam nododamos dokumentus.
- Projekta pārvaldnieks reģistrē projekta izstrādātājus un pasūtītājus, norādot viņu vārdus, uzvārdus, telefona numurus u.c. informāciju.
- Projekta pasūtītājs var atlasīt apskatei dokumentus par viņu interesējošiem projektiem un apakšprojektiem.
- Dokumentu sagatavošanai un apskatei izmanto MS Office saimes aplikācijas. Dokumentus atver tieši no projekta pārvaldnieka darba vietas.
- Sistēmā iebūvētas dažādas atskaites, kuras var izdrukāt no sistēmas.

Projekta pārvaldnieka darba vietas ārējās sistēmas robeža kā GRADE augšējā līmeņa komunikāciju objektu diagramma attēlota (skat. 2. attēls).



2. attēls. Projekta pārvaldnieka darba vietas sistēmas ārējā robeža

3. Sistēmas dati

Nākamais solis aiz sistēmas robežu definēšanas ir sistēmā ienākošo, apstrādājamo un no sistēmas izvadāmo datu identificēšana. Šai solī saskaita sistēmas iekšējos datu failus un ārējo interfeisu failus un nosaka to sarežģītību.

3.1. *Iekšējie datu faili*

Lai datu kopu varētu uzskatīt par iekšēju datu failu, tai jāatbilst šādām prasībām:

- Dati vai vadības informācija veido loģiski saistītu datu kopu, kas atbilst noteiktām prasībām. Ja sistēmā apstrādājamos datus attēlo ER diagrammā, tad katru entīti uzskata par iekšējo datu failu.
- Datu kopas elementus rada, modificē un iznīcina sistēmas iekšienē.
- Šī datu kopa nav uzskaitīta kā sistēmas ārējais datu fails.

Piemēri iekšējiem datu failiem uzskaitīti tabulā (skat. 3.tabula). Tomēr jāatceras, ka, lai datu kopu uzskatītu par iekšēju datu failu, tai jāatbilst augstākminētajām prasībām.

3.tabula. Piemēri, ko uzskatīt un ko neuzskatīt par iekšējo datu failu

Sistēmas iekšējo datu failu piemēri	Piemēri, ko kļūdaini mēdz uzskatīt par sistēmas iekšējiem datu failiem
Specifiskie sistēmas uzturētie dati. Piemēram, informācija par darbinieku, ar kredītkarti veikto transakciju uzskaitījums u.c. specifiski sistēmā apstrādājami dati.	Pagaidu (temporary) un darba faili vai vairākas viena un tā paša faila kopijas vai faili, kas ieviesti tehnisku iemeslu dēļ.
Sistēmas drošības, paroļu, pārbaužu, HELP, kļūdu apstrādes u.c. dati un parametri, ko apstrādā sistēmas iekšienē.	Faili, kuros atlasīta informācija no citiem iekšējiem datu failiem vai ārējiem interfeisa failiem drukāšanai vai attēlošanai.
Trasēšanas, back-up u.c. vēsturiska informācija, ko sistēma īpaši apstrādā.	Indeksu, relāciju u.c. dienesta faili, ja vien tie nesatur vēl kādu informāciju, izņemot atslēgas.
	Faili, kurus apstrādā citas sistēmas, bet izmanto vērtējamā sistēma.
	Dienesta faili informācijas uzkrāšanai par nepabeigtajām transakcijām, ja vien tie netiek īpaši uzturēti.

3.2. *Ārējie interfeisu faili*

Lai datu kopu varētu uzskatīt par ārēju interfeisu failu, tai jāatbilst šādām prasībām:

- Dati vai vadības informācija veido loģiski saistītu datu kopu, kas atbilst noteiktām prasībām.
- Vērtējamā sistēma nolasa datus no šīs datu kopas.
- Datu kopas elementus rada, modificē un iznīcina citas, ārējas attiecībā pret vērtējamo, sistēmas iekšienē.
- Šī datu kopa nav uzskaitīta kā vērtējamās sistēmas iekšējais datu fails.

Piemēri ārējo interfeisu failiem uzskaitīti tabulā (skat. 4.tabula). Tomēr jāatceras, ka, lai datu kopu uzskatītu par ārējo interfeisu failu, tai jāatbilst augstākminētajām prasībām.

4.tabula. Piemēri, ko uzskatīt un ko neuzskatīt par sistēmas ārējā interfeisa failiem

Sistēmas ārējo interfeisa failu piemēri	Piemēri, ko kļūdaini mēdz uzskatīt par sistēmas ārējā interfeisa failiem
Dati, kas nolasīti no citas aplikācijas. Piemēram, dati kurus vērtējamā aplikācijas nolasa no valsts nozīmes reģistriem	No citas sistēmas saņemti dati, kas modificē vērtējamās sistēmas iekšējos datu failus.
Sistēmas drošības, paroļu, pārbaužu, HELP, kļūdu apstrādes u.c. dati un parametri, ko apstrādā ārpus aplūkojamās sistēmas, bet kurus sistēma izmanto.	Vērtējamās sistēmas apstrādāti dati, kurus izmanto cita sistēma.
Trasēšanas, back-up u.c. vēsturiska informācija, ko īpaši apstrādā cita sistēma, bet vērtējamā sistēma lasa.	Pagaidu (temporary) vai darba faili vai vairākas viena un tā paša faila kopijas vai tehnisku iemeslu dēļ ieviesti faili. Faili, kuros atlasīta informācija no citiem ārējo interfeisu failiem drukāšanai vai attēlošanai. Indeksu, relāciju u.c. dienesta faili, ja vien tie nesatur vēl kādu informāciju, izņemot atslēgas.

3.3. *Iekšējo datu failu un ārējo interfeisu failu sarežģītība*

Katrs identificētais iekšējais datu fails un ārējā interfeisa fails jāklasificē kā vienkāršs, vidējs vai sarežģīts. Klasifikācijai izmanto šādus lielumus:

- Datu elements (saīsinājums DET – data element type) – unikāli, nerekursīvi lauki/atribūti. Piemēram, datu tabulas kolonna.
- Apakšelements (saīsinājums RET – record element type) – datu elementu apakštipi. Apakšelementus parasti attēlo ER diagrammā kā entīšu apakšentītes.

Datu elementu skaitīšanai jāievēro:

- Jāsaskaita datu elementu skaits katram nerekursīvam iekšējam datu failam un ārējam interfeisa failam.

- Jāpieskaita viens DET katram iekšējam datu failam (ārējam interfeisu failam), kuram nepieciešama relācija ar kādu citu iekšēju datu failu (ārējā interfeisa failu). Jāraugās, lai par katru relāciju pieskaitītu DET tikai vienam iekšējam datu failam (ārējā interfeisa failam), nevis abiem, kurus saista attiecīgā relācija.

- Kā vienu DET visai datu grupai skaitiet šādus laukus:

  - Laukus, kas tehnisku iemeslu dēļ dublējas dažādos iekšējos datu failos (ārējo interfeisu failos). Piemēram, atslēgas.

  - Atkārtojošos laukus, kuriem ir vienāds formāts un tips vienas vērtības vairāku variantu glabāšanai. Piemēram, jāglabā peļņas summa par janvāri, februāri utt. līdz decembrim.

