KLAIPĖDOS UNIVERSITETAS

Jūros technologijų ir gamtos mokslų fakultetas

Informatikos ir statistikos katedra

Marius Balandis

PROGRAMINĖS ĮRANGOS REIKALAVIMŲ VALDYMO SISTEMOS, GRĮSTOS REIKALAVIMŲ SUSIETUMO VIZUALIZAVIMU, KŪRIMAS

DEVELOPMENT OF SOFTWARE REQUIREMENTS MANAGEMENT SYSTEM BASED ON VISUALIZATION OF REQUIREMENTS RELATIONS

Magistro baigiamasis darbas

Techninių informacinių sistemų inžinerijos studijų programa 621E15004

Klaipėda, 2017

MAGISTRO BAIGIAMŲJŲ DARBŲ LYDRAŠČIO FORMA

Pildo magistro baigiamojo darbo autorius

..... **Marius Balandis**.....
(magistro baigiamojo darbo autoriaus vardas, pavardė)

Baigiamojo darbo tema: Programinės įrangos reikalavimų valdymo sistemos, grįstos reikalavimų susietumo vizualizavimu, kūrimas
(magistro baigiamojo darbo pavadinimas lietuvių kalba)

Patvirtinu, kad bakalauro/magistro baigiamasis darbas parašytas savarankiškai, nepažeidžiant kitiems asmenims priklausančių autorių teisių, visas baigiamasis bakalauro/magistro darbas ar jo dalis nebuvo panaudotas Klaipėdos universitete ir kitose aukštosiose mokyklose.

_____
(magistro baigiamojo darbo autoriaus ir parašas)

Sutinku, kad bakalauro/magistro baigiamasis darbas būtų naudojamas neatlygintinai 5 m. Klaipėdos universiteto studijų procese.

_____
(magistro baigiamojo darbo autoriaus ir parašas)

Pildo magistro baigiamojo darbo vadovas

Magistro baigiamąjį darbą ginti.....
..... (įrašyti - **leidžiu** arba **neleidžiu**)
....2017..... ..... prof. dr. V. Denisovas
(data) (magistro baigiamojo darbo vadovo vardas, pavardė ir parašas)

Pildo katedros, kuruojančios studijų programa, administratorius (sekretorius)

Baigimasis darbas įregistruotas katedroje 2017 .....
(data)

Laima Brazdeikienė.....
(katedros sekretorės vardas, pavardė ir parašas)

Pildo katedros, kuruojančios studiju programą, vedėjas

Magistro baigiamąjį darbą ginti.....
(įrašyti - **leidžiu** arba **neleidžiu**)

2017..... prof. dr. Arūnas Andziulis.....
(data) (katedros vedėjo vardas, pavardė ir parašas)

Recenzentu(-ais) skiriu ....
....
.....
(įrašyti recenzento(ų) vardą, pavardę)

.....2017..... prof. dr. Arūnas Andziulis.....
(data) (katedros vedėjo vardas, pavardė ir parašas)

ANOTACIJA

Programinės įrangos reikalavimų valdymo sistema yra kuriama siekiant optimizuoti įmonėje vykdomų projektų reikalavimų gausos valdymą. Analizuojami esami reikalavimų valdymo esminiai principai, struktūrizuotų hierarchinių duomenų vizualizavimo metodai ir tų metodų panaudojimo galimybės. Pateikiami vizualizavimo metodai skirti grafiškai pateikti struktūrizuotus hierarchinius duomenis. Naudojant Visual Basic .Net programavimo kalbą sukurta sistema, kuri geba vizualizuoti reikalavimus ir jų tarpusavio sąryšius taip suteikdama galimybę efektyviau valdyti reikalavimus. Reikalavimų valdymo sistema yra palaikoma UAB Omega Technology naudojamos „Appframe R4“ sistemos.

PAGRINDINIAI ŽODŽIAI**:** reikalavimų inžinerija, reikalavimų valdymo sistema, programavimo karkasas, programavimo platforma Microsoft .NET, hierarchinių duomenų vizualizavimas, duomenų gavyba, reikalavimų susietumas.

ABSTRACT

Software requirements management system has been created to achieve an optimal management of multiple requirements for existing projects in a software company. Current fundamental methods of requirements management are being analyzed as well as structured hierarchical data visualization methods along with the possibilities of their usage. Visualization algorithms are provided for graphical representation of structured hierarchical requirements data. Using Visual Basic .Net programming language a system was created, which maintains and visualizes requirements and their reciprocal relations, providing an effective way to manage requirements. Requirements' management system is supported by “Appframe R4” which is used by UAB Omega Technology.

KEYWORS: requirements engineering, requirements management system, software framework, Microsoft .NET platform, hierarchical data visualization, data mining, requirements relatedness

PAGRINDINĖS SĄVOKOS IR TERMINAI

Artefaktas – fizinė informacijos dalis, naudojama ar sukuriama programinės įrangos kūrimo procese. Artefaktai yra modeliai, išeities ir vykdomieji failai, programos kodas, komponentų realizacijos (t. y. visi sukurti produktai).

Hierarchija – tai nuosekli lygmenų eilė nuo žemesnių prie aukštesnių; pavaldumo ir priklausymo tvarka.

Netmap vizualizacija – struktūrizuotų hierarchinių duomenų grafinė reprezentacija, siejanti duomenis tam tikru atributu - įprastai tai semantinį sąryšį tarp dvejų mazgų turintis atributas.

Pims – *Project Information Management Systems* – tarptautinės kompanijos Omega AS projektų duomenų valdymo sistemų paketas, palaikomas „Appframe R4“ programavimo karkaso ir skirtas įvairių sistemų, skirtų projektų valdymui, standartizavimui.

Programinė įranga (angl. *software*) – informacijos apdorojimo sistemos programų, procedūrų, taisyklių visuma arba tos visumos dalis kartu su atitinkama dokumentacija

Projektas – tai laikina veikla, orientuota sukurti unikalų produktą ar paslaugą

Reikalavimo atributai – standartizuotas duomenų rinkinys, skirtas apibūdinti reikalavimą

Reikalavimų nepastovumas (angl. *volatility*) – tai reikalavimų kitimas laike (evoliucija), naudotojų poreikių keitimasis, konkurencinės verslo aplinkos įtaka.

Rolė – tai rinkinys sistemos įvairių taisyklių ir apribojimų, siekiant naudotojui priskirti vaidmenį toje sistemoje.

Sistema – tai elementų aibė, kuri charakterizuojama tokiais požymiais: 1) jungumu, t. y. elementai glaudžiai susiję vienas su kitu ir sąlygojami vienas kito bei supančios aplinkos fiksuotų ryšių; 2)funkcionalumu, t. y. sistema pasižymi funkcija, kuri skiriasi nuo ją sudarančių elementų funkcijų.

SRM – „Sistemų Reikalavimų valdymas“ – (angl. *systems requirements management*) – kuriamo programinės įrangos prototipo akronimas „Appframe R4“ sistemoje.

Suinteresuotosios pusės/šalys (angl. *stakeholder*) – asmuo, asmenų grupė arba organizacija, kuri gali turėti įtakos, kuriai gali turėti įtakos arba kuri gali save įsivaizduoti kaip šalį, kuriai gali turėti įtakos projekto sprendimas, veikla ar rezultatas. Suinteresuotosios šalys apima visus projekto darbo grupės narius, taip pat visus vidinius ir išorinius organizacijos suinteresuotuosius subjektus.

Sunburst vizualizacija – skritulinė diagrama (dar vadinama „žiedine“ arba „daugiapakope skrituline“ diagrama), alternatyva plotą užpildančioms vizualizacijoms kaip Treemap, kuri vietoj stačiakampio formos artefaktų uždėstymo, naudoja radialinį. Šios diagramos artefaktai hierarchijos principu išdėstomi ratu, kur diagramos centre - pagrindinis mazgas, o gilesnės viršūnės yra toliau nuo centro. Artefaktų spalva dažnai reprezentuoja tam tikrą vizualizuojamų duomenų atributą.

Vizualizacija – tai duomenų regimasis teikimas, įvairialypės (dažnai hierarchinės) informacijos atvaizdavimas grafiškai (pavyzdžiui, monitoriaus ekrane).

PAVEIKSLŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

TURINYS

Priedai:

1. Kuriamam programinės įrangos prototipui keliami reikalavimai
2. Kuriamo prototipo klasių diagrama
3. Sunburst algoritmo diagrama
4. Netmap algoritmo diagrama
5. Kompaktinis diskas

ĮVADAS

Darbo aktualumas, naujumas, problemos ištyrimo laipsnis ir praktinė nauda.

Skaitmeninės technologijos mums leidžia įvairiais būdais surinkti ir susisteminti įvairius duomenis itin dideliais mastais, bet dauguma mūsų vis dar bando šį informacijos kiekį suvokti remdamiesi įvairiomis reprezentacijomis, kurios daugiau orientuotos į tekstą nei atvaizdavimą ir pateikia ne daugiau informacijos nei popieriaus lapas. Todėl iškyla realus „informacijos perkrovos“ pavojus – atsiranda grėsmė suklysti analizuojant, prioretizuojant ir atsirenkant informaciją. Mūsų sugebėjimas pateikti informaciją naudingu ir intelektualiu būdu krinta palyginus su mūsų gebėjimu kurti ir naudoti neapdorotus duomenis.

Informacijos vizualizavimas panaudoja tuos pažinimo įgūdžius, kurių tekstinė ar skaitinė informacija negali panaudoti. Vizualizavimas yra universaliai daug lengviau suprantamas nei bet kuri kalba, ir leidžia tai, ko tekstas atitikti negali. Tinkama vizualizacija gali rodyti informaciją kontekste ar parodyti kokybišką sudėtingų sistemų būseną, kuomet vos „žvilgtelėjus“ galima greitai peržiūrėti sistemos būsenas ir pokyčius.

Nors reikalavimų specifikacija yra geras būdas dokumentuoti išsamią informaciją, susijusią su naujo ar esamo projekto funkcionalumu, tačiau tai daug resursų eikvojanti dokumentacija, kai siekiama iš jos gauti norimą informaciją kiekvienos jos peržiūros metu.

Perkėlus reikalavimų specifikaciją į grafinę terpę – vizualizavus joje aprašytus reikalavimus, atsiranda galimybė efektyviau gauti norimą informaciją, suvaldyti reikalavimų nepastovumą ir stebėti projekto eigą. Tačiau tokiam procesui yra reikalinga tam pritaikyta programinės įrangos sistema.

Darbo problema. Reikalavimų nepastovumas (dėl vidinių ir išorinių priežasčių) bei nuolat augantis (kintantis) jų skaičius besikeičiančioje verslo aplinkoje, o tradicinių sistemų reikalavimų vaizdavimo būdai (matricos ir medžiai) visai tai tinkamai atvaizduoti.

Darbo objektas. Reikalavimų valdymo sistema, gebanti vizualizuoti didelį programinės įrangos reikalavimų rinkinį, išryškinant reikalavimų charakteristikas it susietumą tarpusavyje.

Darbo tikslas. Sukurti programinės įrangos reikalavimų valdymo sistemos prototipą, sugebantį vizualiai atvaizduoti programinės įrangos reikalavimų gausą ir sąryšius tarp jų.

Darbo uždaviniai:

1. Išanalizuoti esminius reikalavimų valdymo principus, programinių reikalavimų valdymo sistemų struktūrą ir funkcijas.
2. Atlikti praktikoje naudojamų struktūrizuotų duomenų vizualizavimo metodų analizę ir atrinkti metodus, tinkamus reikalavimų vizualizavimui.
3. Pateikti algoritmus Sunburst ir Netmap vizualizavimo metodams, skirtus vizualizuoti programinės įrangos reikalavimus ir jų sąryšius.
4. Programiškai realizuoti pateiktus algoritmus Microsoft .NET platformoje, remiantis programinės įrangos įmonėje sukurtu programavimo karkasu.
5. Išbandyti ir verifikuoti sukurtos reikalavimų valdymo sistemos prototipą, pagrįsta minėtais algoritmais, naudojant realius duomenis ar procesų aprašus.

Tyrimo metodai ir priemonės. Mokslinės literatūros reikalavimų inžinerijos ir valdymo srities analizė; lyginamoji duomenų vizualizavimo metodų analizė; vizualizavimo metodų algoritmavimas, programų sistemų prototipavimas, programavimo karkasų taikymas. Sistemos prototipas yra parašytas *Visual Basic .Net* programavimo kalba, *MS Visual Studio 2015 Community* aplinkoje ir yra palaikomas UAB Omega Technology įmonės naudojamo programavimo karkaso *Appframe R4*, kuris naudoja *MS SQL SERVER 2012* duomenų bazę. Darbe pateiktos diagramos yra nubraižytos naudojant *Visual Paradigm Community Edition* įrankiu.

I. REIKALAVIMŲ VALDYMO PRINCIPŲ IR METODŲ ANALIZĖ

1.1. Programinės įrangos reikalavimai

Programinės įrangos reikalavimas – tai programinės įrangos inžinerijos artefaktas [18, 34], kuriuo aprašomas konkretus suinteresuotųjų pusių poreikis projektuojamai programinei įrangai. Reikalavimas iš esmės apibrėžiamas kaip [22, 44] sąlyga arba sugebėjimas, kurį programinė įranga ar sistema turi atitikti. IEEE standartas programinės įrangos reikalavimą apibrėžia kaip [36]:

- sistemos sugebėjimas, nusakytas vartotojo ar užsakovo, kurio pagalba būtų išsprendžiama problema ar pasiekiamas tam tikras tikslas;
- sistemos sugebėjimas, kuris privalo atitikti ir egzistuoti sistemoje, kuri yra aprašyta standartais, specifikacijomis, reglamentavimais ir kitais formaliais dokumentais;
- suinteresuotosios šalies nustatytas apribojimas;
- dokumentuotas minėtų sąlygų, sugebėjimų ar apribojimų atitikmuo.

Taigi, galima teigti, jog programinės įrangos reikalavimas egzistuoja ir yra paisomas tik tuo atveju jei jis yra dokumentuotas t. y. iš įvairių kitų pateikties formų (pavyzdžiui, verbalinės) reikalavimas turi būti suformuluotas ir užrašytas, taip sudarant reikalavimų specifikaciją.

1.1.1. Programinės įrangos reikalavimų specifikacija

Programinės įrangos reikalavimų specifikacija yra dokumentas, kuris yra naudojamas [7, 9, 18] kaip komunikacijos priemonė tarp užsakovo, naudotojo ir tiekėjo. Reikalavimų inžinerijos etapas pabaigiamas, kai programinės įrangos reikalavimų specifikacija yra baigta ir priimta visų suinteresuotųjų šalių. Šį dokumentą turi teisę redaguoti visos suinteresuotųjų šalių pusės, nes iš pradžių nėra tiksliai žinoma, kas yra reikalinga (tiekėjui), ir tai, kas yra įmanoma (užsakovui), tad sekančiuose projektų vykdymo etapuose reikalavimai gali būti pakeičiami, tačiau tie pakeitimai turi būti kontroliuojami.

Programinės įrangos reikalavimų specifikacija turi ne vieną panaudos tikslą [45]. Tai gali svyruoti nuo užsakomos programinės įrangos reikalavimų specifikavimo tiekėjų konkursui, iki programinės įrangos kūrėjų, kurie siekia aprašyti programinės įrangos reikalavimus, pateikiant savo programinės įrangos reikalavimų dokumentą. Pirmuoju atveju, reikalavimai programinei įrangai yra pakankamai bendri, tam, kad skirtingų tiekėjų skaičius pasiūlytų savo sprendimus, tačiau pakankamai konkretūs, jog būtų apibrėžti visi apribojimai. Antru - programinės įrangos reikalavimų specifikacija yra naudojama fiksuoti reikalavimus ir, jei reikia, pažymėti visus nesuderinamus ir prieštaringus reikalavimus bei apibrėžti sistemą ir testavimo veiklą.

Programinės įrangos reikalavimų specifikacija yra vienas iš pagrindinių oficialių dokumentų, naudojamų bendradarbiavimui tarp suinteresuotųjų šalių. Turint tai omenyje, minimali informacija kuri privalo būti dokumente - programinės įrangos reikalavimų sąrašas, patvirtintas abejų šalių: reikalavimai privalo turėti numatytas charakteristikas ir būti suskirstyti į rūšis.

1.1.2. Programinės įrangos reikalavimų charakteristikos

Reikalavimų charakteristikas galima skirstyti į dvi kategorijas [2] – vartotojo apibrėžtas (prioritetas, rizika, būsena ir kt.) ir sistemos apibrėžtas (indentifikavimo kodas, autorius, pateikimo data ir kt.) savybes. Visų jų paskirtis - ne tik charakterizuoti kiekvieną reikalavimą, bet ir padėti sekti ir stebėti reikalavimų evoliuciją projekto vykdymo metu. Nors galimų charakteristikų, kurias būtų galima registruoti yra gausybė [2, 5, 43, 52], reikėtų suvogti, jog ne visos jos duos naudos siekiant tinkamai suvaldyti programinės įrangos reikalavimus. Iš esmės, registruojamos turėtų būti tik tos reikalavimų savybės, kurios leistų sekti viso vykdomo projekto eigą ir progresą.

Raktinės reikalavimų charakteristikos tūrėtų būti fiksuojamos ir sekamos nuo pat pradžių – tai leistų efektyviau valdyti reikalavimus, kas lemptų [34, 38] mažesnę kaštų sąnaudą vėlesniuose projekto etapuose. Atliktame tyrime [44], siekiančiame nustatyti ryšį tarp reikalavimų charakteristikų ir sistemos defektų, kilusių iš prastai dokumentuotų reikalavimų teigiama, jog šios šešios reikalavimų savybės būtų raktinės charakteristikos: įgyvendinimo trukmė (angl. *time estimates*), svarbumas (angl. *priority*), nuosavybė (angl. *ownership*), netiesiogiai susijusios suinteresuotosios šalys (šalys, kurios nebuvo įtrauktos į pradinius projekto etapus), susijusių reikalavimų skaičius ir reikalavimo pateikimo data. Čia [2, 18, 51] pastebimi tų pačių charakteristikų pasikartojimai – dažniausiai yra akcentuojama reikalavimo rūšis (tipas), reikalavimo svarbumas, reikalavimo šaltinis (autorius, dokumentacija ir t.t. – kitais žodžiais – reikalavimo pagrindimas ir „realumas“) ir su reikalavimu susiję kiti reikalavimai ar jų skaičius.

1.2. Reikalavimų valdymas

Reikalavimų valdymas - tai visų galimų programinės įrangos (ar bet kokio kito projekto) reikalavimų surinkimas, apibrėžimas, modeliavimas, analizavimas ir dokumentavimas iki taško, kai galutinis produktas nebeturi būti toliau tobulinamas. Tačiau realybėje reikalavimai dažniausiai [24, 27, 38] yra neišbaigti, nepastovūs, o jų kiekis nuolatos kinta. To priežastis neretai būna verslo poreikių, technologijų ir rinkos kaita. Be to, [34] kiekviena suinteresuotoji šalis skirtingai suvokia skirtingus aspektus, todėl skiriasi ir programinei įrangai keliami reikalavimai, kurie gali net vienas kitam prieštarauti.

Kadangi reikalavimų skaičius projekto įgyvendinimo metu [36, 38] įprastai tik didėja, o patys reikalavimai kinta, tai šiems procesams suvaldyti kuriamos ir naudojamos reikalavimų valdymo sistemos. Tipiška tokia sistema saugo visų reikalavimų sąrašą, jų charakteristikas ir tarpusavio sąryšius. Esminiai jų panaudos tikslai – palengvinti reikalavimų atsekamumą, sekti reikalavimų evoliuciją bei užtikrinti, jog projekto vystymas „nestovi vietoje“ - tie patys reikalavimai nėra pastoviai pateikiami ir atmetami, arba projekto vykdymas „neina ratais“ t. y. nėra bandoma įgyvendinti tarpusavyje nesuderinamus ar net vienas kitam prieštaraujančius reikalavimus.

1.2.1. Reikalavimų atsekamumas

Reikalavimų atsekamumas - viena iš pagrindinių reikalavimų valdymo veiklų [4, 11, 53, 54]. Atsekamumo apibrėžimas, pirma, apima reikalavimų gyvavimą – reikalavimai įprastai evoliucionuoja: keičiami jų aprašai, atributai, jie yra išskaidomi ar sutraukiami. Todėl visi šie pokyčiai turi būti sekami. Antra, atsekamumas apima reikalavimų susietumo ryšius. Teigiama [10], kad reikalavimas turėtų būti atsekamas iki reikalavimų ar suinteresuotųjų šalių, iš kurių šis reikalavimas kilo (angl. *backward trace*), ir taip pat atsekami reikalavimai, kurie tenkiną minėtąjį reikalavimą (angl. *forward trace*).

1.2.2. Reikalavimų evoliucija

Reikalavimų evoliucija – tai reikalavimų valdymo etapas, kai yra vykdomi reikalavimų pakitimai jau po pradinio reikalavimų valdymo etapo t. y. programinės įrangos reikalavimai evoliucionuoja produkto kūrimo ir net jo palaikymo metu [43]. Galimos pakitimų priežastys aprašytos 1 lentelėje.

1 lentelė. Galimos reikalavimų evoliucijos priežastys [43].

Priežastys	**Kontekstas**
Rinka	Pasikeitę daugelio klientų poreikiai, technologijos ar valdžios reglamentai.
Užsakovas	Pasikeitus užsakovo strateginei krypčiai, organizaciniams aspektams ar politiniai aplinkai.
Projekto vizija	Pasikeitus spendžiamai problemai, produkto krypčiai ir prioritetams, suinteresuotųjų šalių dalyvavimui projekto vystyme ar procesų kaitai.
Reikalavimų specifikacija	Pasikeitus reikalavimų specifikacijai, siekiant panaikinti esamus neaiškumus, prieštaravimus ar pagerinti bendrą supratimą
Produktas	Pasikeitę įgyvendinti techniniai reikalavimai ar projektavimo struktūra

Vykstant reikalavimų kaitai, atsiranda poreikis tuos pakeitimus fiksuoti. [47] Reikalavimų pokyčius, kuriuos reiktų fiksuoti, galima sugrupuoti į dvi grupes:

- pasikeitus reikalavimui: aprašymui ar naudojamai terminologijai;
- pasikeitus reikalavimo atributams: būsenai, šaltiniui, prioritetui ir kt.

Elementariausias būdas fiksuoti reikalavimų evoliuciją – dokumentuoti kiekvieną pokytį paprasčiausiu jų sąrašu. Kitas būdas [30, 43, 47] – reikalavimų versijavimas. Šis metodas leidžia palyginti reikalavimo versijas ar gražinti reikalavimą į senesnę versiją. Atsižvelgus į konkretaus reikalavimo versijų kiekį, galima išmatuoti reikalavimo nepastovumą.

IEEE 982 standartas [30] siūlo galimybę programinės įrangos produkto užbaigtumą išreikšti skaičiumi - programinės įrangos užbaigtumo indeksu (angl. *Software Maturity Index*) - *SMI*. Pakitimų skaičius tarp versijų nurodo produkto stabilumą. Tačiau SMI galima pritaikyti ir reikalavimams. Pasinaudojant reikalavimų versijavimu, galima apskaičiuoti vidutinį reikalavimų pakitimų skaičių (angl. *Avarage Number of Requirements Changes*) - AR_C.

$$AR_c = \frac{CR_C}{n} \quad (1),$$

kur CR_C *- kaupiamasis reikalavimų pakitimų skaičius, rodantis kiek pakitimų įvyko nuo pradinės reikalavimų specifikacijos dokumento versijos, o n – produkto versijų skaičius.*

Pasinaudojant [30] *SMI,* AR_C ir CR_C dydžiais, išvedami nauji dydžiai, skirti pamatuoti reikalavimų kitimą: reikalavimų stabilumo indeksą - *RSI* (angl. *Requirements Stability Index*) ir istorinio reikalavimų užbaigimo indeksą – *HRMI* (angl. *Historical Requirements Maturity Index*). *RSI* yra pranašesnis už *SMI*, nes yra paskaičiuojamas naudojant CR_C, todėl nėra tiesiogiai priklausomas nuo visų reikalavimų skaičiaus R_T. *RSI* reikšmės intervalas yra $0 \leq RSI \leq 1$. Tačiau tam tikrais atvejais, kai CR_C yra didesnis už R_T, *RSI* reikšmė tampa neigiama. [30] Tai reiškia, kad reikalavimai yra labai nepastovūs.

$$RSI = \frac{R_T - CR_C}{R_T} \quad (2).$$

Vietoj CR_C įstačius 1 formulę, gaunamas *HRMI*. Šis indeksas [30], palyginus su *SMI,* nėra toks jautrus dažnam produkto versijų leidimui.

$$HRMI = \frac{R_T - AR_C}{R_T} \quad (3)$$

arba

$$HRMI = \frac{R_T - \frac{CR_C}{n}}{R_T} \quad (4).$$

1.2.3. Reikalavimų svarbumas

Siekiant minimizuoti reikalavimų kaitą, būtina kuo tiksliau nustatyti visų reikalavimų svarbumą [24, 38] dar pradiniuose projekto vystymo etapuose. Tačiau tai padaryti gali būti sudėtinga, nes atskiros suinteresuotosios šalys kiekvieno reikalavimą svarbą gali vertinti skirtingai: [25, 33] užsakovai gali prioretizuoti tuos reikalavimus, kurie naudingi iš verslo pusės, naudotojai – patogumo ir funkcionalumo, o programinės įrangos kūrėjai – sistemos architektūros atžvilgiu ir pan. Todėl yra svarbu nustatyti, kurie reikalavimai turi būti prioretizuojami kiekvienu atveju. Nustatyta, [33] jog tinkamas reikalavimų svarbumo parinkimas padeda:

- suinteresuotosioms šalims nuspręsti, kurie programinei įrangai keliami reikalavimai yra esminiai;
- kiekvienai sekančiai iteracijai sudaryti optimalų reikalavimų sąrašą, kurie turėtų būti įgyvendinti laiku, pagal numatytą planą;
- atrinkti reikalavimų poaibį, kurį įgyvendinus būtų paspartintas sistemos prototipo atsiradimas, taip leidžiant anksčiau pradėti naudotis sistema.
- kt.

1.2.4. Reikalavimų rūšys

Yra naudojama daugybę būdų reikalavimų klasifikavimui ir tvarkant reikalavimus vieno būdo dažnai neužtenka. Tačiau vienas populiaresnių būdų yra reikalavimų klasifikavimas pagal jų rūšis:

- **Funkciniai reikalavimai** (angl. *functional requirements*) - rinkinys bendrų sistemos reikalavimų [40, 45], kurie yra naudojami apsvarstyti kompromisus, sistemos elgseną ir žmogiškuosius aspektus. Funkciniai reikalavimai taip pat aprašo, kaip sistema veiks pagal įprastą operaciją, apsvarstytas pasekmes ir produkto (sistemos) reakciją dėl programinės įrangos gedimo ar netinkamų įvesčių (įėjimų) į sistemą.
- **Našumo reikalavimai** (angl. *performance requirements*) [26, 40, 45] - tai reikalavimai, kurių reikšmė gali būti kiekybinė. Šie reikalavimai apibūdina išmatuojamus dydžius, pavyzdžiui, dažnumą, greitį ir t.t. Nurodytos reikšmės taip pat turi būti išreikštos atitinkamais vienetais, pavyzdžiui metrų, kvadratinių centimetrų, kilometrais per valandą, ir kt.
- **Sąsajos reikalavimai** (angl. *interface requirements*) - tai reikalavimai [19, 26], kurie aprašo, kokią sąsają sistema užtikrina su jos aplinka, vartotojais ir kitomis sistemomis. Šie reikalavimai gali apibūdinti sąsajos išmokstamumą, veiksmingumą, efektyvumą,

įsimenamumą, našumą, saugumą ir kt. Taip sąsajos reikalavimai gali apibrėžti šiuos elementus:

  - grafinių elementų atvaizdavimą (angl. *visual overview of the screen*),
  - atvaizdavimo taisykles (angl. *display rules*),
  - pranešimus (angl. *messaging*),
  - nuorodas (angl. *links*).

