KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Edgaras Norvaiša

Į paslaugas orientuotos architektūros tyrimas įvertinant priežiūros indeksą

Magistro projektas

Vadovas
Doc. dr. Šarūnas Packevičius

KAUNAS, 2017

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Į paslaugas orientuotos architektūros tyrimas įvertinant priežiūros indeksą

Magistro projektas
Programų sistemų inžinerija (kodas 621E16001)

Vadovas
Doc. dr. Šarūnas Packevičius

Recenzentas
Prof. dr. Robertas Damaševičius

Projektą atliko
Edgaras Norvaiša
IFM-5/2 gr. studentas

KAUNAS, 2017

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

(Fakultetas)

EDGARAS NORVAIŠA

(Studento vardas, pavardė)

PROGRAMŲ SISTEMŲ INŽINERIJA, 621E16001

(Studijų programos pavadinimas, kodas)

„Į paslaugas orientuotos architektūros tyrimas įvertinant priežiūros indeksą“

AKADEMINIO SĄŽININGUMO DEKLARACIJA

20 17 m. gegužės 25 d.

Kaunas

Patvirtinu, kad mano, **Edgaro Norvaišo**, baigiamasis projektas tema „*Į paslaugas orientuotos architektūros tyrimas įvertinant priežiūros indeksą*“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

(vardą ir pavardę įrašyti ranka) *(parašas)*

Turinys

LENTELIŲ SĄRAŠAS

PAVEIKSLĖLIŲ SĄRAŠAS

Edgaras, Norvaiša. *Service oriented architecture evaluation based on maintainability index*. Master‘s degree, supervisor assoc. doc. dr. Šarūnas Packevičius; The Faculty of informatics, Kaunas University of Technology.

Research area and field: Software Engineering
Key words: *SOA, code metrics, maintainability*
Kaunas, 2017.

SUMMARY

Software architecture is meant to define system structure which is a core for product development. Defying system structure is first and most important step in software development. It involves a series of decisions based on a wide range of factors, and each of these decisions can have considerable impact on the quality, performance, maintainability, and overall success of the application. Most efficient way to be able to control system characteristics is to use certain architectural models like SOA or any other. In this paper focus object will be how maintainability is effected for SOA based applications.

Based on SOA principles was created a software which allows companies to evaluate their employees. The main customer’ requirement for the system was a high maintainability index which indicates whether the system is easily maintainable and extendable.

This document contains a research and technologies overview for service-oriented architecture implementation. Document also covers analyze of software metrics that were used in the experimental part by evaluating SOA maintainability index and what is effecting it. A sequence of software design realization and implementation is also presented on a project part of this document.

1. ĮŽANGA

1.1. Dokumento paskirtis

Šis dokumentas yra skirtas aprašyti sukurtos sistemos architektūrą ir atlikti jos analizę. Taip pat dokumente yra atliekama teorinė technologijų analizė, susijusi su sistemos architektūros realizacija. Eksperimento dalyje yra pateikti atlikto tyrimo rezultatai ir jų analizė.

1.2. Santrauka

Edgaras Norvaiša. *Į paslaugas orientuotos architektūros tyrimas įvertinant priežiūros indeksą*. Magistro baigiamasis projektas, vadovas doc. dr. Šarūnas Packevičius; Kauno technologijos universitetas, Informatikos fakultetas.

Mokslo kryptis ir sritis: Programų sistemų inžinerija
Reikšminiai žodžiai: *SOA metrikos, programos priežiūra*
Kaunas, 2017.

Programinės įrangos architektūra yra skirta apibrėžti programos struktūrą, kuri yra pagrindas produkto kūrimui. Taigi programos struktūros apibrėžimas yra vienas pagrindinių ir svarbiausių programinės įrangos kūrimo žingsnių. Šis procesas susideda iš įvairių sprendimų ir faktorių, kur kiekvienas jų gali turėti įtakos programos kokybei, priežiūrai, greitaveikai ir bendrai programos sėkmei. Pats efektyviausias būdas teigiamai paveikti pastarasias sistemos charakteristikas yra SOA arba kito architektūrinio modelio panaudojimas.

Šio darbo metu buvo sukurta įmonės darbuotojus leidžianti vertinti programinė įranga, kurios architektūra yra paremta SOA principais. Šios sistemos pagrindinis reikalavimas, kuris buvo pateiktas užsakovo, yra aukštas priežiūros indeksas, kuris indikuoja ar sistema yra lengvai prižiūrima ir praplečiama.

Šiame dokumente pateikiama magistrinio darbo analizė, aptariamos technologijos, skirtos į paslaugas orientuotos architektūros realizavimui. Darbe yra analizuojamos programinės įrangos metrikos, kurios buvo naudojamos eksperimentinėje dalyje, įvertinant SOA priežiūros indeksą bei, kas jam turi įtakos. Dokumento projektinėje dalyje pateikiama programinės įrangos projektavimo ir realizavimo seka. Eksperimentinėje dalyje tiriama, kaip keičiasi SOA priežiūros indeksas, turint skirtingus sistemos architektūros išdėstymus. Eksperimento rezultatai leido nustatyti kaip prižiūrimumas yra paveikiamas naudojant skirtingus SOA principus ir kokie architektūriniai sprendimai gali pagerinti sistemos priežiūrą.

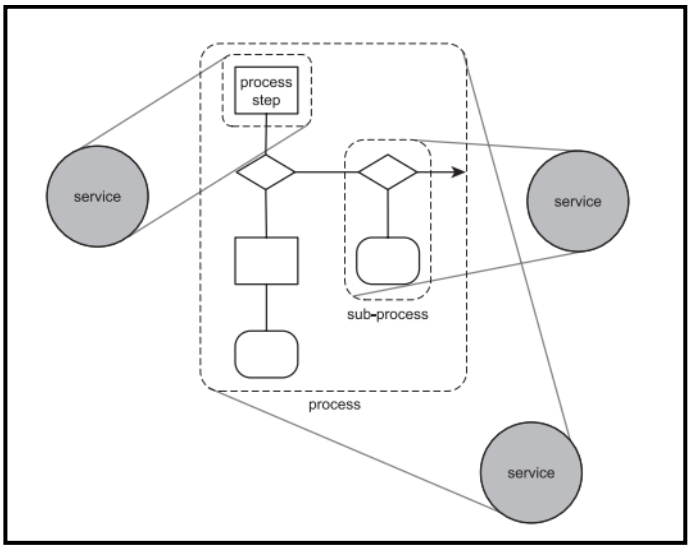
2. ANALITINĖ DALIS

2.1. SOA architektūrinis modelis

SOA – tai į paslaugas orientuotas programinės įrangos architektūrinis modelis (angl. *service-oriented architectur*). SOA yra įvardinama kaip kompiuterijos paradigma. Kitos kompiuterijos paradigmos, tokios, kaip objekto orientacija (angl. *object orientation*), dalijasi tam tikrais principais ir tokiu būdu tarsi apibrėžia SOA pavadinimą. Pagrindinis SOA modelio bruožas yra tai, jog jis išskiria sistemos dalis į atskirus modulius, kurie yra atsakingi tik už konkretų procesą sistemoje. SOA yra siejamas su principais, nusakančiais, kaip projektuoti į paslaugas orientuotą programinę įrangą, kurios komponentai teikia paslaugas kitiems komponentams, naudojant bendravimo protokolus (angl. *communications protocol*) tinkle. Naudojant SOA architektūrą, sisteminis sprendimas yra sudarytas iš atskirų komponentų, kurie gali būti pasiskirstę tiek geografiniu, tiek platformos atžvilgiu. Vadinasi, sistemos komponentų realizacija yra nepriklausoma nuo technologinių sprendimų. Nepaisant to, jog komponentai skiriasi, reikia laikytis apibrėžtų komunikacijos standartų, kuriuos apibrėžia pats komponentas. [1]

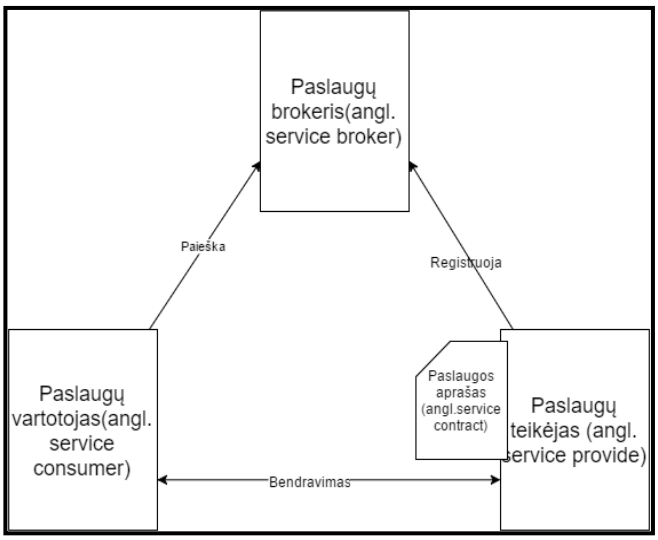
Dažnai literatūroje yra pabrėžiama tai, jog SOA nėra saityno paslauga (angl. *web service*), tačiau saityno paslauga gali būti galutinis rezultatas taikant SOA. Jei sistema bus realizuota panaudojus keletą saityno paslaugų, tai ne visada reiškia, kad tokia sistema bus paremta SOA principu.

Pateiktame paveikslėlyje (žr. 2.1 pav. pav.) yra pavaizduotas pavyzdys, kaip sistemoje gali būti apdorojamas vienas procesas. Kiekviena proceso dalis yra apdorojama skirtinguose komponentuose. Komponentas gali būti suprantamas, kaip atskira saityno paslauga (angl. *web service*) arba kaip sistemos atskiras komponentas, atsakingas tik už jam skirtą logiką. [2]



2.1 pav. Proceso apdorojimas naudojantis paslaugomis

Literatūroje SOA architektūrinio modelio diagrama dažniausiai yra vaizduojama taip, kaip pateiktame paveikslėlyje (žr. 2.2 2.2 pav.). Paslaugų vartotojas siunčia vieną ar daugiau užklausą tiekėjui, taip siekiant rasti paslaugą ir nuspręsti, kaip reikės bendrauti su tos paslaugos tiekėju. Paslaugų tiekėjas apibūdina paslaugos buvimo vietą ir aprašus (WSLD). Jis yra atsakingas, kad paslaugos tiekėjo aprašas (WSDL) būtų pasiekiamas konkrečiam vartotojui. Paslaugos teikėjas yra pagrindinis komponentas, kuriame yra realizuotas tam tikras funkcionalumas. Tiekėjo teikiamas paslaugas ir kaip su juo bendrauti (kokiais protokolais) nusako WSDL. Paslaugos tiekėjas, tai saityno paslauga (angl *web service*). [3]



2.2 pav. SOA architektūrinis modelis

2.2. N-eilės (angl. N-tier) architektūra

N-eilės (angl. N-tier) architektūra literatūros šaltiniuose apibrėžiama, kaip vartotojo-serverio architektūra, kuri yra išskaidoma į detalesnius lygius. Šio tipo architektūra realizuotos programos dar yra vadinamos paskirstytomis (angl. *distributed*) arba daugiasluoksnėmis (angl. *multitier*). Naudojant šį architektūrinį sprendimą, sistemos procesai yra išskiriami į atskirus sluoksnius. Standartiškai n-eilės programose yra išskiriami trys sluoksniai:

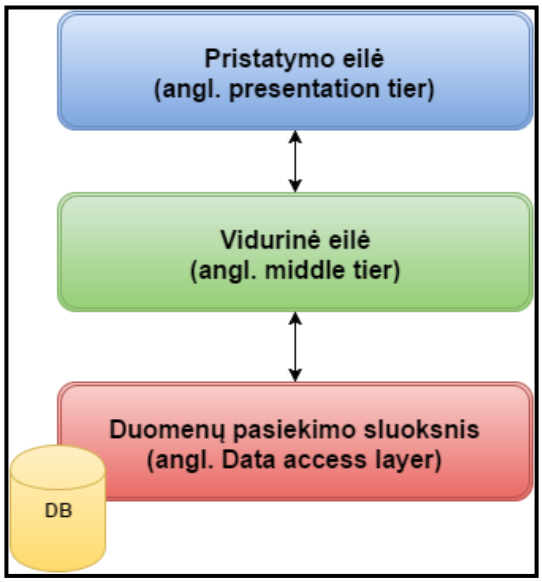
1. Pristatymo eilė (angl. *presentation tier*);
2. Vidurinė eilė (angl. *middle tier*);
3. Duomenų eilė (angl. *data tier*).

Pateiktame paveikslėlyje (žr. 2.3 2.3 pav.) yra pavaizduota programa su trijų sluoksnių architektūra. Pristatymo sluoksnis skirtas vartotojo komunikacijai su programa. Šis sluoksnis atsakingas už technologijas, skirtas vartotojo sąsajos realizavimui (pvz., *Win Froms, WPF, Mobile*) bei logikos, kuri skirta rezultatų, gautų iš vidurinio sluoksnio, atvaizdavimui.

Vidurinė pakopa yra atsakinga už programos procesų vykdymus. Šiame sluoksnyje yra sąrašas operacijų, kurios atlieka tam tikrus loginius veiksmus su duomenimis. Šioje pakopoje yra realizuojami

duomenų pardavimo objektai (angl. DTO), funkcijos, skaičiavimai, duomenų tikrinimas bei kiti komponentai, skirti programos logikai realizuoti (pvz. autorizacija). Literatūroje ši pakopa dar vadinama - „*Business logic layer*“.

Duomenų pasiekimo sluoksnis – duomenų eilė, atsakinga už programos komunikavimą su duomenų baze. Šioje pakopoje yra realizuojamos visos CRUD operacijos. Šis sluoksnis negali būti pasiekiamas vartotojui tiek iš išorės, tiek iš pristatymo (angl. *presentation layer*) pakopos. [4]



2.3 pav. 3-eilės architektūra

3-eilės architektūros privalumai:

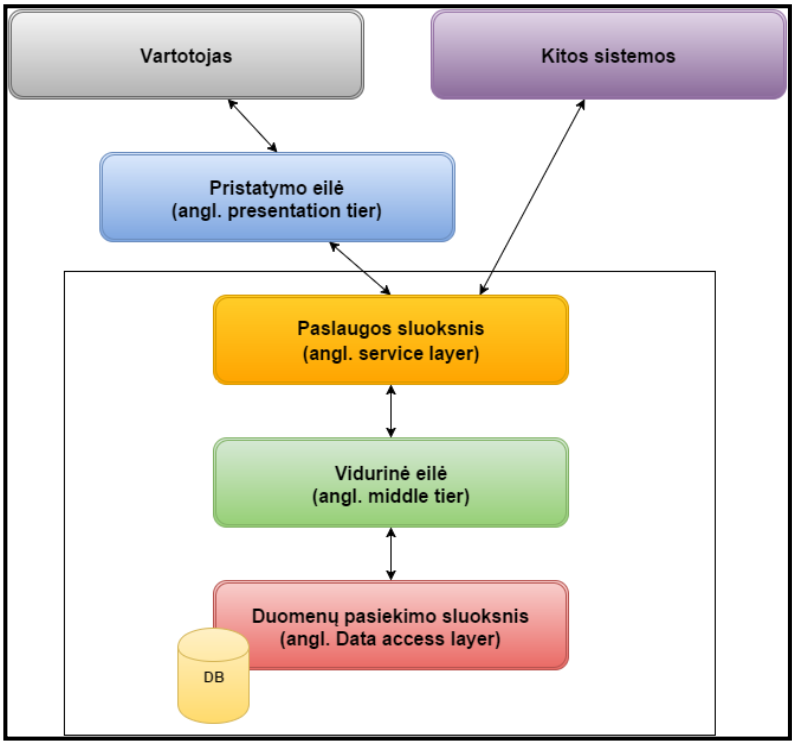
- Priežiūra (angl. *maintainability*) – kiekvienas sluoksnis yra izoliuotas ir nepriklauso vienas nuo kito, todėl atlikti pakeitimus yra saugu, taip nepaveikiant kitų sluoksnių.
- Praplečiamumas (angl. *scalability*) – kiekvienas sluoksnis gali būti praplečiamas horizontaliai, nedarant jokios įtakos kitiems sluoksniams.
- Lankstumo (angl. *flexibility*) – kiekvienas sluoksnis kontroliuojamas ir praplečiamas individualiai.
- Prieinamumas (angl. *availability*) – tokios architektūros programos gali pasinaudoti savo modalumo savybėmis ir suteikti prieigą prie savo sluoksnių.

Literatūroje pateiktas 3-eilės architektūrinis sprendimas, yra tik standartinis variantas. N-eilės architektūra gali turėti kur kas daugiau sluoksnių, kurie bus atsakingi tik už jiems skirtas atsakomybes.

SOA architektūrinis modelis dažnai būna realizuojamas naudojant n-eilės architektūros principus. Paveikslėlyje (žr. 2.4 2.4 pav.) pavaizduota n-eilės sistema, kurioje pridėtas dar vienas sluoksnis – paslaugos sluoksnis (angl. *servce laeyer*). Šis sluoksnis suteikia galimybę pasiekti vidurinį sluoksnį ne tik vartotojams, naudojantiems pristatymo eilę, bet ir kitoms sistemoms, naudojant

bendravimo kanalus (angl. *communication channels*) [1] [5]. Pagrindiniai paslaugos sluoksnio komponentai :

- Paslaugos sąsajos (angl. *service inteface*) – jos paskirtis yra priimti atvykstančias užklausas, nukreipti atitinkamas operacijas, realizuotas vidurinėje eilėje, pagal užklausos tipus. Taip pat pateikti vartotojo užklausų formatus (išsamiau apie tai bus kalbama saityno analizės dalyje).
- Žinučių tipai – keičiantis duomenimis su paslaugų sluoksniu, duomenys yra suformuojami pagal žinučių struktūrą, kad norima operacija būtų atlikta sėkmingai.



2.4 pav. n-eilės architektūra

2.3. Saityno paslauga

Saityno paslauga, tai programinės įrangos komponentas. Ji pratęsia objektiškai orientuotą programinės įrangos kūrimo principą. Turint aibę tokių komponentų, galima juos sujungti į visumą ir panaudoti įvairiose taikomosiose programose bei modeliuoti naujus komponentus. Saityno tarnybų privalumas yra tai, jog modeliuojant programas iš tokių komponentų, nereikia turėti išsamios informacijos apie kiekvieną komponentą, nes jie nėra tvirtai susieti vienas su kitu. Saityno tarnyba yra tarsi savarankiška programa, kuri nepriklauso nei nuo programavimo kalbos, nei nuo vykdymo platformos. Tokia programa turi sąrašą operacijų, kurias gali atlikti ir garantuoti jų aukštą funkcinį laipsnį. Kiekviena operacija turi apibrėžtas įvestis ir išvestis, kurios leidžia suprasti, ką atlieka operacija ir kaip ją iškviesti. Saityno tarnyboms pasiekti yra reikalingas internetas, o bendravimui su jomis yra naudojami plačiai paplitę transporto protokolai: HTTP, SMTP, FTP. Naudojantis WSLD, kuris aprašo saityno tarnybos galimybes, duomenys yra siunčiami SOAP žinutėmis XML formatu [6].

Saityno tarnybos esminiai privalumai:

1. **Saugumas**. Tik saityno tarnyba turi teisę pasiekti duomenų bazę.
2. **Paplitimas**. Saityno tarnybos yra labai plačiai naudojamos, tiek mažose sistemose, tiek didelėse sistemose, kurios naudojasi interneto prieiga.
3. **Paprastumas**. Saityno tarnyba atlieka tik tam tikrą užduočių sąrašą.
4. **Duomenų bazės apkrovimo sumažinimas**. Gauti duomenys iš duomenų bazės gali būti laikinai saugomi tam skirtame podėlyje (angl. *cache*).
5. **Gebėjimas toleruoti gedimus**. Saityno tarnyba gali persijungti nuo pagrindinio duomenų šaltinio prie atsarginio.
6. **Išplečiamumas** (angl. *scalability*). Saityno tarnyba gali paskleisti užklausas tarp paralelinių duomenų struktūrų, neturint detalaus aprašo, kurią struktūrą jam pasirinkti.
7. **Pasiskirstymas tinkle**. Saityno tarnybos gali būti plačiai išsidėsčiusios tinkle.
8. **Sandarumas** (angl. *encapsulation*). Saityno tarnybai galima keisti pagrindinius metodus, kurie bendrauja su duomenų baze, nedarant poveikio saityno tarnybos vartotojams.

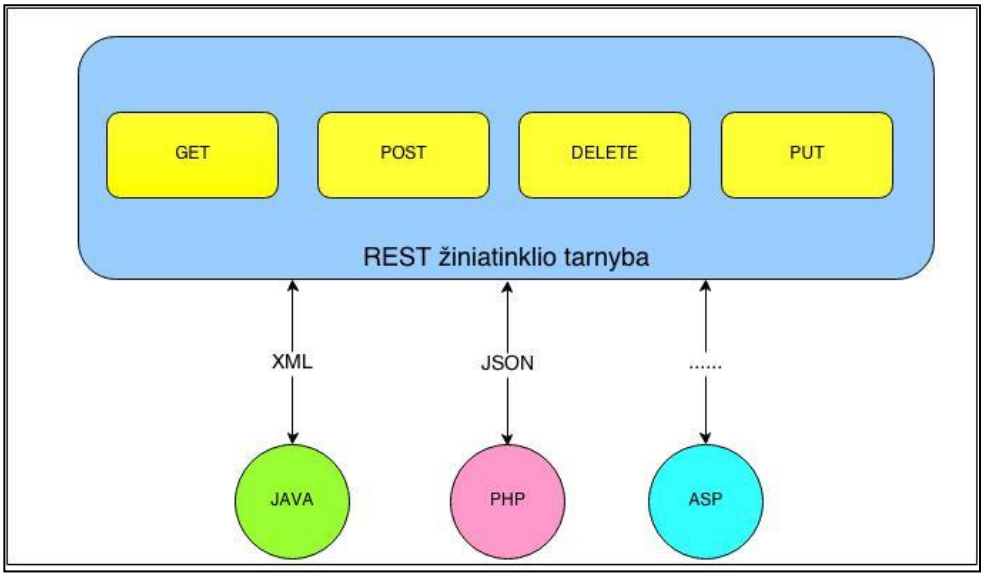
2.3.1. RESTful saityno tarnyba

REST (angl. *Representational State Transfer*) yra alternatyvus architektūrinis stilius, skirtas saityno tarnyboms (angl. *web service*) kurti, gaunant ir manipuliuojant ištekliais su nustatytais standartiniais HTTP metodais, tokiais kaip GET, POST, PUT bei DELETE. Šis architektūros stilius yra ganėtinai naujas ir vis labiau plintantis rinkoje. Iš esmės visos naujausios saityno tarnybos, priklausančios tokioms kompanijoms, kaip *Google, Yahoo, Amazon, Microsoft,* yra paremtos REST stiliumi [7]. Šis architektūrinis stilius buvo sukurtas motyvuojant tuo, kad viskas gali būti padaryta daug paprasčiau negu yra dabar. Pirminėje architektūroje yra teigiama, jog saityno tarnybos remiasi tuo, kad naudojant HTTP vyksta XML duomenų mainai, panaudojant SOAP žinutes [8]. Tačiau saityno tarnybos specifikacijos griežtai nesako, kad SOAP žinutės turi būti siunčiamos per HTTP. Nagrinėjant toliau buvo pagalvota ar apskritai SOAP yra reikalingas šiuo atveju.

Buvo nuspręsta, kad SOAP ir XML reikalauja daug resursų ir pastebėta, jog HTTP suteiktomis galimybėmis komunikavimas gali būti atliekamas ir kitais metodais. HTTP suteikė daug galimybių formuojant REST stilių, nes turėjo komponentų, tokių kaip URL ir kt., kurie galėjo būti panaudoti komunikacijai. 2000 metais vienas iš pagrindinių HTTP specifikacijų autorių R. Fieldingas parašė disertaciją pavadinimu „*Architectural Styles and the Design of Network-based Software Architectures*“ [9], kurioje pristatė REST architektūrinį stilių. Remiantis autoriaus disertacija, galima bendrai suformuluoti REST veikimo principą. RESTful saityno tarnyba priima REST stiliaus pranešimus, kurių metu konkrečius resursus identifikuoja URI. Naudojant HTTP GET užklausa yra gaunamas resursų turinys. HTTP PUT arba POST užklausa yra naudojama resursų pakeitimui arba naujo

sukūrimui. Kuriant naujus resursus, užklausos aprašyme yra nurodomas turinys XML arba kitu formatu. Resursų panaikinimui yra naudojama HTTP DELETE užklausa. Pirmajame paveikslėlyje (žr. 2.5 2.5 pav.) yra pavaizduota REST stiliaus saityno tarnybos koncepcija. Toliau yra pateikiamos REST stiliumi paremtos saityno tarnybos pagrindinės savybės, nusakančios, kodėl šis architektūrinis stilius tampa toks populiarus:

- Lengvi rezultatai – gauti atsakymai iš saityno tarnybos turi nedaug perteklinės informacijos, tai suteikia daugiau greičio bendraujant su tarnyba.
- Atsakymai gauti iš saityno tarnybos yra lengvai skaitomi, nes yra naudojami standartiniai formatai (XML, JSON ir kt.).
- Lengvai ir greitai kuriamas.
- Nereikia rūpintis, kas yra vartotojo dalyje, kokiomis technologijomis yra realizuota jo sistema.
- Didelė dalis rinkoje egzistuojančių saityno tarnybų naudoja šį architektūrinį stilių, todėl atsiranda galimybė lengvai bendrauti su jomis.
- Vartotojui tereikia žinoti URI ir toliau pasirinkęs užklausos tipą, jis gali kreiptis į reikiamą saityno tarnybos operaciją.



