LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

VEIKTSPĒJAS UZLABOŠANA INFORMĀCIJAS SISTĒMĀS AR DAUDZLĪMEŅU ARHITEKTŪRU

BAKALAURA DARBS

Autors: Augusts Ošiņš
Stud. apl. ao08090
Darba vadītājs: M. inž. Dainis Inkins

RĪGA 2014

Saturs

ATTĒLU SARAKSTS

ANOTĀCIJA

Programmatūras izplatība mūsdienās ļoti strauji aug, tā tiek pielietota visdažādākajiem mērķiem, no izklaides līdz finanšu un ražošanas pārvaldīšanu. Izmantojot informācijas sistēmas darījumprocesos, tiek sagaidīts, ka tā atvieglos un paātrinās veicamos darbus. Šī iemesla dēļ ir svarīgi, lai izstrādātās sistēmas būtu ātrdarbīgas.

Darbā apskatītas informāciju sistēmu arhitektūras, to ātrdarbības problēmas un metodes, kā šīs problēmas novērtēt un risināt. Izmantojot iegūto informāciju izvēlēta Metode R, ar kuru veikts KAVIS (Klienti attiecību vadības informācijas sistēma) sistēmas novērtējums un ātrdarbības uzlabošanas pasākumi. Pēc labojumu veikšanas, novērtēta Metodes R efektivitāte un piemērotība sistēmas ātrdarbības noteikšanai.

Atslēgas vārdi: informācijas sistēmas, ātrdarbība, Metode R

ABSTRACT

The prevalence of software is increasing quickly, it is use for many different purposes from entertainment to finance and manufacturing management. When using software in business it is expected that it will facilitate and expedite the process. As a result it is important that the developed systems have good performance.

This work evaluates different information system architectures, their performance issues and methods of evaluating and solving the issues. Method R was chosen in order to evaluate KAVIS (client relation management information system) system's performance and enact improvements. After improving KAVIS system's performance, the effectiveness and suitability of Method R was evaluated.

Keywords: information systems, performance, Method R

IEVADS

Mūsdienu informācijas laikmetā lietojumprogrammu un informācijas sistēmu izplatība un pielietojums pieaug ārkārtīgi strauji. Palielinoties izplatībai, aug arī tas, ko lietotāji sagaida, ka informācijas sistēma spēs veikt un kādu informāciju sniegt un apstrādāt. Informāciju sistēmas kļūst arvien sarežģītākas un apjomīgākas, nepieciešami lielāki resursi gan to izstrādē, gan uzturēšanā, un, neskatoties uz straujo tehnoloģiju attīstību, to ātrdarbības nodrošināšanai nepieciešams veltīt pastiprinātu uzmanību. Šī procesa atvieglošanai ir izstrādātas dažādas metodes, ar kuru palīdzību iespējams noteikt informācijas sistēmas ātrdarbības rādītājus, uz kuriem balstoties pēc tam ir iespējams veikt labojumus, ja tas nepieciešams.

Neliela apjoma sistēmām ātrdarbības jautājums ir mazāk aktuāls, salīdzinot ar lielām, ilgtermiņā izmantotām sistēmām.

Līdz ar to autors vēlas izvirzīt hipotēzi, ka ilgtermiņā izmantotas informācijas sistēmas visaptverošs ātrdarbības novērtējums nav efektīvākā metode tās veiktspējas uzlabošanai.

Tēmas aktualitāti nosaka sekojoši faktori:

- Informācijas sistēmu izplatības pieaugums
- Pieaugošas veiktspējas prasības
- Ilgtermiņā lietotu IS skaita pieaugums

Darba mērķis:

Izpētīt daudzlīmeņu informācijas sistēmu ātrdarbības problēmu cēloņus un ātrdarbības novērtēšanas metodes, un izmantot iegūto informāciju SIA ZZ Dats informācijas sistēmas KAVIS ātrdarbības uzlabošanā.

Darba uzdevumi:

- Izpētīt un aprakstīt plašāk izmantotās informāciju sistēmu arhitektūras;
- Noskaidrot populārākos ātrdarbības problēmu cēloņus informācijas sistēmās, risinājumu metodes un metodes darbu organizēšanai;
- Izvēlēties vienu no apskatītajām ātrdarbības novērtēšanas metodēm un to izmantot KAVIS ātrdarbības uzlabošanai;
- Izdarīt secinājumus.

Pirmajā nodaļā apskatīti un izpētīti informācijas sistēmu veidi, kādu problēmu risināšanai un vajadzību apmierināšanai tie izmantojami. Detalizētāk apskatīti daudzlīmeņu sistēmu veidi, to priekšrocības un trūkumus.

Otrajā nodaļā apskatītas dažādas problēmas, kas saistītas ar informācijas sistēmu veiktspējas samazināšanos. Aprakstīti cēloņi un situācijas, kurās attiecīgās problēmas

izpaužas un metodes, kā uzlabot sistēmas ātrdarbību. Šajā nodaļā apskatīti dažādi testēšanas veidi, kas izmantojami sistēmas veiktspējas novērtēšanai un apskatītas trīs metodes, kā organizēt informācijas sistēmas veiktspējas testēšanu.

Trešajā nodaļā aprakstīta KAVIS sistēmas veiktspējas uzlabošana. Sniegta vispārēja informācija par KAVIS sistēmu, tās izstrādes sākumu, mērķi un esošo stāvokli. Izvēlēta viena no aprakstītajām veiktspējas novērtēšanas metodēm un tās iegūto rezultātu pielietojumus, lai uzlabotu KAVIS Infrastruktūras nodevas moduļa ātrdarbību.

Darbs kopumā izklāstīts 40 lappusēs, tajā iekļautas 3 tabulas un 11 attēli

Darbā izmantoti 24 literatūras avoti un pievienoti 3 pielikumi.

IZMANTOTIE TERMINI

IS – Informācijas sistēma

KAVIS – Klientu attiecību vadības informācijas sistēma

BUVE – Būvniecības pārraudzības un reģistrācijas sistēma

RD VIS – Rīgas pilsētas pašvaldības Vienotā informācijas sistēma

GB – gigabaits, datu apjoms, $2^{30} \approx 10^{9}$ baiti.

TB – terabaits, datu apjoms, $2^{40} \approx 10^{12}$ baiti.

SQL – *Structured Query Language* (Strukturēta Vaicājumu Valoda), vaicājumu valoda, kura tiek izmantota relāciju datubāzu datu pieprasīšanai un manipulācijai

1. INFORMĀCIJAS SISTĒMAS

1.1. Informācijas sistēmu veidi

Mūsdienu tehnoloģiskajā laikmetā aizvien plašākas kļūst lietotāju vēlmes un vajadzības, kā rezultātā arī izstrādātajai programmatūrai un informācijas sistēmām ir jākļūst daudzveidīgākām. Ikvienai problēmai ir iespējami dažādi risinājumi un pieejas, nepieciešami dažādi mērogi, no pāris lietotājiem līdz daudziem miljoniem. Gadu gaitā veidojušās un attīstījušās daudzas un dažādas pieejas, kā veidot un organizēt lietojumprogrammas un informācijas sistēmas. Vienu no veidiem, kā klasificēt dažādās informācijas sistēmas piedāvā M. Šava un D. Garlans:

- Datu plūsmas arhitektūras
  - Caurules un filtri
  - Partijas kārtas apstrāde
- Neatkarīgas komponentes
  - Klientu-serveru sistēmas
  - Paralēlas saziņas procesi
- Virtuālās mašīnas
  - Interpretētāji
  - Noteikumu-bāzētas sistēmas
- Glabātuvju arhitektūras
  - Datubāzes
  - Hiperteksta sistēmas
  - Tāfeles (mākslīgajam intelektam)
- Slāņveida arhitektūras. [7, 439. lpp.]

Būtiski ņemt vērā, ka minētās klasifikācijas ir uztveramas kā vadlīnijas izstrādes procesam, nevis kā stingi noteikts plāns. Veidojot lietojumprogrammas un lielākas informācijas sistēmas par pamatu arhitektūrai var tikt ņemta kāda no minētajām klasifikācijām, tajā pašā laikā smalkākā līmenī pielietojot citu klasifikāciju elementus. Kā, piemēram, interneta veikala sistēmas kopējā arhitektūra varētu būt klienta-servera, kur prezentācijas līmenis ir tīmekļa vietne, darījumprocesu loģikas līmenis ir lietotnes serveris un glabātuves līmenis ir datubāzes serveris. Savukārt darījumprocesu loģikas līmenī tiek izmantoti paralēlās saziņas procesu elementi, lai varētu efektīvāk apkalpot vairākus vietnes lietotājus vienlaicīgi.

Katram lietojumprogrammas arhitektūras veidam ir zināmas priekšrocības un trūkumi, tie ir piemēroti dažādām situācijām. Datu plūsmas arhitektūras ir salīdzinoši vienkāršāk projektēt, ja datus iedomājas kā vienību, kas pārvietojas pa datu plūsmas diagrammu. Šādā veidā iespējam precīzi aprakstīt iespējamos ceļus, nosacījumus, kuriem jāizpildās, lai varētu pa attiecīgo ceļu pārvietoties, kā arī novērtēt, kurās vietās varētu rasties potenciālās problēmas ar robežgadījumiem, vai nekorektiem ievades datiem. Tomēr, datu plūsmas metodei ir trūkums, ka to nav vienkārši pārnest no projektējuma uz realizāciju, pat, ja tiek izmantota objektorientēta programmēšanas metode. [7, 443. lpp.]. Klienta-servera modelim ir salīdzinoši pretējs ieguvums un trūkumi. Šāda veida sistēmu iespējams izvietot uz vairākām neatkarīgām ierīcēm, apkalpot lielu lietotāju skaitu, kā arī vienkāršāk izveidot, vadoties no projektējuma. Trūkumi savukārt ir ievērojami sarežģītāks plānošanas cikls, ņemot vērā vairākas neatkarīgas sistēmas daļas, kurām savā starpā jāsazinās, jāgarantē drošība un ātrdarbība. Virtuālo mašīnu arhitektūras piedāvā iespēju lietotājiem darboties ar vienkāršām komandu kopām, lai veiktu iepriekš definētas darbības. Šāda tipa sistēmas ir iespējams izmantot sarežģītāku uzdevumu veikšanai un tās piedāvā lietotājiem pašiem definēt funkcijas un darbību kopas. Būtiskākais trūkums ir sarežģītā implementācija, jo lietojumprogrammai būtībā jāapstrādā vienkāršota programmēšanas valoda un jāveic atbilstošas darbība, kā arī, lietotājiem nepieciešamas papildus zināšanas izveidotajā valodā [7, 448. lpp.]. Glabātuvju arhitektūras visbiežāk tiek izmantotas kā viena no lielākas informācijas sistēmas daļām. Tās tiek veidotas vieglai un ātrai datu piekļuvei un to manipulācijai, bet bez datu apstrādes.

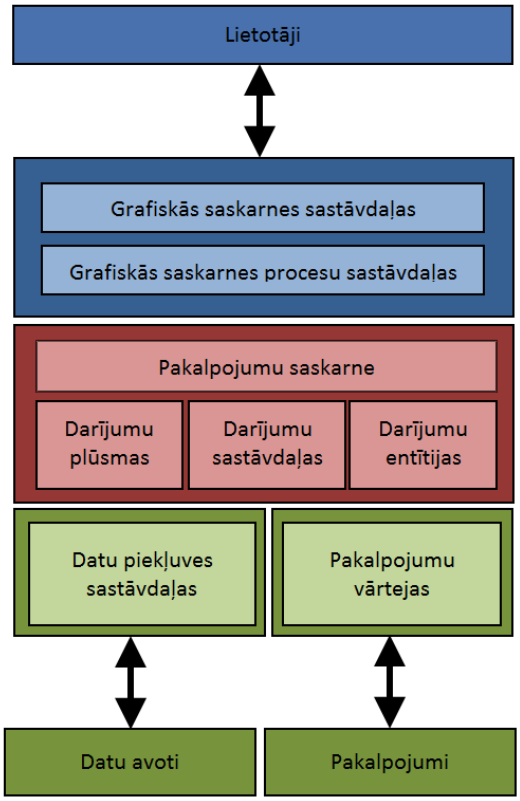
IS projektēšanas laikā izvēlētā arhitektūra būtiski ietekmē tālāko izstrādes procesu un gala sistēmas darbību, tādēļ ir svarīgi novērtēt, kura visprecīzāk atbilst darījumprocesiem. Vēlākos lietojumprogrammas dzīves cikla posmos ir iespējams veikt korekcijas dažādām problēmām, bet nekorekti izvēlētu arhitektūru mainīt nav iespējams.

1.2. Daudzlīmeņu informācijas sistēmas

Daudzlīmeņu informācijas sistēmas arhitektūra ir viens no klienta-servera arhitektūru paveidiem. Šīs arhitektūras pamatā ir nostādne, ka IS tiek sadalīta vairākās fiziskās daļās, un katra no daļām veic specifiskas funkcijas, sazinās ar citām, iepriekš noteiktām, daļām. Neatkarīgās daļas savā starpā sazinās izmantojot definētas saskarnes. Saziņa starp līmeņiem var tikt veikta ar dažādām metodēm, visbiežāk izmantots tiek lokālais tīkls lietojumprogrammām un internets tīmekļa risinājumiem.

Jāuzsver atšķirība starp vairāku līmeņu un vairāku slāņu arhitektūrām. Lietojumprogrammām un IS vairāku līmeņu dalījums ir fiziskā līmenī. Kā, piemēram,

lietojumprogramma, kura sazinās ar pakalpojumu serveri, kurš savukārt sazinās ar datubāzes serveri. Katra no minētajām daļām atrodas atsevišķā fiziskā vietā (lietotāja datora, pakalpojumu servera, datubāzes servera) un starp tām notiek informācijas apmaiņa. Vairāku slāņu arhitektūra savukārt ir loģisks sadalījums konkrētai lietojumprogrammai, vai pat konkrētam līmenim. Tas tiek izmantots, lai efektīvāk organizētu projektējumu, nošķirtu savstarpēji atšķirīgas daļas. Slāņu arhitektūrā atsevišķi programmas moduļi var veikt izsaukumus tikai no zemāk esošiem slāņiem [1, 113. lpp.]. Kā piemērs varētu būt lietojumprogrammas viens, grafiskās saskarnes, līmenis, kurā ir atsevišķi slāņi *Windows* formu attēlošanai, ievades datu apstiprināšanai un saskarnes slānis ar zemāko, darījumprocesu loģikas, līmeni. Cits vairāku līmeņu sistēmas piemērs, kurā līmeņi sadalīti smalkāk slāņos, apskatāms 1.1. attēlā.