Apakšelementu skaitīšanai jāievēro:

- Skaitiet RET katrai iekšējo datu failu (ārējo interfeisa failu) apakšgrupai, ja vien tās nav ieviestas tehnisku apsvērumu dēļ.

- Ja datu elementam nav apakšelementu, skaitiet to kā vienu RET.

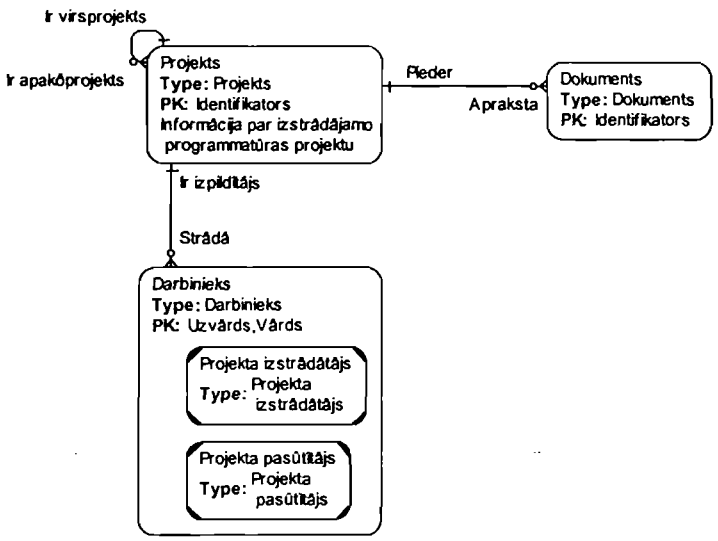
Atkarībā no DET un RET skaita, katru iekšējo datu failu un ārējā interfeisa failu novērtē kā vienkāršu, vidēju vai sarežģītu (skat. 5.tabula).

5.tabula. Ieteikumi sistēmas iekšējo datu failu un ārējo interfeisa failu sarežģītības novērtēšanai

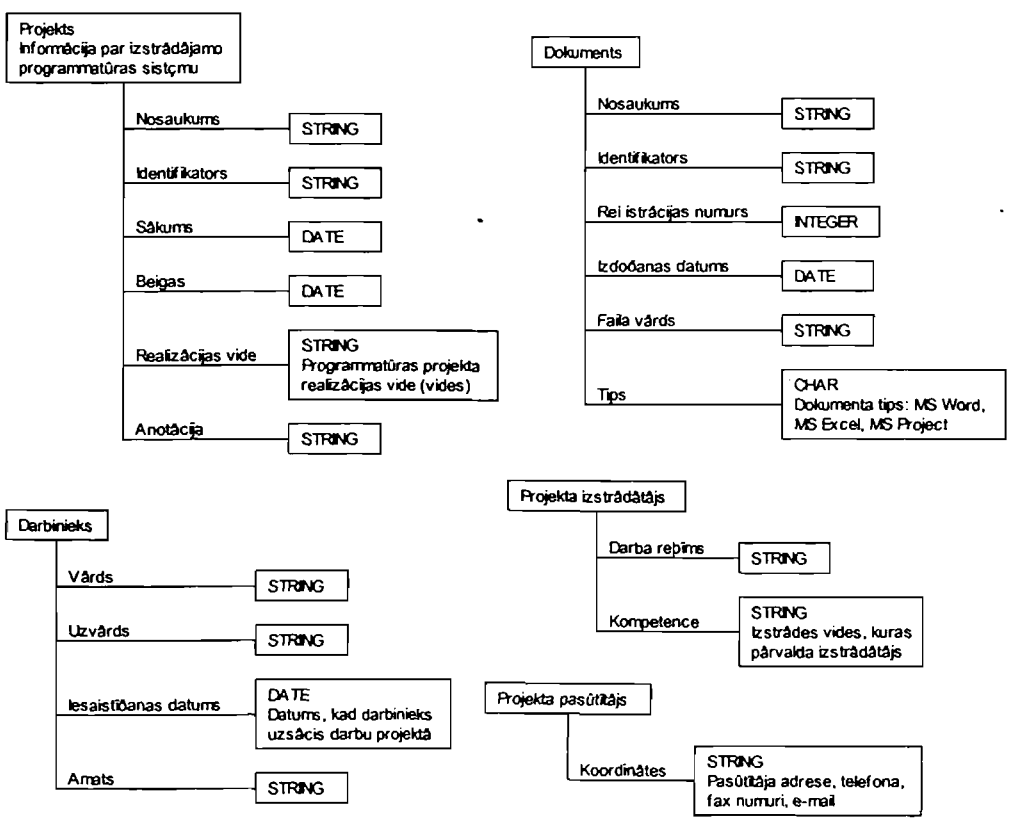
Apakšelementu skaits (RET)	**Datu elementu skaits (DET)**		
	1 - 19	**20 - 50**	**51 +**
1	Vienkārši	Vienkārši	Vidēji
2 - 5	Vienkārši	Vidēji	Sarežģīti
6 +	Vidēji	Sarežģīti	Sarežģīti

3.4. *Piemērs*

Projekta pārvaldnieka darba vietas iekšējo datu failu novērtēšanai izmantosim ER diagrammu (skat. 3.attēls) un entīšu tipus aprakstošas GRADE datu diagrammas (skat. 4.attēls). Aplūkotajā piemērā nav ārējo interfeisu failu.



3.attēls. Projekta pārvaldnieka darba vietas ER modelis



4.attēls. Projekta pārvaldnieka darba vietas uzkrājamos datus aprakstoši datu tipi

Iegūstam šādus rezultātus:

Iekšējs datu fails:	Projekts	
Datu lauki:	Nosaukums	DET
	Identifikators	DET
	Sākums	DET
	Beigas	DET
	Realizācijas vide	DET
	Anotācija	DET
Relācijas:	Ir apakšprojekts..Ir virsprojekts	DET
Apakšelementi:	Nav	RET

Iekšējam datu failam ir 7 DET un 1 RET, tāpēc, saskaņā ar ieteikumiem (5), iekšējais datu fails "Projekts" uzskatāms par vienkāršu.

Iekšējs datu fails:	Dokuments	
Datu lauki:	Nosaukums	DET
	Identifikators	DET
	Reģistrācijas numurs	DET
	Izdošanas datums	DET
	Faila vārds	DET
Relācijas:	Pieder..Apraksta	DET
Apakšelementi:	Nav	RET

Iekšējam datu failam ir 6 DET un 1 RET, tāpēc, saskaņā ar ieteikumiem (5), iekšējais datu fails "Dokuments" uzskatāms par vienkāršu.

Iekšējs datu fails:	Darbinieks	
Datu lauki:	Vārds	DET
	Uzvārds	DET
	Iesaistīšanas datums	DET
	Amats	DET
Apakšelements:	Projekta izstrādātājs	
Datu lauki:	Darba režīms	DET
	Kompetence	DET
Apakšelements:	Projekta pasūtītājs	
Datu lauki:	Koordinātes	DET
Relācijas:	Ir izpildītājs..Strādā	DET
Apakšelementi:	Projekta izstrādātājs	RET
	Projekta pasūtītājs	RET

Iekšējam datu failam ir 8 DET un 2 RET, tāpēc, saskaņā ar ieteikumiem (5), iekšējais datu fails "Darbinieks" uzskatāms par vidēju.

Aizpildām iekšējiem datu failiem un ārējiem interfeisa failiem atbilstošās rindiņas funkcijpunktu skaita aprēķināšanas tabulā (skat. 6.tabula).

