LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

SLIMNIEKU UZSKAITES SISTĒMA, IZMANTOJOT HL7 STANDARTU

KVALIFIKĀCIJAS DARBS

Autors: **Andris Lācis**
Stud. apl. nr.: al10093
Vadītājs: Dr. dat. Māris Vītiņš

Rīga 2012

ANOTĀCIJA

Kvalifikācijas darbā izstrādāts elektroniskā pacientu uzskaites reģistra modulis, izmantojot HL7 standartu – spraudņu dzinējs un spraudņi klasifikatoru publicēšanai un izplatīšanai. Modulis paredzēts, lai ļautu sistēmas lietotājiem pašiem izstrādāt moduļus klasifikatoru izplatīšanai un publicēšanai specifiski savām vajadzībām.

Darbs izstrādāts C# programmēšanas valodā pēc ūdenskrituma modeļa, papildus izmantojot SQL.

C-Sharp, SQL, HL7, e-veselība, spraudņi

ABSTRACT

This paper describes the development of a module for the electronic patient register using the HL7 standard – a plugin support system and plugins for publishing and distributing classifiers. The module provides users with the option to develop their own plugins for classifier publishing and distribution for their own specific needs.

The software is developed in C# programming language using the waterfall model, additionally using SQL.

C-Sharp, SQL, HL7, e-health, plugins

SATURA RĀDĪTĀJS

VĀRDNĪCA

Nosaukums	**Apraksts**
AppDomain	Lietotnes domēns. Microsoft tehnoloģija koda izpildes nodalīšanai. Papildus līmenis starp procesu un pavedieniem. (1)
ER	Entity-Relationship modelis datubāzes entītiju savstarpējo attiecību attēlošanai
II	Instance Identifier – HL7 datu tips (3)
Klasifikators	Klasificējamu datu kopa
Klasifikatora kodu sistēma	Unikāls klasifikatora identifikators
KLOC	Koda rindiņu skaits (tūkst.)
PPA	Programmatūras projektējuma apraksts
PPS	Programmatūras prasību specifikācija
PK	Primārā atslēga – datubāzes tabulas ieraksta unikāls identifikators
Spraudnis	Dinamiski ielādējams programkods, kuru var istrādāt ārējie izstrādātāji specifiski savām vajadzībām
ST	String – HL7 datu tips (3)

IEVADS

Darbā tiek izstrādāts pacientu uzskaites reģistra modulis, kas pieļauj ārējo izstrādātāju izstrādātu spraudņu ielādēšanu un izpildi.

Galvenās problēmas, kas darbā tiek apskatītas, ir dinamiska moduļu ielāde un izlāde, domēnu konfigurēšana un izveidošana, starpdomēnu komunikācija, Windows pakalpju uzstādīšana un izsaukšana.

Darba mērķis ir izstrādāt sistēmu, kas pēc datubāzē pieejamajiem datiem patstāvīgi atrod spraudņus un ielādē un izpilda tos savstarpēji neatkarīgos atmiņas apgabalos, tādejādi nodrošinot sistēmas stabilitāti.

Darbs tiek izstrādāts pēc ūdenskrituma modeļa izstrādes vidē Microsoft Visual Studio 2010. Datu apmaiņai tiek izmantots XML, datubāzes vadībai – Microsoft SQL Server 2008. Spraudņu un sistēmas savstarpējai nodalīšanai tiek izmantoti lietotnes domēni (AppDomains), savukārt sistēmas nodalīšanai no pārējā pacientu uzskaites reģistra tiek izmantotas Windows pakalpes, nodrošinot to darbību atsevišķos procesos. (1)

Galvenais uzsvars darbā tiek likts tieši uz procesu, pavedienu un domēnu pārvaldību, nevis datubāzes struktūru vai projekta tvērumu.

1. PROGRAMMATŪRAS PRASĪBU SPECIFIKĀCIJA

1.1 Ievads

1.1.1 Dokumenta nolūks

Dokumenta nolūks ir aprakstīt pacientu uzskaites sistēmas spraudņu moduļa prasības, kas tiks izmantotas produkta projektēšanā un izstrādē.

1.1.2 Darbības sfēra

Dokuments tiks izmantots par pamatu tālākai spraudņu moduļa izstrādei. Spraudņu modulis ir nepieciešams pacientu uzskaites sistēmas darbībai, un šajā dokumentā ir izklāstīta šī moduļa realizācija, kas ir saistoša spraudņu izstrādātājiem.

1.1.3 Termini un pieņemtie apzīmējumi

Skatīt nodaļu Vārdnīca [4].

1.1.4 Saistība ar citiem dokumentiem

PPS sastādīts balstoties uz LVS 68:1996 Programmatūras prasību specifikācijas ceļvedi.

1.2 Vispārējs apraksts

1.2.1 Produkta perspektīva

Pacientu uzskaites sistēmā ir jāizveido spraudņu modulis, kas nodrošinās iespēju ārējiem izstrādātājiem izstrādāt savus spraudņus klasifikatoru vērtību publicēšanai un izplatīšanai. Modulis jāveido tā, lai tas izmantotu HL7 standartu. Spraudņu modulis ir atkarīgs un ietilpst lielākā projektā, kas netiek aprakstīts šajā dokumentā.

1.2.2 Produkta funkcijas

Risinājumam jāatbalsta sekojoša funkcionalitāte:

- Spraudņu saraksta iegūšana
- Spraudņu ielāde un izpilde
- Sagatavots publicēšanas spraudnis
- Sagatavots izplatīšanas spraudnis

1.2.3 Vispārējie ierobežojumi

Datoram, uz kura sistēma tiks uzstādīta, ir jābūt aprīkotam ar .NET 4.0 ietvaru.

1.2.4 Pieņēmumi un atkarības

Spraudņu modulis ir atkarīgs no pacientu uzskaites reģistra projekta, jo tajā notiek spraudņu moduļa izsaukumi, kā arī klasifikatoru vērtību apstrāde.

1.3 Funkcionālās prasības

1.3.1 Spraudņu saraksta iegūšana

Sistēmai jānodrošina ārējā pakalpe spraudņu saraksta iegūšanai.

Ievade:

- Klasifikatora kodu sistēma (neobligāts, varchar(50))

Apstrāde:

- Datubāzē atrod visus spraudņu tabulas ierakstus ar padoto kodu sistēmu un tos atgriež

Izvade:

- Spraudņu saraksts

1.3.2 Spraudņu ielāde un izpilde

Sistēmai jānodrošina ārēju spraudņu ielāde un izpilde. Jānodrošina vienota saskarne spraudņu izstrādei.