- **Veiklos Reikalavimai** (angl. *operational requirements*) [26, 29, 45] apibrėžia tokius dalykus:
  - kaip veiks sistema,
  - kaip sistema bendraus su jos valdytojais,
  - kiek ir kaip kvalifikuotų sistemos valdytojų bus reikalinga,
  - kokias pareigas turės kiekvienas sistemos valdytojas,
  - kokia pagalbą teiks sistema.
- **Išteklių Reikalavimai** (angl. *resource requirements*) [40, 45] - tai reikalavimai apibūdinantys išteklius, kurie bus reikalingi norint pasiekti projekto tikslus. Šie reikalavimai gali apimti šiuos išteklius:
  - personalo,
  - įrangos ir medžiagų,
  - prieigos prie technologijų,
  - prieigos prie distribucijos kanalų,
  - išorinių paslaugų prieinamumas.
- **Verifikavimo reikalavimai** (angl. *verification requirements*) [45] apibūdina kaip funkciniai ir našumo reikalavimai turi būti patikrinti ir išmatuoti. Matavimai gali apimti simuliacijas, emuliacijas ir testavimus su tikra arba sumodeliuota informacija. Taip pat reikalavimai turėtų nurodyti ar matavimo testai turi būti atliekami stebint klientui ar jo (įmonės) atstovui.
- **Dokumentacijos reikalavimai** (angl. *documentation requirements*) – reikalavimai [45], kurie nurodo kokie dokumentai turi būti pateikiami klientui viso projektu kūrimo metu arba to projekto pabaigoje. Klientui pateikti dokumentai gali apimti projekto specifinę dokumentaciją, taip pat vartotojo vadovus ir bet kokius kitus atitinkamus dokumentus.
- **Kokybės reikalavimai** (angl. *quality requirements*) [26] nurodo visus tarptautinius ir vietinius standartus, kuriais turi būti laikomasi. Šie reikalavimai skiria dėmesį kokybės užtikrinimo planui – pagrindinei kokybės užtikrinimo dokumento daliai. Tipiniai kokybės reikalavimai apima šias sąvokas [45]:
  - kokybės veiksniai (angl. *quality factors*),

    - korektiškumas (angl. *correctness*),
    - patikimumas (angl. *reliability*),
    - efektyvumas (angl. *efficiency*),
    - vientisumas (angl. *integrity*),
    - panaudojamumas (angl. *usability*),
    - prižiūrimumas (angl. *maintainability*),
    - testuojamumas (angl. *testability*),
    - lankstumas (angl. *flexibility*),
    - portatyvumas (angl. *portability*),
    - pakartotinis naudojimas (angl. *reusability*),
    - sąveikumas (angl. *interoperability*),
    - papildomi veiksniai (angl. *additional factors*).
- **Saugumo reikalavimai** (angl. *security requirements*) – tai nefunkcinių reikalavimų [8] aibė, kurios elementai (reikalavimai) yra susiję su sistemos konfidencialumu, vientisumu, ir prieinamumu. Žinoma, šie reikalavimai turi būti įvykdyti siekiant užtikrinti sistemos saugumą. Tačiau jie gali būti parengti skirtingos abstrakcijos lygmenyse [32]. Aukščiausiame abstrakcijos lygmenyje šie reikalavimai atspindi tik saugumo tikslus, o konkrečiai ir aiškiai juos aprašius – puiki pagalba produkto saugumo testavimams projekto vykdymo metu.
- **Patikimumo reikalavimai** (angl. *reliability requirements*) [39] paprastai yra techninių specifikacijų dokumento dalis. Šie reikalavimai gali būti keliami kuriamiems produktams pačių kūrėjų (ne užsakovų) siekiant užtikrinti patikimumą užsakovams. Patikimumas gali būti matuojamas įvairiais būdais, įskaitant klaidų skaičių per x kodo eilučių, vidutinį laiką tarp gedimų ar laiką, išreikšta procentais, kurį sistema turėtų veikti prieš jai sulūžtant.
- **Priežiūros reikalavimai** (angl. *maintainability requirements*) [40] turėtų atsižvelgti į bet kokius numatomus pokyčius programinės įrangos sistemoje, kompiuterinės aparatūros konfigūracijoje ir ypatingai į produkto eksploataciją tose vietose, kur įrangos palaikymas nėra galimas.

1.2.5. Reikalavimų sąryšiai

Reikalavimų specifikacijos dokumente reikalavimai dažniausiai yra surašyti skyriais ar poskyriais, o tai apriboja reikalavimų atsekamumo ryšių atvaizdavimą. Šie ryšiai yra itin svarbūs (2 lentelė), siekiant atlikti įvairias užduotis produkto vystymo metu. Be to, [10, 16, 22, 24] teisingai sudaryti ryšiai tarp reikalavimų leidžia efektyviau valdyti reikalavimus, taip potencialiai išvengiant problemų, galinčių atsirasti vėlesniuose projekto vystymo etapuose.

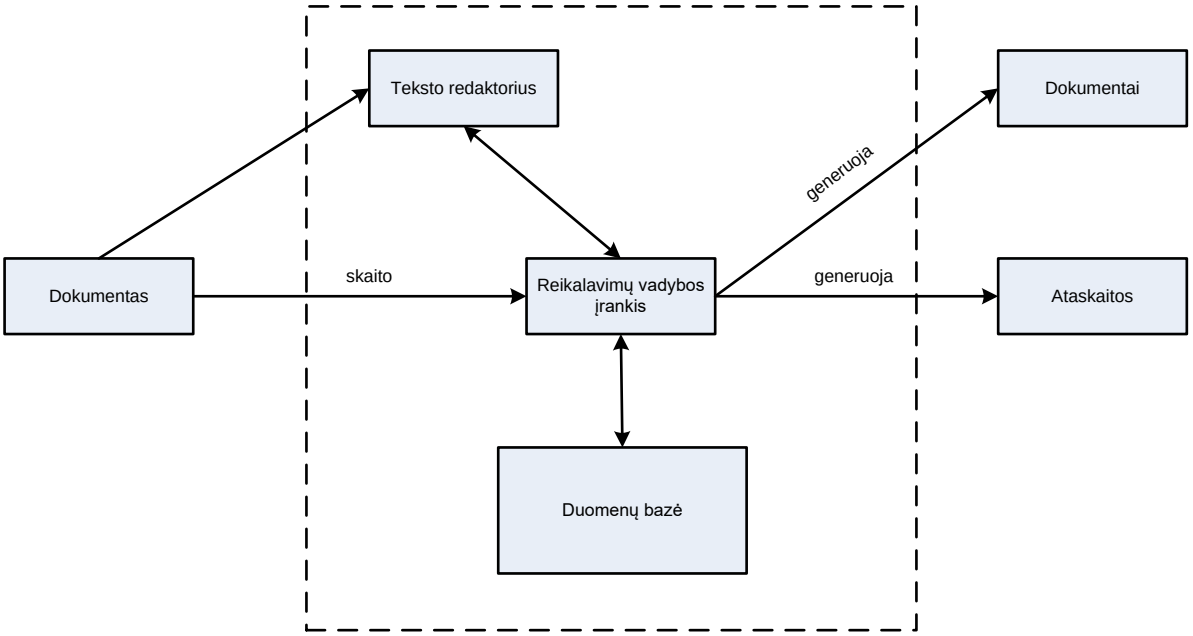
2 lentelė. Galimi reikalavimų sąryšių tipai [16].

Tipas	**Reikšmė**
Apriboja (angl. *constrains*)	Sąryšis skirtas registruoti reikalavimų tarpusavio sąveiką kai vienas reikalavimas apriboja kitą. Dažniausiai tai su saugumu arba našumu susiję reikalavimai.
Dekomponuoja („tėvo“ - „vaiko“ sąryšis)	Šio tipo sąryšiais sudaroma reikalavimų hierarchija.
Įsiterpia (angl. *embeds*)	Sąryšio tipu nusakomas ryšys tarp reikalavimų, kai įvykdant vieną, įvykdomas ir kitas (funkcionalumo atžvilgiu).
Nurodo (angl . *reference*)	Skirtas nurodyti ryšį tarp reikalavimų, kurie tarpusavyje susiję semantiškai.
Tenkina (angl. *satisfies*)	Apibūdina sąryšį tarp reikalavimų, kai vienas papildo kitą.
Dubliuoja (angl. *duplicates*)	Ryšys nustatomas tarp reikalavimų kai abu aprašo tą pačią sistemos funkciją, gebėjimą ar apribojimą.
Anuliuoja (angl. *supersedes*)	Skirtas nusakyti reikalavimų sąryšį, kai vienas iš jų tampa nebeaktualus įgyvendinant kitą.
Prieštarauja (angl. *contradicts*)	Ryšys tarp reikalavimų, kai abu vienu metu negali būti įgyvendinti fiziškai, pavyzdžiui vienas leidžia, o kitas draudžia tą patį funkcionalumą.

Kiekvienas programinės įrangos reikalavimas yra heterogeniškas [5] - tai reiškia, kad reikalavimas gali turėti ne vieną sąryšį su kitais reikalavimais. Dėl šios priežasties reikalavimų specifikacija negali pilnai pavaizduoti visų esamų sąryšių, ypač kai jie gali kisti programinės įrangos kūrimo ir, vėliau, gyvavimo ir palaikymo metu. Todėl tinkamam reikalavimui valdymui reikėtų naudoti įrankius, gebančius atvaizduoti reikalavimų sąryšius.

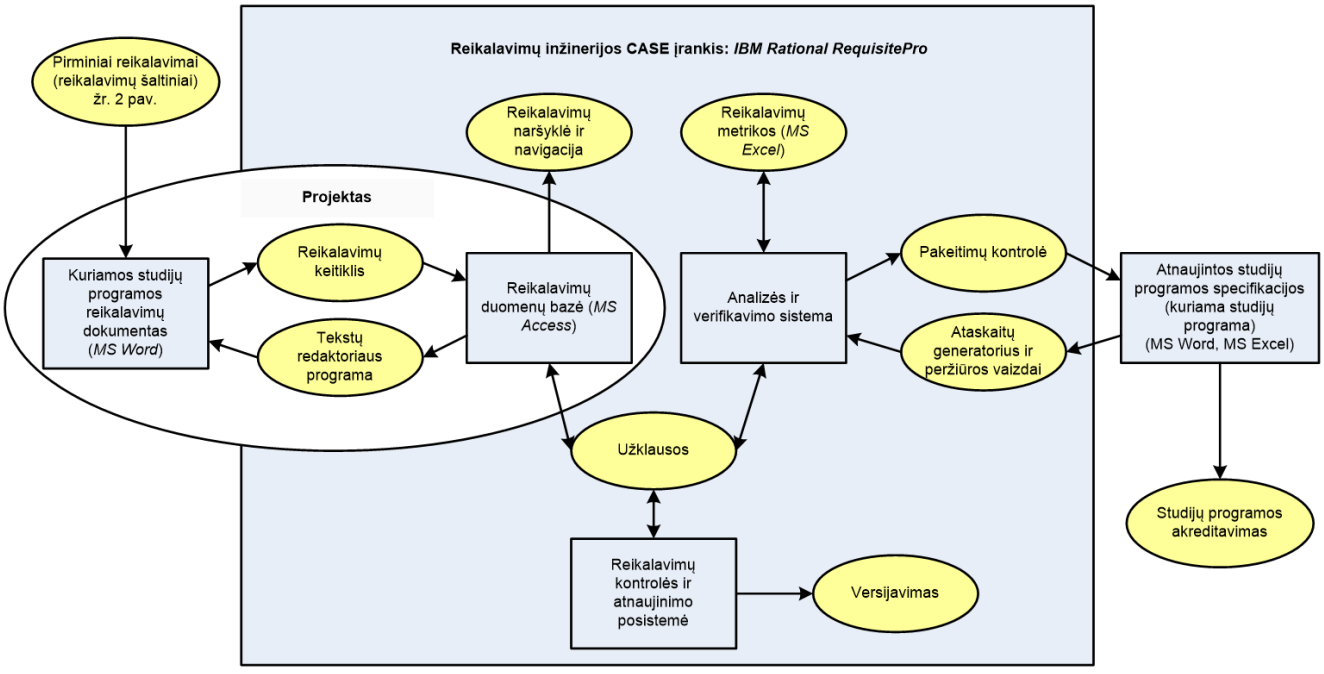
1.3. Reikalavimų valdymo sistemos

Kiekviena iš dabartinių reikalavimų valdymo sistemų, nuo tokių, kurias privaloma įdiegti į kompiuterį iki tokių, kurioms pakanka interneto naršyklės lango, turi savo savitą grafinę naudotojo sąsają, tačiau didžiąją dalį jų sieja vienas bendras bruožas – reikalavimai yra nuskaitomi iš tam tikro formato failų arba su sistema susietų teksto redagavimo įrankių, tada suvienodinami, jog atitiktų valdymo įrankio naudojamą formatą ir įtraukiami į duomenų bazę. Toliau reikalavimai jau gali būti keičiami, generuojamos įvairios jų ataskaitos ar dokumentai. Tokia [41] principinė valdymo įrankių veikimo struktūra pateikta 1 paveiksle.



1 pav. Reikalavimų valdymo sistemų struktūra. Pagal [41]

Reikalavimų valdymo sistemos, kurios atitinka tokią struktūrą (1 pav.) yra labiau orientuotos į patį reikalavimų importavimą iš dokumentų, jų pakeitimą ir eksportavimą į kitus dokumentus procesą, nei į patį reikalavimų valdymą ar jų tarpusavio sąryšių tinkamą atvaizdavimą. Pavyzdžiui, papildyta reikalavimų valdymo sistemos [15], pritaikytos studijų programų akreditavimui (2 pav.), struktūra iš esmės taip pat skiria dėmesį pačiam reikalavimų išgavimui, pakeitimui ir eksportavimui nei tų reikalavimų tinkamam ir patogiam valdymui bei vizualizavimui.

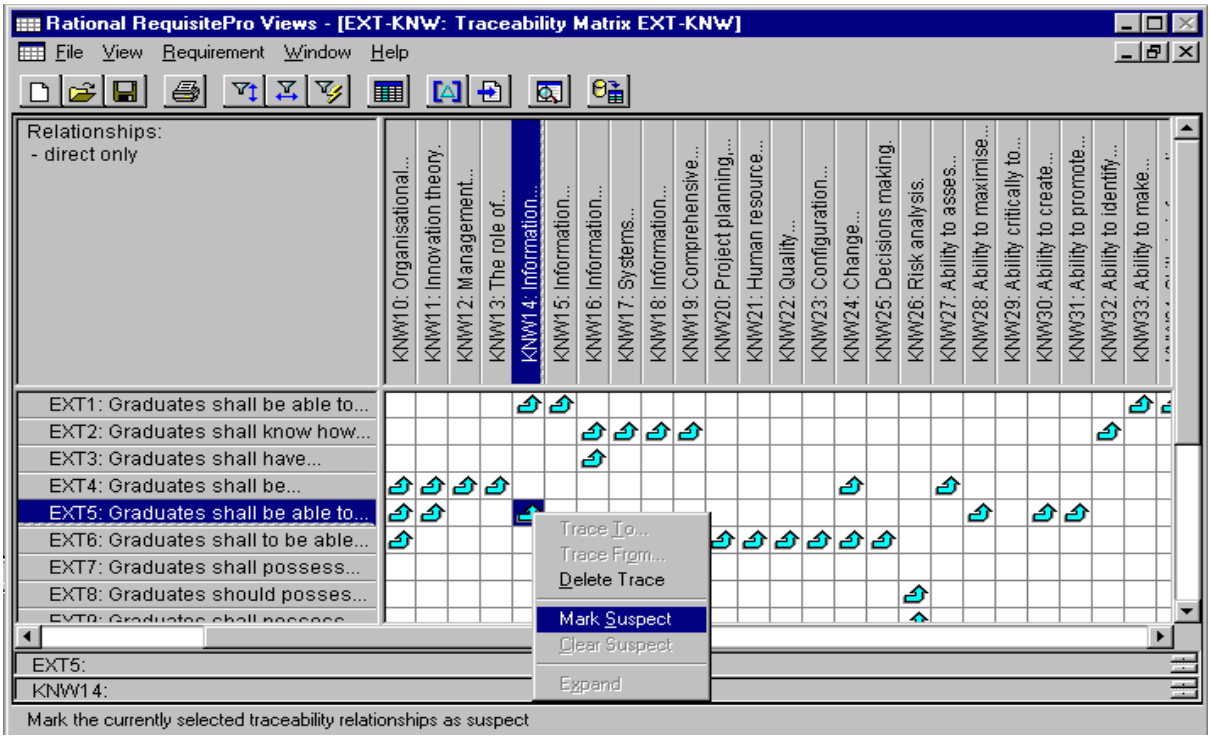


2 pav. Studijų programų reikalavimų vadybos sistema. Pagal [42]

Projekto vykdymo metu, kaip minėta prieš tai, itin svarbus yra efektyvus reikalavimų valdymas – tai potencialiai leidžia sumažinti kaštų sąnaudas sekančiuose projekto etapuose. Tokios struktūros reikalavimų valdymo sistemos negali pilnai suinteresuotosioms šalims parodyti esamą padėtį tarp krūvos reikalavimų vien dėl to, jog tos sistemos nėra pritaikytos arba, geriausiu atveju, pritaikytos minimaliai vizualizuoti pačius reikalavimus ir jų tarpusavio sąryšius.

1.3.1. Reikalavimų susietumo matrica

Reikalavimų susietumo matrica – kaip minėta, plačiai naudojamas metodas, skirtas nagrinėti susietumo ryšius su skirtingų dvejų tipų artefaktais, pavyzdžiui, tarp reikalavimų ir panaudos atvejų arba testavimo protokolų, kuriuos sieja tam tikras ryšys.

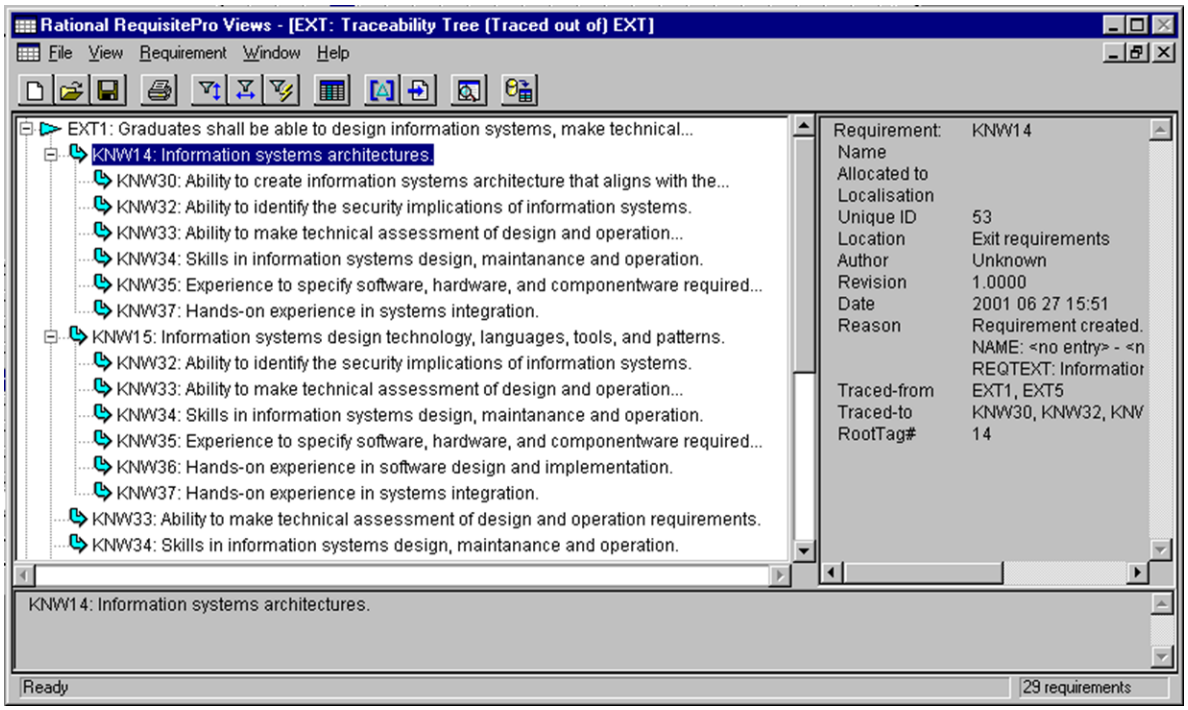


3 pav. Reikalavimų susietumo matrica [38].

Tokios matricos (3 pav.) tikslas yra užtikrinti, jog visi sistemos apibrėžti reikalavimai bus verifikuoji testavimų metu. Įprastai, matrica yra sudaroma kartu su pradiniais reikalavimais ir yra atnaujinama projekto vykdymo metu. [50] Susietumo matricos yra ir lengvai skaitomos, ir suprantamos įvairių profesijų atstovų, todėl yra naudojamos paprasčiausiuose atvejuose, kai tereikia užregistruoti ar patikrinti pavienius ryšius tarp artefaktų. Tačiau, vykdant didesnius projektus, susietumo matricos tampa itin didelės ir sunkiai perskaitomos, ypač jei bandoma suvaldyti skirtingų sluoksnių (atributų) susietumą, naudojant ne vieną matricą.

1.3.2. Reikalavimų vaizdavimas hierarchiniu sąrašu

Šio tipo vizualizacijos – tai sąrašai, pagrįsti hierarchinio medžio principu. Dėl savo paprastumo ir dažno naudojimo, sąrašai nereikalauja specifinių žinių jų nagrinėjimui ir yra lengvai suprantami suinteresuotųjų šalių bei genėtinai intuityvus [6, 17]. Hierarchinio tipo vizualizacijos suteikia aiškų esybių vaizdą ir koncepciją, tačiau turi rimtų trūkumų [37]: jie gali atvaizduoti tik vieno tipo ryšius tarp reikalavimų – dekompoziciją. Todėl įvairių semantinių sąryšių atvaizdavimas nėra galimas. Be to, nėra galimybės matyti viso duomenų masyvo iškart (4 pav.), tad yra labai apribojimas [1] reikalavimų visumos suvokimas. Tai pagrindinė priežastis, dėl kurios reikalavimų sąrašas nėra tinkamas vizualizavimo metodas.



4 pav. Reikalavimų susietumo hierarchinis sąrašas [38].

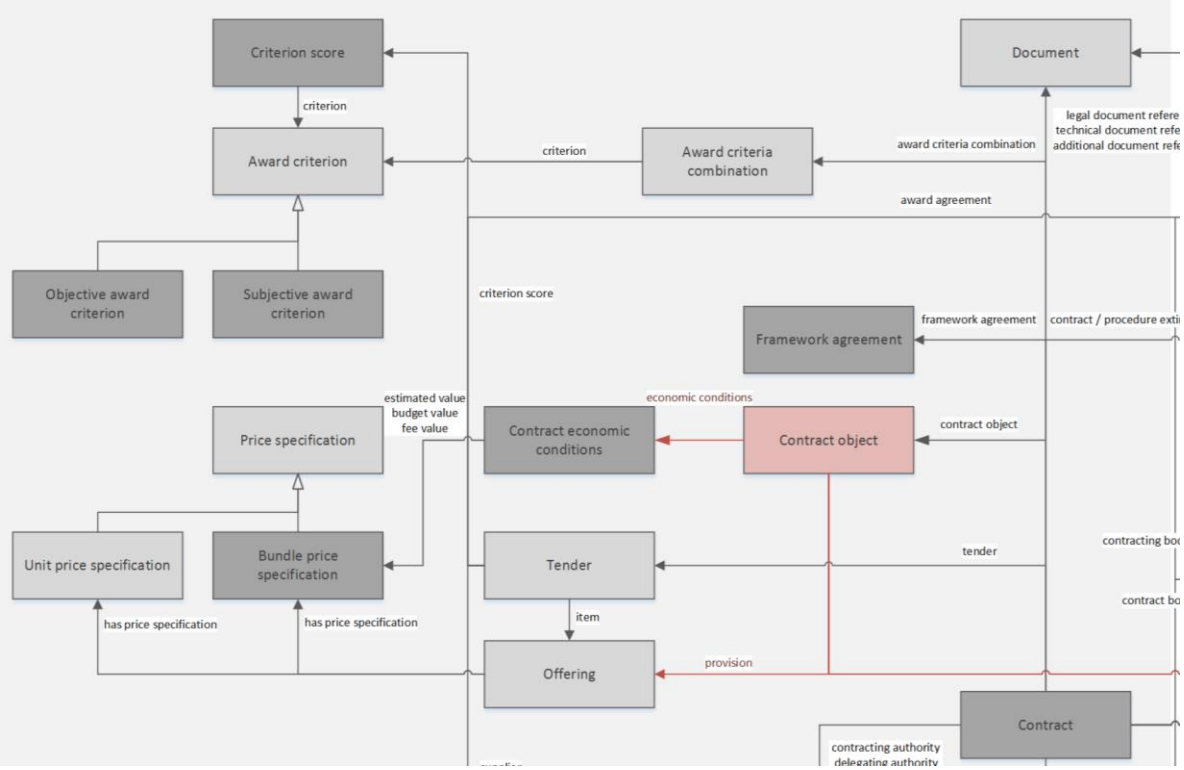
Reikalavimų susietumo hierarchinis sąrašas (4 pav.) – vizualizacijos metodas, pateikiantis reikalavimus, susijusius tarpusavyje „tėvo - vaiko“ ryšiu. Neretai reikalavimai yra susiejami pagal jų kategoriją, aprašomą funkciją ar kitus atributus. Žinomi reikalavimų valdymo įrankiai kaip „IBM Requisite Pro“ ar „IBM DOORS“ naudoja šią atvaizdavimo formą. Bet koks papildomas ryšio tipas, išskyrus dekompoziciją, konvertuotų medžio struktūrą į grafą.

1.3.3. Reikalavimų vizualizavimas grafu

Daugybę skirtingų medžio tipo struktūrų algoritmų yra naudojama duomenų vizualizavimui, pradedant paprastais dvimačiais iki trimačių algoritmų ar hiperbolinių metodų, tačiau jų principas išlieka tas pats – šakos (medžiuose) arba briaunos (grafuose) atitinka sąryšius tarp viršūnių ar mazgų

medžiuose arba grafuose atitinkamai. Grafo tipo vizualizacijos, priešingai nei hierarchinis sąrašas ar bet koks kitas „medžio“ tipo vizualizavimo būdas, leidžia vienu metu atvaizduoti keletą skirtų semantinių sąryšių, įskaitant ir [37] heterogenišką dekompoziciją, sujungiant viršūnę „vaiką“ su visomis kitomis viršūnėms „tėvais“. Dėl šios galimybės toks vizualizavimo būdas naudojamas ne viename [6] duomenų masyvams vaizduoti skirtame įrankyje.

