KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Justinas Prelgauskas

Vizitų registravimo sistemos projektavimas ir testavimas

Magistro darbas

Darbo vadovas
prof. E. Bareiša

Kaunas, 2008

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Justinas Prelgauskas

Vizitų registravimo sistemos projektavimas ir testavimas

Magistro darbas

Recenzentas: doc. V. Pilkauskas 2008-05-19	Vadovas: prof. E. Bareiša 2008-05-19 Atliko: IFM-2/2 gr. stud. Justinas Prelgauskas 2008-05-19

Kaunas, 2008

Turinys

DESIGN AND TESTING OF CALL REPORTING SYSTEM

SUMMARY

This work consists of three major parts. First – engineering part – is analysis and design of call reporting system (codename – "PharmaCODE"). We will provide main details of business analysis and design decisions.

Second part is all about testing and ensuring system quality, mainly by means of static source code analysis tools & methods. We will describe tools being used and provide main results of source code analysis in this part.

And finally, in the third part of this we go deeper into static source code analysis and try to improve one of analysis rules. These days, when there is plenty of evolving web-based applications, security is gaining more and more impact. Most of those systems have, and depend on, back-end databases. However, web-based applications are vulnerable to SQL-injection attacks. In this paper we present technique of solving this problem using secure-coding guidelines and .NET Framework's static code analysis methods for enforcing those guidelines. This approach lets developers discover vulnerabilities in their code early in development process. We provide a research and realization of improved code analysis rule, which can automatically discover SQL-injection vulnerabilities in MSIL code.

1. ĮVADAS

1.1. Dokumento paskirtis

Šis dokumentas yra programų sistemų inžinerijos magistro baigiamasis darbas. Dokumente aprašėme farmacijos rinką ir jos dalyvių tarpusavio sąveiką: projektuojamos sistemos, užsakovo ir klientų atstovybių vietą joje, jų sąveiką. Taip pat aprašėme „Vizitų registravimo sistemą“, pagrindinės jos savybės, architektūrinį vaizdą. Tam tikslui buvo naudojami keletas architektūrinių pjūvių, vaizduojančių tam tikrus sistemos aspektus (statinius, dinaminius ir pan.).

Dokumente užfiksavome ir perteikėme svarbius architektūrinius sprendimus, kurie buvo priimti projektuojant sistemą. Nurodėme kokie komponentai sudaro sistemą, kaip programinė įranga sąveikauja jau su egzistuojančiomis sistemomis, aprašėme sistemos fizinę/operacinę aplinką.

Dokumente detaliau aprašėme atliktus sukurtos sistemos kokybės tyrimus, naudojant statinę išeities kodų analizę. Pasiūlėme patobulintą vienos iš taisyklių realizaciją ir tyrimą. Mūsų patobulinta taisyklė analizuoja išeities tekstus MSIL (angl. Microsoft Intermediate Language) pavidale ir ieško galimų saugumo spragų, susijusių su SQL-įterpinių (angl. SQL - injection) problema.

1.2. Santrauka

Šiame dokumente aprašytas darbas susideda ir trijų pagrindinių dalių. Pirmojoje, inžinerinėje dalyje atlikome vizitų registravimo sistemos (toliau - „PharmaCODE“) analizę ir projektavimą. Čia pateikėme esmines verslo aplinkos, reikalavimų ir konkurentų analizės, o taipogi ir projektavimo detales. Pateikėme pagrindinius architektūrinius sprendimus.

Antrojoje darbo dalyje aprašėme sistemos kokybės tyrimus, naudojant statinės išeities kodų analizės įrankius ir metodus. Šioje dalyje aprašėme kokius įrankius naudojome ir pateikėme pagrindinius kodo analizės rezultatus.

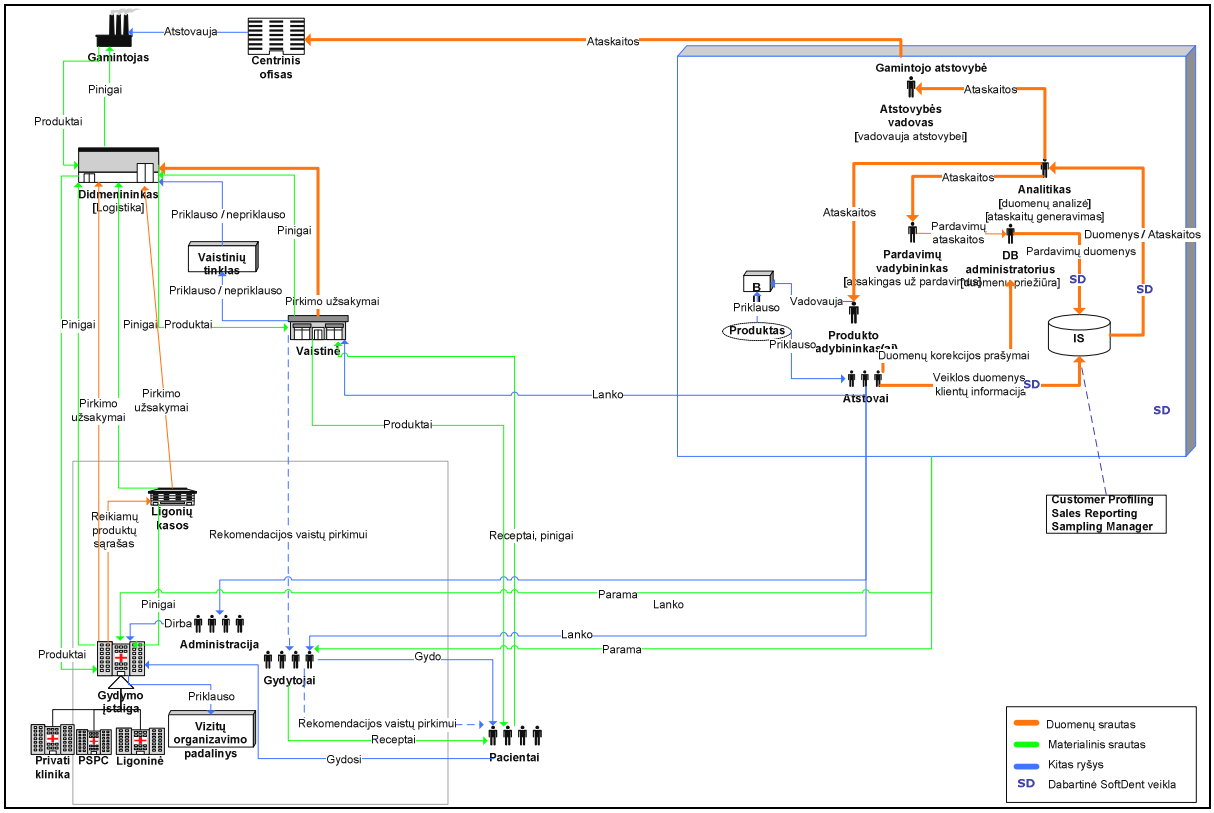
Trečiojoje darbo dalyje gilinomės į išeities tekstų analizės metodus ir įrankius, sukūrėme patobulintą analizės taisyklę. Mūsų taisyklės pagalba pavyko aptikti daugiau potencialių SQL-įterpinių saugumo spragų nei aptiko jos pirmtakė – Microsoft projektuota kodo analizės taisyklė.

2. ANALITINĖ DALIS

2.1. Vizitų registravimo sistemos analitinė dalis

Šioje dalyje analizuojami projektuojamos sistemos būsimų problemų galimi sprendimo būdai. Pristatomas projektas ir jo kontekstas. Pateikiama galimų inžinerinių sprendimo būdų apžvalga.

2.1.1. Verslo aplinkos analizė

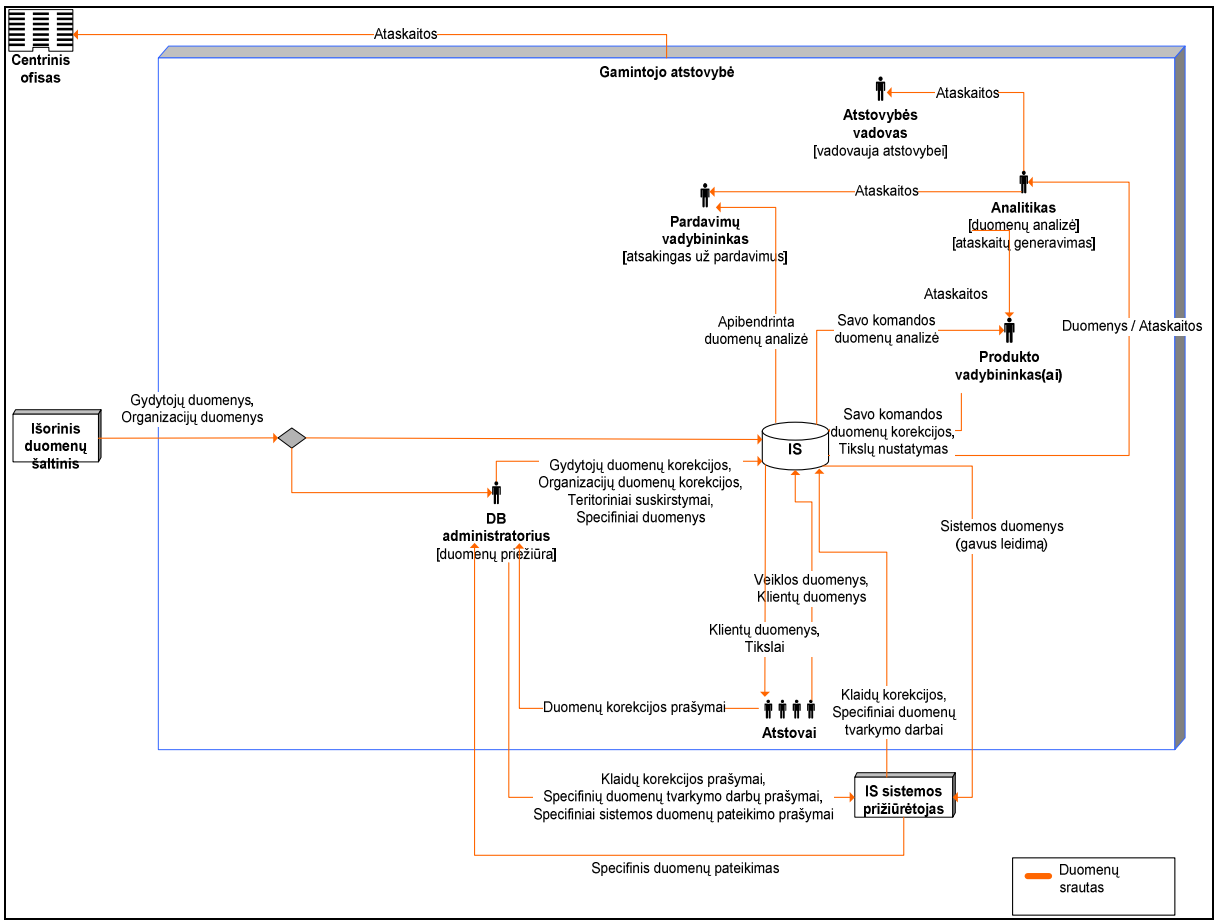


Pav. 1 - gamintojų atstovybių veiklos konteksto diagrama

Aukščiau parodytoje diagramoje atvaizduota visa gamintojo atstovybės veikla ir su ja susijusios kitos organizacijos. Veikla išanalizuota keletu aspektų. Pirmiausia (ir svarbiausia), nurodomi informaciniai/duomenų srautai. Toliau, žaliomis rodyklėmis įvardinti materialieji srautai, apimantys piniginius ir/arba prekių srautus. Kiti (ne informaciniai ir ne materialūs srautai) pažymėti mėlynomis rodyklėmis. Mėlynomis punktyrinėmis linijomis parodomi nestiprios įtakos ryšiai (rekomendacijos). Gamintojų atstovybių veikla buvo išnagrinėta, įtraukiant ir užsakovo (UAB „Softdent“) teikiamas paslaugas. Jos diagramoje pažymėtos „SD“ raidėmis.

Veiklos padalinimas

Galiausiai, išanalizavę visą verslo aplinką, atsirenkame tuos srautus, kurie tiesiogiai įtakoja projektuojamą sistemą – informacijos srautus.



Pav. 2 - gamintojų atstovybių informaciniai srautai

Lentelė nr. 1 – Veiklos įvykių sąrašas

Eil. Nr.	Įvykio pavadinimas	Įeinantys/išeinantys informacijos srautai
1.	Atstovybė teikia ataskaitas centriniam ofisui	(OUT) Ataskaitos centriniam ofisui
2.	Išorinis duomenų šaltinis pateikia duomenis	(IN) Gydytojų, organizacijų duomenys
3.	Analitikas generuoja ataskaitas	(OUT) Ataskaitos vadovui (OUT) Ataskaitos produkto vadybininkui (OUT) Ataskaitos pardavimų vadybininkui
4.	Pardavimų vadybininkas generuoja ataskaitas	(OUT) Apibendrintos ataskaitos pardavimų vadybininkui
5.	Produkto vadybininkas nustato tikslus savo komandai	(IN) Nustatytas komandos tikslas
6.	Produkto vadybininkas administruoja duomenis apie savo komandą	(IN) Informacija apie komandą
7.	Produkto vadybininkas analizuoja savo komandos duomenis, veiklos rezultatus	(OUT) Ataskaitos produkto vadybininkui
8.	Duomenų bazės administratorius koreguoja gydytojų informaciją	(IN) Gydytojų informacija
9.	Duomenų bazės administratorius koreguoja organizacijų informaciją	(IN) Organizacijų informacija
10.	Duomenų bazės administratorius suskirsto	(IN) Teritorijų paskirstymo

	teritorijas	informacija
11.	Duomenų bazės administratorius suveda/koreguoja specifinę informaciją	(IN) Bendro pobūdžio/specifinė informacija
12.	Atstovas pateikia duomenų korekcijos prašymą administratoriui	(IN) Prašymas duomenų korekcijai
13.	Atstovas pasiima informaciją apie klientus	(OUT) Informacija apie klientus
14.	Atstovas pasiima savo nustatytus tikslus	(OUT) Atstovo tikslai

2.1.2. Analogų (konkurencinių CRM sistemų) analizė

Rinkos analizė, verslo galimybės

Pagrindiniai rinkos veikėjai – didelių, universalių produktų adaptuotojai. Kitaip sakant, dauguma šiuo metu rinkoje siūlomų sprendimų šiai sričiai yra horizontalių CRM platformų (tokių kaip „Oracle Siebel“ arba „Microsoft Business Solutions CRM“) pritaikymas vertikaliam sprendimui – farmacijos kompanijų veiklai. CRM sistemas adaptuoja dauguma Lietuvos pagrindinių programinės įrangos kūrimu užsiimančių įmonių: „Alna“, „Sonex“, „Baltic Amadeus“ ir t.t.

Pavyzdžiui, JAV kompanija „C3I”, teikia paslaugas ir programinę įrangą farmacinėms kompanijoms, kaip pagrindą naudodama „Oracle Healthcare Solution“. Iš esmės „Oracle Healthcare Solution“ apima du produktus: „Oracle Siebel CRM“ ir „Oracle Siebel Business Analytics“.

Konkurentų pagrindiniai trūkumai susiję su sistemų lankstumo stoka ir labai aukštais pakeitimų kaštais. Dažnai nutinka taip, jog sistemą diegiant skirtingiems klientams, jie išreiškia specifinius reikalavimus, būdingus tik jų atstovybių verslo procesui. Tokiu atveju lankstumo stoka ir aukšti pakeitimų realizavimo kaštai tampa labai svarbia problema, naudojant (diegiant) konkrečią, jau sukurtą CRM platformą.

Šiame darbe sistema bus projektuojama tiesiogiai dalyvaujant būsimiems klientams-sistemos vartotojams. Taipogi nebus naudojamos esamos CRM platformos. Iš vienos pusės, tai turėtų suteikti sistemai funkcionalumo pranašumą ir neribotas adaptavimo galimybes, iš kitos – sistemos kūrimas gali užtrukti šiek tiek ilgiau, nei pritaikant farmacijos atstovybėms jau sukurtus programinės įrangos gigantų „horizontalius“ CRM sprendimus.

Kitas planuojamas konkurencinis pranašumas – konkurencinga kaina.

IMS Health sprendimas

Viena pirmaujančių pasaulio informacijos ir informacinių sprendimų tiekėja farmacinėje ir sveikatos apsaugos srityje yra **IMS Health** kompanija [24].

Atlikus nuodugnius rinkos tyrimus išsiaiškinta, jog **IMS Health** kompanija savo programinės įrangos pagrindu pasirinko Microsoft SQL Server Analysis Services platformą.

2000 m. pradžioje buvo pristatyti 3 programiniai produktai veikiantys Microsoft Analysis Services pagrindu:

- **IMS Sales Analyzer.** Sprendimų palaikymo sistema skirta pardavimų padalinių vadovams. Pagrindinis jos tikslas – pateikti išsamią pardavimų duomenų analizę. Pagrindinės funkcijos: pardavimų stebėjimas, planavimas ir ataskaitų generavimas. Tai yra Windows pagrindu veikianti sistema užtikrinanti visas Microsoft Excel funkcionalumo galimybes. Sukurta naudojant Visual Studio 6.0.
- **IMS Territory Planner (online).** Sistema padedanti priimant efektyvius taktinius sprendimus, užtikrinanti strateginį nuoseklumą ir duomenų valdymą. Ji pritaikyta eiliniams farmacinių atstovybių darbuotojams efektyviai planuoti savo veiklą. Sukurta naudojant Visual Studio 6.0, įdiegta naudojant Microsoft Internet Information Server 5.0 (IIS).
- **IMS Territory Planner (offline)** užtikrina panašias funkcijas kaip ir online versija, tačiau skirta asmeniniams kompiuteriams neturintiems priėjimo prie interneto [24].

MicroView sprendimas

MicroView ETMS (angl. Electronic Territory Management System) sistema, kaip teigia jos kūrėjai, yra keleto programinių modulių sistema, leidžianti ne tik fiksuoti atstovybės darbuotojų veiklos rezultatus, bet taip pat turi tvirtą sprendimų palaikymo posistemę. „MicroView ETMS“ sudaro kelios pagrindinės posistemės [23]:

- **Call Reporting Module.** Naudodami šį posistemį atstovai gali registruoti savo vizitų informaciją. Šioje posistemėje įdiegta klientų kategorizavimo galimybė. Taip pat yra ir papildomų funkcijų:
  - Vizitų planavimas;
  - Teritorijų, gydytojų bei medicininių organizacijų paskirstymas tarp gydytojų;
  - Vizitų duomenų analizė.
- **Sales Analysis Module.** Ši posistemė suteikia galimybę analizuoti parduotos produkcijos duomenis įvairiais pjūviais. Duomenys gali būti pateikti įvairiais formatais.
- **Cost Checking Module.** Posistemė suteikia galimybę registruoti kiekvienam atstovui išduotą reklaminę medžiagą bei produkcijos pavyzdžius. Taip pat galima registruoti visas su kiekvienu atstovu susijusias išlaidas. Sukaupti duomenys gali būti analizuojami įvairių specializuotų ataskaitų pagalba [23].

Sistema sukurta naudojantis Borland Delphi programinio paketo pagalba, komunikavimas tarp nutolusių sistemos vartotojų ir centrinės duomenų bazės vykdomas naudojant Outlook Express elektroninio pašto programą.

TelyNET sprendimas

TelyNet kompanijos sistema **TelyNET sales ETMS** yra skirta farmacinių kompanijų atstovams. Ši programinė įranga yra pritaikyta darbui vietiniame tinkle (LAN), naudojant Windows operacinę sistemą. Pagrindinės sistemos funkcijos:

- Gydytojų, vaistinių ir medicininių organizacijų duomenų bazės tvarkymas;
- Gydytojų bei teritorijų paskirstymas tarp atstovų;
- Atstovų vizitų bei kitokios veiklos registravimas;
- Atstovų veiklos statistinių rodiklių skaičiavimas;
- Atstovų išlaidų registravimas;
- Kelionių maršrutų planavimas;
- Darbotvarkės sudarymas.