Sistēmai jāspēj patstāvīgi atrast nepieciešamos spraudņus pēc datubāzē glabātās informācijas.

1.3.3 Iebūvēts publicēšanas spraudnis

Sistēmai jānodrošina iepriekš sagatavots klasifikatora vērtību publicēšanas spraudnis, kas veidots pēc vienotās saskarnes.

Spraudnim pēc palaišanas ir nepārtraukti jāstrādā, līdz tas tiek izslēgts.

Uz laika notikumu spraudnim ir jāspēj pārbaudīt, vai publiskajā konfigurācijā norādītajā mapē nav parādījusies jauna vērtību datne.

Ja spraudnis atrod jaunu vērtību datni, tam jāspēj to nolasīt un vērtības nosūtīt pacientu uzskaites reģistram, kas izpildīs vērtību publicēšanu.

1.3.4 Iebūvēts izplatīšanas spraudnis

Sistēmai jānodrošina iepriekš sagatavoti klasifikatora vērtību izplatīšanas spraudņi – pilnu klasifikatoru vērtību izplatīšanai un vērtību izmaiņu publicēšanai – kas veidoti pēc vienotām saskarnēm.

Pēc pacientu uzskaites reģistra izsaukuma spraudnim jāspēj no publiskās konfigurācijas nolasīt ceļu, kur vērtības izplatīt. Nolasītajā vietā jāizveido datne ar padotajām vērtībām.

Pēc izplatīšanas beigām spraudnis ir jāizlādē.

1.4 Veiktspējas prasības

Sistēma nedrīkst kavēt pārējā projekta darbību, tādēļ potenciāli ilgstošas izpildes funkcijas ir jāizpilda asinhroni.

2. PROGRAMMATŪRAS PROJEKTĒJUMA APRAKSTS

2.1 Ievads

2.1.1 Dokumenta nolūks

Spraudņu moduļa programmatūras projektējuma apraksta (PPA) nolūks ir sniegt detalizētu sistēmas uzbūves un tās darbības principu aprakstu.

2.1.2 Darbības sfēra

Dokuments tiks izmantots par pamatu spraudņu moduļa izstrādei.

2.1.3 Termini un pieņemtie apzīmējumi

Skatīt nodaļu Vārdnīca [4].

2.1.4 Saistība ar citiem dokumentiem

2.2 Dekompozīcijas apraksts

2.2.1 Moduļu dekompozīcija

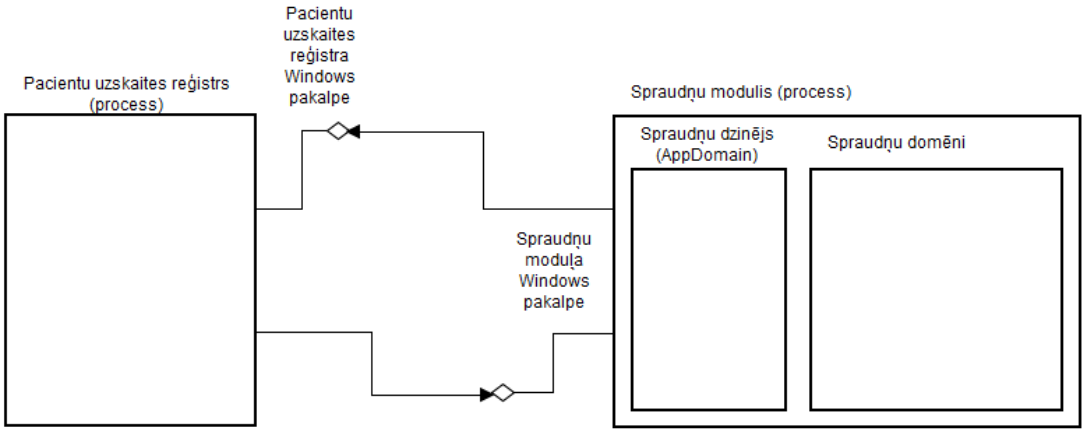
Programmatūra sastāv no šādiem moduļiem:

2.2.1.1 Spraudņu dzinējs

Modulis paredzēts spraudņu ielādei un izpildei. Tas tiks realizēts kā Windows pakalpe, lai nodrošinātu tā izpildi atsevišķā procesā, tādā veidā netraucējot pārējā pacientu uzskaites reģistra darbību. Funkcionalitāte:

- Spraudņu ieslēgšana
- Spraudņu izslēgšana
- Spraudņu ielādēšana
- Spraudņu izpildīšana
- Spraudņu izlādēšana
- Saņemt no publicēšanas spraudņa vērtības
- Izsaukt pakalpi vērtību publicēšanai

2.2.1.1.1 Procesu raksturojums



2.1. att. **Projekta procesu diagramma**

Spraudņu modulis ir jāizpilda atsevišķa procesā no pārējā pacientu uzskaites reģistra, lai netraucētu tā darbību. Gan pacientu uzskaites reģistram gan spraudņu modulim ir jāizveido Windows pakalpes, lai nodrošinātu to atdalīšanu atsevišķos procesos un to savstarpējo komunikāciju. (2)

Jebkāda klasifikatoru apstrāde notiek pacientu uzskaites reģistrā, kas šajā darbā netiek aprakstīts. No šī procesa tiek veikti visi izsaukumi uz spraudņu moduli caur Windows pakalpēm.

Spraudņu moduļa process sākotnēji sastāv no galvenā lietotnes domēna, kurā tiek izpildīta datu piekļuve un spraudņu ielāde. Katrs spraudnis tiek izpildīts atsevišķā domēnā, lai netraucētu viens otra darbību.

Tā, kā klasifikatoru vērtību publicēšanas pilnais cikls ir sarežģīts un laikietilpīgs, pacientu uzskaites reģistrā tas tiek izpildīts asinhroni. Tas nozīmē, ka publicēšanas spraudnim pēc vērtību nosūtīšanas darbs ir jāpārtrauc, līdz tiek saņemta atbilde no pacientu uzskaites reģistra. Lai to realizētu ir jānodrošina abpusēja komunikācija starp spraudni un dzinēju. Tas tiks panākts ar saskarņu palīdzību.

Publicēšanas spraudnis vienlaicīgi var pārraudzīt vairākus klasifikatorus. Tādēļ jānodrošina iespēja spraudnim izšķirt, kuram klasifikatoram atbilst no pacientu uzskaites reģistra saņemtais atbildes ziņojums. Lai to realizētu, nepieciešams spraudņa dzinēja domēnā uzturēt sarakstu ar ziņojumiem un tiem atbilstošajiem spraudņu un klasifikatoru identifikatoriem.