6.tabula. Projekta pārvaldnieka darba vietas iekšējo datu failu un ārējo interfeisu failu skaits

	Sarežģīti	Vidēji	Vienkārši	Kopā
Ievadi	6 * skaits +	4 * skaits +	3 * skaits +	=
Izvadi	7 * skaits +	5 * skaits +	4 * skaits +	=
Iekšēji datu faili	15 * 0 +	10 * 1 +	7 * 2 +	= 24
Ārēji interfeisa faili	10 * 0 +	7 * 0 +	5 * 0 +	= 0
Vaicājumi	6 * skaits +	4 * skaits +	3 * skaits +	=
			Nepieskaņotu funkcijpunktu skaits:	=

4. Sistēmas funkcijas

Nākamais solis aiz sistēmā apstrādājamo datu apjoma un sarežģītības novērtēšanas ir sistēmas funkciju un to sarežģītības novērtējums. Sistēmas funkciju novērtējumā ietilpst:

- Ievadu un to sarežģītības novērtējums.
- Izvadu un to sarežģītības novērtējums.
- Vaicājumu un to sarežģītības novērtējums.

4.1. *Ievadi*

Lai sistēmas funkciju uzskatītu par ievadu, datiem, ar kuriem funkcija operē, jāatbilst šādām prasībām:

- Datus jāsaņem no sistēmas ārpuses.
- Datiem jāizmaina kaut viens iekšējais datu fails.

Funkcijai jāatbilst vienai no šādām prasībām:

- Funkcijas loģikai jābūt atšķirīgai no jebkura cita ievada. Šeit vārds 'atšķirīgs' nozīmē atšķirīgus ievadlaukus, algoritmus vai aprēķinus, kā arī atšķirīgu ārējo interfeisa vai iekšējo datu failu lietošanu.
- Datu elementiem jābūt atšķirīgiem no citā ievadā iekļautajiem.
- Funkcijai jābūt noslēgtai, t.i., funkcijas izpildei beidzoties, sistēmai jāpaliek stabilā stāvoklī.

Ārējo interfeisu failu piemēri, uzskaitīti tabulā (skat. 7.tabula). Tomēr jāatceras, lai sistēmas funkciju uzskatītu par ievadu, tai jāatbilst augstākminētajām prasībām.

7.tabula. Piemēri, ko uzskatīt un ko neuzskatīt par sistēmas ievadiem

Ievadu piemēri	Piemēri, ko mēdz uzskatīt par ievadiem, bet kas nav tādi
Ekrāna ievadformas, kas izmaina iekšējos datu failus, vai no kurām sistēmā nonāk vadības informācija.	No citas aplikācijas nolasīti dati, kuri nenonāk nevienā iekšējā datu failā.
Ziņojumi no citām sistēmām, kas īpaši jāapstrādā.	Datu pieprasījuma ievads (tas neizmaina iekšējos datu failus).
Specifiskie sistēmā apstrādājamie dati, kas sistēmā ienāk no citām aplikācijām. Šiem datiem jāmodificē iekšējie datu faili.	Izvēlnes, kurām pakārtotās funkcijas neizmaina iekšējo datu failu saturu.
Datu failu konvertācija.	Log-on ievads, kas neizmaina nevienu iekšēju datu failu.
Jebkura iekšēja datu faila uzturēšana, arī HELP, ziņojumu, parametru u.c.	Vairāki veidi, kā iniciēt vienu un to pašu sistēmas loģiku. Piemēram, vienas un tās pašas funkcijas izsaukšanai piekārtoti funkcionālie taustiņi. Refresh vai Cancel datiem uz ekrāna. Atbildes uz dažādiem diagnostikas paziņojumiem. Datu apmaiņa starp klientu un serveri vienā un tai pašā sistēmā.

4.1.1. Ievadu sarežģītība

Ievadu skaits kopā ar ievadu sarežģītības novērtējumu nosaka sistēmas ievadu ietekmi uz nepieskaņoto funkcijpunktu skaitu. Ievadu sarežģītību nosaka:

- Datu elementu skaits (saīsinājums – DET – data element type), kuri iekļauti ievadā.
- Datu failu skaits, kurus izmanto ievads (saīsinājums – FTR – file types referenced).

Skaitot datu elementus katram ievadam, jāievēro:

➢ Pieskaitiet vienu DET par katru lietotāja pieprasītu, nerekursīvu lauku/atribūtu, tai skaitā arī ārēju atslēgu, kas ienāk sistēmā, parasti iekšēja datu faila modificēšanai.

➢ Pieskaitiet vienu DET par katru lauku, kas ir sistēmas ģenerēts un tiek glabāts kādā iekšējā datu failā. Piemēram, sistēmas ģenerēts konta numurs.

Skaitiet kā vienu datu elementu (DET) visai lauku grupai:

- Loģisku lauku, kuru fiziski glabā vairākos laukos, bet kuru lietotājs pieprasa kā vienu informācijas vienību. Piemēram, lauki "Datums" un "Laiks" glabājas divos tabulas laukos, bet sistēmā vienmēr tiek attēloti kā vienota informācijas vienība.

- Lauku, kas parādās vairākkārt tehnisku iemeslu dēļ.
- Pieskaitiet vienu DET par visiem sistēmas atgriešanās kodiem, kas ziņo par kļūdām vai funkcijas sekmīgu izpildi.

Lai saskaitītu datu failus, uz kuriem referencējas ievads:

- Pieskaitiet vienu FTR katram iekšējam datu failam, kuru modificē ievads.
- Pieskaitiet vienu FTR katram iekšējam datu failam vai ārējam interfeisa failam, kuru nolasa ievada apstrādes laikā.
- Pieskaitiet tikai vienu FTR par katru iekšēju datu failu, kuru gan izmaina, gan nolasa ievada apstrādes laikā.