Sistemoje taip pat yra papildomas pardavimų analizės įrankis, kurio pagalba produkcijos pardavimų duomenis, ateinančius iš išorinių šaltinių galima perkelti į sistemos duomenų bazę ir perkeltus duomenis analizuoti pagal įvairiausius kriterijus [33].

Esami užsakovo - UAB „Softdent“ sprendimai

Planuojamas pakeisti (angl. refactor) senasis (angl. legacy) UAB „Softdent“ programinis kompleksas yra specialiai pritaikytas farmacines kompanijas atstovaujančioms įmonėms. Programų paketą sudaro dvi pagrindinės programos: **Customer Profiling** bei **Sales Reporting**. Abi programos gali veikti kaip atskiros programos arba kartu. Veikdami kartu abu moduliai naudojasi bendromis duomenų bazėmis. Taigi duomenų pakeitimai atliekami vienoje iš programų automatiškai atsispindi ir kitoje programoje. Pagrindinis abiejų posistemių veikimo kartu privalumas yra galimybė generuoti kombinuotas ataskaitas, kuriuose analizuojami bei lyginami abiejų programų duomenys. Programos sukurtos Microsoft Access priemonėmis.

Pagrindinės - **Customer Profiling** programos CRM funkcijos yra šios: klientų - gydytojų, vaistinių bei medicininių organizacijų duomenų bazės tvarkymas, atstovų veiklos kiekybinis bei kokybinis veiklos įvertinimas, gydytojų potencialo įvertinimas ir kt. Programoje yra suteikta galimybė gydytojus kategorizuoti pagal norimus kriterijus, ir, priklausomai nuo priskirtų kategorijų, iš anksto numatyti vizitavimo dažnį. Naudodamiesi programa atstovai gali planuoti vizitus bei registruoti jau įvykusių vizitų informaciją.

Programos priemonėmis kiekvienam atstovui galima sudaryti jo vizituojamų gydytojų sąrašus bei apibrėžti jo veiklos teritoriją. Pardavimų padalinio vadovai visus turimus duomenis gali analizuoti įvairiais pjūviais naudodami įvairiausius ataskaitų generatorius.

Sales Reporting programos pagrindinis tikslas – produkcijos pardavimų analizė. Farmacinės kompanijos produkcijos pardavimus vykdo ne tiesiogiai, o per didmenininkus. Nustatytais laiko intervalais didmenininkai pateikia detalias pardavimų ataskaitas. Programos priemonių pagalba šių ataskaitų duomenys gali būti importuojami į programos duomenų bazę tolesnei analizei. Duomenys gali būti importuojami iš įvairiausio formato ir struktūros failų. Importuoti duomenys gali būti analizuojami įvairiais pjūviais, priklausomai nuo vartotojo pageidavimų.

Programos sukurtos naudojant Microsoft Access bei Microsoft Visual Basic 6.0 įrankius. Programų on-line versija realizuojama naudojant Microsoft Terminal Server technologiją. Programa bei duomenys yra saugomi terminaliniame serveryje, o sistemos vartotojai internetinio ryšio bei specialios kliento programinės įrangos pagalba jungiasi prie serverio ir serverio aplinkoje dirba su programomis.

Programų sistemų kiekybinis ir/arba kokybinis palyginimas

Išanalizavus anksčiau aptartas sistemų realizacijas nesunku pastebėti jų tarpusavio bendrumus bei skirtumus. Visų pirma labai panaši visų sistemų architektūra. Išskiriami du pagrindiniai posistemiai: atstovų veiklos (CRM) bei pardavimų analizės moduliai.

Vienintelėje **MicroView ETMS** sistemoje yra realizuotas išlaidų registravimo ir apskaitos modulis. Tačiau šios posistemės funkcionalumas apsiriboja išlaidų fiksavimu bei jų analize. Programoje nėra galimybės generuoti kombinuotas ataskaitas, palyginančias išlaidas su atstovų vizitais ir/ar produkcijos pardavimais. Tai pat nėra piniginių resursų paskirstymo funkcijos, kuri planuojama realizuoti kuriamoje sistemoje.

Lyginant tarpusavyje sistemų kūrimo įrankius be abejonės išsiskiria IMS Health kompanija, kuri naudoja pažangesnes technologijas (MS SQL Server) (IMS Health Meets Data Analysis and Business Decision Support Demands with Microsoft technology, 2002) lyginant su kitomis paminėtomis kompanijomis. Technologiniu požiūriu į šią sistemą ir bus lygiuojamasi, pasirenkant MS SQL Server 2005 ir .NET technologijas.

Įvertinti ir palyginti tarpusavyje paminėtų kompanijų programinės įrangos kūrimo sąnaudas bei pardavimo kainas yra faktiškai neįmanoma, kadangi šie duomenys yra konfidencialūs ir viešai neskelbiami.

Situacijos Lietuvoje įvertinimas

18 iš 21 etinių farmacijos kompanijų atstovybių asociacijos (EFA) narių šiuo metu yra UAB „Softdent“ klientai. Etinės farmacijos kompanijos – tai kompanijos, kuriančios naujus, patentinius receptinius vaistus, bei pripažįstančios tarptautinės farmacijos gamintojų asociacijų federacijos (IFPMA) farmacijos preparatų marketingo praktikos kodeksą [22].

Tačiau Lietuvoje yra ir ne etines farmacines kompanijas atstovaujančių įmonių - taipogi veikia keletas nepriklausomų atstovybių. Didžioji dalis Lietuvos farmacinių atstovybių (šiuo metu apie 40) naudoja vienokią ar kitokią UAB „Softdent“ programinę įrangą. Visos šios kompanijos yra potencialūs naujosios sistemos pirkėjai ir vartotojai. Tačiau prognozuojama, kad naujoji sistema gali sudominti ir kitas kompanijas, šiuo metu nenaudojančias „Softdent“ programines įrangos.

2.1.3. Projektavimo metodologijos ir technologijų analizė

Temos aktualumas ir perspektyvumas

Laikas, skirtas farmacijos kompanijų pardavimų atstovams – pristatant kompanijos naujausius vaistus, jų tikėtinus rezultatus ir šalutinius efektus, siūlant vaistų pavyzdžius – nuolatos mažėja. Farmacijos produktų reklamavimas dabar reiškia kur kas daugiau nei vien ramus pietų pasisėdėjimas su gydytojais ir juos aptarnaujančiu personalu. Nuolatos aštrėjanti konkurencija tarp vaistų gamintojų, kintanti ir dinamiška sveikatos apsaugos rinka, kylantys vaistų gamybos ir realizavimo kaštai – visa tai prisideda prie laiko, kurį atstovas praleidžia su gydytoju, mažėjimo.

Tradiciškai buvo nusistovėję tai, kad farmacinės kompanijos pardavimų skyriaus apimtis tiesiogiai įtakodavo tos kompanijos rinkos dalį. Tačiau, pavyzdžiui JAV atstovų kiekiui padvigubėjus iki 80000 jau 1996-tais metais buvo pastebėta gerokai sumažėjusi koreliacija su investicijų grąža (angl. ROI – Return On Investment).

Pardavimų skyriaus efektyvumo didinimas farmacijos pasaulyje dabar turi prasidėti pirmiausia nuo kritinės, varomosios jėgos - informacinių technologijų. Atlikus 90 JAV vaistų gamintojų apklausą paaiškėjo, jog net 26% įmonių IT biudžeto yra skiriama pardavimams ir marketingui, o tai pagrindinai sudaro vadinamosios CRM (angl. Customer Relationship Management) ir BI (angl. Business Intelligence) programų sistemos. Dauguma farmacijos kompanijų prisitaiko CRM sistemas savo specifinėms reikmėms, norėdamos automatizuoti tokius procesus kaip ataskaitų generavimas, prioritetų nustatymas ir pan. CRM sistemos dažniausiai apima ir pardavimų skyriaus automatizuotą valdymą, duomenų sandėliavimą, mobiliuosius įrenginius.

CRM ir BI sistemos atstovybėse naudojamos lygiagrečiai. Atstovas privalo planuoti, vykdyti ir atsiskaityti už vizitus. CRM analizės posistemės ne tik padeda identifikuoti gydytojus, pritaikyti sistemą sąveikauti atstovui su gydytoju jam priimtiniausiu būdu, bet ir pateikia vertingas tendencijas.

SOA ir pakartotinis panaudojamumas

Vienoje iš konferencijų pagrindinis pranešėjas [21] paklausė: „Kiek iš jūsų dirbote kompanijoje, kuri turėjo stambaus masto pakartotinio objektų panaudojimo programą?“ Patalpoje, kurioje buvo susirinkę apie šimtą didelių įmonių darbuotojų, beveik visos rankos pakilo. Tuomet pranešėjas klausė toliau: „Prašau pakelti ranką tuos, kurių įmonėje **vis dar yra** stambaus masto pakartotinio objektų panaudojimo programa“. Liko pakelta tik viena ranka [21].

Šis klausimas buvo pasirinktas neatsitiktinai, o norint pademonstruoti tai, ką visi ir taip gerai supranta: objektų pakartotinis panaudojimas – sunkiai praktiškai įvykdomas reikalas. Nepaisant didelių vadovų, architektų ir programuotojų pastangų, kultūriniai ir verslo skirtumai buvo ir yra paprasčiausiai pernelyg dideli. Pasak garsaus technologijų guru ir daugelio straipsnių bei knygų autoriaus David Chappell [21], verslo logikos pakartotinis panaudojimas yra beveik toks pat keblus su servisais, kaip ir su objektais (klasėmis). Pagal šiuos duomenis būtų galima tikėtis pakartotinai panaudoti tik mažą dalį savo servisų, galbūt tik apie 20%. Jei servisų pakartotinis panaudojamumas yra toks mažas, gal neverta apskritai žavėtis SOA? Pagal tokią statistiką, tikėtina jog tik vienas iš penkių servisų gali būti panaudotas pakartotinai.

Tačiau yra atvejų, kur servisai gali iš tiesų padėti pasistūmėti į priekį ir suteikti daug pranašumų. Bet kas, besiruošiantis kurti sistemą, kuri turės keletą skirtingų prieigų (pvz. JSP portalas, Windows Forms klientas, mobilus įrenginys) iš karto supras, jog kurti sistemą SO architektūra – puikus sprendimas. Idėja, leidžianti prieiti skirtingiems klientams per tą pačią prieigą palengvins gyvenimą abiems pusėms.

Kita vertus, yra ir tokių kompanijų, kurios pasiekia pakankamai aukštą servisų pakartotinio panaudojimo lygį. Žinoma, visa tai pasiekiama su dideliu atsidavimu, kruopštumu, na ir žinoma, vyriausiųjų vadovų palaikymu. Šios kompanijos teigia tokiu būdu smarkiai sutaupančios. Tas pats galiojo ir su objektų pakartotiniu panaudojimu. Tačiau tik pirmaisiais metais. Realybė parodo ką kita: verslo logikos servisų pakartotinis panaudojimas bus sunkiai pasiekiamas tikslas daugeliui, o gal net ir didžiajai daugumai.

Pagrindiniai sunkumai, su kuriais susiduria kiekvienas, besiruošiantis pakartotinai panaudoti servisus:

- Servisų, kuriuos būtų galima panaudoti vėliau, sukūrimas reikalauja ateities pranašavimo. O tai yra labai keblu: kaip servisų kūrėjai gali nuspėti ko reikės ateities taikomosioms programoms?
- Jei programa ir teikia kokį tai servisą, dažniausiai tai būna ne visai tai ko nori klientas. Gali nutikti taip, kad klientas pageidaus papildomos informacijos, arba servisas leis prieigą prie duomenų, kurių klientas neturėtų matyti. Taip serviso kūrėjai gali būti priversti kurti galybes variacijų, norėdami patenkinti kliento reikalavimus.
- Nors vienas organizacijos padalinys ir sukuria servisą, kurį būtų galima panaudoti pakartotinai, jo (serviso) projekto vadovas neturi jokio intereso leisti kitai grupei naudotis šiais resursais. Kas jam/jai iš to? Nors keletas įmonių sprendžia šią problemą centralizuodami pakartotinio panaudojimo objektų ir servisų valdymą, daugeliu atveju tai nėra išeitis. Kokia verslo grupė norėtų atiduoti savo strateginių produktų kontrolę į kitų rankas? Juk kažkas visgi moka pinigus už kiekvieną kodo gabalą, kuris teikia servisą. Ir kas tas mokėtojas bebūtų, tikrai nenorės kad tą servisą naudotų kiti be jų aiškaus sutikimo.

O ką duoda SOA verslo lankstumui? Juk vienas iš SOA šalininkų argumentų yra tas, jog SOA padarys IT sistemas lengviau keičiamas, o verslą – gebantį greičiau prisitaikyti prie kintančio pasaulio. Pagrindinė to priežastis yra ta, jog teikiami servisai gali palengvinti programinės įrangos perkonfigūravimą, ypatingai tada, kai tam sukurti patogūs įrankiai. Visgi ar nėra tai tik dar vienas pakartotinio panaudojimo būdas? Sutikus su tuo, visas verslo lankstumas galų gale atsiremia į servisų pakartotinį panaudojimą.

Microsoft „.NET Remoting“ modelis

Paskirstytasis programavimas tapo beveik esmine viso programų kūrimo dalimi. Dar prieš atsirandant „.NET Remoting“, DCOM (angl. Distributed Component Object Model) buvo pagrindinis Microsoft platformų paskirstytų sistemų kūrimo metodas. Kaip ten bebūtų, DCOM yra pakankamai sudėtingas vidutiniam programuotojui. Taip, kaip .NET Framework pakeitė COM (angl. Component Object Model), taip .NET Remoting pakeičia DCOM [26].

.NET Remoting palaiko atviruosius Interneto standartus, tokius kaip Web-servisai ir SOAP. Kaip jau išsiaiškinome, tai nėra tobuli standartai. Tačiau dažniausiai galima rasti beveik bet kokios problemos sprendimą. Arba galima ją apeiti.

Pagrindinės problemos, su kuriomis susiduria paskirstytosios sistemos yra:

Našumas.

Yra galybė faktorių, galinčių įtakoti paskirstytosios sistemos našumą. Keletas iš pavyzdžių galėtų būti išoriniai faktoriai: tinklo pralaidumas, tinklo apkrautumas, išorinės techninės įrangos sparta (procesoriaus, I/O posistemės, atminties ir pan.). Turint omenyje dabartines

parskirstytų sistemų kūrimo technologijas, našumas ir suderinamumas yra abipusiai nesuderinami dalykai. Pvz. jei sistema privalo veikti taip sparčiai, kaip tik įmanoma, dažniausiai reikės sistemos klientą ir serverį kurti tose pačiose platformose, apriboti tinklo prieigą. Esant tokiems reikalavimams, paskirstytoji sistema galės naudoti efektyvų tinklo protokolą (pvz. TCP) ir siųsti duomenis atitinkamu dvejetainiu formatu. Šie formatai yra daug efektyvesni nei universalūs, tekstiniai formatai, kurie dažniausiai yra privalomi atviruosiuose standartuose (pvz. SOAP) [29].

Saugumas.

Paskirstytoji sistema turi turėti šių trijų problemų sprendimus:

Autentikacija: serveriai privalo žinoti jog klientas yra tas, kuo jis dedasi esąs.

Kriptografija: serveriui atpažinus vartotoją, jis turi užtikrinti komunikacijų saugumą.

Prieigos kontrolė: Serveriui atpažinus klientą, jis turi turėti galimybę nustatyti ką klientas gali daryti (autorizacija). Pavyzdžiui kokias operacijas gali iškviesti ir kokius failus gali skaityti/rašyti [32].

Suderinamumas.

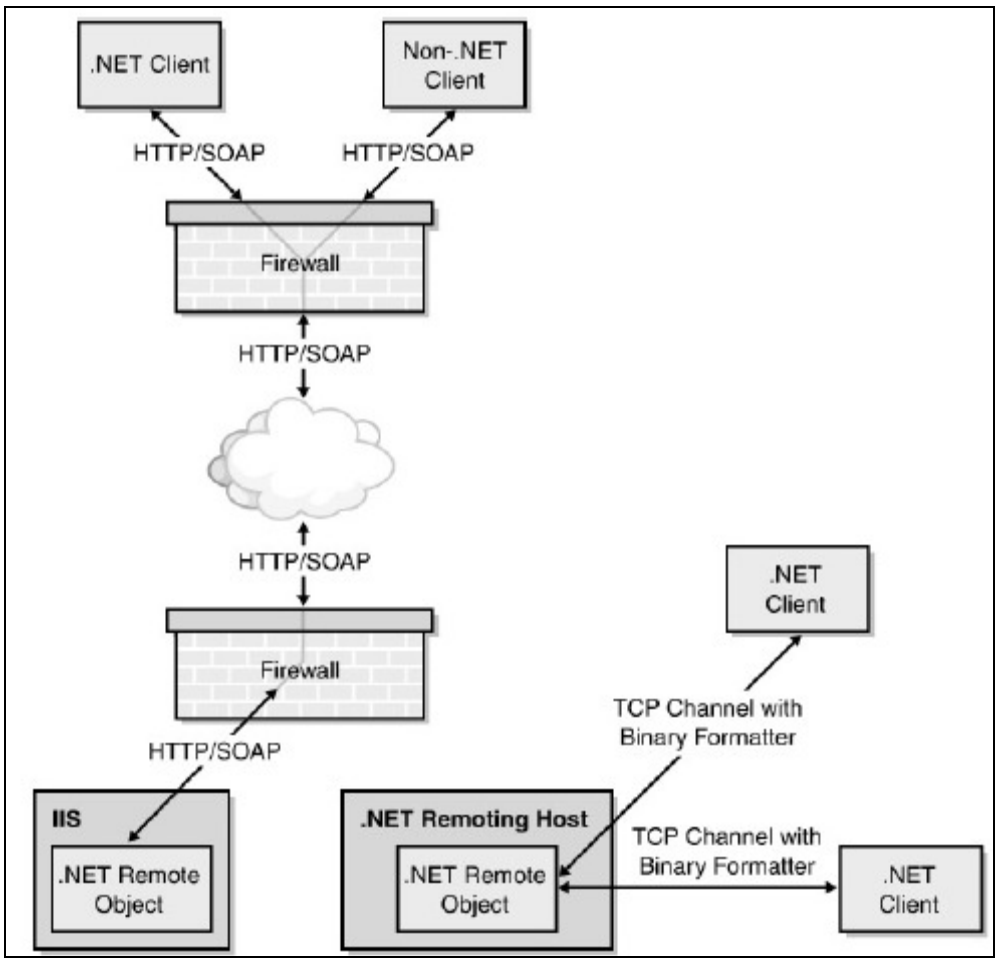
Internetas ir tinklo barikados.

Pačios populiariausios paskirstytosios sistemos iš esmės buvo sukurtos veikti privačiame tinkle. Nors internetas gyvuoja jau nemažai metų, iki šiol jis buvo naudojamas pagrindinai failų persiuntimui, el.paštui ir Web serveriams, teikiantiems HTML puslapius peržiūrai. Dauguma žmonių nenaudojo Interneto kaip paskirstytųjų sistemų komunikavimo terpės. Laikui bėgant, kompanijos pradėjo įgyvendinti tinklo saugumo strategijas, kurios dažniausiai apimdavo „blokavimą visko, išskyrus HTTP protokolą (dažniausiai per 80 portą)". Senųjų paskirstytųjų sistemų komunikavimo formatai ir protokolai dažniausiai reikalaudavo nesaugių portų atidarymo, o tai dažniausiai sudarydavo klientams nemažų problemų ir nesaugumo jausmą. Kaip bebūtų, dabartinės paskirstytosios sistemos privalo komunikuoti per HTTP.

Konfigūravimas.

Objektų ir jų resursų gyvavimo laiko administravimas.