2.2.1.2 Datu piekļuve

Modulis paredzēts datu iegūšanai no datubāzes. Funkcionalitāte:

- Iegūt spraudņa informāciju
- Iegūt spraudņa konfigurāciju
- Iegūt izplatāmās vērtības
- Iegūt izplatāmās vērtību izmaiņas
- Iegūt spraudņu sarakstu
- Iegūt klasifikatoru informāciju
- Nomainīt spraudņa statusu (ieslēgts/izslēgts)

2.2.1.3 Publicēšanas spraudnis

Modulis paredzēts klasifikatora vērtību datņu lasīšanai un datu nosūtīšanai pacientu uzskaites reģistram. Funkcionalitāte:

- No failu sistēmas lasīt klasifikatora vērtību datnes
- Nolasītās vērtības sūtīt uz spraudņu dzinēju
- Dzēst publicēto vērtību datnes

2.2.1.4 Pilno vērtību izplatīšanas spraudnis

Modulis paredzēts klasifikatora pilno vērtību izplatīšanai, saņemot datus no pacientu uzskaites reģistrā realizētām funkcijām. Funkcionalitāte:

- No konfigurācijas nolasīt adresi, kur saglabāt padotās vērtības

- Saglabāt padotās klasifikatoru vērtības datnē

2.2.1.5 Vērtību izmaiņu izplatīšanas spraudnis

Modulis paredzēts klasifikatora vērtību izmaiņu izplatīšanai, saņemot datus no pacientu uzskaites reģistrā realizētām funkcijām. Funkcionalitāte:

- No konfigurācijas nolasīt adresi, kur saglabāt padotās vērtības
- Saglabāt padotās klasifikatoru vērtības datnē

2.2.2 Datu dekompozīcija

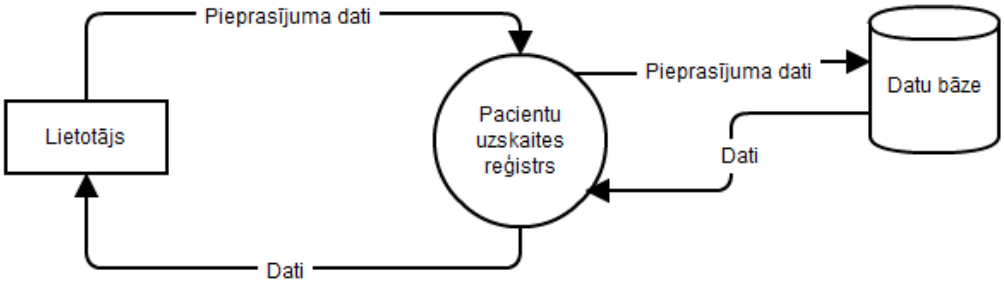
Spraudņu moduļa dati tiek glabāti MSSQL datubāzē, vairākās tabulās:

- Addins – satur datus par reģistrētajiem spraudņiem
- ClassifierAddins – satur saiknes starp klasifikatoriem un spraudņiem

2.3 Atkarību apraksts

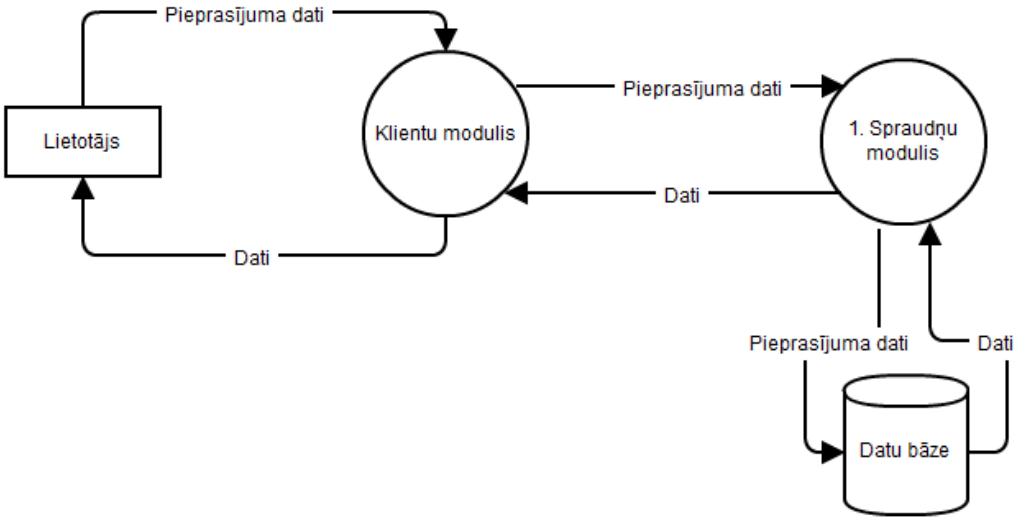
2.3.1 Moduļu atkarības

2.3.1.1 0.līmeņa diagramma



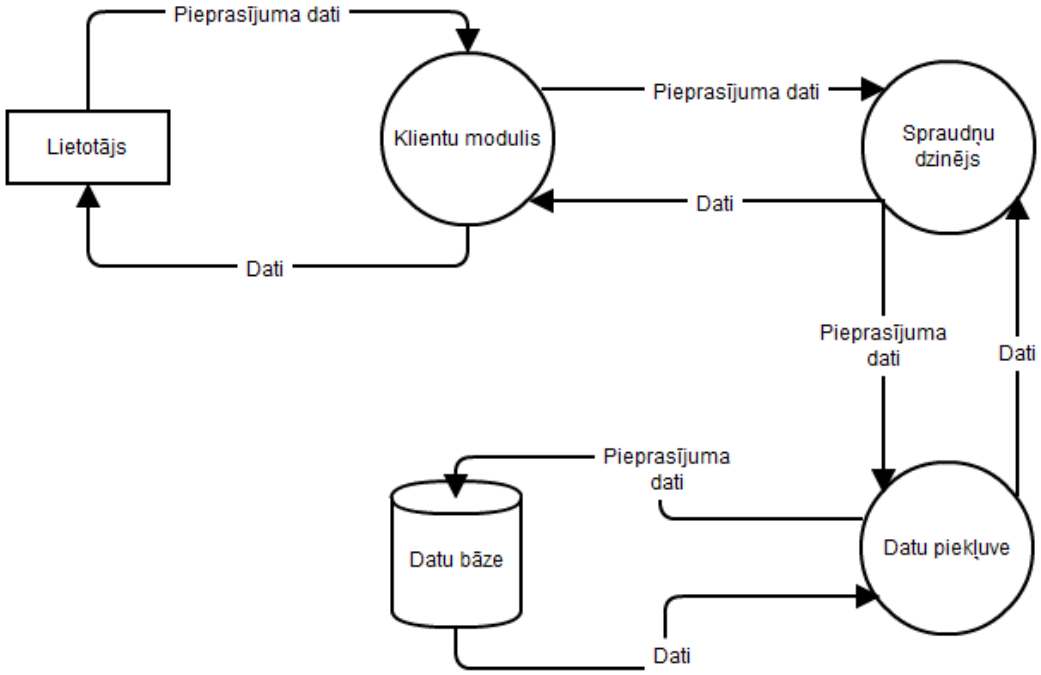
2.2. att. **0. līmeņa datu plūsmas diagramma**