Grafais perteiktos vizualizacijos puikiai tinka [1, 6, 11, 37] perteikti duomenų struktūros visumą. Vis dėlto, šios vizualizacijos neefektyvios [53] kai norima išnaudoti kuo daugiau turimos vietos ekrane. Esant dideliems duomenų rinkiniams, grafu pateikta informacija nėra matoma iš karto visa, o tai gali lemti konteksto praradimą. Be to, esant gausybei sąryšių, ši vizualizacija tampa pernelyg sudėtinga, nes sąryšiai (ir jų pavadinimai) pradeda persidengti.



5 pav. Reikalavimų sąryšių vizualizavimas grafu [12].

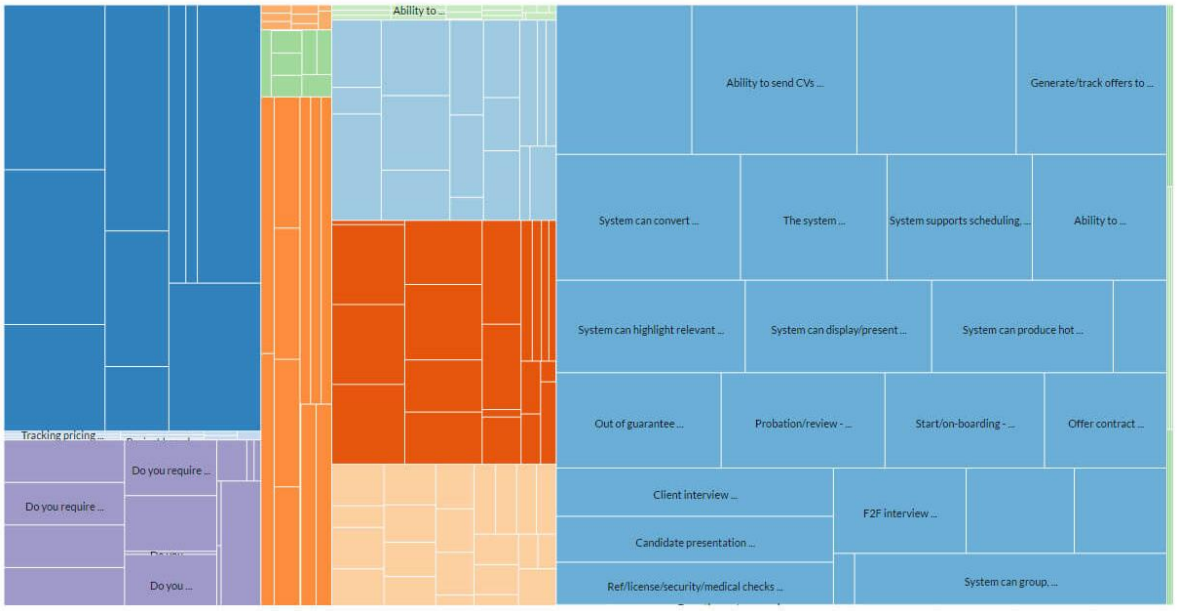
Reikalavimų susietumo ryšius galima atvaizduoti orientuotu grafu (5 pav.), jei grafo viršūnėmis laikysime reikalavimus, o briaunomis - susietumo ryšius [11]. Tai pat kaip ir hierarchinio sąrašo vizualizacija, ši irgi gali pateikti intuityvų sąryšių atvaizdavimą. Grafu grįstos vizualizacijos neretai leidžia išskleisti ir sutraukti kiekvieną mazgą, taip optimizuojant matomą vaizdą, kas gali padėti suinteresuotosioms šalims lengviau orientuotis ir susitelkti tik ties jiems svarbiais reikalavimais, ir reikalavimais, susietais su pastaraisiais. Tačiau, pastebėta, [50] kad žmonės yra linkę perinterpretuoti tokio tipo vizualizacijas, pavyzdžiui, viršūnės (reikalavimai), esančios aukščiau, laikomos svarbesnėmis, nei tos kur žemiau.

1.4. Reikalavimų sąryšių vizualizavimas netradiciniais metodais

Pagrindinis duomenų vizualizavimo tikslas [31, 52] – perteikti turimą informaciją aiškiai ir efektyviai. Dėl to, tinkamiausias būdas perteikti reikalavimų tarpusavio sąryšius – juos tinkamai vizualizuoti. Siekiant aiškesnio „bendro vaizdo“ suinteresuotosioms šalims, skirtingi reikalavimų sąryšių tipai turėtų būti vizualizuojami skirtingai. Laimei, reikalavimų tarpusavio sąryšius galima perteikti įvairiais reikalavimų inžinerijai netradiciniais vizualizavimo metodais. Žemiau pateikiami ir detalizuojami galimi duomenų vaizdavimo metodai, kuriuos pritaikius būtų galima kokybiškiau perteikti reikalavimų tarpusavio susietumą projektuose.

1.4.1. Reikalavimų vizualizavimas Treemap metodu

Treemap vizualizacijos – tai dvimatės (trimatės tokio tipo vizualizacijos vadinamos „Steptree“) vizualizacijos, kurios labai efektyviai užpildo turimą plotą, tuo pačiu atvaizduodamos įvairias hierarchines struktūras, kur jų mazgai – tam tikro dydžio stačiakampiai. Šios vizualizacijos yra naudojamos kai didesnis dėmesys yra skiriamas „lapo“ tipo mazgams – šiuo atveju – reikalavimams, ir skirtos apžvelgti visumą [6, 49], kai pačios hierarchijos yra nereikšmingos. Naudojantis šiuo vizualizacijos metodu galima iš karto pereiti nuo aukščiausio lygio prie žemiausio ir, atvirkščiai, bei matyti jų sąryšius. Treemap struktūra leidžia pilnai išnaudoti turimą erdvę ir be to, prie kiekvieno mazgo galima sieti informaciją, pavyzdžiui, reikalavimo identifikavimo kodą ar trumpą aprašą, o jo dydis rodytų su juo susietų reikalavimų kiekį.



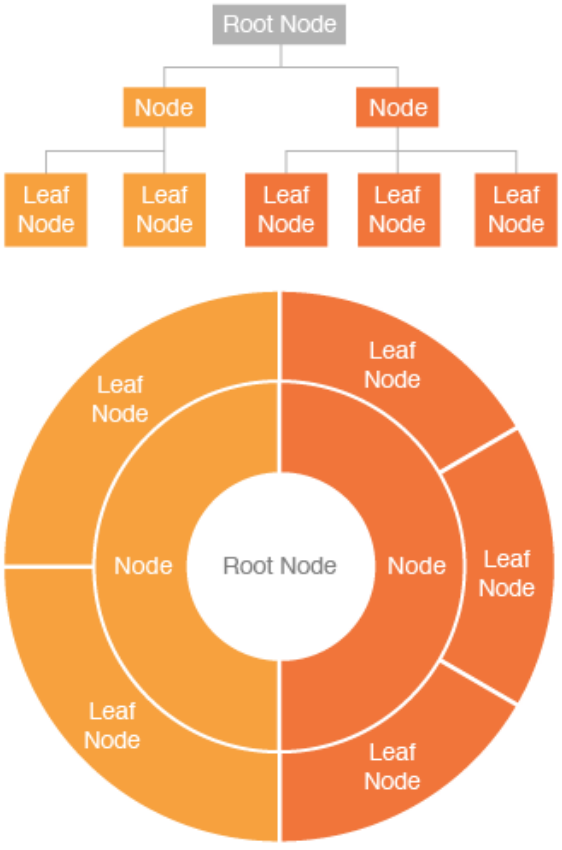
6 pav. Reikalavimų vizualizavimas Treemap metodu („slice-and-dice“ schema) [14].

Nors Treemap vizualizacija gali tą pačią informacija atvaizduoti skirtingomis išdėstymo schemomis (6 pav.), ar naudoti įvairias funkcijas, kurios gerintų struktūros peržiūrą, pavyzdžiui, visos

struktūros priartinimą, atitolinimą ar kt., ja naudojantis galima susidurti su keliomis problemomis: jei viena iš vaizduojamų hierarchijų yra didelė ir gili, tai norint ją suprasti reikalingos [6] ne mažos kognityvinės pastangos; kitas šios vizualizacijos trūkumas – vienu metu naudotojas gali išskleisti tik vieną „šakninį“ reikalavimą (mazgą). Dėl to, šis būdas nėra tinkamas kai norima efektyviai vizualizuoti sudėtingus ar didelius duomenų rinkinius.

1.4.2. Reikalavimų vizualizavimas Sunburst metodu

Sprendžiant iš aukščiau aprašytų duomenų vizualizavimo metodų, jie nėra labai tinkami atvaizduoti didelius kiekius artefaktų. Todėl siūloma naudoti Sunburst vizualizavimo metodus siekiant efektyviai atvaizduoti didelius duomenų rinkinius, nes [20, 6, 50] Sunburst sugeba apdoroti nemažus kiekius artefaktų. Sunburst vizualizacijos metodas - tai grafas, kurio viršūnės išsidėsčiusios apskritimu (7 pav.). Šio tipo vizualizacija suteikia galimybę kokybiškai vizualizuoti reikalavimų kategorizavimą ir susietumo ryšius tarp jų. Šis metodas hierarchijas pateikia per seriją žiedų, kurie sudalinami į tam tikro dydžio mazgus.



7 pav. Sunburst vizualizacijos anatomija [48]

Vizualizacijos centre (7 pav.) patalpinamas pradinis mazgas (gali būti hierarchinio medžio šaknis) ir kiekviename žiede atitinkamai atvaizduojamas hierarchijos lygmuo, pradedant nuo žiedo,

esančio arčiausiai centro. Minėti žiedai yra padalinami į dalis (pavyzdžiui, reikalavimus), priklausomai nuo sąryšio su aukštesnio lygmens žiede esančia dalimi. Dažniausiai, žiedo dalies dydis priklauso nuo to, kiek jis turi sąryšių su žemesnio lygmens žiedo dalimis. Taip vaizduojamas grafas yra kompaktiškesnis ir suteikia naudotojui geresnę orientaciją jame. Priešingai nei kiti vizualizavimo metodai, šis, didėjant artefaktų skaičiui, plečiasi į visas puses. Taip pat, neretai šio tipo vizualizacijos naudoja įvairias spalvų schemas [50], siekiant pavaizduoti tam tikrą duomenų klasifikavimą.

Sunburst vizualizacijos yra dažnai lyginamos [1, 3, 21] su Treemap vizualizacijomis įvairiais tyrimais - jais siekiama parodyti skirtumus, panašumus ar pranašumus tarp šių metodų. Atliktuose tyrimuose [1, 3], lyginant hierarchinį sąrašą, Treemap ir Sunburst vizualizacijų metodus, dalyvių buvo prašoma atlikti įvairias, su navigacija, paieška ir semantiniu sąryšiu susijusias užduotis. Vieno tyrimo [21] metu gauti rezultatai parodė, jog esant nedideliems duomenų kiekiams (500 artefaktų) skirtumas tarp Sunburst ir Treemap vizualizacijos įrankių buvo nežymus. Tačiau, tyrimo dalyviams atliekant tas pačias užduotis su 3000 artefaktų parodė, jog naudojant Sunburst vizualizacijas, tos pačios užduotys buvo atliekamos greičiau ir tiksliau nei naudojant kitus, tyrimuose naudotus metodus. Pasak tyrimo autorių tai lėmė, jog Sunburst vizualizacija yra intuityvesnė nei kitos. Be to, dauguma žmonių [40] yra apmokyti suprasti ir analizuoti skritulines diagramas, o Sunburst tėra „evoliucionavusi“ tokio tipo diagrama.

Sunburst vizualizacija, kaip ir kitos turi savų trūkumų. Nors ji ir intuityvesnė nei Treemap, tačiau ši vizualizacija yra sunkiau suvokiama, nei kiti vizualizavimo metodai (pvz. hierarchinis sąrašas). Kitas šios vizualizacijos trūkumas – turimas plotas negali būti pilnai išnaudotas dėl savo formos. Didžiausia šio metodo problema – išoriniame žiede, kai mazgų skaičius yra didelis, jie (mazgai) tampa smulkūs ir sunkiai suskaitomi. Tačiau ši problema gali būti sušvelninta paslepiant tuo metu neaktualius mazgus, tuomet esančios viršūnės bus perpiešiamos į didesnes.

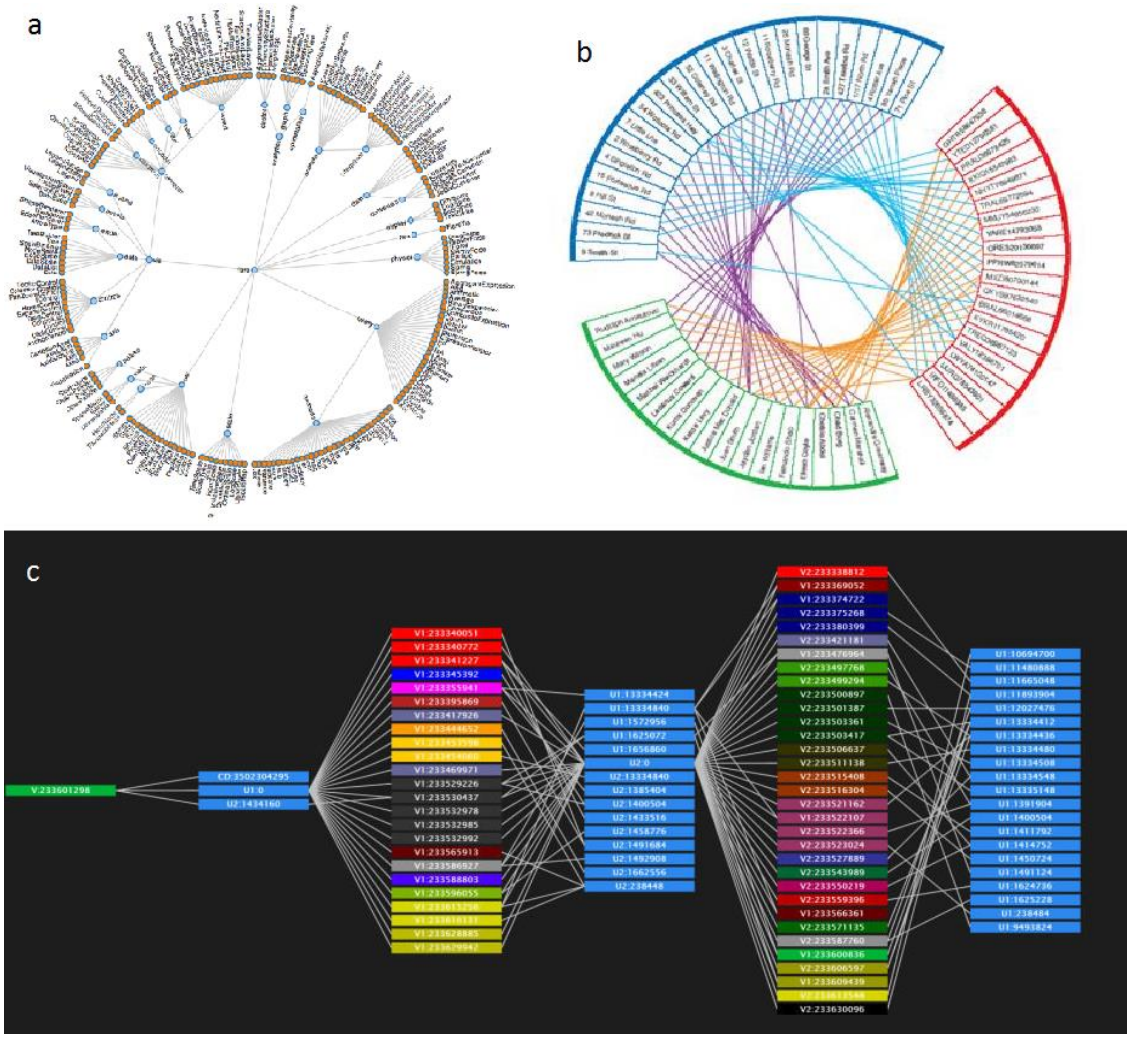
1.4.3. Reikalavimų vizualizavimas Netmap metodu

Netmap vizualizacija šiek tiek skiriasi nuo prieš tai aprašytų vizualizacijų: šis metodas naudojamas kai norima grafiškai pavaizduoti vidinius sąryšius tarp esybių. Kaip ir kitos vizualizacijos, Netmap taip pat gali būti pritaikyta atvaizduoti sąryšius tarp reikalavimų, nes sudaryta iš fundamentalių elementų:

- Mazgo – loginės esybės. Ji gali būti charakterizuota atributais, tad kiekvienas mazgas taip pat gali turėti ir tam tikrą, jam priskirtą reikšmę, pavyzdžiui, kiekvienas mazgas atitinka reikalavimą, bet jo spalva ar tekstas prie jo – reikalavimo prioritetą.

- Briaunos – sąryšio tarp dvejų mazgų. Kaip ir kituose vizualizacijoje, naudojančiuose tokią notaciją, briauna vaizduoja ryšį tarp esybių. Tačiau šiame metode – briauna gali vaizduoti bet kokį semantinį ryšį, o ne tik dekompoziciją. Taip pat čia briaunos gali turėti ir priskirtas reikšmes, pavyzdžiui, sąryšio tipą.

Savo grafine išraiška, ši vizualizacija primena hierarchinį medį ar grafą, tačiau priešingai nei pastarieji, nevaizduoja hierarchinės struktūros, nes čia briaunos vienu metu gali vaizduoti skirtingų tipų sąryšius.



8 pav. Netmap vizualizacijos schemos.

Netmap vizualizacija iš kitų, dar darbe aprašytų metodų, išsiskiria tuo, jog turi įvairių adaptacijų, kuriomis tuos pačius duomenų rinkinius galima atvaizduoti skirtingai [35]:

- „Bullseye“ schema. Šioje schemoje (8a pav.) pirminė mazgų grupė atsiduria figūros centre, kurios sąryšiai „skleidžiasi“ nuo centro į šonus, sudarydami ryšius su kitomis mazgų grupėmis.

- „Netmap“ schema – vyraujanti sąryšių atvaizdavimo schema. Čia (8b pav.). Mazgai vaizduoji viename žiede vienodai. Vidiniame rate yra vaizduojami susietumo ryšiai. Kaip ir vaizduojami mazgai, taip ir tarpusavio ryšiai yra vaizduojami skirtingomis spalvomis. Be to, galimas ir skirtingas ryšių atvaizdavimas - rodykle, linija ir kt. - priklausomai nuo ryšio tipo. Suprantama, atvaizduojant visas viršūnes viename žiede, sumažinamas kiekvieno mazgo vaizduojamasis plotas, o ir ryšiai tampa sunkiai atsekami. Tokiu atveju siūloma [50] naudoti filtrus arba papildomas vizualizacijas.
- „Koloninė“ schema. Šioje (8c pav.) schemoje mazgų grupės išdėstomos kolonomis iš kairės į dešinę. Įprastai, kolonose mazgų rikiavimas nėra taikomas, nes taip potencialiai sumažinamas [35] galimų sąryšių persidengimas.
- „Eilių“ schema. –tai schema, kurioje mazgų grupės išrikiuojamos vertikaliai – nuo viršaus į apačią. Taip pat kaip ir „koloninėje“ schemoje, mazgai grupėse nėra rikiuojami.
- „Eilių/Kolonų“ schema. Hibridinė pastarųjų dvejų schemų vizualizacija [35], kurioje pradinio lygmens grupės išrikiuojamos horizontalia eile, o sekančios – vertikaliai.

1.5. Skyriaus apibendrinimas

Siekiant, jog visi programinei įrangai keliami reikalavimai būtų įgyvendinti, jie turi būti dokumentuoti – tačiau įprasti būdai ir metodai negali tinkamai perteikti reikalavimų visumos ir jų tarpusavio susietumą suinteresuotosioms šalims, o tai, esant didesniam reikalavimų kiekiui, gali vesti į prastą reikalavimų valdymą, o šis – į didesnį kaštų poreikį vykdomam projektui.

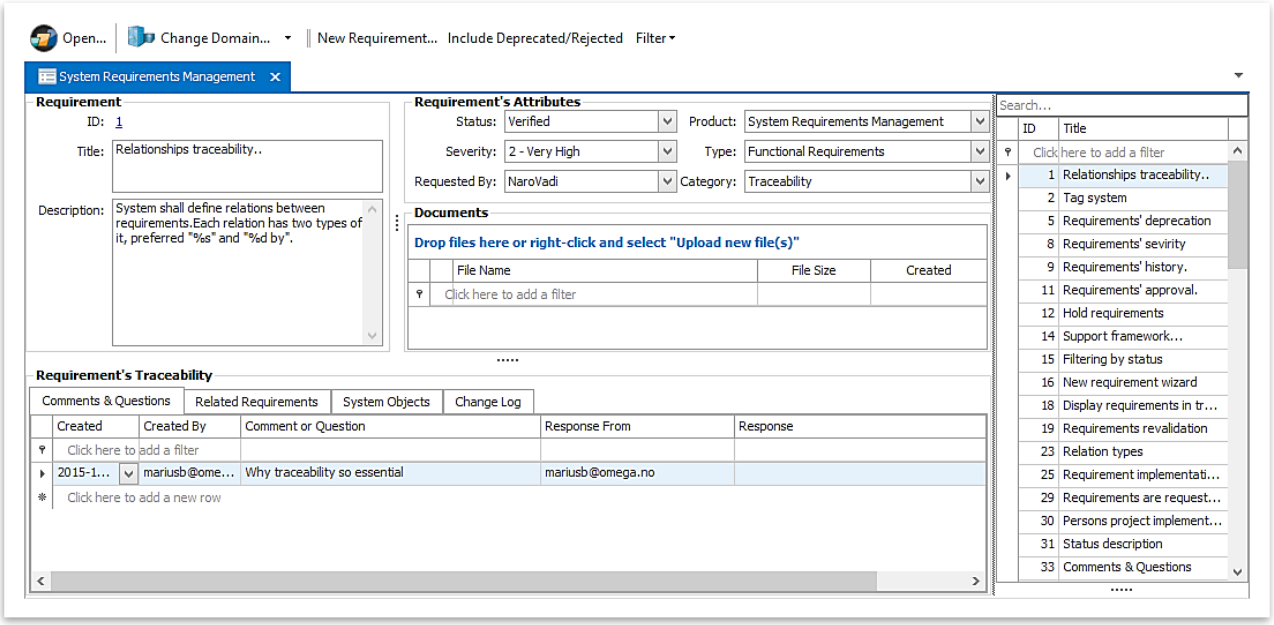
Programinių sistemų, skirtų valdyti reikalavimus, rinkoje dėmesys yra skiriamas reikalavimų importavimui iš įvairių atskirų dokumentų, jų atributų keitimui projekto vykdymo metu ir ataskaitų generavimui – ir nors tai leidžia tam tikram lygmenyje tvarkyti reikalavimus, tačiau negali naudotojams efektyviai pateikti jų visumą ir tarpusavio sąveikas.

Reikalavimų ir tarpusavio sąryšius galima vizualizuoti pritaikant duomenų vaizdavimo metodus, naudojamus duomenų gavyboje – programinė sistema, kuri realizuotų tokius metodus, išsiskirtų iš kitų esamų reikalavimo valdymo sistemų savo gebėjimu reprezentuoti reikalavimų visumą vienu metu, o tokios sistemos tinkamas interaktyvumas leistų efektyvesnį programinės įrangos reikalavimų valdymą.

II. KURIAMOS PROGRAMINĖS ĮRANGOS PROJEKTAVIMAS

2.1. Kuriamo programinės įrangos prototipo aktualumas ir naujumas

Autoriui kuriant reikalavimų valdymo sistemos prototipą buvo susidurta su keliomis problemomis, kurios buvo sėkmingai įspręstos (žr. [28]). Tačiau tuomet nebuvo akcentuojama tai, jog didėjant duomenų kiekiui, ypač kai tie duomenys nėra imitaciniai, o realūs, sudėtingėja reikalavimų valdymas: darosi sudėtinga rasti reikalavimų dublikatus ar prieštaras, sunku aprėpti ir suvokti reikalavimų tarpusavio sąryšius. Be to, didėjant reikalavimų kiekiui taip pat vis sunkiau atlikti navigaciją tarp reikalavimų, mat jų skiriamasis bruožas buvo tik eilės numeris. Taip pat nebuvo jų išskirstymo pagal priklausomumą tam tikram programiniam moduliui, t. y. reikalavimai nebuvo atskirsti pagal projektus, tad reikėdavo naudoti vartoto apibrėžtus filtrus. Verta paminėti, jog sukurtos sistemos prototipo vartotojo grafinė sąsaja (9 pav.) nebuvo pakankamai intuityvi, tad tai tik apsunkino minėtų procesų vykdymą.



9 pav. Sistemos reikalavimų valdymo prototipas, pirmoji versija.

Vartotojo grafinė sąsaja buvo sudaryta iš šių valdymo panelių (9 pav.): „Reikalavimas“, „Atributai“, „Dokumentai“, „Reikalavimo atsekamumas“ ir navigacijos panelė.

Atsižvelgiant į minėtas problemas buvo nuspręsta pakeisti sistemos dizainą taip, jog visas dėmesys butų skirtas reikalavimų atsekamumui ir pačiam jų valdymui. Dėl to pasirinkta reikalavimus ir jų sąryšius vizualizuoti, taip padidinant galimybę geriau suvokti procesus, vykstančius su programinės įrangos reikalavimais. Pasirinkti vizualizavimo metodai - Sunburst ir Netmap: pirmasis metodas, kaip pastebėta analizuojant vizualizavimo metodus, yra labiausiai tinkamas dideliems

duomenų kiekiams – jis leidžia matyti reikalavimų visumą bei, nereikalaudamas papildomų žinių apie jį, geba pakankamai aiškiai atvaizduoti vieno tipo sąryšius tarp reikalavimų. Antrasis metodas pasirinktas dėl galimybės atvaizduoti įvairių tipų sąryšius tarp susijusių reikalavimų, išlaikant tam tikrą vaizdavimo struktūrą.

Šių dvejų, žiedinio ir hierarchinio, metodų kompozicija puikiai tinka vizualizuoti reikalavimų charakteristikas, nes Sunburst vizualizavimo metodas pateikia programinės įrangos reikalavimų susietumo visumą, o Netmap vizualizavimo metodas – konkretaus reikalavimo sąryšius su kitais reikalavimais. Be to, toks dvejų skirtingų vizualizacijų metodų panaudojimo būdas reikalavimų valdymui dar nėra praktiškai išbandytas, tad kuriamas prototipas taip pat tampa ir eksperimentine sistema.

2.2. Sistemos reikalavimų valdymo prototipui keliami reikalavimai

Kuriamam programinio įrankio prototipui dalis funkcinių reikalavimų „ateina“ iš prieš tai kurtos prototipo versijos: reikalavimų peržiūra, jų valdymas, galimybė konfigūruoti sistemą ir t. t. Kartu su jais išlieka ir tos pačios naudotojų rolės, kurios suteikia vienokias ar kitokias prieigos teises prie sistemos ir jose esančių duomenų – tai „SRM administratorius“, „Projekto vadybininkas“ ir „Svečias“. 10 paveiksle pateikta panaudos atvejų diagrama grafiškai pateikia kiekvienos rolės galimybes sistemoje.