Šias pagrindines problemas ir mėgina išspręsti .NET Remoting'as. Pavyzdžiui, nutolusio objekto (Web-serviso) adaptavimas prie pakitusių našumo-suderinamumo sąlygų yra paprasčiausias kelių įrašų pakeitimas konfigūraciniame faile. Klientams, kuriems reikės objektą pasiekti per stipriai apribotą ugniasienę, bus galima sukonfigūruoji prieigą per HTTP ir SOAP.

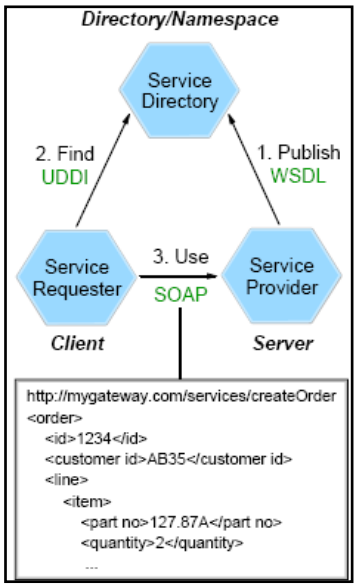


Pav. 3 – .NET Remoting objektų konfigūracijos lankstumas: suderinamumas ir našumas

XML Web-servisai

Trumpas web-servisų aprašas

Web-servisas gali būti apibūdinamas kaip programų sistema, kurios pagrindinis tikslas yra apjungti tinkle esančius kompiuterius. Web-servisai naudoja SOAP (angl. Simple Object Access Protocol) protokolą. SOAP ir HTTP protokolų pagalba Web-servisai perduoda žinutes XML formatu.



Web-servisai turi turėti dvi pagrindines savybes:

- Web-servisas turi aptarnauti visus klientus, nepriklausomai nuo platformos, kurią klientas naudoja;
- Klientas turi turėti galimybę surasti ir naudoti Web-servisus nepriklausomai nuo jų realizacijos ypatybių ir programinės terpės, kurioje yra Web-servisas.

Pav. 4 – SOAP Web-serviso architektūra

Web-servisas yra technologija, leidžianti integruoti skirtingas programų sistemų funkcijas. Jie gali būti realizuoti įvairių programavimo kalbų pagalba, skirtingose platformose, veikti tiek Internete, tiek intranete. Web-servisas yra tarsi naujo tipo web-programa, kurią galima publikuoti ir integruoti internete. Jis gali apimti įvairiausias funkcijas: nuo paprasčiausių skaičiavimų iki galingų ir sudėtingų verslo sistemų. Visa tai padaro Web-servisą dinamišku įrankiu, kurį galima pritaikyti (beveik) kiekvieno kliento poreikiams [25].

XML

XML (angl. Extensible Markup Language) yra SOAP protokolo pagrindas. XML yra kalba, leidžianti žinutės siuntėjui ir jos gavėjui apsikeisti informacija, nepriklausomai nuo platformos ar jų naudojamų technologijų. Web-servisas ir jo naudotojas apsikeičia žinutėmis XML formatu, dažniausiai per HTTP protokolą. XML (kaip ir HTML) yra hierarchinės struktūros, žymėmis (angl. tag) aprašoma kalba. Pagrindinis skirtumas yra tas, jog HTML paskirtis – informacijos atvaizdavimas, o XML – informacijos aprašymas.

SOAP

Pagal W3C (angl. World Wide Web Consortium), SOAP (angl. Simple Object Access Protocol) susideda iš trijų pagrindinių dalių. Pirmoji dalis yra tarsi apvalkalas, aprašantis ką žinutė talpina ir ką su ja reikia daryti. Antroji – aibė kodavimo taisyklių, aprašančių programos duomenų tipus. Ir trečioji dalis susidaro iš formalių aprašų, naudojamų atvaizduojant RPC (angl. Remote Procedure Calls) iškvietimus ir atsakymus. W3C teigia, jog SOAP gali būti suderinamas su įvairiais transportiniais protokolais, tačiau dažniausiai naudojamas – HTTP [31].

WSDL

WSDL yra XML kalba parašytas aprašas, nurodantis kaip naudotis Web-servisu [30].

W3C (2001) apibrėžia WSDL (angl. Web Services Description Language) kaip XML formatą, skirtą aprašyti Web-servisus kaip aibę tinklo galinių taškų, tarpusavyje komunikuojančių žinutėmis. Pagal IBM (2001) - WSDL kalba yra universali, kai kalbama apie metodų naudojimą. Tai ypatingai pastebima, dirbant su UDDI (angl. Universal Description, Discovery and Integration) registrais daugeliu skirtingų būdų.

UDDI

UDDI (angl. Universal Description Discovery and Integration) suteikia vartotojui servisų paieškos ir publikavimo metodus.

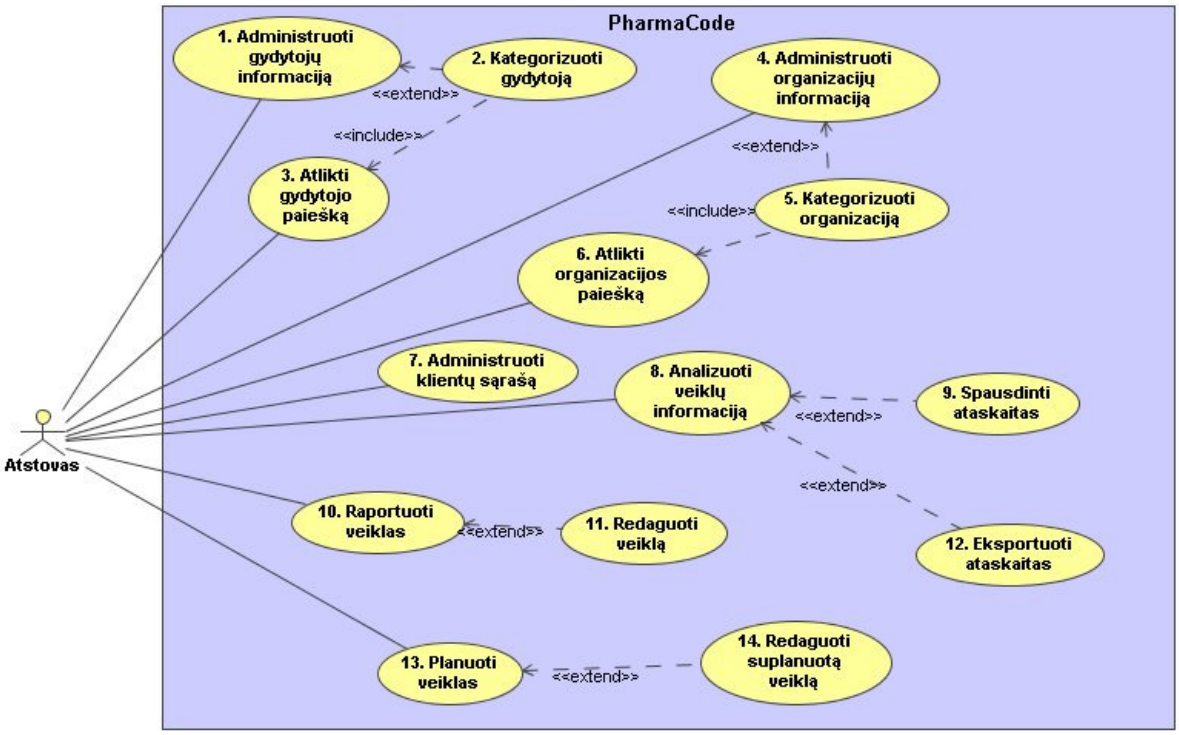
Intel (2005) apibūdina UDDI kaip centralizuotą prieinamų Web-servisų sąrašo saugojimo vietą. Toliau UDDI aprašoma kaip viena „skelbimų vieta“, kurioje klientas susiranda viską ko jis/ji nori, ir tuomet UDDI registras perduoda sąveikos su servisu metodų antraštes.

Web-servisų tinkamumo projektuojamai sistemai analizė

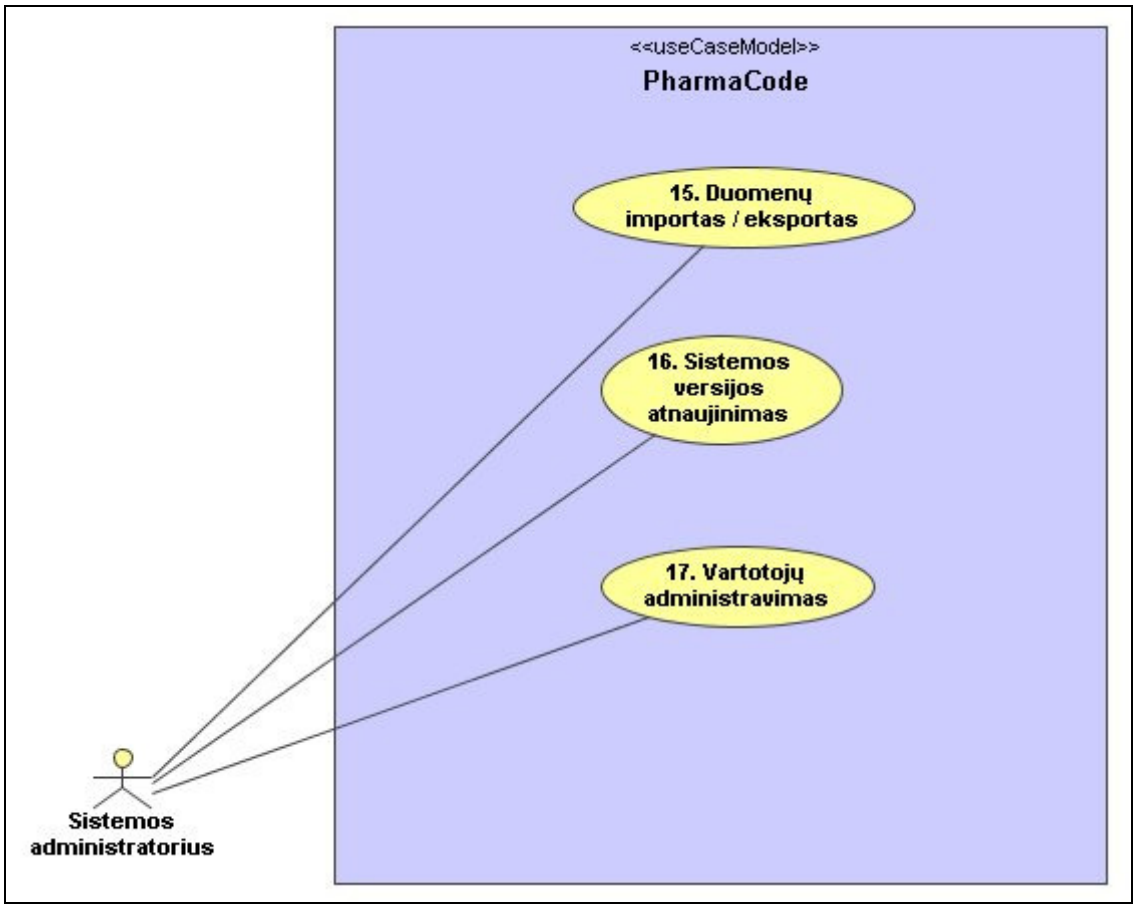
Išanalizavus Web-servisų technologinę specifiką, surinkus informaciją apie tokius protokolus ir technologijas kaip SOAP, XML ir UDDI, buvo prieita išvada, jog prieš naudojant Web-servisus konkrečiai sistemai projektuoti reikia ypatingą dėmesį atkreipti į užsakovo nurodytus reikalavimus sistemos našumui, greitaeigiškumui ir saugumui. Šie faktoriai buvo paminėti beveik visoje išnagrinėtoje SOA literatūroje. Dėl SOAP protokolo specifikos, perduodant žinutes XML formatu, tinklu perduodami sąlyginai dideli duomenų kiekiai. Taip yra todėl, kad žinutės perduodamos su dideliu kiekiu perteklinės informacijos. Tai yra XML kalbos specifika, ir nieko šiuo atžvilgiu negalima padaryti. Kadangi projektuojama sistema bus prieinama ne vien plačiajuosčiu ryšiu, o ir per lėtesnius (pvz. GPRS) tinklus, ši problema gali būti tikrai aktuali, ir kiekvieną planuojamą servisą (pvz. „Reporting“, „Call registering“, „Synchronizing“ ir pan.) reikia gerai išanalizuoti atskirai, atkreipiant dėmesį į duomenų srautus ir jų apimtį. Paaiškėjus, jog duomenų perdavimo sparta ir sistemos našumas netenkina užsakovo reikalavimų, bus galima servisą realizuoti .NET platformos „Remoting“ objektu („Server component“). Pastaroji servisų realizacija nebūtų tokia jautri duomenų apimčiai, kadangi visi tinklu perduodami objektai yra koduojami tiesiai į dvejetainį kodą, o ne į XML formatą. „Neįvelkant“ tikrosios perduodamos informacijos į žymes (angl. tag) būtų padidinamas sistemos našumas, greitaeigiškumas, sumažinamas tinklo apkrovimas, tačiau šis metodas padaro sistemą ne tokią universalią, kaip XML Web-servisai.

2.1.4. Reikalavimų projektuojamai sistemai analizė

Sistemos panaudojimo atvejai



Pav. 5 - sistemos panaudojimo atvejų diagrama atstovo požiūriu



Pav. 6 - sistemos panaudojimo atvejų diagrama administratoriaus požiūriu

Programų sistemos funkcijos

- **Administruoti gydytojų informaciją –** šis panaudojimo atvejis apima visas gydytojų duomenų administravimo funkcijas:
    - **Įvesti naują gydytoją** – Atstovas gali įvesti naują gydytoją į duomenų bazę. Atstovas taip pat gali ištrinti gydytoją iš duomenų bazės.
      Bus galimybė apriboti atstovui teisę įvesti/ištrinti gydytoją.
    - **Redaguoti gydytojo informaciją** – Visą apie gydytojus įvestą informaciją vėliau bus galima redaguoti. Galimybė apriboti atstovui teisę redaguoti gydytojų informaciją.
    - **Kategorizuoti gydytoją** – Gydytojai gali būti kategorizuojami nustatant jiems atitinkamus "VIP" (svarbumo) lygius. VIP lygis gali būti įrašomas rankiniu būdu, arba suskaičiuojamas pagal tam tikrą klausimyną. Gydytojai kategorizuojami dvejopai:
        - **Bendrasis gydytojo VIP lygis** – Vienam gydytojui nustatomas tik vienas, bendras VIP lygis.
        - **Gydytojo VIP lygis pagal produktą** – gydytojui nustatomi keli VIP lygiai, priklausomai nuo produkto. Tuomet bendrasis VIP lygis suskaičiuojamas pagal pagrindinį šio gydytojo produktą, arba pagal aukščiausią VIP lygį tarp visų produktų.

      Galimybė apriboti atstovui teisę kategorizuoti gydytojus.
- **Administruoti organizacijų informaciją** – šis panaudos atvejis apima visas organizacijų informacijos administravimo funkcijas:
    - **Įvesti naują organizaciją** – atstovas gali įvesti naują organizaciją į duomenų bazę. Atstovas taip pat gali ištrinti organizaciją iš duomenų bazės.
      Galimybė apriboti atstovui teisę įvesti naują organizaciją/ištrinti organizaciją.
    - **Redaguoti organizacijų informaciją –** Visą apie organizacijas įvestą informaciją bus galima redaguoti. Galimybė apriboti atstovui teisę keisti organizacijų informaciją.
    - **Kategorizuoti organizaciją** – Organizacijos kategorizuojamos nustatant jų VIA (svarbumo) lygius. VIA lygiai gali būti nustatomi rankiniu būdu, arba suskaičiuojami pagal klausimyną.
- **Atlikti gydytojų paiešką** – programos pagalba turi būti paprasta ieškoti ir peržiūrėti informaciją apie gydytojus. Paiešką galima atlikti pagal visus duomenų laukus.

- **Atlikti organizacijų paiešką** – programos pagalba turi būti paprasta ieškoti ir peržiūrėti informaciją apie organizacijas. Paiešką galima atlikti pagal visus duomenų laukus.
- **Administruoti klientų sąrašą** – kiekvienas atstovas turi savo klientų sąrašą, t.y. gydytojų ir vaistinių, kurias jie vizituoja, sąrašą. Atstovas turi galimybę redaguoti tik savo klientų sąrašą:
    - **Įtraukti naują klientą** – atstovas gali įtraukti gydytoją/vaistinę į savo klientų sąrašą.
    - **Išmesti klientą** – Atstovas gali išmesti gydytoją/vaistinę iš savo klientų sąrašo.

  Galimybė apriboti atstovui teisę redaguoti klientų sąrašą.
- **Raportuoti veiklas** – šis panaudos atvejis apima įvairių veiklų raportavimo galimybės:
    - **Raportuoti vizitą** – Vizitas yra viena iš veiklos rūšių. Jis gali būti dviejų tipų : gydytojų arba vaistinių vizitas. Atstovas taip pat gali ištrinti savo vizitų informaciją.
    - **Raportuoti susitikimą** – Susitikimas yra toks veiklos tipas, kai vienu metu aplankomi keletas klienų.
    - **Raportuoti užduotį** – Užduotis yra veikla, nesusijusi su pagrindinėmis atstovo veiklomis. Tai gali būti liga, atostogos, kursai ir pan.
    - **Redaguoti veiklą** – atstovas gali redaguoti visų savo veiklų (vizitų, susitikimų, užduočių) informaciją.

- **Planuoti veiklas** – atstovas turi galimybę suplanuoti savo ateities veiklas. Sistemoje planuojamų veiklų informacija yra ta pati, kaip ir raportuojamų. Pagrindinis skirtumas – veikla turi indikatorių „Jau įvykdytas/planuojamas“. Jei tai planas – indikatorius nustatomas į „Planuojamas“. Jei planas atliktas, indikatorius nustatomas į „Jau atliktas“. Atstovas gali redaguoti tiek planuojamos, tiek atliktos veiklos informaciją.

Analizuoti veiklų informaciją – programoje turi būti realizuota patogi galimybė analizuoti veiklų istoriją įvairiais pjūviais. Atstovas gali analizuoti tiek savo, tiek ir kitų atstovų veiklos informaciją.

Sistemos kontekstas

Būsima sistema turės sąsają su dviem išorinėmis sistemomis:

- Senosiomis UAB „Softdent“ CRM sistemomis;
- „PharmaZOOM“ – farmacijos rinkos produktų pardavimų analizės sistema.

Realizuojant sąsają su senosiomis sistemomis bus būtina realizuoti patogius migravimo (perėjimo prie naujesnių programų) įrankius. Sąsaja su „PharmaZOOM“ pagrinde apims vaistų pardavimų statistikos išgavimą tolesnei statistinei analizei, siejant juos su vizitavimų veikla (koreliacijos, regresinė analizė ir pan.).

Vartotojų charakteristikos

Numatomi šie sistemos vartotojai:

- Analitikas – analizuoja atstovybės veiklos rezultatus
- DB administratorius – asmuo atsakingas už duomenų įvedimą ir tvarkymą. Naudodamasis sistemos priemonėmis įveda bei tvarko duomenis duomenų bazėje.
- Pardavimų vadybininkas – pardavimų padalinio vadovas. Jis naudodamasis sistemos priemonėmis atlieka duomenų analizę, gali naudoti pardavimų prognozės įrankius, prireikus koreguoja duomenis.
- Produktų vadybininkas – atsakingas už produktų informaciją ir jos integralumą CRM sistemoje.

Būsimi sistemos vartotojai turėtų būti įgiję bendrus darbo su kompiuteriu įgūdžius. Pageidautina, kad sistemos vartotojai turėtų patirties dirbant su panašiomis sistemomis.

Vartotojo tikslai ir problemos

Pagrindinis vartotojų tikslas - padidinti ir sekti vizitavimo efektyvumą. Patį vizitų raportavimo procesą padaryti kaip galima skaidresnį, lankstesnį, mobilesnį. Šiuo metu siūlomos programos negali pilnai patenkinti šių poreikių, o ypač mobilumo, todėl šią ir kitas spragas bus stengiamasi ištaisyti naujoje programų sistemoje.