1.1. att. **Vairāku līmeņu un slāņu projektējuma piemērs**

1.2.1. Viena līmeņa

Viena no visvienkāršākajām arhitektūrām, kura oriģināli izmantota lieldatoros, kur darbība notiek ar lieldatoru, caur termināļa saskarni. Viena līmeņa arhitektūras galvenā priekšrocība ir tās vienkāršība. Visas nepieciešamās daļas, datu attēlošana, datu apstrāde, datu ielāde un saglabāšana, atrodas vienā vietā gan fiziski, gan loģiski [13]. Mūsdienās visbiežāk šī arhitektūra tiek izmantota nelielās lietojumprogrammās, kas paredzētas vienam lietotājam un darbojas uz viena datora. Šīs īpašības ir arī viena līmeņa arhitektūras lielākie trūkumi, tai ir

ierobežots mērogs, to ir sarežģīti papildināt. Lietojumprogrammas darbība viena līmeņa arhitektūrā iekšēji var tikt sadalīta vairākos loģiskos slāņos, līdzīgi kā vairāku līmeņu arhitektūrās tas tiek darīts fiziski. Šāds risinājums palīdz nodalīt programmas loģiskās daļas vienu no otras, uzlabo to pārskatāmību un uzturēšanu. Trūkums ir sistēmas nedalāmība, kā rezultātā, ir sarežģīti tajā veikt izmaiņas, jo viena izmaiņa bieži izraisa izmaiņas citās sistēmas daļās [14].

1.2.2. Divu līmeņu

Divu līmeņu IS arhitektūra ir viens no klienta-servera modeļa apakš veidiem. Vēsturiski divu līmeņu arhitektūra attīstījās līdz ar datorikas attīstību. Palielinoties datoru skaitam, to pieejamībai un, jo īpaši, interneta izplatībai, aktuāla kļuva nepieciešamība spēt darboties ar daudziem lietotājiem vienlaicīgi. Tipiskas divu līmeņu informācijas sistēmas tiek lietotas mazos un vidējos uzņēmumos, līdz aptuveni 100 lietotājiem [15]. Būtiska klienti-serveru sazināšanās īpašība ir, ka katrs savienojums ir asimetrisks, klients var veikt tikai pieprasījumus, savukārt serveris var šos pieprasījumus apstrādāt un uz tiem atbildēt. Saziņa izmantojot bieži ir sinhrona, klients gaida atbildi no servera pirms tālāka darba turpināšanas [1, 112. lpp]. Viens IS līmenis var funkcionēt reizē kā klients un serveris, piemēram, darījumprasību loģikas līmenis attiecība pret prezentācijas līmeni darbojas kā serveris, savukārt attiecībā pret datubāzes līmeni kā klients.

Šim modelim ir ievērojamas priekšrocības salīdzinot ar viena līmeņa modeli. Ņemot vērā, ka datu glabāšana tiek veikta vienā kopējā vietā, ir iespējams veidot centralizētas IS, kurās visi lietotāji strādā ar kopīgu datu kopumu. Salīdzinot ar lielākām un sarežģītākām sistēmām, divu līmeņu sistēmas ir lētāk izstrādāt un vienkāršāk realizēt, tajā pašā laikā nezaudējot pārāk daudz no ieguvumiem, kas saistīti ar dalītu sistēmu. Neskatoties uz minētajām priekšrocībām, divu līmeņu modelim ir būtiski trūkumi.

Izmantotā darījumprocesu loģika atrodas lietojumprogrammās uz individuālām darbstacijām. Tas nozīmē, ka lietojumprogrammas izmaiņu gadījumā, uzstādīšana jāveic uz visiem datoriem atsevišķi, kas savukārt rada papildus administrēšanas slodzi un potenciālu darbu apstāšanos versiju maiņas laikā [11].

Katram lietotājam ir individuāls savienojums ar datubāzi, kas nozīmē, ka ir liels slogs savienojumu veidošanā un tīkla noslodzē, kā arī, katrai darbstacijai ir nepieciešami atbilstošās datubāzes draiverprogrammas, lai savienojumu varētu izveidot. [16]

Cita problēma ir saistīta ar IS drošību. Ņemot vērā, ka darījumprocesu loģika un lietotāju tiesību ierobežojumi tiek piemēroti uz darbstacijām, ir potenciāls, ka uzbrukuma gadījumā notiek nesankcionēta piekļuve un manipulācija ar datiem. Uzbrukumi var tikt veikti

gan izmantojot neatļautas lietojumprogrammas izmaiņas, gan datubāzes datiem piekļūstot tiešā veidā, apejot lietojumprogrammā iestrādātos ierobežojumus.

Būtisks trūkums divu līmeņu arhitektūrai priekš mūsdienu lietojumprogrammām, no kurām liela daļa piedāvā tīmekļa bāzētu funkcionalitāti, ir nespēja nodrošināt pastāvīgu savienojumu starp līmeņiem [4, 73. lpp].

Divu līmeņu arhitektūras pielietojums mūsdienās samazinās, neskatoties uz tās priekšrocībām. Ievērojamo drošības un tīmekļa saderības trūkumu dēļ, projektēšanai izmantotas tiek citas, kā, piemēram, trīs un vairāk līmeņu arhitektūras.

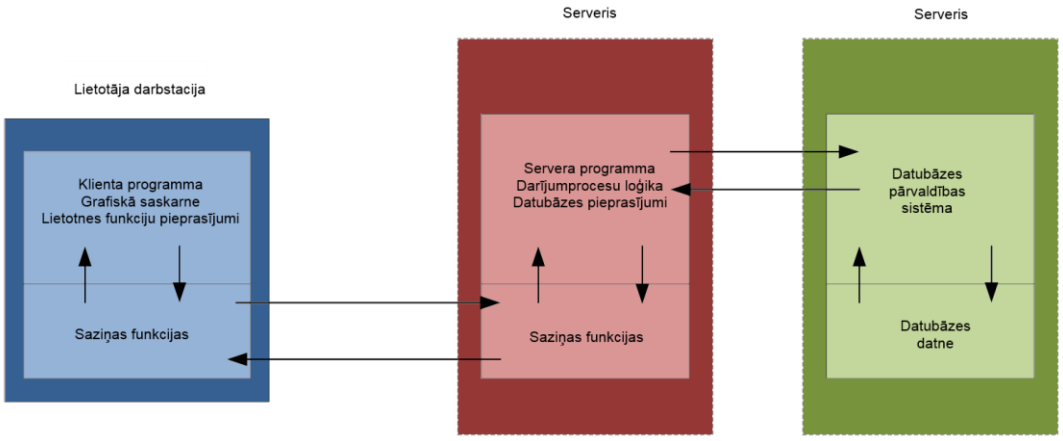
1.2.3. Daudzlīmeņu

Daudzlīmeņu IS arhitektūra ir nākamais loģiskais solis pēc divu līmeņu arhitektūras. Informācijas sistēma tiek sadalīta vēl vairākās atsevišķās daļās. Populārākais dalījums ir trijās: prezentācijas, darījumprocesu loģikas un datu glabāšanas. Individuālie līmeņi var sastāvēt no vairākiem slāņiem, lai uzlabotu funkciju organizāciju. Daudzlīmeņu arhitektūras galvenās priekšrocības ir tās modularitāte un mērogojamība.

Prezentācijas līmenis. Šajā līmenī tiek veidota saziņa starp informācijas sistēmas lietotāju un pārējiem līmeņiem. Klasiskajā daudzlīmeņu arhitektūrā, šajā līmenī ir tikai vizuālie elementi un datu ievade, izvade, tomēr reālās sistēmās šajā līmenī ir darījumprocesu loģikas elementi, kā, piemēram, kādas sistēmas daļas un ievades lauki ir pieejami lietotājam, kādas ir iespējamās vērtības, piekļuves tiesību kontrole.

Darījumprocesu loģikas līmenis. Šis ir „resnākais" no līmeņiem, kurā atrodas visas darījumprocesu loģikai nepieciešamās funkcijas, kā arī drošības risinājumi un saskarnes saziņai ar citiem līmeņiem. Šajā līmenī notiek visu procesu organizācija, kas saistīta ar lietotāju darbību. Ar datubāzes līmeni atļauts sazināties tikai no darījumprocesu loģikas līmeņa, kas būtiski uzlabo drošību sistēmai, jo lietotājam nav iespējama tieša piekļuve un manipulācija ar datiem.

Datu glabāšanas līmenis. Atbild par datu glabāšanu un piekļuvi tiem. Līdzīgi kā prezentācijas līmenim, klasiskajā modelī datu glabāšanas līmenī nav darījumprocesu loģikas elementu, tomēr, arī šajā līmenī bieži vien ir nepieciešama daļa no darījumprocesu loģikas, kas saistīta ar datu savstarpējām atkarībām. [17]



1.2. att. **3-līmeņu arhitektūra [18]**

Veidojot daudzlīmeņu informācijas sistēmas, īpaša uzmanība jāpievērš saskarnēm starp līmeņiem. Visa datu plūsma informācijas sistēmā pārvietojas izmantojot šīs saskarnes, tādēļ ir būtiski, lai sistēma tiktu projektēta ar nodomu, ka datu apmaiņa starp līmeņiem ir reta, bet apjomīga. Šis princips ir pretējs tam, kādu var ievērot starp slāņiem, kur patērēto resursu daudzums uz vienu savienojumu ir salīdzinoši ļoti neliels. Kā piemēru var minēt informatīvu portālu, kurā klients vēlas uzzināt, kādi sabiedriskie transporti pieejami viņam vēlamajā maršrutā nākamajā dienā. Viņa pieprasījuma rezultātā uz darījumprasību loģikas serveri, pa internetu, tiek nosūtīts viens pieprasījums „lūdzu atsūtīt visus sabiedriskos transportus, šajā datumā, šajā maršrutā". Pakalpojumu serveris šo pieprasījumu saņemot, veic vienu, līdzīgu pieprasījumu datubāzes serverim. Kad informācija saņemta, pakalpojumu serveris var veikt daudzas, individuālas darbības ar datiem, tos sagrupējot, izceļot izdevīgākos piedāvājumus, kur katra darbība ir viens pieprasījums, bet tādēļ, ka tie ir viena līmeņa ietvaros, tie ir ātri. Pēc informācijas sagatavošanas, tā visa vienā lielā atbildē tiek atgriezta prezentācijas slānim, kurš tad to attēlo lietotājam.

Daudzlīmeņu arhitektūras izmantošanai ir daudzi būtiski ieguvumi.

- Iespēja sadalīt informācijas sistēmu vairākās neatkarīgās daļās, kuras izvietotas uz dažādiem datoriem un serveriem. Šādā veidā tiek sadalīta slodze uz lielāku aparatūras daudzumu, uzlabojot sistēmas kopējo veiktspēju.
- Piekļuve datubāzes līmenim ir nepieciešama tikai no darījumprocesu loģikas līmeņa, kā rezultātā, datubāzes piekļuves draiveri nepieciešami tikai vienā vietā, nevis uz visu lietotāju datoriem. Gadījumos, kad mainās datubāzes struktūra, ir iespējams, ka jāmaina tikai darījumprocesu loģikas līmeņa programmatūra un prezentācijas līmenī izmaiņas nemaz nav nepieciešamas.
- Iespējams uzlabot drošību, nodalot sensitīvās daļas atsevišķi, aiz ugunsmūra. Kombinācijā ar datu apstrādes izvietošanu darījumprocesu loģikas līmenī,

nesankcionēta piekļuve uzbrucējam atļautu veikti tikai tās pašas darbības, kas pieejamas lietotājam, citi pieprasījumi tiktu noraidīti. [19]

- Vieglāk veikt izmaiņas atsevišķajām daļām, ņemot vērā, ka saskarnes starp līmeņiem ir definētas. Izmainot vienu no līmeņiem, ja nemainās saskarnes, ir iespējams, ka nav nepieciešama pārējo līmeņu maiņa.
- Iespējams paplašināt sistēmas mērogu būtiski nemainot tās arhitektūru. Palielinoties lietotāju skaitam, aprēķinu prasībām vai citu iemeslu dēļ, iespējams uzlabot IS veiktspēju gan uzlabojot aparatūru (*scale up*), gan pievienojot papildus aparatūru jau esošajai (*scale out*) [20].
- Kritisku kļūdu gadījumos, kā, piemēram, zudusi elektrība kādam no serveriem, pārējie līmeņi var turpināt darbību un, sniegt lietotājiem informāciju par radušos situāciju.

Neskatoties uz daudzajiem ieguvumiem, daudzlīmeņu informācijas sistēmām ir arī vērā ņemami trūkumi.

- Ievērojami sarežģītāk projektēt, ieviest un administrēt daudzlīmeņu sistēmu, kurā tiek izmantotas dažādas tehnoloģijas, vairāki serveri, savienojumi starp tiem un lietotāju datoriem [5, 481. lpp.]. Projektējumā jāņem vērā pieejamā aparatūra, infrastruktūra, kādi resursi izmantojami sistēmas izstrādē. Tā jāprojektē, lai būtu ērta gan lietotājiem ikdienas darbā, gan administratoriem to uzturot.
- Sarežģīti nodrošināt pilnīgu drošību starp līmeņiem, ņemot vērā, ka tiem ir jāspēj brīvi sazināties savā starpā. Ugunsmūris starp prezentācijas un darījumprocesu loģikas līmeņiem sniedz daļu no aizsardzības, tomēr tas saglabā ievainojamību, ja uzbrucējs spēj iekļūt iekšējā tīklā un piekļūt serveru līmeņiem tiešā veidā.
- Iespējamas ātrdarbības problēmas, kas saistītas ar nekorekti veidotām vai izmantotām saskarnēm, gadījumos, ja saziņa starp līmeņiem notiek biežāk nekā nepieciešams. Saziņu starp līmeņiem ieteicams veikt tikai, ja tas nepieciešams un to darīt iespējami reti.
- Starplīmeņu saziņa tiek veikta izmantojot lokālo tīklu vai internetu, kas var radīt saziņas pārtrūkumus un lēnu informācijas sūtīšanu. Atkarībā no IS specifikas, iespējams nepieciešami vairāki neatkarīgi interneta pieslēgumi, ja sistēmai jābūt pieejamai nepārtraukti, kā, piemēram, banku sistēmām.
- Sarežģītāk apstrādāt kļūdas, kas rodas sistēmas darbības rezultātā. Katra kļūda, kas rodas, jāpadod starp līmeņiem, lai to varētu reģistrēt kļūdu žurnālā un attēlot lietotājam korektu kļūdas paziņojumu. Izstrādātājiem arī grūtāk atrast kļūdas cēloņus, ņemot vērā,

ka ar izstrādes un atkļūdošanas rīkiem ne vienmēr ir iespējama programmas darbības vērošana starp dažādiem līmeņiem.