2.3.1.2 1. līmeņa diagramma



2.3. att. **1. līmeņa datu plūsmas diagramma**

2.3.1.3 2. līmeņa diagramma

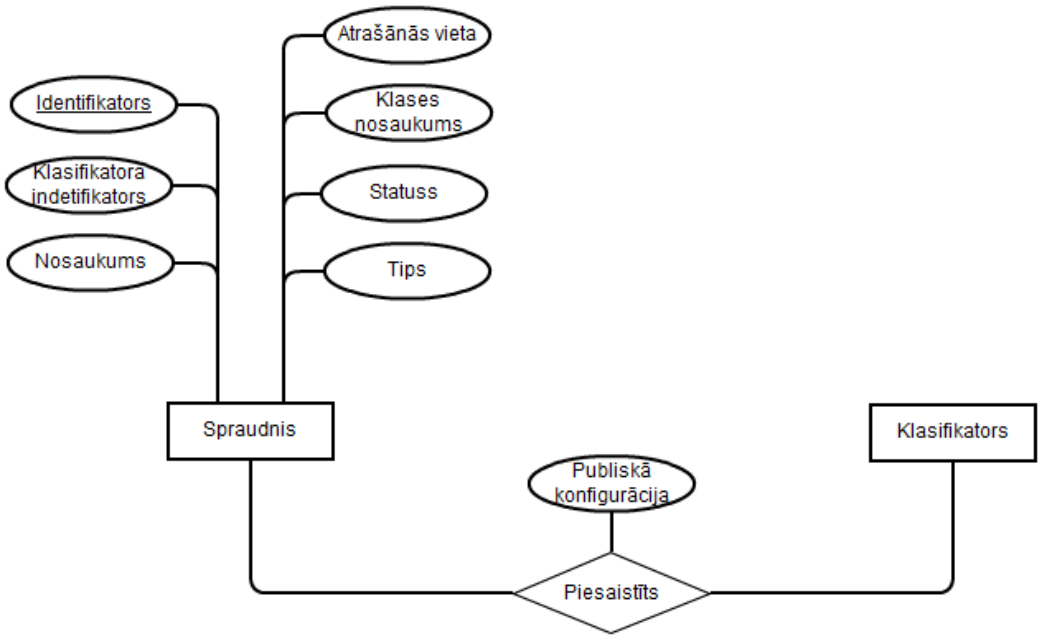


2.4. att. **2. līmeņa datu plūsmas diagramma**

2.3.2 Datu atkarības

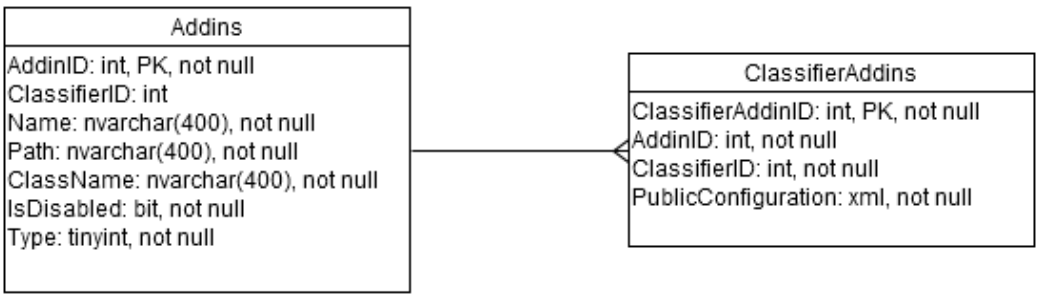
Šajā sadaļā tiek attēlotas tikai tās datu atkarības daļas, kas tieši nepieciešamas spraudņu moduļa realizācijai.

2.3.2.1 ER modelis



2.5. att. **ER modelis**

2.3.2.2 Datubāzes fiziskais modelis



2.6. att. **Datubāzes fiziskais modelis**

2.4 Detalizēts projektējums

2.4.1 Funkciju detalizēts projektējums

2.4.1.1 StartAddin

2.1. tabula

StartAddin funkcija

Atribūts	**Apraksts**
Nosaukums	StartAddin
Apraksts	Funkcija realizē spraudņa ieslēgšanu
Ievaddati	AddinID (int)
Apstrāde	- Tabulā tiek nomainīts spraudņa statuss (IsDisabled) - Tiek izsaukta publicēšanas spraudņu ielādēšanas funkcija
Izvade	Nav

2.4.1.2 StopAddin

2.2. tabula

StopAddin funkcija

Atribūts	**Apraksts**
Nosaukums	StopAddin
Apraksts	Funkcija realizē spraudņa izslēgšanu
Ievaddati	AddinID (int)
Apstrāde	- Tabulā tiek nomainīts spraudņa statuss (IsDisabled) - Tiek izsaukta publicēšanas spraudņu izlādēšanas funkcija
Izvade	Nav

2.4.1.3 GetClassifierAddinList

2.3. tabula

GetClassifierAddinList funkcija

Atribūts	**Apraksts**
Nosaukums	GetClassifierAddinList
Apraksts	Iegūst klasifikatoriem piesaistīto spraudņu sarakstu
Ievaddati	ClassifierID (II)
Apstrāde	No tabulas Addins tiek atlasīti visi spraudņi, kuri ir piesaistīti kādam klasifikatoram un kuri ir ieslegti (IsDisabled = false)
Izvade	AddinID (II) Name(ST) AddinType(LVCR_MT000049UV01.AddinType Notes(ST)