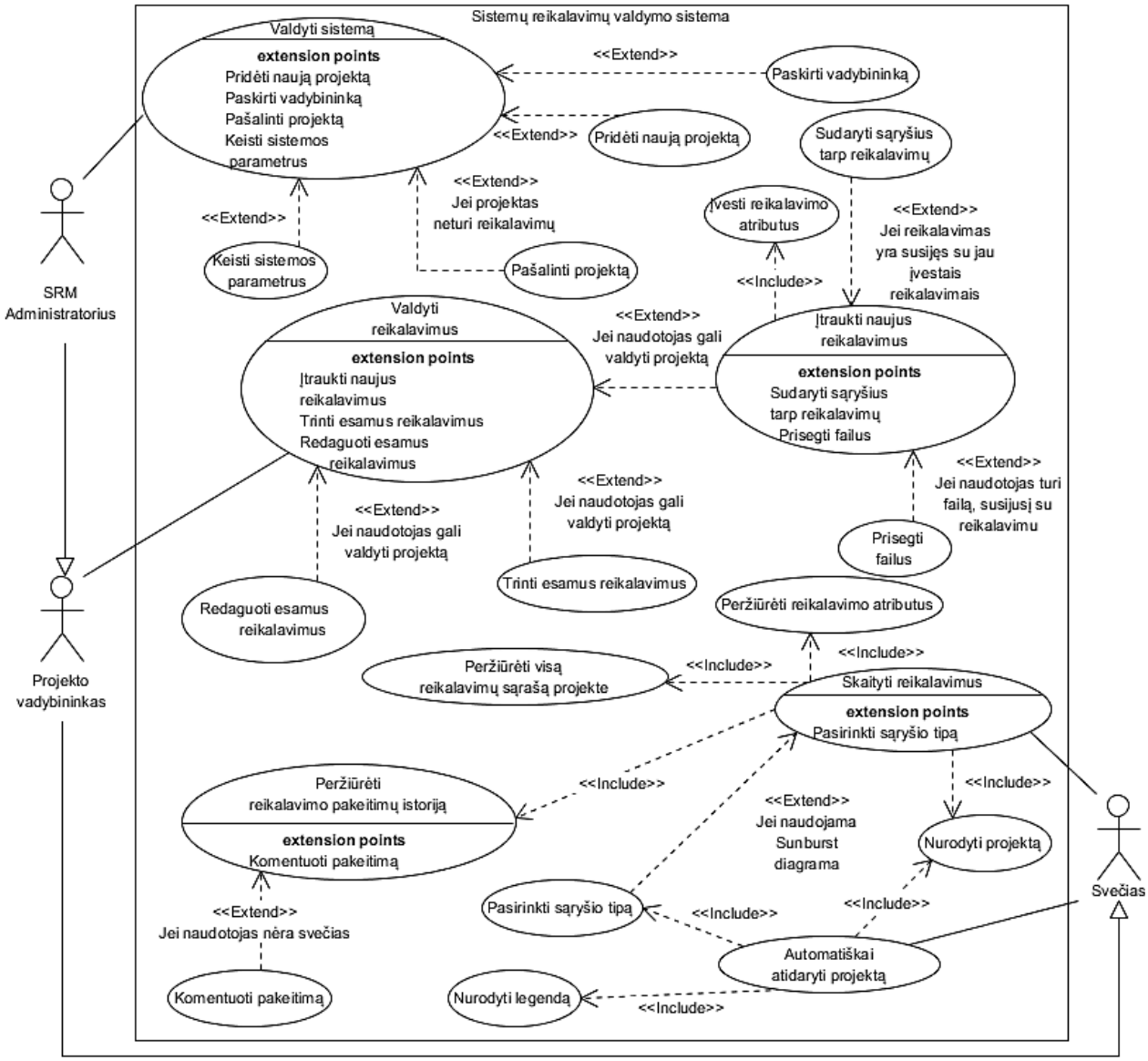
Kadangi naujai kuriamas sistemos prototipas yra pagrįstas grafine vizualizacija, natūralu, kad sistemai yra keliami nauji, su grafine naudotojo sąsaja susiję, reikalavimai: apibrėžiamos reikalavimų vaizdavimo taisyklės, apribojamas sąryšių tarp reikalavimų sudarymas, detalizuojami grafinės naudotojo sąsajos elementai ir jų galimybės bei charakteristikos.

Visi kuriamam prototipui keliami reikalavimai gali būti sugrupuoti į 4 grupes:

- Naudotojų rolės. Šiais reikalavimais yra apibrėžiamos sistemoje naudojamos rolės bei jų galimybės ir apribojimai.
- Reikalavimų valdymas. Reikalavimai apibrėžia naudojamus atributus reikalavimų specifikacijai, jų numatytas reikšmes ir svarbumą. Taip pat apibūdina galimus veiksmus su reikalavimais ir jų susietumą su kitais sistemos objektais (failais, grafiniais elementais, duomenų bazės objektais, trečių šalių programine įranga ir t. t.)
- Grafinės naudotojo sąsajos elementų charakteristikos. Šie reikalavimai nusako, kaip sistema turi reaguoti į tam tikrus naudotojo veiksmus, kokia informaciją pateikti ir kaip ją tinkamai atvaizduoti.

- Reikalavimų sąryšiai. Nusakomos sąryšių tarp reikalavimų charakteristikos, jų naudojimo galimybės.

Pilnas reikalavimų sąrašas pateikiamas priede Nr. 1.



10 pav. Panaudos atvejų diagrama

10 pav. pateikta panaudos atvejų diagrama rodo kokias galimybes turi kiekviena rolė reikalavimų valdymo sistemoje: žemiausio lygmens rolė „Svečias“ gali atlikti tik fundamentalius veiksmus, kurių pakanka peržiūrėti sistemoje egzistuojančių projektų reikalavimus. Šios rolės nariai turėtų būti visų suinteresuotųjų šalių nariai, jų klientai ir kt. asmenys. Šiai rolei pasiekiami panaudos atvejai:

- skaityti reikalavimus,
- peržiūrėti reikalavimo atributus,

- peržiūrėti visą reikalavimų sąrašą projekte,
- peržiūrėti reikalavimo pakeitimų istoriją,
- nurodyti projektą,
- automatiškai atidaryti projektą,
- pasirinkti sąryšio tipą,
- nurodyti legendą.

Aukštesnio lygmens rolė – „Projekto vadybininkas“. Šiai rolei priklausyti gali tik „Pims“ sistemoje registruoti naudotojai. Rolės teisės papildomai leidžia jos nariams valdyti jiems paskirtų projektų reikalavimus, keisti jų (projektų) konfigūraciją. Jos nariams pasiekiami panaudos atvejai:

- valdyti reikalavimus,
- redaguoti esamus reikalavimus,
- įtraukti naujus reikalavimus,
- trinti esamus reikalavimus,
- įvesti reikalavimo atributus,
- sudaryti sąryšius tarp reikalavimų,
- prisegti failus,
- komentuoti pakeitimą.

Rolės nariai paveldi „Svečio“ pasiekiamus panaudos atvejus.

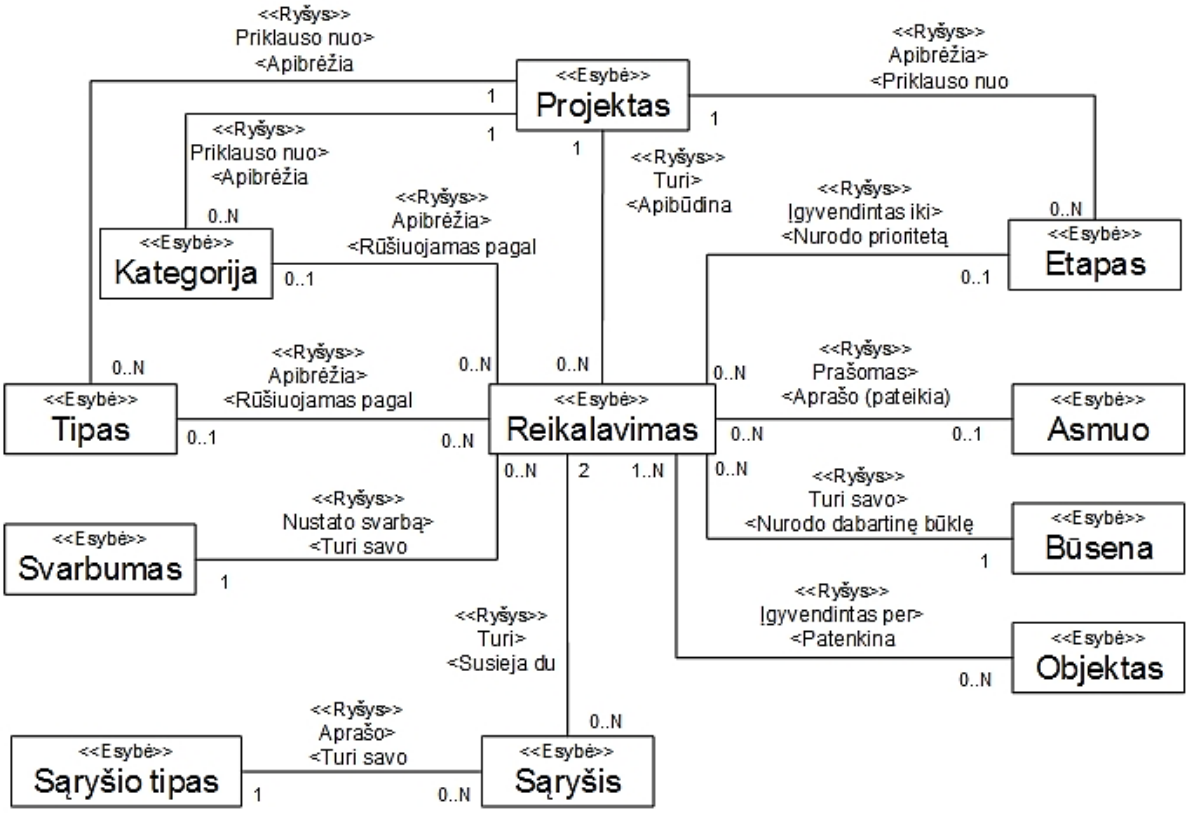
Paskutinė ir aukščiausio lygmens rolė – „SRM Administratorius“. Šios rolės narys paskiria projektams atsakingus asmenis, gali valdyti ir konfigūruoti visą sistemą. Kaip ir „Projekto vadybininkas“, šios rolės narys gali būti tik sistemoje registruotas naudotojas. Šios rolės nariui pasiekiami pasiekimai panaudos atvejai:

- valdyti sistemą,
- keisti sistemos parametrus,
- pašalinti projektą,
- pridėti naują projektą,
- paskirti projekto vadybininką.

Rolės nariai paveldi „Projekto vadybininko“ pasiekiamus panaudos atvejus.

2.3. Esybių-ryšių modelis

Esybių-ryšių diagrama (ER) – viena iš populiariausių diagramų, siekiant pavaizduoti duomenų struktūrą. Joje (11 pav.), šiuo atveju, vaizduojama kuriamai sistemai reikalinga duomenų struktūra - pagal ją bus sudaryta duomenų bazė, kurioje bus saugoma visa sistemai reikalinga ir vartotojams pasiekiama informacija.



11 pav. Esybių ryšių diagrama

Kuriamoje reikalavimų valdymo sistemoje, reikalavimai yra grupuojami pagal projektus, kuriems jie atitinkamai priklauso. Kiekvienas projektas gali būti apibūdintas tiek daugybės reikalavimų, tiek neapibūdintas jais visai, jei projektas dar tik inicijuojamas. Taip pat kiekvienas projektas apibrėžia savo etapus (angl. *milestone*), reikalavimų kategorijas (angl. *category*) ir tipus (angl. *type*). Pagal pastarąsias esybės galima spręsti apie kiekvieno reikalavimo pobūdį, t. y. kokią projekto funkciją ar verslo taisyklę galimai aprašo konkretus reikalavimas.

Visi reikalavimai yra pateikiami tam tikro asmens, ir priklausomai nuo projekto – tas asmuo gali būti tiek fizinis, tiek juridinis. Tačiau, tam tikrais atvejais, pavyzdžiui projekto įvedimo į sistemą metu, visi projektą aprašantys reikalavimai gali būti gauti iš vieno šaltinio – projekto dokumentacijos.

Nepriklausomai nuo projekto, kiekvienas į sistemą įtrauktas reikalavimas iškart turi du jį apibūdinančius atributus: svarbumą (angl. *severity*) ir būseną (angl. *status*). Pirmasis atributas nurodo

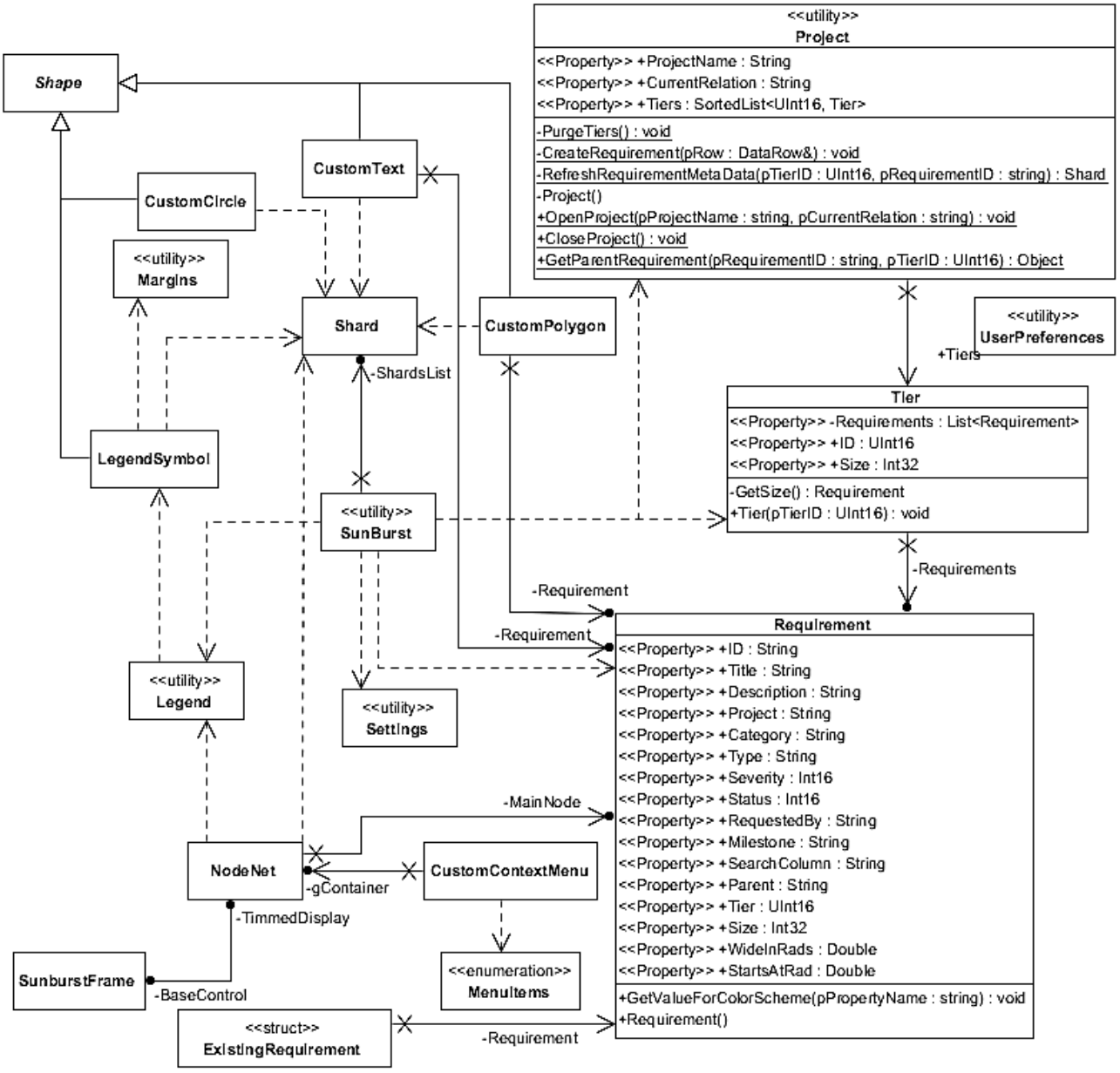
reikalavimo svarbumą sėkmingam projekto užbaigimui , o antrasis – konkrečiu laiko mementu esamą reikalavimo įgyvendinimo etapą, pavyzdžiui, „pateiktas“, „atmestas“, „įgyvendintas“ ir kt. Reikalavimas vienu momentu gali turėti tik vieną būseną ir būti konkretaus svarbumo - tuo pačiu metu jis negali būti „labai svarbus“ ir „beveik nesvarbus“.

Priklausomai nuo reikalavimo aprašymo, jis gali turėti daug sąryšių (angl. *relations*) su kitais reikalavimais arba neturėti jų visai. Nors vienas reikalavimas ir gali turėti daug sąryšių, tačiau vienas sąryšis gali jungti du, ir tik du reikalavimus. Kadangi reikalavimai tarpusavyje gali turėti skirtingus sąryšius, pavyzdžiui „prieštarauja“, „papildo“ ir kt., tai kiekvienas sudarytas ryšys tarp dvejų reikalavimų yra nusakomas sąryšio tipo (angl. *relation type*). Reikalavimo sąryšis, nepriklausomai nuo sąryšio tipo, su savimi yra negalimas. Sąryšių tipai, taip pat kaip ir reikalavimų svarbumas ir būsena, dėl sistemoje esančių projektų lengvesnio skaitomumo, nepriklauso nuo konkretaus projekto ir yra vienodi visuose projektuose.

Nepriklausomai nuo reikalavimo tipo, kategorijos ir kitų jo atributų, galiausiai jis yra realizuojamas per objektus, kurių visuma sudaro galutinį projekto rezultatą (produktą, paslaugą ir pan.). Vienas reikalavimas gali būti įgyvendintas per ne vieną objektą, taip pat kaip vienas objektas gali realizuoti ne vieną reikalavimą.

2.4. Klasių diagrama

Siekiant realizuoti pasirinktų vizualizacijų algoritmus reikalinga duomenų struktūra, kuri semantiškai atitiktų sudaromų diagramų struktūrą – tai leistų optimaliai sudaryti ir nubraižyti reikiamą diagramą, o ir pats duomenų valdymas būtų paprastesnis, nes jų pokytis būtų lengvai reprezentuojamas grafiniame pavidale. Kadangi Sunburst diagrama sudaryta iš žiedų, o šie iš elementų, kur kiekvienas elementas atitinką tam tikrą artefaktą duomenų bazėje t. y. kiekvienas elementas vaizduoją projekto reikalavimą, tai reiškia, kad: fundamentalus konstruojamos duomenų struktūros objektas yra *reikalavimas*, kurie (objektai) bus apjungiami į masyvus (žiedus) su indeksais, atitinkančiais to masyvo (žiedo) *lygmenį*. Suprantama, reikalavimai privalo priklausyti vienam iš sistemoje egzistuojančių *projektų*. Tuo tarpu Netmap vizualizacijai, dėl savo paprastumo, tėra reikalingas tik *reikalavimas*. Ši duomenų struktūra yra vaizduojama 12 pav. pateikta klasių diagrama.



12 pav. Kuriamos programinės sistemos klasių diagrama.

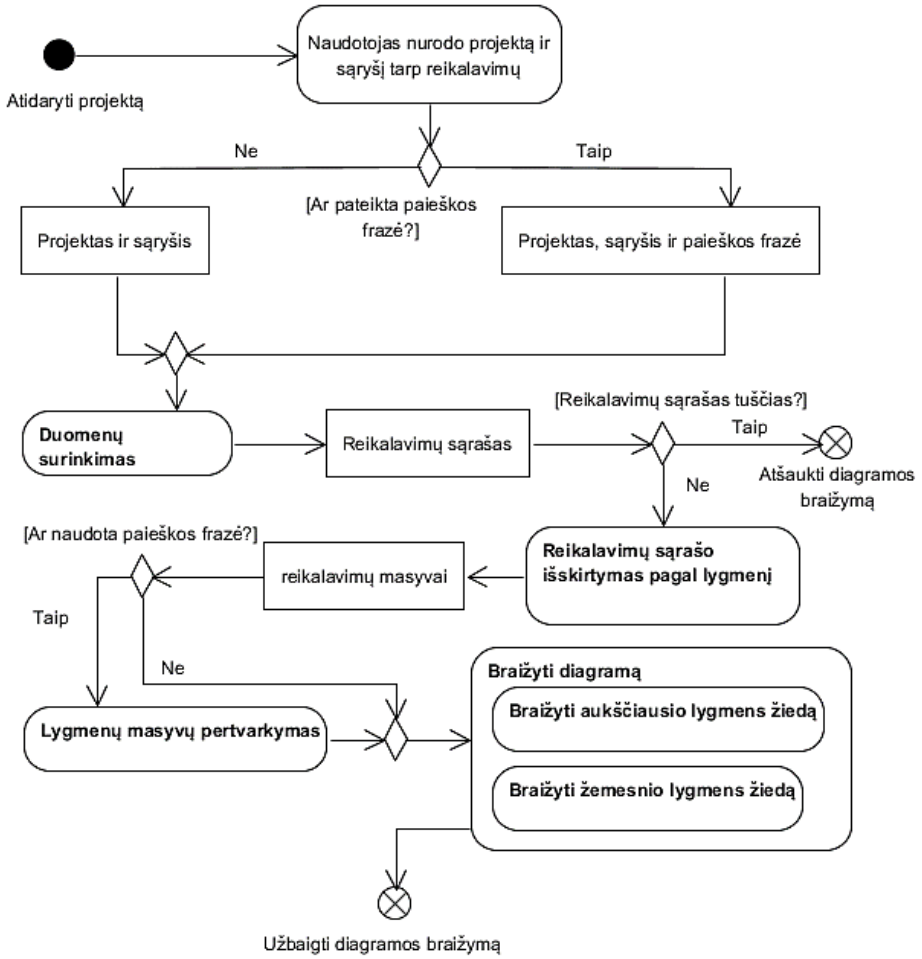
Diagramoje (12 pav.) detalizuotos tik tos klasės, kurios sudaro esminę duomenų struktūrą. Detalesnis visų klasių vaizdas pateikiamas 2 priede.

Programinės įrangos prototipas suprojektuotas taip, kad klasės *Requirement* objektais yra operuojama visoje sistemoje, kas kartą nekuriant naujų objektų, bet perdavinėjant projekto atidarymo pradžioje sukurtų ar naujai įtrauktų *Requirement* klasės objektų adresus (kopijuojama rodyklė) tarp skirtingų klasių objektų ir metodų - tai ne tik leidžia padidinti sistemos našumą, bet ir sumažinti kreipimųsi į duomenų bazę skaičių. Taip pat atkreiptina, jog yra realizuojamos penkios „utility“ tipo klasės: šios klasės naudojamos kaip statinių metodų ir parametrų rinkiniai, siekiant supaprastinti esminių reikšmių, pvz. projekto kodo, spalvų paletės, diagramų elementai ir t.t. pasiekiamumą ir naudojimą.

Modeliuojant sistemą susidurta su problema, jog standartinės .Net bibliotekos figūros neturi visų reikiamų atributų, tad nuspręsta realizuoti abstrakčią klasę *Shape* kurios atributus ir metodus paveldėtų klasės *CustomCircle*, *CustomPolygon*, *CustomText* ir *LegendSymbol* – taip bus išvengiamas kodo dubliavimas šiose klasėse, siekiant realizuoti jų objektus.

2.5. Sunburst metodo algoritmas reikalavimų vizualizavimui

Siekiant įgyvendinti pasirinktus uždavinius ir tikslą, reikalingas įrankis, su kuriuo būtų galima vaizduoti reikalavimus, tačiau tam tinkamo įrankio nėra arba jie nėra lengvai pasiekiami naudojimui. Todėl buvo nuspręsta sukurti tokį įrankį, gebantį atvaizduoti turimus duomenis Sunburst vizualizacijos metodu. Šis vizualizacijos metodas bus naudojamas kaip pagrindinis įrankis, kurio pagalba bus vaizduojami naudotoją dominančio projekto reikalavimų visuma per pasirinktą jų sąryšių tipo prizmę. Verta paminėti, kad gairės kaip sukurti minėtą įrankį, kaip ir tokie įrankiai, nėra viešai prieinamos. Dėl to buvo nuspręsta pateikti savo algoritmą (13 pav.), skirtą Sunburst vizualizavimo metodui.



13 pav. Sunburst algoritmo schema

Algoritmas inicijuojamas kai naudotojas pasirenka projektą ir nurodo sąryšio tipą, kuriuo turi būti vaizduojami projekto reikalavimai. Taip pat, papildomai, naudotojas gali apriboti ketinamų vaizduoti reikalavimų kiekį nurodydamas filtrą – paieškos frazę (13 pav.). Naudotojui nurodžius šiuos parametrus, inicijuojamas I etapas - duomenų surinkimas: čia yra surenkami ne tik kiekvieno, parametrus atitikusio reikalavimo meta-duomenys, bet kartu ir apskaičiuojama, kokio lygmens (angl. *tier*) yra kiekvienas konkretus reikalavimas ir kiek jis (reikalavimas) turi jam priklausančių reikalavimų, esančių visuose už jo paties žemesniuose lygmenyse, kitais žodžiais – paskaičiuojamas reikalavimo dydis.

Kiekvienam reikalavimui ***R*** jo pozicija ir dydis apskaičiuojama pagal autoriaus pasiūlytą ir žemiau pateiktą algoritmą:

1. Surenkamos visos reikalavimų poros su nurodytu tarpusavio sąryšiu, kuriuose reikalavimas ***R*** yra žemesnio lygmens reikalavimas;
2. Apskaičiuojamas reikalavimo ***R*** lygis sudarytame medyje pagal nurodytą tarpusavio sąryšį. Verta paminėti, kad medis nėra generuojami pilnai – vietoj to, sudaroma tik atšaka, kurios viršūnė – reikalavimas ***R***;
3. Apskaičiuojamas sudaryto medžio, kur reikalavimas ***R*** yra to medžio šaknis, dydis - t. y. reikalavimo dydis.

Gali būti taip, jog dėl naudotojo nurodytų atrankos parametrų (pvz. projektas dar neturi jam priklausančių reikalavimų arba nei vienas reikalavimas neatitiko paieškos frazės) sistema negalėjo gražinti sąrašo, kuriame būtų bent vienas reikalavimas. Tokiu atveju, vizualizavimo projektas yra nutraukiamas, nes paprasčiausiai įrankis neturi ką atvaizduoti.

Jei gražintas reikalavimų sąrašas nebuvo tuščias, tai šis yra išskirstomas pagal reikalavimų lygmens indeksą į masyvus, pagal kurie reprezentuoja Sunburst diagramos žiedus. Kiekvienas sudarytas masyvas turi lygmens indeksą, kuris nurodo, kurie reikalavimai jam priklauso ir kurį žiedą jis atitinka.

Jei duomenų surinkimui buvo naudota paieškos frazė, tai papildomai atliekami veiksmai, kurie pertvarko minėtus masyvus taip, jog masyvų lygmens indeksai, nesukeičiant jų sekos sudarytų nepertraukta seka, pradedant nuliu,: pavyzdžiui, jei suskirstytų reikalavimų masyvų indeksai sudarė aibę $\{1, 2, 4\}$ tai ši pertvarkoma į aibę su elementais $\{0, 1, 2\}$.