2.5 pav. REST architektūriniu stiliumi paremta saityno tarnyba

2.3.2. *ServiceStack* ir *ASP.NET Web API* palyginimas.

Išsiaiškinus kas yra saityno tarnyba (angl. *web service*) ir REST stilius, buvo atliekama rinkoje egzistuojančių produktų analizė, kuriais galima realizuoti REST tarnybas. Buvo lyginami du produktai, tai *ASP.NET Web API* ir *ServiceStack*. Naudojantis šias struktūras aprašančiais šaltiniais [10] [11] buvo suformuota lentelė (žr. 2.1 2.1 lentelė.), kurioje yra pateikiama informacija apie tai, kokias galimybes siūlo šios dvi struktūros. Pirmoje lentelėje yra pateikiamos tik tos siūlomos galimybės, kurios yra aktualiausios renkantis vieną iš jų.

2.1 lentelė. *ASP.NET Web API* ir *ServiceStack* siūlomos galimybės

ASP.NET Web API	**ServiceStack**
Panašus programavimo stilius į MVC	Yra sugeneruojamas metaduomenų puslapis, kuriame pateikiamos visos saityno galimybės ir operacijos
Priegloba (angl. *hosting*) yra ribota	Priegloba (angl. *hosting*) gali būti įvairių tipų - *Website, Console App, Windows Service*
Lengva naudotis ir greitai perprantama	Vienas greičiausių „*serializer*“.
Palaiko JSON, XML ir *FormData* turinio tipus	Daug papildomų įskiepių (angl. *Plug-in*), kurie suteikia papildomo funkcionalumo
Iš karto yra integruotas Visual Studio šablonuose	Integruoti tipai duomenims ir turinio deryboms (angl. content negotiation): JSON, XML, CSV, JSV, SOAP, *ProtoBuf*, HTML, *Text, byte[], Stream, IStreamWriter*
Neseniai išleista į rinką struktūra	Jau ilgą laiką egzistuojanti rinkoje struktūra

ServiceStack – tai atviro kodo struktūra, kuri suteikia galimybę realizuoti REST, SOAP ir kt. saityno tarnybas, naudojant C# programavimo kalbą. *ServiceStack* yra alternatyva tokioms struktūroms, kaip WCF/REST, Web-API, MVC ir kt. Ši struktūra turi daug naujų bibliotekų, kurios leidžia pasirūpinti tiek kuriamos saityno tarnybos saugumu, tiek resursų apdorojimu, valdymu ir kt. Realizuota saityno tarnyba naudojant šią struktūrą gali veikti tiek MONO struktūroje, tiek .NET, todėl jis gali veikti, tiek *Linux,* tiek *Windows* serveryje. Ši struktūra suteikia nemažai privalumų, tačiau apibendrinant *ServiceStack* kūrėjai labiausiai išskiria šiuos struktūros bruožus :

- Paprastumas ir intelektualumas – naudojant tvirtai aprašytus DOT, struktūra savarankiškai sugeneruoja metaduomenų puslapį, kuriame matomos visos tarnybos galimybės (WSDL, operacijos ir kt.)
- Nereikia jokių XML nustatymų (.config failų).
- Supaprastina ir C# programavimą, ir HTTP protokolą.
- Realizavus saityno tarnybą, ji iš karto palaiko XML, JSON, SOAP ir kt.
- Daugiaplatformiškumas

2.4. Vartotojo sąsaja

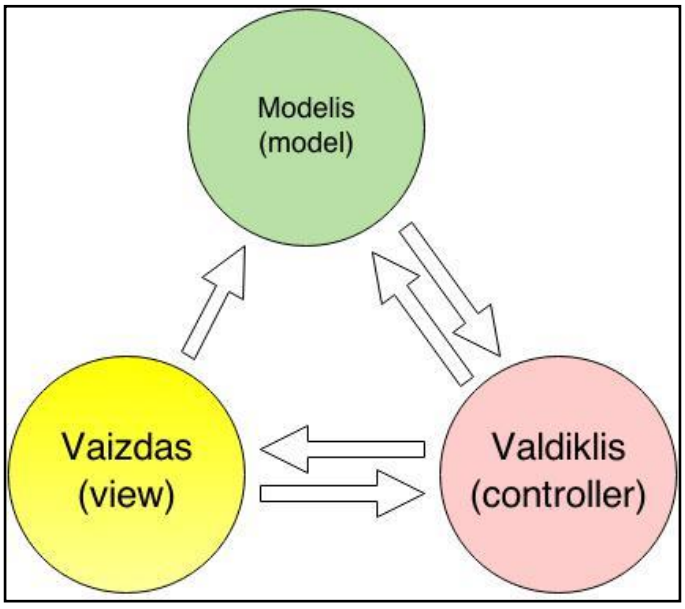
Šiame poskyryje bus analizuojami technologiniai sprendimai, skirti dinamiškų vartotojo sąsajų realizacijai. Pradžioje bus analizuojamas MVC šablonas ir *ASP.NET MVC 4* struktūra.

2.4.1. *ASP.NET MVC 4* apžvalga

Pradžioje bus analizuojama iš kokių komponentų susideda MVC šablonas, už ką jie yra atsakingi ir kaip veikia šis šablonas. Išsiaiškinus apie MVC šabloną bus atliekama detalesnė *ASP.NET MVC* 4 analizė, kurioje bus analizuojama, kokias galimybes siūlo ši struktūra ir kodėl ji buvo pasirinkta programuojant vartotojo sąsają.

MVC (angl. *model – view – controller*), tai struktūrinis šablonas, skirtas vartotojo sąsajos realizacijai (angl. *design pattern*), kuris susideda iš trijų dalių, atsakingų už skirtingus darbus. Tokiu būdu yra atskiriama vartotojo sąsajos logika nuo verslo logikos (angl. *business logic*).

- Modelis (angl. *model*) – atsakingas už duomenis, jų logiką ir funkcijas, kurios naudojamos duomenų apdorojimui.
- Vaizdas (angl. *view*) – ši dalis atsakinga už tai, kaip duomenys bus pateikiami vartotojui.
- Valdiklis (angl. *controller*) – priima įvestis iš vartotojo ir, remdamasis jomis, kontroliuoja modelį ir vaizdą, kurį mato vartotojas.



2.6 pav. MVC šablono architektūra

Naudojant šį šabloną internetinė programa yra tarsi suskaidoma į tris atskiras dalis ir taip leidžia programuotojui koncentruotis į kiekvieną dalį atskirai (žr. 2.6 pav.). Pavyzdžiui, programuotojas gali dirbti su atvaizdavimo dalimi, negalvodamas apie logines operacijas su duomenimis, nes tai yra atliekama modelio dalyje. Tokiu būdu kiekviena sistemos dalis tampa atsakinga tik už tai, kas jai

priklauso. Tokia sistema tampa struktūrizuota ir lengvai suprantama programuotojui. Šiuo metu šis šablonas yra labai paplitęs įvairiose struktūrose, tokiose kaip *PHP, APS.NET, Python, Java* ir kt. [12]

Vartotojo sąsajai realizuoti buvo pasirinkta *ASP.NET MVC 4* struktūra, nes visas projektas bus paremtas Microsoft technologijomis, tiek saityno tarnyba (angl. *web service*), tiek vartotojo sąsaja. Esant tokiam vieningam technologiniam sprendimui bus išvengta daug problemų, susijusių su tarnybos ir sąsajos suderinamumu. Analizuojant *ASP.NET MVC 4* struktūrą buvo rasta alternatyva- *ASP.NET Web Forms*, tačiau, išsiaiškinus MVC šablono teikiamus privalumus, ši alternatyva nebuvo pasirinkta. *ASP.NET MVC 4* struktūra turi integruotą platų bibliotekų pasirinkimą, leidžiantį nesudėtingai išspręsti problemas, susijusias tiek su saityno tarnybos komunikacija, tiek su vartotojo sąsajos realizacija. MVC šablono dėka ši struktūra išskaido vartotojo sąsają į atskiras dalis, tokiu būdu projektas tampa struktūrizuotas. Tokios struktūrizuotos sistemos atliekamų veiksmų seka tampa panaši, o tai leidžia naudoti abstrakčius metodus, taip sumažinant kodo apimtį. Visą tai suteikia galimybę efektyviau planuoti programavimo darbus.

Microsoft kompanija jau ilgą laiką tobulina *ASP.NET MVC* struktūrą, pradedant nuo pirmosios versijos, išleistos 2007 metais ir tęsiant jos tobulinimą. Šiuo metu yra išleista *ASP.MVC* 5.1.2 versija. Su kiekviena versija yra suteikiama vis daugiau galimybių, tačiau pasirinktoje *ASP.NET MVC 4* struktūroje pagrindinės galimybės, kurias išskiria *Microsoft* [13] yra šios :

- Kartu su *ASP.NET MVC 4* yra integruota *ASP.NET Web API* struktūra, kuri leidžia kurti saityno tarnybas.
- Atnaujintas svetainės dizaino šablonas, kuris dar labiau pagražina svetainę ir automatiškai prisitaiko tiek prie įprastos naršyklės, tiek prie mobilių įrenginių.
- Atsirado naujas svetainės dizaino šablonas, kuris skirtas tik mobiliems įrenginiams.
- Naujas svetainės atvaizdavimo rėžimas, kuris leidžia sistemai pasirinkti, kokį vaizdą rodyti vartotojui atsižvelgiant į tai, kokio tipo naršykle yra naudojamasi.
- Daug įvairių kitų privalumų kuriant internetines svetaines mobiliesiems įrenginiams.
- *Azure* SDK palaikymas.
- Pridėta nauja *Entity Framework* 5 versija.
- Galimybė valdiklius patalpinti į bet kokį projekto katalogą.
- Susiejimo (angl. *bundling*) ir mažinimo (angl. *minification*) funkcijos, mažinančios HTTP užklausų skaičių, kurių metu yra gaunami įvairūs šaltiniai, tokie kaip *CSS* ar *JavaScript*. Dabar šie šaltiniai yra suspaudžiami į vieną failą.
- Atsiranda galimybė naudoti Facebook ar kitų puslapių, kurie naudoja *OAuth* ir *OpenID,* prisijungimus.
- Ir kt.

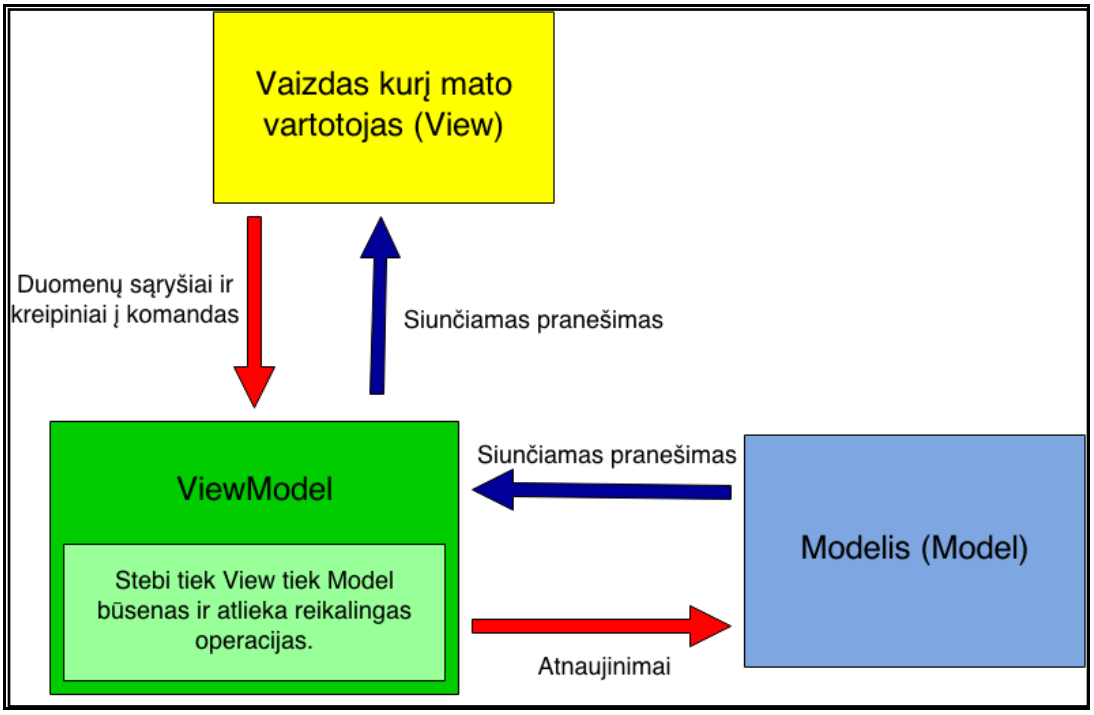
2.4.2. *AngularJS* ir *Knockout* palyginimas

Remiantis technine užduotimi, kurioje yra reikalaujama, kad vartotojo sąsaja būtų dinamiška, yra pasirinktas MVVM šablonas. Šiame poskyryje bus aprašomas šio šablono veikimo principas ir atliekama rinkoje egzistuojančių technologinių sprendimų lyginamoji analizė, kurios pagalba bus pasirinktas vienas iš produktų.

MVVM, tai architektūrinis šablonas, kuris yra panašus į MVC šabloną. Šis šablonas skirtas realizuoti modernias vartotojo sąsajas. Šablonas suteikia tarsi atskirus lygius, kurie atskiria vartotojo matomą vaizdą nuo naudojamos logikos, sprendžiančios, kaip tas vaizdas turi būti atvaizduojamas. Jis susideda iš trijų dalių [14]:

- Modelis (angl. *model*) – atsakingas už informacijos saugojimą. Jis tarsi atstovauja duomenų struktūrą. Ši dalis neatsako už pačių duomenų atvaizdavimo formatą. Vienintelė išimtis yra duomenų patikrinimas (angl. *validation*).
- Vaizdas (angl. *view*) – atsakingas už duomenų atvaizdavimą vartotojui naudojant HTML ir kt. taip pat, kokiais formatais bus pateikiami duomenys, kurie yra modelyje. Būtent per šią dalį vartotojas valdo ir stebi informaciją. MVVM šablone vaizdas tampa dar labiau interaktyvus palyginus su MVC šablonu.
- *ViewModel* – atsakingas už tam tikrą logiką ir sąryšius, susijusius su duomenimis aprašytais modelyje bei atvaizduotais duomenimis vartotojui. Iš esmės jį galima sulyginti su MVC šablono valdikliu, kuris šiuo atveju stebi vartotojo įvestis, ir modelyje aprašytus duomenis bei įvykus pakitimams vienoje iš šių dalių jis viską atnaujina.

Trečiame paveikslėlyje (žr. 2.7 pav.) detaliai pateikta, kaip veikia šis šablonas. Vartotojas mato vaizdą savo ekrane, kuriame esantys įvesties laukai turi sąryšius su modeliu. Už šiuos ryšius atsako *ViewModel*. Vartotojui keičiant įvestis vaizde, *ViewModel* gauna informaciją apie šiuos vartotojo veiksmus ir siunčia atnaujintus duomenis į modelį bei taip atnaujina jį. Viskas gali vykti ir kita kryptimi, tai reiškia, kad įvykus pakitimams modelyje, yra siunčiamas pranešimas į *ViewModel*, kuris sąryšių pagalba atnaujina atitinkamus elementus vartotojo lange.



2.7 pav. MVVM šablono veikimo principas

Išsiaiškinus MVVM šablono veikimo principą buvo atliekama rinkoje egzistuojančių produktų analizė, kurių pagalba galima realizuoti dinamiškas vartotojo sąsajas. Buvo lyginami du produktai, tai *AngularJS* ir *Knockout*. Remiantis šaltiniu [15] buvo sudaryta lentelė (žr. 2.2 lentelė), kurioje yra pateikiama informacija apie tai, kokias galimybes siūlo šios dvi *JavaScript* bibliotekos.

2.2 lentelė. *AngularJS* ir *Knockout* MVVM bibliotekų palyginimas

AngularJS	***Knockout***
MVVM architektūra	
Palaiko visas naršykles iki IE8	Palaiko visas naršykles iki IE6
Palaiko priklausomybių injekcijas (angl. *dependency injection*)	Stebi duomenų pakitimus apjungiant visus kintamuosius į *"observable function*", taip sukurdamas priklausomybių grafiką, kuris automatiškai kontroliuoja tolimesnius veiksmus
DOM paremtas atvaizdavimo variklis (angl. *templating engine*)	
Turi testavimui skirtų įrankių	Neturi testavimui skirtų įrankių
Reikia daugiau laiko, siekiant perprasti veikimo principus	Paprasta naudotis, aiški dokumentacija su daug pavyzdžių
Maršrutai (angl. *routing*)	-
Turi integruotą duomenų tikrinimą (angl. *validation*)	Reikia papildomos bibliotekos, siekiant (angl. *Plugin*) patikrinti duomenis

Atlikus šių dviejų *JavaScript* bibliotekų palyginimą, galima daryti išvadas jog *Knockout* yra labiau tinka į mažos apimties projektus, taip pat nereikalauja daug žinių, siekiant perprasti kaip ji veikia todėl projektui buvo pasirinkta *Knockout* biblioteka, kuri padės spręsti tam tikras problemas UI dalyje.

2.5. SQL ir NoSQL duomenų bazių palyginimas

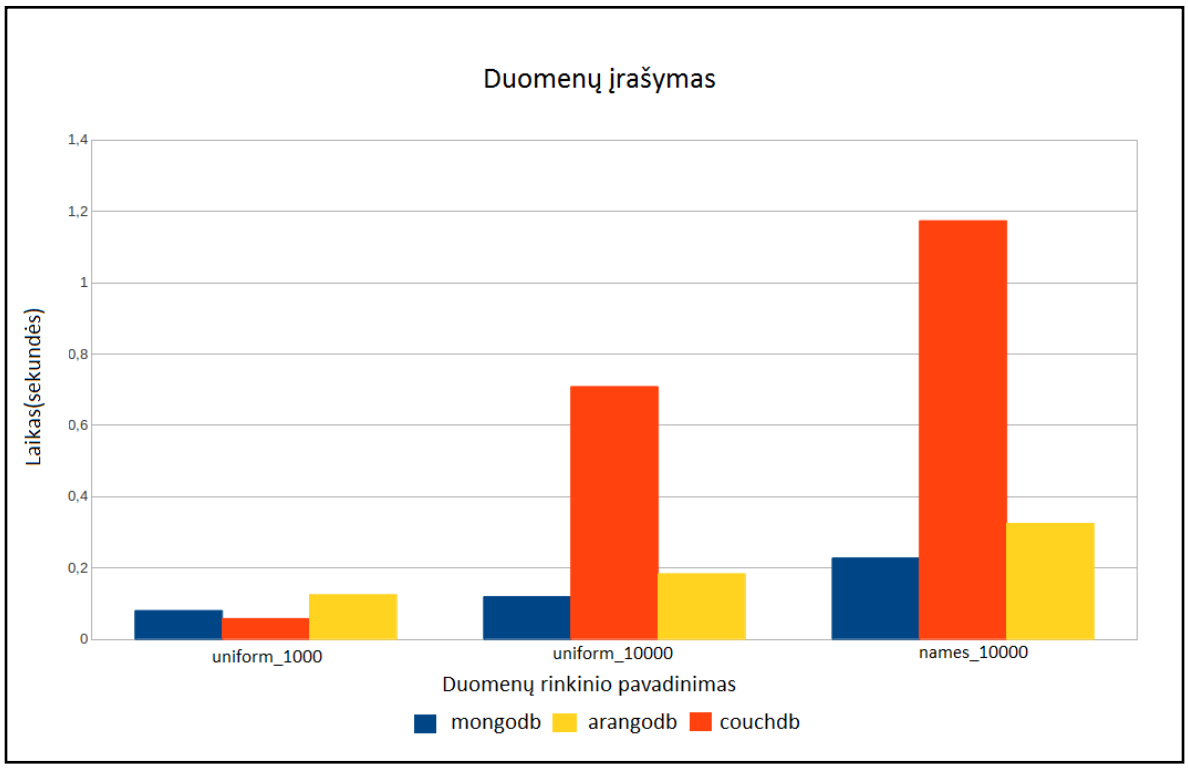
NoSQL, tai toks duomenų bazės tipas, kuris apima įvairias technologijas, leidžiančias efektyviau tvarkyti vartotojo duomenis nei SQL DB tipu. Tačiau šis efektyvumas ne visada vienodai suprantamas ir jaučiamas vartotojui. Bendruoju atveju NoSQL duomenų bazės yra orientuotos į tokius projektus, kuriuose yra daug duomenų ir greitis dirbant su jais turi būti didelis. NoSQL atsisako sudėtingų sąryšių saugojimo tarp įrašų, taip pat duomenų schemos nėra fiksuotos, priešingai nei reliacinės duomenų bazės (SQL). SQL duomenų bazėje norint patalpinti duomenis, prieš tai turi būti aprašyta duomenų schema ir tik tada duomenys galės būti talpinami. NoSQL duomenų schemos yra dinamiškos [16]. Detalesnis SQL ir NoSQL palyginimas yra pateiktas 2.3 lentelė lentelėje.

2.3 lentelė. SQL ir NoSQL palyginimo lentelė

	SQL duomenų bazės	**NoSQL duomenų bazės**
Tipai	Vieno tipo (SQL) duomenų bazės	Siūlo įvairius tipus, tokius kaip: *key-value* – kiekvienas įrašas saugomas kaip atributas kartu su savo reikšme. *document databases* – saugomo įrašo reikšmė yra dokumentas susidedantis iš atributų ir reikšmių rinkinio. *wide-column stores* - duomenys saugomi struktūrizuotai, stulpeliais. Duomenų įrašas susideda iš dinamiškų stulpelių. *graph databases* – šis tipas naudojamas saugoti informacijai apie tinklą
Sukūrimo data	1970 metais	2000 metais SQL pasiekė tam tikrus limitus, susijusius su praplečiamumu, nestruktūrizuotų duomenų saugojimu ir kt.
Pavyzdžiai, kas naudoja	*MySQL, Postgres, Oracle Database*	*MongoDB, Cassandra, HBase, Neo4j*
Duomenų modelis	Kiekvienas įrašas yra saugomas kaip eilutė lentoje, kurio stulpelis saugo tam tikrą dalį įrašo	Įvairus, priklausomai nuo DB tipo
Duomenų schema	Struktūra ir duomenų tipai yra fiksuoti	Dinaminė schema
Plėtros modelis	Maišytas, tiek atviro kodo (pvz. *MySQL*), tiek uždaro (pvz. *Oracle Database*)	Atviro kodo
Duomenų valdymas	Naudojant klasikines SQL užklausas	Naudojantis objektiniu programavimu orientuota programavimo sąsaja

Siekiant pasirinkti tinkamą duomenų bazę, buvo atliekama rinkoje egzistuojančių NoSQL duomenų bazių lyginamoji analizė, siekiant įvertinti ar NoSQL tinkama kuriamam projektui. Pasirinkti produktai, tai *ArangoDB, MongoDB* bei *CouchDB*. Remiantis *ArangoDB* kompanijos testavimais [17], yra pateiktas grafikas, kuris nurodo, kaip skirtingos duomenų bazės vykdo duomenų įrašymą (žr. 2.8 pav.). Į duomenų bazes yra įrašomi skirtingų dydžių duomenų rinkiniai. Pateiktoje diagramoje (žr. 2.8

pav.) matyti, kad su mažesniais duomenų rinkiniais (nuo 1000 iki 10000) *MongoDB* įrašymas atliekamas greičiau, lyginant su kitomis DB.



2.8 pav. Duomenų įrašymo palyginimas

Didinant duomenų apimtis buvo pastebėta, kad duomenų įrašymą greičiau atlieka *ArangoDB*. Iš pateikto grafiko galima daryti išvadą, kad *MongoDB* duomenų bazė labiau tinka projektams, kuriuose nėra atliekami didelės apimties duomenų įrašymai. Projektuose, kuruose įrašomų duomenų apimtys yra didelės, geriau naudoti *ArangoDB*. *CouchDB* duomenų bazė atliktuose testuose pasirodė prasčiausiai, nes naudojant ją, esant didesniems duomenų kiekiams, įrašymo laikas ženkliai pailgėja.

MongoDB yra viena iš pirmaujančių atviro kodo NoSQL duomenų bazių [18], kurios įrašai yra dokumentai, susidedantys iš atributų ir reikšmių porų. Dokumentai panašūs į JSON objektus ir kiekviena reikšmė dokumente gali saugoti ne tik paprastas reikšmes, bet ir kitus dokumentus ar net dokumentų masyvus.

Atlikus duomenų bazių analizę, buvo pasirinkta SQL tipo duomenų bazė, nepaisant NoSQL tipo duomenų bazių privalumų SQL suteikia tvirtus sąryšius bei galimybe naudotis ORM įrankiais, tokiais kaip *EntityFramework*. Sistemoje nebus atliekami didelės apimties duomenų įrašymai, todėl poreikio naudoti nestandartinio tipo duomenų bazės nėra.

2.6. Programinės įrangos metrikos

Programinės įrangos metrikos (PI) – tai kiekybinis programinės įrangos įvertinimas, kuris gali būti naudojamas kaip vienas iš reikalavimų programinei įrangai. PI metrikos plačiai naudojamos tiek

tarp PI kūrėjų, tiek tarp sistemos vartotojų. PI užsakovai dažnai pateikia tam tikras programinės įrangos metrikas, kurias turi atitikti kuriama programinė įranga. Metrikos gali būti skaičiuojamas keliais būtai ir naudojant kelis algoritmus, tačiau dauguma rinkoje egzistuojančiu produktų, naudoja Halstead‘o technologijas.