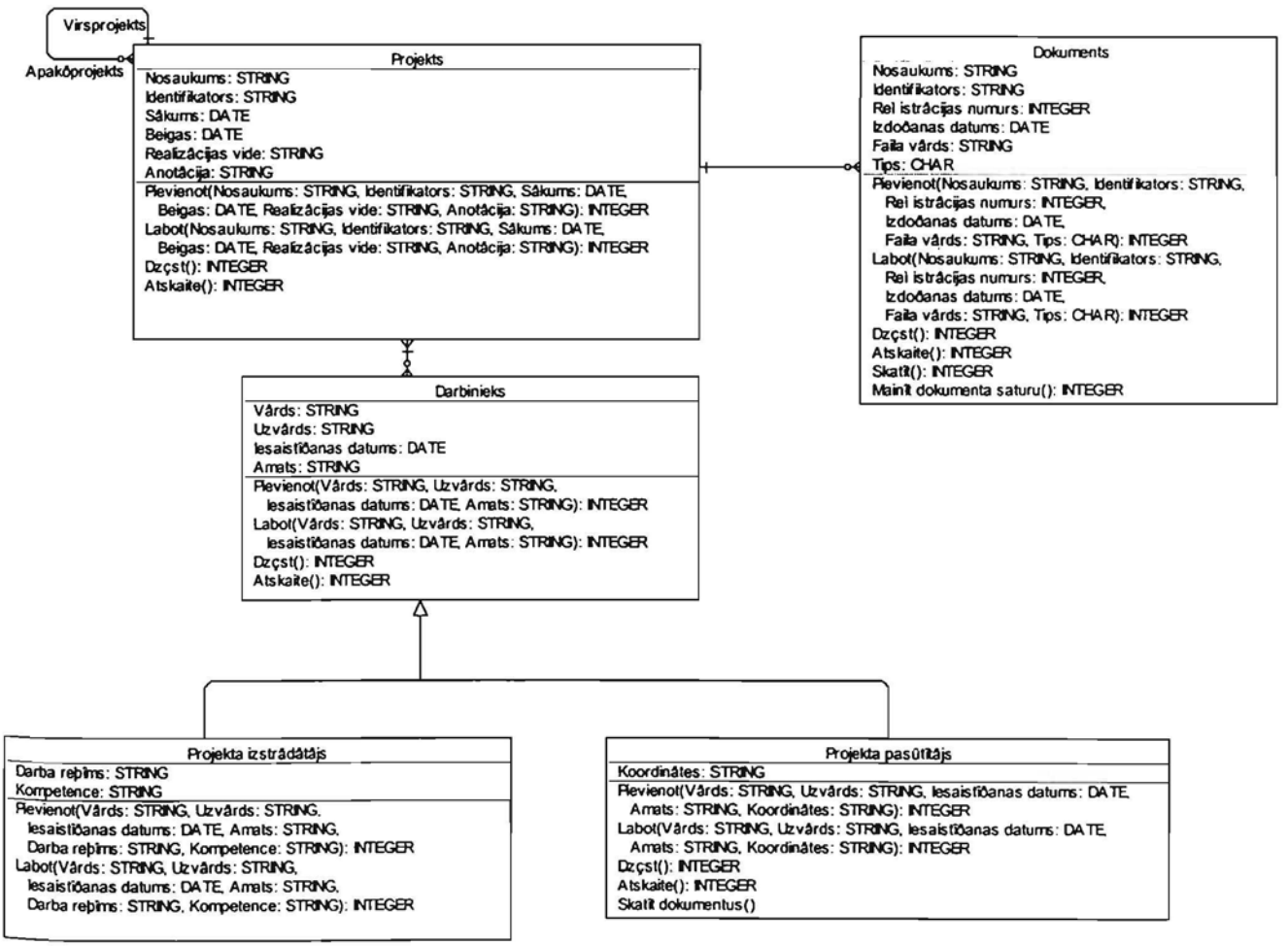
Atkarībā no DET un FTR skaita, katru ievadu novērtē kā vienkāršu, vidēju vai sarežģītu (skat. 8.tabula).

8.tabula. Ieteikumi ievadu sarežģītības noteikšanai

Datu failu skaits, kurus izmanto ievads (FTR)	**Datu elementu skaits (DET)**		
	1 – 4	5 - 15	16 +
0 - 1	Vienkārši	Vienkārši	Vidēji
2 - 3	Vienkārši	Vidēji	Sarežģīti
3 +	Vidēji	Sarežģīti	Sarežģīti

4.1.2. Piemērs

Projekta pārvaldnieka darba vietas ievadu skaita un sarežģītības noteikšanai izmantosim klašu diagrammu (skat. 5.attēls). Ievadu skaita un sarežģītības noteikšanai var izmantot sistēmas funkcionalitāti aprakstošas diagrammas. Ja sistēmas projektēšanai lieto GRADE, var izmantot biznesa procesu diagrammas (BP) un/vai procesu diagrammas (PD).



5.attēls. Projekta pārvaldnieka darba vietas klašu diagramma

Rezultātā iegūstam šādus sistēmas ievadus:

Ievads:	funkcija "Pievienot(Nosaukums: STRING, Identifikators: STRING, Sākums: DATE, Beigas: DATE, Realizācijas vide: STRING, Anotācija: STRING): INTEGER" klasei "Projekts"	
Datu lauki:	Nosaukums	DET
	Identifikators	DET
	Sākums	DET
	Beigas	DET
	Realizācijas vide	DET
	Anotācija	DET
	Funkcijas atgriežamais kļūdas kods	DET
Datu faili:	Projekts	FTR

Ievadam ir 7 DET un 1 FTR, tāpēc, saskaņā ar ieteikumiem (8), šis ievads uzskatāms par vienkāršu.

Ievads:	funkcija "Labot(Nosaukums: STRING, Identifikators: STRING, Sākums: DATE, Beigas: DATE, Realizācijas vide: STRING, Anotācija: STRING): INTEGER" klasei "Projekts"	
Datu lauki:	Nosaukums	DET
	Identifikators	DET
	Sākums	DET
	Beigas	DET
	Realizācijas vide	DET
	Anotācija	DET
	Funkcijas atgriežamais kļūdas kods	DET
Datu faili:	Projekts	FTR

Ievadam ir 7 DET un 1 FTR, tāpēc, saskaņā ar ieteikumiem (8), šis ievads uzskatāms par vienkāršu.

Ievads:	funkcija "Dzēst(): INTEGER" klasei "Projekts"	
Datu lauki:	Primārās atslēgas lauks	DET
	Funkcijas atgriežamais kļūdas kods	DET
Datu faili:	Projekts	FTR

Ievadam ir 2 DET un 1 FTR, tāpēc, saskaņā ar ieteikumiem (8), šis ievads uzskatāms par vienkāršu.

Ievads:	funkcija "Pievienot(Nosaukums: STRING, Identifikators: STRING, Reģistrācijas numurs: INTEGER, Izdošanas datums: DATE, Faila vārds: STRING, Tips: CHAR): INTEGER" klasei "Dokuments"	
Datu lauki:	Nosaukums	DET
	Identifikators	DET
	Reģistrācijas numurs	DET
	Izdošanas datums	DET
	Faila vārds	DET
	Tips	DET
	Funkcijas atgriežamais kļūdas kods	DET
Datu faili:	Dokuments	FTR
	Nolasa informāciju no faila "Projekts", lai piesaistītu dokumentu noteiktam projektam	FTR

Ievadam ir 7 DET un 2 FTR, tāpēc, saskaņā ar ieteikumiem (8), šis ievads uzskatāms par vidēju.

Ievads:	funkcija "Labot(Nosaukums: STRING, Identifikators: STRING, Reģistrācijas numurs: INTEGER, Izdošanas datums: DATE, Faila vārds: STRING, Tips: CHAR): INTEGER" klasei "Dokuments"	
Datu lauki:	Nosaukums	DET
	Identifikators	DET
	Reģistrācijas numurs	DET
	Izdošanas datums	DET
	Faila vārds	DET
	Tips	DET
	Funkcijas atgriežamais kļūdas kods	DET
Datu faili:	Dokuments	FTR

Ievadam ir 7 DET un 1 FTR, tāpēc, saskaņā ar ieteikumiem (8), šis ievads uzskatāms par vienkāršu.

Ievads:	funkcija "Dzēst(): INTEGER" klasei "Dokuments"	
Datu lauki:	Primārās atslēgas lauks	DET
	Funkcijas atgriežamais kļūdas kods	DET
Datu faili:	Dokuments	FTR

Ievadam ir 2 DET un 1 FTR, tāpēc, saskaņā ar ieteikumiem (8), šis ievads uzskatāms par vienkāršu.