Trīs līmeņu sistēmas ir vienkāršākais no daudzlīmeņu arhitektūras apakšveidiem. Daudzlīmeņu sistēmām dalījums parasti ir līdzīgs trīs līmeņu variantam – tām ir prezentācijas un datubāžu līmeņi, un smalkāk sadalīts vidējais, darījumprocesu, līmenis. Darījumprocesu līmeņos bieži tiek koplietotas darījumprocesu entītijas, tādējādi ļaujot centralizēt izmantojamo loģiku, nodrošinot, ka tās modifikāciju gadījumā izmaiņas veicamas vienā vietā [21].

Zinot, kādas arhitektūras ir pieejamas un kādas ir to priekšrocības un trūkumi, sākot informācijas sistēmas izstrādi, var izdarīt informētu izvēli. Vispārīga vadlīnija arhitektūras izvēlei redzama 1.1. tabulā., tajā gan norādīta salīdzinoši šaura iespējamo arhitektūru kopa – daudzlīmeņu arhitektūras.

1.1. tabula

Vadlīnijas lietojumprogrammas arhitektūras izvēlei [4, 74. lpp.]

Lietojumprogrammas vide	Ieteicamais modelis
Darbstacijas lietojumprogrammas	Viena līmeņa
Lietojumprogrammas iekšējā tīklā ar daudziem lietotājiem un vienu datubāzi	Divu līmeņu
Lietojumprogrammas iekšējā tīklā ar daudziem lietotājiem un vairākām datubāzēm	n-līmeņu

Zināšanas par arhitektūrām un to darbību nepieciešamas arī sistēmu uzturēšanas procesā, apzinot potenciālās problemātiskās vietas, iespējams veikt korekcijas pirms problēmas ir radušās.

2. ĀTRDARBĪBAS UZLABOŠANA

Ikdienā izmantotās informācijas sistēmas kļūst arvien sarežģītākas, tās apkalpo vairāk cilvēku un tiek izmantotas uz daudzveidīgākām ierīcēm. Lietotāji sagaida, ka IS strādās korekti un ātri, lieki netērējot viņu laiku. Lai spētu apmierināt šīs prasības, IS izstrādes un uzturēšanas laikā, arvien lielāks uzsvars un uzmanība jāvērš darbībām, kas saistītas ar IS ātrdarbības novērtēšanu, traucējumu cēloņu noteikšanu un novēršanu. Šī procesa atvieglošanai ir izstrādātas vairākas ātrdarbības testēšanas metodes, no kurām trīs tiks apskatītas.

2.1. Problēmu tipiskie cēloņi, to risināšanas metodes

Pirms var sākt pilnvērtīgi testēt informācijas sistēmas, lai noteiktu vai un kādas problēmas tām ir ar ātrdarbību, ir jāapzina, kādi ir izplatītākie ātrdarbības problēmu cēloņi. No visiem iespējamiem ātrdarbības problēmu cēloņiem tiks apskatīti tie, kas saistīti ar daudzlīmeņu informācijas sistēmām.

Primārais cēlonis lielākajai daļai no ātrdarbības problēmām ir nepilnīgs vai slikts informācijas sistēmas projektējums. Pilnvērtīgā projektējumā jau no prasību specifikācijas brīža būtu jānoskaidro, kāds ir sistēmas plānotais apjoms, lietotāju skaits, uz kādas aparatūras tā tiks uzstādīta, kādas ir pieejamās tehnoloģijas un licences. Visu šo faktoru kombinācija noteiks, kā izstrādātā informācijas sistēma funkcionēs un vai ar laiku neradīsies ātrdarbības problēmas.

Ciešs slāņu un līmeņu savienojums. Pats par sevi, ciešs slāņu un līmeņu savienojums nav būtisks ātrdarbības problēmu cēlonis. Potenciāli, cieši savienojumi var pat radīt veiktspējas uzlabošanu attiecīgajās vietās. Tomēr šāds risinājums rada problēmas, ja sistēmas ātrdarbības kopējai uzlabošanai ir vēlme pievienot papildus aparatūru (*scale out).* Šādā gadījumā jāpārveido visas saistītās saskarnes.

Saziņa starp līmeņiem. Saziņai starp līmeņiem tiek izmantotas precīzi definētas saskarnes. Definīcijas sevī ietver funkciju nosaukumus, kādi ieejas dati tiek sagaidīti un, kāda datu kopa tiks atgriezta atbilstošajiem ieejas datiem. Problēmas rodas, ja saskarnes ir nekorekti veidotas vai tiek nepareizi izmantotas. Starp līmeņiem izmantotajām saskarnēm jābūt ar iespējami vispārīgākiem ieejas datu nosacījumiem, lai nebūtu nepieciešamība veikt datu pieprasījumus atkārtoti un visa vajadzīgā informācija tiktu nodota vienā reizē. Ņemot vērā, ka saziņa starp līmeņiem ir salīdzinoši lēna, nepamatoti liels izsaukumu skaits var radīt jūtamu ātrdarbības kritumu.

Laiksakritība. Ņemot vērā, ka informācijas sistēmas lielākoties izmanto daudz lietotāju vienlaikus, ir būtiski, lai viens lietotājs ar savu darbību nevarētu traucēt vai pat bloķēt citu lietotāju darbu. Izmantojot sinhronus savienojumus, process nevar turpināt darbu, kamēr nav saņemta atbilde – iespējama citu lietotāju bloķēšana, ja tiem arī ir nepieciešama informācija no attiecīgā procesa. Lai novērstu šādas situācijas, datu pārsūtīšanai starp līmeņiem ieteicams izmantot asinhronus izsaukumus, kuru sākums un beigas nav atkarīgi viens no otra. Iespējams arī apjomīgus izsaukumus sadalīt vairākos, mazākos, tomēr ar šo pieeju jābūt piesardzīgam, jo sadalot pārāk smalki var rasties veiktspējas problēmas dēļ daudzajiem īsajiem pieprasījumiem.

Resursu pārvaldība. Palielinoties apstrādāto datu apjomam un sistēmu sarežģītībai ir svarīgi darba gaitā racionāli izmantot pieejamos resursus. Veidotajām datu struktūrām nevajadzētu tērēt liekus resursus un pēc iespējas ieteicams izmantot jau esošas datu struktūras, jaunu radīšanas vietā. Modernās izstrādes platformās, piemēram, *Java* un *Microsoft .NET Framework*, ir pieejamas automātiskas nevajadzīgo resursu atbrīvošanas sistēmas, tomēr tās ne visās situācijās šo procesu veic optimāli.

Kešdarbe. Datu kešdarbe ir viens no veidiem, kā risināt problēmas, kas saistītas ar liela apjomu pārsūtīšanu starp līmeņiem. Šādi iespējams saglabāt datus ātrai piekļuvei, neveicot sarežģītus vai ilgus datu pieprasījumus uz datubāzi. Problēmas rodas situācijās, ja kešdarbe tiek veikta ar datiem, kas bieži mainās, izmantojot procesora resursus kopiju atkārtotai veidošanai, vai arī, kešdarbes kopijas tiek saglabātas pārāk ilgi, lieki izmantojot atmiņas resursus.

Datu struktūras un algoritmi. Būtiska nozīme projektējumā ir pareiza datu struktūru un algoritmu izvēle. Datu struktūra un apstrādes algoritms, kas bez problēmām darbojas ar dažiem desmitiem ierakstu, var nebūt pietiekams vairākiem tūkstošiem ierakstu [22]. Tomēr jāņem vērā, ka algoritmi, kas efektīvi spēj strādāt ar ļoti lieliem datu apjomiem bieži ir ievērojami sarežģītāki. Rezultātā, tos ir sarežģīti ieviest un ir lielāka iespēja ieviešot pielaist kļūdas. Šī iemesla dēļ, bieži ir izdevīgāk sākotnēji izmantot vienkāršus risinājumus, kas nespēj strādāt ar liela apjoma datiem, kamēr datu apjoms ir neliels. Tad, laikam ejot uz priekšu, ja rodas nepieciešamība, nomainīt vienkāršos algoritmus un datu struktūras pret efektīvākiem risinājumiem. Šāds risinājums arī nodrošina to, ka, ja izveidotā funkcionalitāte nav sniegusi cerēto, nav ieguldīts pārlieku ilgs laiks tās izstrādē.

Liela daļa no minētajām, iespējamām problēmām ir saistītas ar informācijas sistēmas projektējumu, un tās ir iespējams noteikt un labot vēl pirms sistēmas nodošanas ekspluatācijā. Tomēr ne vienmēr šāds risinājums ir iespējams, ir problēmas, kurām nevar sagatavoties un paredzēt, tās bieži ir jārisina pēc problēmu atklāšanas.

Informācijas sistēmas reizēm tiek izstrādātas un izmantotas ilgus laika periodus, daudzus gadus, pat gadu desmitus. Laikam ejot, ja tiek turpināta IS izstrāde un funkciju papildināšana, oriģināli izmantotais aprīkojums var neatbilst jaunajām prasībām.

Līdzīga problēma aparatūras novecošanai ir izmantoto tehnoloģiju novecošana. Izstrādājot informācijas sistēmas tiek izmantotas tajā brīdī esošās tehnoloģijas, kas vislabāk, visērtāk un vislētāk spēj palīdzēt izstrādes procesā. Citu tehnoloģiju un ietvaru (*framework*) izmantošana var ievērojami paātrināt un atvieglot izstrādes procesu, bet jāņem vērā, ka šādi izstrādātām sistēmām ir noteikts mūžs. Cenšoties uzlabot sistēmas ātrdarbību, uzstādot jaunu serveri, var negūt maksimālo efektu no jaunās aparatūras, ja izstrādātā sistēma spēj izmantot tikai vienu no procesora kodoliem un tā ir rakstīta 32-bitu arhitektūrā, kā rezultātā var adresēt tikai 4 GB no pieejamās operatīvās atmiņas.

Problēma ar ko saskaras ilgtermiņā izmantotas sistēmas ir neizbēgams datu daudzuma pieaugums. Sistēmas darbības procesā tiek apstrādāta un saglabāta informācija, veidoti jauni datubāzes ieraksti, jauni lietotāji un klienti. Paplašinoties funkcionalitātei, paātrinās arī datu apjoma pieaugumu ātrums. Pieaugot datu apjomam, algoritmi, kas izmantoti datu atlasei, organizēšanai, grupēšanai, filtrēšanai var kļūt pārāk lēni.

Problēma, kuru ir ļoti grūti paredzēt ir pēkšņa sistēmas popularitāte. Sistēmām ar publisku piekļuvi ir zināms aptuvens lietotāju skaits, kas sistēmu izmanto dažādos dienas laikos, cik lietotāju ir dienā kopā, kā slodze sadalās pa nedēļām, mēnešiem, ceturkšņiem un gadiem. Izmantojot šo informāciju var plānot, cik lieli resursi būs nepieciešami dažādos laika posmos, kādas ir plānotās maksimālās slodzes. Šādam plānam jāietver arī zināma resursu rezerve, neparedzētām situācijām. Kā piemēru neparedzētai popularitātei var minēt gadījumu Lielbritānijā, kad dažus gadus atpakaļ valdība nolēma publiskot 1901. gada tautas skaitīšanas datus. Pirmajā dienā pēc interneta vietnes darbības sākšanas, datus varēja apskatīties, bet darbība bija nedaudz lēnāka, kā varētu vēlēties. Jau nākamajā dienā, vietnes apmeklētājus sagaidīja īsa atvainošanās, ka pakalpojums īslaicīgi nebūs pieejams. Pagāja vairākas nedēļas, līdz vietne atsāka darbību [10, 8. lpp.].

Ātrdarbības problēmu novēršanai ir iespējami dažādi risinājumi, atbilstoši problēmas cēloņiem. Līdzīgi kā daudzas citas ar IS projektējumiem saistītas problēmas, veiktspējas problēmu risināšanai patērējamo resursu daudzums palielinās, jo vēlāk problēmas tiek atklātas.

Efektīvākie risinājuma veidi ir izsargāšanās no problēmām, tām neļaujot veidoties IS dzīves laikā. Projektējot IS, paredzot gala sistēmas apjomu, lietotāju skaitu un datu apjomu iespējams prognozēt vajadzīgo resursu daudzumu, izvēlēties atbilstošu aparatūru un

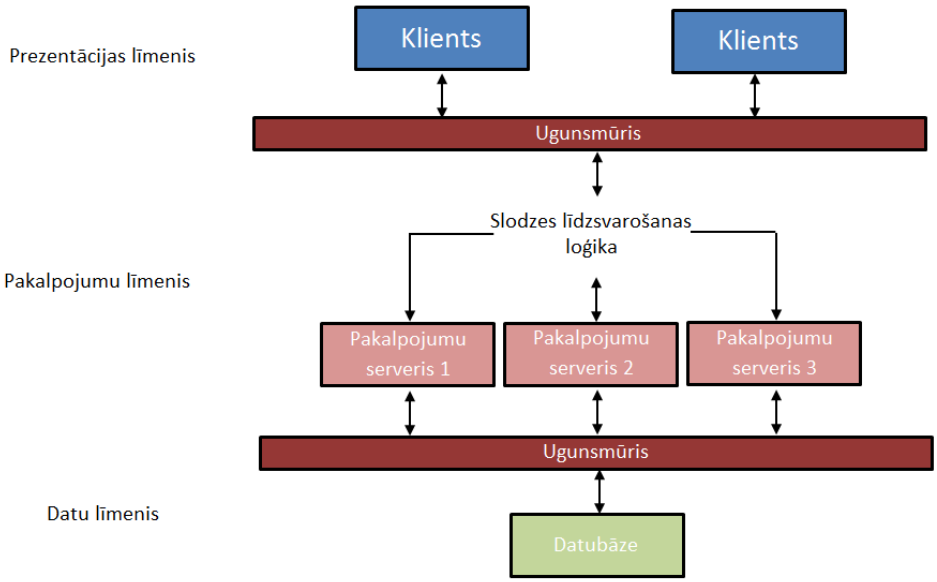
programmatūru. Laicīgi paredzot, ka kāda no prasībām varētu pieaugt, iespējams palielināt pieejamo resursu daudzumu iepriekš.