2.4.2 Datu detalizēts projektējums

2.4.2.1 Tabula „Addins”

2.4. tabula

Tabula „Addins”

Apraksts	Tabula spraudņu informācijas glabāšanai		
Nosaukums	**Datu tips**	**Atslēga**	**Apraksts**
AddinID	int	PK	Automātiski palielinās, unikāls spraudņa identifikators
ClassifierID	int		Piesaistītais klasifikators (null)
Name	nvarchar(400)		Spraudņa nosaukums
Path	nvarchar(400)		Spraudņa atrašanās vieta
ClassName	nvarchar(400)		Spraudņa pilnais klases nosaukums
IsDisabled	bit		Pazīme, vai spraudnis ir neaktīvs
Type	tinyint		Spraudņa tips

2.4.2.2 Tabula „ClassifierAddins”

2.5. tabula

Tabula „ClassifierAddins”

Apraksts	Tabula spraudņu un klasifikatoru saikņu glabāšanai		
Nosaukums	**Datu tips**	**Atslēga**	**Apraksts**
ClassifierAddinID	int	PK	Automātiski palielinās, unikāls ieraksta identifikators
ClassifierID	int		Klasifikatora identifikators
AddinID	int		Spraudņa identifikators
PublicConfiguration	xml		Spraudņa publiskā konfigurācija

3. TESTĒŠANAS DOKUMENTĀCIJA

3.1 Ievads

Darba beigās programmatūra vēl netika izstrādāta līdz galam, tāpēc visi testi tika veikti izstrādes laikā. Lietotājam tieša saskare ar spraudņu moduli nav paredzēta, visi izsaukumi notiek caur citiem projektiem, kuros tiek veiktas ievaddatu pārbaudes, tāpēc izstrādes gaitā liela uzmanība tika pievērsta tieši tam, kā tiek ielādēti spraudņi.

3.2 Testēšanas plāns

Lai izstrādes gaitā būtu iespējams testēt izstrādāto kodu, datubāzē tika ievadīti vienkārši testa dati, kā arī izstrādāta testa vide, no kuras atsevišķi bija iespējams pārbaudīt katru funkciju un moduli.

Tika pārbaudīts:

- vai SQL procedūru izsaukumi atgriež atbilstošos datus
- vai spraudņa dzinēja kļūdas gadījumā netiek pārtraukta pārējā koda izpilde

3.3 Testēšanas žurnāls

3.1. tabula
Testēšanas žurnāls

Testa Nr.	Apraksts	Rezultāts	Piezīmes
1.	Atgriezt spraudņu informāciju pēc spraudņa identifikatora	Izpildās	
2.	Atgriezt spraudņu informāciju pēc klasifikatora kodu sistēmas	Izpildās	
3.	Pilno vērtību izplatīšanas spraudņa darbība	Izpildās	
4.	Vērtību izmaiņu izplatīšanas spraudņa darbība	Izpildās	
5.	Publicēšanas spraudņa ieslēgšana	Izpildās	
6.	Publicēšanas spraudņa izslēgšana	Izpildās datubāzē	Spraudņa izslēgšana nav pārbaudīta
7.	Publicējamo vērtību nolasīšana	Izpildās	
8.	Publicējamo vērtību nosūtīšana pacientu uzskaites reģistram	Izpildās	
9.	Publicējamo vērtību datnes pagaidu pārsaukšana publicēšanas laikā	Izpildās	
10.	Publicējamo vērtību datnes dzēšana pēc veiksmīgas publicēšanas	Neizpildās	Funkcionalitāte nav realizēta
11.	Ziņojuma identifikatora saņemšana no pacientu uzskaites reģistra	Izpildās	
12.	Ziņojuma identifikatora	Neizpildās	Nav atrisināta

	nodošana atbilstošajam spraudnim		problēma ar koplietošanas sarakstu
13.	Lietotnes domēna uzstādīšana	Izpildās	
14.	Lietotnes domēna izveidošana	Izpildās	
15.	Spraudņa ielāde domēnā	Izpildās	
16.	Domēna izlāde	Izpildās	

4. PROJEKTA ORGANIZĀCIJA

Projekts tika veidots pēc ūdenskrituma metodes, kurā projekta soļi tiek izpildīti secīgi. Vispirms tika apkopotas visas projekta prasības un izveidota programmatūras prasību specifikācija. Balstoties uz PPS tika izstrādāts projektējums un izveidots programmatūras projektējuma apraksts, pēc kura tika izstrādāta sistēma.

Spraudņu moduli izstrādāja viens cilvēks, taču tam ir cieša saistība ar pacientu uzskaites reģistru, tāpēc visas izstrādes laikā notika komunikācija ar citiem izstrādātājiem.

Lai arī sistēmas prasību nav daudz, tā ir plaša, tāpēc šī darba ietvaros tā nav izstrādāta līdz galam. Līdz ar to testēšanas dokumentācijā tika aprakstīta tikai izstrādes laika testēšana, pārbaudot atsevišķu funkciju darbību.

5. KVALITĀTES NODROŠINĀŠANA

Lai nodrošinātu produkta kvalitāti, programmatūra tika izstrādāta balstoties uz labo praksi, uzņēmuma vadlīnijām un Latvijas valsts standartiem.

Izstrādes gaitā kods tika komentēts, pārskatāmi strukturēts, mainīgo, funkciju un klašu nosaukumi tika veidoti tā, lai to nosaukums virspusēji izskaidrotu to nozīmi. Tā rezultātā tika samazināts pārpratumu skaits izstrādē.

Koda fragmenti, kuri tiek izmantoti vairākkārt, tika izdalīti atsevišķās klasēs un funkcijās, lai nepiesārņotu kodu.

Izstrādes laikā tika veikta testēšana, un atrastās kļūdas un nepilnības uzreiz tika labotas.

Tika izmantota TortoiseSVN konfigurācijas pārvaldības sistēma, kas ļoti atviegloja kvalitatīva koda izstrādi.

6. KONFIGURĀCIJU PĀRVALDĪBA

Spraudņu modulis ir daļa no lielāka projekta, kura izstrādē piedalās vairāki cilvēki, līdz ar to konfigurāciju pārvaldības sistēma ir nepieciešama. Šī projekta izstrādei tiek izmantota TortoiseSVN sistēma.

Tortoise SVN nodrošina vienkāršu, ērtu un ātru versiju pārvaldību. Jebkurā brīdī ir iespējams iegūt iepriekšējās projekta versijas.