Ievads:	funkcija "Pievienot(Vārds: STRING, Uzvārds: STRING, Iesaistīšanas datums: DATE, Amats: STRING, Darba režīms: STRING, Kompetence: STRING): INTEGER" klasei "Projekta izstrādātājs" funkcija "Pievienot(Vārds: STRING, Uzvārds: STRING, Iesaistīšanas datums: DATE, Amats: STRING, Koordinātes: STRING): INTEGER" klasei "Projekta pasūtītājs"	
Datu lauki:	Klase "Darbinieks"	
	Vārds	DET
	Uzvārds	DET
	Iesaistīšanas datums	DET
	Amats	DET
	Apakšklase "Projekta izstrādātājs"	
	Darba režīms	DET
	Kompetence	DET
	Apakšklase "Projekta pasūtītājs"	
	Kompetence	DET
	Funkcijas atgriežamais kļūdas kods	DET
Datu faili:	Darbinieks	FTR
	Nolasa informāciju no faila "Projekts", lai piesaistītu izstrādātāju noteiktam projektam	FTR

Ievadam ir 8 DET un 2 FTR, tāpēc, saskaņā ar ieteikumiem (8), šis ievads uzskatāms par vidēju.

Ievads:	funkcija "Labot(Vārds: STRING, Uzvārds: STRING, Iesaistīšanas datums: DATE, Amats: STRING, Darba režīms: STRING, Kompetence: STRING): INTEGER" klasei "Projekta izstrādātājs" funkcija "Labot(Vārds: STRING, Uzvārds: STRING, Iesaistīšanas datums: DATE, Amats: STRING, Koordinātes: STRING): INTEGER" klasei "Projekta pasūtītājs"	
Datu lauki:	Klase "Darbinieks"	
	Vārds	DET
	Uzvārds	DET
	Iesaistīšanas datums	DET
	Amats	DET
	Apakšklase "Projekta izstrādātājs"	
	Darba režīms	DET
	Kompetence	DET
	Apakšklase "Projekta pasūtītājs"	
	Kompetence	DET
	Funkcijas atgriežamais kļūdas kods	DET
Datu faili:	Darbinieks	FTR

Ievadam ir 8 DET un 1 FTR, tāpēc, saskaņā ar ieteikumiem (8), šis ievads uzskatāms par vienkāršu.

Ievads:	funkcija "Dzēst(): INTEGER" klasei "Darbinieks"	
Datu lauki:	Primārās atslēgas lauks	DET
	Funkcijas atgriežamais kļūdas kods	DET
Datu faili:	Darbinieks	FTR

Ievadam ir 2 DET un 1 FTR, tāpēc, saskaņā ar ieteikumiem (8), šis ievads uzskatāms par vienkāršu.

Aizpildām ievadiem atbilstošās rindiņas funkcijpunktu skaita aprēķināšanas tabulā (skat. 9.tabula).

9.tabula. Projekta pārvaldnieka darba vietas ievadu skaits

	Sarežģīti	Vidēji	Vienkārši	Kopā
Ievadi	6 * 0 +	4 * 2 +	3 * 7 +	= 29
Izvadi	7 * skaits +	5 * skaits +	4 * skaits +	=
Iekšēji datu faili	15 * 0 +	10 * 1 +	7 * 2 +	= 24
Ārēji interfeisa faili	10 * 0 +	7 * 0 +	5 * 0 +	= 0
Vaicājumi	6 * skaits +	4 * skaits +	3 * skaits +	=
			Nepieskaņotu funkcijpunktu skaits:	=

4.2. *Izvadi*

Lai sistēmas funkciju uzskatītu par izvadu, datiem, ar kuriem funkcija operē, jāatbilst šādām prasībām:

- Datus jāpārsūta ārpus sistēmas robežām.

Funkcijai jāatbilst vienai no šādām prasībām:

- Funkcijai jābūt noslēgtai, t.i., tai beidzoties, sistēmai jāpaliek stabilā stāvoklī.
- Funkcijas loģikai jābūt atšķirīgai no jebkura cita izvada. Šeit vārds 'atšķirīgs' nozīmē atšķirīgus ievadlaukus, algoritmus vai aprēķinus, kā arī atšķirīgu ārējo interfeisa un iekšējo datu failu lietošanu.

Izvadu piemēri uzskaitīti tabulā (skat. 10.tabula). Tomēr jāatceras, ka lai funkciju uzskatītu par izvadu, tai jāatbilst augstākminētajām prasībām.

10.tabula. Piemēri, ko uzskatīt un ko neuzskatīt par sistēmas izvadiem

Sistēmas izvadu piemēri	Piemēri, ko mēdz uzskatīt par izvadiem, bet kas tādi nav
Atskaites, kuru sagatavošanai nepieciešami datu apstrādes algoritmi. Piemēram, dažādi mēneša kopsavilkumi.	Vienādas atskaites ar atšķirīgām datu vērtībām.
Dati, faili vai ziņojumi, kurus nosūta citai aplikācijai. Piemēram, atskaite, kuru attēlo MS Word vai MS Excel.	Kopsavilkuma lauki detalizētā atskaitē.
Vairāki datu faili, ja katru failu sagatavo atsevišķi un nosūta citām aplikācijām.	*Refresh* vai *Cancel* funkcijas uz ekrāna rādāmajiem datiem.
Trasēšanas u.c. atskaites, kuras veido datu migrācijas laikā.	Datu pārkārtošana vai pārgrupēšana bez jebkādiem citiem aprēķiniem.
Informatīvi ziņojumi, kas nav kļūdu paziņojumi vai apstiprinājuma vaicājumi.	Datus, kas glabājas sistēmā, bet kurus nolasa cita sistēma.
Atvasināta vai aprēķināta informācija, ko attēlo uz ekrāna.	Vaicājuma atbildes daļa.
Datu grafiskie attēlojumi. Piemēram, dažādas stabiņu vai riņķa diagrammas.	*Help*.
	Vairākus veidus, kā iniciēt vienu un to pašu izvades procesu.
	Kļūdu ziņojums ievaddatu pārbaudes laikā.
	Apstiprinājuma ziņojums, ka datu apstrāde beigusies utml.
	Apstiprinājuma ziņojums. Piemēram, brīdinājums par raksta izmešanu utml.
	Identiski dati, kurus nosūta dažādām aplikācijām (tos uzskaita tikai vienu reizi).
	Datu apmaiņa starp pakešu un *on-line* režīmu vienā un tai pašā aplikācijā.
	Datu apmaiņa starp klientu un serveri vienā un tai pašā aplikācijā.

4.2.1. Izvadu sarežģītība

Izvadu skaits kopā ar izvadu sarežģītības novērtējumu nosaka sistēmas izvadu ietekmi uz nepieskaņoto funkcijpunktu skaitu. Izvadu sarežģītību nosaka:

- Datu elementu (saīsinājums – DET – data element types) skaits, kas iekļauti izvadā.
- Datu failu skaits, kurus izmanto izvads (saīsinājums – FTR – file types referenced).

Skaitot datu elementus katram izvadam, jāievēro:

- Pieskaitiet vienu DET par katru lietotāja pieprasītu nerekursīvu lauku/atribūtu, kas parādās izvadā.

- Neskaitīt sistēmas automātiski ģenerētos atribūtus. Piemēram, lappuses numurus.
- Neskaitiet virsrakstus, kolonnu virsrakstus vai lauku virsrakstus.