Programinės įrangos metrikos skirstomos į tris pagrindines grupes:

- Produkto metrikos
- Proceso metrikos
- Projekto metrikos

Programinės įrangos kokybė yra dalis programinės įrangos metrikų. Siekiant įvertinti programinės įrangos kokybę yra naudojamos visos metrikos iš minėtų grupių, tačiau kokybės metrikos labiausiai siejamos su produkto ir proceso metrikomis. Nepaisant to, jog kokybė yra labiausiai siejama su produkto ir proceso metrikomis, įtakos kokybei gali turėti ir projekto metrikos: įmonės dydis, programuotojų skaičius ir kt. Apibendrinant, programinės įrangos kokybė turi būti vertinama visos programinės įrangos gyvavimo cikle, susiejant projekto kūrimo dalis su procesais ir galutinio produkto parametrais bei metrikomis. [19]

Produkto metrikos apibūdina galutines programinės įrangos charakteristikas, tokias kaip programos dydį, sudėtingumą, greitaveiką ir kokybės lygį. Ši metrika leidžia spręsti apie galutinio produkto kiekybinį įvertį. Šios metrikos padeda inžinieriams geriau suprasti programinės įrangos struktūrą. Šios metrikos, naudojant griežtai apibrėžtas taisykles, leidžia sistemingai įvertinti PI kokybę.

Proceso metrikos apibūdina programinės įrangos kūrimo procesus ir yra susijusios su proceso kokybe. Proceso metrikos naudojamos įvertinti įvairių procesų efektyvumą ir veiksmingumą. Vienos iš metrikų, priklausančių šiai grupei, gali būti: kokybės kaina (angl. *Cost of quality*), defektų pašalinimas (angl. *defect removal efficiency*), testavimo efektyvumas (angl. *testing efficiency*) ir kt.

Projekto metrikos apibūdina programavimo procesą viso programinės įrangos gyvavimo ciklo metu. Šiai grupei yra priskiriamos šios metrikos: kaina, programavimo tvarkaraščiai, produktyvumas, programuotojų skaičius ir kitos su resursais susijusios metrikos. Paprastai šiai grupei priskiriamos metrikos, geriausiai apibūdinančios komandos darbą ir projekto gyvavimo ciklą. [20]

2.7. Programinės įrangos kodo metrikos

Šiame skyriuje pateikiama programinės įrangos kodo metrikų analizė, kurios metu analizuojamos metrikos, leidžiančios atlikti PI kokybinį vertinimą. Analizėje pateiktos metrikos buvo naudojamos eksperimentinėje dalyje, kurios metu buvo įvertinta SOA architektūra paremta sistema.

2.7.1. Kodo eilučių skaičius

Kodo eilučių skaičius yra viena dažniausiai naudojamų metrikų programinės įrangos kūrime. Literatūroje ši metrika žymima kaip LOC (angl. *lines of code*). Ši metrika leidžia įvertinti programinės įrangos dydį eilučių skaičiumi, tačiau viena iš pagrindinių problemų yra nuspręsti, kurios kodo eilutės yra tinkamos, o kurios ne. Vis tobulėjant programavimo kalboms ši problema tampa didesnė, nes siekiant atlikti tas pačias logines operacijas skirtingose kalbose, reikia skirtingo skaičiaus eilučių. Siekiant teisingai paskaičiuoti kodo eilutes reikia nuspręsti, kas bus traktuojama kaip programinio kodo eilutė, o kas ne. Žemiau yra pateikiami galimi būdai, kaip tai galima padaryti:

- Skaičiuojant tik vykdomas eilutes
- Skaičiuojant vykdomas eilutes ir duomenų priskyrimus
- Skaičiuojant vykdomas eilutes, duomenų priskyrimus ir komentarus
- Skaičiuoti eilutes įvedimo ekrane

Įrankiuose, skirtuose matuoti programinės įrangos kodo metrikas, yra naudojami skirtingi kodo eilučių matavimo metodai. Dažniausiai įrankius pasirenka programavimo komanda pagal savo vidinius įmonės standartus arba vadovaujasi gautais reikalavimais iš užsakovo. Šis aspektas yra labai svarbus, nes skirtingi įrankiai skaičiuodami LOC gali įtraukti komentarus arba skaičiuoti fizines eilutes vietoje loginių. Taikant skirtingus metrikų matavimo įrankius gautume skirtingas reikšmes.

Loginės eilutės aprašo konkrečias programos logines operacijas. Dažniausiai tai būna loginės operacijos, tokios kaip *for, if, print, while* ir tt. Priešingai nei fizinės eilutės, šios eilutės tiksliau apibrėžia programos apimtį. Fizinės eilutės aprašo tokias eilutes, kurios gali būti sudarytos ir iš kelių arba iš vienos loginės eilutės. Naudojant fizines eilutes programos apimtis tampra priklausoma nuo programos kodo stilistikos. [19]

Apibendrinant galima pastebėti, jog taikant skirtingus kodo eilučių skaičiavimo algoritmus galime gauti skirtingus duomenis. Tai parodo, koks netikslus gali būti kodo eilučių skaičiavimo metodas.

2.7.2. Ciklomatinis sudėtingumas

Ciklomatinio sudėtingumo metrika yra viena labiausiai naudojamų metrikų. Šią metriką išrado Tomas J. McCabe 1976 m. Ši metrika yra viena tiksliausiai paskaičiuojamų ir yra labai plačiai naudojama įrankiuose skirtuose skaičiuoti PI metrikas. Ši metrika skirta įvertinti programinės įrangos išeities kodo sudėtingumą. Ši metrika yra kilusi iš grafų teorijos ir dažnai literatūroje apibrėžiama išeities kodo visų grafo kelių skaičiumi. Ciklomatinis sudėtingumas yra išreikštas ir matematine išraiška:

$$M=E-N+2P \qquad (1)$$

- M – ciklomatinis sudėtingumas
- E – briaunų skaičius grafe
- N – viršūnių skaičius grafe
- P – sujungtų komponentų skaičius

Turint matematinę išraišką nesunkiai galima paskaičiuoti ciklomatinį sudėtingumą, tačiau norint tai padaryti, reikia sudaryti grafą. Sudarytas grafas turi atvaizduoti visus programinės įrangos logines operacijas – nepriklausomus kelius. Turint grafą būtų galima gauti viršūnių, briaunų ir sujungtų komponentų skaičius, kuriuos įstačius į formulę gautume programos ciklomatinį sudėtingumą. Praktikoje sudarinėti grafiką yra gana sunkus procesas todėl ši formulė turi kitą išraišką, kurios pagalba paskaičiavimas yra kur kas lengvesnis:

$$M=\#IFS+\#Loops+1 \quad (2)$$

- #IFS – sąlyginių sakinių skaičius,
- #Loops – ciklų skaičius programoje.

McCabe rekomendacijose yra pateikiama ciklomatinio sudėtingumo maksimali skaitinė reikšmė – 10. Programinės įrangos išeities kodui viršijant šią reikšmę programos kodas traktuojamas kaip sudėtingas. Šiuo atveju programos kodą reikia skaidyti į atskirus metodus arba supaprastinti. Skaičiuoti ciklomatinį sudėtingumą funkcinėms programavimo kalboms yra kur kas lengviau negu objektinėms programavimo kalboms. Visais atvejais ciklomatinis sudėtingumas skaičiuojamas ne visai programai, o atskiroms programoms dalims. Programos ciklomatinio sudėtingumo skaičiavimų rezultate yra pateikiama vidutinė sudėtingumo reikšmė [19].

2.7.3. Priežiūros indeksas

Priežiūros indeksas, tai išvestinė metrika, skirta matuoti programinės įrangos išeities kodo priežiūrą. Ši metrika susideda iš metrikų, tokių kaip: ciklomatinis sudėtingumas, kodo eilučių skaičius ir Halstead pastangų arba apimties. Skirtinguose šaltiniuose yra pateikiamos šiek tiek kitokios priežiūros indeksą leidžiančios paskaičiuoti formulės. Vienose priežiūros indekso formulėse yra naudojama Halstead pastangų (angl. *effort*) metrika, kituose Halstead apimties (angl. *volume*). Prižiūrimumo indeksas skaičiuojamas atskiriems programos komponentams [21]. Visos programos prižiūrimumui įvertinti turi būti naudojamos vidutinės atskirtų komponentų reikšmės. Microsoft Visual Studio aplinkoje norint paskaičiuoti programos priežiūros indeksą yra naudojama formulė, pateikta toliau:

$$\text{MAX}(0,(171 - 5.2 * \log(HV) - 0.23 * (CC) - 16.2 * \log(LOC))*100 / 171) \quad (3)$$

- MI – priežiūros indeksas
- HE –Halstead‘o apimtis

- CC – ciklomatinis sudėtingumas
- LOC – kodo eilučių skaičius

Šios formulės pagalba yra gaunama priežiūros indekso reikšmė, kurios pagalba galima daryti išvadas apie programos kodo priežiūrą. Svarbu paminėti tai, kad skirtinguose šaltiniuose yra pateikiami skirtingi intervalai, kurie nusako priežiūros indekso vertes. Žemiau patiektoje lentelėje (žr. 2.4 lentelė) yra pateikiamos ribinės reikšmės, kurios yra nustatytos *Microsoft* kompanijos.

2.4 lentelė. MI ribinės reikšmės

MI reikšmė	Programos įvertinimas
>= 20	geras prižiūrimumas
10 - 19	vidutinis prižiūrimumas
< 10	blogas prižiūrimumas

2.7.4. Halstead‘o metrikos

Programinės įrangos metrikoms paskaičiuoti yra naudojama įvairių algoritmų bei modelių, tačiau didžiąją dalį šių algoritmų sudaro rinkiniai taisyklių, kurias atrado Maurice Howard Halstead. 1977 metais M.S Halstead‘as pristatė modelius ir funkcijas, skirtas išmatuoti programos sudėtingumą, programavimo pastangas bei galimų klaidų skaičių. Minėtų trijų metrikų skaičiavimas yra paremtas keliais baziniais parametrais, kurie gaunami iš programos kodo. Literatūroje teigiama, jog kompiuterinė programa yra algoritmo realizacija, kuri apibrėžiama operatorių ir operandų skaičiumi [19]. Halstead‘o pateikiamos metrikos gali būti tiek sudėtinės, tiek išvestinės. Visoms metrikoms paskaičiuoti yra reikalingi šie pradiniai parametrai:

- n1 – unikalių operatorių skaičius.
- n2 – unikalių operandų skaičius.
- N1 – operatorių pasirodymo skaičius.
- N2 – operandų pasirodymo skaičius.

Naudojant šiuos pradinius parametrus Halstead išvedė sistemingą funkcijų rinkinį, susidedantį iš šių metrikų:

Programos ilgis – suma operatorių pasirodymo skaičiaus ir operandų pasirodymo skaičiaus. Žymima raide N.

$$N = N1 + N2 \qquad (4)$$

Programos žodynas - unikalių operatorių ir operandų suma. Tai parodo, kiek yra unikalių operatorių ir operandų programos tekste. Žymima raide n.

$$n = n1 + n2 \qquad (5)$$

Programos apimtis - minimalus skaičius bitų, reikalingų dekoduoti programą, arba algoritmo įgyvendinimo dydis. Žymima raide V (angl. *volume*). Ši metrika naudojama priežiūros indekso paskaičiavimui.

$$V = N \log_2(n) \quad (6)$$

Minimali programos apimtis - parodo glaustą programos versiją, reikalingą suprogramuoti kodą, skirtą konkrečiai problemai išspręsti. Žymima raide V*.

$$V^* = (2 + n2) * \log_2(2 + n2) \quad (7)$$

Programos lygis - parodo santykį tarp minimalios programos apimties ir realios programos apimties (angl. *level*). Žymima raide L.

$$L = \frac{V^*}{V} \quad (8)$$

Programos sudėtingumas - Ši metrika yra atvirkščias dydis programos lygiui. Žymima raide D (angl. *difficulty*).

$$D = \frac{n1}{2} * \frac{N2}{n2} \quad (9)$$

Programavimo pastangos – gana abstrakti metrika, tačiau turi funkcinę išraišką. Skaičiuojant yra imamas santykis tarp programos lygio ir programos žodyno arba sudėtingumo ir žodyno sandauga. Žymimas raide E (angl. *effort*). Ši metrika naudojama priežiūros indekso paskaičiavimui.

$$E = D * V \quad (10)$$

3. PROJEKTINĖ DALIS

3.1. Dokumento paskirtis

Architektūros specifikacijos dokumentas pateikia darbuotojų vertinimo programos, pagal SOA architektūrinį modelį, struktūrą architektūriniu požiūriu. Dokumentu galės naudotis sistemą kuriantys ir su sistemos veikimu norintys susipažinti vartotojai. Žmonės, kurie atliks programavimo darbus, galės pasinaudoti šiuo dokumentu siekiant sužinoti kaip sistema veiks, su kokiais komponentais bendraus ir kaip bus realizuota. Kita vartotojų grupė galės peržvelgti dokumentą siekiant suprasti, kokie sprendimai buvo priimti norint realizuoti sistemą ir, galbūt panaudoti sprendimus kitiems projektams.

3.2. Apžvalga

Dokumentas apima sistemos architektūrinę pusę įvairiais atžvilgiais: statiniu, dinaminiu, vartotojo, duomenų, išdėstymo vaizdais. Dokumentą sudaro šie skyriai:

- 3.3 skyrius: Pateikiama architektūra.
- 3.4 skyrius: Aprašo programinės įrangos tikslus ir reikalavimus turinčius esminį poveikį architektūrai.

- 3.5 skyrius: Aprašo panaudojimo atvejus ir jų scenarijus.
- 3.6 skyrius: Aprašo sistemos statinį vaizdą: sistemos išskaidymą į paketus ir jų aprašymą.
- 3.7 skyrius: Aprašo sistemos dinaminį vaizdą naudojant diagramas: sąveikos, būsenos ir veiklos.
- 3.8 skyrius: Aprašo techninę įrangą, kurioje sistema bus patalpinta, vaizdą.
- 3.9 skyrius: Aprašo duomenų bazės modelį.
- 3.10 skyrius: Aprašo architektūros įtaką sistemos priežiūrai, išplėtimui, patikimumui ir kitas kokybės savybes.

3.3. Architektūros pateikimas

Siekiant pilnai pateikti sistemos architektūrą yra naudojami vaizdai, kurių aprašymai yra pateikti žemiau esančioje lentelėje (žr. 3.1 lentelė). Lentelėje pateiktų vaizdų aprašymai nurodo UML standartuose naudojamus elementus.

3.1 lentelė. Architektūrai patekti naudojami vaizdai

Vaizdas	**Aprašymas**	**Modeliavimo elementai**
Panaudojimo atvejų	Apibūdina sistemos panaudojimo atvejus ir jų scenarijus.	Panaudojimo atvejų diagrama (angl. *use case*)
Sistemos statinis	Apibūdina sistemoje naudojamus objektus ir sąryšius tarp jų.	Klasių diagrama (angl. *class*)
Sistemos dinaminis	Apibūdina sistemos dinaminį vaizdą.	Sąveikos diagrama (angl. *interaction*), būsenos diagrama (angl. *state*), veiklos diagrama (angl. *activity*)
Išdėstymo	Apibūdiną aplinką kurioje, sistema veiks bei sąryšius tarp skirtingų aplinkų.	Išdėstymo modelis
Duomenų	Apibūdiną sistemos duomenų objektus bei sąryšius tarp jų.	Duomenų modelis

3.4. Architektūros tikslai ir apribojimai

Reikalavimai, kurie turi įtakos sistemos architektūrai:

- Sistema realizuota vartotojo-serverio principu.
- Sistemos paslaugos turi būti pasiekiamos naudojant REST principus - siunčiant užklausas JSON formatu.
- Duomenų bazė bus tame pačiame serveryje, tačiau architektūrinis sprendimas turi leisti pakeisti DB tipą nereikalaujant didelių pakeitimų visoje sistemoje.
- Sistemos vartotojo sąsaja pateikiama anglų kalba.
- Visų funkcijų vykdymas turi būti greitas – asinchroninių užklausų galimybė.
- Sistema turi užtikrinti, kad vartotojo duomenų teisės nebus pažeidžiamos - duomenys nebus viešinami ar kitaip naudojami.

- Sistema turi gebėti aptarnauti vidutiniškai 50 vartotojų vienu metu.
- Sistema turi gebėti dirbti su duomenų baze, turinčia iki 100 tūkstančiu įrašų.
- Sistema turi funkcionuoti nepriklausomai nuo vartotojo naudojamos platformos.
- Sistemoje turi būti realizuota apsauga nuo pagrindinių internetinių pažeidimų.

Architektūros apribojimai:

- Sistemos projektavimas turi būti atliekamas turint reikalavimų specifikaciją, o įgyvendinimas turi būti atliekamas tik turint aiškią ir detalią programinės įrangos architektūros specifikaciją.
- Architektūra turi remtis SOA ir N-eilės architektūriniais modeliais.
- Sistemos projektavimas turi būti atliekamas naudojant UML diagramas.
- Darbo grupę turi sudaryti: projekto vadovas, programuotojas ir testuotojas.

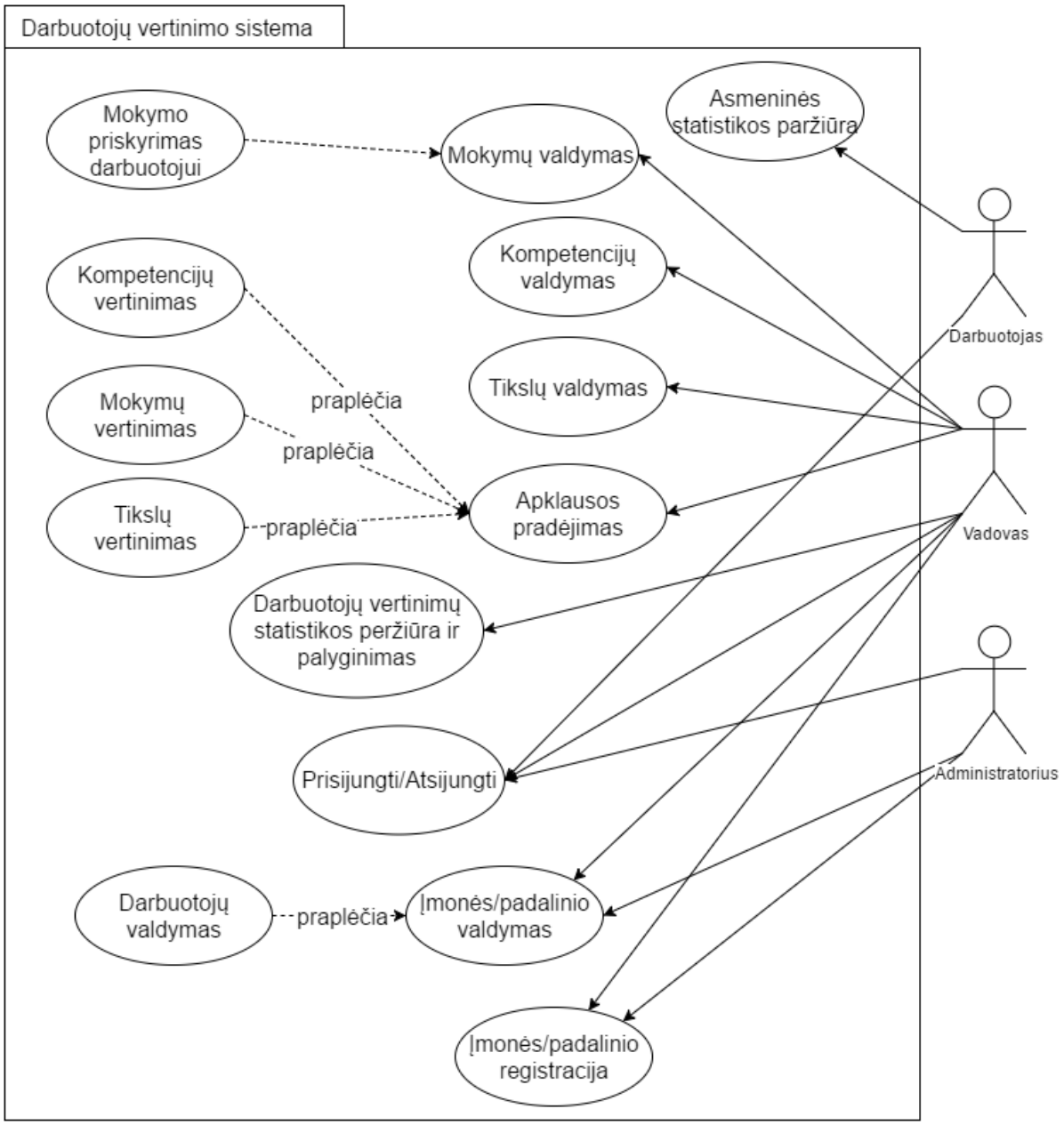
3.5. Panaudojimo atvejų vaizdas

3.5.1. Panaudojimo atvejų modelis

Žemiau yra pateiktas panaudojimo atvejų modelis (žr. 3.1 pav.), kuriame pavaizduoti trys vartotojai, sąveikaujantys su nurodytais panaudojimo atvejais.

Vartotojai:

1. Administratorius – sistemos valdytojas.
2. Vadovas – darbuotojus vertinantis vartotojas.
3. Darbuotojas – vertinamasis vartotojas.



3.1 pav. Panaudojimų atvejų modelis

Žemiau pateiktose lentelėse yra pateikiama kiekvieno panaudojimo atvejo specifikacija, su pagrindiniu jos scenarijumi. Kiekviena lentelė aprašo, kokius veiksmus sistemoje gali atlikti vartotojas ir, kokius rezultatus sistema gali pateikti.

3.2 lentelė. Panaudos atvejis - Kompetencijų valdymas

Panaudos atvejis	Kompetencijų valdymas
Tikslas	Kurti, peržiūrėti, redaguoti ir šalinti kompetencijas sistemoje, pagal kurias padalinio vadovas vertins savo padalinį
Aktoriai	Vadovas
Ryšys su kitais PA	-
Nefunkciniai reikalavimai	Aiškus ir patogus redagavimo-kūrimo langas. Kompetencijų sąrašo atvaizdavimas su galimybe šalinti kompetencijas. Kiekvienas vadovas kuria individualias kompetencijas savo padaliniui, kurios bus naudojamos darbuotojų vertinimams.
Prieš-sąlyga	Kompetencijos atvaizduojamos sąraše; Atvaizduojamas kompetencijos redagavimo langas;
Sužadinimo sąlyga	Mygtuko paspaudimas pagal pasirinktą funkcionalumą norint redaguoti, šalinti, peržiūrėti ar sukurti naują kompetenciją.
Po-sąlyga	Atliekamas veiksmas priklausomai nuo prieš-sąlygos. Šalinimas, redagavimų išsaugojimas, atvaizdavimas, naujos kompetencijos išsaugojimas.
Pagrindinis scenarijus	Vadovas meniu juostoje spaudžia ant kompetencijų nuorodos; Atvaizduojamas vadovo sukurtų kompetencijų sąrašas; Vadovas gali šalinti kiekvieną kompetenciją paspaudęs šalinimo mygtuką; Vadovas gali peržiūrėti ir redaguoti pasirinktą kompetenciją ir paspaudęs redagavimo mygtuką; Vadovas gali sukurti naujas kompetencijas paspaudęs naujo įrašo kūrimo mygtuką; Atvaizduojamas sėkmingai atlikto veiksmo pranešimas;
Alternatyvus scenarijus	Nutrūkus interneto ryšiui vadovo veiksmai gali būti neįvykdomi; Klaidos pranešimo atvaizdavimas; Kuriant ar redaguojant naują kompetenciją yra neužpildomi reikalaujami įvesties laukai;

3.3 lentelė. Panaudos atvejis - Mokymų valdymas

Panaudos atvejis	Mokymų valdymas
Tikslas	Kurti, peržiūrėti, redaguoti ir šalinti mokymus sistemoje, kurie, esant poreikiui, vėliau galės būti priskiriami darbuotojams.
Aktoriai	Vadovas
Ryšys su kitais PA	-
Nefunkciniai reikalavimai	Aiškus ir patogus redagavimo-kūrimo langas. Mokymų sąrašo atvaizdavimas su galimybe šalinti mokymus. Kiekvienas vadovas kuria mokymus, kuriuos bus galima priskirti darbuotojams gerinti jų kvalifikaciją.
Prieš-sąlyga	Mokymų atvaizdavimas sąraše; Atvaizduojamas mokymo redagavimo langas;
Sužadinimo sąlyga	Mygtuko paspaudimas pagal pasirinktą funkcionalumą norint redaguoti, šalinti, peržiūrėti ar sukurti naują kompetenciją.
Po-sąlyga	Atliekamas veiksmas priklausomai nuo prieš-sąlygos. Šalinimas, redagavimų išsaugojimas, atvaizdavimas, naujo mokymo išsaugojimas.
Pagrindinis scenarijus	Vadovas meniu juostoje spaudžia ant mokymų nuorodos; Atvaizduojamas vadovo sukurtų mokymų sąrašas; Vadovas gali šalinti kiekvieną kompetenciją paspaudes šalinimo mygtuką; Vadovas gali peržiūrėti ir redaguoti pasirinktą kompetenciją ir paspaudus redagavimo mygtuką; Vadovas gali sukurti naujus mokymus paspaudęs naujo įrašo kūrimo mygtuką; Atvaizduojamas sėkmingai atlikto veiksmo pranešimas;

Alternatyvus scenarijus	Nutrūkus interneto ryšiui vadovo veiksmai gali būti neįvykdomi; Klaidos pranešimo atvaizdavimas; Kuriant ar redaguojant naują mokymą yra neužpildomi reikalaujami įvesties laukai;