7. DARBIETILPĪBAS NOVĒRTĒJUMS

Projekta darbietilpības noteikšanai tika izmantota Basic COCOMO metode. Lai pēc šīs metodes veiktu darbietilpības novērtējumu, ir jāzin aptuvenais programkoda rindiņu skaits.

Šī darba ietvaros viena persona sarakstīja aptuveni 2000 programkoda rindiņas, neskaitot SQL procedūras, lietotņu konfigurācijas un testa vidi.

Izmantojot Basic COCOMO vienādojumus

Darbietilpība = 2.4 * KLOC ^ 1.05 = 2.4 * 2 ^ 1.05 = 4.9

Izstrādes laiks = 2.5 * Darbietilpība ^ 0.38 = 2.5 * 4.9 ^ 0.38 = 4.5

Cilvēku skaits = Darbietilpība / Izstrādes laiks = 4.9 / 4.5 = ~1

Aprēķinātā aptuvenā projekta darbietilpība vienai personai ir 4,9 personmēneši.

Aprēķinātā darbietilpība ir neatbilstoša patiesībai, jo pie projekta tika intensīvi strādāts nepilnus 3 mēnešus.

Šī darbietilpības neatbilstība radusies, jo aprēķinot darbietilpību netiek ņemti vērā vairāki faktori, piemēram, izstrādātāja prasmes un pieredze, izmantotie rīki, metodes.

8. REZULTĀTI

Darba beigās ir tikusi izstrādāta lielākā daļa no spraudņu moduļa.

Lai arī programmatūras prasību ir maz un projektējums nav plašs, spraudņu modeļa realizācija ir sarežģīta. Darba rezultātā ir iegūta plaša pieredze darbā ar procesiem, pavedieniem, lietotnes domēniem un Windows pakalpēm.

Papildus PPS un PPA noteiktajai funkcionalitātei tika izstrādāti papildus moduļi izstrādes atvieglošanai, piemēram žurnalēšanas klase.

Projekta izstrāde notika veiksmīgi pateicoties pieredzei darbā ar izstrādes vidi Microsoft Visual Studio 2012, kā arī šīs izstrādes vides ērtajai lietošanai.

9. SECINĀJUMI

Liela nozīme ir labi strukturētam kodam, galvenokārt viegli saprotami veidotiem mainīgo, klašu, funkciju un vārdtelpu nosaukumiem.

Komentāri ir noderīgi, taču nedrīkst pārspīlēt ar koda komentēšanu, jo pārāk daudz komentāru kodā traucē paša koda lasīšanu un izpratšanu.

Konfigurāciju pārvaldība lielos projektos ir obligāta, bez tās izstrādātāju starpā radīsies daudz konfliktu, kā rezultātā darba ražīgums būs zems.

10. AVOTI

10.1 Elektroniskie informācijas avoti

1. *Microsoft Development Network* [tiešsaiste]. Microsoft, 2012. Pieejams: http://www.msdn.com
2. *Stack Overflow* [tiešsaiste]. Stack Exchange Inc., 2012. Pieejams: http://www.stackoverflow.com
3. *Test Level 7* [tiešsaiste]. Intelliware Development Inc., 2012. Pieejams: http://tl7.intelliware.ca/public/messages/dataTypes/index.faces