Skaitiet tikai vienu DET visai lauku grupai:

- Tiem laukiem, kurus fiziski glabā vairākos laukos, bet kuru lietotājs pieprasa kā vienu informācijas vienību.
- Katrai datu sērijai grafiskajā attēlojumā. Piemēram, riņķa diagrammai ir divi DET – viens par iezīmēm, otrs par skaitliskajām vērtībām.

Lai saskaitītu datu failus, uz kuriem referencējas izvads:

- Pieskaitiet vienu FTR katram iekšējam datu failam vai ārējam interfeisa failam, kuru nolasa izvada sagatavošanas laikā.

Atkarībā no DET un FTR skaita, katru izvadu novērtē kā vienkāršu, vidēju vai sarežģītu (skat. 11.tabula).

11.tabula. Ieteikumi izvadu sarežģītības noteikšanai

Datu failu skaits, uz kuriem referencējas (FTR)	**Datu elementu skaits (DET)**		
	1 - 5	6 - 19	20 +
0 - 1	Vienkārši	Vienkārši	Vidēji
2 - 3	Vienkārši	Vidēji	Sarežģīti
4 +	Vidēji	Sarežģīti	Sarežģīti

4.2.2. Piemērs

Projekta pārvaldnieka darba vietas izvadu skaita un sarežģītības noteikšanai izmantosim klašu diagrammu (skat. 5.attēls). Izvadu skaita un sarežģītības noteikšanai var izmantot arī sistēmas funkcionalitāti aprakstošas diagrammas. Ja sistēmas projektēšanai izmanto GRADE, var izmantot biznesa procesu diagrammas (BP) un/vai procesu diagrammas (PD).

Rezultātā iegūstam šādus sistēmas izvadus:

Izvads:	funkcija "Atskaite(): INTEGER" klasei "Projekts" Funkcijas izpildes rezultātā izdrukā atskaiti par visiem projektiem un apakšprojektiem.	
Datu lauki:	Nosaukums	DET
	Identifikators	DET
	Sākums	DET
	Beigas	DET
	Realizācijas vide	DET
	Anotācija	DET
	Virsprojekta nosaukums	DET
Datu faili:	Projekts	FTR

Izvadam ir 7 DET un 1 FTR, tāpēc, saskaņā ar ieteikumiem (11), šis izvads uzskatāms par vienkāršu.

Izvads:	funkcija "Atskaite(): INTEGER" klasei "Dokuments" Funkcijas izpildes rezultātā izdrukā atskaiti par visiem dokumentiem un projektiem, kādiem katrs dokuments piesaistīts.	
Datu lauki:	Nosaukums	DET
	Identifikators	DET
	Reģistrācijas numurs	DET
	Izdošanas datums	DET
	Faila vārds	DET
	Tips	DET
	Projekta nosaukums	DET
	Projekta identifikators	DET
Datu faili:	Dokuments	FTR
	Projekts	FTR

Izvadam ir 8 DET un 2 FTR, tāpēc, saskaņā ar ieteikumiem (11), šis izvads uzskatāms par vidēju.

Izvads:	funkcija "Atskaite(): INTEGER" klasei "Projekta izstrādātājs" funkcija "Atskaite(): INTEGER" klasei "Projekta pasūtītājs" Funkciju izpildes rezultātā rodas divas atskaites attiecīgi par projekta izstrādātājiem vai projekta pasūtītājiem. Abas atskaites skaitīsim kā vienu, ievērojot to, ka tās ir gandrīz vienādas (lielākā daļa informācijas pārklājas).	
Datu lauki:	Klase "Darbinieks"	
	Vārds	DET
	Uzvārds	DET
	Iesaistīšanas datums	DET
	Amats	DET
	Apakšklase "Projekta izstrādātājs"	
	Darba režīms	DET
	Kompetence	DET
	Apakšklase "Projekta pasūtītājs"	
	Kompetence	DET
	Projekta nosaukums	DET
	Projekta identifikators	DET
Datu faili:	Darbinieks	FTR
	Projekts	FTR

Izvadam ir 9 DET un 2 FTR, tāpēc, saskaņā ar ieteikumiem (11), šis izvads uzskatāms par vidēju.

Aizpildām izvadiem atbilstošās rindiņas funkcijpunktu skaita aprēķināšanas tabulā (skat. 12.tabula).

12.tabula. Projekta pārvaldnieka darba vietas izvadu skaits

	Sarežģīti	Vidēji	Vienkārši	Kopā
Ievadi	6 * 0 +	4 * 2 +	3 * 7 +	= 29
Izvadi	7 * 0 +	5 * 2 +	4 * 1 +	= 14
Iekšēji datu faili	15 * 0 +	10 * 1 +	7 * 2 +	= 24
Ārēji interfeisa faili	10 * 0 +	7 * 0 +	5 * 0 +	= 0
Vaicājumi	6 * skaits +	4 * skaits +	3 * skaits +	=
			Nepieskaņotu funkcijpunktu skaits:	=

4.3. *Vaicājumi*

Vaicājumam ir divas daļas: ievaddaļa un rezultāts. Lai sistēmas funkciju uzskatītu par vaicājumu, tai jāatbilst šādām prasībām:

- Ievadītajam pieprasījumam jānāk no sistēmas ārpuses.
- Rezultātiem jāiziet ārpus sistēmas.
- Datus jānolasa no viena vai vairākiem iekšējiem datu vai ārējā interfeisa failiem.
- Rezultāti nesatur aprēķinus.
- Pēc pieprasījuma izpildes sistēmai jāpaliek stabilā stāvoklī.
- Neizmaina nevienu iekšēju datu failu.

Vaicājumu piemēri doti tabulā (skat. 13.tabula). Jāatceras, lai sistēmas funkciju uzskatītu par vaicājumu, tai jāatbilst augstākminētajām prasībām.

13.tabula. Vaicājumu piemēri

Vaicājumu piemēri	Piemēri, ko mēdz uzskatīt par vaicājumu, bet kas nav tāds
Sistēmas specifiskie dati, kurus pēc pieprasījuma nolasa no viena vai vairākiem iekšējiem datu vai ārējā interfeisa failiem.	Vairāki veidi, kā izsauc vienu un to pašu sistēmas loģiku. Tas jāskaita tikai vienu reizi.
Lietotāja funkcijas, piemēram, *View*, *Lookup*, *Display*, *Browse*.	Vaicājumi, kas pieejami no vairākām vietām.
Iebūvētie vaicājumi. Piemēram, datu atlase, lai izvēlētos, kuru ierakstu labot vai izmest.	Izvēlnes punkti, kuru izvēle neatgriež datus.
Sistēmas uzturētie dati, parametri un *setup* informācija, ja vien tā netiek izmantota aprēķinos.	Datu aprēķini.
Logon informācijas ievads, ja vien ievadītā informācija neizmaina nevienu iekšēju datu failu.	Datu pārkārtošana bez to pārlasīšanas.
Katru *Help* līmeni ieskaitīta kā vienu vaicājumu. Ja ir sistēmas *Help*, *Help* par katru ievadformu un par katru ievadlauku, tad *Help* sistēma jāvērtē kā 3 vaicājumi.	Atbildes ziņojumlodziņos.
Datu ieguve pa sakaru kanāliem.	Sistēmas *On-line* dokumentācija.
Meila nolasīšana no *Mailbox*.	Datu apmaiņa starp klientu un serveri vienā un tai pašā aplikācijā.
Listbox u.c. saraksti, kas atgriež apstrādājamo datu sarakstus.	Rezultāti, kurus neiegūst no iekšējiem datu failiem. Piemēram, *hard-coded* dati.