3.4 lentelė. Panaudos atvejis - Darbuotojų valdymas

Panaudos atvejis	Darbuotojų valdymas
Tikslas	Užregistruoti naujus darbuotojus (pavaldinius) sistemoje, kurie bus vertinami. Redaguoti pasirinktą darbuotoją ir pakeisti jo informaciją ar slaptažodį.
Aktoriai	Vadovas
Ryšys su kitais PA	Įmonės/padalinio valdymas
Nefunkciniai reikalavimai	Patogiai išdėstyti įvesties laukai. Įvesties laukų tikrinimas.
Prieš-sąlyga	Priimamas naujas darbuotojas, kurio nėra sistemoje. Arba norima redaguoti darbuotojo informaciją ar slaptažodį.
Sužadinimo sąlyga	Padalinio/įmonės valdymo lange pateikiama nuoroda į naujo darbuotojo registraciją arba pateikiamas egzistuojančių darbuotojų sąrašas.
Po-sąlyga	Sistemoje užregistruojamas naujas darbuotojas, kuris galės stebėti informaciją, susijusia su jo darbo kokybe. Redaguojamo darbuotojo informacija atnaujinama. Pakeitus slaptažodį vadovas turi jį perduoti darbuotojui.
Pagrindinis scenarijus	Įvedami darbuotojo duomenys naujo darbuotojo registracijos lange; Spaudžiama išsaugoti; Sukuriamas naujas darbuotojas, kuris galės prisijungti prie sistemos; Redaguojama informacija; Spaudžiama išsaugoti; Informacija atnaujinta; Redaguojamas slaptažodis (paspaudus redagavimo nuorodą) Spaudžiama išsaugoti; Slaptažodis pakeistas
Alternatyvus scenarijus	Nutrūkus interneto ryšiui vadovo veiksmai gali būti neįvykdomi; Klaidos pranešimo atvaizdavimas; Registruojant darbuotoją neužpildomi reikalaujami įvesties laukai;

3.5 lentelė. Panaudos atvejis - Įmonės/padalinio registracija

Panaudos atvejis	Įmonės/padalinio registracija
Tikslas	Užregistruoti naują įmonę/padalinį, kurios darbuotojai bus vertinami.
Aktoriai	Vadovas Administratorius
Ryšys su kitais PA	-
Nefunkciniai reikalavimai	Pagrindinė įmonė neturi viršesnės įmonės. Padaliniai susieti per pavaldumo sąryšį.
Prieš-sąlyga	Norima registruoti naują įmonę ar įmonės padalinį.
Sužadinimo sąlyga	Naujos įmonės/padalinio registracijos nuorodos paspaudimas
Po-sąlyga	Sistemoje yra užregistruojama nauja įmonė/padalinys. Užregistravus reikia užregistruoti ir vadovą atsakingą už įmonę/padalinį. Vadovas gali pradėti padalinių tolimesnius veiksmus – darbuotojų, kompetencijų ir kitos informacijos registraciją
Pagrindinis scenarijus	Įvedami naujos įmonės duomenys registracijos lange; Spaudžiama išsaugoti; Sukuriama nauja įmonė;
Alternatyvus scenarijus	Nutrūkus interneto ryšiui vartotojo veiksmai gali būti neįvykdomi; Registruojant neužpildomi reikalaujami įvesties laukai; Egzistuoja įmonė/padalinis su tokiu pačiu pavadinimu; Klaidos pranešimo atvaizdavimas;

3.6 lentelė. Panaudos atvejis - Tikslų valdymas

Panaudos atvejis	Tikslų valdymas
Tikslas	Sukurti naujus tikslus pasirinktam darbuotojui
Aktoriai	Vadovas
Ryšys su kitais PA	-
Nefunkciniai reikalavimai	Patogiai ir aiškiai išdėstytas naujų tikslų langas (sąrašas); Prie kiekvieno tikslo pateikiama šalinimo nuoroda. Paspaudus tikslo vardą patenkama į jo redagavimo langą.
Prieš-sąlyga	Darbuotojas neturi naujų tikslų
Sužadinimo sąlyga	Vadovas spaudžia ant nuorodos meniu juostoje „*Goals*“ ir patenką į langą, kuriame pateikiami visi tikslai, užregistruoti vadovo įmonėje. Vadovas spaudžia ant naujo tikslo kūrimo nuorodos ir jam patiekiama forma, kurioje jis užpildo tikslo informaciją ir, iki kada tikslas turi būti atliktas, bei kuriam darbuotojui tikslas priskiriamas; Spaudžiama ant šalinimo nuorodos;
Po-sąlyga	Darbuotojui yra priskiriami nauji tikslai, kurie kito pokalbio metu bus įvertinami.
Pagrindinis scenarijus	Vadovas įveda naujus tikslus, komentarus ir įverčius prie kiekvieno tikslo bei atlikimo datą ir spaudžia išsaugoti; Darbuotojas mato jam priskirtus naujus tikslus; Šalinimo atveju vadovas spaudžia pasirinkto tikslo šalinimo nuorodą; Tikslas pašalinamas; Redagavimo atveju vadovas spaudžia ant tikslo pavadinimo, tikslų sąraše ir patenka į redagavimo langą; Vadovas pakeičia tikslo informacija ir spaudžia „*Save*“ nuorodą;
Alternatyvus scenarijus	Nutrūkus interneto ryšiui vadovo veiksmai gali būti neįvykdomi; Registruojant neužpildomi reikalaujami įvesties laukai; Klaidos pranešimo atvaizdavimas;

3.7 lentelė. Panaudos atvejis - Mokymų priskyrimas

Panaudos atvejis	Mokymų priskyrimas
Tikslas	Vadovas nori priskirti naują mokymą savo darbuotojui
Aktoriai	Vadovas
Ryšys su kitais PA	Mokymų valdymas
Nefunkciniai reikalavimai	Aiškiai ir patogiai išdėstytas mokymų sąrašas.
Prieš-sąlyga	Darbuotojui reikia mokymų siekiant pakelti jo kvalifikaciją. Prieš naujų mokymų priskyrimą darbuotojas gali turėti mokymų iš ankščiau.
Sužadinimo sąlyga	Iš mokymų sąrašo vadovas pasirinkęs reikiamus mokymus spaudžia „priskirti“.
Po-sąlyga	Darbuotojui yra priskiriami nauji mokymai ir jis juos gali matyti savo statistikos puslapyje
Pagrindinis scenarijus	Pasirenka reikiamus mokymus; Spaudžiama priskyrimo nuoroda – patenkama į langą, kur pasirenkamas darbuotojas; Spaudžia „priskirti“; Atvaizduojamas sėkmingo priskyrimo pranešimas;
Alternatyvus scenarijus	Nutrūkus interneto ryšiui vadovo veiksmai gali būti neįvykdomi; Klaidos pranešimo atvaizdavimas;

3.8 lentelė. Panaudos atvejis - Tikslų vertinimas

Panaudos atvejis	Tikslų vertinimas
Tikslas	Darbuotojo apklausos metu įvertinti darbuotojo pasiektus tikslus
Aktoriai	Vadovas
Ryšys su kitais PA	Apklausos pradėjimas
Nefunkciniai reikalavimai	Aiškiai išdėstyti darbuotojui priskirti tikslai. Vadovas gali pakomentuoti kiekvieną įvertinimą; Vadovas pasirenka įvertinimą, kaip tikslas buvo atliktas – laiku ar ne;
Prieš-sąlyga	Sąrašas su dar neįvertintais tikslais, kurie buvo apibrėžti nuo praėjusio darbuotojo vertinimo
Sužadinimo sąlyga	Pasirenkamas tikslo įvertinimas bei komentarai
Po-sąlyga	Darbuotojo tikslo statusas pasikeičia į įvertinto tikslo
Pagrindinis scenarijus	Pradedama paklausa; Pateikiami darbuotojo tikslai; Pasirankami įvertinimai prie kiekvieno tikslo (ar tikslas atliktas ir ar laiku); Apklausos pabaigoje išsaugoma apklausa su tikslų įvertinimais;
Alternatyvus scenarijus	Nutrūkus interneto ryšiui vadovo veiksmai gali būti neįvykdomi; Klaidos pranešimo atvaizdavimas;

3.9 lentelė. Panaudos atvejis - Mokymų vertinimas

Panaudos atvejis	Mokymų vertinimas
Tikslas	Darbuotojo apklausos metu įvertinti darbuotojui priskirtus mokymus
Aktoriai	Vadovas
Ryšys su kitais PA	Apklausos pradėjimas
Nefunkciniai reikalavimai	Aiškiai išdėstyti darbuotojui priskirti mokymai. Vadovas gali pakomentuoti kiekvieną įvertinimą;
Prieš-sąlyga	Sąrašas su dar neįvertintais mokymais, kurie buvo apibrėžti nuo praėjusio darbuotojo vertinimo
Sužadinimo sąlyga	Atliktas mokymas pažymimas kaip atliktu arba ne. Taip pat galima pateikti komentarą;
Po-sąlyga	Darbuotojo mokymo statusas pasikeičia į įvertinto mokymo
Pagrindinis scenarijus	Pradedama paklausa; Pateikiami darbuotojo mokymai; Pasirenkami įvertinimai prie kiekvieno mokymo (pažymima ar mokymas atliktas ar ne); Apklausos pabaigoje išsaugoma apklausa su mokymų įvertinimais;
Alternatyvus scenarijus	Nutrūkus interneto ryšiui vadovo veiksmai gali būti neįvykdomi; Klaidos pranešimo atvaizdavimas;

3.10 lentelė. Panaudos atvejis - Kompetencijų vertinimas

Panaudos atvejis	Kompetencijų vertinimas
Tikslas	Darbuotojo apklausos metu įvertinti darbuotojo kompetencijas
Aktoriai	Vadovas
Ryšys su kitais PA	Apklausos pradėjimas
Nefunkciniai reikalavimai	Aiškiai pateikiamas padalinio kompetencijų sąrašas, pagal kurias vertinamas darbuotojas; Kiekviena kompetencija vertinama 5 balų sistemoje; Pateikiamas grafikas su vertinamo darbuotojo kompetencijų įvertinimais iš ankstesnių apklausų;
Prieš-sąlyga	Sąrašas su dar neįvertintomis kompetencijomis;
Sužadinimo sąlyga	Pasirenkami kompetencijų įverčiai ir pateikiami komentarai

Po-sąlyga	Išsaugomi darbuotojo nauji kompetencijų balai;
Pagrindinis scenarijus	Pradedama paklausa; Pateikiamas padalinio kompetencijų sąrašas; Pasirenkami įvertinimai prie kiekvienos kompetencijos; Apklausos pabaigoje išsaugoma apklausa su kompetencijų įvertinimais;
Alternatyvus scenarijus	Nutrūkus interneto ryšiui vadovo veiksmai gali būti neįvykdomi; Klaidos pranešimo atvaizdavimas;

3.11 lentelė. Panaudos atvejis - Asmeninės statistikos peržiūra

Panaudos atvejis	Asmeninės statistikos peržiūra
Tikslas	Darbuotojas nori peržiūrėti savo statistiką
Aktoriai	Darbuotojas
Ryšys su kitais PA	-
Nefunkciniai reikalavimai	Visa darbuotojo statistika pateikiama aiškiai viename lange; Lange pateikiama informacija apie jo organizaciją; Lange pateikiami darbuotojo vadovo kontaktai; Lange pateikiama darbuotojo informacija – kontaktai; Lange pateikiamas grafikas su darbuotojo tikslais ir mokymais; Lange pateikiami sąrašai su darbuotojui priskirtais tikslais ir mokymais; Lange pateikiamas darbuotojo kompetencijų įvertinimų grafikas su galimybe palyginti ketvirčių įvertinimus; Pateikiami tik einamų metų duomenys;
Prieš-sąlyga	Darbuotojas nori pasitikrinti savo statistiką;
Sužadinimo sąlyga	Darbuotojas prisijungia prie sistemos;
Po-sąlyga	Darbuotojui pateikiamas statistikos langas su visa informacija;
Pagrindinis scenarijus	Darbuotojas prisijungia prie sistemos; Pateikiamas darbuotojo statistikos langas su duomenimis;
Alternatyvus scenarijus	Nesėkmingas prisijungimas prie sistemos;

3.12 lentelė. Darbuotojų vertinimų statistikos peržiūra ir palyginimas

Panaudos atvejis	Darbuotojų vertinimų statistikos peržiūra ir palyginimas
Tikslas	Pasirinkto darbuotojo statistikos patiekimas
Aktoriai	Vadovas
Ryšys su kitais PA	-
Nefunkciniai reikalavimai	Grafiškai pateikiami kompetencijų įvertinimai su galimais filtrais: Metų ketvirtis Metai Darbuotojas su kuriuo lyginami kompetencijų vertinimai Vidutinių reikšmių naudojimas Grafiškai pateikiami darbuotojo atlikti tikslai ir mokymai (kairėje pasirinktas darbuotojas, dešinėje lyginamo darbuotojo). Filtravimas tik pagal metus;
Prieš-sąlyga	Norima pamatyti darbuotojo statistiką ir palyginti su kitu darbuotoju;
Sužadinimo sąlyga	Pasirenkamas darbuotojas iš darbuotojų sąrašo „*Evaluate*“ puslapyje ir spaudžiama „*Statistic*“;
Po-sąlyga	Pateikiama detali informacija apie darbuotoją ir galimybė filtruoti duomenis bei lyginti su pasirinktu kitu darbuotoju;
Pagrindinis scenarijus	Meniu spaudžiama „*Evaluate*“ Pasirenkamas darbuotojas ir spaudžiama „*Statistic*“ Pateikiamas langas su statistika ir filtrais;
Alternatyvus scenarijus	Nutrūkus interneto ryšiui vadovo veiksmai gali būti neįvykdomi; Klaidos pranešimo atvaizdavimas;

3.13 lentelė. Įmonės/padalinio valdymas

Panaudos atvejis	Įmonės/padalinio valdymas
Tikslas	Redaguoti pasirinktą įmonę ar padalinį
Aktoriai	Administratorius Vadovas
Ryšys su kitais PA	Darbuotojų valdymas;
Nefunkciniai reikalavimai	Pateikiama padalinio ar įmonės redagavimo forma su įvestimis, kuri yra tikrinama.
Prieš-sąlyga	Norima redaguoti pasirinktą padalinį; Vadovas redaguoja tik savo padalinį; Administratorius gali redaguoti visus;
Sužadinimo sąlyga	Pasirenkamas padalinys – administratorius iš padalinių sąrašo pasirenka norimą padalinį; Vadovas spaudžia nuorodą meniu juostoje, kuri nukreipia jį į jo padalinio redagavimo langą;
Po-sąlyga	Padalinio ar įmonės duomenys yra atnaujinami; Taip pat pateikiama padalinio detali informacija;
Pagrindinis scenarijus	Administratorius spaudžia meniu juostą, kuri nukreipia į padalinių sąrašą; Pasirenkamas padalinys: Pateikiama padalinio informacija ir forma su redagavimo laukais; Esant poreikiui keičiama padalinio informacija; Spaudžiamas išsaugojimo mygtukas; Vadovas spaudžia meniu nuorodą, kuri nukreipia vadovą į langą, kur pateikiama jo įmonės informacija su redagavimo forma; Esant poreikiui, vadovas keičia įmonės informaciją; Spaudžia išsaugojimo mygtuką;
Alternatyvus scenarijus	Nutrūkus interneto ryšiui vartotojo veiksmai gali būti neįvykdomi; Redagavimo/registravimo metu neužpildomi reikalaujami įvesties laukai; Klaidos pranešimo atvaizdavimas;

3.14 lentelė. Apklausos pradėjimas

Panaudos atvejis	Apklausos pradėjimas
Tikslas	Atlikti darbuotojo metų ketvirtinį įvertinimą;
Aktoriai	Vadovas
Ryšys su kitais PA	Tikslų vertinimas; Mokymų vertinimas; Kompetencijų vertinimas;
Nefunkciniai reikalavimai	Pasirenkamas metų ketvirtis, kurs bus vertinamas;
Prieš-sąlyga	Su darbuotoju yra suderinamas pokalbio laikas.
Sužadinimo sąlyga	Pasirinkus darbuotoją paspaudžiama nuoroda „*Start new survey*“
Po-sąlyga	Pradedamas darbuotojo metinio ketvirčio vertinimas; Pateikiama vertinimo forma;
Pagrindinis scenarijus	Meniu juostoje spaudžiama „*Evaluate*“ nuoroda; Pasirenkamas vertinamas darbuotojas; Spaudžiama nuoroda „*Evaluate*“; Pateikiama naujos apklausos forma; Pasirenkamas neįvertintas ketvirtis; Spaudžiama „*Next*“ Pateikiama vertinimo forma;
Alternatyvus scenarijus	Nutrūkus interneto ryšiui vartotojo veiksmai gali būti neįvykdomi; Klaidos pranešimo atvaizdavimas; Visi ketvirčiai jau įvertinti, todėl vertinimo pradėti negalima. Reikia laukti naujų metų;

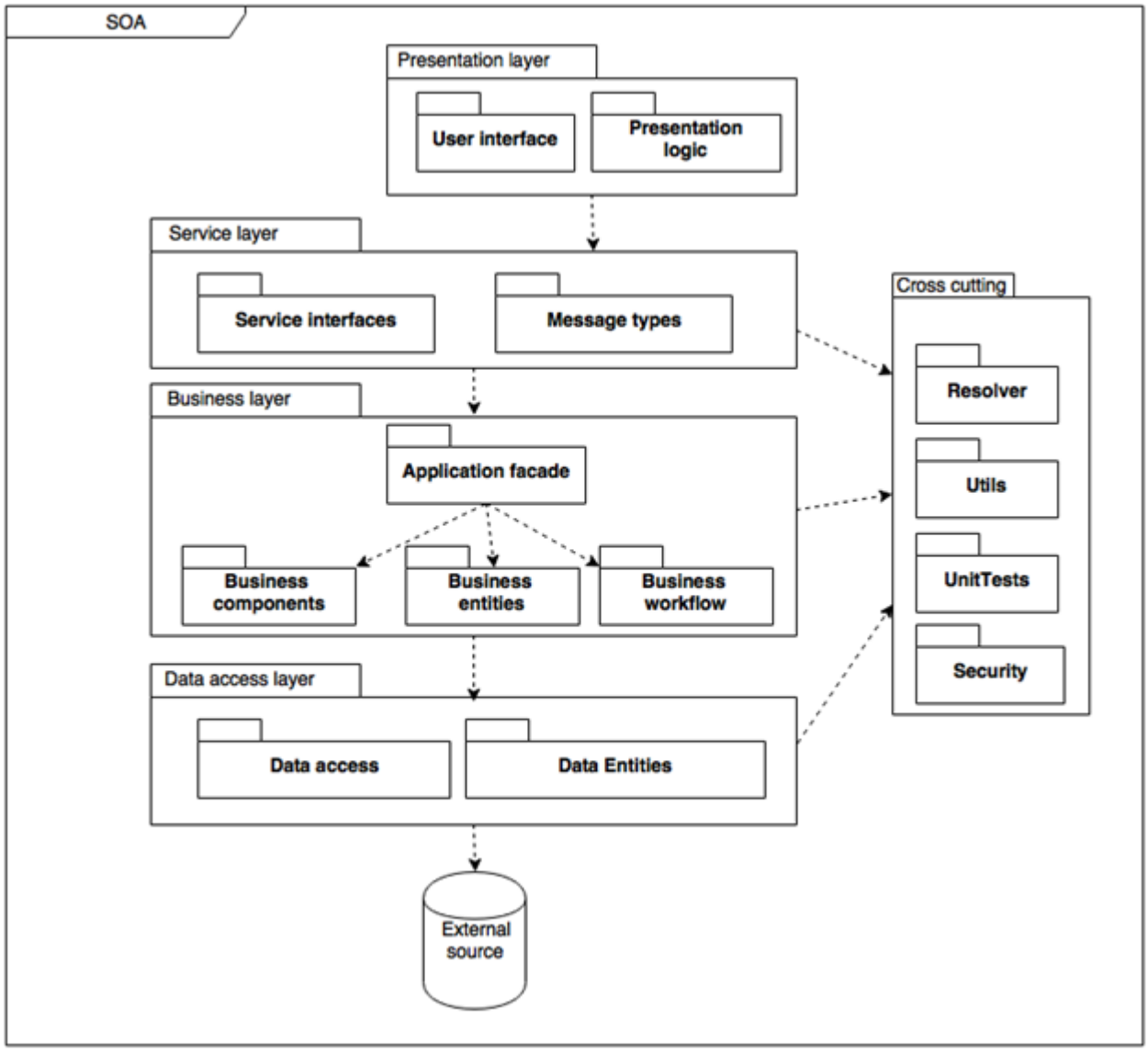
3.15 lentelė. Prisijungti/ Atsijungti

Panaudos atvejis	Prisijungti/Atsijungti
Tikslas	Prisijungti prie sistemos
Aktoriai	Vadovas; Darbuotojas; Įmonės administratorius;
Ryšys su kitais PA	-
Nefunkciniai reikalavimai	-
Prieš-sąlyga	Vartotojas nori prisijungti prie sistemos; Vartotojas nori atsijungti;
Sužadinimo sąlyga	Vartotojui reikia pasinaudoti sistema; Vartotojas baigia darbą;
Po-sąlyga	Vartotojas autentifikuojamas ir gali naudoti sistemą; Vartotojo sesija nutraukiama;
Pagrindinis scenarijus	Vartotojas suveda prisijungimo vardą ir slaptažodį; Sistema patikrina duomenis; Vartotojas prijungiamas prie sistemos; Vartotojas spaudžia atsijungimo nuorodą; Vartotojo sesija nutraukiama ir jis atjungiamas nuo sistemos;
Alternatyvus scenarijus	-

3.6. Sistemos statinis vaizdas

3.6.1. Apžvalga

Sistemą sudaro penki pagrindiniai paketai, kurie padalina ją į tam tikrus lygmenis. Toks sistemos padalinimas yra realizuojamas vadovaujantis n-eilės architektūriniu modeliu, įterpiant papildomą paslaugų lygmenį (angl. *service layer*). Lygmuo leis sistemai būti pasiekiamai ne tik vartotojui, bet ir kitoms, išorinėms sistemoms (žr. 3.2 pav.).



3.2 pav. Sistemos išskaidymas į paketus

3.6.2. Paketų detalizavimas

N-eilės (angl. *N-tier*) architektūra literatūros šaltiniuose apibrėžiama, kaip vartotojo-serverio architektūra, kuri yra išskaidoma į detalesnius lygius. Šio tipo architektūra realizuotos programos dar yra vadinamos paskirstytomis (angl. *distributed*) arba daugelio lygmenų (angl. *multitier*). Naudojant

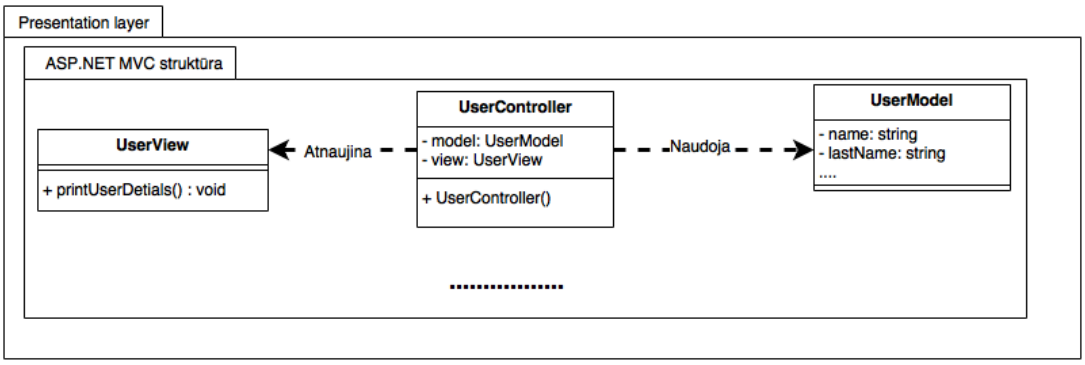
šį architektūrinį sprendimą, sistemos procesai yra išskiriami į atskirus lygmenis. Standartiškai n-lygmens programose yra išskiriami trys lygmenys:

- Pristatymo lygmuo-eilė (angl. *presentation tier/layer*)
- Verslo logikos lygmuo-eilė (angl. *business tier/layer*)
- Duomenų lygmuo-eilė (angl. *data tier/layer*)

Pateikimo lygmuo (angl. *presentation layer*) atsakingas už vartotojo grafinę sąsają ir sąsajos logiką. Projektuojant sistemą ir realizuojant šį architektūrinį lygmenį, bus naudojama *ASP.NET MVC* struktūra. Ši struktūra bus atsakinga tik už grafinę sąsają ir komunikaciją su paslaugų lygmeniu (angl. *service layer*). Pateikimo lygmens paketas sudarytas iš smulkesnių paketų:

- Vartotojo sąsaja (angl. *user interface*) – sąsają aprašantys failai naudojant HTML, CSS ir *Javascript* kalbas.
- Sąsajos logika (angl. *presentation logic*) – logika bus realizuota MVC šablono pagalba.

Vartotojo sąsajos ir sąsajos logikos paketus apjungia *ASP.NET MVC* struktūra. Detalesnis vaizdas, pateiktas paveikslėlyje, (žr. 3.3 pav.) vaizduoja tai, jog *UserController* yra atsakingas už logiką, skirtą komunikacijai su paslaugų lygmeniu ir duomenų paruošimu atvaizdavimui. *UserModel* klasė atsakinga už atvaizdavimui skirto modelio aprašymą, o *UserVew* atsakingas už atvaizdavimą. Daugtaškis vaizduoja, jog likusi lygmens dalis realizuota panašiu principu.