11. PIELIKUMI

11.1 1. pielikums. Programmatūras izejas kods

11.1.1 AddinsEngine klase

```
/// <summary>
/// Spraudņu dzinējs
/// </summary>
[Serializable]
public class AddinsEngine : MarshalByRefObject, IPublishEngine
{
  /// <summary>
  /// Izsauc PublishValues pakalpi klasifikatora vērtību publicēšanai
  /// </summary>
  /// <param name="values">Klasifikatora vērtības</param>
  /// <param name="addinID">Izsaucošā spraudņa identifikators</param>
  /// <param name="codeSystem">Klasifikatora kodu sistēma</param>
  public void PublishValues(String values, int addinID, String codeSystem)
  {
        AddinLogger.Instance.LogInformation("Engine PublishValues called.");

        // Izveido objektu, ko sūtīt PublishValues pakalpei
        LVCR_IN000001UV01MCAI_MT700201UV01_LV01Subject2 subject =
              new LVCR_IN000001UV01MCAI_MT700201UV01_LV01Subject2();

        // Aizpilda objektu ar padotajām vērtībām
        subject.Classifier = (LVCR_MT000001UV01ClassifierStructure)
              System.Xml.Linq.XElement.Parse(values);
        AddinLogger.Instance.LogInformation("Values parsed.");

        // Izveido PublishValues pakalpes klienta objektu
        PublishValuesClient client = new PublishValuesClient();
        AddinLogger.Instance.LogInformation("PublishValues client created.");

        // Izsauc PublishValues pakalpi
        client.PublishValues(subject, addinID, codeSystem);
  }

  /// <summary>
  /// PublishValues pakalpes asinhronās atbildes saņemšanas funkcija
  /// </summary>
  /// <param name="messageID">Ziņojuma identifikators</param>
  /// <param name="message">Ziņojums</param>
  public void SendReply(Guid messageID, String message)
  {
        // Iegūst ziņojumu un domēnu sarakstus
        List<MessageIDListEntry> messages = AddinLists.GetMessages();
        List<DomainListEntry> domains = AddinLists.GetDomains();

        int AddinID = new int();
        String CodeSystem = null;
        IPublishAddin Instance = null;

        // Iegūst ziņojumam atbilstošā spraudņa identifikatoru un klasifikatora kodu sistēmu
        foreach (var item in messages)
        {
              if (item.MessageID == messageID)
              {
                    AddinID = item.AddinID;
                    CodeSystem = item.CodeSystem;
                    break;
```

```
                }
        }

        // Iegūst spraudņa eksemplāru, kurā ielādēts attiecīgais spraudnis
        foreach (var item in domains)
        {
                if (item.AddinID == AddinID)
                {
                        Instance = item.AddinInstance;
                        break;
                }
        }

        // Atrastajam spraudņa eksemplāram izsauc funkciju PublishComplete
        if (Instance != null)
        {
                Instance.PublishComplete(message, CodeSystem);
                MessageIDList.Instance.Remove(messageID);
        }
}

/// <summary>
/// Aptur noradītā spraudņa darbību
/// </summary>
/// <param name="appDomain">Spraudņa identifikators</param>
public void StopAddin(int addinID)
{
        AddinLogger.Instance.LogInformation("StopAddin called.");

        // Izsauc funckiju, kas datubāzē nomaina spraudņa aktivitātes pazīmi
        AddinsDataAccess.SetIsDisabled(
                addinID,
                true);
        AddinLogger.Instance.LogInformation("SetIsDisabled (true) call finished.");

        // Izsauc funkciju, kas aptur spraudņa darbību
        StopPublishAddins(addinID);
}

/// <summary>
/// Ieslēdz spraudni
/// </summary>
/// <param name="addinID">Spraudņa identifikators</param>
public void StartAddin(int addinID)
{
        AddinLogger.Instance.LogInformation("StartAddin called.");

        // Izsauc funckiju, kas datubāzē nomaina spraudņa aktivitātes pazīmi
        AddinsDataAccess.SetIsDisabled(
                addinID,
                false);
        AddinLogger.Instance.LogInformation("SetIsDisabled (false) call finished.");

        // Izsauc funkciju, kas ieslēdz publicēšanas spraudni
        CallPublishAddins(addinID);
}

/// <summary>
/// Sagatavo visus nepieciešamos datus izplatīšanas spraudņu izpildīšanai
/// </summary>
/// <param name="codeSystem">Klasifikatora kodu sistēma</param>
public void CallDistributeAddins(Object codeSystem)
```

```
  {
        try
        {
            // Atgriež sarakstu ar spraudņu informāciju (ClassName, Type, Path) padotajam
            // klasifikatoram, kur spraudņu tips ir "Distribute"
            List<AddinInformation> AddinInformation =
AddinsDataAccess.GetAddinInformation(
                (int)AddinType.Distribute,
                codeSystem: codeSystem.ToString()
            );

            AddinLogger.Instance.LogInformation("GetAddinInformation call finished.");
            if (AddinInformation.Count == 0)
            {
                AddinLogger.Instance.LogInformation("No active addins were found for
given classifier.");
                return;
            }

            // Iegūst klasifikatora vērtības izplatīšanai
            ClassifierValues DistributeValues =
AddinsDataAccess.GetClassifierValues(codeSystem.ToString());

            AddinLogger.Instance.LogInformation("GetClassifierValues call finished.");
            if (DistributeValues == null) throw new Exception("No classifier values were
retrieved.");

            // Iet cauri spraudņu sarakstam, katru no tiem izpildot
            foreach (var currentAddin in AddinInformation)
            {
                ExecuteDistributeAddin(
                    currentAddin,
                    DistributeValues
                );
                AddinLogger.Instance.LogInformation(String.Format(
                    "{0} addin execution finished.",
                    currentAddin.Type));
            }
        }
        catch (Exception ex)
        {
            AddinLogger.Instance.LogException(ex);
        }
  }

  /// <summary>
  /// Sagatavo nepieciešamos datus publicēšanas spraudņu izpildīšanai
  /// </summary>
  /// <param name="addinID">Addin to be executed</param>
  public void CallPublishAddins(int addinID)
  {
        try
        {
            // Atgriež sarakstu ar spraudņu informāciju (ClassName, Type, Path) padotajam
            // klasifikatoram, kur spraudņu tips ir "Publish"
            List<AddinInformation> AddinInformation =
AddinsDataAccess.GetAddinInformation(
                (int)AddinType.Publish,
                addinID: (int?)addinID
            );

            AddinLogger.Instance.LogInformation("GetAddinInformation call finished.");
```

```
                if (AddinInformation.Count == 0)
                {
                        AddinLogger.Instance.LogInformation("Given addin is not of type
\"Publish\" or it is not assigned to any classifier.");
                        return;
                }

                // Izsauc funkciju, kas ieslēdz publicēšanas spraudni
                ExecutePublishAddin(AddinInformation);

                AddinLogger.Instance.LogInformation(String.Format(
                        "{0} addin started.",
                        AddinInformation[0].Type));
        }
        catch (Exception ex)
        {
                AddinLogger.Instance.LogException(ex);
        }
  }

  /// <summary>
  /// Izslēdz publicēšanas spraudni
  /// </summary>
  /// <param name="addinID">Spraudņa identifikators</param>
  public void StopPublishAddins(int addinID)
  {
        try
        {
                // Iegūst domēnu sarakstu
                List<DomainListEntry> domains = AddinLists.GetDomains();

                // Iet cauri domēnu sarakstam, izslēdzot padoto spraudni
                foreach (var domain in domains)
                {
                        if (domain.AddinID == addinID)
                        {
                                AddinLists.Remove(addinID, domain.Domain, domain.Instance);
                                AppDomain.Unload(domain.Domain);
                        }
                }
        }
        catch (Exception ex)
        {
                AddinLogger.Instance.LogException(ex);
        }
  }

  /// <summary>
  /// Ielādē un izpilda izplatīšanas spraudni
  /// </summary>
  /// <param name="currentAddin">Spraudņa informācija</param>
  /// <param name="DistributeValues">Vērtības izplatīšanai</param>
  private void ExecuteDistributeAddin(AddinInformation currentAddin, ClassifierValues
DistributeValues)
  {
        AppDomain AddinDomain = null;

        // Ielādē spraudni atsevišķā aplikāciju domēnā
        try
        {
                AddinLogger.Instance.LogInformation("ExecuteAddin (distribute) called.");
```

```
                    // Uzstāda aplikāciju domēnu
                    AppDomainSetup addinDomainSetup = new AppDomainSetup();
                    addinDomainSetup.ApplicationBase =

  Path.GetDirectoryName(System.Reflection.Assembly.GetEntryAssembly().Location);
                    addinDomainSetup.ApplicationName = "DistributeAddin";
                    addinDomainSetup.ConfigurationFile = currentAddin.Path + @".config";

                    // Izveido domēnu
                    AddinDomain =