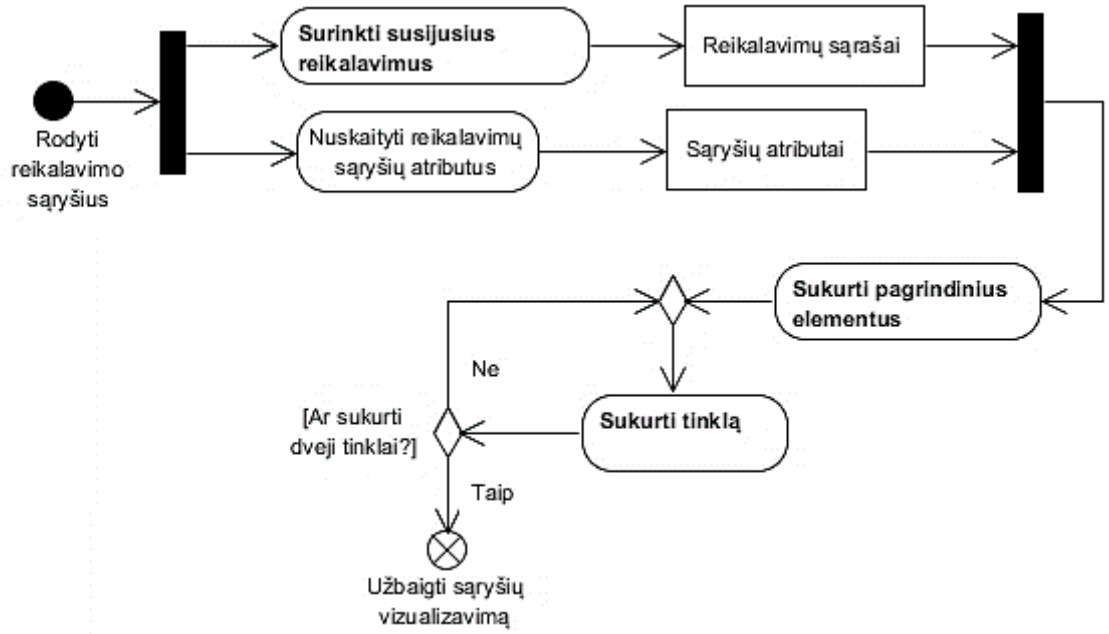
Galiausiai, inicijuojamas diagramos braižymas: su kiekvieno sudaryto masyvo, pradedant aukščiausiu lygmeniu (indeksas lygus 0), kiekvienam reikalavimui yra braižomas daugiakampis:

1. Apskaičiuojamas daugiakampio plotis ir kampas (rad);
2. Apskaičiuojamas daugiakampio plotas (s. v.)^2 ;
3. Nustatoma daugiakampio padėtis žiede;
4. Apskaičiuojamos daugiakampio viršūnių koordinatės;
5. Nubraižomas daugiakampis;
6. Daugiakampio centre užrašomas vizualizuoto reikalavimo ID.

Detalesnė Sunburst vizualizacijos algoritmo diagrama pateikta 3 priede.

2.6. Netmap metodo algoritmas reikalavimų susietumo vizualizavimui

Netmap vizualizacija bus taikoma siekiant grafiškai pavaizduoti naudotoją dominančio reikalavimo visus egzistuojančius sąryšius pasirinktame projekte. Reikalavimų vizualizavimas šiuo metodu yra paprastesnis ir vienu metu vykdomas tik vienam, vartotojo pasirinktam, reikalavimui ***R***:



14 pav. Netmap algoritmo schema

Realizuojant Netmap metodą reikalavimui ***R*** taikomas toks algoritmas (žr. 14 pav.):

1. Surenkami visi, nepriklausomai nuo sąryšio tipo, su ***R*** susiję reikalavimai, kur ***R*** yra aukštesnio lygmens arba žemesnio lygmens reikalavimas, taip sudarant du atskirus sąrašus.
2. Iš duomenų bazės gaunami sąryšių atributai (tipas, aprašas, spalva).
3. Sukuriami pagrindiniai grafiniai elementai, reikalingi Netmap vizualicijai:
    a. sukuriamas fonas, atskiriantis Sunburst diagramą, nuo Netmap diagramos.

b. sukuriami diagramai reikalingi valdikliai (navigacijai tarp mazgų, detalesnei mazgo peržiūrai ir k.t.),
c. reikalavimui ***R*** įrankio centre nubraižoma figūra - pagrindinis mazgas.

4. Priklausomai nuo susijusių reikalavimų skaičiaus, turimas plotas virš pagrindinio mazgo, ir plotas po jo, yra padalijamas į lygias dalis, ir sudaromi tinklai turimiems sąrašams atvaizduoti:
    a. Apskaičiuojamas reikalingas kolonų skaičius tinkle
    b. Priklausomai nuo kolonų skaičiaus apskaičiuojamas reikalingų eilių skaičius tinkle
    c. Sudaromas tinklas, kurio dydis yra $K \; x \; E$, čia K – reikalingų kolonų skaičius, o E – reikalingų eilių skaičius
    d. Kiekvienam to lygmens reikalavimui:
        i. nubraižomas jį atvaizduojantis mazgas
        ii. nubraižomas sąryšis tarp jį atvaizduojančio ir pagrindinio diagramos mazgo
    e. galiausiai į tinklą įtraukiama „greito pridėjimo“ funkcija (grafinis elementas)

Sudarius abu, žemesnio ir aukštesnio lygmens, reikalavimų tinklus, Netmap diagrama yra paruošta naudojimui.

Detalesnė Netmap vizualizacijos algoritmo diagrama pateikta 4 priede.

III. KURIAMOS PROGRAMINĖS ĮRANGOS PROTOTIPO REALIZACIJA

3.1. Sunburst algoritmo realizacija

Sunburst vizualizacijos algoritmas vykdomas skirtinguose aplinkose (T-SQL serveris, .Net programavimo platforma) siekiant optimaliai išnaudoti turimas galimybes. Algoritmas inicijuojamas kai vartotojas pasirinka sistemoje egzistuojantį projektą ir norimą sąryšių tipą.

Sunburst diagramos braižymas inicijuojamas, kai yra tik nurodomas projekto kodas ir reikalavimų sąryšio tipas. Tai užtikrina žemiau pateiktas pseudo kodas:

```
IF project or relation type is not selected
 EXIT
END IF
READ Requirements of selected project by arranging them by selected type with
   provided search criteria
```

Pastaba. Čia ir žemiau naudojamas pseudo kodo standartas, pasiūlytas [13].

READ komanda - tai duomenų surinkimo etapo iniciavimas. Šis etapas vykdomas duomenų bazės lygmenyje: tokiu būdu įrankiui duomenys iškarto būna paruošti tolimesniam jų naudojimui - elementariu būdu surenkami nurodyto projekto reikalavimų meta duomenys ir iš karto apskaičiuojamas kiekvieno reikalavimo lygmuo bei susietų reikalavimų skaičius visuose lygiuose:

```
READ requirements` attributes
 FOR EACH requirement
  CALCULATE requirement`s tier and ones of inferior tier that depend on it
       based on its ID, Project, Relation Type and Search Criteria
 END FOR
RETURN collected data
```

Pateiktame pseudokode *CALCULATE* komanda įgyvendinama per funkciją, kurios paskirtis paskaičiuoti reikalavimo lygį (angl. *tier*) ir su juo susijusių žemesnių lygių reikalavimų skaičių. Jos (f-jos) veikimas yra nagrinėjimas žemiau.

```
MAKE list T with „parent“ requirement‘s ID and „child“ requirement‘s ID
 WHERE relation of given type has „parent“ and „child“ belongs to given
    project and also it matches search criteria
```

Čia pseudo kodo fragmente aprašomas pagal nurodytą sąryšio tipą susijusių reikalavimų sąrašas - reikalavimų ID iš sąryšių lentelės, kur aukštesnio lygmens ID priklauso skaičiuojamajam reikalavimui.

Sudarius minėtą sąrašą, toliau vykdoma:

```
MAKE list L from
 READ „parent" and „child" from list T
  WHERE „child" = given requirement
 AND
 READ „parent" and „child" from list T
  WHERE „child" = „parent" from list L
CALCULATE size of list L
```

Šios rekursijos pagalba, sudaroma medžio šaka, kurio viršūne laikomas reikalavimas R. Gautas medžio dydis laikomas reikalavimo R lygmeniu - lygmuo 0 reiškia, jog šis reikalavimas neturi už save „aukštesnių" reikalavimų nurodytu sąryšiu ir yra aukščiausio lygmens. Kuo didesnis lygmens indeksas, tuo reikalavimas Žiedinėje diagramoje bus labiau nutolęs nuo centro.

Turimas susijusių reikalavimų sąrašas taip pat yra panaudojamas ir skaičiuojant jo „dydį" būsimoje diagramoje.

```
MAKE list C from
 READ „parent" and „child" from list T
  WHERE „parent" = given requirement
 AND
 READ „parent" and „child" from list T
  WHERE „parent" = „child" from list C

CALCULATE size of list C
```

Aprašyta rekursinės funkcijos pagalba sudaromas medis, kurio viršūnė yra reikalavimas R. Viršūnių skaičius lems, kokį plotą užims reikalavimas R Sunburst diagramoje.

Apskaičiuotas reikalavimo lygmuo ir susijusių reikalavimų skaičius yra gražinami atgal, užbaigiant pradinį duomenų surinkimą – taip užbaigiama *READ* komanda.

```
IF failed to read
 EXIT
END IF
IF requirements > 0
 FOR EACH requirement sorted by tier asc
  IF requirement is first of its tier
   CREATE new array for its tier
  END IF
  ADD requirement to array of its tier
 END FOR
END IF
```

Sunburst algoritmas yra nutraukiamas jei dėl nenumatytų priežasčių nepavyksta nuskaityti duomenų arba jei jų paprasčiausiai nėra. Jei duomenys surinkti sėkmingai - Gauti duomenys iš minėtos procedūros yra išskirstomi pagal jų apskaičiuota lygmenį į duomenų masyvus, kur kiekvienas masyvas yra atitikmuo Sunburst diagramos žiedui. Pateiktame pseudokode galima pastebėti, jog reikalavimai, prieš juos išskirstant, yra išrikiuojami pagal jų lygmenį - nuo aukščiausio (0) iki žemiausio (*n*) - taip siekiama optimizuoti duomenų masyvus tolimesniems veiksmams. Jei buvo naudojama paieška atrinkti reikalavimams, papildomai yra atliekami veiksmai žemiau pateiktame pseudokode.

```
IF search criteria was applied
 IF created tiers’ arrays > 1
  FOR EACH tiers’ array
   IF tier is highest
    FOR EACH requirement in array
     SET set requirement’s tier to highest
    END FOR
   ELSE
    FOR EACH requirement in array
     IF superior tier is empty
      MOVE requirement to superior tier
     ELSE
      SET requirement’s tier to be inferior than its "parent’s" tier
     END IF
    END FOR
   END IF
  END FOR
 ELSE
  FOR EACH requirement in array
   SET requirement’s tier to highest
  END FOR
 END IF
END IF
```

Jei atrinkti duomenys sudarė tik vieną masyvą, duomenų pertvarkyti nereikia ir minėti papildomi veiksmai yra praleidžiami – tėra įsitikinama, kad tas masyvas būtų arčiausiai centro. Priešingu atveju, tikrinamas kiekvienas masyvas ir ieškoma aukščiausio lygmens masyvo. Jį radus jo elementai bei jis pats yra „perkeliami“ į lygį, kurio indeksas yra *0* - šio masyvo elementai bus arčiausiai centro. Visiems likusiems masyvams yra taikoma tokia logika: kiekvienam jo reikalavimui yra tikrinama ar per vieną lygmenį aukštesniame masyve egzistuoja susijęs reikalavimas - jei neegzistuoja, tai reikalavimas perkeliamas į aukštesnį lygmenį, priešingu atveju – įsitikinama, kad tarp tų reikalavimų nebūtų „tarpo“ t. y. reikalavimas perkeliamas į vienu lygmeniu žemesnį masyvą negu yra jo susijusio (aukštesnio) reikalavimo lygis.

```
IF tier of the ring is highest (ring is closest to center)
 FOR EACH requirement in tier
  CALCULATE width of polygon`s inner arc in radians
  CALCULATE starting position of polygon`s inner arc
  IF (starting position of polygon`s inner arc + calculated width of
    polygon`s inner arc) > 2*π
   CALCULATE width of polygon`s inner arc again with constraint
  END IF
  WHILE width of drawn arc < width of polygon`s inner arc
   CALCULATE a point to extent drawn arc
  END WHILE
  IF width of drawn arc < width of polygon`s inner arc
   CALCULATE extra point to extent drawn arc to match polygon`s inner arc
  END IF
  FOR EACH point of drawn inner arc starting from last one
   CALCULATE a point of outer arc
  END FOR
   CALCULATE used space of Sunburst's current ring
  DRAW polygon
 END FOR
```

Aukščiau pateiktu pseudo kodo yra aprašytas procesas, kuris vykdomas su masyvu, atitinkančiu žiedą, esančio arčiausiai diagramos centro: paskaičiuojamas kiekvienam masyvo elementui skirto diagramos artefakto plotis. Tada randama jau figūromis užimto ploto „pabaiga“ skritulyje – ties ja prasidės naujas artefaktas. vienas iš būdų kaip tai padaryti - sumuoti prieš tai nubrėžtų figūrų pločius. Brėžiant pirmąjį žiedą labai svarbu, kad paskutinis daugiakampis nepersidengtų su pirmuoju, nes jiems persidengus žemesnių lygmenų reikalavimai gali pradėti vienas kitą slėpti, „išeiti“ iš jiems galimo ploto ir pan. Dėl to prieš pradedant skaičiuoti vidinio lanko viršūnių koordinates reikia patikrinti, ar visų tą žiedą sudarančių figūrų pločių sudedamoji nėra didesnė nei 2π. Jei yra – perskaičiuojamas paskutinės figūros plotis – iš 2π atimama jau nubrėžtų figūrų pločių suma.

Kitų lygmenų masyvams yra taikoma šiek tiek kitokia žingsnių seka:

```
ELSE
 FOR EACH requirement in tier
  FIND "parent" of current requirement
  CALCULATE "parent" width
  CALCULATE width of polygon`s inner arc in radians
  CALCULATE starting position of polygon`s inner arc
  WHILE width of drawn arc < width of polygon`s inner arc
   CALCULATE a point to extent drawn arc
  END WHILE
  IF width of drawn arc < width of polygon`s inner arc
   CALCULATE extra point to extent drawn arc to match polygon`s inner arc
  END IF
  FOR EACH point of drawn inner arc starting from last one
   CALCULATE a point of outer arc
  END FOR
  CALCULATE used space of Sunburst's current ring
  DRAW polygon
 END FOR
END IF
```

Kiekvienam to masyvo elementui pirmiausia surandamas jo „tėvinis“ elementas aukštesnio lygmens žiede. Priešingai nei aukščiausio lygmens žiede, artefakto plotas priklausys ne tik nuo jau „užimto“ ploto bet ir nuo to, kokio ploto yra jo „tėvinis“ reikalavimas.

Nepriklausomai nuo žiedo lygmens, diagramos artefaktai yra konstruojami vienodai.

3.1.1. Diagramos artefakto formavimas

Diagramos artefaktas (daugiakampis) yra pradedamas formuoti nuo jo vidinio lanko kairiausios viršūnės (taško). Jų skaičius priklauso nuo to, kokiu žingsniu jos yra dedamos, ir nuo to, kokį plotį daugiakampis turės užimti. Kiekvienos vidinio lanko viršūnės pozicija randama žemiau pateiktomis trigonometrinėmis lygtimis, išskyrus paskutinę - gali būti taip jog paskutinė ji tokiu būdu atsidurs už numatyto ploto ribų, tad ji yra skaičiuojama atskirai, užtikrinant, kad taip nenutiktų.

$$CALCULATE\ X = x_0+(r + h*t)*Cos(\alpha)$$
$$CALCULATE\ Y = Y_0+(r + h*t)*Sin(\alpha)$$

<u>Pastaba</u>: pateiktos šios ir kitos matematinės išraiškos yra detalizuojamos sekančiame poskyryje.

Sudarius vidinį daugiakampio lanką, toliau formuojamas išorinis lankas: kiekvienai, pradedant nuo paskiausios viršūnės vidiniame lanke, apskaičiuojama išorinio lanko viršūnės pozicija tam tikru atstumu, tiesėje, einančioje per diagramos centrą ir minėtą vidinę viršūnę.

```
CALCULATE radius R from center to outer arc
CALCULATE distance h between center of Sunburst and point of inner arc
CALCULATE X = X0 + R / H * (vᵢx - x0)
CALCULATE Y = y0 + R / H * (vᵢy - y0)
```

Galiausiai apskaičiavus visas daugiakampio viršūnes, belieka nubrėžti pačią figūrą, ją užpildyti atitinkama spalvą ir pažymėti. Dėl pasirinktos .Net programavimo aplinkos, sudarytą daugiakampį galima užpildyti iškart, tačiau reikalingi papildomi veiksmai, norint jam suteikti tekstinį identifikavimą. Minėtas tekstas, siekiant estetinio vaizdo, turėtų būti nubrėžtos figūros centre ir būti pasuktas atitinkamu kampu.

```
FIND coordinates of the middle of inner arc as p1
FIND coordinates of the middle of outer arc as p2
CALCULATE X =(p₁x+p₂x)/2
CALCULATE Y =(p₁y+p₂y)/2
```

Čia pseudo kodo fragmente yra apskaičiuojama daugiakampio centro koordinatės randant vidinio ir išorinio lankų vidurio taškus ir įstatant jų atitinkamas koordinačių reikšmes į kode pateiktas lygtis, kur p_1 – vidinio lanko vidurio taškas, o p_2 – išorinio lanko vidurio taškas.

Priklausomai nuo ketvirčio, kuriame yra daugiakampio centro taškas (pati figūra gali užimti ne vieną ketvirtį), yra skaičiuojamas arksinusas (esant taškui II ir IV ketvirčiuose) arba arkkosinusas (I ir III ketvirčiuose). Kadangi .Net bibliotekos trigonometrinės funkcijos gražina dydžius radianais, o teksto pasukimas nurodomas laipsniais, reikia kampų reikšmes konvertuoti, ir jei taškas I ir III ketvirčiuose, papildomai pasukti tekstą 90° kampu pagal laikrodžio rodyklę, taip išlaikant tinkamą teksto orientaciją.

```
CALCULATE distance d between the point of center of the diagram and the
      calculated point
IF calculated point is in II or IV quarter
 CALCULATE Arcsin(Abs(y-y₀)/d) * 180 / π
ELSE
 CALCULATE (Arccos(Abs(y-y₀)/d) * 180 / π) - 90
END IF
```

3.1.2. Diagramos artefakto braižymas

Sunburst diagrama yra skritulys, tad bet kuris taškas, esantis jo užimame plote, gali būti apskaičiuotas naudojant šias parametrizuotas trigonometrines funkcijas:

$$x_i = x_0 + (r + ht)cos\alpha_i \quad (5),$$

$$y_i = y_0 + (r + ht)sin\alpha_i \quad (6),$$

čia x_i ir y_i yra skaičiuojamosios viršūnės v_i koordinatės; x_0 ir y_0– Sunburst diagramos centro koordinatės; r – atstumas nuo diagramos centro iki pirmojo žiedo vidinės kraštinės t. y. vidinio apskritimo spindulys; h - daugiakampio aukštinė; t - žiedo lygmuo, o α_i – kampas tarp teigiamos x ašies ir taško, kuriame turėtų būti brėžiama viršūnė v_i. Minėtas taškas apskaičiuojamas prieš tai, prie viršūnės v_{i-1} taško pridėjus žingsnį, kuriuo yra dedamos viršūnės.

Pavyzdžiui, jei laikysime kad diagramos centras yra taške (0; 0) ir skaičiuotume aukščiausio lygmens žiede (*t = 0*), pirmos figūros viršūnių koordinatės, su sąlyga, kad jos plotis yra $\frac{\pi}{2}$ rad, o vidinis spindulys lygus 25, tai:

1. Kadangi ši figūra pirma žiede, tai jos pradinė viršūnė v_1 yra ant teigiamos x ašies ir nutolusi nuo centro per 25 vienetus t. y. jos koordinatės yra (25; 0).
2. Žingsnis, kuriuo dedami taškai lemia, kaip „apvaliai“ atrodys daugiakampis: 15a pav. matomas daugiakampis, kurio žingsnis yra 45° , o 15b pav. - 1°. Nors pavyzdžiui pasirinkta yra sąlygiškai ribinės reikšmės (žingsnis gali būti tiek mažesnis tiek didesnis nei šios dvi reikšmės), tačiau jos puikiai iliustruoja žingsnio dydžio svarbą.
3. Skaičiuojame viršūnių koordinates pasirinktu 45° žingsniu. Kadangi jau turime pirmą viršūnę, skaičiuojame sekančią viršūnę v_2:

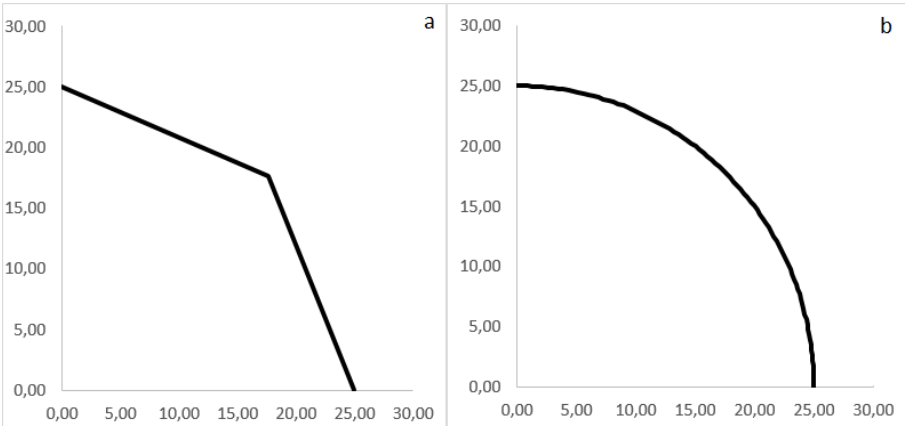
$$v_{2_x} = 0 + (25 + 50 \cdot 0) \cdot cos\frac{\pi}{4} = 25\frac{\sqrt{2}}{2} \approx 17.68$$

$$v_{2_y} = 0 + (25 + 50 \cdot 0) \cdot sin\frac{\pi}{4} = 25\frac{\sqrt{2}}{2} \approx 17.68$$

4. Trečia, ir paskutinė viršūnė v_3 yra kampu $\frac{\pi}{2}$ nuo pradinės viršūnės, tad jos koordinatės bus:

$$v_{3_x} = 0 + (25 + 50 \cdot 0) \cdot cos\frac{\pi}{2} = 0$$

$$v_{3_y} = 0 + (25 + 50 \cdot 0) \cdot sin\frac{\pi}{2} = 25$$



15 pav. Daugiakampio vidinis lankai su pasirinktų dydžių žingsniais

Pateiktame paveiksle aiškiai matyti skirtumas tarp naudojamo žingsnio: kairėje jis per didelis tad daugiakampiai tampa „kampuoti“, ko pabaigoje – Sunburst diagrama visiškai nebus panaši į skritulinę diagramą; dešinėje, kai taikytas žingsnis yra lygus 1° , daugiakampio vidinis lankas tampa apvalainas – tai yra tai ko būtent ir yra siekiama.

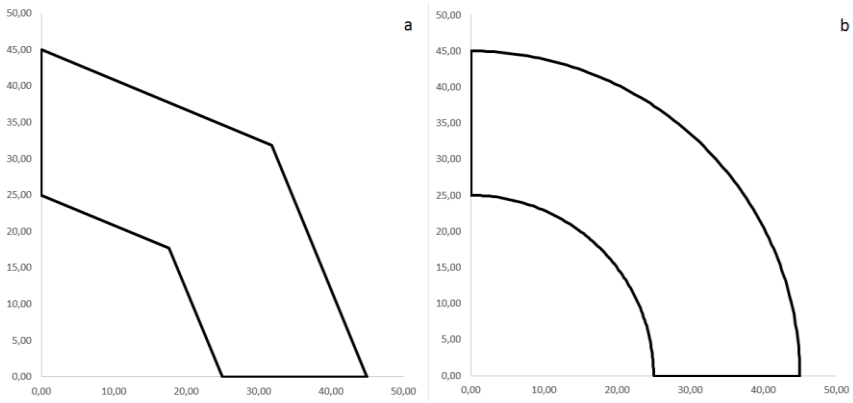
Apskaičiavus visas brėžiamo daugiakampio vidinio lanko viršūnes, inicijuojamas išorinio lanko viršūnių pozicijų skaičiavimas – kiekvienai viršūnei v_i jos pozicija randama formulėmis:

$$x_j = x_0 + \frac{(r+h)}{H} \cdot (x_i - x_0) \quad (7),$$

$$y_j = y_0 + \frac{(r+h)}{H} \cdot (y_i - x_0) \quad (8),$$

čia x_j ir y_j yra skaičiuojamosios viršūnės v_j koordinatės; r - pagrindinio mazgo spindulys; h – daugiakampio aukštinė; H - atstumas tarp diagramos centro taško ir taško v_i.

Apskaičiavus šias viršūnes $\{(45;\ 0), (31.82;\ 31.82), (0;\ 45)\}$, galima brėžti daugiakampį. 16a pav. vaizduojamas daugiakampis, kuris buvo braižomas su 45° žingsniu, o 16b pav. – su 1° žingsniu.