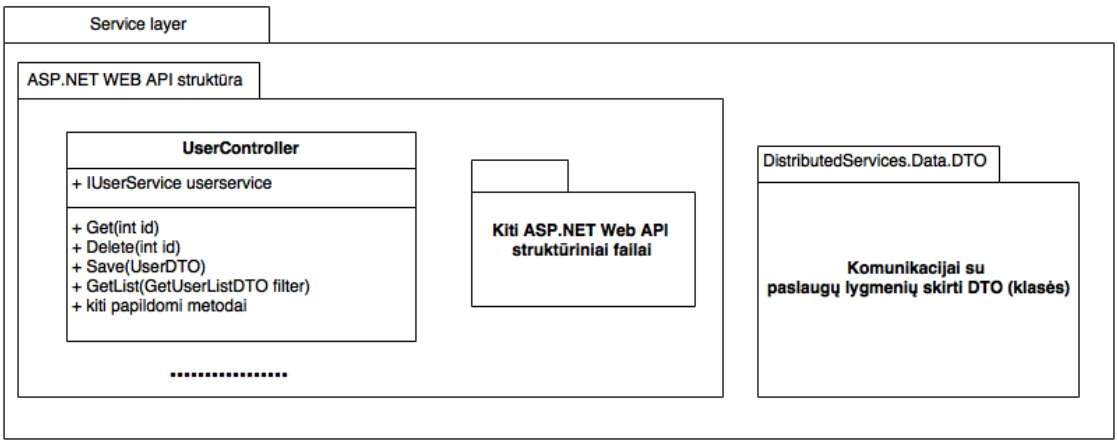
3.3 pav. MVC struktūros atvaizdavimo lygmens detalus vaizdas

Paslaugų lygmuo (angl. *service layer*) atsakingas už sistemos paslaugų suteikimą išorinėms sistemoms, naudojant nustatytus bendravimo protokolus ir formatus. Dažniausiai šis lygmuo yra laikomas pristatymo lygmeniu, nes yra atsakingas tik už sąsają su sistemos paslaugomis. Siekiant realizuoti REST paslaugų tarnybą (angl. *web service*), kuri bus pasiekiama iš išorės naudojant HTTP protokolą ir jo metodus (POST, GET, PUT, DELETE), bus naudojama *ASP.NET Web Api* struktūra. Siunčiamų duomenų formatas bus JSON tipo. Šis lygmuo taip pat bus atsakingas už saugumą. Tik autorizuoti vartotojai galės bendrauti su paslaugų teikėju - API. Paslaugų lygmuo naudos papildomus paketus iš kompleksinio (angl. *cross cutting*) paketo. Šis paketas bus pasiekiamas tarp visų lygmenų,

tačiau nesusies jų tarpusavyje. Kompleksiniame pakete bus saugomos tik pagalbinės klasės, kurios bus pasiekiamos visiems lygmenims, išskyrus pateikimo lygmeniui. Paslaugų lygmens paketas sudarytas iš smulkesnių paketų (žr. 3.4 pav.):

- Paslaugos sąsajos (angl. *service interfaces*) – sąsajos aprašančios paslaugas. *ASP.NET WEB API* projektas pasirūpina sąsajų suteikimu.
- Žinučių tipai (angl. *Message types*) – klasės (DTO), aprašančios duomenų pardavimo objektus. *DistributedServices.Data.DTO* klasių biblioteka saugos šiuos DTO.

Paveikslėlyje (žr. 3.4 pav.) pateikiamas tik vienos sąsajos valdiklis *UserController*, kuris priims užklausas iš išorinių sistemų, HTTP protokolo pagalba. Paveikslėlyje pateikiama tik viena klasė, nes likusios klasės bus realizuotos panašiu principu. Kiti vidiniai paketai nedetalizuojami, nes jie priklauso *WEB API* standartinei struktūrai.

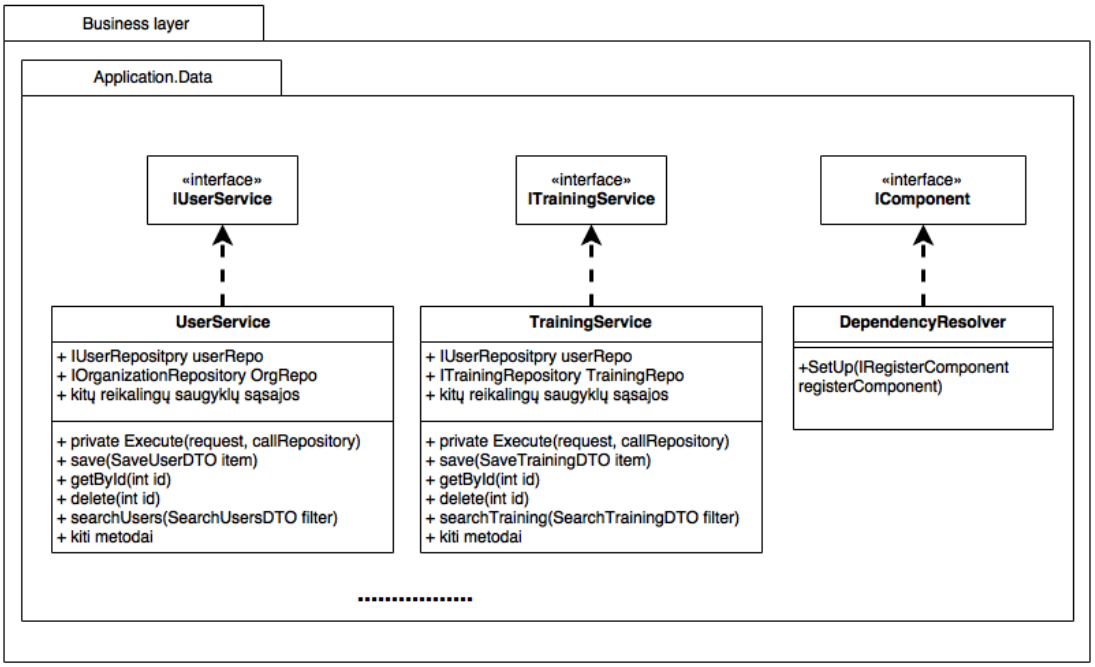


3.4 pav. Paslaugų lygmens detalus vaizdas

Verslo logikos lygmuo (angl. *business layer*) bus atsakingas už sistemos pagrindines logines operacijas. Šiame lygmenyje yra sąrašas operacijų, kurios atlieka tam tikrus loginius veiksmus su duomenimis. Šioje pakopoje yra realizuojami duomenų perdavimo objektai (angl. *DTO*), funkcijos, skaičiavimai, duomenų tikrinimas bei kiti komponentai, skirti programos logikai realizuoti (pvz. autorizacija). Šiame lygmenyje konkreti operacija gali pasiekti skirtingas saugyklas (angl. *repositories*) iš duomenų pasiekimo lygmens ir manipuliuoti jomis, siekiant realizuoti reikiamą operaciją. Verslo logikos lygmens paketas sudarytas iš smulkesnių paketų:

- Verslo komponentai (angl. *business components*) – papildomos klasės dalyvaujančios loginėse operacijose.
- Verslo subjektai (angl. *business entities*) – subjektai, kuriais operacijos manipuliuoja.
- Verslo logika (angl. business workflow) – operacijas aprašančios sąsajos ir jas realizuojančios klasės.

Verslo logikos paketai yra apjungiami į vieną biblioteką *Application.Data*, kuri atsakinga už operacijų logikas ir sąsajų pateikimą paslaugų lygmeniui. Pateiktame (žr. 3.5 pav.) paveikslėlyje pavaizduotos kelios sąsajos ir jas realizuojančios klasės. Kiekviena paslaugų klasė gali pasiekti duomenų lygmenį naudojant saugyklų (angl. *repository*) sąsajas. Pateiktame (žr. 3.5 pav.) paveikslėlyje daugtaškis vaizduoja tai, kad likusi lygmens realizacija yra panaši, skiriasi tik esybės (pvz. pateikta tik *User* ir *Training* esybės). *DependencyResolver* klasė atsakinga už sąsajų priklausomybių išsprendimą naudojant MEF struktūrą.

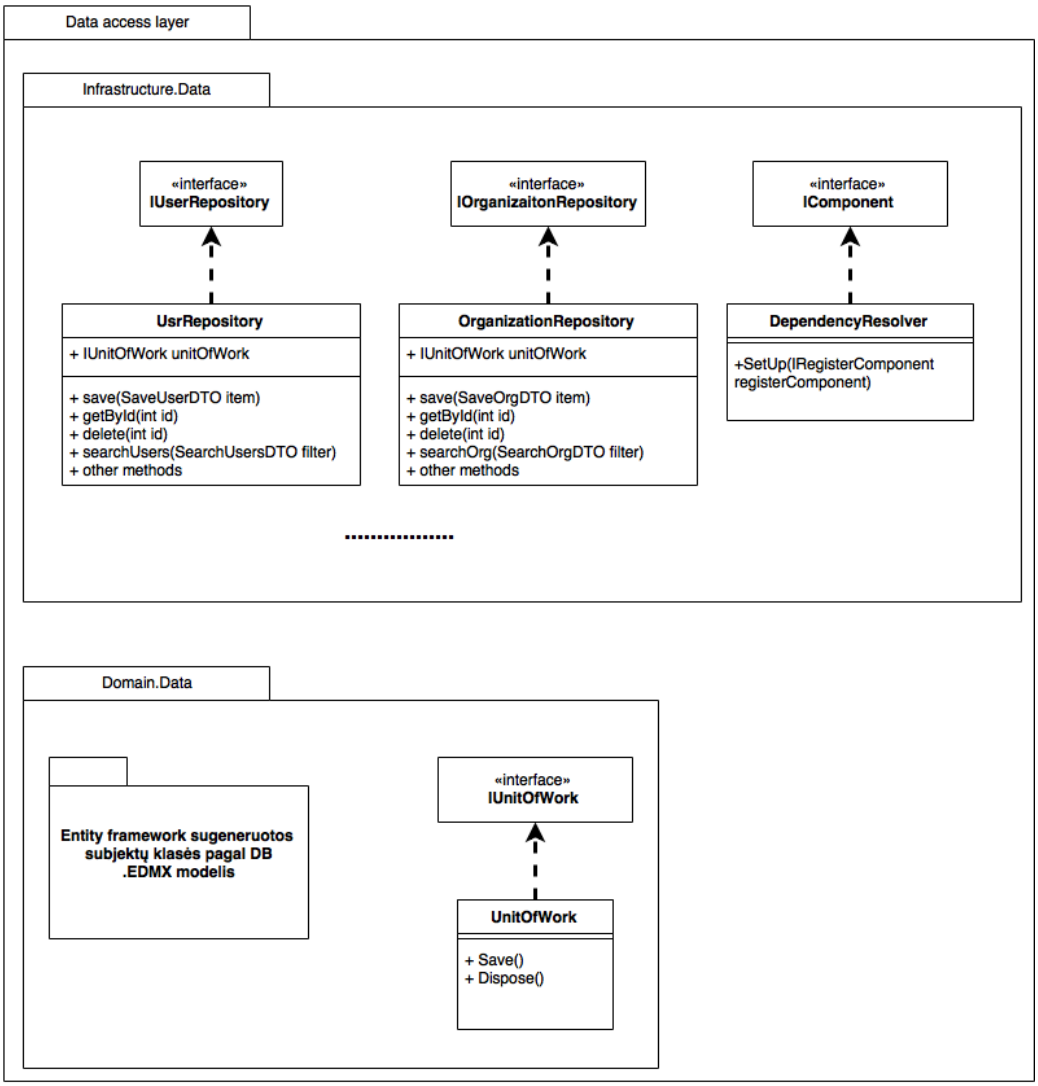


3.5 pav. Verslo logikos lygmens detalus vaidas

Duomenų pasiekimo lygmuo (angl. *data access layer*) bus atsakingas už komunikaciją su duomenų baze. Šio lygmens funkcionalumas, kaip ir kitų lygmenų funkcionalumas, bus izoliuotas nuo išorės ir komunikuoti su juo galės tik aukštesnis lygmuo. Šiuo atveju, verslo logikos lygmuo galės komunikuoti su duomenų pasiekimo lygmeniu, tačiau paslaugų lygmuo kreiptis į duomenų pasiekimo lygmenį negalės. Duomenų pasiekimo lygmenyje bus naudojama „*ORM mapper entity framework*“ struktūra. Ši struktūra sugeneruos klases pagal duomenų bazės lentelių subjektus ir leis prisijungti prie DB. Duomenų pasiekimo lygmens paketas sudarytas iš smulkesnių paketų:

- Duomenų pasiekimas (angl. *Data access*) – sąsajos leidžiančios pasiekti ir manipuliuoti duomenimis iš verslo logikos lygmens. Projekte už šį funkcionalumą atsakinga *Infrastucture.Data* biblioteka.
- Duomenų esybės (angl. *Data entities*) – *Entity framework* pagalba sugeneruotos esybių klasės. Projekte už šį funkcionalumą atsakinga *Domain.Data* biblioteka.

Paveikslėlyje (žr. 3.6 pav.), *Infrastructure.Data* bibliotekoje yra pateikiamos sąsajos ir jų realizacijos. Sąsajas realizuojančios klasės atsakingos už duomenų manipuliavimo operacijas – duomenų gavimą, įrašymą, šalinimą ir atnaujinimą. Paveikslėlyje yra pateikiamos tik dviejų esybių pasiekimo klasės – *User* ir *Organization*, nes likusių esybių pasiekimo klasės realizuojamos identiškai. *DependencyResolver* klasė atsakinga už sąsajų priklausomybių išsprendimą naudojant MEF struktūrą.



3.6 pav. Duomenų pasiekimo lygmens detalus vaidas

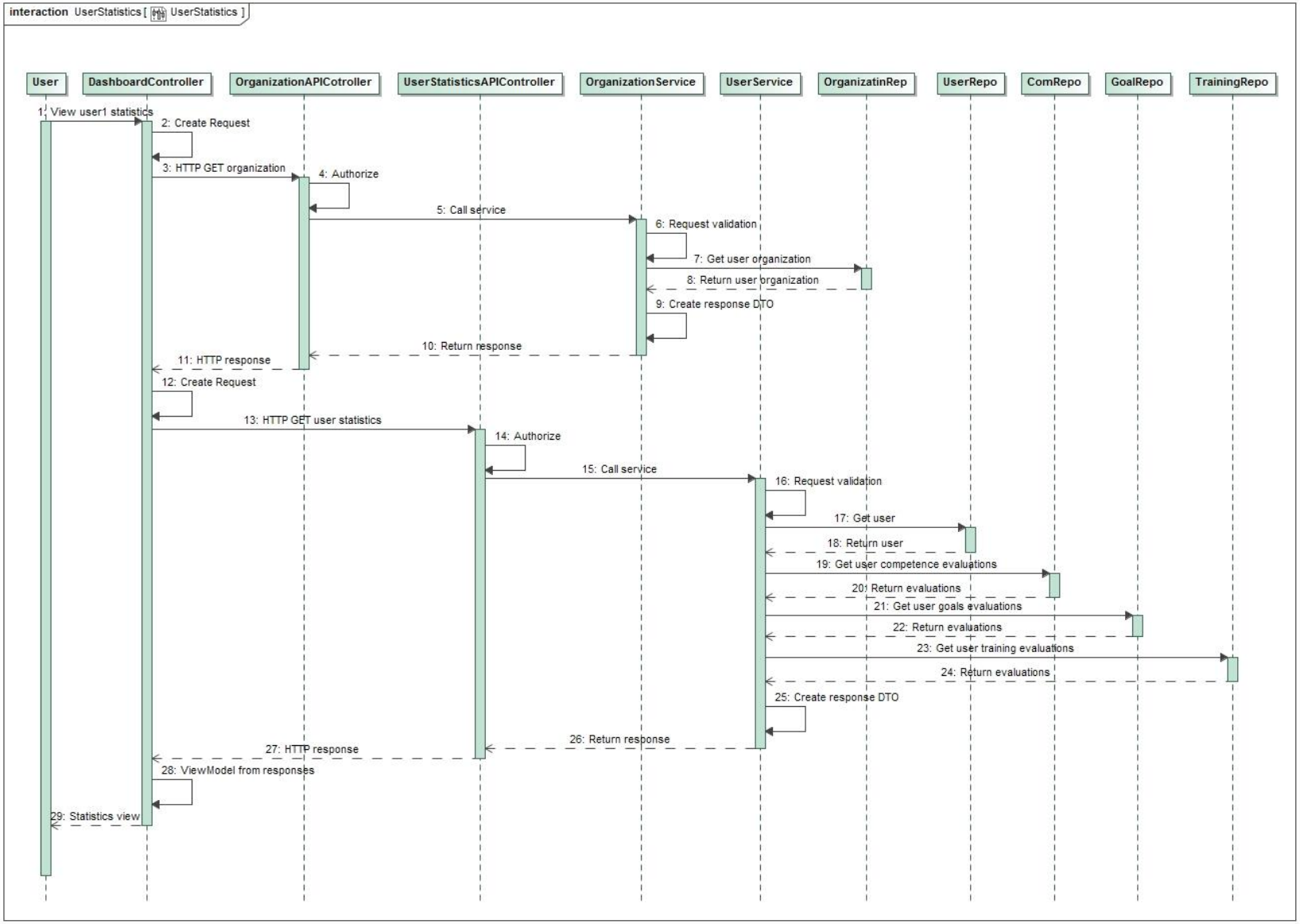
3.7. Sistemos dinaminis vaizdas

3.7.1. Sekų diagramos

Žemiau esančiame paveikslėlyje yra pavaizduota sekų diagrama (žr. 3.7 pav.), kurioje yra realizuota standartinė šios sistemos vartotojo veiksmų seka. Šioje sekų diagramoje pavaizduojama tai, jog kiekvienas lygmuo gali pasiekti tik žemesnį sistemos lygmenį. Likusieji veiksmai sistemoje yra atliekami identiška veiksmų seka, skiriasi tik klasės, kuriomis yra manipuliuojama. Vartotojo

prisijungimas prie sistemos nėra pavaizduotas, nes veiksmų seka yra identiška - skiriasi tik klasės. Detalizuojami visi žingsniai, esantys sekos diagramoje:

1. Puslapio lange pasirenkamas darbuotojas, kurio statistiką norima pamatyti. Siunčiama užklausa į *DashboardController*.
2. *DashboardController* valdiklyje yra formuojama HTTP užklausa į API gauti informaciją apie pasirinkto vartotojo organizaciją.
3. Siunčiama HTTP užklausa į *OrganizationAPIController* gauti informaciją apie organizaciją.
4. API valdikliuose yra pasirūpinama autorizacija - ar vartotojui leidžiama pasiekti tam tikrus valdiklius API dalyje.
5. Iš *OrganizationAPIController* valdiklio yra kreipiamasi į *OrganizationService* paslaugą (angl. service).
6. *OrganizationService* patikrina gautos užklausos duomenų saugumą prieš kreipiantis į duomenų pasiekimo lygmenį.
7. Iš *OrganizationService* yra kreipiamasi į duomenų pasiekimo lygmenį, kad gauti informaciją iš duomenų bazės apie pasirinkto vartotojo organizaciją.
8. Gražinama informacija apie organizaciją.
9. Formuojamas rezultatų objektas (DTO), kuris bus gražintas iš API į kliento valdiklį – *DashboardController*.
10. Gražinamas suformuotas DTO į aukštesnį lygį.
11. Gražinama informacija apie organizaciją, užklausą atsiuntusiam klientui – *DashboardController*.
12. Kuriama nauja užklausa gauti informaciją apie darbuotojo vertinimus.
13. – 16. Veiksmai analogiški. Skiriasi tik klasės/valdikliai į kuriuos yra kreipiamasi.
14. – 24. Vaizduoja, jog paslaugų lygmuo gali naudoti daugiau nei vieną duomenų saugyklą (angl. *repository*) iš duomenų pasiekimo lygmens.
15. -27. Veiksmai analogiški - gražinamas suformuotas rezultatas.
16. Formuojamas atvaizdavimo modelis, kuris bus naudojamas atvaizduojant duomenis vartotojo lange.
17. Vartotojui pateikiamas vaizdas su pasirinkto darbuotojo statistika ir organizacija, kuriai darbuotojas priklauso.

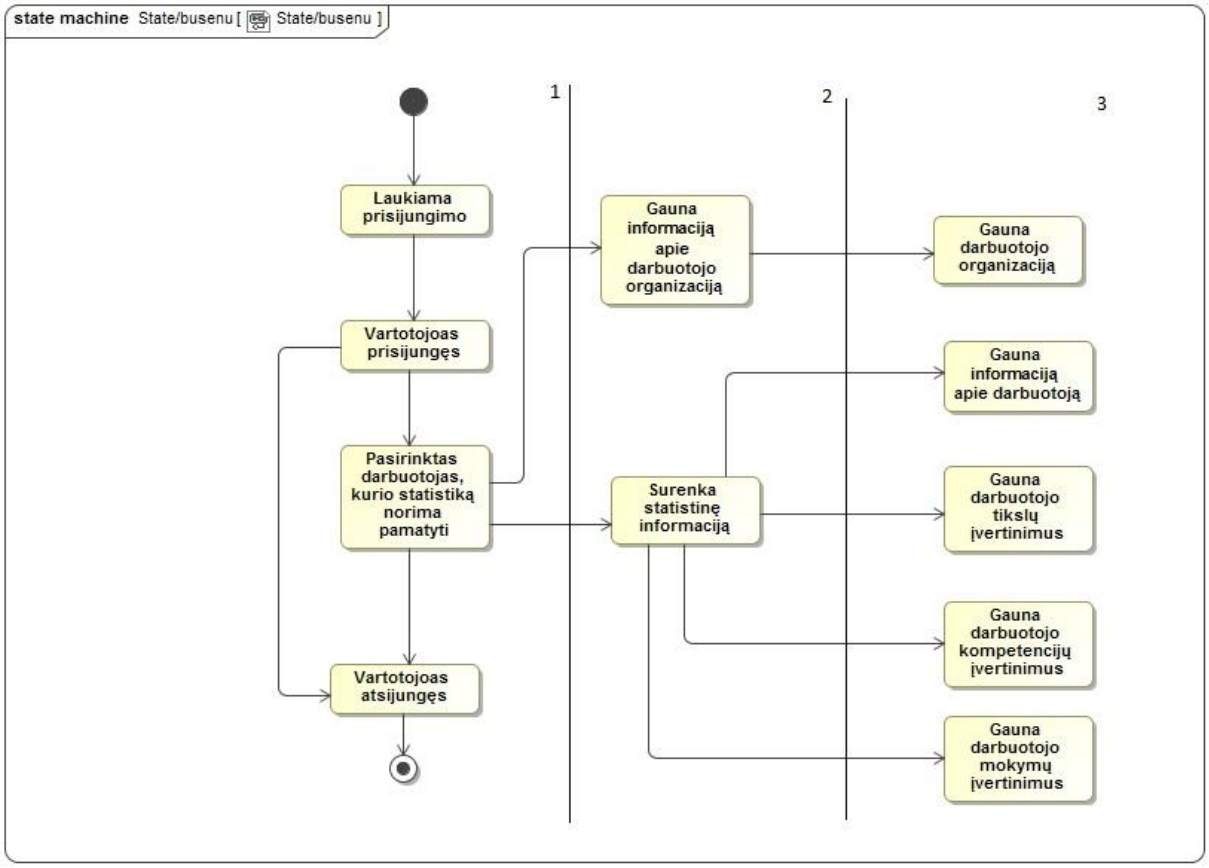


3.7 pav. Standartinė sekos diagrama

3.7.2. Būsenų diagrama

Tam pačiam scenarijui, koks buvo realizuotas sekų diagramoje (žr. 3.7 pav.), yra pavaizduota būsenų diagrama (žr. 3.8 pav.). Šioje diagramoje yra pavaizduotos būsenos, kuriose yra sistema duotajame scenarijuje. Pirmiausiai sistema laukia kol vartotojas prisijungs, nes neprisijungęs vartotojas sistemos naudoti negali. Kai vartotojas pateikia savo prisijungimo duomenis, jo būsena pasikeičia į prisijungusio vartotojo. Kuomet yra pasirenkamas darbuotojas, kurio statistiką norima peržiūrėti, sekančios būsenos pakeičiamos lygiagrečiai – pasikeičia būsenos į „Surenka statistinę informaciją“ ir „gauna informaciją apie darbuotojo organizaciją“. Tokiu atveju, veiktų asinchroninis principas, kuomet vienu metu būtų dvi būsenos, kurios, savo ruožtu, išsišakotų į tolimesnes būsenas. Galima pastebėti, jog būsenos pasiskirsto tarp 3-jų lygmenų. Tokiu principu veiktų visa sistema.