Bendri apribojimai

Kadangi jau sukurti sistemos komponentai remiasi senstelėjusia technologija, naujai kuriama programų sistema turės visiškai naują architektūrą, tačiau esamiems klientams reikės pasiūlyti patogias migravimo, importavimo iš senosios sistemos galimybes.

Naudoti šias technologijas:

- MS SQL Server 2005 (taip pat Analysis ir Reporting Services).
- MS .NET Framework 2.0.

2.2. SQL-įterpinių saugumo problemos analizė

Šiame skyrelyje analizuojame SQL-įterpinių saugumo problemą, pateikiame jų atakų tipus ir galimus apsisaugojimo nuo jos būdus.

2.2.1. Įvadas

Greitai besivystantys internetiniai sprendimai apima vis daugiau sričių. Tai, kad nemaža dalis šių sprendimų yra pažeidžiami, rodo atlikti statistiniai tyrimai [18]. 2004 m. atliktas 250 internetinių sistemų tyrimas parodė, jog bent 92% iš jų buvo pažeidžiamos vienokių ar kitokių atakų [19]. Viena didžiausių pažeidimų grupių yra SQL-įterpinių atakos (toliau - SQLĮA), kurios buvo įtrauktos į didžiausių internetinių sistemų grėsmių dešimtuką [20].

2.2.2. SQL įterpinių atakos ir jų tipai

Šiame skyrelyje aprašysime SQLĮA, o toliau – trumpai paminėsime pagrindinius jų tipus.

SQLĮA – tai ataka, kurios metu įsibrovėlis gali pasinaudoti netikrinama (arba nepakankamai tikrinama) programos įvestimi tam, kad pakeistų suprogramuotų SQL užklausų logiką, semantiką ar sintaksę [1]. Tai padaroma įvedimo metu į sakinį įterpiant naujus raktinius žodžius arba operatorius. Formalų SQLĮA aprašymą galima rasti [2].

Standartinis pavyzdys, aprašant SQLĮA, yra ši vartotojo identifikavimo metu vykdoma SQL užklausa (pateikta C# kalba):

```
string sqlCommand =    "SELECT UserID FROM Users WHERE Username='" + username +
                       "' AND password = '" + password + "'";
```

Identifikavimo metu vartotojas prisijungimo formoje įveda vartotojo vardą ir slaptažodį, o šios užklausos pagalba grąžinamas jo identifikacijos numeris. Toliau programa, naudodamasi užklausos rezultatais, gali, pavyzdžiui, suteikti vartotojui administratoriaus teises sistemoje. Programa, naudojanti tokio pobūdžio užklausas, yra pažeidžiama SQLĮA. Pavyzdžiui, jei vartotojas vartotojo vardo laukelyje įveda „`administrator' --`“, o slaptažodžio laukelyje bet ką, pvz. „`slaptažodis`“, tuomet suformuojama tokia SQL užklausa:

```
SELECT UserID FROM Users WHERE Username='administrator' -- ' AND password = 'slaptažodis'
```

SQL kalboje simboliai „`--`“ reiškia komentarą. Duomenų bazių valdymo sistema (toliau - DBVS) ignoruoja viską, kas rašoma toliau už šio simbolio. Tokiu būdu pakeičiama pati SQL užklausos WHERE sąlyga. Užklausa grąžina nurodyto vartotojo vardo ID, visiškai netikrindama slaptažodžio.

Būtina pabrėžti, jog šis pavyzdys demonstruoja patį paprasčiausią SQLĮA pavyzdį. Vien dėl to, kad tokio tipo pavyzdžiai yra labai dažnai naudojami kaip iliustracijos literatūroje, nederėtų susidaryti nuomonės, jog tai vienintelis SQLĮA tipas. Iš tiesų egzistuoja

platus spektras sudėtingų SQLĮ spragą išnaudojančių metodų, detaliau aprašomų [3]. Čia paminėsime pagrindinius jų tipus.

Pasikartojimu paremtos loginės išraiškos (angl. tautologies)

Pasikartojimu paremtos loginės išraiškos – ko gero vienas paprasčiausių ir dažniausiai naudojamų SQLĮA tipų. Jo esmę sudaro loginės išraiškos, kurios visada išsprendžiamos į „`TRUE`“. Pavyzdžiui, išraiška „`1 = 1`“. Pažeidžiamai programai įvesties metu paduodama tokio tipo išraiška. Rezultatai gali būti įvairiausi ir tai priklauso nuo konteksto, kuriame naudojamas šis metodas. Tačiau dažniausiai jis panaudojamas prisijungimo/identifikavimo formų apėjimui ir duomenų nuskaitymui. Jei anksčiau aprašytai užklausai „`Username`“ parametrui paduotume reikšmę „`' OR 1=1 -- `“, turėtume tokią užklausą:

```
SELECT UserID FROM Users WHERE Username='' OR 1=1 --' AND password = ''
```

Kadangi WHERE sąlyga visada lygi „`TRUE`“, užklausa grąžins visų lentelėje esančių vartotojų ID.

„UNION“ užklausos

Nors tautologinės SQLĮ atakos leidžia apeiti identifikacijos formas, jos yra nelanksčios tais atvejais, kai iš duomenų bazės norime paimti specifinę informaciją. „UNION“ užklausos šiuo atžvilgiu yra sudėtingesnis ir patogesnis SQLĮA tipas, leidžiantis pasiekti šį tikslą. Jų pagalba prie gerų užklausų rezultatų galime prijungti norimus papildomus duomenis. Tokio tipo atakų metu įsibrovėlis įterpia tokios formos užklausas:

```
UNION <Įterptoji užklausa>
```

Pavyzdžiui, jei anksčiau demonstruotai užklausai vartotojo vardo vietoje įvestume „`' UNION SELECT Password FROM Users WHERE Username='administrator' --`“, programa sukonstruotų tokią užklausą:

```
SELECT UserID FROM Users WHERE Username='' UNION
SELECT Password FROM Users WHERE Username='administrator' --' AND password = ''
```

Jei užklausos rezultatai vienu ar kitu būdu parodomi vartotojui, taip įsibrovėlis jos pagalba išsiaiškina administratoriaus slaptažodį.

Prišlietos (angl. piggybacked) užklausos

Metodas panašus į aukščiau minėtąjį. Jo esmė – prie tikrosios užklausos prijungti papildomą užklausą. Jei ataka pavyksta, duomenų bazei nusiunčiamos vykdyti kelios užklausos vietoje numatytos vienos. Pirmoji užklausa dažniausiai būna tikroji. Tokio tipo atakos ypatingai pavojingos, kadangi įsibrovėlis tokio metodo pagalba gali įterpti praktiškai

bet kokią užklausą ar bet kokio tipo kitą komandą. Mūsų pavyzdyje vartotojo vardo vietoje įvedus „`0'; DROP TABLE Users --`“, programa sugeneruotų tokią užklausą:

```
SELECT UserID FROM Users WHERE Username='0'; DROP TABLE Users --' AND password = ''
```

DBVS interpretuos tokią užklausą kaip dvi atskiras užklausas, atskirtas kabliataškiu ir vykdys abi. Antrosios užklausos vietoje gali būti ir kito tipo SQL sakinys, pvz. „INSERT“, įterpiantis naują vartotoją į lentelę. Tačiau pademonstruoto pavyzdžio atveju būtų ištrinti visi vartotojų duomenys, kas galėtų sukelti katastrofiškas pasekmes.

Atkreipiame dėmesį į tai, jog kai kuriose DBVS SQL komandų atskyrimui nėra naudojami apskritai jokie skyrikliai, todėl vien skyriklio ieškojimas nėra efektyvus apsisaugojimo būdas nuo šio tipo atakų.

Blogai suformuotos užklausos

Prišlietos ir „UNION“ užklausos įsibrovėliams leidžia atlikti įvairias komandas, tačiau dažnai tai būna sunku, nežinant duomenų bazės schemos. Tai išsiaiškinti padeda specialiai tam tikslui blogai suformuotos užklausos. Šio metodo esmė - dažnai programų kūrėjų leidžiami parodyti pernelyg detalūs klaidų pranešimai, suteikiantys informacijos apie schemą. Programų kūrėjai turėtų pasirūpinti, kad visa su programos derinimu susijusi informacija būtų registruojama. Vartotojams turi būti parodoma tik jiems skirta ir ne pernelyg detali informacija. Įsibrovėliai specialiai į užklausas įterpia „šiukšlių“, sukeliančių užklausos sintaksės, logines ar duomenų tipų naudojimo klaidas. Pavyzdžiui, jei vartotojo vardo laukelyje įrašytume tokį tekstą: „`0' AND password = convert(int, (select top 1 name from sysobjects where xtype='u')) --`“. Programa sugeneruotų tokią užklausą:

```
SELECT UserID FROM Users WHERE Username='0' AND password = Convert(int, (SELECT TOP 1 NAME
FROM sysobjects WHERE xtype='u')) --' AND password = ''
```

Lentelėje „`sysobjects`“ yra saugomi duomenų bazės meta-duomenys, t.y. informacija apie bazėje esančius objektus. Įterptoji užklausa paima pirmosios vartotojų lentelės (filtras „`xtype='u'`“) pavadinimą iš meta-duomenų lentelės. Tuomet bando konvertuoti lentelės vardą į `int` tipą. Kadangi lentelės vardas yra „`string`“ tipo, konvertavimas yra negalimas ir duomenų bazė grąžina klaidą. Jei klaida nėra slepiama nuo vartotojo, įsibrovėlis gali pamatyti pranešimą, pavyzdžiui, *„Microsoft OLE DB Provider for SQL Server (0x80040E07) Error converting nvarchar value, CreditCards' to a column of data type int.“*

Iš šio pranešimo įsibrovėlis gali sužinoti, jog:

1) naudojama DBVS – Microsoft SQL Server;

2) duomenų bazėje yra lentelė, pavadinimu „`CreditCards`“.

Naudojant šį metodą įsibrovėlis labai panašiai gali išsiaiškinti kiekvieno lentelėje esančio laukelio pavadinimą ir duomenų tipą. Žinodamas šią informaciją apie duomenų bazės (toliau – DB) schemą, jis gali žymiai tiksliau parinkti tolesnių atakų metodus. Blogai suformuotos užklausos yra kaip įvadinis įsilaužimo žingsnis, skirtas surinkti informaciją apie DB.

Nuspėjimas

Metodas labai panašus, kaip ir blogai suformuotos užklausos. Skirtumas tik tas, jog čia išvadas apie DB leidžia daryti skirtingas DBVS elgesys skirtingų užklausų metu (šis elgesys gali būti vizualiai ir nematomas, pvz. nebūti jokių klaidų pranešimų). Tam tikslui sukuriamos SQLĮA, kurios priverčia užklausą elgtis skirtingai, priklausomai nuo jos rezultatų. Vienas iš tokių užklausų tipų yra „*laiko atakos*“. Jų pagalba informacija yra surenkama atsižvelgiant į DBVS vėlinimą. Užklausos yra formuojamos sąlygos sakinių „`IF-THEN`“ principu, vienoje iš vykdymo šakų įterpiant komandą „`WAITFOR`“. Jei ši komanda bus vykdoma, ji sukels numatytą DBVS vėlinimą ir įsibrovėlis žinos kuri iš sąlygos sakinio šakų buvo vykdoma. Pavyzdžiui, jei į vartotojo vardo laukelį įterptume tokį teksto fragmentą: „`vartotojas' AND ASCII(SUBSTRING((SELECT TOP 1 NAME FROM sysobjects),1,1)) > X WAITFOR 10 -- `“, programa suformuotų šią užklausą:

```
SELECT UserID FROM Users WHERE Username='vartotojas' AND ASCII(SUBSTRING((SELECT TOP 1
NAME FROM sysobjects),1,1)) > X WAITFOR 10 --' AND password = ''
```

Šios užklausos pagalba mėginame paimti pirmosios duomenų bazės lentelės vardo pirmąjį simbolį. Jį konvertuojame į *ASCII* kodą ir palyginame su *X*. Jei reikšmė yra didesnė, įsibrovėlis pastebės, jog DBVS vėlina 10 sekundžių. Tokiu būdu, naudojant binarinės paieškos strategiją galime išsiaiškinti visų lentelių visų laukelių vardus. Kitas nuspėjimo atakų tipas yra „*aklasis*“ SQLĮ, aprašomas [3].

Alternatyvus kodavimas

Daugelis SQLĮA remiasi specialiųjų simbolių (tokių kaip kabutės, taškai, brūkšniai, kabliataškiai) naudojimu. Tačiau dažniausiai programose būna jau suprogramuota apsauga nuo tokio tipo simbolių įterpimo į SQL užklausas. Alternatyvus kodavimas gali pagelbėti tai apeiti. Įvairių tipų kodavimas (ASCII, šešioliktainis ar „Unicode“) kartu su kitais SQLĮ metodais gali padėti apeiti tokias apsaugas, kurios paprasčiausiai ieško „blogųjų simbolių“ paduodamame tekste. Programuotojai galbūt ir galėtų ieškoti „blogųjų simbolių“, užkoduotų visais variantais, tačiau tokiu būdu apsisaugoti vis vien labai sunku, kadangi atakos gali paliesti įvairius programų sistemos sluoksnius. Pavyzdžiui, iš pradžių atrodanti nelogiška

simbolių eilutė, naudojantis duomenų bazėje esančiomis funkcijomis gali būti paversta į reikiamą simbolį. Vartotojo vardo laukelyje įterpsime „`0'; exec(char(0x73687574646f776e)) --`“. Tokiu būdu bus suformuota ši užklausa:

```
SELECT UserID FROM Users WHERE Username='0'; exec(char(0x73687574646f776e)) --' AND
password = ''
```

Ši ataka naudojasi funkcija „`char()`“ ir ASCII kodavimu tam, kad suformuotų komandą „`shutdown`“.

Pasinaudojimas duomenų bazės procedūromis

Viena iš intensyviai skelbiamų idėjų – naudoti DB procedūras (angl. stored procedures) SQLĮA problemos sprendimui. Iš tiesų, jos leidžia programų kūrėjams naudotis šiek tiek aukštesniu abstrakcijos lygmeniu, į procedūras įtraukti verslo logikos elementų. Tačiau klaidinga manyti, jog galime išvengti SQLĮA problemos, naudodami vien DB procedūras. Procedūrų saugumas (beje, kaip ir bet kurios kitos programinės įrangos (toliau – PĮ) dalies) priklauso nuo to, kaip jos buvo programuojamos ir ar buvo laikomasi atitinkamų saugumo standartų. Parametrizuotos DB procedūros gali būti lengvai pažeidžiamos SQLĮA, kaip ir bet kuris kitas programos kodas. [6] darbe bandoma gvildenti ši problema ir pastebima, jog procedūrų saugumui SQLĮA srityje skiriama pernelyg mažai dėmesio, o kai kurioje literatūroje DB procedūros klaidingai laikomos netgi SQLĮA panacėja.

2.2.3. Esantys problemos sprendimo būdai

Šiame skyrelyje apžvelgsime jau esančius šios problemos (SQLĮA) sprendimus, jų tipus, o taip pat ir tyrėjų darbus šioje srityje. Trumpai aprašysime žinomų problemos sprendimo metodų esmę, įvardindami jų privalumus ir trūkumus.

Apsaugojimas nuo SQLĮA palaikančios struktūros (angl. framework) lygmenyje

Keletas palaikančių struktūrų kūrėjų (tokių kaip Microsoft, Java, The Apache Software Foundation) siūlo pagalbą, sprendžiant šią problemą. Pavyzdžiui, Apache Software Foundation palaikanti struktūra *Struts* [10] turi *validatorių*. Jis tikrina ar vartotojo įvedami duomenys atitinka iš anksto nustatytą formatą. Jei *validatorių* sukonfigūruojame taip, kad jis neleistų įvesti meta-simbolių, galėtume apsisaugoti nuo SQLĮA. Tačiau jei mes privalome leisti įvesti meta-simbolius, *validatorius* nepadės.

Duomenų žymėjimas ir atranka (angl. tainted data-tracking/taint-based technique)

Šio metodo esmė – pažymėti duomenis ir sekti iš kur jie ateina - iš programos ar iš vartotojo įvedimo. Kai suformuojama SQL užklausa, pirmiausia analizuojamas jos sintaksės medis. Jei kuris nors raktinis žodis ateina iš vartotojo įvestų parametrų, SQL užklausa stabdoma [11, 12]. Yra sukurta įvairių metodikų, žyminčią duomenis tiek tekstinių eilučių, tiek pavienių simbolių lygmenyje. Yra idėjų žymėti įvedamus duomenis netgi bitų lygmenyje, taip apsaugant nuo galimų bitais manipuliuojančių operacijų, formuojant SQL užklausas.

Statinė analizė

SQL vykdymo taškuose (angl. hotspot) tikrinama ar SQL užklausa atitinka iš anksto nustatytą reguliariąją išraišką [13]. Jei vykdymo metu aptinkama, jog SQL užklausa nepriklauso nei vienai reguliariai išraiškai, ji atmetama. Yra ir kitų metodų, besiremiančių statine analize. Pavyzdžiui G.Wassermann ir Z.Su [14] pasiūlė metodą, naudojantį statinę analizę ir automatinį sprendimų priėmimą. Šis metodas tikrina ar sugeneruotos SQL užklausos neturi tautologijų.

Dinaminė analizė arba testavimas juodosios dėžės principu

Šio metodo esmę sudaro tinklapio testavimas juodos dėžės principu, naudojantis iš anksto nustatytais pradiniais duomenimis ir atakų šablonais. Būdas panašus į „WAVES“ [8], tačiau „WAVES“ autoriai į savo sprendimą dar įtraukė ir besimokančių sistemų elementų. Šis metodas greitas ir efektyvus, tačiau neįtraukiant žinių apie išeities tekstus neįmanoma aptikti sudėtingesnių SQLĮA variantų.

Kombinuota statinė ir dinaminė analizė

AMNESIA (angl. **A**nalysis and **M**onitoring for **NE**utralizing **S**QL-**I**njection **A**ttacks) [15] yra vienas iš metodų, kombinuojančių statinę ir dinaminę analizę. Statinės analizės fazėje sukuriami SQL užklausų modeliai. Modeliai sukuriami peržiūrint visus sąveikos su DB taškus. Dinaminėje fazėje sistema perima jau paruoštas vykdyti užklausas ir patikrina ar jos atitinka anksčiau suformuotus modelius. Užklausos, neatitinkančios modelių yra traktuojamos kaip SQLĮA. AMNESIA sistemos tikslumas priklauso nuo statinės analizės. Jei užklausų generavimui naudojami tam tikri obfuskavimo (angl. obfuscation) kodai ir/ar sudėtingesni generavimo metodai, statinės analizės žingsnis tampa ne toks tikslus. Kitas sprendimas – SQLGuard [16]. Jis programos vykdymo metu tikrina ar užklausos atitinka nustatytą modelį. Modeliai čia ne užrašomi iš anksto, kai vien statine analize grindžiamuose sprendimuose. Čia modeliai išvedami pačios analizuojančios sistemos. Ji išveda modelius lygindama užklausų struktūras prieš ir po vykdymo. Vartotojai, norėdami pasinaudoti SQLGuard turi perrašyti kodą.

SQL instrukcijų atranka (angl. randomization), pažymint atsitiktiniais simboliais

Šio metodo esmę sudaro įrankis, leidžiantis programuotojams kurti modifikuotas užklausas, kurios prie kiekvieno SQL raktinio žodžio prideda atsitiktinę slaptą simbolių seką. SQL užklausų procesorius taip pat modifikuojamas taip, kad galėtų suprasti ir atskirti šiuos atsitiktinius simbolius. Užklausos vykdymo metu, užklausų procesorius supranta tik tuos raktinius žodžius, kurie turi pridėtus slaptus simbolius. Jei įsibrovėlis mėgins įterpti raktinių žodžių – jie neturės pridėtų slaptų simbolių, o tai sukels sintaksės klaidas [9], vadinasi, metodo saugumas paremtas slapta simbolių seka, ir jei įsibrovėlis gali ją sužinoti, tai metodas netenka prasmės.