16 pav. Nubrėžti daugiakampiai su pasirinktų dydžių žingsniais

Užbaigus diagramos artefakto išorinio lanko viršūnių braižymą, iš esmės turime jau sukonstruotą diagramą, tačiau siekiant, jog diagrama būtų intuityvesnė, jos elementus dar reikia pažymėti reikalavimų kodais:

1. apskaičiuojamas teksto centro taškas lygtimis

$$x = \frac{x_1+x_2}{2} \quad (9),$$

$$y = \frac{y_1+y_2}{2} \quad (10),$$

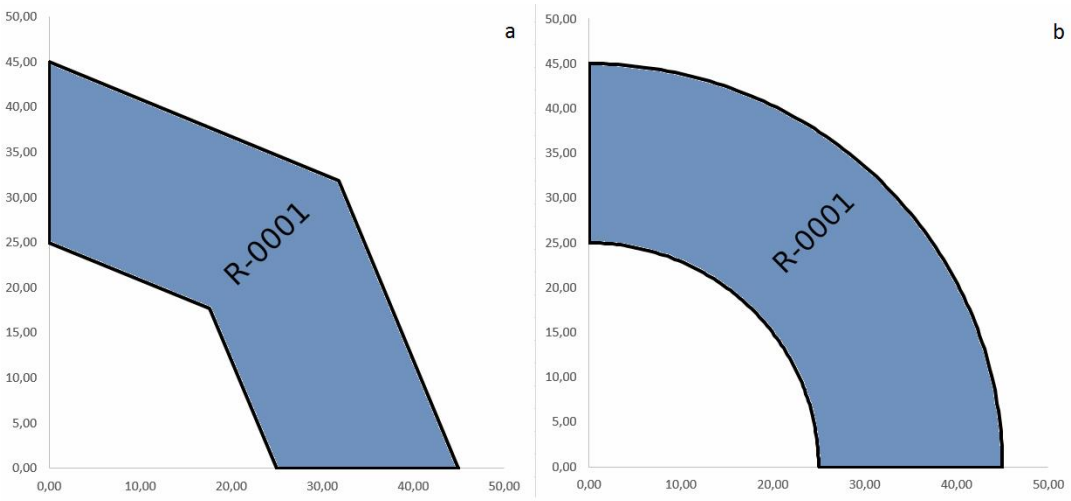
Čia $(x_1; y_1)$ *daugiakampio vidinės lanko centro koordinatės, o* $(x_2; y_2)$ *– išorinio lanko centro koordinatės.*

$$x = \frac{17.68 + 31.82}{2} = 24.75$$

$$y = \frac{17.68 + 31.82}{2} = 24.75$$

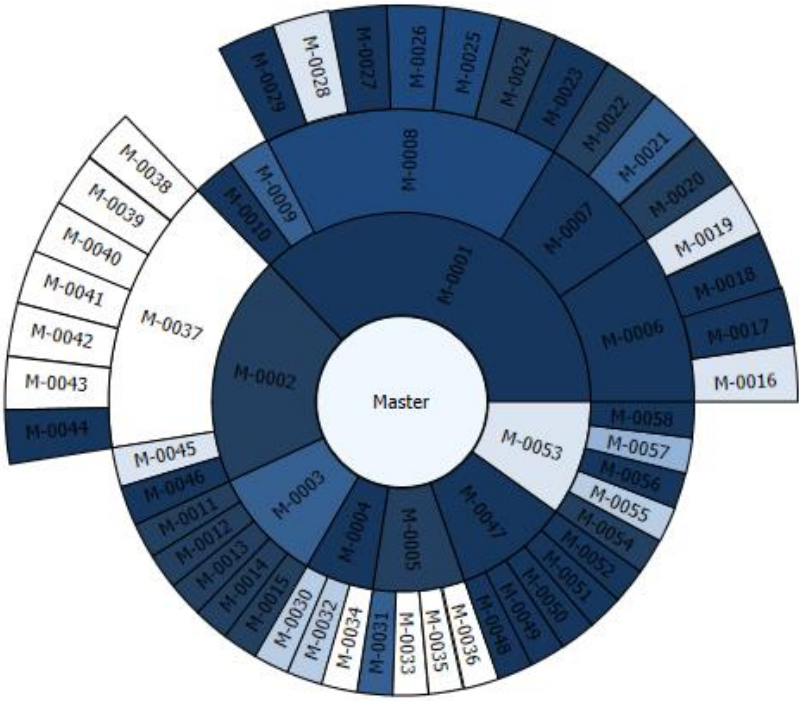
2. Apskaičiuojamas kampas, kuriuo reiks pasukti tekstą – kadangi detalizuojamas artefaktas yra I ketvirtyje, tai taikoma formulė yra (žr. 3.1.1 poskyryje pateiktus pseudo kodus):

$$\cos^{-1}\left(\frac{24.75}{\sqrt{2 \cdot (24.75 - 0)^2}}\right) \cdot \frac{180}{\pi} - 90 = -45\,°$$



17 pav. Užbaigti daugiakampiai su pasirinktų dydžių žingsniais

17 pav. pateikta, kaip galimai atrodys (a pav.- 45° , b pav.– 1° žingsniu) pilnai nubraižytas Sunburst diagramos artefaktas, kuris reprezentuoja programinės įrangos reikalavimą. Suprantama, diagrama bus sudaroma iš daugybės tokių elementų, tad pavyzdys, kaip atrodytų visa sudaryta diagrama, pateikiamas žemiau:



18 pav. Pilnai nubraižyta Sunburst diagrama

Pateiktame 18 pav. matoma užbaigta Sunburst diagrama, kuri atvaizduoja *ExcelSafe* programinės įrangos reikalavimus [33]. Diagramą sudaro jos centre esantis skritulys su projekto kodu ir trys žiedai, reprezentuojantys hierarchinę struktūrą tarp reikalavimų. Diagrama sudaryta naudojant dekompozicijos tipo sąryšį ir spalvų paletę, skirtą reikalavimo kategorijos atributui.

3.2. Netmap algoritmo realizacija

Netmap vizualizacijos algoritmas yra inicijuojamas kai naudotojas norėdamas pamatyti jį dominančio reikalavimo visus sąryšius du kartus pelės klavišu spragteli ant Sunburst diagramos elemento.

Pirmiausia, iš duomenų bazės yra surenkami duomenys: susisijusių reikalavimų dveji sąrašai, kur pirmajame pasirinktas reikalavimas yra sąryšio aukštesnio lygmens reikalavimas, o antrajame – žemesnio lygmens. Taip pat iš duomenų bazės yra nuskaitomi ir visų sąryšių atributai – tipas, aprašas, reprezentuojanti spalva.

Žemiau pateiktame pseudokode matyti, kad pagrindiniai vizualizacijos elementai yra kuriami tik egzistuojant tam tikrai sąlygai – kai vizualizacija yra inicijuojama. Kitais atvejais, pavyzdžiui, atliekant navigaciją tarp mazgų, pakeitus įrankio lango dydį ir t. t. šie elementai neturi būti piešiami, taip išvengiant dubliuotų elementų atsiradimo.

```
READ lists of surrounding tiers requirements
READ relations types
IF mandatory elemenents are not created or require redrawing
 DRAW mandatory elements
END IF
```

Privalomi Netmap vizualizacijos elementai – jos centre esantis pagrindinis mazgas ir šios vizualizacijos algoritmui įtakos neturintys elementai „fonas“ ir „uždarymo mygtukas“. Pastarieji du laikomi privalomais (esminiais) dėl to, jog pirmasis skirtas atskirti abejas vizualizacijas, o antrasis – grįžti atgal prie Sunburst diagramos.

Žemesnio lygmens mazgai, talpinami po pagrindiniu mazgu ir aukštesnio lygmens mazgai, talpinami virš pagrindinio mazgo, nors ir yra svarbūs pačiam algoritmui, tačiau nelaikomi privalomais vizualizacijoje, nes atitinkamai reikalavimų sąrašai gali būti tušti.

```
ADD transperant background
ADD close button
ADD main node that represent focused requirement
CREATE net of higher tier related requirements
CREATE net of lower tier related requirements
```

Minėtiems reikalavimų sąrašams yra konstruojami tinkleliai, kuriuose bus vaizduojami mazgai reprezentuojantys sąrašo elementus:

```
CALCULATE how many columns will be required
CALCULATE how many rows will be required
ADD empty grid
FOR EACH required column
 ADD column to grid
END FOR
FOR EACH required row
 ADD row to grid
END FOR
FOR EACH related requirement
 ADD relation between main node and current requirement
 ADD node for current requirement
 ADD node to grid
```

Sudarius minėtus tinklelius (žr. 3.2.1 poskyrį) jų langeliai yra užpildomi stačiakampiais, reprezentuojančiais reikalavimus tame sąraše ir vizualizuojami ryšiai tarp mazgų. Toks tinklelių pildymas vykdomas nuo viršutinio, kairiausiai esančio langelio, prioretizuojant eilutės užpildymą.

Kai eilutė yra pilnai sudaryta, reikalingas „perėjimas“ į kitą eilutę, jog būtų išlaikyta norima grafinė struktūra:

```
 IF current row is filled
  BEGIN adding to next row
 ELSE
  CONTINUE adding to current row
 END IF
END FOR
```

Galiausiai, kai visi sąraše esantys reikalavimai yra apdoroti, pridedamas grafinis elementas, kurio paskirtis – rezervuoti vietą galimai naujam sąryšiui (jei naudotojas norėtų tokį sudaryt diagramoje, reikalavimo sąryšių peržiūros metu).

```
ADD placeholder for new node
IF current layer belongs higher tier requirements
 ADD layer to the top of the screen
ELSE
 ADD layer to the bottom of the screen
END IF
```

3.2.1. Mazgų tinklo formavimas

Kaip minėta, kiekvienam egzistuojančiam reikalavimų sąrašui (jų maksimaliai gali būti tik du) yra kuriamas tinklelis, kurio dydis priklauso nuo to kiek reikalavimų yra tame sąraše, plius viena pozicija, skirta naujo sąryšio sudarymui.

Sudaromojo tinklelio kolonų skaičius apskaičiuojamas lygtimi:

$$[c] = \frac{w-d}{l+d} \quad (11),$$

kur w *– vizualizacijai skirtas plotis,* d *– atstumas tarp mazgų,* l *– mazgo plotis, o* $[c] = max\left\{n \in Z, n \leq \frac{w-d}{l+d}\right\}$. *Jei* $c > m+1, tai\ c = m+1$, m *- sąraše esančių reikalavimų skaičius*.

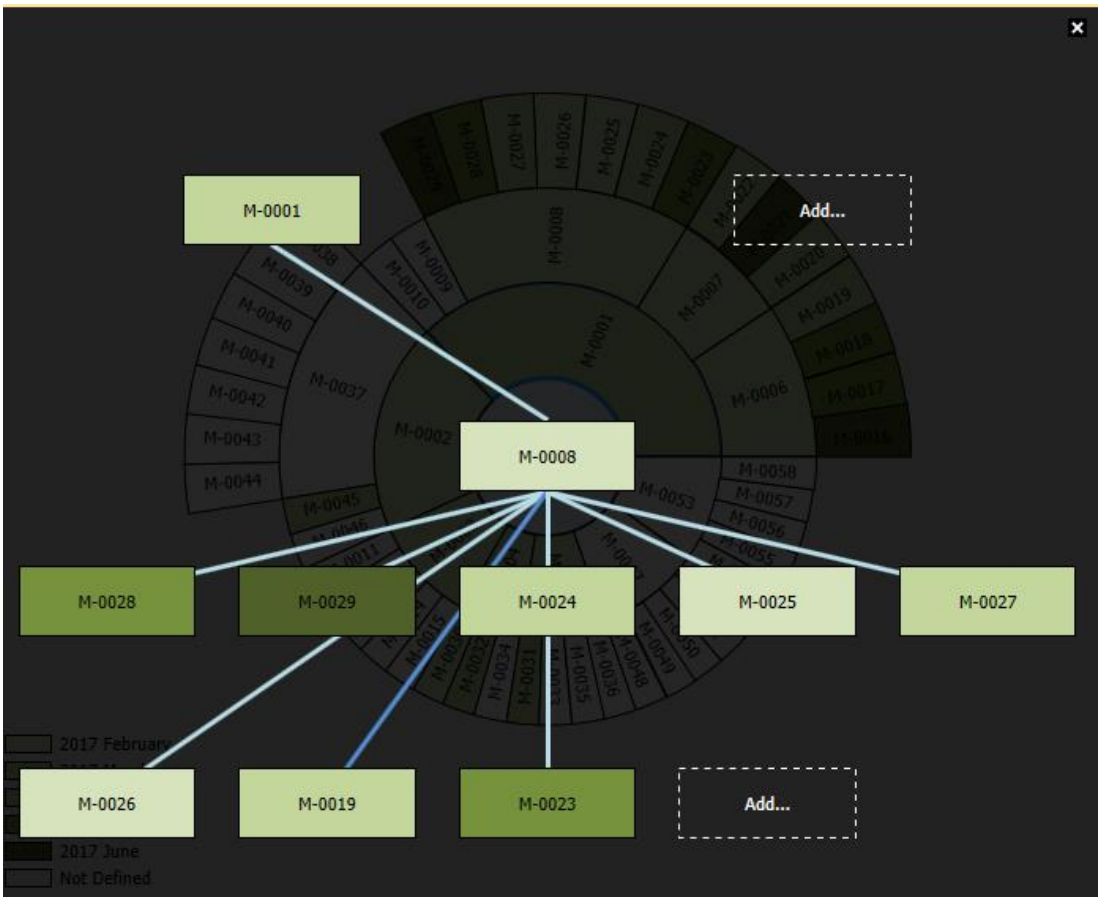
Sudaromojo tinklelio eilių skaičius priklauso nuo apskaičiuoto reikiamų kolonų skaičiaus ir yra lygus:

$$[r] = \frac{m+1}{c} \quad (12),$$

kur $[r] = min\left\{n \in Z, n \geq \frac{m+1}{c}\right\}$.

Apskaičiavus tinklelio dydį, jis užpildomas mazgais (šis procesas aprašytas prieš tai pateiktuose pseudo kodo fragmentuose).

Kai abu reikalavimų sąrašai yra perkelti į tinklelius, Netmap diagramos braižymas užbaigiamas ir gaunamas, pavyzdžiui, toks vaizdas:



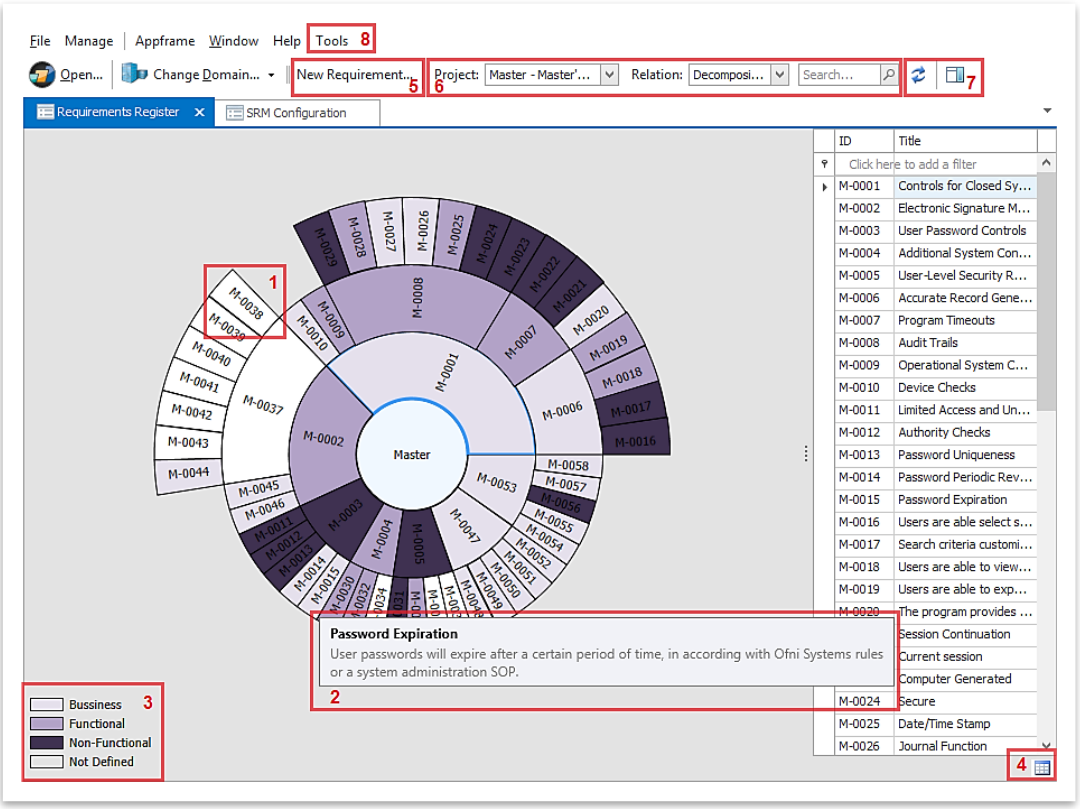
19 pav. Netmap diagrama

Netmap diagrama (19 pav.) sudaryta iš šių elementų:

1. Pagrindinis mazgas, reprezentuojantis naudotojo Sunburst diagramoje pasirinktą reikalavimą.
2. Mazgas, reprezentuojantis aukštesnio lygmens reikalavimą.
3. Mazgas, reprezentuojantis žemesnio lygmens reikalavimą.
4. Sąryšis tarp reikalavimų.
5. Rezervuotos vietos galimiems naujiems sąryšiams tarp pasirinkto ir kitų projekto reikalavimų.
6. Netmap vizualizacijos fonas, skirtas atskirti diagramas.
7. Diagramos uždarymo mygtukas.

3.3. Grafinės naudotojo sąsajos elementai

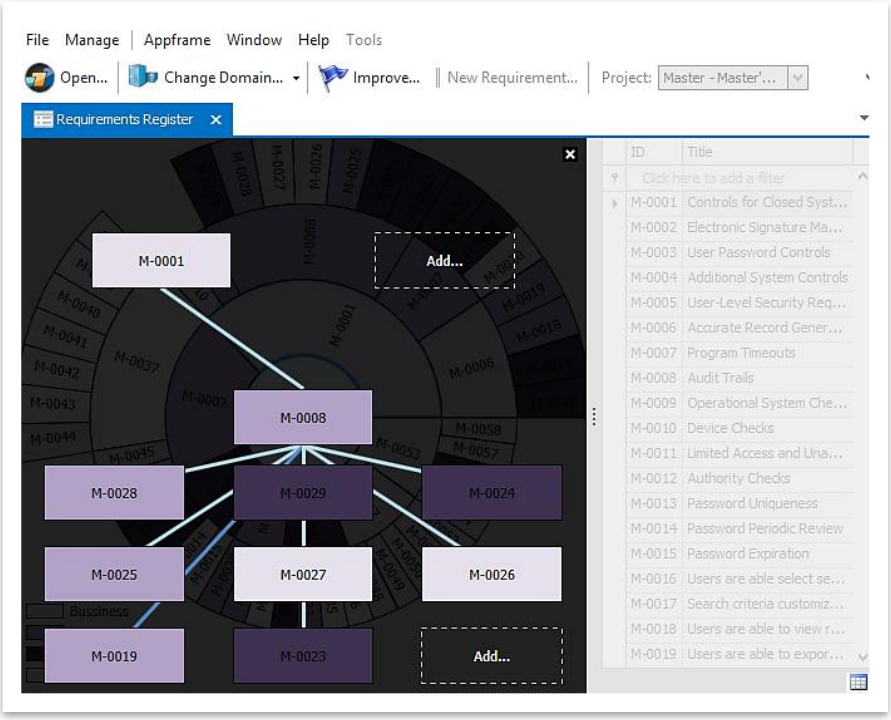
Sukurtas programinės įrangos prototipas turi grafinę vartotojo sąsają, valdomą įvykiais (angl. *event-driven user interface*) – tai būtinas sistemos elementas, siekiant jog sudaromos Sunburst ir Netmap diagramos būtų interaktyvios o pati sistema - efektyvi. 20 pav. pateikiamas pagrindinis įrankio prototipo langas:



20 pav. Pagrindinis SRM prototipo grafinės naudotojo sąsajos langas

Pagrindinį grafinės naudotojo sąsajos langą sudaro šie elementai:

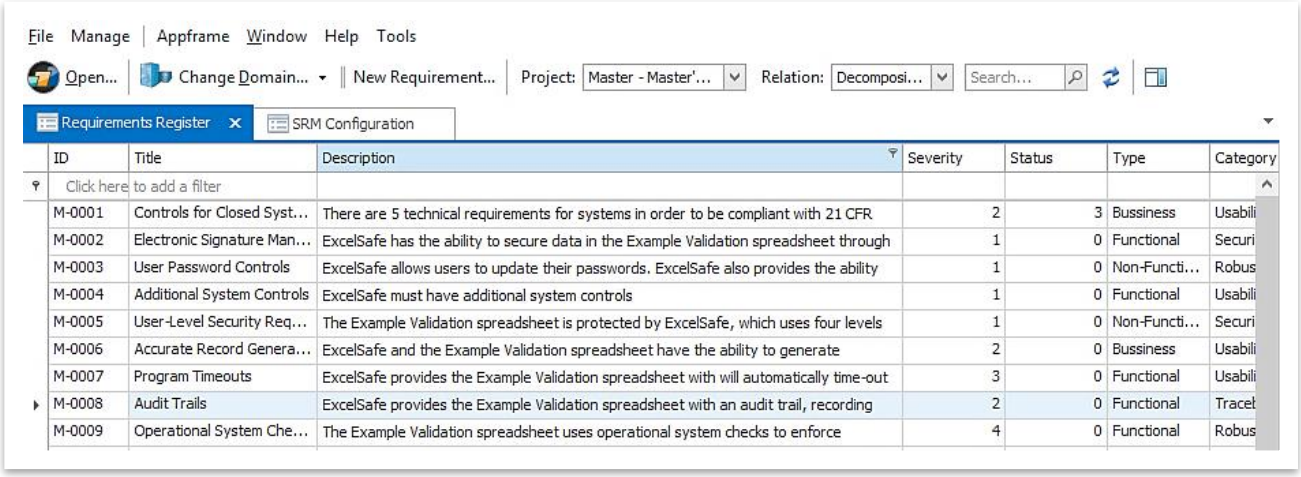
1. Reikalavimas. Šis Sunburst diagramos elementas atvaizduoja projekte esantį reikalavimą, pvz. reikalavimą „M-0038“. Naudotojui žymekliu du kartus spustelėjus jį, pateikiama Netmap diagrama (21 pav.), skirta tam reikalavimui.



21 pav. Pagrindinis įrankio prototipo langas Netmap diagramos peržiūros metu.

Kol Netmap diagrama yra aktyvi, tol visi įrankio valdikliai yra neaktyvūs – tai leidžia naudotojui suprasti, jog tol kol ši diagrama rodoma, visas dėmesys turi būti skiriamas būtent jai: naudotojas gali peržiūrėti tuo metu matomų reikalavimų charakteristikas, nagrinėti reikalavimų sąryšius jais sekdamas sujungtus mazgus ar sudarinėti naujus sąryšius.

2. Greita reikalavimo peržiūra. Naudotojui užvedus žymeklį virš vieno iš reikalavimų yra pateikiamas reikalavimo pavadinimas ir aprašas. Laukelis yra rodomas tol, kol žymeklis yra virš reikalavimo.
3. Legenda. Pateikiama informacija apie diagramoje naudojamą spalvų paletę.
4. Reikalavimų sąrašo peržiūra. Naudotojui spustelėjus šį mygtuką išskleidžiamas per visą ekraną dešinėje esantis reikalavimų sąrašas (22 pav.). Jei sąrašas jau išskleistas – jis gražinamas į pradinę padėtį. Kai sąrašas yra rodomas sutrauktu pavidalu, jame pasirinktus reikalavimą, jo atitikmuo yra pažymimas Sunburst diagramoje, kas leidžia greičiau jį aptikti pačioje diagramoje.

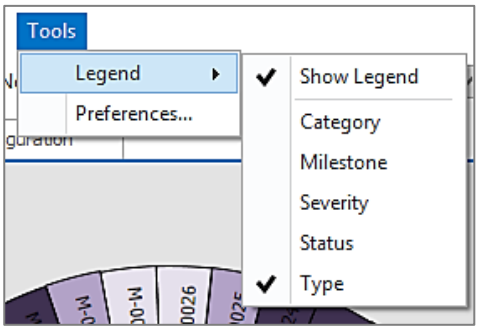


ID	Title	Description	Severity	Status	Type	Category
Click here to add a filter						
M-0001	Controls for Closed Syst...	There are 5 technical requirements for systems in order to be compliant with 21 CFR	2	3	Bussiness	Usabili
M-0002	Electronic Signature Man...	ExcelSafe has the ability to secure data in the Example Validation spreadsheet through	1	0	Functional	Securi
M-0003	User Password Controls	ExcelSafe allows users to update their passwords. ExcelSafe also provides the ability	1	0	Non-Functi...	Robus
M-0004	Additional System Controls	ExcelSafe must have additional system controls	1	0	Functional	Usabili
M-0005	User-Level Security Req...	The Example Validation spreadsheet is protected by ExcelSafe, which uses four levels	1	0	Non-Functi...	Securi
M-0006	Accurate Record Genera...	ExcelSafe and the Example Validation spreadsheet have the ability to generate	2	0	Bussiness	Usabili
M-0007	Program Timeouts	ExcelSafe provides the Example Validation spreadsheet with will automatically time-out	3	0	Functional	Usabili
M-0008	Audit Trails	ExcelSafe provides the Example Validation spreadsheet with an audit trail, recording	2	0	Functional	Traceb
M-0009	Operational System Che...	The Example Validation spreadsheet uses operational system checks to enforce	4	0	Functional	Robus

22 pav. Į projektą įtrauktų reikalavimų sąrašas

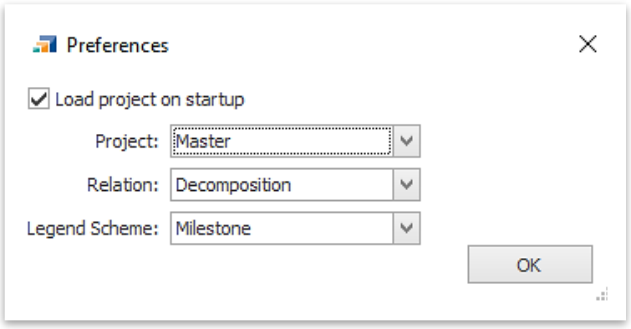
Šiame sąraše (22 pav.) yra pateikiami visi į projektą įtraukti reikalavimų ir jų atributai. Reikalavimų sąrašas nėra skirtas reikalavimų tarpusavio sąryšių vaizdavimui, jo paskirtis - informacinio pobūdžio: naudotojas turi galimybę filtruoti sąrašą pagal kiekvieną atributą kartu ar atskirai, juos redaguoti.

5. Naujo reikalavimo įtraukimas į sistemą. Naudotojui spustelėjus mygtuką, atidaromas dialogo langas, skirtas naujo reikalavimo užregistravimui (žr. skirsnį „Naujų reikalavimų įtraukimas į sistemą“).
6. Vaizduojamo projekto, reikalavimų sąryšio ir reikalavimų atrinkimo valdikliai. Jais manipuliuodamas naudotojas gali pasirinktį jį dominantį projektą, peržiūrėti reikalavimų sudaromas struktūras pagal pasirinktą ryšį ar atsirinkti jam aktualius reikalavimus.
7. Duomenų atnaujinimo (kairėje) ir reikalavimų sąrašo paslėpimo/rodymo (dešinėje) mygtukai.
8. Įrankių meniu. Jame pateikiamos galimybės pakeisti naudotojo parinktis įrankyje (23 pav.).



23 pav. Naudotojui pasiekiami įrankiai

„Legend“ įrankis leidžia keisti diagramoje naudojamą spalvų paletę pagal pasirinktą atributą, rodyti ar slėpti legendą lango kampe. „Preferences...“ įrankis atidaro dialogo langą (24 pav.):

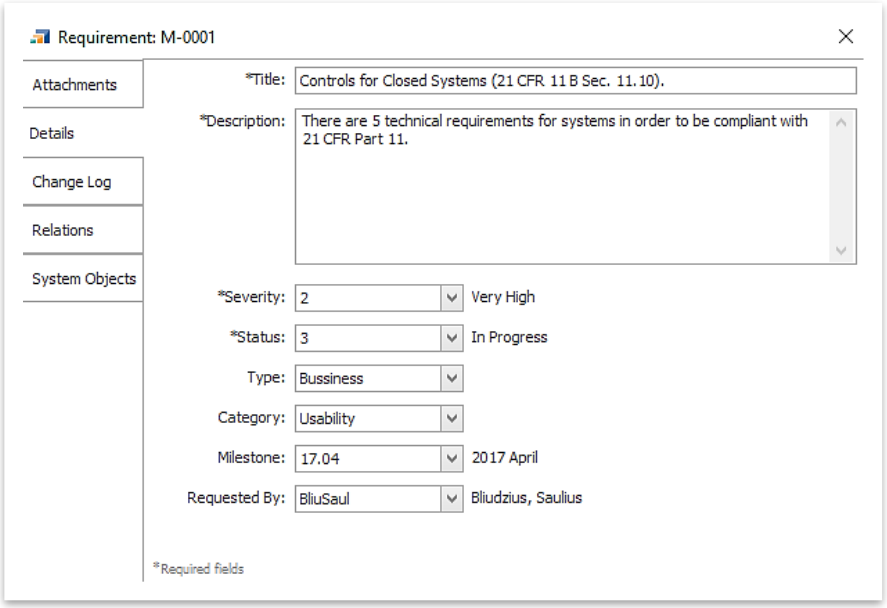


24 pav. Naudotojo parinktys įrankio atidarymui

Atsidariusiame dialogo lange naudotojas gali pasirinkti ar sistema turi sudaryti diagramą pagal nurodytą sąryšį, pritaikant pasirinktą legendą norimam projektui įrankio atidarymo metu.