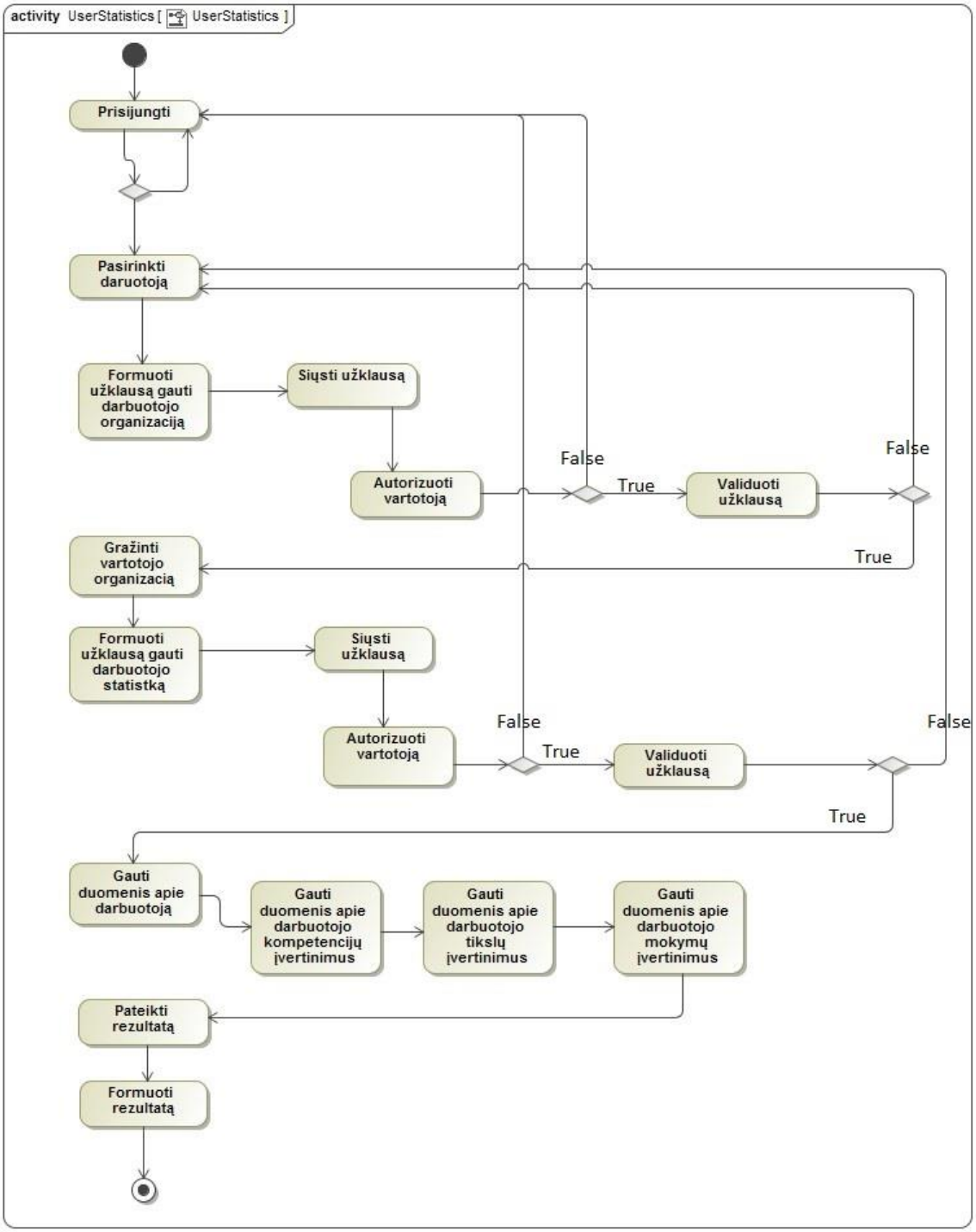
1. Pateikimo lygmuo – tiesioginius būsenų pakitimus lemia vartotojas.
2. Paslaugų lygmuo – pateikimo lygmuo gali naudoti paslaugas (angl. *services*) lygiagrečiai.
3. Verslo logikos lygmuo – būsenos pasikeičia ir laukiama, kol yra surenkami visi reikalingi duomenys.



3.8 pav. Būsenų diagrama – statistikos gavimas

3.7.3. Veiklos diagrama

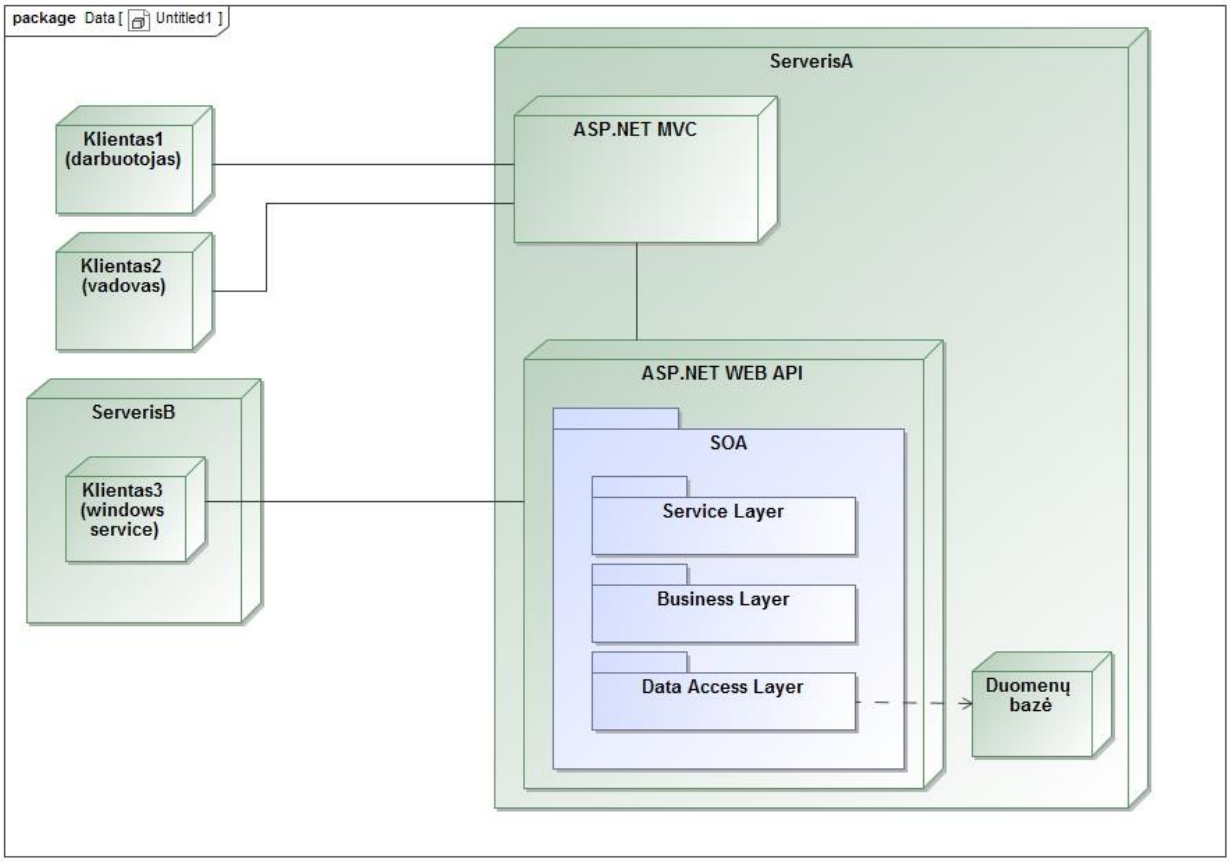
Naudojant tą pačią vartotojo veiksmų seką (kaip ir praeitose diagramose), buvo sugeneruota būsenų diagrama. Visos nepavaizduotos sistemos būsenos keičiasi panašia tvarka, skiriasi tik subjektai (šioje diagramoje (žr. 3.9 pav.) vaizduojamas darbuotojo statistikos gavimas).



3.9 pav. Veiklos diagrama – statistikos gavimas

3.8. Išdėstymo vaizdas

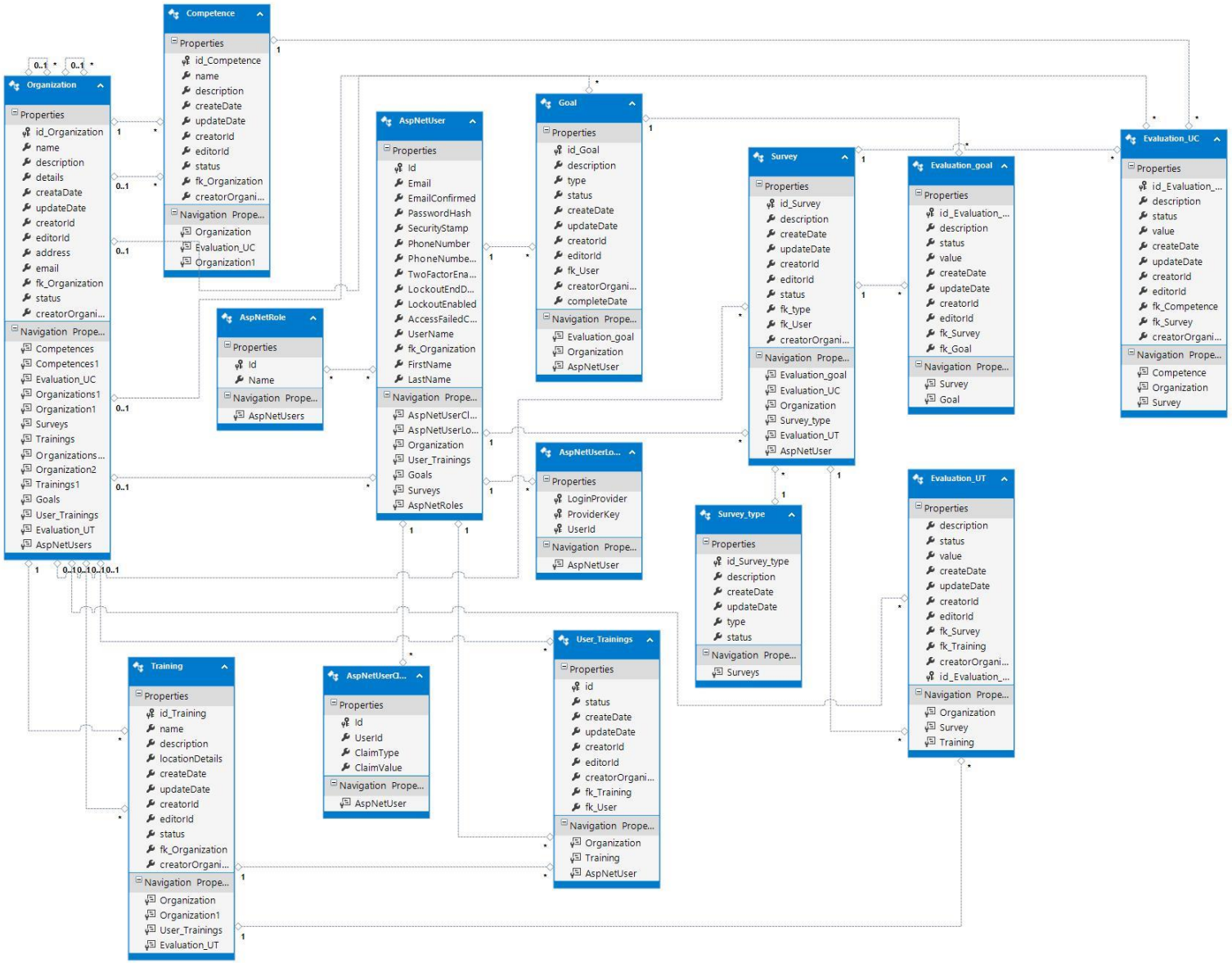
Žemiau pateiktame paveikslėlyje yra pavaizduotas sistemos išdėstymo vaizdas (žr.3.10 pav.), kuriame matyti, jog vartotojai gali pasieki serverį, kuriame yra veikianti sistema. Vartotojai, interneto pagalba, tiesiogiai bendrauja su „*ServerisA*“ veikiančia *ASP.NET MVC* realizuota sistema (internetinė svetainė). *ASP.NET MVC* realizuota svetainė sąveikauja su „*ServerisA*“ veikiančia *ASP.NET WEB API* sistema, kuri suteikia sąsajas su pagrindinėmis sistemos paslaugomis. „*ServerisB*“ vaizduoja, jog su pagrindine sistema gali bendrauti ne tik standartiniai vartotojai, internetinės svetainės pagalba, bet ir kitos sistemos (pvz. *Windows Service*). Komunikacijai su sistema svarbu, kad būtų veikiantis interneto ryšys. SOA pakete pavaizduoti pagrindiniai sistemos lygiai, kurie realizuoja pagrindinę sistemos logiką.



3.10 pav. Išdėstymo vaizdas

3.9. Duomenų vaizdas

Žemiau esančiame paveikslėlyje yra pateiktas kuriamos sistemos duomenų modelis (žr. 3.11 pav.), kuris reikalingas duomenims saugoti, kurie bus panaudoti sistemos kūrimui.



3.11 pav. Duomenų bazės modelis

3.10. Kokybė

Sistemos kokybės įvertintinimui gali būti naudojami šie parametrai: išbaigtumas, suprantamumas, pernešamumas, nuoseklumas, patikimumas, testuojamumas, pataisomumas, glaustumas, panaudojamumas, efektyvumas ir saugumas.

Architektūros įtaka sistemos kokybei:

- Užtikrina sistemos saugumą, nes vartotojo autorizacija yra vykdoma MVC ir API dalyje.
- Sistema naudoja tokius architektūrinius sprendimus, kaip SOA, kurių pagalba programa yra lengviau suprantama.
- Sistema yra lengvai pernešama į kitas aplinkas, nes tereikia perkelti programos vykdomuosius failus ir duomenų bazę.
- Sistemos architektūra yra orientuota į SOA, todėl jos priežiūra yra lengvesnė, mat sistema yra padalinta į konkrečias paslaugas.
- Sistemos architektūrinis sprendimas palengvina testavimą, nes galima rašyti testus konkrečioms paslaugoms.
- Sistema gali veikti kurį laiką be pagrindinių paslaugų, kurios yra pasiekiamos naudojant API. Tuo pasirūpina MVC dalis, kuri pateikia aiškius pranešimus vartotojui apie sistemos sutrikimus.
- Taisant sistemos paslaugas yra likviduojamos klaidos visuose vietose, kuriuose yra naudojamos paslaugos.
- Sistemos paslaugos gali būti panaudojamos kitose paslaugose.

4. TYRIMO IR EKSPERIMENTINĖ DALIS

4.1. Tyrimo tikslas

Įvertinti, kaip keičiasi priežiūros indeksas skirtingais SOA išdėstymo atvejais ir kas lemia priežiūros indekso kitimą bei kaip pagerinti sukurtos sistemos priežiūrą.

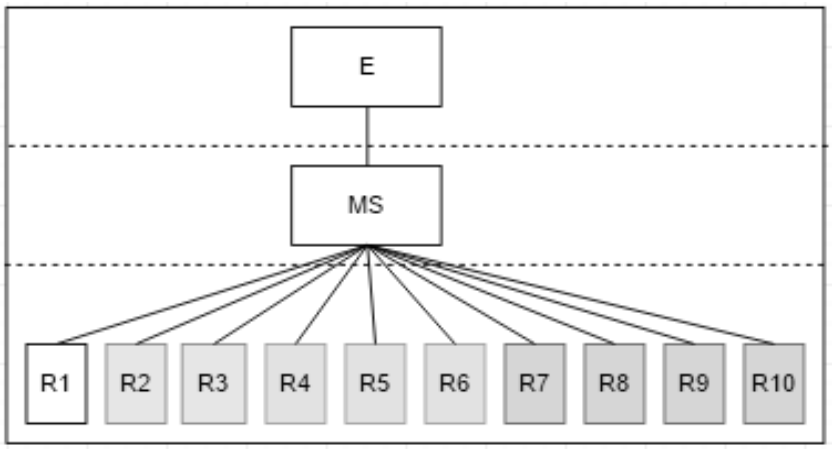
4.2. Eksperimento metodologija

Siekiant įvertinti į paslaugas orientuotos architektūros priežiūros indeksą ir pagerinti sukurtos sistemos (darbuotojų vertinimo) priežiūrą, buvo sukurta papildoma eksperimentinė programa, atitinkanti SOA principus. Eksperimento pradžioje yra išmatuojamas sukurtos darbuotojų vertinimo sistemos priežiūros indeksas, kuris apibūdins ar sistema yra lengvai prižiūrima. Eksperimento eigoje yra naudojama eksperimentinė programa, kurios architektūra yra modifikuojama. Po kiekvienos modifikacijos yra vertinamas programos priežiūros indeksas ir kitos metrikos. Eksperimentinė programos struktūra, beveik visais eksperimento atvejais, atitinka SOA principus. Tai reiškia, jog

programa yra sudaryta iš paslaugų rinkinio, kurių pagalba yra įgyvendinamas pagrindinį programos algoritmas-paslauga (angl. *main service*). Kad eksperimentinė programa atitiktų pagrindinės programos (darbuotojų vertinimo sistema) struktūrą, eksperimentinė programa yra suprojektuota naudojant n-eilės architektūrą. Pirmoje eilėje (angl. *service/presentation tier*) yra realizuota viena operacija, skirta inicijuoti eksperimentinės programos pagrindinę paslaugą. Vidurinėje eilėje (angl. *business tier*) yra realizuojamos visos programos paslaugos, skirtos įgyvendinti atskiras pagrindinės paslaugos operacijas. Taip pat šioje eilėje yra pateikiama operacija, aprašanti pagrindinę paslaugą. Žemiausioji eilė (angl. *data tier*) yra atsakinga už duomenų pateikimą programos vidurinėje eilėje esančioms paslaugų operacijoms. Programos architektūra yra išdėliota taip, jog vertikaliai yra programos eilės, o horizontaliai – paslaugos.

Eksperimente yra naudojami keturi skirtingi scenarijai, kurių metu pagrindinė eksperimentinės programos paslauga, skirtingais būdais, naudoja kitas paslaugas, kad įgyvendintų savo pagrindinę operaciją. Kiekvienam scenarijui yra naudojamas skirtingas programos architektūros išdėstymas. Kiekvieno scenarijaus metu yra vykdomi eksperimentiniai žingsniai, kurių metu prie pagrindinės paslaugos yra prijungiamos papildomos paslaugos tol, kol paskutinio žingsnio metu yra pilnai realizuojama pagrindinė operacija. Po kiekvieno eksperimentinio žingsnio yra skaičiuojamos programinės įrangos bendras priežiūros indeksas bei kitos kodo metrikos. Metrikų skaičiavimams yra naudojami integruoti įrankiai, esantys Visual Studio aplinkoje. Priežiūros indekso skaičiavimui Visual Studio naudoja tą pačią formulę, kokia yra nurodyta literatūros šaltiniuose, tačiau formulė yra modifikuota, kad rezultate reikšmė būtų tarp 0 ir 100. Visual Studio priežiūros indekso ribos yra pateikiamos 2.7.3 skyriuje.

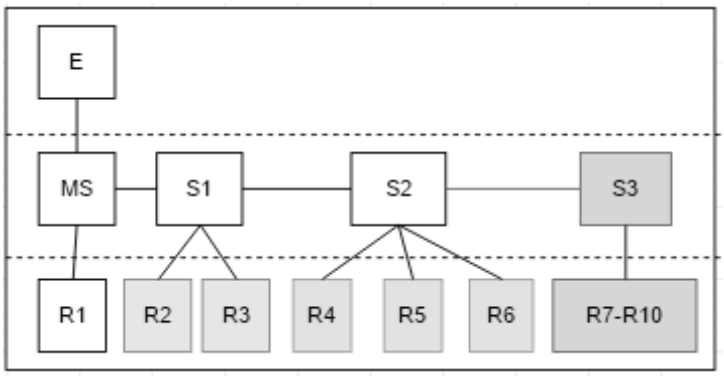
Paveikslėlyje (žr. 4.1 pav.) yra pateikiamas pirmasis eksperimento scenarijus. Pirmuoju scenarijaus atveju, programa yra sudaryta tik iš pagrindinės paslaugos, kuri naudoja dešimt duomenų eilės paslaugų (angl. *repositories*).



4.1 pav. Pirmas eksperimento scenarijus

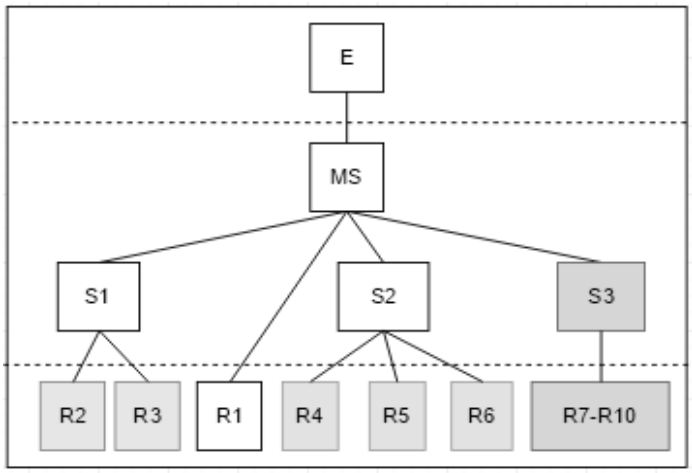
- E – paslaugos sąsaja (angl. *service endpoint*), leidžianti iškviesti paslaugų operacijas.
- MS – pagrindinė paslauga (angl. *main service*), įgyvendinanti pagrindinę programos operaciją.
- SX – kitos paslaugos, kurias naudoja pagrindinė paslauga. X - nurodo paslaugos numerį. Kiekviena paslauga turi po vieną operaciją, kuri iškviečia kitų paslaugų operacijas.
- RX – duomenų saugyklos (angl. *repositories*), kurios taip pat yra paslaugos, tačiau yra orientuotos į darbą su duomenimis. Kiekviena paslauga turi po vieną operaciją – duomenų įrašymo. X - nurodo saugyklos numerį.

Antruoju scenarijaus atveju (žr. 4.2 pav.) programos struktūra sudaryta iš 4 paslaugų vidurinėje eilėje. Šiuo atveju, pagrindinė paslauga (MS) naudoja S1 paslaugos operaciją, kuri iškviečia R2, R3 duomenų saugyklų operacijas. S1 paslauga taip pat naudoja S2 paslaugą, kad pasiektų kitas duomenų saugyklas (R4, R5, R6), kurios reikalingos MS pagrindinei operacijai atlikti. S2 paslauga naudoja S3, kad pasiektų likusias duomenų saugyklų paslaugas (R7, R8, R9, R10). Šio scenarijaus metu, rezultatas yra toks, jog pagrindinė paslauga MS naudoja tik vieną duomenų saugyklą R1 ir vieną paslaugą S1.



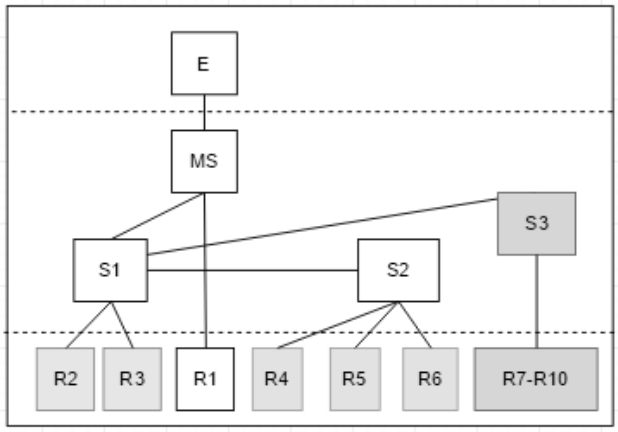
4.2 pav. Antras eksperimento scenarijus

Trečiuoju scenarijaus atveju (žr. 4.3 pav.), pagrindinė paslauga MS naudoja 1 duomenų saugyklą ir 3 paslaugas – S1, S2, S3. Šio scenarijaus atveju, pagrindinė paslauga atlieka tą pačią operaciją kaip ir pastaraisiais scenarijais, tačiau operacijos logika yra paskirstoma kelioms paslaugoms. Šiuo atveju programos struktūra yra labiau orientuota į paslaugas, o tai leidžia paslaugai naudoti kitų paslaugų operacijas. Reikia atkreipti dėmesį, kad toks paslaugų išdėstymas atliekamas tik vidurinėje eilėje, nes vadovaujantis n-eilės architektūros principais, ši eilė yra atsakinga už programos verslo logiką (angl. *business logic*).



4.3 pav. Trečias eksperimento scenarijus

Ketvirtuoju scenarijaus atveju, (žr. 4.4 pav.) programos struktūra panaši į pirmojo scenarijaus struktūrą. Šiuo atveju, S1 operacija naudoja S2 ir S3 operacijas. Pagrindinis skirtumas tarp šio ir kitų eksperimento metu vykdomų scenarijų yra tai, jog skiriasi paslaugų sąryšiai su kitomis paslaugomis. Tai reiškia, kad programos logika gali būti skirstoma skirtingais būdais skirtingoms paslaugoms, tačiau pagrindinės paslaugos vykdoma logika-operacija turi išlikti nepakitusi.



4.4 pav. Ketvirtas eksperimento scenarijus

Eksperimento pabaigoje yra įvertinama, kaip keičiasi priežiūros indeksas, esant skirtingiems paslaugų pasiskirstymams, kas turi įtakos šiems pokyčiams bei kokios koreliacijos yra tarp skirtingų programos kodo metrikų. Remiantis eksperimentine medžiaga, išvadose yra pateikiami pasiūlymai, kaip galima pagerinti sukurtos pagrindinės (darbuotojų vertinimo) sistemos priežiūrą.

4.3. Eksperimento aplinka

Eksperimento aplinkos ir programinės įrangos parametrai:

- Visual Studio 2015 Professional
- Programavimo kalba C# 6 (NET 4.5)
- Operacinė sistema: Windows 10 Professional 64 bitų
- Operatyvioji atmintis: 6 GB DDR3
- Procesorius: Intel Core i7-3632QM

4.4. Eksperimento rezultatų analizė

Eksperimento pradžioje yra įvertinamas sukurtos darbuotojų vertinimo sistemos priežiūros indeksas ir kitos kodo metrikos. Vertinimų rezultatai pateikti tolimesnėje lentelėje (žr. 4.1 lentelė). Pristatymo eilėje esančių modulių priežiūros indeksas yra mažiau aktualus, nes didžiąją dalį kodo šiuose moduliuose sudaro .NET struktūros – *ASP.NET MVC 4* ir *ASP.NET WebAPI*. Remiantis vertinimo rezultatais galima teigti, jog mažiausias priežiūros indeksas yra duomenų eilėje – indekso reikšmė yra 54. Panagrinėjus programos kodą ir atsižvelgiant į LOC metriką, kuri yra didžiausia tarp visų modulių, galima teigti, kad didžioji sistemos logika yra realizuojama ne verslo logikos eilėje, bet duomenų eilėje. Verslo logikos eilėje esantis modulis *Application.Data* yra atsakingas už sistemos verslo logiką, tačiau atsižvelgiant į šio modulio LOC metriką, galima teigti, jog modulio apimtis yra gerokai mažesnė lyginant su *Infrastructure.Data* moduliu. Galima daryti prielaidą, kad *Application.Data* modulyje yra mažas paslaugų pasiskirstymas ir paslaugų skaičius. Likusieji moduliai *Domain.Data* ir *Application.Data.DTO* yra atsakingi už esybių ir duomenų perdavimo klasių aprašymus. Šių modulių metrikos yra geros, nes moduliuose nėra vykdomos sudėtingos operacijos.