Jau ekspluatācijā esošai sistēmai, iespējas uzlabot veiktspēju ir ierobežotas. Visbiežāk uzlabojumi saistīti ar programmas nepietiekamo daļu pārrakstīšanu. Veicamās izmaiņas var būt salīdzinoši nelielas, piemēram, kārtošanas algoritma maiņa no ievietošanas kārtošanas uz apvienošanas kārtošanu, bet tā var būt arī apjomīga un sarežģīta, ja, piemēram, jāmaina starplīmeņu saziņai izmantotā tehnoloģija.

Kešdarbe ir viens no veidiem, kā ievērojami uzlabot sistēmas ātrdarbību, kurā notiek darbs ar lieliem un lielākoties nemainīgiem datu kopumiem. Veicot datu kešdarbi prezentācijas un darījumprocesu loģikas līmeņos, iespējams samazināt izsaukumu skaitu uz datubāzes līmeni un samazināt pārsūtītās informācijas apjomus. Iespējama arī datu kešdarbe datubāzes līmenī, glabājot lielus datu apjomus, kurus bieži pieprasa, operatīvajā atmiņā. Šāds risinājums ievērojami paātrina datu atlasi, bet var radīt citas ātrdarbības problēmas, ja kopējais atmiņas apjoms nav pietiekams visām darbībām [22].

Vienkāršākais veids, kā uzlabot IS veiktspēju, ir uzlabot serverus un datorus, uz kuriem tā ir izvietota (*scale up*). Šādu uzlabojumu rezultātā nav jāveic nekādas papildus izmaiņas IS darbībā un, ja IS arhitektūra tam ir piemērota, uzlabojumu veikšana var notikt lietotājiem to nepamanot. Veicot aparatūras maiņu, jānovērtē, vai gūtie veiktspējas uzlabojumi atbilst ieguldītajiem līdzekļiem un cik ilgam laikam ar iegūto resursu rezervi pietiks [20].

Alternatīvs aparatūras uzlabošanas process ir paralēli strādājošu serveru pievienošana darījumprocesu loģikas un datubāzes līmeņos (*scale out*). Paralēli strādājošie serveri sadala ienākošo slodzi, to savā starpā izlīdzinot. Piemēru vienkāršai trīs līmeņu informācijas sistēmai ar trīs darījumprocesu loģikas serveriem var redzēt 2.1 attēlā. Paralēlu serveru risinājumam ir arī citi ieguvumi, neskaitot ātrdarbības uzlabošanu. Kritisku kļūdu gadījumā, ja viens no paralēlajiem serveriem pārtrauc darbību, tā pāris turpina darbu, neietekmējot kopējo sistēmas darbību. Kā rezultātā problēmu risināšanai pieejams vairāk laika, kas nebūtu iespējams pilnīgas sistēmas darbības pārtraukuma gadījumā [20].



2.1. attēls **Sadalītas slodzes piemērs [20]**

Viens no sarežģītākajiem risinājumiem ir esošo tehnoloģiju maiņa. Atkarībā no izvēlētajām tehnoloģijām un cik cieši tās ir integrētas IS darbībā, ir iespējams, ka šāds risinājums nemaz nav iespējams, un izdevīgāk ir projektēt un realizēt jaunu sistēmu. Biežāk šādu risinājuma pamatā ir citas prasības, kā, piemēram, jauna darījumprocesu loģika vai arī vecā IS vairs netiek izmantota pilnībā, un ātrdarbības ieguvumi, kas rodas jaunās sistēmas izstrādes rezultātā, ir tikai pozitīvs blakusprodukts.

Ātrdarbības problēmām ir ļoti dažādi cēloņi un to risināšanai pieejams līdzvērtīgs daudzums risinājumu. Ne visas problēmas iespējams atrisināt ar visiem risinājumiem un dažādu risinājumu realizācijai ir jāiegulda dažāds resursu daudzums.

2.2. Ātrdarbības novērtēšanas testi

Informācijas sistēmām iespējams testēt un novērtēt dažādus veiktspējas parametrus, kas saistīti ar visas sistēmas darbību, daudzu lietotāju vienlaikus darbu, individuālu lietotāju pieredzi. Katram no mērāmajiem parametriem ir savas metodes, kā veiktspējas datus iegūt, un veidi, kā iegūtos datus attēlot, analizēt un izmantot tālāku darbību veikšanai. Veiktspējas testēšana tiek veikta visā IS dzīves ciklā, tomēr izstrādes procesa laikā iespējams testēt tikai individuālu funkciju veiktspēju un kopēju priekšstatu par sistēmas darbību iespējams iegūt tikai, kad visas daļas ir realizētas [6, 472. lpp.].

Sākotnējie testi tiek izmantoti, lai noskaidrotu sistēmas darbību tipiskās situācijās. Pārbaudītas tiek visas sistēmas funkcijas individuāli, katra funkcija tiek testēta vairākas reizes, katrā ar lielāku vienlaicīgu lietotāju skaitu. Maksimālais lietotāju skaits katrai funkcijai var

atšķirties un ir atkarīgs no IS specifikācijas, tomēr tas ir salīdzinoši neliels un netuvojas sistēmas plānotajai maksimālajai veiktspējai. Pārbaudot visas funkcijas individuāli, ir vienkāršāk izolēt, kurās no daļām varētu būt problēmas, kas nav tik vienkārši izdarāms, ja tiek veikta kopēja darījumprocesu testēšana, kuriem ir daudzas neatkarīgas daļas. Visas problēmas, kas noskaidrotas veicot sākotnējos testus, var tikt atrisinātas pirms sarežģītāku testu veikšanas, kā rezultātā nākamo testu rezultāti ir skaidrāki un tajos var meklēt specifiskākas problēmas. [12, 8. lpp.]

Slodzes testi tiek izmantoti, lai noskaidrotu kāda ir sistēmas darbība pie dažādām slodzēm, kas nepārsniedz maksimālo paredzēto. Slodzes testi parasti ietver plašāku sistēmas vienlaicīgu noslogošanu, nekā sākotnējie testi, kā rezultātā tie precīzāk atbilst reālās pasaules situācijām, kādās IS tiks lietota, tomēr tas nozīmē, ka ir sarežģītāk analizēt iegūtos rezultātus. Tajos var būt redzams, ka pie zināma slodzes līmeņa sistēmas darbība nav apmierinoša, bet nav iespējams precīzi noskaidrot, kuras sistēmas daļas ir lēnās darbības cēlonis. [3, lpp. 10]

Stresa testi ir pēkšņa sistēmas slodzes kāpināšana līdz vai pāri maksimālajai plānotajai. Šos testus ieteicams veikt pēc tam, kad citas veiktspējas problēmas, kas noskaidrotas sākotnējos un slodzes testos, ir novērstas. Balstoties uz stresa testa rezultātiem, var izdarīt secinājumus par IS kopējo kapacitāti un sniedz administratoriem informāciju, kā sistēma salūzt pie pārslodzes un kā pēc tam atgūstas. [12, 11. lpp.]

Ilgtermiņa testi (*soak test*) tiek izmantoti, lai novērtētu sistēmas darbību ilgstošos laika periodos. Tie tiek izmantoti, lai noskaidrotu, vai kādā procesā nav atmiņas noplūdes, vai arī nav problēmas ar savienojumu noildzi starp līmeņiem. Veicot ilgtermiņa testus to ieteicams darīt iespējami ilgi, vai līdz brīdim, kad noskaidrotas galvenās tendences. [12, 11. lpp.]

Apjoma testēšana izmantojama, lai noskaidrotu, kā sistēma strādā ar liela apjoma datubāzēm un failiem. Šāda veida testēšana ne vienmēr ir nepieciešama, tomēr sistēmai ir risks, ja visa pārējā testēšana tiek veikta ar neliela apjoma datiem un datubāzēm. Veicot apjoma testēšanu, nav nepieciešams liels lietotāju slogs rezultātu iegūšanai. [12, 11. lpp.]

Sistēmas atbildes laika testi norāda, cik ilgs laiks paiet no brīža, kad lietotājs uzsāk kādu darbību, līdz brīdim, kad ir iespējams redzēt rezultātus vai veikt nākamo darbību [2, 21. lpp.]. Dažādām sistēmas daļām un funkcijām ir dažādi pieļaujamie atbildes laiki, kā, piemēram, atverot jaunu formu lietojumprogrammā atbildes laikam būtu jābūt dažu milisekunžu robežā, bet pieprasot specifiskus datus par pēdējo desmit gadu periodu, atbildes laiks var būt desmit un vairāk sekundes.

Mazāk tehnisks, bet bieži pats svarīgākais pārbaudes veids, kā noteikt vai sistēma darbojas pietiekami ātri, ir tās lietotāju apmierinātība. Ja visas sistēmas funkcijas var izpildīt, un lietotājiem nav būtisku sūdzību, vai sūdzību nav vispār, var secināt, ka sistēmas darbība ir

optimāla un nav nepieciešamas darbības veiktspējas uzlabošanai. Šai metodei gan ir būtisks trūkums, ka ar to var novērtēt tikai esošo situāciju, bet nav iespējams paredzēt, kāda būs ātrdarbība nākotnē, ar vairāk lietotājiem, lielākiem datu apjomiem.

Veiktspējas testus veicot ir ļoti svarīgi būt precīzam, kādi dati tiks vākti testēšanas laikā. Piemēram, ja mērķis ir noskaidrot sistēmas atbildes laiku, tad iegūstamie dati ir 1) pilns atbildes laiks, 2) procesora patērētais resurss, 3) tīkla savienojuma laiks, 4) datubāzes piekļuves laiks, 5) gaidīšanas laiks [8, 209. lpp.]. Tas ir svarīgi, lai varētu veikt iespējami precīzus novērtējumus, kuras sistēmas daļas nestrādā pieņemamā ātrumā. Tikai izmantojot iegūtos datus kopumā, var novērtēt, vai iegūtie rezultāti ir apmierinoši, vai arī ir jāveic darbības to uzlabošanai.

2.3. Testēšanas metodes

2.3.1. Pirmā metode

Pirmā no apskatītajām veiktspējas testēšanas metodēm aprakstīta tehniskajā dokumentā „*Performance Testing Methodology*" [12]. Tajā izklāstītas galvenās procesa daļas, kas nepieciešamas IS veiktspējas pilnvērtīgai testēšanai. Aprakstītā metode izmantojama informācijas sistēmas vispārējai ātrdarbības novērtējuma iegūšanai, ko pēc tam var izmantot individuālu sistēmas daļu analīzei.

Projekta novērtēšana. Šajā procesa daļā tiek veikta informācijas iegūšana par esošo situāciju un kāds rezultāts tiek sagaidīts pēc testēšanas veikšanas. Pirms plānošanas sākšanas būtu jābūt zināmām visām prasībām (lietotāju skaitam, datu apjomam, izmantojamajiem rīkiem, atvēlētajam laikam, vai vispār ir iespējams izmantot automatizētu testus konkrētai sistēmai u.c.), lai varētu pilnvērtīgi plānot tālākos procesus.

Plānošana. Testēšanas plāns ir visaptverošs testēšanas procesa apraksts. Tajā ir iekļautas visu posmu detaļas, nosacījumi, potenciālie riski. Lai varētu sasniegt veiksmīgu testēšanas noslēgumu, visiem testēšanas plāna punktiem būtu jābūt izpildītiem.

Automatizēšana. Viena no svarīgākajām testēšanas daļām, tajā tiek veidoti automātiskie testi, kuru izpildes rezultātā var izdarīt secinājumu par sistēmas darbību. Veidojot testus rūpīgi jāiepazīstas ar testējamo sistēmu, jau to rakstīšanas laikā piefiksējot funkcijas, kuru darbība šķiet lēna. Šādi iespējams konstatēt problemātiskas vietas pirms automatizētās testēšanas sākšanas. Sistēmas izmaiņu gadījumā jāpārbauda visi izveidotie testi, vai tie turpina korekti strādāt.

Testu izpilde. Veiktspējas testēšana ir iespējams veikt dažādos veidos, tomēr pilnvērtīga testēšana iekļauj dažādas metodes, kā tas aprakstīts 2.1 tabulā.

2.1. tabula

Veiktspējas testēšanas veidi

Veiktspējas testēšana	
Testēšanas veids	Apraksts
Sākotnējais tests	Noskaidrot veiktspējas sākotnējos parametrus
Slodzes tests	Simulēt reālas darbības slodzi uz sistēmu
Stresa tests	Pārslogot sistēmu
Ilgtermiņa tests	Ilgstošs sistēmas tests
Apjoma tests	Liela datu apjoma/caurplūdes tests

Detalizētāku testēšanas veidu aprakstu var apskatīt 2.2 nodaļā.

Rezultātu analīze. Rezultātu analīze ir potenciāli sarežģītākā testēšanas daļa. Veiksmīgai analīzei nepieciešams, lai izstrādātie testi sniegtu vēlamo sistēmas skatu, kuru izmantojot var izdarīt secinājumus par sistēmas darbību.

Ziņošana. Iegūtos rezultātus ir jānoformē pārskatāmā veidā un jāziņo par tiem, lai citi būtu informēti par iegūtajiem rezultātiem un varētu veikt turpmākas darbības, ja tādas nepieciešamas.

2.3.2. Otrā metode

Otrā apskatītā metode aprakstīta grāmatā „*Software Testing. Principles and Practices*" [9]. Arī šajā metodē ātrdarbības novērtēšanas process ir sadalīts vairākās individuālās daļās un visu informācijas sistēmas atsevišķo daļu ātrdarbība tiek novērtēta reizē.

Prasību apkopošana. Atšķirībā no funkcionalitātes testēšanas, kur ir vienkārši definējami ieejas un izejas dati, veiktspējas testēšanai nepieciešama specifiska dokumentācija par testēšanas vidi un sagaidāmajiem rezultātiem. Veiktspējas testēšanas prasībām būtu jābūt testējamām, ar skaidru uzlabošanas potenciālu un kvantitatīvām. Ir divu veidu prasības, vispārīgas, kuras attiecas uz visām sistēmas daļām, un specifiskas, kuras piemērojamas atsevišķām funkcijām.

Testu rakstīšana. Testu veidošana sevī ietver: darījumprasības, kas tiks testētas, soļi šo prasību izpildīšanai, pakalpojumi un operētājsistēmas parametri, kas var ietekmēt testēšanas rezultātu, resursu un to uzstādījumu apzināšana, sagaidāmie rezultāti. Veiktspējas testus ieteicams veikt pakāpeniski, sākot ar nelielām slodzēm un vērtībām, tās katrā solī palielinot. Testus veidojot arī jāuzsver, vai ir kādas sistēmas daļas, kuras nav iespējams automatizēti testēt, piemēram, tādas, kurās daļa no patērētā laika ir saistīta ar lietotāja reakcijas laiku.