4.3.1. Vaicājumu sarežģītība

Vaicājumu skaits kopā ar to sarežģītības novērtējumu sastāda kopīgo vaicājumu ietekmi uz vērtējamās sistēmas funkcijpunktu skaitu. Vaicājumu sarežģītību nosaka:

- Vaicājumā izmantoto datu elementu (saīsinājums – DET – data element types) skaits.
- Datu failu skaits, kurus izmanto vaicājums (saīsinājums – FTR – file types referenced).

Skaitot datu elementus vaicājumiem, jāievēro:

- Pieskaitiet vienu DET par katru lietotāja pieprasītu nerekursīvu lauku/atribūtu, kas ienāk sistēmā no ārpuses vaicājuma ievaddaļā.
- Pieskaitiet vienu DET katram laukam, kurā ievada datu atlases kritēriju.
- Pieskaitiet vienu DET par katru lietotāja pieprasītu nerekursīvu lauku/atribūtu, kas iziet ārpus sistēmas robežām vaicājuma izvaddaļā.
- Neskaitiet sistēmas ģenerētos un uzturētos mainīgos. Piemēram, lappušu numurus u.c.
- Neskaitiet virsrakstus, tai skaitā kolonnu virsrakstus.

Skaitiet vienu DET visai lauku grupai:

- Uz visām sistēmas atbildēm par kļūdām vai to, ka darbs pabeigts.
- Par spēju noteikt, ka vaicājums izpildās.
- Par katru funkcionālo taustiņu vai komandrindu, kas nosaka, ka jāsākas darbībai.
- Par loģisku lauku, kuru fiziski glabā vairākos laukos, bet kuru lietotājs pieprasa kā vienu informācijas vienību.
- Par lauku, kas satur vienādu informāciju, bet tehnisku iemeslu dēļ dublējas vairāk nekā vienā laukā.

Lai saskaitītu datu failus, uz kuriem referencējas izvads:

- Pieskaitiet vienu FTR katram iekšējam datu vai ārējā interfeisa failam, kuru nolasa vaicājuma apstrādes laikā.

Atkarībā no DET un FTR skaita, katru vaicājumu novērtē kā vienkāršu, vidēju vai sarežģītu (skat. 14.tabula).

14.tabula. Ieteikumi vaicājumu sarežģītības novērtēšanai

Datu tipu vai failu skaits, uz kuriem referencējas (FTR)	**Datu elementu skaits (DET)**		
	1 - 5	6 - 19	20 +
0 - 1	Vienkārši	Vienkārši	Vidēji
2 - 3	Vienkārši	Vidēji	Sarežģīti
4 +	Vidēji	Sarežģīti	Sarežģīti

4.3.2. Piemērs

Projekta pārvaldnieka darba vietas vaicājumu skaita un sarežģītības noteikšanai izmantosim klašu diagrammu (skat. 5.attēls). Tāpat vaicājumu skaita un sarežģītības noteikšanai var izmantot arī sistēmas funkcionalitāti aprakstošas diagrammas. Ja sistēmas projektēšanai izmanto GRADE, var izmantot biznesa procesu diagrammas (BP) un/vai procesu diagrammas (PD).

Rezultātā iegūstam šādus sistēmas vaicājumus:

Vaicājums:	funkcija "Skatīt dokumentus()" klasei "Projekta pasūtītājs" Funkcijas izpildes rezultātā parādās visu to dokumentu saraksts, par kuriem būtu interese konkrētajam projekta pasūtītājam, no šī saraksta var izvēlēties un skatīties dokumentus.	
Datu lauki vaicājuma ievaddaļā:	Iekļaujamie projekti	DET
Datu lauki vaicājuma izvaddaļā:	Dokumenta nosaukums	DET
	Dokumenta identifikators	DET
	Projekta nosaukums, kuram piesaistīts konkrētais dokuments	DET
Datu faili:	Dokuments	FTR
	Projekts	FTR

Vaicājumam ir 4 DET un 2 FTR, tāpēc, saskaņā ar ieteikumiem (14), šis vaicājums uzskatāms par vienkāršu.

Aizpildām izvadiem atbilstošās rindiņas funkcijpunktu skaita aprēķināšanas tabulā (skat. 15.tabula). Sasummējot iegūstam, ka sistēmas funkcionalitāte ir 70 funkcijpunkti.

15.tabula. Projekta pārvaldnieka darba vietas vaicājumu skaits

	Sarežģīti	Vidēji	Vienkārši	Kopā
Ievadi	6 * 0 +	4 * 2 +	3 * 7 +	= 29
Izvadi	7 * 0 +	5 * 2 +	4 * 1 +	= 14
Iekšēji datu faili	15 * 0 +	10 * 1 +	7 * 2 +	= 24
Ārēji interfeisa faili	10 * 0 +	7 * 0 +	5 * 0 +	= 0
Vaicājumi	6 * 0 +	4 * 0 +	3 * 1 +	= 3
			Nepieskaņotu funkcijpunktu skaits:	= 70

5. Attēli

6. Tabulas

7. Atsauces

[COC2] – C.Abts, B.Boehm, B.Clark, S.Devnani-Chulani. COCOMO II Model Definition Manual,_University of Southern California, 68 p.

[GARM] – D. Garmus, D. Herron. Measuring the Software Process._Prentice-Hall, Inc., USA, 1996.

II Pielikums. Mērījumu datu savākšanas formas

Projekta izstrāde

Informācija par projektu			
Projekta nosaukums			
Projekta vadītājs			
Formas aizpildīšanas datums			
Projekta uzsākšanas datums		Projekta beigu datums	

Izstrādes vide	
Projekta tips	
Uzņēmējdarbības sfēra, kurai programmatūra tiek izstrādāta	
Izstrādājamās programmatūras apjoms	

Plānotā projekta izstrādes darbietilpība (cilvēkdienas)	
Projekta izstrādes reālā darbietilpība (cilvēkdienas)	
Plānotais projekta izstrādes kalendārais laiks (mēneši)	
Projekta izstrādes reālais kalendārais laiks (mēneši)	

Dzīves cikla process	**Plānotā darbietilpība (cilvēkdienas)**	**Reālā darbietilpība (cilvēkdienas)**	**Plānotais kalendārais laiks (mēneši)**	**Reālais kalendārais laiks (mēneši)**
Koncepcijas izstrāde				
Prasību specificēšana				
Projektēšana				
Programmēšana				
Testēšana				
Ieviešana				
Uzturēšana				
Reinženierija				
Darba vides nodrošināšana				
Konfigurācijas pārvaldība				
Dokumentēšana				
Kvalitātes nodrošināšana				
Pārvaldīšana				
Mācības				
Laika zudumi				

Projekta darbietilpība

Projekta nosaukums	
Projekta vadītājs	
Formas aizpildīšanas datums	

Dzīves cikla process	Cilvēkdienas
Koncepcijas izstrāde	
Prasību specificēšana	
Projektēšana	
Programmēšana	
Testēšana	
Ieviešana	
Uzturēšana	
Reinženierija	
Darba vides nodrošināšana	
Konfigurācijas pārvaldība	
Dokumentēšana	
Kvalitātes nodrošināšana	
Pārvaldīšana	
Mācības	
Laika zudumi	