4.1 lentelė. Darbuotojų vertinimo programos kodo metrikos

Modulis	MI	CC	LOC
Pristatymo eilė (angl. *service/presentation tier*)			
WebClient	85	603	1286
WebAPI	83	789	1478
Verslo logikos eilė (angl. *business tier*)			
Application.Data.DTO	92	434	443
Application.Data	81	261	646
Duomenų eilė (angl. *data tier*)			
Domain.Data	93	532	541
Infrastructure.Data	54	593	1941

Įvertinus sukurtos sistemos metrikas buvo vykdomi eksperimentai su keturiais scenarijais, kurie yra aprašomi eksperimentų metodologijos skyriuje (žr. 4.2 skyrius). Pirmojo eksperimento scenarijaus (žr. 4.1 pav.) rezultatai yra pateikiami žemiau esančioje lentelėje (žr. 4.2 lentelė). Kiekviena lentelės

eilutė vaizduoja eksperimentinio scenarijaus žingsnį, kurio metu prie pagrindinės programos paslaugos yra prijungiama nauja paslauga arba duomenų saugyklos paslauga (angl. *repository*). Pirmieji du stulpeliai vaizduoja, kiek paslaugų ir saugyklų naudoja pagrindinė programos paslauga. Pirmojo scenarijaus atveju, visais žingsniais buvo prijungiamos tik naujos duomenų saugyklų paslaugos, kurios leidžia įgyvendinti pagrindinės paslaugos operaciją. Remiantis eksperimento metu gautais rezultatais, galima teigti, jog pirmojo scenarijaus metu programos priežiūros indeksas aiškiai mažėja. Kitos kodo metrikos, tokios, kaip ciklomatinis sudėtingumas (angl. *cyclomatic complexity*) ir kodo sąsaja (angl. *code coupling*) didėja. Taip nutinka, nes kiekvienu žingsniu prie pagrindinės paslaugos yra prijungiama nauja duomenų saugyklos paslauga, kuri padidina pagrindinės paslaugos operacijos sudėtingumą.

4.2 lentelė. Pirmojo scenarijaus rezultatai

Nr.	**Naudojamos paslaugos**	**Naudojamos duomenų saugyklų paslaugos**	**Kodo metrikos**			
			MI	***CC***	**LOC**	***Kodo sąsaja* (angl. *coupling*)**
1	0	1	93	3	4	2
2	0	2	92	4	5	3
3	0	3	89	5	6	4
4	0	4	88	6	7	5
5	0	5	86	7	8	6
6	0	6	86	8	9	7
7	0	7	85	9	10	8
8	0	8	84	10	11	9
9	0	9	84	11	12	10
10	0	10	84	12	13	11

Antrojo eksperimento scenarijaus (žr. 4.2 pav.) rezultatai yra pateikiami žemiau esančioje lentelėje (žr. 4.3 lentelė). Kiekvieno eksperimentinio žingsnio metu prie pagrindinės paslaugos yra prijungiama nauja paslauga. Prijungta nauja paslauga realizuoja mažą dalį pagrindinės paslaugos operacijos. Kiekviena naujai prijungta paslauga turi vis didesnį naudojamų duomenų saugyklų skaičių. Kaip pateikta lentelėje (žr. 4.3 lentelė), šeštojo eksperimentinio žingsnio metu pagrindinė paslauga naudoja tik 2 paslaugas iš verslo logikos eilės. Šio žingsnio metu naudojamos 2 paslaugos, leidžia pasinaudoti 6 duomenų saugyklų operacijomis.

4.3 lentelė. Antrojo scenarijaus rezultatai

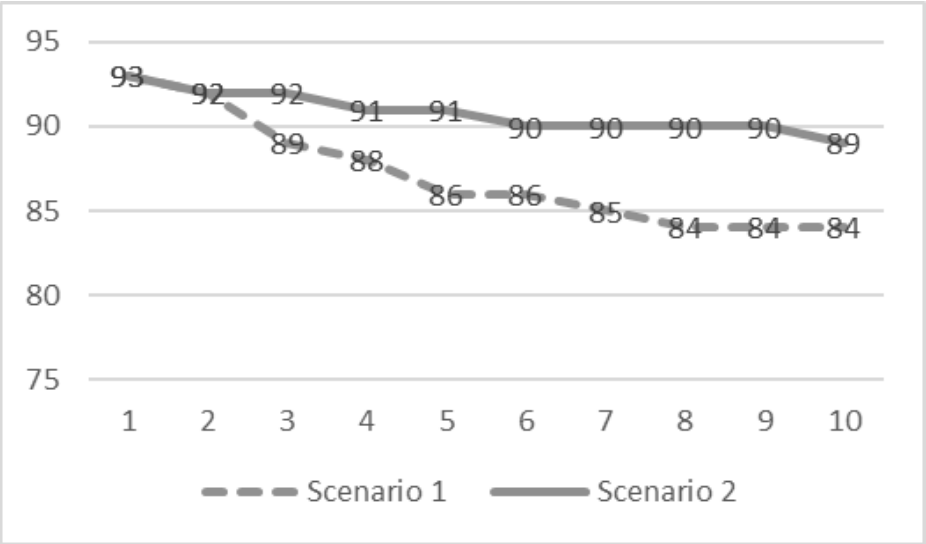
Nr.	Naudojamos paslaugos	Naudojamos duomenų saugyklų paslaugos	Kodo metrikos			
			MI	***CC***	**LOC**	***Kodo sąsaja* (angl. *coupling*)**
1	0	1	93	3	4	2
2	1	2	92	7	9	1
3	1	3	92	8	10	5
4	2	4	91	12	15	7
5	2	5	91	13	16	8
6	2	6	90	14	17	9
7	3	7	90	18	22	11
8	3	8	90	19	23	12
9	3	9	90	20	24	13
10	3	10	89	21	25	14

Vykdant pirmojo ir antrojo scenarijaus šeštąjį žingsnį yra gaunamos skirtingos kodo metrikos, nors naudojamų duomenų saugyklų skaičius yra toks pats. Pirmojo scenarijaus metu gaunama MI yra 86, antrojo scenarijaus metu - 90. Lyginant visus pirmojo ir antrojo scenarijaus metu gautus MI rezultatus, galima matyti, jog antrojo scenarijaus metu MI mažėjimo tempas yra lėtesnis (žr. 4.5 pav.). Viena iš šio kitimo priežasčių yra susijusi su Halstead‘o apimtimi (angl. *Halstead volume*), kuri nurodo programos algoritmo įgyvendinimo dydį – algoritmo apimtį. Eksperimentinėje programoje vertinamas algoritmas susideda iš operacijų, aprašytų duomenų saugyklų paslaugose. Antrojo scenarijaus metu pagrindinės programos algoritmo operacijos yra paskirstomas tarp trijų paslaugų. Šiuo atveju, pagrindinės paslaugos vykdomų operacijų skaičius sumažėja iki dviejų, nes likusiomis algoritmo operacijomis pasirūpina kitos paslaugos. Vaizdinis pasiskirstymas pateiktas antrojo scenarijaus paveikslėlyje (žr. 4.2 pav.), kuriame pagrindinė paslauga MS naudoja tik R1 ir S2 paslaugas. Priešingas atvejis yra gaunamas pirmojo scenarijaus metu (žr.4.1 pav.), kuomet pagrindinė paslauga tiesiogiai naudoja visas 10 duomenų saugyklų operacijų. Šiuo atveju, pagrindinės paslaugos algoritmas nėra paskirstomas ir yra vykdomas vienoje paslaugoje. Pirmojo scenarijaus metu vidutinė Halstead‘o apimtis yra gerokai didesnė už antrojo scenarijaus. Taip yra todėl, kad antrojo scenarijaus algoritmas yra paskirstytas į atskiras paslaugas, o rezultate skaičiuojant MI yra naudojama vidutinė Halstead‘o apimtis. Halstead‘o apimties reikšmė yra išreikšta per MI formulę, nes Visual Studio naudojama MI formulė yra šiek tiek modifikuota.

- Pirmojo scenarijaus Halstead‘o apimtis – ~0,038387
- Antrojo scenarijaus Halstead‘o apimtis - ~0,000649

Atlikus eksperimentus su likusiais scenarijais, priežiūros indeksas išlieka toks, kaip antrojo scenarijaus metu. To priežastis yra tai, jog skirtingo scenarijaus metu kodo metrikos pasiskirsto

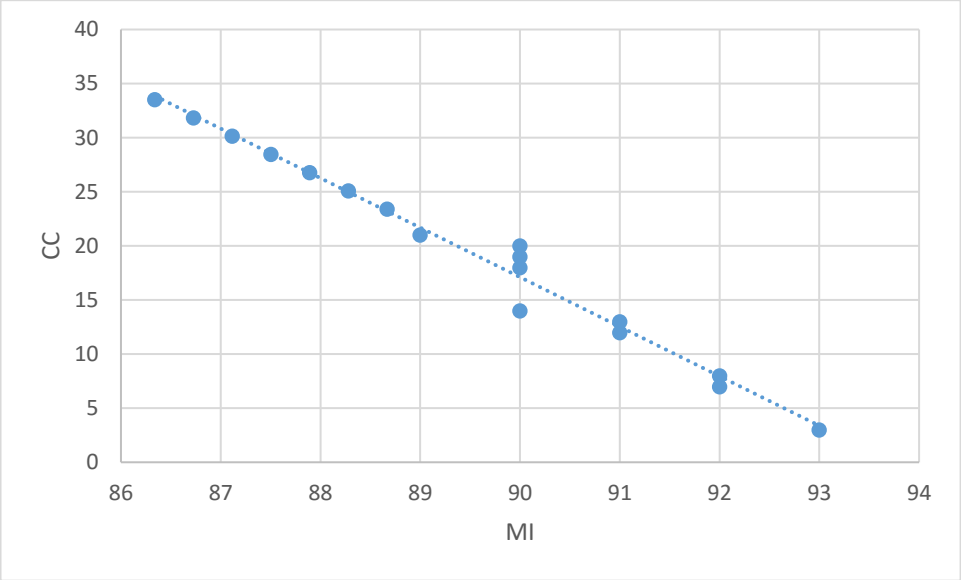
skirtingai, bet skaičiuojant programos MI yra naudojamos vidutinės programos metrikų reikšmės. Trečiojo ir ketvirtojo scenarijaus metu nebuvo pridedama jokių naujų paslaugų, kito tik sąsajos tarp paslaugų.



4.5 pav. MI kitimas 1 ir 2 scenarijaus metu

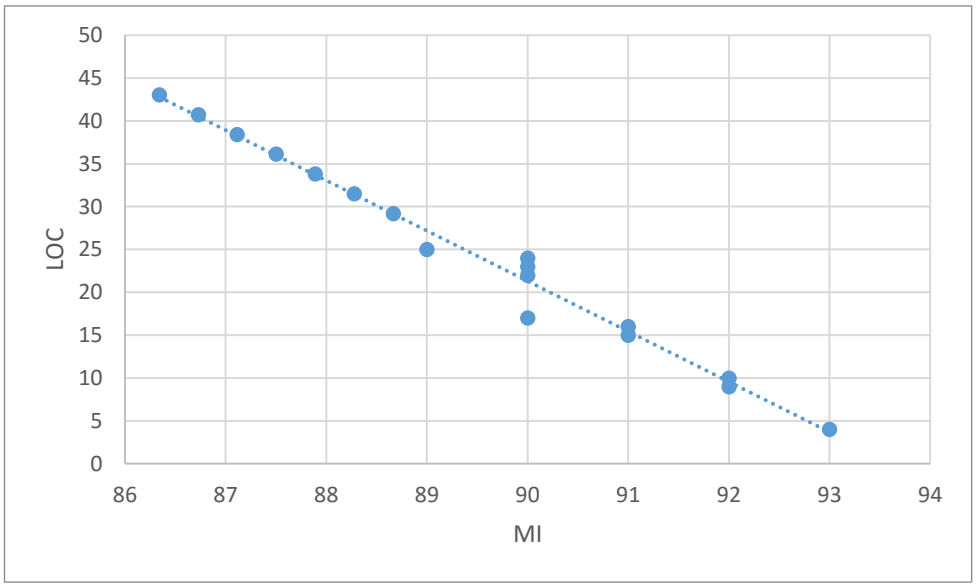
Kitos priežastys, kurios turi įtskos MI yra CC ir LOC. Kad nustatyti, kaip kitos metrikos įtakoja MI, buvo vertinamas statistinis ryšys tarp MI ir pasirinktos metrikos (*Pirsono* – tiesinio ryšio stiprumo matas) bei skaičiuojamas koreliacijos koeficientas – koreliacijos stiprumui nustatyti. Šie skaičiavimai buvo atliekami naudojant antrojo scenarijaus duomenis (žr. 4.3 lentelė), kurie buvo gauti didinant programos algoritmą – paslaugų skaičių. Analizuojant eksperimento metu gautus duomenis svarbu atkreipti dėmesį į tai, kad vienos metrikos gali turėti įtakos kitų metrikų kitimui. Vienas iš pavyzdžių, kuomet didėjanti LOC metrika gali padidinti CC ir Halstead‘o apimtį. Taip gali nutikti, jei programos kodo apimtis (LOC) didėja dėl programos funkcionalumo didinimo. Šiuo atveju, programos algoritmas didėja ir sudėtingėja, taip darydamas įtaką kitoms metrikoms ir priežiūros indeksui.

Žemiau pateiktame paveikslėlyje (žr. 4.6 pav.) yra pateikiamas grafikas, kaip keičiasi priežiūros indeksas kintant ciklomatiniam sudėtingumui. Iš grafiko galima pastebėti, jog mažėjant programos ciklomatiniam sudėtingumui, didėja programos MI metrika. Šiuo atveju turime neigiamą koreliacijos koeficientą, kuris yra pateikiamas žemiau pateiktoje lentelėje (žr. 4.4 lentelė).



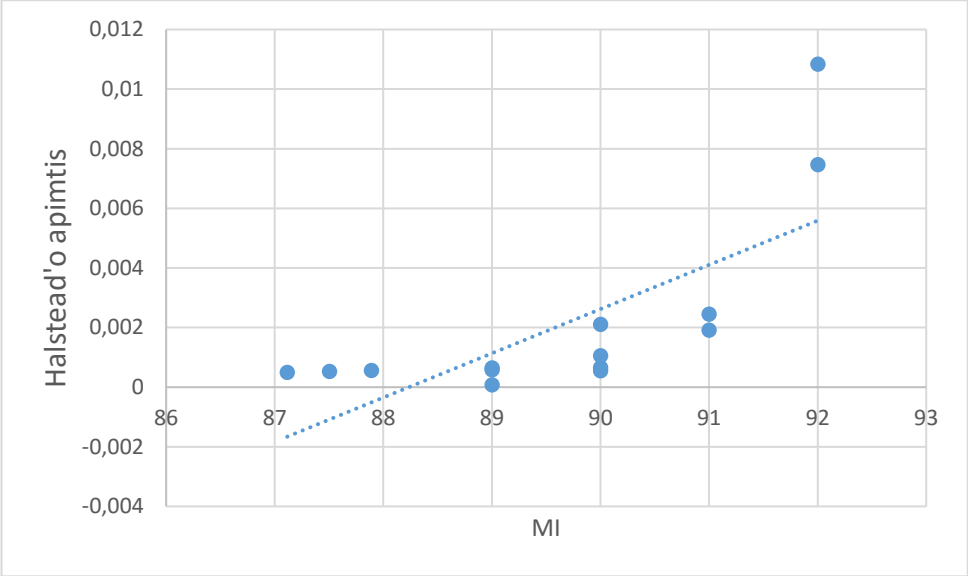
4.6 pav. CC ir MI koreliacija

Pateiktame paveikslėlyje (žr. 4.7 pav.) yra pateiktas grafikas, vaizduojantis kaip kinta MI reikšmė, keičiantis programos kodo eilučių skaičiui - LOC. Iš grafiko aiškiai matyti, jog mažėjant LOC metrikai programos priežiūros indeksas didėja.



4.7 pav. LOC ir MI koreliacija

Paveikslėlyje 4.8 pav. pateiktas grafikas, kuriame yra pavaizduota, kaip kinta priežiūros indeksas, keičiantis Halstead‘o apimties metrikai. Iš gautų rezultatų galima pastebėti, jog didėjant programos algoritmui (atsirandant naujoms paslaugoms) vidutinė Halstead‘o apimtis mažėja. Tačiau svarbu atkreipti dėmesį į tai, kad mažėjant Halstead‘o apimčiai tuo pačiu metu CC ir LOC metrikos keičiasi ir daro įtaką MI metrikos reikšmei. Jei didėtų tik Halstead‘o apimtis, o likusios metrikos nekistų, tokiu atveju MI mažėtu, tačiau toks scenarijus nėra įmanomas.



4.8 pav. *Halstead Volume* ir MI koreliacija

Žemiau esančioje lentelėje (žr. 4.4 lentelė) yra pateikiami eksperimento metu gautų metrikų koreliacijos koeficientai, kurie yra paskaičiuoti pagal 1 formulę.

$$r = \frac{\sum(x-\bar{x})(y-\bar{y})}{\sqrt{\sum(x-\bar{x})^2 \sum(y-\bar{y})^2}} \qquad (1)$$

Formulėje esanti *x* reikšmė yra MI, o *y* yra kita pasirinkta metrika. Šie koeficientai yra koreliacijos stiprumo matas. Jei dviejų kintamųjų koreliacijos koeficientas lygus nuliui, tai tie kintamieji yra statiškai nepriklausomi. Iš gautų rezultatų galima daryti išvadas, jog didėjant programos ciklomatiniam sudėtingumui, LOC ir *Coupling* metrikoms programos priežiūros indeksas mažėja, nes *r* reikšmės artimos -1. Esant teigiamam Halsteaed'o apimties koeficientui galima pastebėti, jog MI didėja. Reikia atkreipti dėmesį, į tai, kad šis dėsningumas galioja tuo atveju, jei keičiantis programai, kinta visos priežiūros indeksą įtakojančios metrikos. Iš eksperimento metu gautų duomenų, kurių pagalba yra paskaičiuoti šie statistiniai ryšiai, galima pastebėti, jog Halstead'o apimties ryšys su MI metrika yra šiek tiek silpnesnis.

4.4 lentelė. Koreliacijų koeficientai

Nr.	**Koreliacijos koeficientai**	
	Metrika	***r***
1	Ciklomatinis sudėtingumas (CC)	-0,96
2	Kodo sąsaja (angl. *Code coupling*)	-0,94
3	Kodo eilučių skaičius (LOC)	-0,96
4	Halstead apimtis (HV)	0,84

5. IŠVADOS

1. Suprojektuota ir realizuota programinė įranga, leidžianti vertinti įmonės darbuotojus ir kurios architektūra atitinka SOA principus.
2. Tyrimo metu išmatuoti sukurtos programinės įrangos atskirų eilių priežiūros indeksai, kurių pagalba buvo nustatyta, jog žemiausias priežiūros indeksas yra duomenų eilėje. Šios eilės priežiūros indeksas 38% mažesnis už likusių programos eilių vidutinį priežiūros indeksą.
3. Tyrimo eigoje buvo išsiaiškinta, jog sukurtos darbuotojų vertinimo sistemos paslaugų pasiskirstymas nėra teisingas. Didžioji programos paslaugų dalis yra realizuota duomenų eilėje, o pačios paslaugos yra didelės apimties ir sudėtingumo. Šios eilės apimtis yra 44% didesnė už likusių eilių vidutinę apimtį.
4. Atlikus tyrimą buvo nustatyta, jog paskirstant programos algoritmą į atskiras paslaugas (pritaikius SOA principus), galutinė programos priežiūros indekso reikšmė padidėja 10%. Didėjant programos apimčiai, didėja ir priežiūros indekso reikšmė.
5. Atlikus tyrimą ir palyginus pirmojo ir antrojo scenarijaus rezultatus galima teigti, jog paskirsčius programos algoritmą į atskiras paslaugas, du kartus padidėja tokios metrikos, kaip ciklomatinis sudėtingumas bei LOC, tačiau sumažėja vidutinė Halstead‘o apimtis. Rezultate priežiūros indekso reikšmė gaunama 10% didesnė – programa yra lengviau prižiūrima.
6. Tyrimo metu analizuojant ryšius tarp priežiūros indekso ir jį veikiančių metrikų, kuomet programos algoritmas didėja, buvo nustatyta, jog mažėjant priežiūros indeksui, programos ciklomatinis sudėtingumas ir LOC didėja, o Halstead‘o apimtis mažėja.
7. Siekiant padidinti sukurtos darbuotojų vertinimo sistemos priežiūros indeksą, buvo nuspręsta mažinti paslaugų apimtis ir didinti jų skaičių 50%. Taip pat paslaugas, kurios nesusijusios su duomenų operacijomis, perkelti į vidurinę eilę.
8. Norint gauti aukštą programinės įrangos priežiūros indeksą, reikia atkreipti dėmesį į tai, jog priežiūros indekso reikšmė yra priklausoma nuo kelių metrikų: Halstead‘o apimties, ciklomatinio sudėtingumo bei LOC. Reikia įvertinti ir tai, jog šios metrikos gali paveikti viena kitą.

6. LITERATŪRA

[1] A. S. Dino Esposito, Microsofth .NET Architecting Applications for the Enterprise, Microsoft Press, 2009.

[2] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design, 01 01 2005.

[3] D. Panda, „An Introduction to Service-Oriented Architecture from a Java Developer Perspective," 01 26 2005. [Tinkle]. Available: http://archive.oreilly.com/pub/a/onjava/2005/01/26/soa-intro.html. [Kreiptasi 22 10 2016].

[4] C. Hartwich, „Why It Is So Difficult to Build N-Tiered Enterprise," 2001. [Tinkle]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.19.5496&rep=rep1&type=pdf. [Kreiptasi 26 10 2016].

[5] Microsoft, „N-Tier Data Applications Overview," [Tinkle]. Available: https://msdn.microsoft.com/en-us/library/bb384398.aspx. [Kreiptasi 29 10 2016].

[6] S. Fahmy, „Web Services Software Architecture," W3Schools, [Tinkle]. Available: ttp://ceur-ws.org/Vol-169/paper3.pdf. [Kreiptasi 20 10 2016].

[7] T. DelChiaro, „Why Use "REST" Architecture for Web Services?," [Tinkle]. Available: http://edn.embarcadero.com/article/40467. [Kreiptasi 21 04 2016].

[8] „Web Services Architecture," W3C, 04 02 2004. [Tinkle]. Available: http://www.w3.org/TR/ws-arch/. [Kreiptasi 15 04 2016].

[9] R. T. Fielding, „Architectural Styles and the Design of Network-based Software Architectures," 2000. [Tinkle]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm. [Kreiptasi 21 04 2016].

[10] M. A. Team, „What's New in ASP.NET Web API 2.1," Microsoft, 20 01 2014. [Tinkle]. Available: http://www.asp.net/web-api/overview/releases/whats-new-in-aspnet-web-api-21#new-features. [Kreiptasi 22 04 2016].

[11] „ServiceStack wiki," ServiceStack, [Tinkle]. Available: https://github.com/ServiceStack/ServiceStack/wiki. [Kreiptasi 22 04 2016].

[12] „Model-View-Controller," Microsoft, [Tinkle]. Available: http://msdn.microsoft.com/en-us/library/ff649643.aspx. [Kreiptasi 23 04 2016].

[13] „ASP.NET MVC 4," Microsoft, [Tinkle]. Available: http://www.asp.net/whitepapers/mvc4-release-notes. [Kreiptasi 24 04 2016].

[14] „The MVVM Pattern," Microsoft, 10 02 2012. [Tinkle]. Available: http://msdn.microsoft.com/en-us/library/hh848246.aspx. [Kreiptasi 24 04 2016].

[15] B. T. a. R. Kagan, „The Battle of Modern Javascript Frameworks,“ 10 04 2013. [Tinkle]. Available: http://www.softfinity.com/blog/the-battle-of-modern-javascript-frameworks-part-i/. [Kreiptasi 25 04 2016].

[16] J. COGSWELL , „SQL vs. NoSQL: Which Is Better?,“ 16 07 2012. [Tinkle]. Available: http://news.dice.com/2012/07/16/sql-vs-nosql-which-is-better/. [Kreiptasi 26 04 2016].

[17] J. Steemann, „Bulk inserts in MongoDB, CouchDB, and ArangoDB,“ ArangoDB, 04 09 2012. [Tinkle]. Available: https://www.arangodb.org/2012/09/04/bulk-inserts-mongodb-couchdb-arangodb. [Kreiptasi 10 05 2016].

[18] „DB-Engines Ranking,“ Solid IT, 05 2014. [Tinkle]. Available: http://db-engines.com/en/ranking. [Kreiptasi 10 05 2016].

[19] S. H. Kan, Metrics and Models in software quality engineering, Canada: Person education inc., 2003.