Išankstinis SQL sakinio paruošimas

Metodo esmė yra ta, jog atskiriamas pats SQL sakinys ir jo struktūra nuo jam paduodamų parametrų. Parametrai paduodami vėliau, o kaip juos įterpti, nurodo SQL sakinio struktūroje aprašyti parametrų vietos žymekliai (angl. placeholders). Tokį metodą rekomenduoja ir Microsoft, knygoje [4], tiek .NET kodo, tiek ir duomenų bazės procedūrų apsaugojimui. Saugumą apsprendžia tai, jog sakinio struktūra lieka nepakitusi nepriklausomai nuo to, kokius parametrus vartotojas įveda, nes jie bus vis vien traktuojami kaip parametrai, o ne raktiniai žodžiai.

Kitose aplinkose, pvz. *Hibernate*, naudojamas „`prepare`“ sakinys [17], atliekantis sakinio ir jo parametro atskyrimo darbą. Metodo trūkumas yra tas, jog norint apsaugoti senąsias programas, reikia perrašyti kodą.

Šiame darbe remsimės šiuo metodu.

Dirbtiniu intelektu paremtas metodas

Iš esmės, dirbtiniu intelektu paremtas metodas yra ta pats „*Dinaminės analizės, arba testavimo juodosios dėžės principu“* grindžiamas metodas. Skirtumas tik tas, jog dirbtiniu intelektu paremtos sistemos geba tobulinti pačios save.

F.Valeur ir kiti [7] pasiūlė įsilaužimų fiksavimo sistemą (angl. Intrusion Detection System), paremtą besimokančių sistemų metodais. Sistema apmokoma naudojant standartinių užklausų aibę. Jų pagrindu sukuriamas modelis ir sistemos veikimo metu tikrinama, ar visos užklausos atitinka modelį. Visos sistemos kokybė tiesiogiai proporcinga apmokymo duomenų aibės kokybei. Apmokius sistemą netinkamais ar prastais pradiniais duomenimis ji gali pernelyg dažnai neteisingai atmesti (angl. false positive) arba neteisingai priimti (angl. false negative) užklausas. Neteisingas atmetimas reiškia, jog sistema aptinka saugią užklausą, bet ją atmeta. Neteisingas priėmimas priešingai – reiškia nesaugios užklausos palaikymą saugia. Pastarasis variantas kur kas pavojingesnis, kadangi esant neteisingiems priėmimams paliekamos neaptiktos saugumo spragos.

Kita dirbtiniu intelektu paremta sistema – „WAVES“ [8]. „WAVES“ yra internetinis robotas (angl. web-crawler), aptinkantis pažeidžiamas vietas ir formuojantis atakas toms pažeidžiamoms vietoms pagal iš anksto žinomą šablonų ir atakų metodikų rinkinį. „WAVES“ stebi programos atsaką į poveikį ir gerina atakavimą, naudojantis besimokančių sistemų metodais. „WAVES“ pranoksta tradicinius įsilaužimo testavimo metodus, kadangi stengiasi gerinti atakavimo metodologiją, tačiau ji vis vien negali ištestuoti taip nuodugniai, kaip tai būtų galima padaryti naudojant tradicinius metodus.

3. PROJEKTINĖ DALIS

3.1. Sistemos architektūros pateikimas

Projekto dokumentacija buvo sudaryta pagal RUP (angl. Rational Unified Process) architektūros specifikavimo šabloną.

Didžiosios architektūros dalies specifikavimui buvo naudojama UML 2.0 notacija. Aukštesnio abstrakcijos lygio atvaizdams (tokioms diagramoms kaip išdėstymo, posistemių) dar buvo naudojami ir „Microsoft Visual Studio Team System“ architektūros specifikavimo ir diagramų braižymo įrankiai.

Dabar šiek tiek informacijos apie šiame dokumente naudojamus modelius.

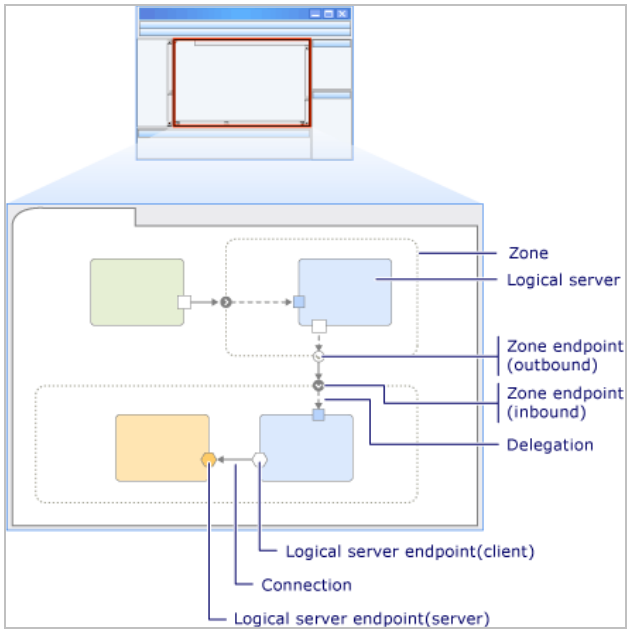
1997 OMG (angl. Object Management Group) išleido UML (angl. Unified Modeling Language). Kadangi iki tol egzistavo galybė modeliavimo notacijų, vienas iš UML sukūrimo tikslų buvo suteikti programų kūrimo bendrijai stabilią ir vieningą modeliavimo kalbą ir notaciją. UML yra būtent *kalba,* bet ne *metodologija*, kas leidžią ją pritaikyti įvairiausiems įmonių poreikiams. Nors UML sudaro ne tik diagramos, tačiau jos sudaro pagrindą. Čia paminėsiu pagrindines diagramas, kurios buvo naudojamos specifikuojant šiame dokumente aprašomos sistemos architektūrą: panaudojimo atvejų, klasių, sekų, būsenų ir veiklos diagramos. Be UML diagramų, projekto architektūrai buvo naudojamos ir kelio specifinės diagramos.

Kartu su „Visual Studio Team System“ Microsoft išleido ir paketą, palengvinantį paskirstytų sistemų architektūros kūrimą ir atnaujinimą. Naudojant paskirstytųjų sistemų dizainerį galima atlikti tokius aukšto abstrakcijos lygmens darbus:

- Projektuoti, peržiūrėti, konfigūruoti ir tarpusavyje jungti programų sistemas (dažniausiai atlieka programų sistemų architektas).
- Kurti loginį duomenų centrų vaizdą, kuriuose planuojama diegti kuriamas sistemas (kuria infrastruktūros architektai).
- Įvertinti kaip bus talpinamos programų sistemos fizinėje infrastruktūroje (atlieka programų sistemų architektas).
- Įgyvendinti (sugeneruoti) realias programų sistemas, naudojantis architektūriniu aprašu (programuotojai).

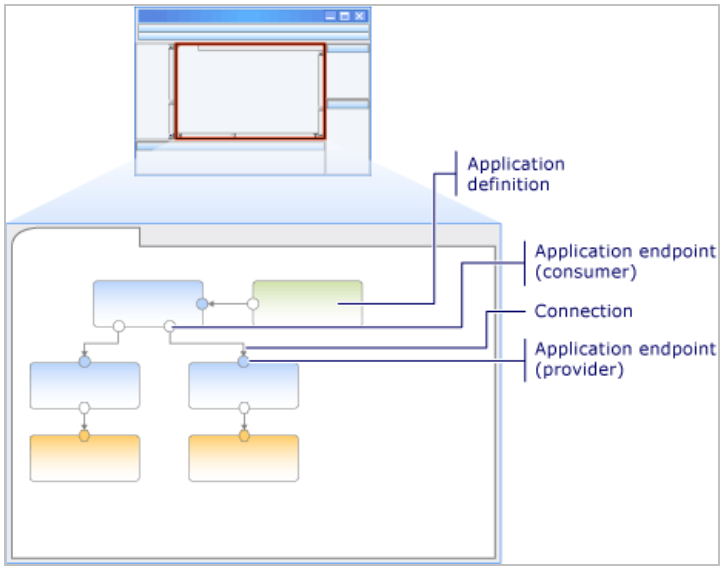
Šiame dokumente naudosime tokius diagramų tipus:

- Loginį duomenų centro vaizdą (angl. **Logical Datacenter Diagram**);



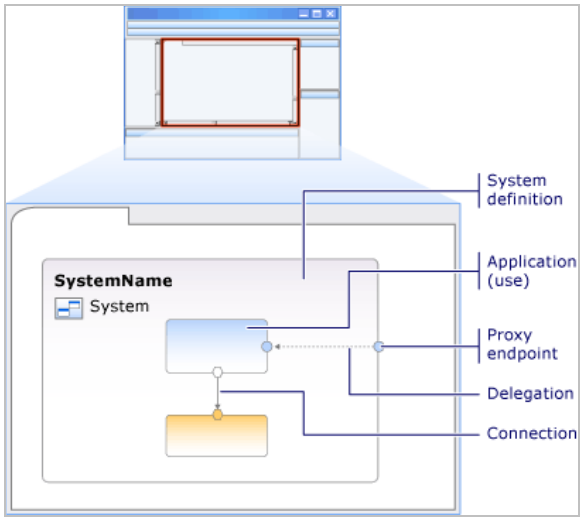
Pav. 7 - Loginio duomenų centro diagramos šablonas

- Taikomųjų programų (angl. **Application Diagram**) diagrama:



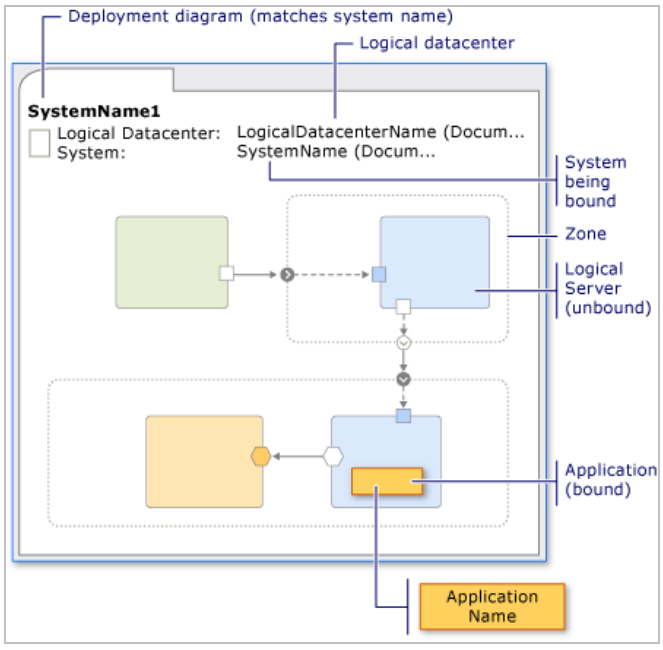
Pav. 8 - Taikomųjų programų diagramos šablonas

- Sistemų (angl. **System Diagram**) diagramos:



Pav. 9 - Sistemos diagramos šablonas

- Išdėstymo diagrama (angl. **Deployment Diagram**):



Pav. 10 - Išdėstymo diagramos šablonas

3.2. Architektūros tikslai ir apribojimai

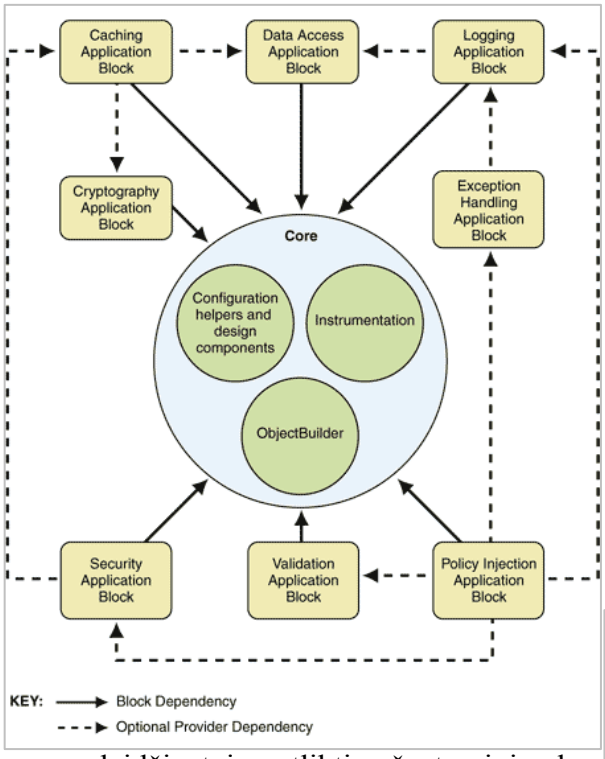
Iš anksto buvo žinoma, jog kuriant sistemą bus panaudoti trečios šalies GUI (angl. Graphical User Interface) - COTS (angl. Commercial Off-the-Shelf) komponentai. Sistema sąsajos lygmenyje naudos būtent įmonės „Infragistics NET Advantage“ GUI komponentų biblioteką, todėl jau kuriant architektūrą buvo į tai atsižvelgta.

Kadangi būsimos sistemos pagrindiniai vartotojai – farmacinių atstovybių atstovai dažnai keliauja pas savo klientus ir labai dažnai pasitaiko, jog tuo metu, kai reikalinga sąveika su sistema (reikia raportuoti vizitą, sužinoti informaciją apie klientą) nėra interneto ryšio, buvo nuspręsta realizuoti sistemą taip, jog ji palaikytų du režimus: tiesioginį (angl. online) ir atjungtąjį (angl. off-line). Tai padarė labai didelę įtaką projektuojamos sistemos architektūrai.

Didelių reikalavimų portabilumui nėra, tačiau visgi, norint pagerinti šią programų sistemos nefunkcinę charakteristiką bus svarstoma apie sistemos web ir PDA kliento versijas. Šiame darbe apsiribojome tik Windows kliento sukūrimu, tačiau architektūra buvo sukurta taip, jog palaikytų keletą galimų sąsajų tipų ateityje: įvairius Windows klientus, PDA (angl. Portable Digital Assistant) ir web klientus. Tai įtakojo pasirinkti SOA (angl. Service-Oriented Architecture) architektūrą ir paslaugas teikti per SOAP (angl. SOAP) protokolą, kas leistų lengvai prijungti įvairaus tipo ir platformos klientus, įgalintų pakartotinį servisų panaudojimą.

Norint sumažinti sistemos kūrimo kaštus, kaip rekomendacija buvo kaip galima efektyviau panaudoti dabar užsakovo įmonėje jau sukurtų sistemų pakartotinai panaudojamas dalis, o taip pat ir atvirosios licensijos trečiųjų šalių pakartotinio panaudojimo komponentus. Pagrindiniai tokie komponentai, verti paminėjimo yra užsakovo įmonės klasių biblioteka, skirta sąveikai su duomenimis palengvinti (yra pakete „Softdent.Data.dll“), saugumo klasių biblioteka, esanti pakete „Softdent.Security.dll“. Kita labai svarbi ir verta paminėjimo yra „Microsoft Patterns and Practices“ grupės palaikomas paketas „Enterprise Library 3.0“, susidedantis iš keleto naudingų ir labai dažnai programų sistemose naudojamų programų blokų (angl. Application Block) [28]:

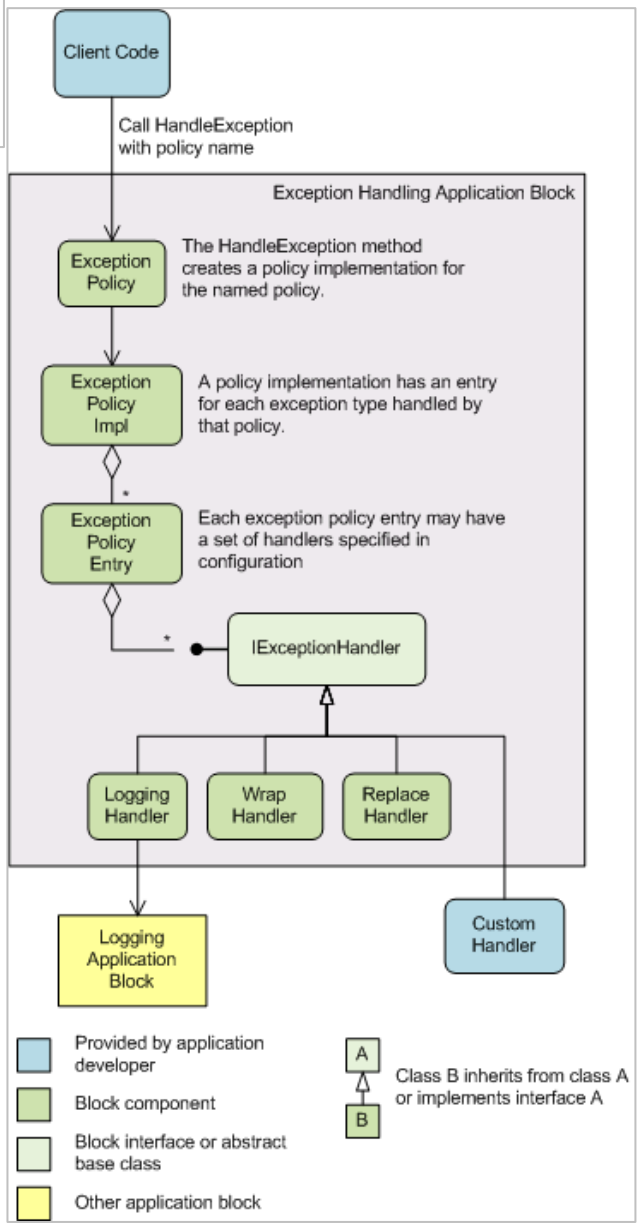
- Saugumo blokas;
- Įvykių registravimo (angl. Event Logging);
- Išimčių apdorojimo;
- Keršavimo;
- Duomenų prieigos;
- Kriptografijos.



Pav. 11 - Microsoft Enterprise Library architektūra ir programinių blokų tarpusavio priklausomybės

Šios sistemos architektūrai pasitarnaus išimčių apdorojimo ir įvykių registravimo blokai. Išimčių apdorojimo blokas (angl. Exception Handling Application Block) yra sukurtas pasiekti šiems tikslams:

- Apimti dažniausiai pasitaikančią išimčių apdorojimo logiką ir užduotis, leidžiant jas atlikti rašant minimalų kodo kiekį:
- Išlaisvinti programuotojus nuo poreikio kaskart rašyti tą patį ar panašų kodą skirtingiems projektams, kuriuose iš esmės atliekamos tos pačios išimčių apdorojimo užduotys.
- Leisti lengvai keisti ir konfigūruoti išimčių apdorojimo politiką, kuomet ji jau pritaikyta ir suprogramuota, užtikrinant, jog pasikeitimai būtų vienalaikiai ir suderinami.
- Įtraukti geriausią ir profesionalų siūlomą išimčių apdorojimo mechanizmą, aprašomą [27].



Pav. 12 - Microsoft išimčių apdorojimo programinio bloko architektūra

Įvykių registravimo blokas glaudžiai siejasi su išimčių apdorojimo bloku, kadangi Enterprise Library architektūroje išimtis yra tarsi įvykis, kuris kažkuriuo tai norimu būdu gali būti registruojamas. Na, o šios sistemos atveju saugumo, audito ir kitais sumetimais neapdorotų išimčių registravimas yra būtinas. Naudojant Enterprise Library įvykių registravimo programinį bloką supaprastėja įprastų įvykių registravimo funkcijų realizavimas. Programuotojai gali naudoti programinį bloką norėdami fiksuoti informaciją įvairiausiose informacijos saugojimo vietose:

- Operacinės sistemos įvykių registre (angl. Event Log)
- Išsiųsti el.pašto žinute
- Duomenų bazėje
- Žinučių eilėje
- Tekstiniame faile
- WMI (angl. Windows Management Instrumentation) įvykyje
- Savo suprogramuotoje saugykloje

Kaip matome programinis blokas iš tiesų efektyvus, kadangi galima saugoti auditui reikalingą informaciją įvairiausiose, praktiškai bet kokiose vietose. Pačioje programoje nėra nurodoma saugojimo vieta. Tai padaroma konfigūracijos failo pagalba. Tai reiškia, jog sistemos operatoriai, o taip pat ir programuotojai gali keisti įvykių registravimo elgesį nekeičiant pačios programos kodo.