Veiktspējas testu automatizēšana. Veiktspējas testi ir ļoti piemēroti automatizēšanai, jo tie savā būtībā iekļauj lielu daudzumu atkārtošanos. Pilnvērtīga, liela apjoma veiktspējas testēšana nebūtu iespējama, ja to nevarētu automatizēt, ņemot vērā iesaistīto lietotāju skaitu un apkopojamo rezultātu apjomu. Automatizētam vajadzētu būt visam testēšanas procesam, ne tikai funkciju testēšanai, tādēļ, lai būtu iespējams veikt testēšanu bez papildus resursu pielikšanas katrā testēšanas reizē. Lai šo nodrošinātu, testiem nevajadzētu saturēt iepriekš definētas, cieti iešūtas vērtības, kuras potenciāli varētu mainīties sistēmai attīstoties, kas radītu papildus ieguldīta darba nepieciešamību, testu veicot atkārtoti.

Veiktspējas testu izpildīšana.

Korekti automatizētiem testiem, izpildīšana neprasa ievērojamu resursu ieguldīšanu to izpildīšanā. Sarežģītākā daļa lielākoties ir iegulto rezultātu savākšanā un analīzē. Veicot testus nepieciešams arī piefiksēt informāciju par testa norisi, lai atkārtotas testēšanas gadījumā būtu dati ar ko salīdzināt. Šī informācija ietver: sākuma un beigu laikus, operētājsistēmas notikumu žurnāla ierakstus, resursu izmantošanas apjomu (procesora, atmiņas, cietā diska, tīkla u.c.), testos iegūtās veiktspējas rādītājus.

Veiktspējas testu rezultātu analīze.

Rezultātu analīze ir svarīgākais un sarežģītākais veiktspējas testēšanas posms. Lai veiksmīgi analizētu iegūtos rezultātus ir nepieciešamas zināšanas par testējamo sistēmu, analītiska domāšana, statistikas zināšanas. Pirms datu analīzes, tie vispirms ir jāapstrādā un jāsakārto.

1. Jāaprēķina vidējās vērtības un standarta novirze.
2. Jāattīra no trokšņainiem datiem, tad iegūtajai datu kopai jāaprēķina vidējās vērtības un standarta novirze.
3. Ja sistēmas darbībā tiek izmantota datu kešdarbe, jānošķir situācijas, kur tā tiek izmantota un, kur dati tiek apstrādāti un tad prezentēti.
4. Jāatšķir veiktspējas dati situācijām, kad visi sistēmas resursi pieejami procesa veikšanai un situācijas, kad procesam paralēli tiek veiktas arī citas darbības.

Pēc datu attīrīšanas un apstrādes var veikt to analīzi. Tos izmantojot var secināt, cik konsekventa ir sistēmas veiktspēja, kādi faktori ietekmē veiktspēju pozitīvi un negatīvi, kādas ir maksimālās slodzes, pie kurām sistēmas darbība ir apmierinoša, kā arī, kuras no noteiktajām veiktspējas prasībām tika izpildītas.

2.3.3. Metode R

Optimizācijas Metode R pirmo reizi tika aprakstīta grāmatā „*Optimizing Oracle Performance*” [10]. Neskatoties uz to, ka uzsvars grāmatai ir uz *Oracle* datubāzu veiktspējas

problēmu risināšanu, aprakstītā un izmantotā metode ir izmantojama plašāku problēmu risināšanai. Metodes R pamatā ir vienkāršs mērķis: „Strādā, lai vispirms novērstu lielāko atbildes laika sastāvdaļu no darījumprocesu loģikas svarīgākās lietotāja darbības". Būtiska atšķirība no citām veiktspējas novērtēšanas metodēm ir, ka šo nevar realizēt nošķirtu no sistēmas darījumprocesiem. Personai, kas šo metodi pielieto ir jābūt padziļinātām zināšanām par sistēmas darbību, lai varētu veikt informētus novērtējumus par iegūtajiem datiem.

Metodes R izpildāmo darbību kopa ir salīdzinoši neliela un vienkārša, salīdzinot ar iepriekš aprakstītajām metodēm.

1. Izvēlēties lietotāju darbības, kurām *darījumiem* nepieciešams uzlabot veiktspēju.
2. Apkopot pareiza apjoma diagnosticējošus datus, kuri palīdzēs identificēt nepietiekamas veiktspējas cēloņus katrai izvēlētajai lietotāju darbībai, kamēr tās darbojas nepietiekami ātri.
3. Izpildīt optimizācijas darbības, kurām ir lielākais potenciālais ieguvums. Ja labākais izvēlētais variants nesniedz pietiekamu ieguvumu, pārtraukt veiktspējas uzlabošanas procesu līdz tiek veiktas kādas citas izmaiņas.
4. Atgriezties pie 1. punkta. [3, 20. lpp.]

Vidējos un lielos uzņēmumos darbinieki bieži ir sadalīti nodaļās, kur katra atbild par savu specifisku daļu, tīkla administratori, datubāzu administratori, lietojumprogrammas izstrādātāji u.c. Kā rezultātā, katra individuālā nodaļa atbild par savas atbildīgās daļas optimizāciju un iespēju robežās to arī optimizē, bet tas ne vienmēr nozīmē, ka visa izstrādātā sistēma ir optimizēta. Metodes R pamatā ir nostādne, ka, veicot mērījumus no lietotāja skatu punkta, tiek iegūta informācija par visiem iesaistītajiem faktoriem. Uzlabojot veiktspēju svarīgākajām darbībām, tiek panākts, ka iegūst maksimālu efektu no ieguldītajiem resursiem.

Var secināt, ka, veicot IS veiktspējas novērtēšanu un uzlabošanu, ir nepieciešams liels informācijas daudzums par esošo sistēmu un tās stāvokli. Šīs informācijas iegūšanai ir izstrādāti dažādi testi, kas piemēroti specifiska šķēluma informācijas iegūšanai un apkopojot dažādas šo datu kopas, iespējams iegūt precīzu pārskatu par IS stāvokli.

No apskatītajām metodēm, pirmās divas ir līdzīgas savā būtība un atšķiras tikai ar specifisku darbību uzsvaru. Savukārt trešā apskatītā metode būtiski atšķiras gan ar testēšanas procesu, gan mērķi. Pirmās divas metodes ir piemērotākas sistēmām, kas vēl ir izstrādes fāzē, kad iespējams veikt visaptverošus sistēmas testus pirms tā tiek nodota lietošanai. Sistēmām, kuras jau ievērojamu laiku tiek izmantotas, un, kurām nav dokumentētu un uzturētu veiktspējas testēšanas plānu, efektīvāka ir Metode R. To izmantojot, var koncentrēt izpētes un uzlabošanas laiku un resursus uz jau noteiktām, svarīgām vietām, kuru labošana sniegtu lielāko labumu.

3. INFORMĀCIJAS SISTĒMAS KAVIS ĀTRDARBĪBAS UZLABOŠANA

3.1. KAVIS sistēmas apraksts

Lietojumprogrammas KAVIS izstrāde tika uzsākta 2005. gadā. Sākotnēji, programmatūra tika izstrādāta ar mērķi, lai analizētu darbu ar klientiem un noteiktu prasības Rīgas pašvaldības klientu pārvaldībai, kā arī sniegto pakalpojumu kvalitātes un pieejamības uzlabošanai. Izstrādātā lietojumprogramma ir RD VIS (Rīgas pilsētas pašvaldības Vienotā informācijas sistēma) sastāvdaļa. Fragments no pirmās versijas specifikācijas: „*Programmatūra tiks veidota, izmantojot Microsoft .NET tehnoloģijas un WEB servisus, kas ļauj paplašināt produkta funkcionalitāti nākotnē. Informācijas sistēmu iespējams integrēt ar RD VIS. Ir iespējama sistēmas pilnveidošana un uzlabošana. Uzkrātos datus iespējams analizēt un tādējādi uzlabot RD piedāvāto pakalpojumu kvalitāti.*" [23, 12. lpp.] Pēc sistēmas izstrādes un ieviešanas tika realizēti plānotie mērķi – analizētas klientu pārvaldības prasības un iespējas uzlabot sniegtos pakalpojumus, kā rezultātā tika veikti tālāki sistēmas paplašinājumi.

KAVIS sistēma ir veidota trīs līmeņos – lietojumprogrammas, darījumprocesu servisu un datubāzes. Tomēr patiesais izpildījums stingri nepieturas pie trīs līmeņu arhitektūras nosacījuma, ka visām darījumprasībām jābūt iekļautām tikai vidējā līmenī, un daļa no darījumprasībām ir realizētas gan lietojumprogrammā, gan datubāzes līmenī.

Sākotnēji izstrādātās lietotjumprogrammas un saistīto daļu tehniskās prasības, salīdzinot tās ar mūsdienās sastopamām, bija nelielas. Citāts no specifikācijas:

„Sistēmai jāspēj pilnvērtīgi darboties, izmantojot sekojošus tehniskos līdzekļus:

- Darba stacija:
    - Procesors: Intel Pentium 200 MHz;
    - Operatīvā atmiņa: 64 MB;
    - Monitora izšķirtspēja: 1024x768, 16 bit krāsām.
- Datu bāžu serveris:
    - Procesors: – Intel Pentium IV 1 GHz;
    - Operatīvā atmiņa: 2 GB;
    - Tīkla karte: 100 Mbit;
    - Diska vieta: 60 GB (SCSI, RAID ar turpmāko paplašināšanas iespēju).
- Aplikāciju serveris:
    - Procesors: – Intel Pentium IV 1 GHz;

- Operatīvā atmiņa: 2 GB;
- Tīkla karte: 100 Mbit;
- Diska vieta: 60 GB (SCSI, RAID ar turpmāko paplašināšanas iespēju).” [23, 150. lpp.]

Apskatot šīs prasības, var skaidri redzēt, ka mūsdienu prasībām, kas saistītas ar vizuāli pievilcīgām grafiskajām saskarnēm, ērtu un ātru izstrādi *.NET Famework* vidē un pieaugošajam pieejamo pakalpojumu skaitam, ir jāaug līdz ar sistēmu. Šobrīd izmantoto serveru specifikāciju var skatīt 3.1. tabulā.

3.1. Tabula

Aparatūras specifikācija

	Izstrādes vide	Produkcija
Pakalpojumu serveri		
Procesors	Intel Xeon E320@1.86 GHz, 4 kodoli	Intel Xeon E5-2630@2.30 GHz, 6 kodoli
Operatīvā atmiņa	6 GB	6 GB
Datubāzes serveri		
Diska vieta	600 GB	5 TB
Datu apjoms	450 GB	3 TB
Operatīvā atmiņa	8 GB	192 GB

Salīdzinot izstrādes vides un produkcijas serverus jāņem vērā, ka produkcijā tiek izmantots slodzes līdzsvarošanas (*load-balancing*) mehānisms un pakalpojumu serveri patiesībā ir divi. Ir skaidri redzams, ka gadu gaitā ir ievērojami augušas gan informāciju sistēmu prasības, gan pieejamie resursi to darbināšanai.

KAVIS sistēma turpināja attīstīties, tai tika paplašināts piedāvāto pakalpojumu skaits, šobrīd KAVIS sistēmas galvenās funkcijas ir:

- Klientu datu uzskaite;
- Klientu un notikumu datu pārskati;
- RD VIS informatīvā sadaļa;
- Zināšanu bāze;
- Darba uzdevumu reģistrēšana un atsekošana;
- Izvērstas sistēmas administrēšanas funkcijas, tai skaitā lietotāju un pilnvarošanas administrēšana pašvaldības portālam;

- Pakalpojumu pieprasījumi un pakalpojumu vadība;
- Dokumentu plūsmas;
- Attīstības komitejas sēžu informatīvā nodrošināšana. [24, 40. lpp.]

3.2. KAVIS ātrdarbības uzlabošana

Līdz šim KAVIS sistēmai nav veikti metodiski veiktspējas novērtējumi un uzlabojumi. Sistēmas izstrādes gaitā, situācijās, ja kāda daļa no funkcijām, balstoties uz lietotāju atsauksmēm, nestrādā pietiekami ātri, ir veiktas nepieciešamās izmaiņas un labojumi, bet process nav dokumentēts vai automatizēts.

Ņemot vērā sistēmas KAVIS apjomu, darbā apskatīts tiks viens no sistēmā piedāvātajiem pakalpojumu moduļiem – infrastruktūras nodeva. Infrastruktūras nodevas moduļa izstrāde uzsākta 2009. gadā. Moduļa ietvaros tiek piedāvāta iespēja aprēķināt īpašumu infrastruktūras nodevu, izrakstīt atbilstošus maksājumu paziņojumus, veikt maksājumu paziņojumu apmaksu, pārskatīt maksājumu paziņojumu apmaksas stāvokli.

Infrastruktūras moduļa darbība ir saistīta ar divu citu sistēmu darbību, BUVE un Repozitoriju. Sistēma BUVE tiek izmantota datu iegūšanai par īpašumiem un projektējumiem. Savukārt Repozitorijs tiek izmantots izveidoto dokumentu glabāšanai un piekļuvei. Veidojot jaunus maksājumu paziņojumu, korekcijas dokumentus un atgādinājuma vai brīdinājuma vēstules, to datnes tiek saglabātas Repozitorijā.

Esošās situācijas rezultātā, KAVIS ātrdarbības uzlabošanai izvēlēta tika Metode R. KAVIS sistēmai šobrīd nav izstrādāts vienots veiktspējas testēšanas plāns un šāda plāna izstrāde ir ārpus šī darba mēroga, kā rezultātā pirmo divu metožu izmantošana nav iespējama. Metodes R izvēle ļauj koncentrēt darbu mazākā sistēmas daļā, izpētot tās darbību, izvēloties dažas problemātiskās vietas, kurās uzlabot darbības ātrumu un veikt labojumus.

Vadoties pēc Metodes R nostādnes, tika izvēlētas darbības, kuras tika analizētas šī darba ietvaros: infrastruktūras nodevas kartiņas atvēršana un maksājumu paziņojumu saraksta ierakstu ielasīšana.