[20] S. K. Simardeep Kaur, „A Review on Metrics in SOA,“ 2016. [Tinkle]. Available: http://www.ijarcce.com/upload/2016/july-16/IJARCCE%2081.pdf. [Kreiptasi 20 02 2017].

[21] R. Land, „Measurements of Software Maintainability,“ Mälardalen University, Västerås, Sweden.

7. TERMINAI IR SANTRUMPŲ ŽODYNAS

UML - (ang. *Unified Modeling Language*) standartinė kalba, skirta specifikuoti, vizualizuoti ir dokumentuoti programinės įrangos sistemas, taip pat naudojama ir verslo modeliavimui ir kitokioms ne programinėms sistemoms.

SOA - SOA – tai į paslaugas orientuotas programinės įrangos architektūrinis modelis (angl. *service-oriented architecture*). SOA yra įvardijama kaip kompiuterijos paradigma. Kitos kompiuterijos paradigmos, tokios, kaip objektinė orientacija (angl. *object orientation*) dalijasi tam tikrais principais ir tokiu būdu tarsi apibrėžia SOA pavadinimą. Pagrindinis SOA modelio bruožas yra tai, jog jis išskiria sistemos dalis į atskirus modulius, kurie yra atsakingi tik už konkretų procesą sistemoje.

SQL - (ang. *Structured Query Language*) tai kompiuterinė kalba, skirta darbui su duomenų rinkiniais ir jų tarpusavio santykiais

MEF - (angl. *Managed Extensibility Framework*) biblioteka skirta kurti lengvai praplečiamas sistemas, kurios gali būti apjungiamas viena su kita be jokių papildomų konfiguracijų.

API - (angl. *Application Programming Interface*) programos sąsaja, kurios pagalba kitos sistemos gali bendrauti viena su kita.

REST - Architektūrinis stilius skirtas kurti tinkle veikiančias programas.

JSON - Duomenų formatas, skirtas perduoti duomenis tarp serverio ir internetinės programos.

HV - Halstead‘o apimtis (angl. *Halstead Volume*).

MI - Priežiūros indeksas (angl. *Maintainability index*).

LOC - Kodo eilutės (angl. *Lines of code*).

CC - Ciklomatinis sudėtingumas (angl. Cyclomatic complexity).

Service oriented architecture evaluation based on maintainability index

Edgaras Norvaiša
Kaunas University of Technology
Faculty of software engineering
Kaunas, Lithuania
edgaras.norvaisa@ktu.edu

Šarūnas Packevičius
Kaunas University of Technology
Faculty of software engineering
Kaunas, Lithuania
Sarunas.packevicius@ktu.edu

Software architecture is meant to define system structure which is a core for product development. Defying system structure is first and most important step in software development. It involves a series of decisions based on a wide range of factors, and each of these decisions can have considerable impact on the quality, performance, maintainability, and overall success of the application. Most efficient way to be able to control system characteristics is to use certain architectural models like SOA or any other. In this paper focus object will be how maintainability is effected for SOA based applications.

Keywords— SOA, metrics, maintainability

INTRODUCTION

Decision to analyze SOA maintainability was made because most of the enterprise projects chose to use SOA as a base for architecture but this does not mean that it guaranties good application characteristics. This research is focused to analyze and define how maintainability index is changing dependently on using SOA structure by comparing different system layouts. Main goal of experiment was to find out how maintainability index changes for SOA based application and what is effecting these changes. To initiate experiment analysis of SOA and software metrics was made. Explaining principles of SOA architecture and metrics that are used to measure software maintainability. Experiment part describes several approaches how SOA can be structured and how that structure impacts maintainability for whole application. The result will allow determine how maintainability is effected by using SOA principles and what architectural approach should be chosen to rise system maintainability.

DESCRIPTION OF SERVICE ORIENTED ARCHITECTURE

Definition of SOA

Service oriented architecture as Mamoun A. Hirzalla explains "SOA is emerging as a promising development paradigm, which is based on encapsulating application logic within independent, loosely-coupled stateless services, that interact via messages using standard communication protocols and can be orchestrated using business process languages, The notion of a service is similar to that of a component, in that services, much like components, are independent building blocks that collectively represent an application."[8]. This statement describes that SOA focuses on having loosely-coupled services that are like independent building blocks used for building application. The benefit of SOA is that it provides loosely coupling where in a result we have flexible, scalable and reusable application.

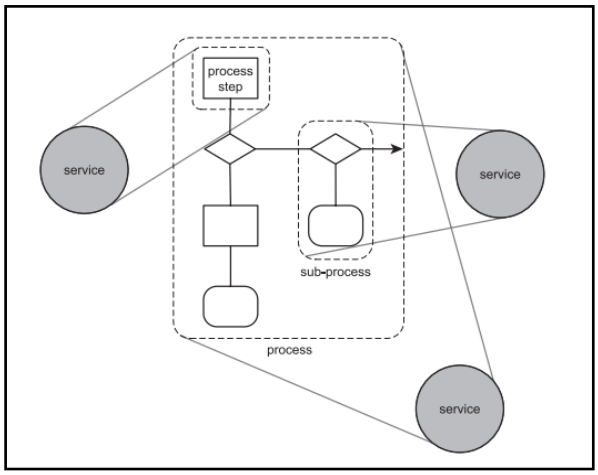


Figure 1. Distributed application services.

Figure 1 illustrates how services can look like and how they can interact with each other. It shows that a set of services can be a part of a bigger process that can be used to describe a larger business cases or algorithm. To process a certain business case, we call a service which is responsible for it. Service can be built from other smaller services which implement smaller parts of a main service. To communicate with services a communication layer is needed which would provide interface and description of network services. Services can be accessed directly an invoking client or through service broker (ESB) which looks up the address of required services through a registry component, retrieves the Web Service Definition Language (WSDL) file, and then binds to that service during the invocation process. ESB responsible for routing and translating requests and responses between requestors and service providers (e.g. figure 2). In the context of Service Oriented Architecture, Web Services are used to facilitate communication between service providers and service consumers. Web based applications are communicating using the concepts as XML, SOAP, REST, WSDL and UDDI. [5]

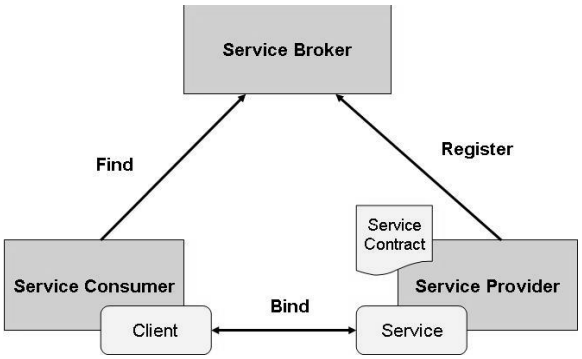


Figure 2. SOA design.

As mentioned before SOA is paradigm, which is based on application logic encapsulation. In such case, there is a need to have a structure for encapsulated logic – services [1].

A. N-tier architecture

In Christoph Hartwich article about difficulties in building n-tier enterprise application he describes how application can be divided in to separate tiers. "A tier is a layer that corresponds to a process or a collection of processes. A tier contains all artifacts of a software system that can be associated with the tier's process(es)." [2]. N-tier approach divides application architecture in to separate layers which are loosely coupled. Application can have multiple tiers but in most cases, there are main 3 tiers. Each tier has its own responsibility and it's only accessible for neighbors and greater tier. Top tier is a presentation tier which provides an interface for clients to business logic. Second tier is a business logic where system logic is predefined in to services. Business tier can be accessed only by presentation tier where business tier can access data tier. Data tier is responsible for granting access to repositories. Data tier can be accessed only by business tier services (e.g. figure 3).

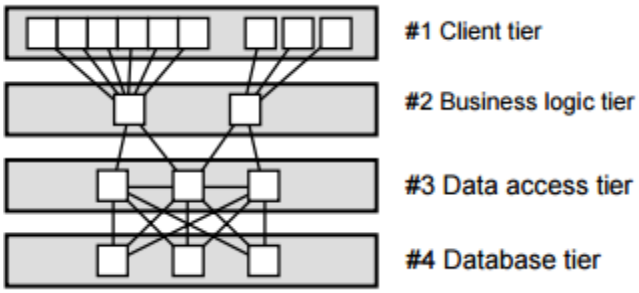


Figure 3. N-tier structure.

To summarize SOA is an enterprise architectural style rather than an application architectural style. SOA doesn't focus on individual application architecture where n-tier does it. In the end, we have a set of distributed services across the application which are published by service broker and can be accessed by other consumers (e.g. figure 4).

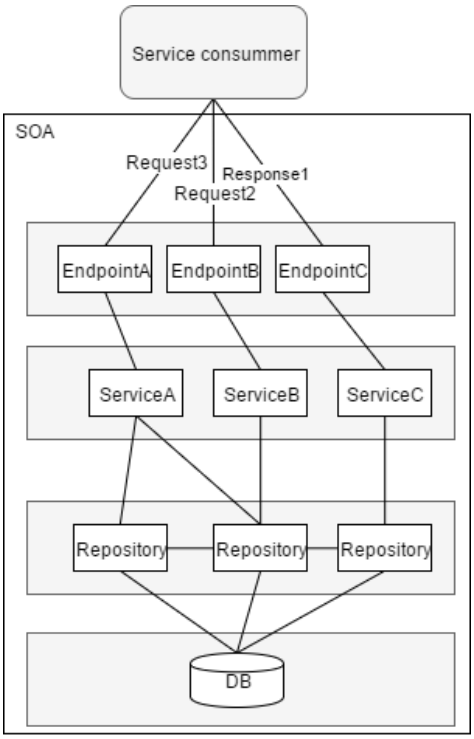


Figure 4. SOA based design.

SOFTWARE METRICS

To define software quality and complexity software metrics are used. Software metrics are widely used among software developers and clients who makes requirements for their software quality. There are multiple methods and algorithms how to calculate software metrics but these days in most case Halstead techniques are used. In this topic, we will define few of the metrics that was used to calculate maintainability index in experiment [6].

Lines of code (LOC) metric

One of the metrics is line of code (LOC) which is used to determine size of software. Using this metric, it is possible to evaluate size of a software but to do it correctly a prime calculation object need to be defined. It can be done in multiple ways [7]:

Calculate only executable lines
Calculate executable lines and assignments of values
Calculate executable lines, assignments and comments
Calculate lines in input screen

Because of multiple approaches of how code lines can be calculated LOC is not very inaccurate metric although it's used in maintainability metric calculation.

McCabe's cyclomatic complexity metric

Second important metric in evaluating software is McCabe's cyclomatic complexity. It is one of the most used software metrics. This metric was invented by Thomas J. McCabe 1976 year. This is one of the most accurate software metrics that can be calculated and is widely used in software which calculating metrics of other software programs. Cyclomatic complexity (CC) is classical graph theory cyclomatic number, indicating the

number of regions in a graph. As applied to software, it is the number of linearly independent paths that comprise the program. As so it can be used to indicate effort required to do testing. This metric was design to indicate a program's testability and maintainability [7]. The general formula to compute it is:

$$M = V(G) = E - N + 2P \quad (1)$$

V(G) – Cyclomatic number of G
E – Number of edges
N – Number of nodes

P – Number of unconnected parts of the graph

In functional programing calculation of cyclomatic complexity is easy because each operation is executed after each other. In OO (object oriented) case it is much complicated because operations are divided in functions, classes and objects. Best approach to calculate CC for OO is to calculate it per functions. By McCabe CC should not be bigger than 10 so if function or a method has bigger CC than 10 code needs to be simplified or divided. Using this metric for software evaluation it allows to optimize a whole size and complexity of application as well as increase maintainability [7].

Halstead metrics

Last metric which is used in MI calculation is Halstead volume. Halstead software science measurements where introduced in 1977 to estimate the program difficulty and other features like development effort and project number of faults. A computer program, according to software science, is a collection of tokens that can be classified as either operators or operands [7]. The measures of Halstead are based on

n1 = number of unique or distinct operators
n2 = number of unique or distinct operands
N1 = total usage of all the operators
N2 = total usage of all the operands

Based on these primitive measures Halstead developed a system of equations expressing the total vocabulary. Following metrics [7]:

$$N = N1 + N2 \quad (2)$$

$$n = n1 + n2 \quad (3)$$

$$V = N \log_2(n) \quad (4)$$

$$D = \frac{n1}{2} * \frac{N2}{n2} \quad (5)$$

$$E = D * V \quad (6)$$

N – program length
n – program vocabulary
V – volume
D – difficulty
E - effort

Maintainability index

Maintainability index is a composite metric that incorporates several traditional source code metrics into a single number that indicates relative maintainability. As Oman and Hagemeister explains MI is comprised of weighted Halstead metrics (effort or volume), McCabe's Cyclomatic Complexity, lines of code (LOC), and number of comments [3] [4]. The original formula to compute MI is:

$$MI = 171 - 5.2 * \ln(HV) - 0.23 * CC - 16.2 * \ln(LOC) \quad (7)$$

MI – maintainability index
HV –Halstead volume per module
CC – cyclomatic complexity per module
LOC – lines of code

Having full understanding of how software can be evaluated and measured this knowledge was used to initiate experiment for evaluating SOA based software maintainability index. In next topics experiment methodology will be defined.

METHODOLOGY OF THE RESEARCH

To evaluate maintainability index for SOA principles based application experimental program was created. In experimental part maintainability will be measured using different application structure layouts. System structure in most of the cases will be service oriented so there will be a set of services that will implement certain logic of one of the main service. Main service will consume other services in different ways – straight calls to service or will use other services to call other service operation. Basic layout of all structures will be based on n-tier principles so there will be free layers – presentation tier which will only define main service interface and provide ability to call it from other clients. Business tier will be responsible for all the services that will be used to solve a main service operation, as well as main service will be in this tier. Repository tier will be responsible for data access and all upper tier services will be able to consume these repositories.

For calculating metrics in experiment integrated Visual Studio tools were used because experiment is executed on .NET environment using C# programing language. Formula used for measuring maintainability in Visual Studio is the same. The new definition merely transforms the index to a number between 0 and 100. Furthermore, Visual Studio provides an interpretation:

MI >= 20 high maintainability
10 <= MI < 20 moderate maintainability
MI < 10 low maintainability

In experiment four different structures will be measured. First structure in figure 5 contains of only main service which calls data insert operations in ten repositories. Where:

E – end point for client to execute main service
MS – main service which handles main application logic operation
SX – other services that are consumed by main service
RX – repositories that provides access to data insert operations.

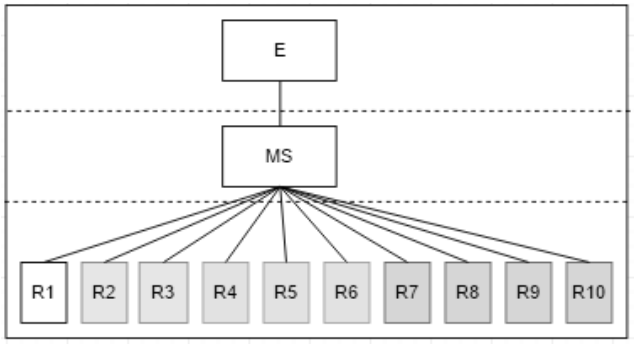


Figure 5. Experiment scenario 1.

Second structure consists of 4 services where main service is calling S1 service operation which is calling repositories R2, R3. S1 operation is calling S2 operation which works in same principle as S1 only number of consumed repositories is greater. Rest application structures are displayed in 6-8 figures. Main difference in structure is that the main service operation is distributed across the multiple services. Each structure only has a different operation call flow. As for example man service in figure 7 is calling distributed S1, S2 and S3 services to complete main MS operation.

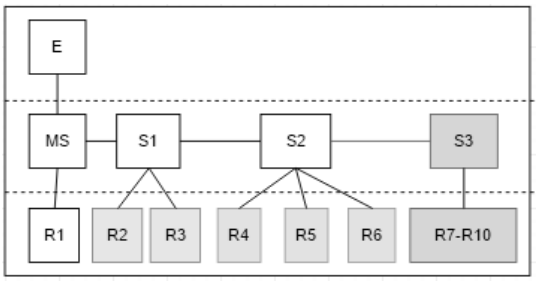


Figure 6. Experiment scenario 2.

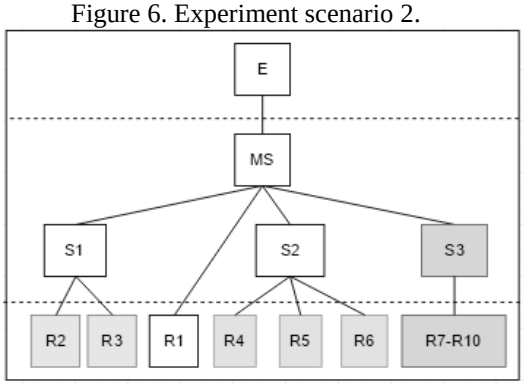


Figure 7. Scenario 3.

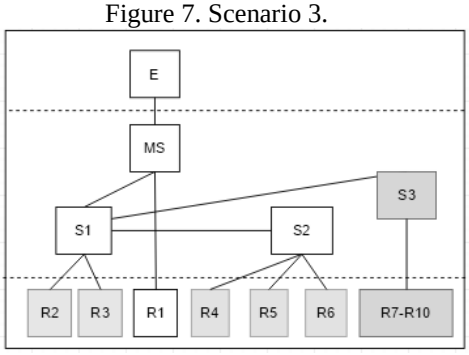


Figure 8. Scenario 4.

Experiment results

Experiment results of scenario 1 (Figure 5) are represented in table 1. Each row represents experiment step where we add a new service or repository to application structure. At the first scenario, the main service is not calling any other service it only calls repositories that allows completing main service logic-operation. This kind of structure is illustrating a default cases in applications where we don't use distributed services (non SOA approach). Service column in a table 1 defines how many services are consumed by the man service. Same approach goes for repositories column. Metrics column represents all the metrics like maintainability index, cyclomatic complexity and code coupling which were calculated by Visual Studio integrated tools. After executing experiment for first scenario we can see that MI is decreasing. This happens because each time we add a new repository to main service a complexity of application is increasing.

Scenario 1 results

Services	Repositories	Metrics		
		MI	***CC***	***Coupling***
0	1	93	3	2
0	2	92	4	3
0	3	89	5	4
0	4	88	6	5
0	5	86	7	6
0	6	86	8	7
0	7	85	9	8
0	8	84	10	9
0	9	84	11	10
0	10	84	12	11

Second scenario (Figure 6) results are represented in table 2. In this scenario, we define that main service on each experiment step starts to consume a new service which is providing functionality that resolves a small logical case of the main service operation. Also, after each experiment step we connect a new service which has more repositories in use then the previously added service. Compare to table 1 we can see that maintainability index is decreasing much slower. Despite that fact that cyclomatic complexity and code coupling is increasing much faster. Results of how maintainability index decreases are displayed in figure 9.

Scenario 2 results

Services	Repositories	Metrics		
		MI	***CC***	***Coupling***
0	1	93	3	2
1	2	92	7	1
1	3	92	8	5
2	4	91	12	7
2	5	91	13	8

Services	Repositories	Metrics		
		MI	*CC*	*Coupling*
2	6	90	14	9
3	7	90	18	11
3	8	90	19	12
3	9	90	20	13
3	10	89	21	14

Comparing the first and the second scenario results difference is noticeable on how fast MI is decreasing. Difference can be seen in figure 9. The reason why it happens is related to Halstead Volume metric which measures volume of algorithm. In this experiment algorithm, basically is the operations that main service is executing. On second scenario, main service operation logic is distributed across the other services so average Halstead Volume is smaller than it is on a first scenario. This is also can be seen on scenarios figures 5 and 6 of how many relations MS has to other objects. On figure 5 main service operation consists of several operations that are calling 10 repositories. On the other case in figure 6 we see that main service is only calling 1 repository and 1 service so in total main service operation consists of 2 calls. In scenario 2 rest main service operation logic is distributed in S1, S2 and S3 services but all these services consist of few operations that calls repositories and other service. In the end at scenario 2 we have average Halstead Volume metric which is smaller than in a first scenario.

Remaining scenarios showed in figures 7 and 8 results are the same as we had in scenario 2. Maintainability index changes in a same number only other metrics like CC is different. Reason again is related to Halstead Volume metric where we have a different value across whole modules-services but in the end maintainability index is calculated using average Halstead Volume value.

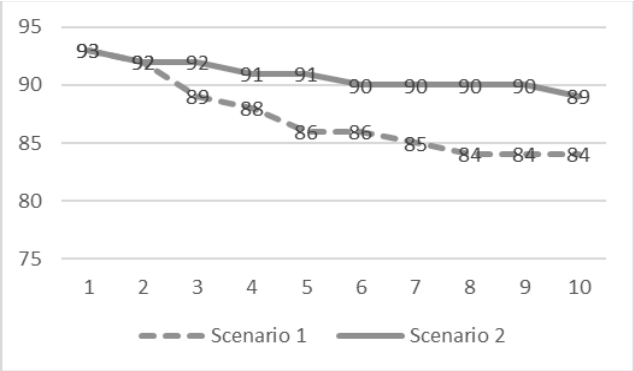


Figure 9. Scenario 1 and 2 chart of MI metrics per experimental step.

- CORRELATION BETWEEN METRICS

After experiment correlations where calculated for different types of metrics. For all calculations maintainability index was used as a prime metric and secondary metric was used from MI formula. Table 3 represents calculations results where first and second values (CC, Code coupling) has strong negative values. This means that if one of these metrics are increasing maintainability index is decreasing. Third metric – Halstead Volume has strong and positive correlation between MI. This means that when HV is increasing MI is increasing to. This positive correlation between MI and HV was noticed during experiment part when code metrics where measured for different system layouts in topic V.

CORRELATIONS BETWEEN METRICS

Nr.	Correlations with MI and metric	
	Metric	*r*
1	Cyclomatic complexity (CC)	-0.96
2	Code coupling	-0.94
3	Halstead Volume (HV)	0.84

CONCLUSIONS

1. Analysis part shows that SOA allows to have loosely coupled application components that can be distributed not even across application scope but also outside it.
2. Metrics analysis part shows that there are methods that allows evaluate software quality and complexity.
3. Experimental evaluation of different application layouts gave different maintainability index results. In case when we have no distributes services across application it becomes harder to maintain the application. In another case when we have distributed services (service oriented) maintainability of application is much easier. However, this conclusion is only applicable when we have a bigger scale application. This is one of the reason why SOA is used in enterprise.
4. Practical benefit of this research is that we can see that SOA approach is better for bigger scale application instead of small size projects. Also, research clearly shows that having big logical operations in one place makes system hardly maintainable.
5. For easy maintainable system, best architectural approach is to use n-tier by distributing system in to layers – vertically. Also, it is good to distribute system logic across the service – horizontally.
6. For SOA development recommendation is to keep services as much as possible isolated. This will reduce code coupling and increase code maintainability as well as reusability.

- REFERENCES

[1] S. K. Simardeep Kaur, „A Review on Metrics in SOA,“ 2016. [Tinkle]. Available: http://www.ijarcce.com/upload/2016/july-16/IJARCCE%2081.pdf. [Kreiptasi 20 02 2017].

[2] C. Hartwich, „Why It Is So Difficult to Build N-Tiered Enterprise,“ Freie Universitaet Berlin, Berlin, 2001.

[3] R. Land, „Measurements of Software Maintainability,“ Mälardalen University, Västerås, Sweden.

[4] K. D. Welker, „The Software Maintainability Index Revisited,“ Idaho National Engineering and Environmental Laboratory, 2001.

[5] T. Erl, Service-Oriented Concepts, Technology, and Design, United States: R.R. Donnelley in, 2005.

[6] L. Westfall, 12 Steps to Useful Software Metrics, The Westfall Team, 2005.

[7] S. H. Kan, Metrics and Models in software quality engineering, Canada: Person education inc., 2003.

[8] M. A. Hirzalla, Service Oriented Architecture Metrics Suite for Assessing and Predicting Business Agility Results of SOA Solutions, Chicago, Illinois: College of Computing and Digital Media, 2012.

lentelė 8.1 Pirmojo eksperimentinio scenarijaus duomenys

Nr.	Paslaugos	Naudojamos duomenų saugyklos	MI	CC	Coupling	LOC	HV
T1	0	1	93	3	2	4	0,11653
T2	0	2	92	4	3	5	0,077291
T3	0	3	89	5	4	6	0,11238
T4	0	4	88	6	5	7	0,092413
T5	0	5	86	7	6	8	0,112587
T6	0	6	86	8	7	9	0,074631
T7	0	7	85	9	8	10	0,071445
T8	0	8	84	10	9	11	0,070571
T9	0	9	84	11	10	12	0,051486
T10	0	10	84	12	11	13	0,03839
T13	0	13	81	13	13	16	0,041978
T15	0	15	80	14	15	18	0,034743
T18	0	18	79	15	18	20	0,027507
T20	0	20	79	22	21	23	0,021589

lentelė 8.2 Antrojo eksperimentinio scenarijaus duomenys

Nr.	Paslaugos	Naudojamos duomenų saugyklos	MI	CC	Coupling	LOC	HV
T1	0	1	93	3	2	4	0,11653
T2	1	2	92	7	1	9	0,0108451
T3	1	3	92	8	5	10	0,00747258
T4	2	4	91	12	7	15	0,00245956
T5	2	5	91	13	8	16	0,00192
T6	2	6	90	14	9	17	0,00211
T7	3	7	90	18	11	22	0,00106
T8	3	8	90	19	12	23	0,000662
T9	3	9	90	20	13	24	0,000554
T10	3	10	89	21	14	25	0,000649
T13	4	13	89	23	15	29	0,000587
T15	4	15	88	25	16	31	0,000579
T18	5	18	88	26	17	36	0,000505
T20	6	20	85	28	18	40	0,000489