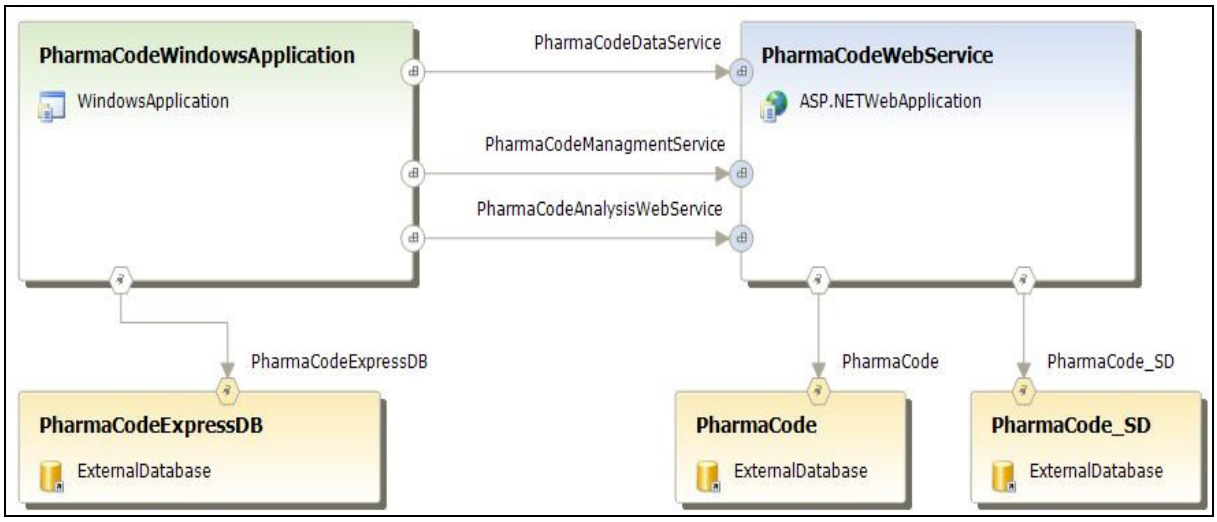
Nors „Patterns & Practices" [28] grupė siūlo ir kitus architektūrinius sprendimus, pvz. saugumui ir duomenų prieigai, tačiau čia bus naudojami jau anksčiau paminėti užsakovo įmonės komponentai, kurie sukurti atsižvelgiant į tos pačios „Patterns and practices" grupės rekomendacijas.

Iš anksto nustatyti ir projektavimo įrankiai:

Kaip pagrindinis architektūros modeliavimo įrankis buvo naudojamas „No Magic Inc. MagicDraw UML".

Kai kurioms iš diagramų braižyti buvo naudojama ir „Microsoft Visual Studio Team Editon".

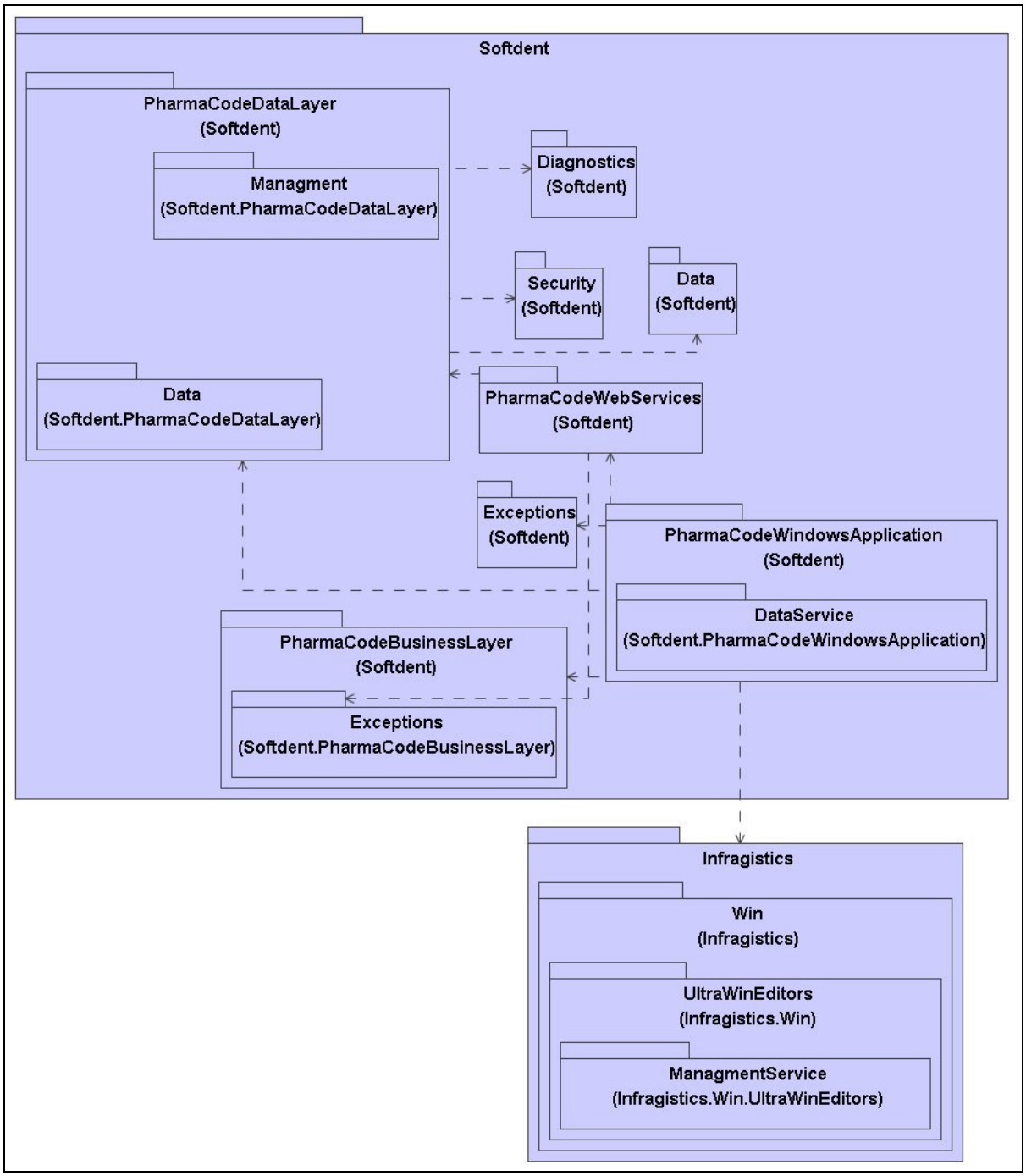
3.3. Sistemos statinis vaizdas



Pav. 13 - Sistemos taikomųjų programų apjungimo schema

Aukščiau pateiktame paveikslėlyje pavaizduota programų sistemos apjungimo diagrama aukščiausiame abstrakcijos lygmenyje, išskiriant tik fizines taikomąsias programas, o jų pačių nedetalizuojant. Iš paveikslėlio matome, jog kliento taikomoji programa jungiasi prie „PharmaCodeWebService“ programos publikuojamų servisų: „PharmaCodeDataService“ (skirtas duomenų sinchronizavimui, „PharmaCodeManagementService“ (skirtas išimčių apdorojimui) ir „PharmaCodeAnalysisWebService“ (skirtas analizuoti veiklos rezultatus, generuoti ataskaitas ir pan.). „PharmaCodeWindowsApplication“ taip pat jungiasi prie lokalios, duomenų bazės, skirtos palaikyti „Off-line“ darbo režimą.

Pats servisas savo ruožtu jungiasi prie centrinės sistemos klientų duomenų bazės „PharmaCode“, tuomet, išsiaiškinus kliento duomenų bazės vietą, jungiasi prie kliento duomenų bazės ir vykdo komandas. Paveikslėlyje pavaizduota tik vieno kliento duomenų bazė, pavadinimu „PharmaCode_SD“, tačiau šią schemą turinčių duomenų bazių bus tiek, kiek bus sistemos klienčių-įmonių.



Pav. 14 - "PharmaCode" programų sistemos paketų diagrama

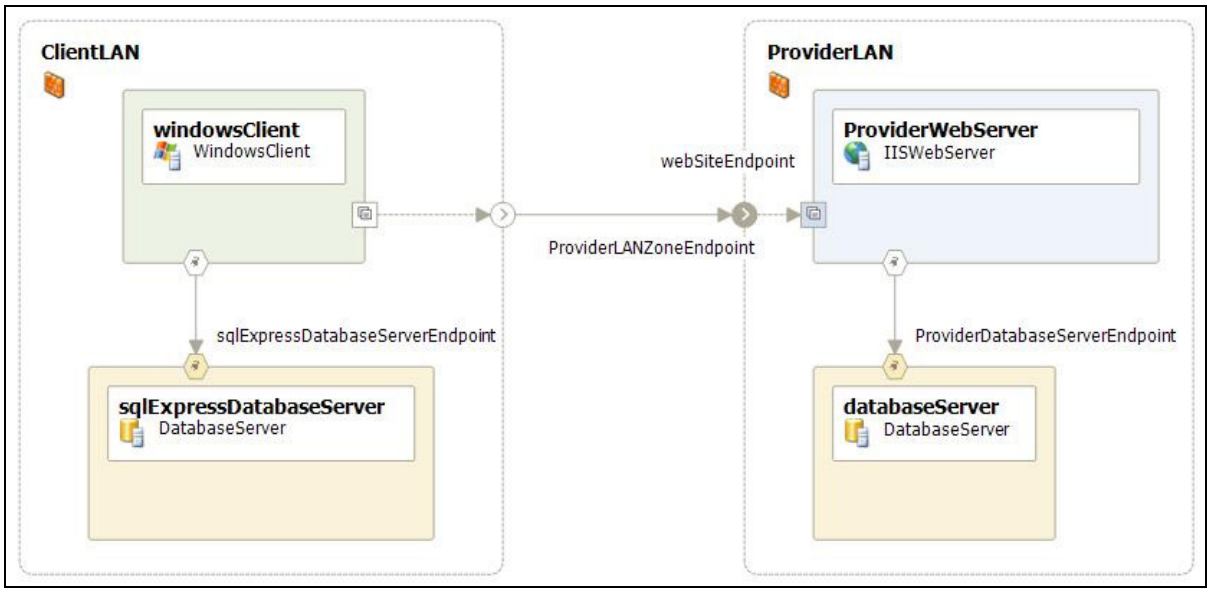
Paketų detalizacija ir detalus aprašas pateikaimas projekto dokumentacijoje, todėl šiame dokumente jų nedetalizuosime.

Infragistics.Win

Kadangi šis paketas yra COTS (angl. Commertial off-The Shelf) produktas, skirtas pagerinti GUI, jo nedetalizuosime.

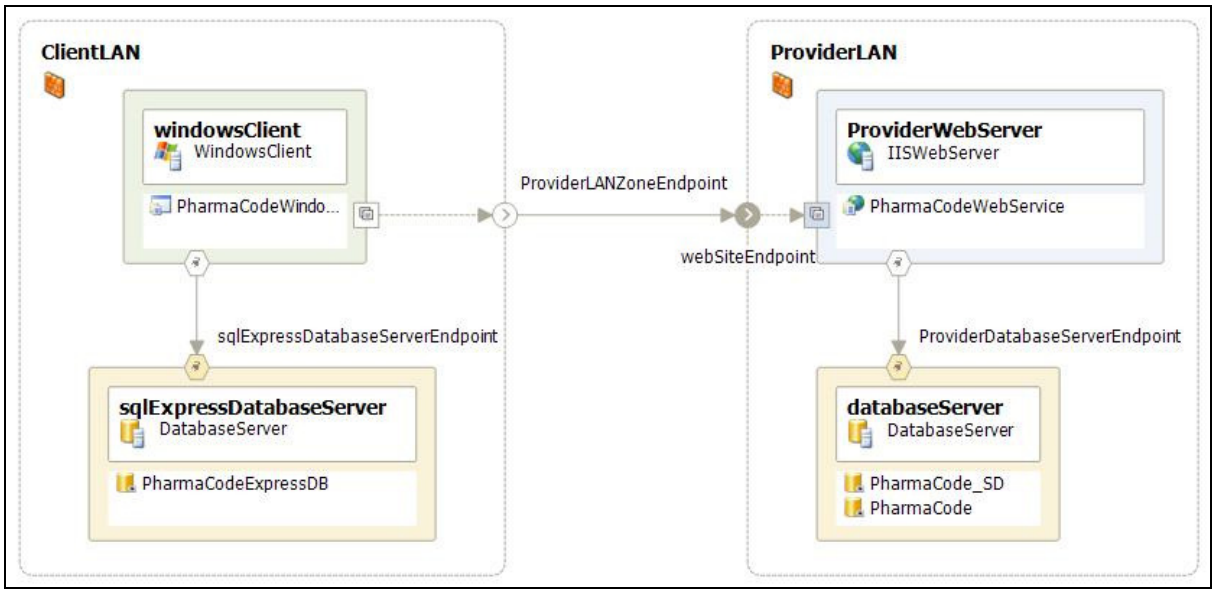
Išdėstymo (angl. „Deployment“) vaizdas

Sistemą fiziškai išdėstyti planuojama dviejų tipų konfigūracijose: standartine konfigūracija ir lokalia konfigūracija. Standartinės konfigūracijos atveju duomenų bazės serveris ir aptarnaujantys servisai bus nutolusiame, paslaugos teikėjo tinkle.



Pav. 15 - Standartinės konfigūracijos fizinė aplinka

Standartinės konfigūracijos atveju, aptarnaujantys servisai ir centrinės duomenų bazės bus laikomos paslaugos tiekėjo tinkle (paveikslėlyje pažymėta „ProviderLAN“). Kliento pusėje (paveikslėlyje regionas pažymėtas „ClientLAN“) turės būti įdiegta MS Windows operacinė sistema, taip pat joje turės būti įdiegtas, arba „PharmaCodeWindowsApplication“ diegimo metu įdiegiamas „Microsoft Framework 2.0” ir „Microsoft SQL Server Express“. To pilnai užtenka norint paruošti darbo vietą kliento pusėje. Paslaugos tiekėjo pusėje reikės dviejų serverių: „Internet Information Server“ ir „Microsoft SQL Server Enterprise Edition“.

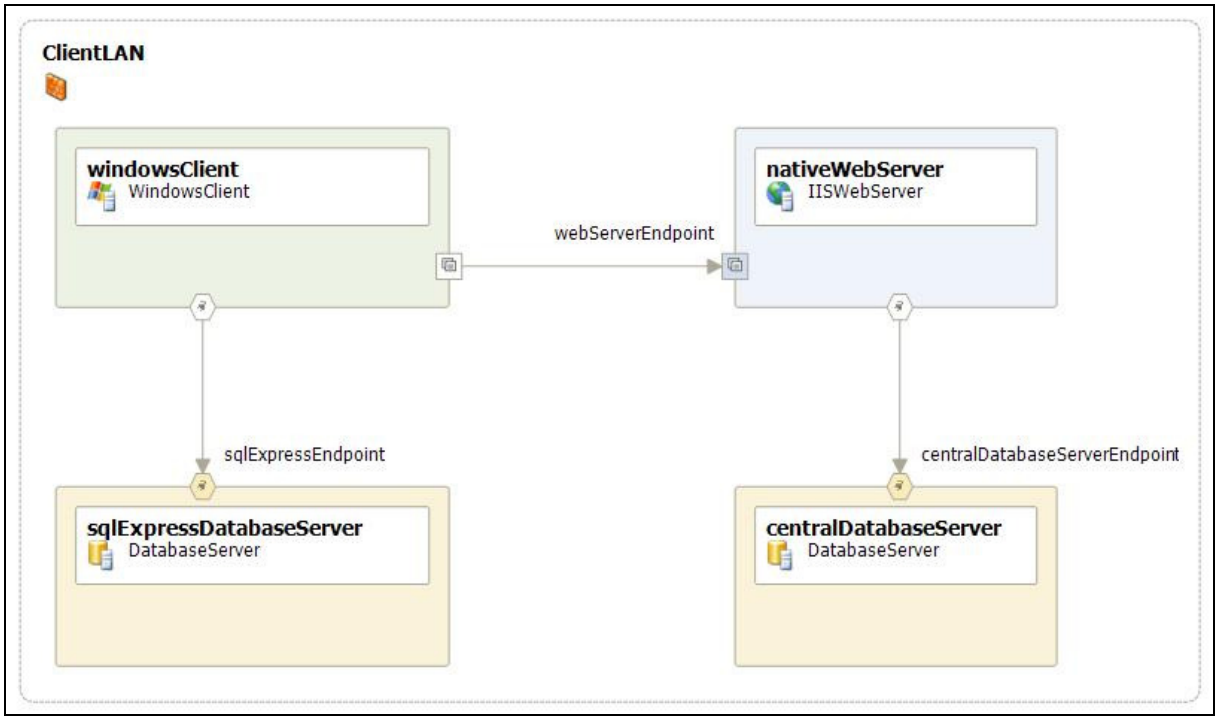


Pav. 16 - Taikomųjų programų išdėstymas standartinėje fizinėje aplinkoje

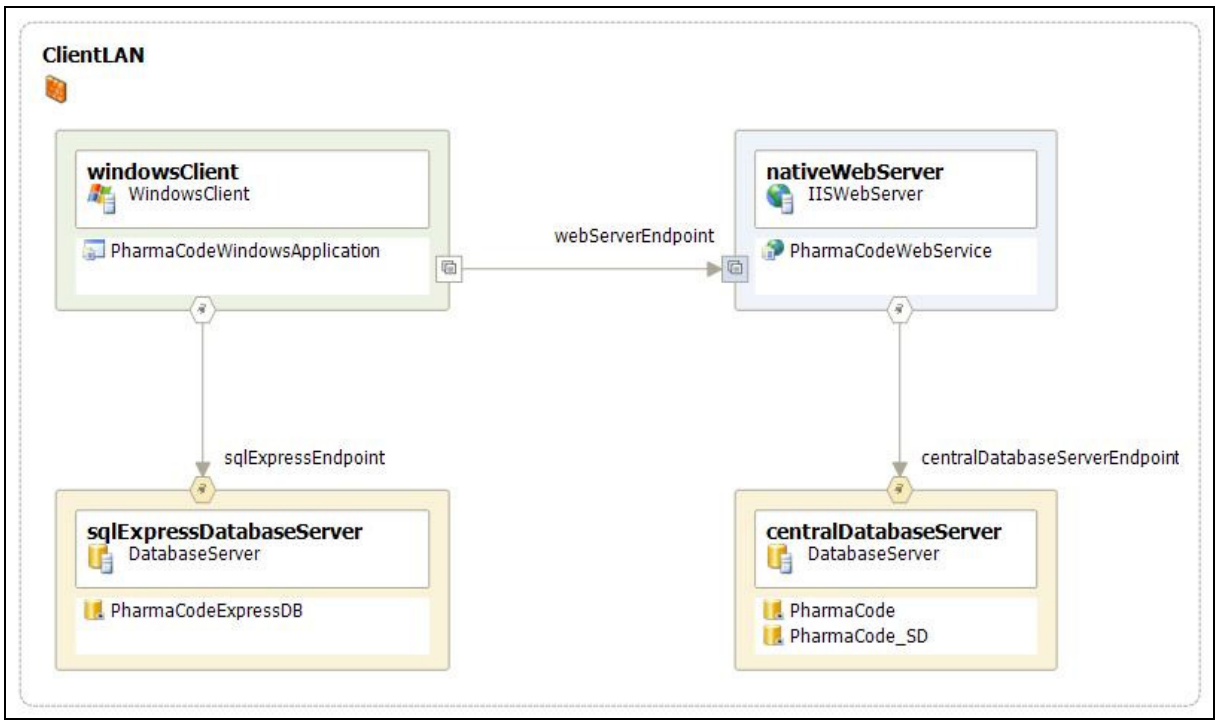
Kliento pusėje bus instaliuojama „PharmaCodeWindowsApplication“ taikomoji programa, o taip pat „PharmaCodeExpressDB. Paslaugos tiekėjo „Internet Information“ serveryje bus įdiegtas „PharmaCodeWebService“ servisų rinkinys, susidedantis iš „PharmaCodeDataService“ – duomenų transakcijoms, „PharmaCodeManagementService“ – administravimo užduotims ir „PharmaCodeAnalysisWebService“ – servisas skirtas veiklos analizei ir ataskaitų generavimui. Duomenų bazių serveryje bus centrinė paslaugos tiekėjo duomenų bazė – „PharmaCode“, ir po duomenų bazę kiekvienam klientui. Kaip pavyzdys parodyta duomenų bazė „PharmaCode_SD“.