Visi ātrdarbības mērījumi, novērtējumi un sistēmas izmaiņas tika veiktas izstrādes vidē. Neskatoties uz to, ka starp izstrādes vidi un produkcijas vidi ir atšķirības, kā, piemēram, serveru parametri, infrastruktūra (interneta pieslēgums, lokālā tīkla konfigurācija), serveru operētājsistēmas, lietotāju skaits un datu apjoms, iegūtos datus var izmantot par pamatu ātrdarbības novērtēšanai un uzlabošanas veikšanai. Mērījumu veikšanā izmantotais dators ir ievērojami jaudīgāks par vidusmēra KAVIS lietotāja datoru, kas jāņem vērā, analizējot iegūtos datus. Datora parametri:

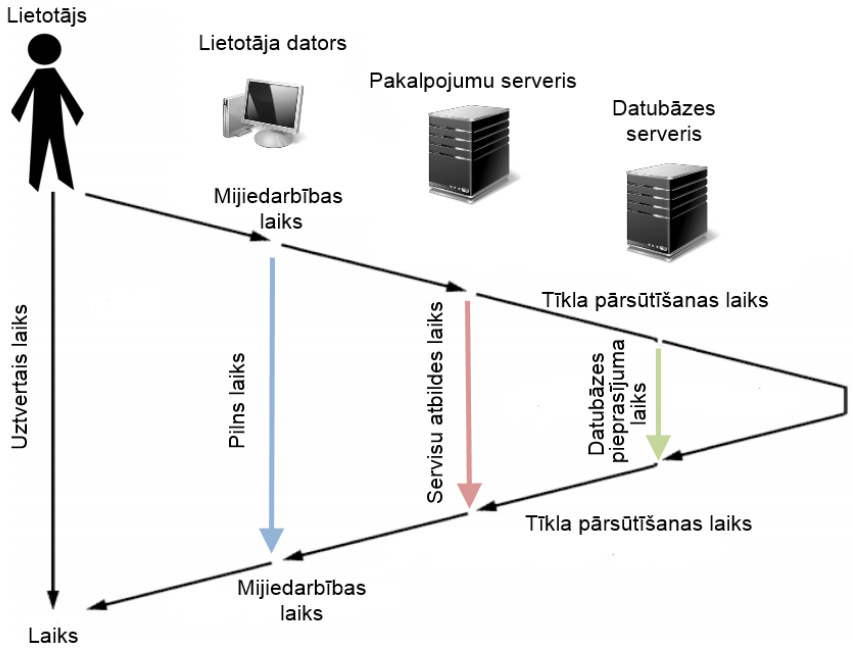
- Procesors: Intel i5-E3450 3.10 GHz
- Operatīvā atmiņa: 8 GB
- Diska vieta: Intel SSD 180 GB

Tā kā mērījumi veikti izstrādes vidē, jāņem vērā zināmi ierobežojumi un faktori. Izstrādes vide, salīdzinot ar produkciju, ir nestabilāka, jo tajā tiek veikta programmas attīstība un izstrāde. Šī iemesla dēļ testi tika veikti laikos, kad sistēmu lietoja neliels daudzums cilvēku. Nebija iespējams iegūt individuālo pakalpojumu lietotņu procesora noslodzi un operatīvās atmiņas patēriņu, jo tās visas apvienotas vienā lietotņu kopā resursu pārvaldīšanai.

Infrastruktūras nodevas kartiņas atvēršana.

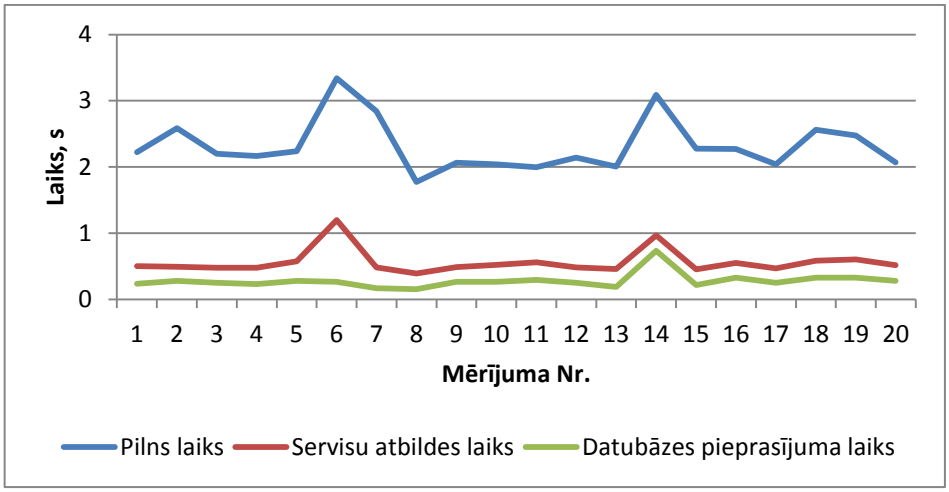
Darbs ar nodevu kartiņām ir viena no primārajām funkcijām strādājot ar infrastruktūras nodevu moduli. Darbinieki ikdienā strādā ar ievērojamu skaitu kartiņām un to lēnā ielāde traucē produktivitātei. Infrastruktūras nodevas kartiņas formu var apskatīt 2. pielikumā.

Mērījumi tika veikti trīs laika periodiem: no pieprasījuma brīža, līdz tiek atvērta kartiņa (pilns laiks); no pakalpojumu servisa izsaukuma brīža līdz atbildes saņemšanai (servisu atbildes laiks); no datubāzes vaicājuma nosūtīšanas brīža līdz datu saņemšanai (datubāzes vaicājuma laiks). Servisu atbildes laiks sevī ietver arī datubāzes pieprasījuma laiku un pilns laiks ietver abu summu un to laiku, kas izmantots datu pārsūtīšanai, apstrādei un attēlošanai. Vizuālu patērētā laika attēlojumu var apskatīt 3.1. attēlā [2, 21. lpp.].



3.1. att. **Patērēto laiku attēlojums**

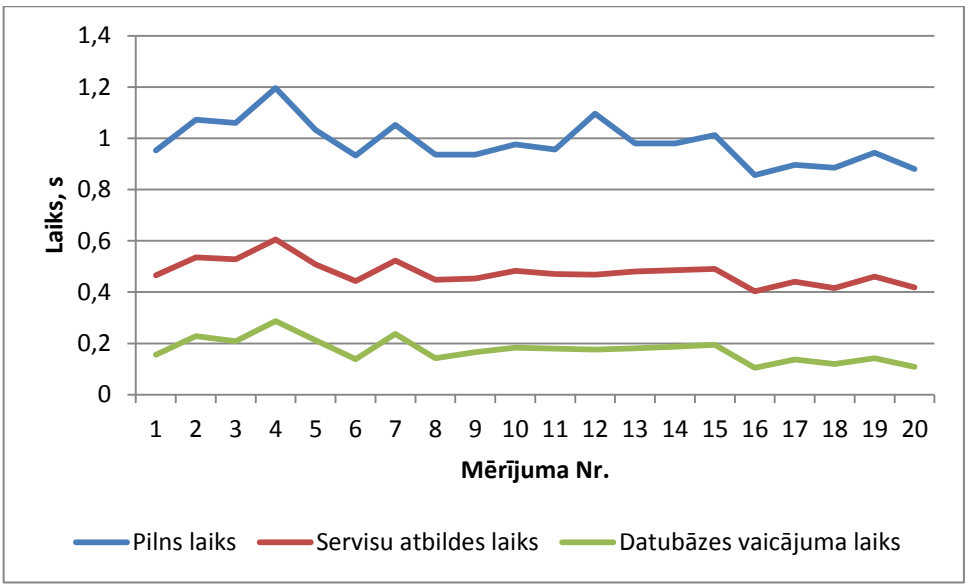
Attiecīgie mērījumi tikai veikti 20 reizes aptuveni 15 minūšu periodā, lai nodrošinātu, ka ārējie apstākļi mazāk ietekmētu rezultātus un to potenciālo ietekmi varētu rezultātos saskatīt. Šī un turpmāko testu iegūtos datus var apskatīt 1. pielikumā.



3.2. att. **Infrastruktūras nodevas kartiņas atvēršanas laiks sekundēs**

Iegūtajos datos var saskatīt, ka kopējais atbildes laiks ir diezgan vienmērīgs, tomēr tajā ir redzamas instances, kad saziņa starp kādu no līmeņiem ir bijusi lēnāka. Tas varētu būt bijis saistīts vai nu ar noslogotu iekšējo tīklu un lēnāku datu pārraidi, vai arī ar noslogotiem attiecīgajiem serveriem, kā rezultātā tiem bija nepieciešams ilgāks laiks atbildes sagatavošanai. Neskatoties uz to, šīs novirzes sastāda mazāk par vienu piektdaļu no kopējā atbildes laika un pie kopējiem secinājumiem var netikt ņemtas vērā. Lielāko atbildes laiku sastāda periods pēc tam, kad visi nepieciešamie dati jau ir saņemti. Jāņem arī vērā, ka dators, uz kura darbināta lietojumprogramma, ir jaudīgāks par tipisku sistēmas lietotāja datoru, kuram šī pati darbība iespējams prasa vēl ilgāku laiku.

Izmantojot šo informāciju, tālāka analīze tika veikta funkcijām, kas apstrādā saņemtos infrastruktūras nodevas kartiņas datus. Noskaidrots, ka pēc datu saņemšanas tiek veidotas sarežģītas datu struktūras, kurās glabā visu par kartiņu saistīto informāciju – klienta, maksātāja datus, objektu datus, maksājumu datus, pievienotos dokumentus un korekcijas u.c., un nepieciešams ilgs laiks to korektai izveidei. Tomēr, šajā procesā atrastas zināmas nepilnības, veidojot kartiņas kopijas un kopējot datus. Šīs nepilnības novērstas un, pēc labojumu veikšanas, veikti atkārtoti mērījumi.



3.3. att. **Infrastruktūras nodevas kartiņas atvēršanas laiks sekundēs, pēc labojumiem**

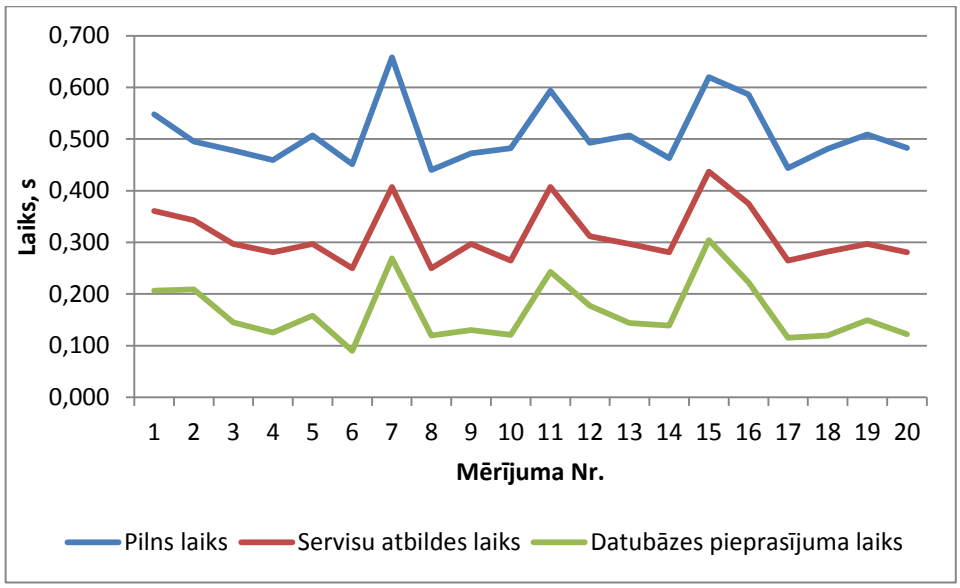
Redzams, ka infrastruktūras nodevas kartiņas atvēršanas procesa optimizācija ir bijusi veiksmīga un datu objektu izveidošana vairs aizņem tikai aptuveni pusi no kopējā atbildes laika. Atvēršanas laiks iekļaujas arī vienas sekundes robežās, kas ir noteiktā prasība šādām darbībām.

Maksājumu paziņojumu saraksta ierakstu ielasīšana.

Maksājumu paziņojumu saraksts dod iespēju KAVIS lietotājam atlasīt izveidotos infrastruktūras maksājumu paziņojumus, izvēloties dažādus filtrus. Pēc filtru izvēles un pogas Meklēt nospiešanas, vēlamā filtra parametri tiek nosūtīti pakalpojumu serverim, kur tie tiek noformēti kā *SQL* vaicājumi un nosūtīti uz datubāzi, un atpakaļ tiek saņemts saraksts ar maksājumu paziņojumiem, kuri atbilda nosacījumiem. Maksājumu paziņojumu saraksta formu iespējams apskatīt 2. pielikumā.

Analogi kartiņas atvēršanas mērījumiem, arī šajā situācijā, patērētais laiks tika mērīts trijiem darbības posmiem.

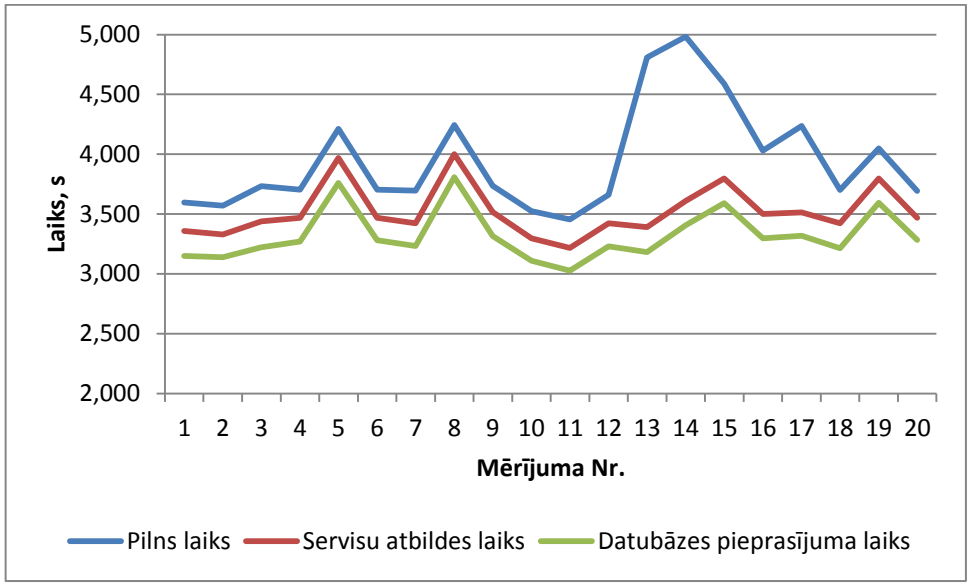
Pirmie mērījumi tika veikti ar nelielu datu apjomu, izvēloties viena mēneša periodu. Mērījumu rezultātus var redzēt 3.1. attēlā, saņemtais datu apjoms ir 23 ieraksti.