Par problēmziņojumu uzkrājamie dati

Nr.	Nosaukums	Raksturojums	Aizpildītājs	Kad aizpilda
1.	Problēmas identifikators	Unikāls problēmas identifikators	Problēmas reģistrētājs	Problēmu reģistrējot
2.	Problēmas apraksts	Problēmas apraksts brīvā tekstā, kam tomēr nevajadzētu būt pārāk garam (līdz 200 simboliem)	Problēmas reģistrētājs	Problēmu reģistrējot
3.	Problēmas tips	Kļūda, uzlabojums	Problēmas reģistrētājs	Problēmu reģistrējot
4.	Fiksēšanas datums		Problēmas reģistrētājs	Problēmu reģistrējot
5.	Modulis	Programmatūras modulis, kurā problēma atklāta	Problēmas reģistrētājs	Problēmu reģistrējot
6.	Versija	Tā attiecīgā moduļa versija, kurā šī problēma atklāta	Problēmas reģistrētājs	Problēmu reģistrējot
7.	Atrašanās vieta	Problēmas atrašanās vieta; kā tai piekļūt, lai atkārtotu, ieraudzītu utml.	Problēmas reģistrētājs	Problēmu reģistrējot
8.	Autors	Problēmziņojuma autors	Problēmas reģistrētājs	Problēmu reģistrējot
9.	Atbildīgais izpildītājs	Projekta izstrādātājs, kas atbildīgs par attiecīgās problēmas risināšanu	Projekta izpildītājs, kurš atbildīgs par darbu plānošanu.	Plānojot problēmas risinājumu
10.	Problēmas stāvoklis	Patreizējais statuss. Iespējamie problēmas stāvokļi: Atvērta, Atlikta, Atsaukta (autors pārdomājis), Noraidīta (projekta izpildītāji neuzskata par problēmu), Novērsta, Precizējama, Nododama ekspluatācijā	Problēmas reģistrētājs – tikai "Atvērta", "Atsaukta" Atbildīgais izpildītājs – jebkurš statuss	"Atvērta" – problēmu reģistrējot, "Nododama ekspluatācijā" – nododot pasūtītājam, pārējie – pēc vajadzības
11.	Stāvokļa datums	Datums, kad problēmas stāvoklis mainīts	Darbinieks, kurš aizpildījis lauku "Problēmas statuss"	Tad, kad stāvoklis tiek izmainīts
12.	Testēšanas datums	Datums, kad problēma notestēta	Testētājs	Beidzot testēšanu
13.	Testēšanas atskaite	Saite uz testēšanas atskaiti	Testētājs	Beidzot testēšanu
14.	Versija	Programmatūras versija/laidiens, ar kuru attiecīgā problēma nodota ekspluatācijā	Atbildīgais izpildītājs	Nododot ekspluatācijā

III Pielikums. Uzņēmuma ALFA Mērījumu programmas plāns

Metriku kopa	Vienība	Reģistrācijas biežums	Atbildīgais par reģistrāciju	Apkopošanas biežums	Vērtības piemērs	Datu ievades forma	Pieejamība projektā
Informācijas par projektu	Teksts	Projektu uzsākot	Projekta vadītājs	Projektu uzsākot		Informācija par projektu (skat. II pielikumu)	Jā
Izstrādājamās programmatūras apjoms (jaunas programmatūras izstrādes projektam) Uzturamās programmatūras apjoms (uzturēšanas projektam)	Funkcijpunkti, programmrindiņas	Projektu uzsākot	Mērījumu programmas vadītājs	Projektu uzsākot	1200 funkcijpunkti	Informācija par projektu (skat. II pielikumu)	Nē
Plānotā projekta izstrādes darbietilpība	Cilvēkdienas	Projektu uzsākot, mainot projekta plānu	Projekta vadītājs	Projektu uzsākot, projektu beidzot	970 cilvēkdienas	Informācija par projektu (skat. II pielikumu)	Jā
Projekta izstrādes reālā darbietilpība	Cilvēkdienas	Projektu beidzot	Projekta vadītājs	Projektu beidzot	1300 cilvēkdienas	Informācija par projektu (skat. II pielikumu)	Jā
Plānotā projekta katra dzīves cikla darbietilpība	Cilvēkdienas	Projektu uzsākot, mainot projekta plānu	Projekta vadītājs	Projektu uzsākot, regulāri projekta gaitā (piemēram, reizi mēnesī), projektu beidzot	Projektēšana – 68 cilvēkdienas Kodēšana – 102 cilvēkdienas Testēšana – 180 cilvēkdienas	Informācija par projektu (skat. II pielikumu)	Nē
Katra dzīves cikla reālā darbietilpība	Cilvēkdienas	Katru dienu	Projekta vadītājs	Regulāri projekta gaitā (piemēram, reizi mēnesī), projektu beidzot	2002. gada 5. jūlijā: Projektēšana – 5 cilvēkdienas Kodēšana – 0 cilvēkdienas Testēšana – 0 cilvēkdienas	Projektu beidzot - Informācija par projektu (skat. II pielikumu) Projekta gaitā – Informācija par darbietilpību (skat. II pielikumu)	Nē
Plānotais projekta izstrādes kalendārais	Mēneši	Projektu uzsākot, mainot projekta plānu	Projekta vadītājs	Projektu uzsākot, regulāri projekta	12 mēneši	Informācija par projektu (skat. II	Jā

Metriku kopa	Vienība	Reģistrācijas biežums	Atbildīgais par reģistrāciju	Apkopošanas biežums	Vērtības piemērs	Datu ievades forma	Pieejamība projektā
laiks				gaitā (piemēram, reizi mēnesī), projektu beidzot		pielikumu)	
Projekta izstrādes reālais kalendārais laiks	Mēneši	Projektu beidzot	Projekta vadītājs	Projektu beidzot	14 mēneši	Informācija par projektu (skat. II pielikumu)	Jā
Plānotais projekta katra dzīves cikla kalendārais laiks	Mēneši	Projektu uzsākot, mainot projekta plānu	Projekta vadītājs	Projektu uzsākot, regulāri projekta gaitā (piemēram, reizi mēnesī), projektu beidzot	Projektēšana – 1 mēnesi Kodēšana – 2 mēnešus Testēšana – 1 mēnesi	Informācija par projektu (skat. II pielikumu)	Jā
Reālais katra dzīves cikla procesa kalendārais laiks	Mēneši	Attiecīgajam etapam beidzoties Piemēram, programmatūras arhitektūras projektēšanas process ir beidzies, kad ir izstrādāts un nodots pasūtītājam Programmatūras arhitektūras projektējums, Datu bāzes arhitektūras projektējums u.c. projektējumi.	Projekta vadītājs	Regulāri projekta gaitā (piemēram, reizi mēnesī), projektu beidzot	2002. gada 5. jūlijā: Projektēšana – 1.2 mēneši	Informācija par projektu (skat. II pielikumu)	Jā
Informācija par atklātajām kļūdām		Kļūdu atklājot, kļūdas novēršanas gaitā	Projekta vadītājs	Regulāri projekta gaitā (piemēram, reizi mēnesī), projektu beidzot		Skat. II pielikumu	Jā