Klientą ir paslaugos tiekėją jungia nesaugi terpė, todėl kanalas turės būti apsaugojant naudojant SSL arba kitą, galbūt aukštesniame lygmenyje įgyvendintą saugaus komunikavimo modelį.

Gali pasitaikyti atvejų kai, pavyzdžiui, dėl įmonės duomenų saugumo politikos, klientai negalės laikyti duomenų paslaugos tiekėjo tinkle. Tuomet bus siūloma duomenų bazių ir aptarnaujančių servisų serverius laikyti lokaliame tinkle.



Pav. 17 - Lokali kliento fizinė aplinka



Pav. 18 - Taikomųjų programų patalpinimas lokalioje fizinėje aplinkoje

Iš esmės ši konfigūracija skiriasi nuo standartinės tik tuo, kad visi serveriai ir programinė įranga bus instaliuojama kliento tinkle. Šiuo atveju reikės tiek pat fizinių elementų, o taip pat ir tiek pat loginių elementų (taikomųjų programų ir duombazių).

Architektūros kokybė

Pirmiausias ir turbūt vienas svarbiausių architektūros suteikiamų privalumų yra išplečiamumas. Kadangi serverio pusė paslaugas teiks SOAP protokolą, bus patogu praplėsti sistemą, pritaikant ją įvairių tipų ir platformų vartotojams. Pavyzdžiui, bus galima suprogramuoti web, PDA klientines programas, kitų (ne WIN) platformų klientus.

Nors reikalavimuose pernešamumui nebuvo nurodyti kažkokie tai aukšti reikalavimai, pernešamumas bus pagerintas dėka lankstaus sinchronizacijos mechanizmo. Vartotojai, perėję prie kito fizinio kompiuterio galės instaliuoti programą ir susinchronizuoti informaciją su serveriu, priklausomai nuo prisijungto vartotojo ID.

Patikimumui ir atstatomumui užtikrinti bus nuolat sekamas sistemos darbas, fiksuojami ir protokoluojami įvykiai įvykių registre (angl. event log) tiek kliento tiek serverio pusėse. Duomenų integralumui užtikrinti ten, kur įprastinių reliacinių DB priemonių nepakanka, bus naudojamos papildomos Microsoft SQL Server priemonės – trigeriai, stored procedūros, veikiančios transakcijų režime.

4. TYRIMO DALIS

Tiriamoji dalyje aprašysime programų sistemos kokybės tobulinimo galimybes. Pirmoje dalyje pateiksime projekto kokybės vertinimą, antroje – realizuoto produkto išeities tekstų statinės analizės rezultatus, trečiojoje – pasiūlysime ir pateiksime patobulintos statinės kodo analizės taisyklės realizaciją.

4.1. Projekto kokybės analizė

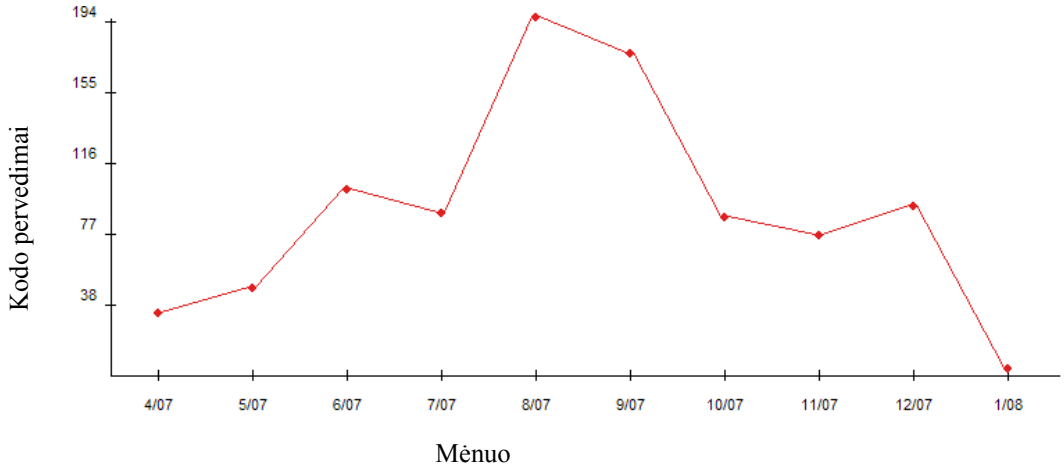
Projekto kodo saugyklos (SVN) statistika

Lentelė nr. 2 - Bendra kodo saugyklos informacija

Kodo gyvavimo laikas saugykloje (sav.):	38
Bendras kodo pervedimų į saugyklą skaičius:	898
Bendras failų keitimų skaičius:	7506

Lentelė nr. 3 - Programuotojų aktyvumas

	Vidutinis	**Maksimalus**
Kodo pervedimų kiekis/sav.	23	68
Failų pakeitimų skaičius/sav.	197	870



Pav. 19 - Kodo pervedimai pagal datą

Projekto defektų valdymo sistemos (BugZilla) statistika

Pav. 20 - Defektų skaičiaus kitimas laike

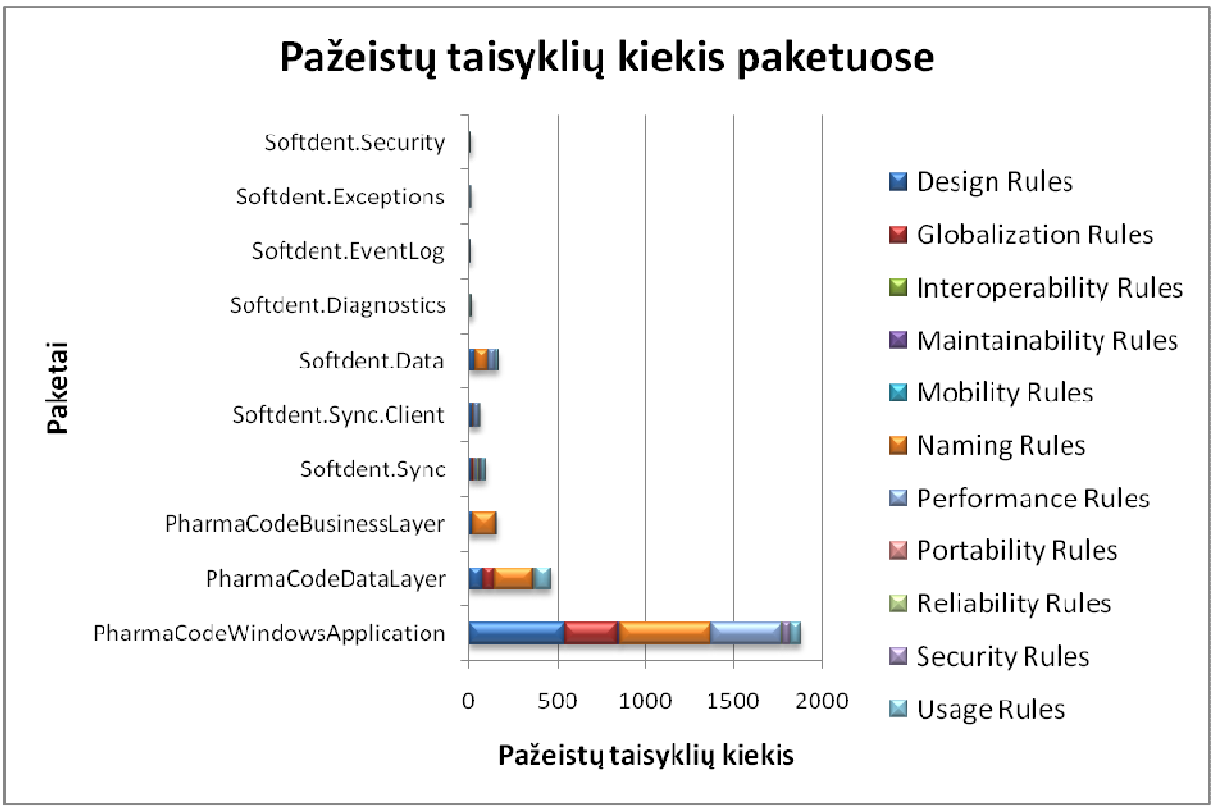
4.2. Statinė išeities tekstų analizė

Šioje dalyje aprašysime atliktą darbą, analizuojant realizuotos sistemos išeities tekstus.

Analizei buvo pasirinkti 10 pagrindinių sistemos paketų: PharmaCodeDataLayer, PharmaCodeWindowsApplication, PharmaCodeBusinessLayer, Softdent.Sync, Softdent.Sync.Client, Softdent.Data, Softdent.Diagnostics, Softdent.EventLog, Softdent.Exceptions, Softdent.Security. Šių paketų išeities tekstai yra analizės tyrimo objektai.

Tyrimą atlikome su *visomis* Microsoft statinės kodo analizės taisyklėmis, rekomenduojamomis ir aprašomomis gide „.NET Framework Design Guidelines for Class Library Developers“. Analizei naudojome į Visual Studio Team System integruotą įrankį „FxCOP“.

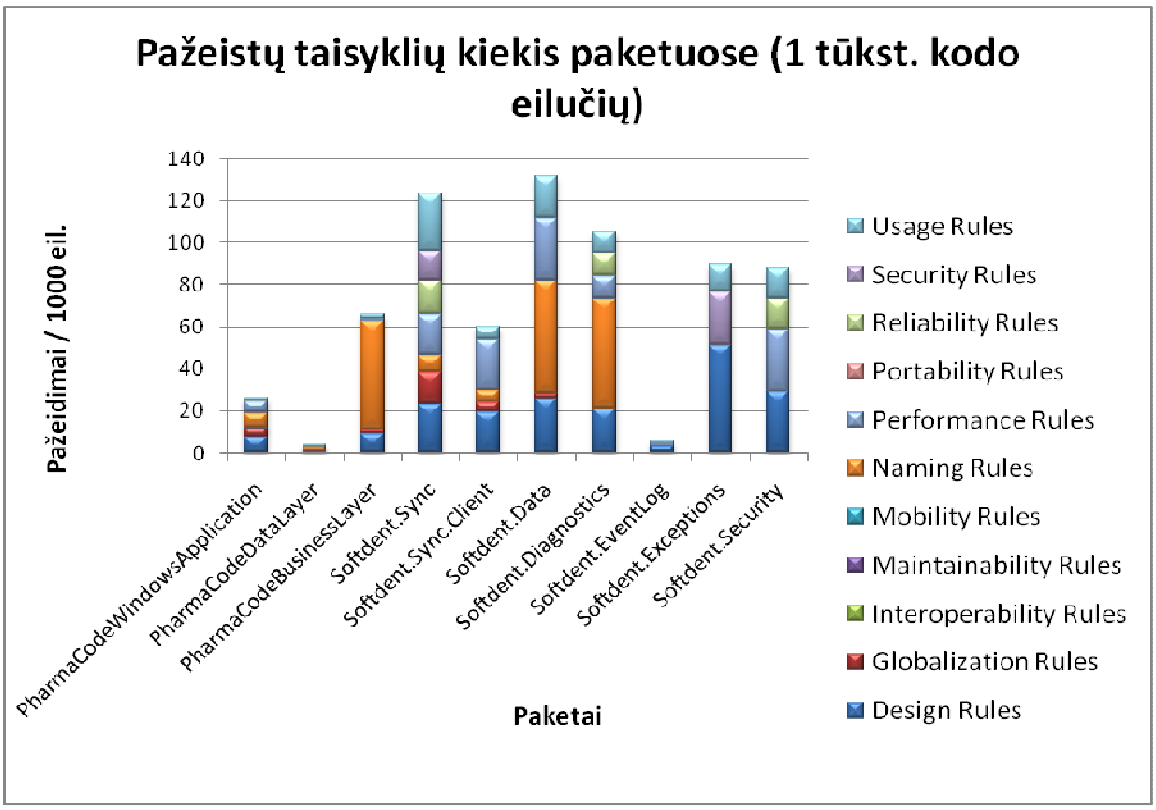
Toliau pateikiame esminius analizės rezultatus.



Pav. 21 - Pažeistų taisyklių kiekis paketuose

Iš šios diagramos matome, jog išeities tekstų analizės metu kiekybiškai daugiausia pažeidimų aptikta pakete „PharmaCodeWindowsApplication“, viso – 1879. Tarp jų didžiausią dalį užėmė „Design rules“, „Naming rules“ ir „Performance rules“ pažeidžiamų taisyklių grupės. Galbūt būtų galima daryti išvadą, jog tarp visų sistemos paketų prasčiausia kokybe pasižymi „PharmaCodeWindowsApplication“, tačiau mes dar neįvertinome kiekvieno iš paketų apimties.

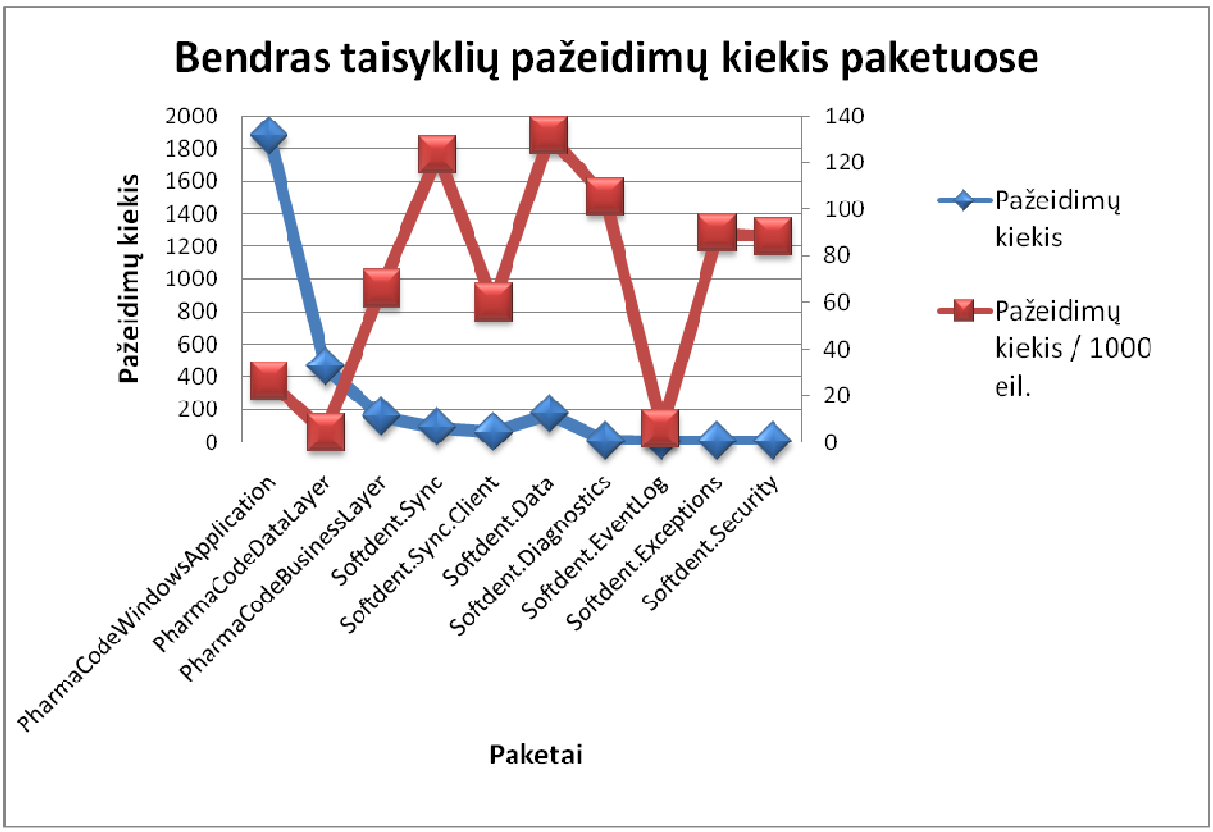
Toliau pateikiamoje diagramoje atsižvelgiame ir į kodo eilučių kiekį, o tuos pačius rezultatus pateikiame vienam tūkstančiui kodo eilučių:



Pav. 22 - Pažeistų taisyklių kiekis paketuose (1 tūkst. kodo eilučių)

Kaip matome, vaizdas pasikeičia iš esmės, kadangi kiekybiškai daugiausiai galimų defektų turintys paketai (tokie kaip „PharmaCodeWindowsApplication“ ir „PharmaCodeDataLayer“) taipogi yra didžiausi ir pagal kodo apimtį. Todėl, paskaičiavus pažeidimų kiekį vienam tūkst. kodo eilučių pasirodo, jog prasčiausia kokybe pasižymi „Softdent.Sync“ ir „Softdent.Data“ paketai. O tai yra labai svarbu, kadangi abu šie paketai yra pakartotinai naudojami keliuose projektuose.

Toliau pateikiame apibendrintą grafiką, vienoje skalėje atvaizduojantį tiek pažeidimų kiekį, tiek ir santykinį pažeidimų kiekį, leidžiantį šiuos du dydžius palyginti tarpusavyje:



Pav. 23 - Bendras taisyklių pažeidimų kiekis paketuose

Apibendrinę surinktą statistiką (pilną statistiką pateikiame priede nr.2), sudarome rekomendacijas tolesniems sistemos tobulinimams:

- Sistemos kode daugiausia taisyklių pažeidimų (724) aptikta „Design rules“ taisyklių grupėje (~191.397 pažeidimų/1000 eil.);
- „Naming rules“ grupėje buvo 955 pažeidimai, tačiau taisyklės bendro pobūdžio jų pažeidimai nesukelia rimtesnių pasekmių.
- Rekomenduojamos taisyti šios („Design rules“ grupės) problemos:
    1) (146 kartai) „CA1031: Do not catch general exception types“
    2) (234 kartai) „CA1051: Do not declare visible instance fields“
    3) (154 kartai) „CA1062: Validate arguments of public methods“
- Rekomenduojama įdiegti automatizuotą šių taisyklių tikrinimą kasdieniame išeities tekstų kompiliavimo procese (galbūt kartu su vienetų testavimu).

4.3. Papildomos taisyklės realizacija

SQLĮ atakoms išvengti Microsoft siūlo naudoti *išankstinio sakinio paruošimo* metodologiją, aprašytą [4]. Ją siūloma taikyti tiek .NET kodui, tiek DB procedūroms. Mūsų metodas yra paremtas šiomis rekomendacijomis.

Mes, statinės kodo analizės metodais, siūlome *palengvinti* programų kūrėjams įgyvendinti Microsoft saugaus programavimo rekomendacijas.