3.4. att. **Maksājumu paziņojumu saraksta ielādes laiki (mēnesis)**

Kā redzams, kopējā atbildes laika vidējā vērtība ir 0,5 sekundes, kas ir pieņemams atbildes laiks šādai funkcijai. Katra no pieprasījuma daļām sastāda aptuveni 1/3 no kopējā patērētā laika, kas nozīmē, ka datu pārsūtīšana starp līmeņiem un datu pieprasījuma veikšana šajā situācijā tiek veikta aptuveni vienādā laika periodā.

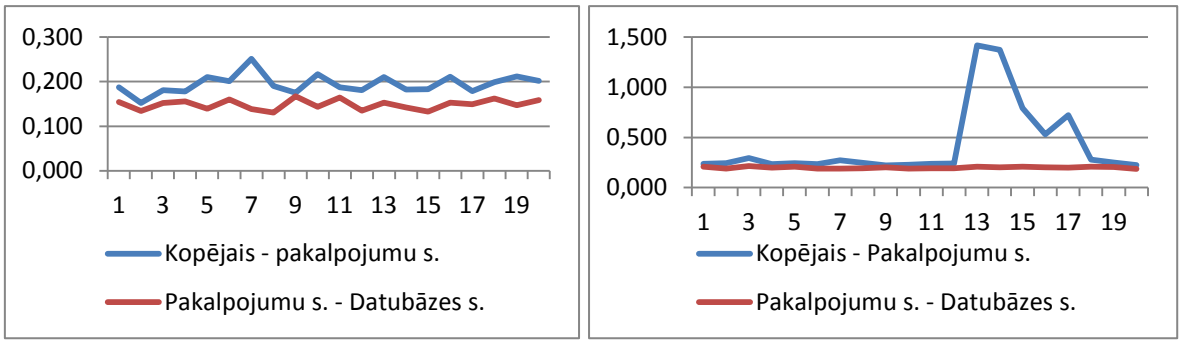
Lai pārbaudītu, kā kopējo atbildes laiku ietekmē lielāka apjoma dati, nākamais tests tika veikts ar viena gada periodu, datu apjoms ir 221 ieraksts.



3.5.att. **Maksājumu paziņojumu saraksta ielādes laiki (gads)**

Iegūtajos rezultātos var ļoti uzskatāmi redzēt, ka ielādējot lielāka apjoma datus, ievērojami krītas ātrdarbība. Redzams, ka praktiski viss pieaugums ir saistīts ar ilgāku datubāzes pieprasījuma laiku, norādot, kur tālāk meklēt problēmas un tās labot. Salīdzinot datu pārraides laikus starp līmeņiem, atšķirība starp pirmo un otro testu sēriju ir neliela, 0,05 sekunžu apjomā gan starp lietojumprogrammu un pakalpojumu serveri, gan pakalpojumu

serveri un datubāzes serveri. Mērījumu sērijā, kurā tika apskatīts gada periods, var redzēt, ka 13.-17. mērījumos bija vērojama ievērojami lēnāka saziņa starp lietojumprogrammas līmeni un pakalpojumu serveri, bet tā uz kopējo mērījumu fona ir anomālija (īslaicīgi pārslogots lokālais tīkls vai kāds cits iemesls).



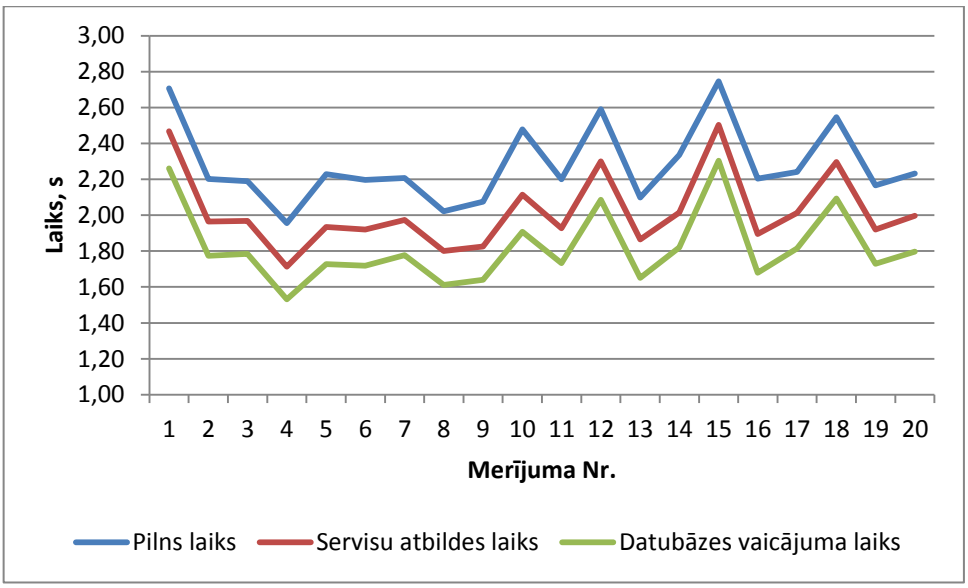
(a) (b)

3.6.att. **Maksājumu paziņojumu saraksta ielādes laiku starpības (sekundēs):**

(a) mēneša mērījumi, (b) gada mērījumi

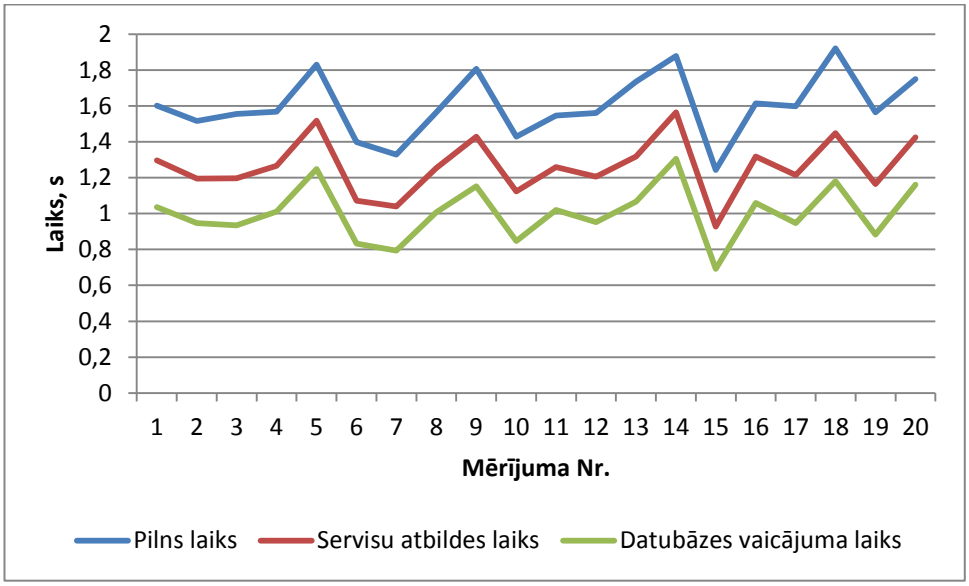
No iegūtajiem datiem var secināt, ka būtiskākā ātrdarbības zaudēšanas vieta ir datu iegūšana no datubāzes. Visi pārējie saziņas posmi un datu apstrāde abās testu sērijās ir līdzvērtīgi, ņemot vērā lielāko datu apjomu otrajā testu sērijā.

Ātrdarbības uzlabošanai tika izpētīts atbilstošais *SQL* vaicājums. Izmantojot *PL/SQL Developer* rīku iespējas, tika analizētas vaicājuma sastāvdaļas un to izpildes laiki, rīka sniegto informāciju var apskatīt 3. pielikumā. Analizējot rezultātu, saskatāmās problēmas ir vairāki *Table Access Full* (pilna tabulas piekļuve) atsevišķām tabulām. Tas liecina, ka, izpildot vaicājumu, tiek veikta norādīto tabulu pilna datu pārlase, nevis izmantota ātrāka datu atlase izmantojot indeksus. Zinot šo informāciju, tika noskaidrots, kuriem datu laukiem trūkst ārējās atslēgas uz citiem tabulu laukiem, un šīs atslēgas tika pievienotas. Pēc izmaiņu veikšanas tika atkārtoti mērījumi gada periodam.



3.7.att. **Maksājumu paziņojumu saraksta ielādes laiki pēc pirmajiem labojumiem (gads)**

Mērījumos redzams, ka uzlabojumi ir saskatāmi, tomēr kopējais atbildes laiks joprojām nav pietiekami labs. No kopējā atbildes laika, datubāzes vaicājuma izpilde turpina būt ilgākais posms. Turpinot vaicājuma analīzi, noskaidrots, ka daļa no nepieciešamās informācijas tiek iegūta ar datubāzes procedūras palīdzību. Šai procedūrai tiek padots maksājuma identifikators un saņemta informācija par attiecīgā maksājuma apmaksas stāvokli un nokavējuma naudu. Problēmu rada tas, ka šī procedūra tiek izsaukta katram ierakstam atsevišķi. Lai to novērstu, vaicājums tika pārrakstīts, lai vispirms atrastu interesējošos maksājumus, izveidotu sarakstu ar šo maksājumu identifikatoriem un attiecīgajai procedūrai padotu visu sarakstu vienlaikus. Pēc informācijas saņemšanas, tā tiek apvienota ar pārējiem datiem un atgriezta lietotājam.



3.8.att. **Maksājumu paziņojumu saraksta ielādes laiki pēc otrajiem labojumiem (gads)**

Analizējot infrastruktūras maksājumu paziņojumu saraksta formas darbību, tika veiksmīgi noteikti vairāki ātrdarbību traucējoši faktori, kas saistīti ar datubāzes vaicājumu.

Noskaidrots, ka datubāzes atsevišķiem ierakstiem trūka ārējās atslēgas un daļa no datiem tika iegūta neefektīvā veidā. Pēc šo faktoru identificēšanas, tika veikta atbilstošo daļu labošana, kā rezultātā datu ielādes laiks arī lielākiem datu apjomiem ir apmierinošs.

Neskatoties uz ierobežotajiem datiem, kas tika iegūti par sistēmas moduļa darbību, bija iespējams noteikt vairākas problēmas, kas bija par lēnās veiktspējas cēloņiem. Novēršot noteiktos traucējumus, tika veiksmīgi uzlabota izvēlēto formu darbība.

REZULTĀTI

Darba gaitā tika izpildīti galvenie izvirzītie mērķi:

- Izpētītas plašāk izplatītās informācijas sistēmu arhitektūras
- Noskaidroti informācijas sistēmu ātrdarbības traucējumu cēloņi
- Izpētītas metodes ātrdarbības novērtēšanai informācijas sistēmā, izvēlēta Metode R KAVIS ātrdarbības noteikšanai
- Izmantojot iegūtos ātrdarbības rādītājus, veikti uzlabojumi KAVIS sistēmai.

Darbu izstrādājot veiksmīgi uzlabota KAVIS sistēmas infrastruktūras moduļa atsevišķu funkciju ātrdarbība. Iegūtos datus un novērtējumus iespējams izmantot tālākai infrastruktūras un citu moduļu ātrdarbības novērtēšanai un uzlabošanai.

SECINĀJUMI

Bakalaura darba izstrādes laikā gūtās atziņas formulētas secinājumos:

1. Gadu gaitā izveidotas daudzas un dažādas informāciju sistēmu arhitektūras, kur katra piemērota savādāku problēmu risināšanai.
2. Daudzlīmeņu informācijas sistēmu arhitektūra ir viena no visplašāk izplatītajām, tā sevī ietver arī viena līmeņa sistēmas, kas visbiežāk ir vienkāršas lietojumprogrammas, un daudzlīmeņu, kas ir liela apjoma daudzlietotāju sistēmas.
3. Daudzlīmeņu informāciju sistēmas ir ievērojami sarežģītāk projektēt un realizēt, salīdzinot ar viena vai divu līmeņu, tomēr tām ir būtiskas priekšrocības mērogojamībā, maksimālo lietotāju ziņā un pieejamībā.
4. Informācijas sistēmu ātrdarbības problēmu cēloņi ir ļoti dažādi, tie visbiežāk saistīti ar sliktu projektējumu, nepareizu realizāciju vai nepiemērotu aparatūru.
5. Ātrdarbības problēmu risināšanai ir pieejamas dažādas metodes, tomēr to nekorekts pielietojums var radīt citas, tai skaitā ātrdarbības, problēmas.
6. Ātrdarbības problēmu novērtēšanai pieejami dažādi testi, kā sākotnējie testi, slodzes testi, stresa testi, ilgtermiņa testi, apjoma testi un atbildes laika testi.
7. Katrs no testa veidiem sniedz atsevišķu daļu no sistēmas ātrdarbības kopainas, un ieteicams tos lietot paralēli, lai iegūtu pilnvērtīgu priekšstatu par sistēmas darbību.
8. Ātrdarbības testēšanas metodes var nosacīti sadalīt divās daļās, visaptverošās un specifiskās. Visaptverošās metodes tiek izmantotas vispārēja sistēmas stāvokļa noteikšanai, no kura var izdarīt secinājumus par atsevišķu sistēmu darbību. Specifiskās metodes tiek izmantotas jau zināmu sistēmas daļu analīzei.
9. Ir sarežģīti sākt veidot testēšanas dokumentāciju liela apjoma sistēmai, kurai šādas dokumentācijas nav.
10. Zinot specifiskas darījumprocesu darbības, kurām nepieciešams uzlabot ātrdarbību, izdevīgi pielietot Metodi R to analīzei un pēc novērtējuma iegūšanas arī uzlabošanai.