3.4. Naujų reikalavimų įtraukimas į sistemą

Viena iš esminių įrankio funkcijų – užregistruoti naujus reikalavimus sistemoje. Siekiant užtikrinti, jog visa reikiama informaciją apie reikalavimą būtų surinkta, šis procesas vykdomas reikalavimo kūrimo dialoge, kur kūrimo etapai yra išskirstyti į atskiras korteles (25 pav.), taip susisteminant ir supaprastinant patį kūrimo procesą.

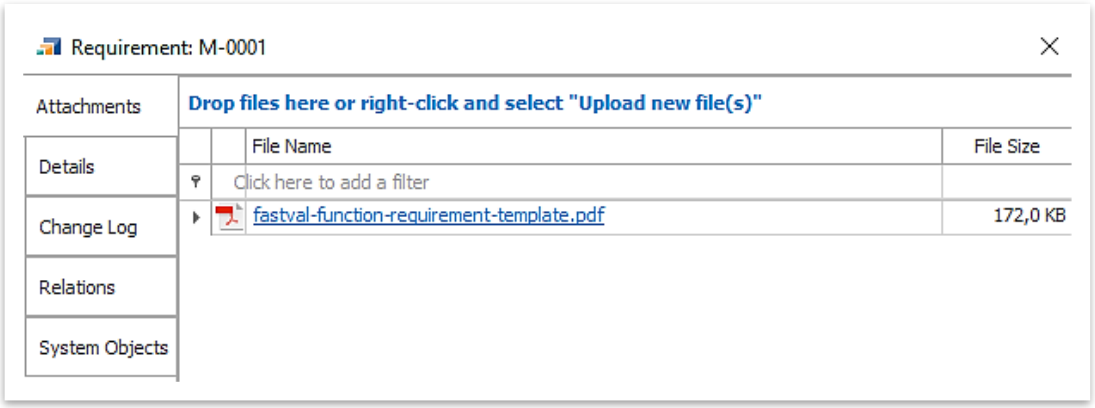


25 pav. Naujo reikalavimo registravimas

Pirmiausia, naudotojas privalo užpildyti būtinus naujai kuriamo reikalavimo atributus (25 pav.): pavadinimą ir aprašą. Pradinis reikalavimo svarbumas yra numatytas įrankiui keltais reikalavimais, tad jei naudotoją tenkiną parinkta reikšmė – jis gali ją palikti. Naujai registruojamas reikalavimas visada įtraukiamas su būsena „proposed“, tad šitos reikšmės naudotojas kūrimo metu pasirinkti negali.

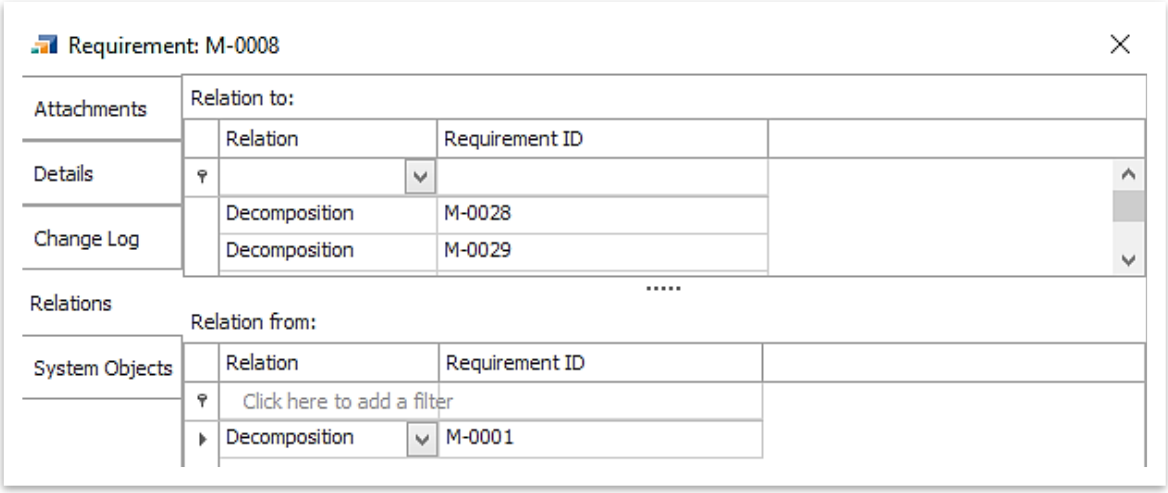
Užpildžius privalomus laukus, atsiranda „Create“ mygtukas – jį paspaudus yra sukuriamas reikalavimas, kurio ID rodomas dialogo pavadinime, o kitos kortelės tampa aktyvios:

- „Attachements“ kortelė (26 pav.). Joje naudotojas prie reikalavimo gali prisekti įvarius priedus: šaltinius, susijusius failus, pavyzdžius ir t.t.



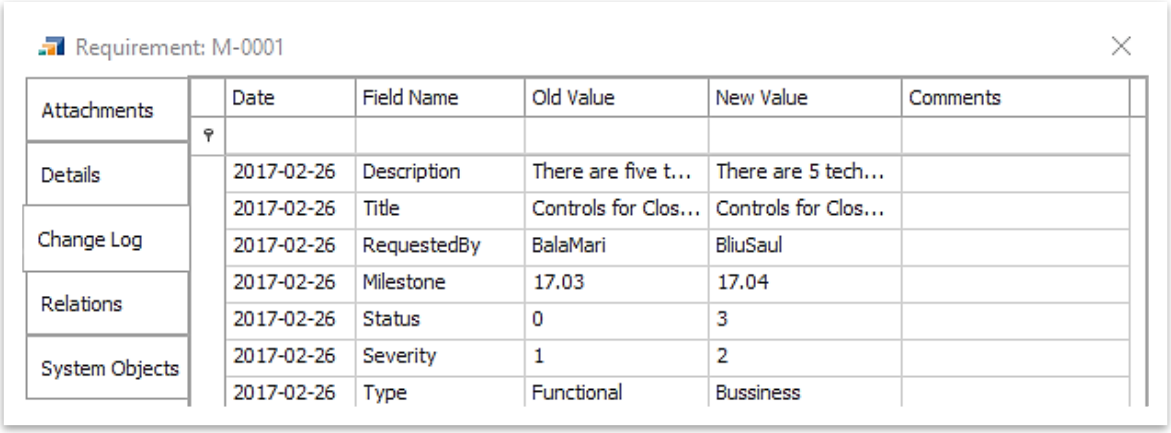
26 pav. Priedų kortelė

- „Relations“ kortelėje (27 pav.) naudotojas gali sudaryti ryšius naujai sukurtam reikalavimui.: tereikia į atitinkamą sąrašą įtraukti naują įrašą, pasirenkant sąryšio tipą iš duoto jų sąrašo, o po to – galimą reikalavimą (reikalavimai yra filtruoji priklausomai nuo jau egzistuojančių sąryšių ir šiuo metu pasirinkto sąryšio tipo).



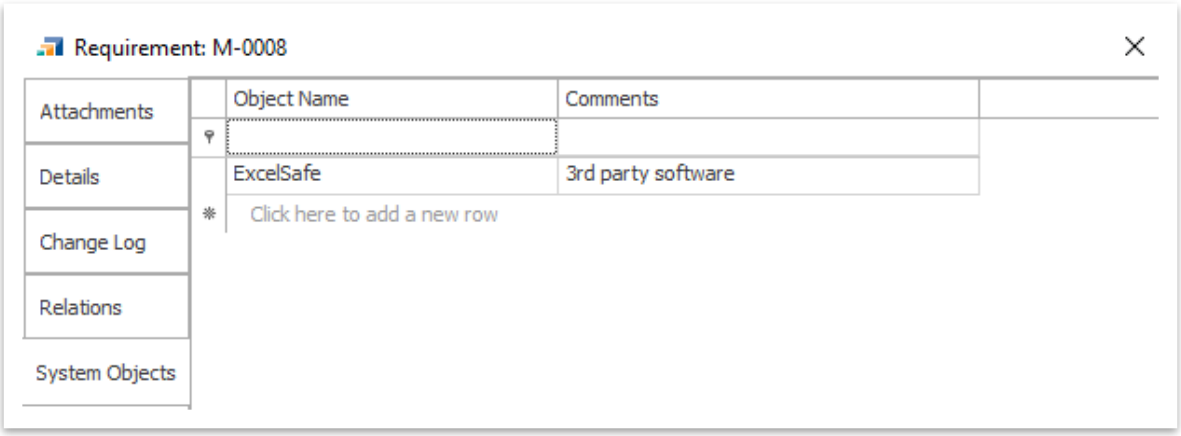
27 pav. Sąryšių kortelė

- „Change Log“ kortelė (28 pav.) leidžia peržiūrėti reikalavimo atributų pokyčius. Kaip minėta, tai labai svarbus reikalavimų valdymo aspektas, todėl pokyčiai turi būti sekami nuo to momento kai reikalavimas yra tik įtraukiamas į sistemą. Ši kortelė suteikia naudotojui galimybę prie pokyčio palikti komentarą ar pastabą bei filtrų pagalba rasti tam tikrus pakeitimus – tačiau jis pateiktų duomenų redaguoti negali.



28 pav. Reikalavimo atributų pokyčių kortelė

- „System Objects“ kortelėje (29 pav.) naudotojas gali nurodyti, kokie sistemos objektai, trečių šalių produktai ar kt. objektai vienaip ar kitaip yra susiję su šiuo reikalavimu. Čia objektai nėra pateikiami sąraše, tad naudotojas turi juos įvesti pats.



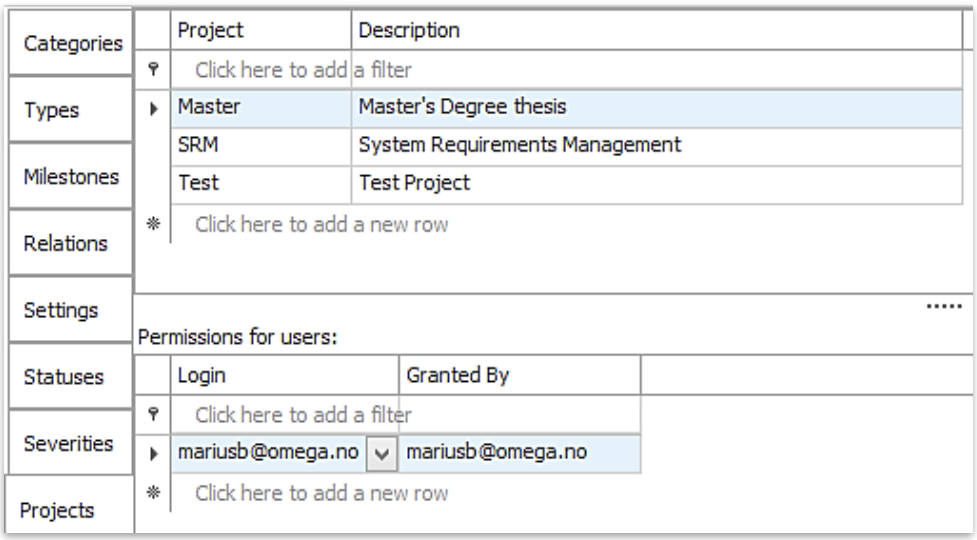
29 pav. Sistemos objektų kortelė

Tokios pat sudėties dialogas yra atidaromas kai naudotojas nori peržiūrėti jau sistemoje egzistuojantį reikalavimą detaliau

3.5. Programinės įrangos konfigūravimas

Programinės įrangos prototipas leidžia sistemoje vienu metu saugoti skirtingus projektus. Todėl atitinkamos rolės naudotojai privalo turėti galimybę įtraukti naujus projektus, valdyti ir konfigūruoti

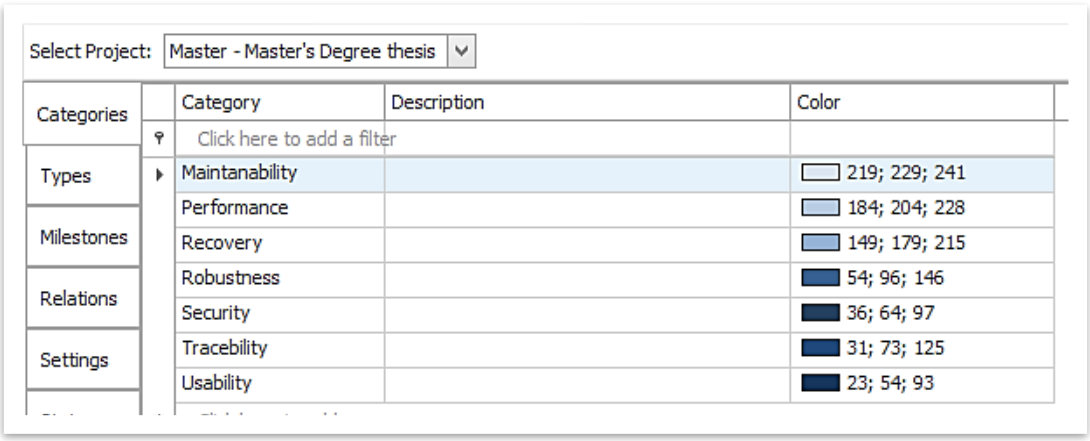
esamus: SRM administratoriai kiekvienam projektui gali paskirti atsakingą asmenį, nustatyti naudojamas reikšmes ar spalvų paletes.



30 pav. Projektų konfigūravimo kortelė

30 paveiksle matomas sistemoje egzistuojančių projektų sąrašas. Šis sąrašas vėliau yra matomas ir renkantis projektą pagrindiniame įrankio lange. Po minėtu sąrašu yra pateikiamas kitas sąrašas, kuriame rodomi naudotojai, atsakingi už projektą, pavyzdžiui, už projektą „Master“ yra atsakingas naudotojas, kurio prisijungimo vardas yra „mariusb@omega.no“.

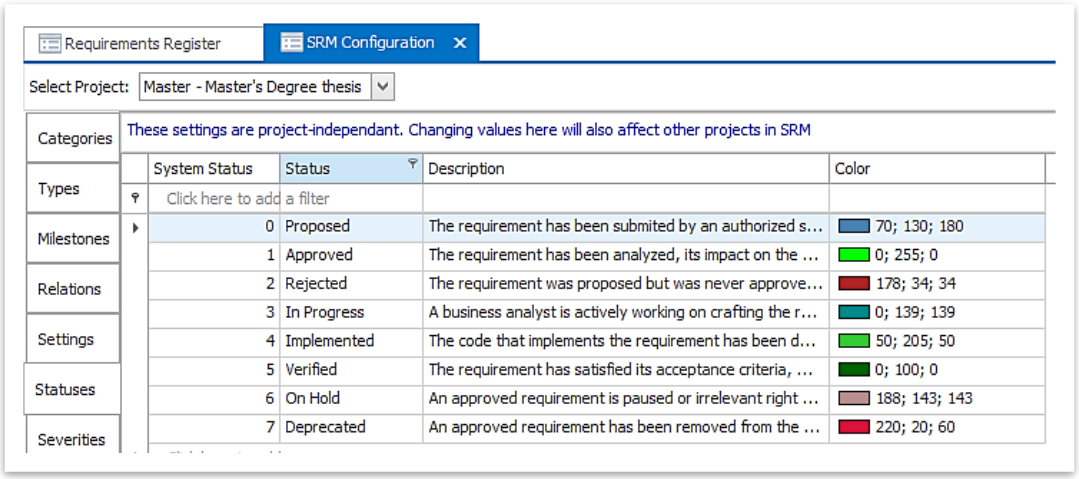
Naudotojui pasirinkus projektą iš sąrašo (31 pav.) galima jo konfigūracija:



31 pav. Projekto reikšmių konfigūravimas

Kiekvienam projektui atskirai priklausantys parametrai – reikalavimų kategorijos („Categories“) ir tipai („Types“), projekto etapai („Milestones“) ir bendri nustatymai („Settings“) yra išskirstyti į kortelės atitinkamai (31 pav.) . Jose, išskyrus „Settings“, pateikiami reikšmių sąrašai, kuriose saugoma reikšmė, aprašas ir spalva, reprezentuojanti minėtą reikšmę.

Sistemoje taip pat galima konfigūruoti reikšmes, kurios yra bendros visiems joje esantiems projektams: tam skirtos sąryšių („Relations“), būsenų („Statuses“) ir svarbumo („Severities“) kortelės:



32 pav. Sisteminių reikšmių konfigūravimas

Nors jų konfigūravimas yra identiškas kaip ir anksčiau minėtų, nuo projektų priklausomų, kortelių konfigūravimui, juose (32 pav.) naudotojas informuojamas, jog bet koks reikšmių pokytis šiose kortelėse gali turėti įtakos visiems projektams. Taip pat, priešingai nei nuo projekto priklausomos reikšmės, šios turi sisteminę reikšmę – dažnu atveju tai skaičius (indeksas) kuris duomenų bazėje naudojamas kaip pirminis raktas.

3.6. Sistemos programinės realizacijos ypatumai

Visi vizualizavimo algoritmai ir sistemos prototipas yra programiškai įgyvendinti *MS Visual Basic* .Net programavimo kalba, *MS Visual Studio 2015 Community* aplinkoje.

Sukurta reikalavimų valdymo sistema yra palaikoma „Appframe R4“ programavimo karkaso, naudojamo tarptautinės kompanijos Omega AS programinėms sistemoms kurti, teikti jų palaikymą ir naujinimus - tai reiškia, kad sistema atitinka vidinius standartus, užtikrina skirtingų rolių palaikomumą ir duomenų saugumą net iki duomenų bazės lentelės eilučių lygmens. Tačiau toks sistemos įdiegimas į didesnę sistemą reiškia, kad pastaroji turi būti įdiegta ir prieinama kompiuteryje, norint naudotis reikalavimų valdymo sistema. Taip pat, norint naudotis sistemą, privaloma turėti prisijungimo vardą į Pims sistemą

3.7. Sukurto sistemos prototipo testavimas ir verifikavimas

Sukurtos programinės įrangos sistemos prototipo testavimas buvo atliekamas naudojant realaus programinio įrankio *ExcelSafe* reikalavimus: šiam programiniam įrankiui kelti reikalavimai paimti iš

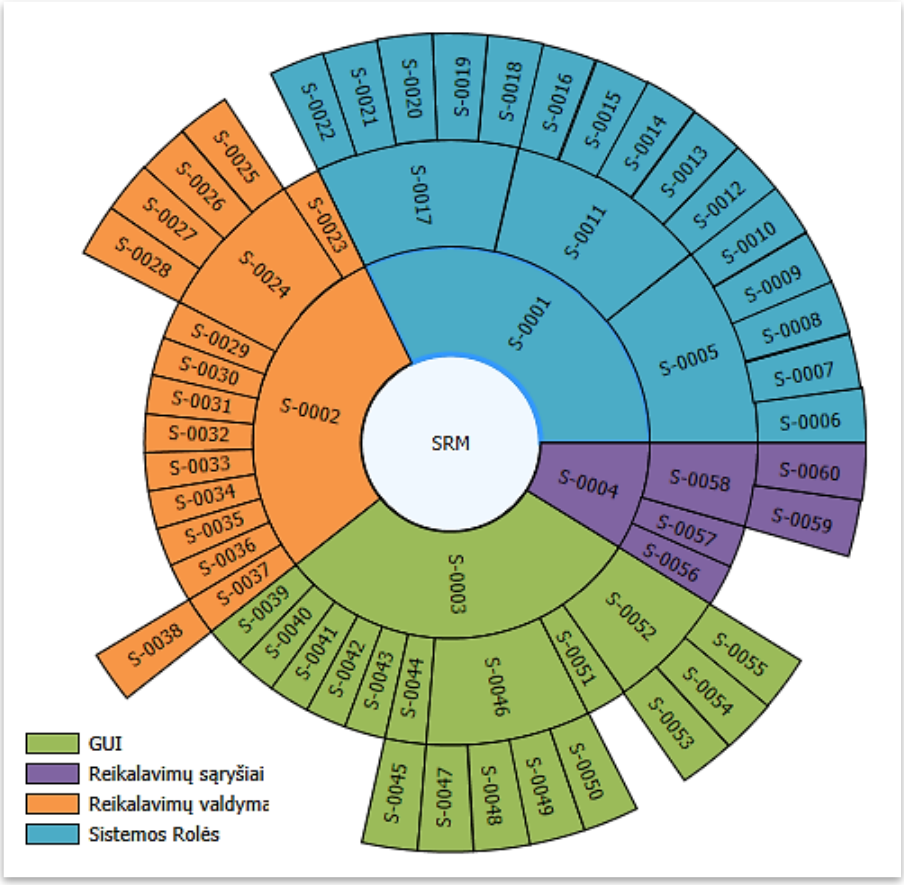
jo reikalavimų specifikacijos ir suvesti į sistemą sudarant atitinkamus dekompozicijos sąryšius, taip išlaikant originalią reikalavimų hierarchiją. Vėliau minėti duomenys buvo keičiami, pertvarkomi ir kitaip modifikuojami, siekiant išbandyti sukurto prototipo lankstumą ir atsparumą.

Nustatyta, kad sukurtas SRM prototipas leido visiems naudotojams sėkmingai atlikti numatytus reikalavimų inžinerijos etapus, t. y., apdoroti, vizualizuoti ir kitaip leisti manipuliuoti įvestais programinio įrankio reikalavimų duomenimis:

- „SRM administratorius“ galėjo:
    - Peržiūrėti projekte registruotus reikalavimus, jų tarpusavio sąryšius, atributus ir pakeitimų istoriją;
    - užregistruoti ir modifikuoti reikalavimus nebūdamas atsakingas to projekto asmuo;
    - konfigūruoti projekto nustatymus.
- „Projekto vadybininkas“ galėjo:
    - Peržiūrėti projekte registruotus reikalavimus, jų tarpusavio sąryšius, atributus ir pakeitimų istoriją;
    - užregistruoti ir modifikuoti reikalavimus tik būdamas atsakingas to projekto asmuo;
    - konfigūruoti projekto nustatymus.
- „Svečias“ galėjo:
    - Peržiūrėti projekte registruotus reikalavimus, jų tarpusavio sąryšius, atributus ir pakeitimų istoriją.

Atlikus testavimą galima teigti, jog sukurta sistema sugeba apdoroti, vizualizuoti ir kitaip leisti skirtingų rolių naudotojams manipuliuoti sistemoje esančiais programinio įrankio projektais ir jų reikalavimais. Tai rodo, jog prototipas yra tinkamas naudojimui nedidelio masto projektams, esant nedideliam naudotojų kiekiui projektui.

Sistemos verifikavimas atliktas „rekursiniu būdu“ naudojant pačios vizualizavimo sistemos reikalavimų rinkinį - sutraukus visus priede Nr. 1 esančius reikalavimus į sistemą, ir nustačius kiekvieno iš jų kategoriją į atitinkamą (atsižvelgiant į tai, kaip jie sugrupuoti priede) dekompozicijos sąryšiu gaunamas toks Sunburst diagramos vaizdas:



33 pav. Meta-reikalavimų dekompozicija

Diagramoje (33 pav.) naudojama „Category“ spalvų schema, kuri aiškiai pateikia, kurie reikalavimai kuriai grupei (kategorijai) priklauso. Dėl aiškumo, šalia pateikiama ir spalvų legenda.

Verifikavimo rezultatai rodo, jog sukurtas prototipas yra tinkamas naudojimui vidutinio masto projektams, esant nedideliam naudotojų kiekiui viename projekte.

Sistemos atsparumas ir gebėjimas veikti be klaidų nebuvo išbandytas su itin dideliais duomenų rinkiniais, nes realių duomenų (tokio dydžio) gauti nepavyko, o naudoti imitacinius - netikslinga, nes duomenys gali būti sudaryti taip, jog sistema visiškai nesusidurtų su problema (duomenys sudarytų pilną skritulį Sunburst diagramoje) arba viena tam tikro sąryšio hierarchija būti labai masyvi taip „peržengdama“ prototipo galimybes.

IV. REZULTATAI IR IŠVADOS

Pagrindiniai darbo **rezultatai** atitinka iškeltus uždavinius ir yra tokie:

1. Išanalizuoti reikalavimų valdymo esminiai principai ir reikalavimų valdymo sistemų struktūra bei atlikta praktikoje naudojamų duomenų vizualizavimo metodų lyginamoji analizė.
2. Sudaryti Sunburst ir Netmap vizualizavimo metodams skirti algoritmai, grafiškai pateikiantys programinės įrangos reikalavimus ir jų sąryšius.
3. Suprojektuota ir realizuota reikalavimų valdymo sistemos duomenų bazė bei programiškai realizuoti sukurti algoritmai Sunburst ir Netmap vizualizacijoms. Visi nauji sistemos moduliai sukurti naudojant UAB Omega Technology kompanijos sukurtą programavimo karkasą „Appframe R4“ ir Visual Basic .Net programavimo kalbą.
4. Autoriaus sukurtas reikalavimų valdymo sistemos prototipas išbandytas ir verifikuotas naudojant realius reikalavimų duomenis ir skirtingų vaidmenų naudotojus.

Įvertinus atliktą metodų analizę, realizavus vizualizavimo algoritmus ir visą reikalavimo valdymo sistemą ir išanalizavus gautus rezultatus, galima daryti tokios išvadas:

1. Tinkamam reikalavimų valdymui svarbu laiku pradėti registruoti reikalavimų pokyčius, tinkamai nustatyti jų svarbumą projekto sėkmei bei nuo pat pradžių vystyti jų atsekamumą: registruoti tik esminius reikalavimų atributus ir jų kitimą, reikalavimų tarpusavio sąveikas.
2. Reikalavimų ir jų sąryšių vizualizavimui galima naudoti įvairius vizualizacijų metodus ir jų kombinacijas: vieni metodai yra tinkamesni, kai norima pateikti projekte esančių reikalavimų ir jų sąryšių visumą, kiti išryškinti konkrečių reikalavimų susietumą su kitais to projekto reikalavimais.
3. Žiedinio (Sunburst) ir hierarchinio (Netmap) duomenų vizualizavimo metodų kombinacija yra efektyvi, nes pirmasis metodas padeda tinkamai reprezentuoti programinei sistemai keliamų reikalavimų visumą, o antrasis – išgryninti ir pateikti konkretaus reikalavimo sąryšius su kitais reikalavimais.
4. Pateikti vizualizavimo algoritmai (ir jų kombinacija) yra pritaikyti ir optimizuoti būtent reikalavimų valdymui: kitokioms duomenų struktūroms diagramos gali būti neefektyvios ar net iškraipyti informaciją.
5. Pateikta programinė sistema, kuri realizuoja sukurtus algoritmus bei gali grafiškai pateikti projekto reikalavimų informaciją visiems naudotojams, bet apriboti jos (informacijos) modifikavimą priklausomai nuo naudotojo rolės.

6. Sistemos verifikacijos metu panaudotas nepriklausomas teksto duomenų masyvas – pačios kuriamos sistemos reikalavimų specifikacija. Verifikacijos rezultatai rodo, kad sukurta sistema:
    - geba grafiškai naudotojams pateikti joje esančius reikalavimus;
    - atvaizduoja reikalavimų tarpusavio sąryšius, išlaikant jų semantinę prasmę;
    - leidžia naudotojams identifikuoti reikalavimus pagal jų charakteristikas dėka naudojamų spalvų schemų, kas pagerina bendrą projekto reikalavimų visumos suvokimą ir analizavimą.