Microsoft Visual Studio Team System yra panašaus pobūdžio statinės kodo analizės taisyklė „CA2100: Review Sql queries for security vulnerabilities“, kuri talpinama „Security“ taisyklių grupėje. Taisyklės tikslas – peržiūrėti visus sąveikos su DB taškus (angl. interaction-points) ir informuoti programuotoją apie tas sąveikos vietas, kuriose užklausa lipdoma iš teksto gabalų. Sąveikos taškai .NET platformoje - tai klasės, realizuojančios „`System.Data.IDbCommand`“ ir „`System.Data.IDbDataAdapter`“ interfeisus. Taisyklė ieško programos MSIL (angl. Microsoft Intermediate Language) kode komandų, iškviečiančių „`System.Data.IDbCommand`“ arba „`System.Data.IDbDataAdapter`“ „`set_CommandText`“ funkciją. Vienas iš funkcijos „`set_CommandText`“ parametrų yra komandos tekstas. Kodo analizės taisyklė tikrina, ar tas parametras – tekstinė eilutė yra sujungta iš keleto teksto gabalų. Jei taip – ši vieta traktuojama kaip potencialiai nesaugi, ir siūloma ją perrašyti naudojantis Microsoft rekomendacijomis, aprašytomis [4]. Perrašius komandą, atsižvelgiant į rekomendacijas, komandos tekstą sudaro vientisa (ne iš teksto gabalų sudaryta) eilutė (žiūr. Pav. 25). Žemiau pateikiame tą pačią funkciją, parašytą su spragomis (kai užklausa formuojama iš teksto gabalų), ir korektišką jos variantą (parašytą pagal [4] rekomendacijas).

```
private int authenticate_without_params(string username, string password)
{
    SqlCommand comm = new SqlCommand(
        "SELECT UserID FROM Users WHERE Username='"+username+
        "' AND Password='"+password+"'",
       this.sqlConMaster);
    return (int)comm.ExecuteScalar();
}
```

Pav. 24 - Užklausa, pažeidžiama SQLĮA, nes formuojama iš teksto gabalų

```
private int authenticate_with_params(string username, string password)
{
    SqlCommand comm = new SqlCommand(
        "SELECT UserID FROM Users WHERE Username=@Username AND Password=@Password",
        this.sqlConMaster);
    comm.Parameters.Add(new SqlParameter("@Username", username));
    comm.Parameters.Add(new SqlParameter("@Password", password));
    return (int)comm.ExecuteScalar();
}
```

Pav. 25 - Užklausa, parašyta pagal [4] rekomendacijas

Esamos Microsoft kodo analizės taisyklės trūkumas yra tas, jog ji, analizuodama programų kodus, tikrina taisyklės galiojimą tik vienos funkcijos lygmenyje. Taisyklė visiškai neatsižvelgia į kviečiamų funkcijų steką ar vykdymo eigą. Tam tikslui parašėme programą su testiniais atvejais, specialiai šiai kodo analizės taisyklės spragai tikrinti. Testiniai atvejai detaliau aprašyti **5** sk. – eksperimentinėje dalyje.

Mūsų siūlomas sprendimas tikrina ne tik „set_CommandText“ funkcijai paduodamą tekstinę eilutę, bet papildomai aptinka ir tas užklausas, kurios paduodamos kaip funkcijos parametrai.

Taisyklę realizavome C# programavimo kalba, naudodamiesi „FxCOP“ taisyklių kūrimo sąsaja. Kaip kuriamos kodo analizės taisyklės, detaliau aprašyta J.Kresowaty [5] darbe.

4.4. Išvados

Atlikus tiriamąjį darbą, padarytos šios išvados:

1) Įvertinus projekto eigą, programuotojų aktyvumą ir surinktą defektų valdymo sistemos statistiką darome išvadą, jog projektas bus paruošas naudojimui ir atitiks pageidaujamą kokybę iki plane nurodyto termino;
2) Atlikę išeities tekstų statinę analizę suformavome kiekybinius ir santykinius tikėtinų defektų įverčius projekto paketams. Vėliau šie įverčiai bus panaudoti aptikti kritinėms projekto vietoms;
3) Palyginę suformuotus kiekybinius ir santykinius defektų įverčius paketuose „PharmaCodeDataLayer“ ir „PharmaCodeWindowsApplication“ darome išvadą, jog didelę įtaką išeities tektų kokybei daro žmogiškasis faktorius. Tikėtina, jog paketas, kuriame didesnę dalį kodo sudaro sugeneruotas kodas – pasižymės aukštesne kokybe;
4) Atlikę siūlomų kodo analizės taisyklių tyrimą aptikome, jog viena iš saugumo grupės taisyklių gali būti patobulinta. Realizavome patobulintą šios taisyklės analogą.

5. EKSPERIMENTINĖ DALIS

Savo realizuotą statinės kodo analizės taisyklę išbandėme su magistrinio darbo projektu - realia, veikiančia CRM (angl. Customer Relationship Management) sistema „PharmaCODE“, kurią sudaro apie 250 tūkst. kodo eilučių. Taisyklę taipogi išbandėme ir su specialiai tam parašyta „Hello World“ programa. Į „Hello World“ programą įtraukėme keletą funkcijų, reprezentuojančių įvairius užklausų formavimo būdus.

5.1. Bandymai su specialiai tam parašytu kodu

„Hello World“ programa buvo kaip pradiniai įėjimo duomenys mūsų parašytai kodo analizės taisyklei. Joje realizavome keletą užklausų programavimo variantų, iš kurių tik vienas yra geras, t.y. nėra pažeidžiamas SQLĮA. Žemiau pateikiame testinių variantų, kuriuos turėjo programa „Hello World“, lentelę.

Lentelė nr. 4 - „Hello World“ programoje esantys testiniai atvejai

Testinio atvejo apibūdinimas	**Testuojamas programos kodas**
1) Kodas, parašytas laikantis saugaus programavimo rekomendacijų	Žiūr. **Pav. 25.**
2) Kodas, formuojantis užklausą iš teksto gabalų.	Žiūr. **Pav. 24.**

3) Kodas, formuojantis užklausą iš teksto gabalų, tačiau pati užklausa paduodama kaip parametras „query“.	`private int authenticate_building_query_localy(string username, string password)`
	`{`
	`  string query = "SELECT UserID FROM Users WHERE Username='" + username + "' AND Password='" + password + "'";`
	`  SqlCommand comm = new SqlCommand(query, this.sqlConMaster);`
	`  return (int)comm.ExecuteScalar();`
	`}`

4) Kodas, formuojantis užklausą iš teksto gabalų, tačiau pati užklausa paduodama kaip išorinis funkcijos „execureScalar1“ parametras	`private int executeScalar1(string qr)`
	`{`
	`  SqlCommand comm = new SqlCommand(qr, this.sqlConMaster);`
	`  return (int)comm.ExecuteScalar();`
	`}`
	`private int authenticate_higher_in_stack(string username, string password)`
	`{`
	`  string query = "SELECT UserID FROM Users WHERE Username='" + username + "' AND Password='" + password + "'";`
	`  return this.executeScalar1(query);`
	`}`

Žemiau pateikiame kodo mūsų kodo analizės taisyklės realizacijos palyginimą su jau egzistuojančia taisykle. Kaip matome iš lentelės, mūsų sukurta kodo analizės taisyklė aptinka visus variantus, tame tarpe ir tuos, kai užklausa paduodama kaip funkcijos parametras.

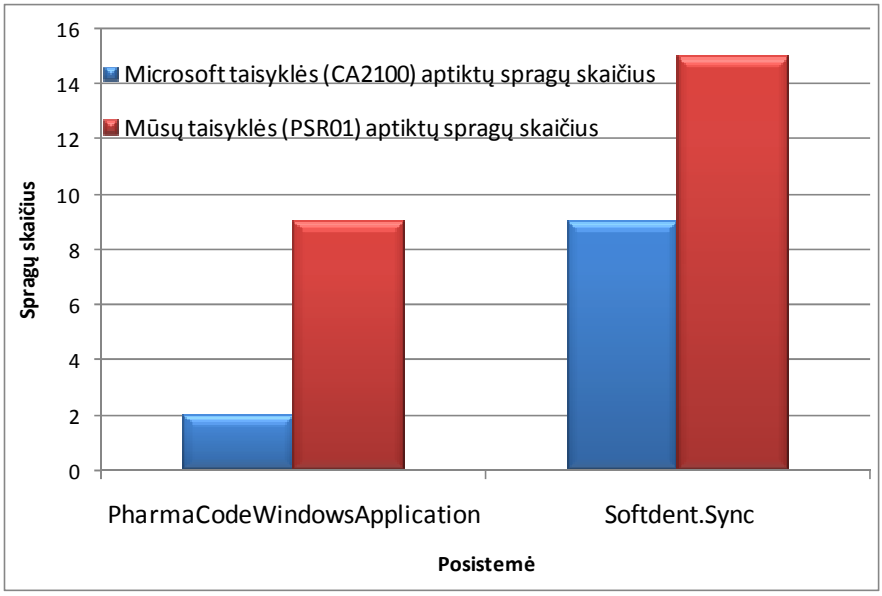
Lentelė nr. 5 - Rezultatų palyginimas su dirbtiniais testiniais atvejais, programoje „Hello World“

Testinio atvejo nr.	**Microsoft taisyklės CA2100 rezultatas**	**Mūsų taisyklės PSR01 rezultatas**	**Teisingas rezultatas**
1	Užklausa gera	Užklausa gera	Užklausa gera
2	Užklausa bloga	Užklausa bloga	Užklausa bloga
3	Užklausa bloga	Užklausa bloga	Užklausa bloga

4	Užklausa gera	Užklausa bloga	Užklausa bloga
VISO:	3 iš 4	4 iš 4	

5.2. Bandymai su realia sistema – „PharmaCODE“

PharmaCODE sistemos tyrimo metu buvo atliktas dviejų paketų – „PharmaCodeWindowsApplication“ ir „Softent.Sync“ kodo analizė. Žemiau pateikiame bandymų rezultatus vaizduojantį grafiką.



Pav. 26 - Aptiktų spragų sistemoje „PharmaCODE“ skaičiaus palyginimas

5.3. Eksperimentų rezultatai

Kaip matome iš tyrimo rezultatų, mūsų taisyklė aptinka tas spragas kurių jos pirmtakė (Microsoft CA2100 saugumo taisyklė) neaptiko. Tačiau sugriežtinus taisyklės realizaciją, padidėja tikimybė jog ji pernelyg dažnai neteisingai atmes saugias SQL užklausas, jas laikydama nesaugiomis. Taip yra todėl, kad ne visada įmanoma visas užklausas realizuoti vienu, statiniu SQL sakiniu. Dažnai būna taip, jog reikia suprogramuoti nuosavą SQL užklausų generavimo metodą, koks ir buvo realizuotas „PharmaCODE“ sistemoje. Tokiems klaidingiems perspėjimams išjungti programuotojai gali naudoti funkcijos atributą:

```
[System.Diagnostics.CodeAnalysis.SuppressMessage
          ("Prelgauskas.FxCopSecurityRules", "PSR01")]
```

Jei programuojama SQL užklausa yra formuojama iš teksto gabalų, o programuotojas yra užtikrintas, jog nei vienas jos elementas nėra įvedamas vartotojo – jis gali uždėti šį atributą. Tokiu būdu sumažinamas klaidingų perspėjimų (angl. false positives) skaičius.

Planuojami tolimesni darbai:

1) patobulinti šį algoritmą, leidžiantį atlikti analizę atsižvelgiant į instrukcijų vykdymo grafą [5], 17-18 psl. Tokiu būdu būtų automatiškai sekamas „System.Diagnostics.CodeAnalysis.SuppressMessage“ atributas, ir kitos

analizuojamos funkcijos, kviečiančios šią, „saugiąją“ funkciją, būtų ignoruojamos ir traktuojamos kaip saugios;

2) panašiu, statinės analizės būdu spręsti SQLĮA problemą DB procedūrose (angl. stored procedures). Šis metodas gali būti tinkamas ir DB procedūroms, kadangi (kaip ir .NET kode) rekomenduojama formuoti užklausas kaip vientisą tekstinę eilutę su vietos skyrikliais, nurodančiais parametrų vietą [4].

6. IŠVADOS

Pagrindiniai atlikti darbai ir pasiekimai:

1) Suprojektuota ir įdiegta komercinė CRM sistema;
    - Naudojama .NET platforma;
    - „Infragistics“ grafinės sąsajos komponentai;
    - Microsoft Enterprise Library;
    - Microsoft Sync Services;
2) Statinė kodo analizė gali būti pritaikyta kritinių kodo vietų aptikimui (t.y. tokių vietų išeities tekstuose, kuriuose gali būti daugiausia potencialių klaidų). Sudarytos rekomendacijos tolesniam kodo tobulinimui, remiantis šiomis aptiktomis „kritinėmis vietomis“.
3) Sukurta papildoma statinės kodo analizės taisyklė, aptinkanti daugiau tos pačios rūšies saugumo spragų (SQLĮ), nei jos pirmtakė – analogiška Microsoft analizės taisyklė;

7. LITERATŪRA

1. **M.Muthuprasanna, Ke Wei, Suraj Kothari**. Eliminating SQL Injection Attack – A Transparent Defense Mechanism. Proc. 8th IEEE International Symposium on Web Site Evolution (WSE'06), 2006.
2. **Z.Su, G.Wassermann**. The Essence of Command Injection Attacks in Web Applications. Proc. 33rd annual Symposium on Principles of Programming Languages, 2006.
3. **W.G.Halfond, J.Viegas, A.Orso**. A Classification of SQL-Injection Attacks and Countermeasures. Proc. IEEE International Symposium on Secure Software Engineering, 2006.
4. **M.Howard, D.LeBlanc**. Writing secure code, second edition. Microsoft press, 2003.
5. **J.Kresowaty**. FxCop: Writing Your Own Custom Rules (DRAFT), http://www.binarycoder.net/fxcop/pdf/fxcop.pdf, 2008.
6. **Ke Wei, M.Muthuprasanna, Suraj Kothari**. Prevening SQL Injection Attacks in Stored Procedures. Proc. IEEE Australian Software Engineering Conference (ASWEC'06), 2006.
7. **F.Valeur, D.Mutz, G.Vigna**. A Learning-Based Approach to the Detection of SQL Attacks. Proc. Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA), 123-140 psl., 2005.
8. **Y.Huang, S.Huang, T.Lin, C.Tsai**. Web Application Security Assessment by Fault Injection and Behavior Monitoring. Proc. 12th International World Wide Web Conference (WWW03), 148-159 psl., 2003.
9. **S.W.Boyd, A.D.Keromytis.** SQLrand: Preventing SQL injection attacks. Proc. 2nd Applied Cryptography and Network Security (ACNS) Conference, psl. 292-304, 2004.
10. **Struts**. Apache Struts project. http://struts.apache.org/ .
11. **A.Nguyen-Tuong, S.Guarnieri, D.Greene, J.Shirley, D.Evans**. Automatically hardening web applications using precise tainting. Proc. 20th IFIP International Information Security Conference, 2005.
12. **W.G.Halfond, A.Orso, P.Manolios**. WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation. IEEE Transactions on Software Engineering, vol.34, no.1, 2008.
13. **A.Christensen, A.Moller, M.Schwartzbach**. Precise analysis of string expressions. Proc. International Static Analysis Symposium (SAS'03), 2003.
14. **G.Wassermann, Z.Su**. An Analysis Framework for Security in Web Applications. Proc. FSE Workshop on Specification and Verification of Component-Based Systems (SAVCBS), psl 70-78, 2004.
15. **W.Halfond, A.Orso**. AMNESIA: Analysis and Monitoring for Neutralizing SQL-Injection Attacks. Proc. 20th IEEE/ACM International Conference on Automated Software Engineering (ASE), psl. 174-183, 2005.
16. **G.T.Buehrer, B.W.Weide, P.A.G.Sivilotti**. Using parse tree validation to prevent SQL injection attacks. Proc. 5th International Workshop on Software Engineering and Middleware SEM, psl. 106-113, 2005.
17. **Hibernate**. Hibernate.org. http://www.hibernate.org/ .
18. **Computer Security Institute**. Computer crime and security survey, http://www.gocsi.com/press/20020407.jhtml, 2002.
19. **WebCohort Inc.** Only 10% Web applications secured against common hacking techniques, http://www.imperva.com/company/news/2004-feb-02.html, 2004.
20. **Open Web Application Security Project (OWASP)**. Top Ten most critical web application vulnerabilities, http://www.owasp.org/documentation/topten.html, 2005.
21. **D.Chappel**. SOA and reality of reuse, http://www.davidchappell.com/HTML_email/Opinari_No16_8_06.html#article, 2006.
22. **Etinių farmacijos kompanijų atstovybių asociacija**. Etinių farmacijos kompanijų atstovybių asociacijos oficialus tinklapis, http://www.efa.lt/naujas/index.php, 2006.

23. **Exmarket Holding**. Strong marketing tool to manage and to follow-up your bussiness, MicroView ETMS, 2001.
24. **IMS Health**. IMS Health Meets Data Analysis and Business Decision Support Demands with Microsoft technology, http://www.imshealth.com/vgn/images/portal/cit_40000873/40068853IMS%20SAles%20Analyzer.pdf, 2002.
25. **M.Keen, A.Acharya, S.Bishop**. Patterns: Implementing a SOA using Enterprise Service Bus. IBM, 2004.
26. **S.McLean, J.Naftel, K.Williams**. Microsoft .NET Remoting. Redmond: Microsoft Press, 2003.
27. **K.Jones, E.Jezierski, J.Hogg, R.Leibovitz, C.Campbell**. Building Distributed Applications: Exception Management Architecture Guide. MSDN Library, http://msdn2.microsoft.com/en-us/library/ms954599.aspx, 2003.
28. **Microsoft**. Microsoft Patterns and Practices grupės rekomendacijos architektams. Patterns and Practices, http://msdn2.microsoft.com/en-us/practices/default.aspx, 2006.
29. **D.Sprott, L.Wilkes**. Understanding Service-Oriented Architecture. Microsoft Architect Journal, 2004.
30. **Oracle, Inc**. Oracle home page: http://www.oracle.com/, 2006.
31. **E.Pulier, H.Taylor**. Understanding Enterprise SOA. Manning, 2006.
32. **P.Robert, B.Andreas**. A Secure Web Service: Specification on how to implement a secure Web service in a healthcare environment. Vaxjo University, 2005.
33. **TelyNet**. TelyNET sales ETMS: The whole relevant information on Medical Visit management, http://www.telynet.es/en/products/Cat%C3%A1logo%20de%20TelyNET%20sales%20ETMS%20Ingles.pdf, 2006.

8. TERMINŲ IR SANTRUMPŲ ŽODYNAS

API - Programinė sąsaja (iš angl. Application Programming Interface)

COM - Component Object Model

CRM - Customer Relationship Management

COTS - Commercial Off-The-Shelf software.

DBVS - Duomenų bazių valdymo sistema.

DCOM - Distributed Component Object Model

ETMS - (iš anglų kalbos – Electronic Territory Management System)

GUI - Graphical User Interface.

HTTP - Hypertext Transfer Protocol

IIS - Internet Information Server.

IDE - Integruota programų kūrimo aplinka (angl. Integrated Development Environment)

LAN - Local Area Network

MSIL - (iš anglų kalbos – Microsoft Intermediate Language) tarpinė išeities tekstų atvaizdavimo forma .NET aplinkoje.

.NET - Microsoft programų kūrimo platforma.

OO - Object Orientation

OMG - Object Management Group.

PDA - Portable Digital Assistant.

RUP - Rational Unified Process

SO - Service Orientation

SOAP - Simple Object Access Protocol

SOA - Service-Oriented Architecture

SSL - Secure Sockets Layer.

SQL - (iš anglų kalbos – Structured Querying Language) duomenų bazės užklausų programavimo kalba.

SQLĮ - SQL-įterpinys.

SQLĮA - SQL-įterpinių ataka.

UDDI - Universal Description Discovery and Integration

UML - Unified Modeling Language.

XML - Extended Markup Language

WSDL - Web Servines Description Language

W3C - World Wide Web Consortium

9. PRIEDAI

9.1. Publikacija

Pateikiame papildomai paruoštą ir konferencijoje „Informacinės Technologijos‘2008“ pristatytą publikaciją:

SQL-įterpinių saugumo problemos sprendimas panaudojant statinės kodo analizės metodą

prelgauskas_it2008_2.doc

9.2. Statinės išeities tekstų analizės statistika

Pateikiame lentelę su visa projekto paketų išeities tekstų analizės statistika.

rezultatai.xls