AVOTU UN IZMANTOTĀS LITERATŪRAS SARAKSTS

1. Jalote. P, *A Concise Introduction to Software Engineering*, London, Springer-Verlag, 2008, 296 pg.

2. Asböck S, *load testing for eConfidence*, Segue Software, 2000, 142 pg.

3. Millsap C., Holt J., *Optimizing Oracle Performance*, O'Reilley Media, 2003, 388 pg.

4. James K. L., *Software Engineering*, PHI Learning, 2008, 388 pg.

5. Sommerville I., *Software Engineering 9th ed.*, Addison-Wesley, 2010, 792 pg.

6. Pressman R. S., *Software engineering A Practitioner's Approach 7th ed.*, McGraw-Hill Higher Education, 2009, 928 pg.

7. Braude E. Bernstein M., *Software Engineering: Modern Approaches 2nd ed.*, Wiley, 2010, 800 pg.

8. Naik K., Priyadarshi T., *Software Testing and Quality Assurance Theory and Practice*, Wiley, 2008, 616 pg.

9. Gopalaswamy R., Desikan S., *Software Testing: Principles and Practices*, Pearson Education, 2009, 487 pg.

10. Molyneaux I., *The Art of Application Performance Testing*, O'Reilly Media, 2009, 158 pg.

11. *Why Multi-Tier?* [tiešsaiste]. – [atsauce 01.05.2014.] Pieejams: http://www.remobjects.com/da/why-multitier.aspx

12. Plessis J., *Performance Testing Methodology*, Micro to Mainframe Offices, 2008. – [atsauce 31.05.2014.]. Pieejams: http://www.stickyminds.com/sites/default/files/article/file/2013/XUS206223565file1_0.pdf

13. *Three-tier Application Model* [tiešsaiste]. – [atsauce 01.05.2014.] Pieejams: http://msdn.microsoft.com/en-us/library/aa480455.aspx

14. Booth. J, *Multi-layered Architectural Design* [tiešsaiste]. – [atsauce 01.05.2014.] Pieejams: http://www.jamesbooth.com/weblayersi.htm

15. *Client/Server and the N-Tier Model of Distributed Computing* [tiešsaiste]. – [atsauce 01.05.2014.] Pieejams: http://n-tier.com/articles/csovervw.html

16. *N-Tier – The Future of Computing* [tiešsaiste]. – [atsauce 01.05.2014.] Pieejams: http://www.n-tier.com/Future.html

17. *N-Tier Data Applications Overview* [tiešsaiste]. – [atsauce 01.05.2014.] Pieejams: http://msdn.microsoft.com/en-us/library/bb384398.aspx

18. Kambalyal C., *3-Tier Architecture* – [atsauce 01.05.2014.] Pieejams: http://channukambalyal.tripod.com/NTierArchitecture.pdf

19. *Architectural Patterns and Styles* [tiešsaiste]. – [atsauce 01.05.2014.] Pieejams: http://msdn.microsoft.com/en-us/library/ee658117.aspx

20. *Physical Tiers and Deployment* [tiešsaiste]. – [atsauce 01.05.2014.] Pieejams: http://msdn.microsoft.com/en-us/library/ee658120.aspx

21. *N-Tier Architecture Application - An Overview* [tiešsaiste]. – [atsauce 01.05.2014.] Pieejams: http://www.peterindia.net/N-tierApplicationsView.html

22. *Design Guidelines for Application Performance* [tiešsaiste]. – [atsauce 01.05.2014.] Pieejams: http://msdn.microsoft.com/en-us/library/ff647801.aspx

23. Klētnieks L., Žeiris E., Actiņš J., *Klientu attiecību vadības informācijas sistēma „KAVIS”* ZZ Dats, 2005

24. Strazdiņš M., *Rīgas domes Vienotā informācijas sistēma,* ZZ Dats, 2009

PIELIKUMI

1. pielikums

Infrastruktūras nodevas kartiņas atvēršanas laiks sekundēs

2,223	2,586	2,199	2,168	2,236	3,339	2,843	1,776	2,063	2,041	1,996	2,144	2,005	3,090	2,278	2,272	2,038	2,561	2,474	2,070
0,501	0,495	0,478	0,478	0,575	1,198	0,486	0,391	0,488	0,522	0,561	0,484	0,461	0,967	0,453	0,552	0,467	0,585	0,604	0,518
0,235	0,282	0,250	0,234	0,281	0,266	0,172	0,156	0,266	0,266	0,297	0,250	0,188	0,734	0,219	0,328	0,250	0,328	0,328	0,282

Infrastruktūras nodevas kartiņas atvēršanas laiks sekundēs, pēc labojumiem

0,952	1,072	1,060	1,196	1,032	0,932	1,052	0,936	0,936	0,976	0,956	1,096	0,980	0,980	1,012	0,856	0,896	0,884	0,944	0,880
0,465	0,535	0,528	0,605	0,508	0,443	0,523	0,448	0,453	0,483	0,470	0,468	0,480	0,485	0,490	0,403	0,440	0,415	0,460	0,418
0,156	0,229	0,208	0,288	0,212	0,139	0,237	0,142	0,166	0,183	0,179	0,175	0,181	0,187	0,195	0,104	0,137	0,120	0,142	0,109

Maksājumu paziņojumu saraksta ielādes laiki (mēnesis)

0,548	0,495	0,478	0,459	0,507	0,451	0,658	0,440	0,472	0,482	0,594	0,493	0,507	0,463	0,620	0,586	0,444	0,481	0,509	0,483
0,361	0,343	0,297	0,281	0,297	0,250	0,407	0,250	0,297	0,265	0,407	0,312	0,297	0,281	0,437	0,375	0,265	0,282	0,297	0,281
0,206	0,209	0,145	0,125	0,158	0,090	0,269	0,119	0,130	0,121	0,243	0,177	0,144	0,139	0,304	0,222	0,115	0,120	0,150	0,122

Maksājumu paziņojumu saraksta ielādes laiki (gads)

3,598	3,571	3,732	3,704	4,211	3,704	3,695	4,245	3,737	3,525	3,456	3,662	4,808	4,982	4,590	4,030	4,236	3,700	4,048	3,694
3,360	3,328	3,438	3,469	3,969	3,469	3,422	4,000	3,516	3,297	3,218	3,422	3,391	3,609	3,797	3,500	3,515	3,422	3,797	3,469
3,151	3,138	3,223	3,270	3,761	3,281	3,232	3,808	3,315	3,110	3,027	3,231	3,181	3,406	3,590	3,298	3,317	3,215	3,593	3,283

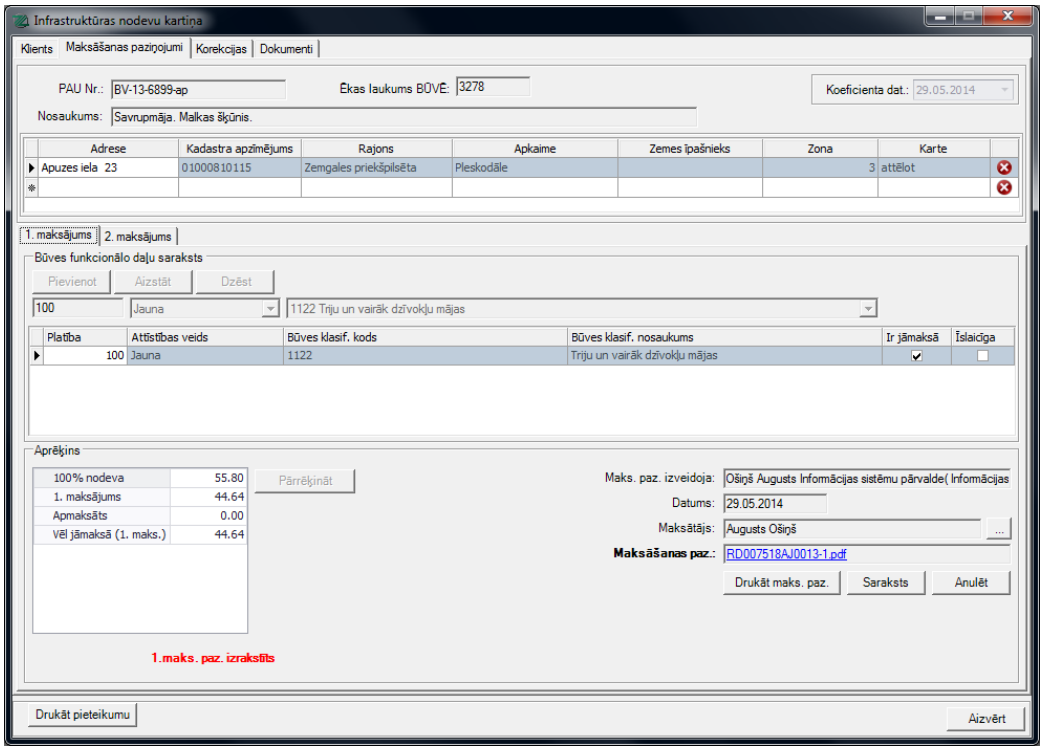
Maksājumu paziņojumu saraksta ielādes laiki pēc pirmajiem labojumiem (gads)

2,707	2,201	2,191	1,953	2,234	2,198	2,200	2,024	2,070	2,480	2,199	2,588	2,095	2,336	2,747	2,210	2,240	2,549	2,166	2,234
2,468	1,971	1,970	1,718	1,934	1,919	1,970	1,794	1,832	2,113	1,922	2,300	1,863	2,013	2,502	1,889	2,017	2,297	1,920	2,004
2,258	1,772	1,784	1,525	1,725	1,711	1,778	1,608	1,638	1,906	1,728	2,085	1,650	1,818	2,310	1,685	1,818	2,090	1,732	1,795

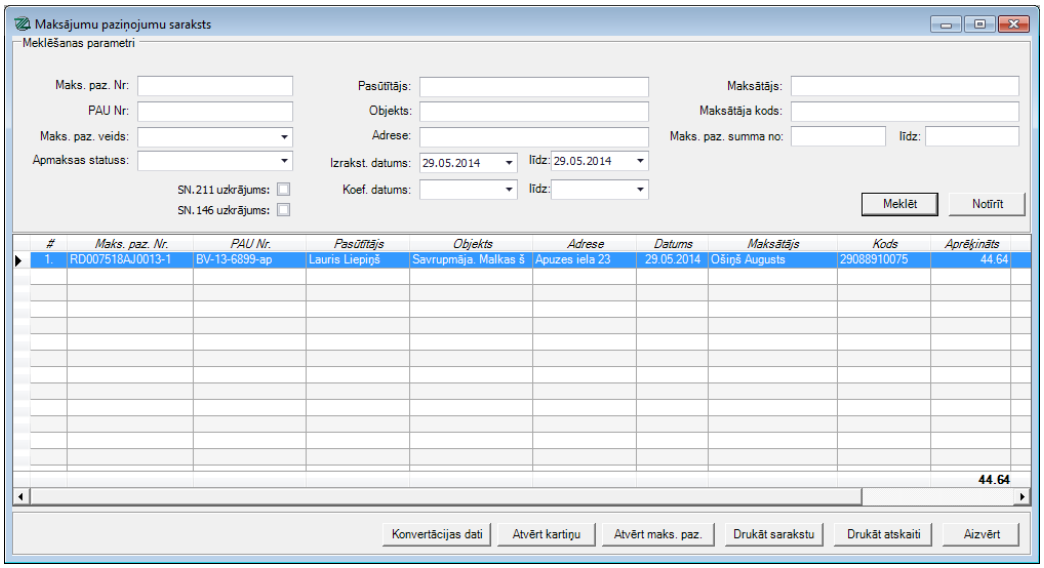
Maksājumu paziņojumu saraksta ielādes laiki pēc otrajiem labojumiem (gads)

1,602	1,515	1,556	1,568	1,830	1,398	1,328	1,563	1,807	1,428	1,546	1,559	1,736	1,879	1,243	1,613	1,597	1,921	1,565	1,749
1,296	1,195	1,196	1,266	1,518	1,072	1,040	1,253	1,429	1,123	1,260	1,205	1,318	1,563	0,928	1,317	1,214	1,448	1,164	1,425
1,036	0,947	0,935	1,011	1,248	0,833	0,794	1,007	1,152	0,847	1,021	0,953	1,066	1,305	0,692	1,060	0,946	1,181	0,883	1,161

Infrastruktūras nodevu kartiņa



Maksājumu paziņojumu saraksts



Maksājumu paziņojumu saraksta vaicājuma analīze

Description	Object owner	Object name	Cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = CHOOSE			365	1	807
SORT AGGREGATE				1	14
TABLE ACCESS FULL	INFRA	IN_TERMINI	20	1	14
SORT AGGREGATE				1	7
INDEX RANGE SCAN	INFRA	IN_ATGAD_VESTULES_IDX	1	3	21
NESTED LOOPS			365	1	807
NESTED LOOPS			364	1	489
MERGE JOIN CARTESIAN			363	1	433
HASH JOIN			340	1	429
VIEW	INFRA		27	487	191391
UNION-ALL					
NESTED LOOPS					
NESTED LOOPS			5	1	15
TABLE ACCESS FULL	INFRA	IN_IZMAINA	3	3	24
INDEX UNIQUE SCAN	INFRA	IN_KOREKCIJA_ID_PK	0	1	
TABLE ACCESS BY INDEX ROWID	INFRA	IN_KOREKCIJA	1	1	7
TABLE ACCESS FULL	INFRA	IN_1_MAKSAJUMS	17	478	27246
NESTED LOOPS					
NESTED LOOPS			4	2	26
TABLE ACCESS FULL	INFRA	IN_IZMAINA	2	7	42
INDEX UNIQUE SCAN	INFRA	IN_KOREKCIJA_ID_PK	0	1	
TABLE ACCESS BY INDEX ROWID	INFRA	IN_KOREKCIJA	1	1	7
TABLE ACCESS FULL	INFRA	IN_2_MAKSAJUMS	10	9	243
TABLE ACCESS FULL	CRM	KLIENTS	312	150270	5409720
BUFFER SORT			52	4358	17432
TABLE ACCESS FULL	INFRA	IN_PROJEKTS	23	4358	17432
TABLE ACCESS BY INDEX ROWID	INFRA	IN_BUVE	1	1	56
INDEX UNIQUE SCAN	INFRA	IN_BUVE_PK	0	1	
REMOTE		EBP_PROJEKTI	1	1	318

REĢISTRĀCIJAS LAPA

Bakalaura darbs „Veiktspējas uzlabošana informācijas sistēmās ar daudzlīmeņu arhitektūru” izstrādāts Latvijas Universitātes Datorikas fakultātē. Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantojot tikai tajā norādītos informācijas avotus un iesniegtā darba elektroniskā kopija atbilst izdrukai. Piekrītu sava darba publicēšanai internetā:

Autors: Augusts Ošiņš _____ 02.06.2014
St. apl. Nr. ao08090

Ar savu parakstu apliecinu, ka esmu lasījis augšminēto bakalaura darbu un atzīstu to par piemērotu aizstāvēšanai Latvijas Universitātes datorzinātņu bakalaura studiju programmas gala pārbaudījumu komisijas sēdē.

Darba vadītājs: M. inž. Dainis Inkins _____ 02.06.2014

Darbs iesniegts Datorikas fakultātē 2014. gada 2. jūnijā.

Ar savu parakstu apliecinu, ka darba versija ir augšupielādēta LU informatīvajā sistēmā.

Metodiķe: Ārija Sproģe _____

Recenzents: _____

Darbs aizstāvēts bakalaura darbu gala pārbaudījumu komisijas sēdē

2014. gada __. jūnijā prot. Nr. _____

Komisijas sekretārs: _____