                            AppDomain.CreateDomain("DistributeDomain", null, addinDomainSetup);

                    AddinLogger.Instance.LogInformation("Domain \"DistributeDomain\" created.");

                    // Izveido spraudņa eksemplāru
                    var AddinInstance =
                            AddinDomain.CreateInstanceFromAndUnwrap(
                                    currentAddin.Path,
                                    currentAddin.ClassName
                            );

                    AddinLogger.Instance.LogInformation(String.Format(
                            "Addin instance from file '{0}' with type '{1}' created.",
                            currentAddin.Path,
                            currentAddin.ClassName));

                    Boolean IsDistributeIncAddin;
                    Boolean IsDistributeFullAddin;
                    IDistributeFullAddin AddinFullInstance = null;
                    IDistributeIncAddin AddinIncInstance = null;

                    // Pārbaude, kuru saskarni spraudnis izmanto (IDistributeFullAddin vai
IDistributeIncAddin
                    // Uzstāda pazīmes
                    #region Interfaces Check
                    try
                    {
                            AddinIncInstance = (IDistributeIncAddin)AddinInstance;
                            IsDistributeIncAddin = true;
                            AddinLogger.Instance.LogInformation("Addin instance successfully cast to
IDistributeIncAddin");
                    }
                    catch (InvalidCastException)
                    {
                            IsDistributeIncAddin = false;
                    }

                    try
                    {
                            AddinFullInstance = (IDistributeFullAddin)AddinInstance;
                            IsDistributeFullAddin = true;
                            AddinLogger.Instance.LogInformation("Addin instance successfully cast to
IDistributeFullAddin");
                    }
                    catch (InvalidCastException)
                    {
                            IsDistributeFullAddin = false;
                    }
                    #endregion

                    // Pārbauda pazīmju kombinācijas
                    // Ja spraudnis izmanto abas saskarnes - kļūda
```

```
                    if ((IsDistributeFullAddin == true) &&
                            (IsDistributeIncAddin == true))
                    {
                            throw new Exception("Addin cannot implement both Inc and Full
interfaces.");
                    }
                    else
                    // Ja spraudnis neizmanto nevienu no saskarnēm - kļūda
                    if ((IsDistributeFullAddin == false) &&
                            (IsDistributeIncAddin == false))
                    {
                            throw new Exception("Addin doesn't implement the neccessary interfaces.");
                    }
                    else
                    // Ja spraudnis izmanto Inc saskarni, atbilstošajam eksemplāram izsauc Distribute
funkciju
                    if (IsDistributeIncAddin == true)
                    {
                            AddinIncInstance.Distribute(
                                    currentAddin.Configuration,
                                    DistributeValues.IncValues.Classifier.Untyped.ToString()
                            );

                            AddinLogger.Instance.LogInformation("'Distribute' for incremental values
call finished");
                    }
                    else
                    // Ja spraudnis izmanto Full saskarni, atbilstošajam eksemplāram izsauc Distribute
funkciju
                    if (IsDistributeFullAddin == true)
                    {
                            AddinFullInstance.Distribute(
                                    currentAddin.Configuration,
                                    DistributeValues.FullValues.Classifier.Untyped.ToString()
                            );

                            AddinLogger.Instance.LogInformation("'Distribute' for full values call
finished");
                    }
            }
            catch (Exception ex)
            {
                    AddinLogger.Instance.LogException(ex);
            }
            finally
            {
                    // Pēc izpildes izlādē aplikāciju domēnu
                    if (AddinDomain != null)
                    {
                            String domainName = AddinDomain.FriendlyName;
                            AppDomain.Unload(AddinDomain);

                            AddinLogger.Instance.LogInformation(String.Format(
                                    "AppDomain '{0}' unloaded.",
                                    domainName));
                    }
                    else
                    {
                            AddinLogger.Instance.LogInformation("No AppDomain was loaded.");
                    }
            }
    }
```

```
  /// <summary>
  /// Ielādē un izpilda publicēšanas spraudni
  /// </summary>
  /// <param name="currentAddin">Spraudņa informācija</param>
  private void ExecutePublishAddin(List<AddinInformation> addinInformation)
  {
            AppDomain AddinDomain = null;

            // Ielādē spraudni atsevišķā aplikāciju domēnā
            try
            {
                        AddinLogger.Instance.LogInformation("ExecuteAddin (publish) called.");

                        // Uzstāda aplikāciju domēnu
                        AppDomainSetup addinDomainSetup = new AppDomainSetup();
                        addinDomainSetup.ApplicationBase =

   Path.GetDirectoryName(System.Reflection.Assembly.GetEntryAssembly().Location);
                        addinDomainSetup.ApplicationName = String.Format("({0})PublishAddin",
addinInformation[0].AddinID);
                        addinDomainSetup.ConfigurationFile = addinInformation[0].Path + @".config";

                        // Izveido aplikāciju domēnu
                        AddinDomain =
                                    AppDomain.CreateDomain(
                                                String.Format("({0})PublishDomain",
addinInformation[0].AddinID),
                                                null,
                                                addinDomainSetup
                                    );

                        AddinLogger.Instance.LogInformation("Domain" +
String.Format("({0})PublishDomain", addinInformation[0].AddinID) + " created.");

                        // Izveido spraudņa eksemplāru domēnā
                        var AddinInstance = (IPublishAddin)
                                    AddinDomain.CreateInstanceFromAndUnwrap(
                                                addinInformation[0].Path,
                                                addinInformation[0].ClassName
                                    );

                        AddinLogger.Instance.LogInformation(String.Format(
                                    "Addin instance from file '{0}' with type '{1}' created.",
                                    addinInformation[0].Path,
                                    addinInformation[0].ClassName));

                        // Spraudņa dzinēja eksemplārs, ko padot spraudnim
                        IPublishEngine Instance = (IPublishEngine)
                                    AppDomain.CurrentDomain.CreateInstanceAndUnwrap(
                                    Assembly.GetExecutingAssembly().FullName,
                                    "ClassifierRegister.Addins.AddinsEngine"
                        );

                        // Ieslēdz spraudni
                        AddinInstance.Start(
                                    addinInformation,
                                    Instance
                        );
                        AddinLogger.Instance.LogInformation(String.Format("({0}) Publish addin started",
addinInformation[0].AddinID));
```

```
                    // Pievieno ierakstu domēnu sarakstam
                    AddinLists.Add(
                        addinInformation[0].AddinID,
                        AddinDomain,
                        Instance
                    );
                }
                catch (Exception ex)
                {
                    AddinLogger.Instance.LogException(ex);
                }
            }
        }
```

12. DOKUMENTĀRĀ LAPA

Kvalifikācijas darbs „***Pacientu uzskaites sistēma, izmantojot HL7 standartu***” izstrādāts Latvijas Universitātes Datorikas fakultātē.

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

Autors: ***Andris Lācis*** _____ **___.05.2012**.

Rekomendēju darbu aizstāvēšanai
Darba vadītājs: ***Dr. dat., Māris Vītiņš***_____ **___.05.2012.**

Recenzents: ***M. dat., Agris Šnepsts***

Darbs iesniegts 24.05.2012.
Kvalifikācijas darbu pārbaudījumu komisijas sekretārs: *Dainis Dosbergs* _____

Darbs aizstāvēts kvalifikācijas darbu pārbaudījuma komisijas sēdē
___.06.2012. prot. Nr. _____, vērtējums _____ (_____)

Komisijas sekretārs(-e): _____