    Verifikacijos metu taip pat pastebėta, kad informacijos tikslumas ir nauda priklauso nuo to, kaip toji informacija bus tvarkoma pačių naudotojų.

V. LITERATŪRA

[1] A.Kaifori, C. Halatsis, G. Lepouras, V. Costas, E. Giannopoulou. 2007. *Ontology Visuzalization Methins – A Survey*. Grecce.

[2] A. Moreira, J. Araújo & I. Brito. 2002. *Crosscutting Quality Attributes for Requirements Engineering.* Italy.

[3] A. Heinicke, C. Liao, K. Walbaum, J. Bützler & C. M. Schlick. 2015. *User centered evaluation of interactive data visualization forms for document management systems*. Chair and Institute of Industrial Engineering and Ergonomics of RWTH Aachen University, Bergdriesch 27, 52062 Aachen, Germany.

[4] A. Noyer, P. Iyenghar, E. Pulvermueller, F. Pramme & G. Bikker. 2015. *Traceability and Interfacing Between Requirements Engineering and UML Domains using the Standardized ReqIF Format*. Germany.

[5] B. Nuseibeh, J. Kramer, A. Finkelstein. 1994. *A Framwork for Expressing the Relationships Between Multiple Views in Requirements Specification.* IEE Transactions on Software Engineering Vol. 20 No. 10.

[6] C. Stab, M. Breyer, K. Nazemi, D. Burkhardt, C. Hofmann, D. W. Fellner. 2010. *SemaSun: Visualization of semantic knowledge based on an improved sunburst visualization metaphor.* Fraunhofer Institute for Computer Graphics Research Fraunhoferstrasse 5, 64283 Darmstadt, Germany.

[7] Daniel Waterman. 2011. *Ofni Systems: Example Validation Spreadsheet Functional Requirements Specification.*

[8] Donald G. Firesmith. 2005. *A Taxonomy of Security-Related Requirements*. Software Engineering Institute.

[9] Donn Le Vie, Jr. 2010. *Writing Software Requirements Specifications*.

[10] Grant Zemont. 2005. *Towards Value-Based Requirements Traceability*.

[11] H. Schwarz, J. Ebert, A. Winter. 2010. *Graph-based Traceability – A Comprehensive Approach*.

[12] http://contsem.unizar.es/def/sector-publico/images/0.1.4_contract_e.png [tikrinta 2017-05-24]

[13] http://ib.compscihub.net/wp-content/uploads/2015/04/IB-Pseudocode-rules.pdf [tikrinta 2017-05-24]

[14] http://systemimplementationmanager.com/wp-content/uploads/2016/06/treemap.jpg [tikrinta 2017-05-24]

[15] E. Hull, K. Jackson & J. Dick. 2002. *Requirements Engineering: 2nd edition.* JAV: Springer Science.

[16] IBM Knowledge Center. 2017. *Link types in requirements projects*..

[17] IBM Knowledge Center. *Traceability Tree view.*

[18] Jo Atlee. 2006. *Capturing the Requirements.*

[19] John K. G & L. A. Stauffer. 1999. *A Taxonomy for Design Requirements from Corporate Customers.* Research in Engineering Design

[20] J. Stasko & E. Zhang. 2003. *Focus Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualizations.* GVU Center and College of Computing Georgia Institute of Technology Atlanta, GA 30332-0280.

[21] J. Stasko, R. Catrambone, M. Guzdial & K. McDonald. 2000. *An Evaluation of Space-Filling Information Visualizations for Depicting Hierarchical Structures.*

[22] K. Wiegers & J. Beatty. 2013. *Software Requirements, Third Edition.*

[23] Klaipėdos universiteto senatas. 2010. *Klaipėdos universiteto studentų savarankiškų rašto ir meno darbų bendrųjų reikalavimų aprašas*. Klaipėda

[24] K. Wnuka, T. Gorschek, S. Zahda. 2012. *Obsolete software requirements*. Swedan.

[25] L. Lehtola, M. Kauppinen, S. Kujala. 2004. *Requirements Prioritization Challenges in Practice.* Helsinki University of Technology, Software Business and Engineering Institute, P.O. Box 9210, FIN-02015 HUT, Finland.

[26] L. Chung, J. Cesar & S. do Prado Leite. 2011. *On Non-Functional Requirements in Software Engineering*.

[27] L. Jiang, A. Eberlein, B. H. Far. 2004. *A Methodology for Requirements Engineering Process Development*.

[28] Marius Balandis. 2015. *Įmonės programinio produkto reikalavimų valdymo sistemos prototipas.* informatikos bakalauro darbas. Vadovas prof. dr. V. Denisov. Klaipėdos universitetas, Gamtos ir matematikos mokslų fakultetas, Informatikos katedra. 49 p. Klaipėda.

[29] Martin Glinz. 2007. *On Non-Functional Requirements*. Department of Informatics, University of Zurich, Switzerland.

[30] Massimo Felici. 2004. *Observational Models of Requirements Evolution*.

[31] M. Barlow, J. Galloway, & H. A. Abbass. 2011. *Mining Evolution Through Visualization.* Virtual Environments and Simulation Lab (V.E.S.L), School of Computer Science, University of New South Wales, Australian Defence Force Academy Campus, Canberra, Australia.

[32] P. Hope, P. White. 2005. *Software Security Requirements.*

[33] P. Berander & A. Andrews. 2010. *Requirements Prioritization.* in Engineering and Managing Software Requirements, edited by A. Aurum and C. Wohlin, Springer Verlag.

[34] Paul T. F. Noordveld. 2013. *How can requirements management be improved by knowledge management principles?*. Utrecht University.

[35] P. Duffett and R. Vernik. 1997. *Software System Visualisation: Netmap Investigations*. Australia.

[36] P. Zielczynski. 2008. *Requirements Management Using IBM® Rational® RequisitePro®*.

[37] P. Heim, S. Lohmann, K. Lauenroth, J. Ziegler. 2008. *Graph-based Visualization of Requirements Relationships*. University of Duisburg-Essen, Germany.

[38] Prof. Dr. Vitalijus Denisovas. 2004. *Modernioji programų sistemų inžinerija: įvadas į sistemų inžineriją*.

[39] Reliasoft. 2007. *Reliability Requirements and Specifications*.

[40] Robert Halligan. *Types of Requirements*.

[41] S. Gudas, J. Tekutov, V. Denisovas. 2014. *Requirements Enhancement Approach Based on the Problem Domain Model*.

[42] S. Gudas, J. Tekutov, V. Denisovas. 2010. *Study program requirements engineering method and information system. Information Sciences*, vol. 53, pp.106–126.

[43] S. McGee, D. Greer. 2011. *Software Requirements Change Taxonomy: Evaluation by Case Study*.

[44] S. Park, F. Maurer, A. Eberlein & T. Fung. 2010. *Requirements Attributes to Predict Requirements Related Defects.* Canada.

[45] Stephen Armitage. 1996. *Software Requirements Specification*.

[46] Stephen Few. 2007. *Save the Pies for Dessert*.

[47] Sue Burk. 2011. *Tools and techniques for tracking changes to software requirements*.

[48] The Data Visualisation Catalogue. 2017. *Sunburst Diagram.*

[49] T. Bladh, D. A. Carr, J. Scholl. 2004. *Extending Tree-Maps to Three Dimensions: A Comparative Study*. Sweden.

[50] T. Merten, D. J¨uppner. 2011. *Improved Representation of Traceability Links in Requirements Engineering Knowledge using Sunburst and Netmap Visualizations*.

[51] M. Unterkalmsteiner, R. Feldt, T. Gorschek. 2013. A Taxonomy for Requirements Engineering and Software Test Alignment. ACM Trans. Softw. Eng. Methodol. V, N,. 39 pages.

[52] V. Castañeda, L. Ballejos, L. Caliusco, R. Galli. 2010 .*The Use of Ontologies in Requirements Engineering.* Global Journal of Researches in Engineering. Vol. 10 Issue 6 (Ver 1.0)

[53] V.Shukla, G. Aurio, C. Baron. 2011. *A Graph-Based Requirement Traceability Maintenance Model.* France.

[54] X. Chen, J. Hosking, J. Grundy. 2012. *Visualizing Traceability Links between Source Code and Documentation*. Austria

SANTRAUKA

Marius Balandis

Programinės įrangos reikalavimų valdymo sistemos, grįstos reikalavimų susietumo vizualizavimu, kūrimas: techninių informacinių sistemų inžinerijos magistro baigiamasis darbas. Vadovas prof. dr. V. Denisov. Klaipėdos universitetas, Jūros technologijų ir gamtos mokslų fakultetas, Informatikos ir statistikos katedra. Klaipėda, 2017, 70 p.

Darbo tikslas – sukurti programinės įrangos reikalavimų valdymo sistemą, sugebančią vizualiai atvaizduoti programinės įrangos reikalavimų gausą ir sąryšius tarp jų. Darbo uždaviniai: išanalizuoti reikalavimų valdymo esminius principus; apžvelgti praktikoje naudojamus duomenų vizualizavimo metodus; pateikti algoritmus Sunburst ir Netmap vizualizavimo metodams, skirtus vizualizuoti programinės įrangos reikalavimus ir jų sąryšius; realizuoti pateiktus algoritmus; išbandyti sukurtą sistemos prototipą, pagrįsta minėtais algoritmais, naudojant realius duomenis ar procesų aprašus. Apžvelgtos fundamentalios reikalavimų valdymo programinės įrangos kūrimo projektuose charakteristikos, išnagrinėji naudojami ir potencialiai galimi duomenų vizualizacijos metodai. Pasirikus tinkamiausius vizualizacijos metodus, sukurti Sunburst ir Netmap vizualizavimo algoritmai bei programinės įrangos sistema, realizuojanti minėtus metodus ir veikianti UAB Omega Technology naudojamoje „Appframe R4“ programavimo aplinkoje. Sukurta programinė įranga išbandyta su realiais duomenimis.

PAGRINDINIAI ŽODŽIAI**:** reikalavimų inžinerija, reikalavimų valdymo sistema, programavimo karkasas, programavimo platforma Microsoft .NET, duomenų vizualizavimas, duomenų gavyba

SUMMARY

Marius Balandis

Development of software requirements management system based on visualization of requirements relations: Master thesis of technical information systems' engineering. Supervisor prof. dr. V. Denisov. Klaipėda University, Sea technology and nature studies faculty, IT and statistics department. Klaipėda, 2017, 70 p.

Purpose of the paper – to create a software prototype for systems requirements management, capable of visually showcasing software requirements, their abundance and relations between them. Tasks of the paper: to analyze main principles of requirements management; to review in practice used data and visualizing methods; to showcase algorithms for the "Sunburst" and "Netmap" visualization methods, designated to visualize software requirements and their relations; to realize presented algorithms; to try out the created system's prototype, based on the afore mentioned algorithms using real data or the descriptions of the processes. Reviewed fundamental requirements management characteristics for software creation in the projects, analyzed currently used and potentially possible data visualization methods. The most suitable visualization methods "Sunburst" and "Netmap" have been chosen to create a software prototype based on "Appframe R4" framework, which is used by UAB Omega Technology. Created software prototype was tested with real data.

KEYWORS: requirements engineering, requirements management system, software framework, Microsoft .NET platform, data visualization, data mining.

1 PRIEDAS

Kuriamam programinės įrangos prototipui keliami reikalavimai

1. Vartotojai skirstomi į tris grupes. Žemiausio lygmens teisės yra fundamentalios ir galioja visiems kitų lygmenų vartotojams, nebent konkrečios teisės yra perrašomos to vartotojo grupės teisėmis.

   1.1. SRM administratoriai - aukščiausio lygmens vartotojai, turintys visas teises į visus egzistuojančius projektus.

      1.1.1. SRM administratoriai turi teisę kurti naujus, redaguoti ir trinti esamus reikalavimus visuose, sistemoje esančiuose, projektuose.

      1.1.2. Šios grupės nariai gali į sistemą įtraukti naujus projektus ir pašalinti esamus, su sąlyga jog šalinami projektai neturi registruotų reikalavimų.

      1.1.3. SRM administratoriai gali keisti visus sistemos parametrus, nepriklausomai nuo projekto.

      1.1.4. Taip pat šie vartotojai nustato kitų vartotojų prieigos lygius prie konkrečių projektų.

   1.2. Šios grupės nariais gali būti tik sistemoje registruoti vartotojai.

   1.3. Projektų vadybininkai - vidutinio lygmens vartotojų grupė, kurios narys atsakingas tik už jam priskirtus projektus.

      1.3.1. Projektų vadybininkai gali kurti naujus, redaguoti ir trinti esamus reikalavimus tik jiems paskirtuose projektuose.

      1.3.2. Gali keisti jiems priklausančių projektų sistemos parametrus.

      1.3.3. Projektų vadybininkai negali valdyti vartotojų prieigos lygio prie projektų, (įskaitant ir jam priklausančius projektus).

      1.3.4. Projektų vadybininkai gali būti tik sistemoje registruoti vartotojai.

      1.3.5. Projekto vadybininkas jam nepriklausančiuose projektuose turi būti laikomas „svečiu“

   1.4. Svečiai – tai visi kiti likę vartotojai. Šios grupės nariai vartotojų teisių atžvilgiu yra žemiausio lygmens

      1.4.1. Svečiai negali redaguoti, trinti ar kurti naujų reikalavimų ar kaip kitaip keisti jų duomenis.

      1.4.2. jie negali keisti jokių nustatymų.

      1.4.3. Svečiai gali skaityti reikalavimus.

      1.4.4. Svečiai gali pasirinkti, kokį projektą, sąryšį ir legendą rodyti.

      1.4.5. Svečiai gali pasirinkti kokį projektą, sąryšį ir legendą sistema automatiškai rodys atidarant įrankį.

2. Reikalavimų valdymas

2.1. Reikalavimo identifikavimo numeris turi būti kodas sudarytas iš:

- projekto kodo pirmoji raidė. Jei jau egzistuoja projektas su reikalavimais kurie prasideda tokia raide, tuomet naudojamos pirmos dvi projekto kodo, jei jos užimtos – pirmos trys ir t. t.,
- simbolio, skiriančio kodo dalis (pageidautina „-“)
- reikalavimo eilės numeris. Skaitmenų kiekis kiekvienam projektui nurodomas atskirai. Jei nėra nurodoma, tada skaitmenų skaičius yra lygus 1.

2.2. Sistema turi leisti vartotojams užregistruoti naujus reikalavimus jau egzistuojančiuose projektuose.

2.2.1. Nauji reikalavimai turi būti registruojami su galimybe nustatyti jiems tokius atributus (toliau - metaduomenys):

- pavadinimas (angl. *title*),
- aprašymas (angl. *description*),
- svarbumas (angl. *severity*),
- būsena(angl. *status*),
- tipas (angl. *type*),
- kategorija (angl. *category*),
- etapas (angl. *milestone*),
- pageidauta (kieno?) (angl. *requested by*).

2.2.2. Užregistravus reikalavimą, vartotojas turi galimybę iškart pateikti papildomą informaciją apie reikalavimą:

- Priedai (failai)
- Sąryšiai su kitais reikalavimais
- Sistemos objektai, tenkinantys šį reikalavimą.
- Tai pat išlieka galimybė keisti pateiktus reikalavimo atributus.

2.2.3. Naujai užregistruotas reikalavimas visada turi būti būsenos „pasiūlytas“ (angl. proposed)

2.2.4. Naujai registruojamas reikalavimas pagal nutylėjimą yra vidutinio svarbumo.

2.3. Sistemoje užregistruoti reikalavimai yra redaguojami bet kuriuo metu, nepriklausomai nuo esamų jų metaduomenų

2.4. Visi reikalavimų metaduomenų pakeitimai turi būti registruojami, su galimybe peržiūrėti ir komentuoti jų kitimo istoriją.

2.5. Reikalavimai turi būti skirstomi pagal kategoriją.

2.6. Reikalavimai turi būti skirstomi pagal tipą.

2.7. Prie reikalavimo turi būti nurodytas jo šaltinis - tai gali būti tiek projekto dokumentacija, tiek (fizinio / juridinio) asmens vardas.

2.8. Reikalavimas gali neturėti nurodyto šaltinio jei jis pateikiamas projekto pradinėje stadijoje

2.9. Reikalavimo metaduomenys turi savo spalvų schemas (išskyrus „pavadinimas“, „aprašas“ ir „pageidauta“).

2.10. Reikalavimų „svarbumo“ ir „būsenos“ galimos reikšmės visuose projektuose yra vienodos (taip pat ir jų spalvų schemos).

2.11. Reikalavimai turi būti redaguojami tik per tam skirtą vartotojo grafinės sąsajos elementą (pavyzdžiui modalinį dialogą).

2.12. Reikalavimo redagavimo įrankis tūrėtų būti pasiekiamas iš:

- reikalavimų sąrašo,
- reikalavimo sąryšių vizualizacijos (iš kiekvieno tuo momentu vaizduojamo reikalavimo)

3. SRM privalo turėti vartotojo grafinę sąsają.

3.1. Vartotojai gali neuždarydami įrankio perjungti projektą, keisti vaizduojamą sąryšį ir spalvų schemą.

3.2. Vartotojas privalo turėti galimybę perkrauti projekto duomenis, nepersijungiant tarp skirtingų projektų ir iš naujo neatidarant įrankio.

3.3. Sistema turi rodyti spalvų schemos legendą, su galimybe ją bet kada įjungti ir išjungti.

3.4. Vartotojas turi galimybę pasirinkti, kokį projektą, sąryšį ir spalvų schemą sistema rodys tik atsidarius įrankį.

3.5. Visos egzistuojančios spalvų schemos yra konfigūruojamos.

3.6. Sistema turi suteikti galimybę ieškoti reikalavimų pagal nurodytą frazę ar bet kokią kitą semantinę simbolių eilutę.

3.6.1. Reikalavimų paieška vykdoma ieškant nurodytos reikšmės šiuose reikalavimų metaduomenyse:

- aprašymas,
- pavadinimas,
- etapas,

- būsena,
- pageidauta

3.7. Vartotojas turi galimybę matyti visą tuo momentu pasirinkto projekto reikalavimų sąrašą.

3.7.1. Turi būti galimybė filtruoti reikalavimų sąrašą.

3.7.2. Reikalavimų sąrašas turi turėti du formatus:

- minimalistinis sąrašas – jame turi būti rodoma tik reikalavimų ID ir pavadinimai;
- detalus sąrašas – jame rodomi visi reikalavimų metaduomenys.

3.7.3. Vartotojas turi galimybę paslėpti, o prireikus, ir vėl matyti reikalavimų sąrašą.

3.7.4. Reikalavimų sąraše vaizduojami duomenys negali būti redaguojami.

3.8. Pagrindinės vizualizacijos, vaizduojančios visus projekto reikalavimus turėtų identifikuoti savo elementus pagal atskiriamąjį (unikalų) metaduomenį, pavyzdžiui – reikalavimo kodą.

3.9. SRM braižomos vizualizacijos turėtų būti interaktyvios.

3.9.1. Vizualizacijų elementai galėtų vartotojams suteikti papildomą informaciją apie kiekvieną reikalavimą.

3.9.2. Vizualizacija, skirta konkretaus reikalavimo sąryšių atvaizdavimui turėtų leisti vartotojams „keliauti“ tarp susietų reikalavimų.

3.9.3. Vizualizacijos turėtų būti semantiškai susijusios.

4. Reikalavimai privalo turėti tarpusavio sąryšius.

4.1. Reikalavimų sąryšiai visuose projektuose yra vienodi (įskaitant spalvų schemą).

4.2. Reikalavimo sąryšis, nepriklausomai nuo sąryšio tipo, su savimi yra negalimas.

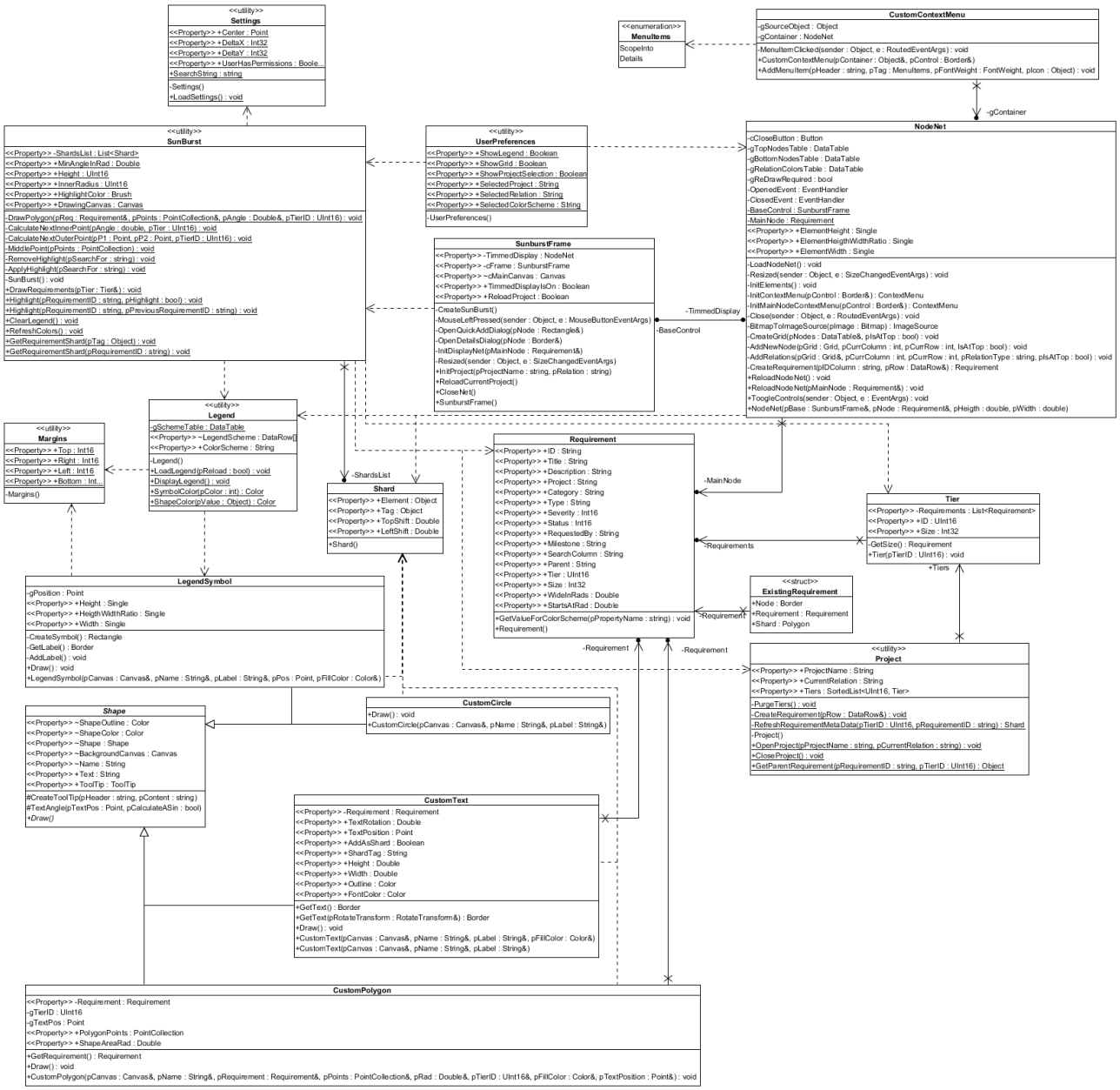
4.3. Reikalavimų sąryšiai turi tipus, kuriais jie (sąryšiai) yra apibūdinami.

4.3.1. Reikalavimų sąryšių tipai turi spalvų schema.

4.3.2. Sąryšių spalvų schema visuose projektuose vienoda.

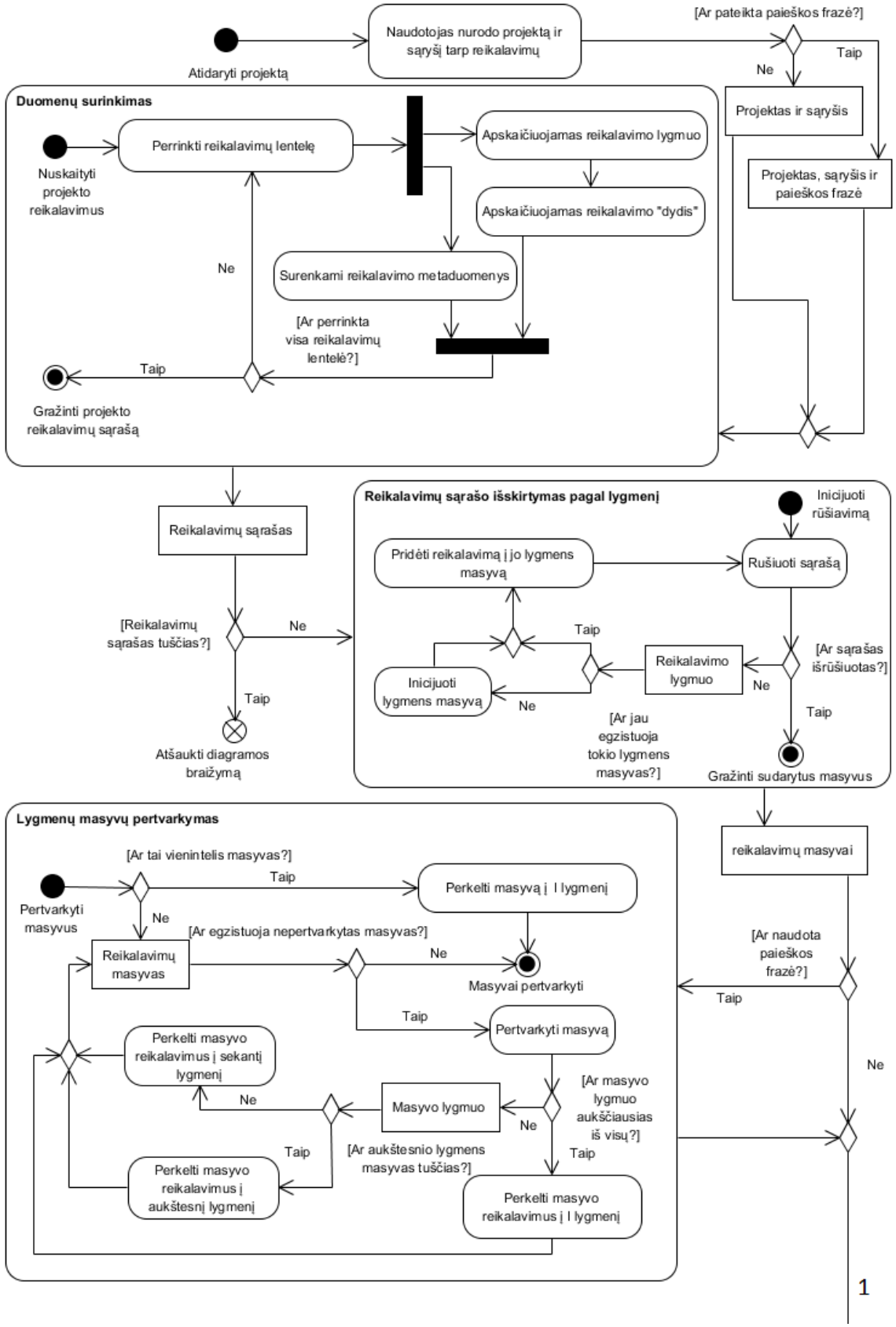
2 PRIEDAS

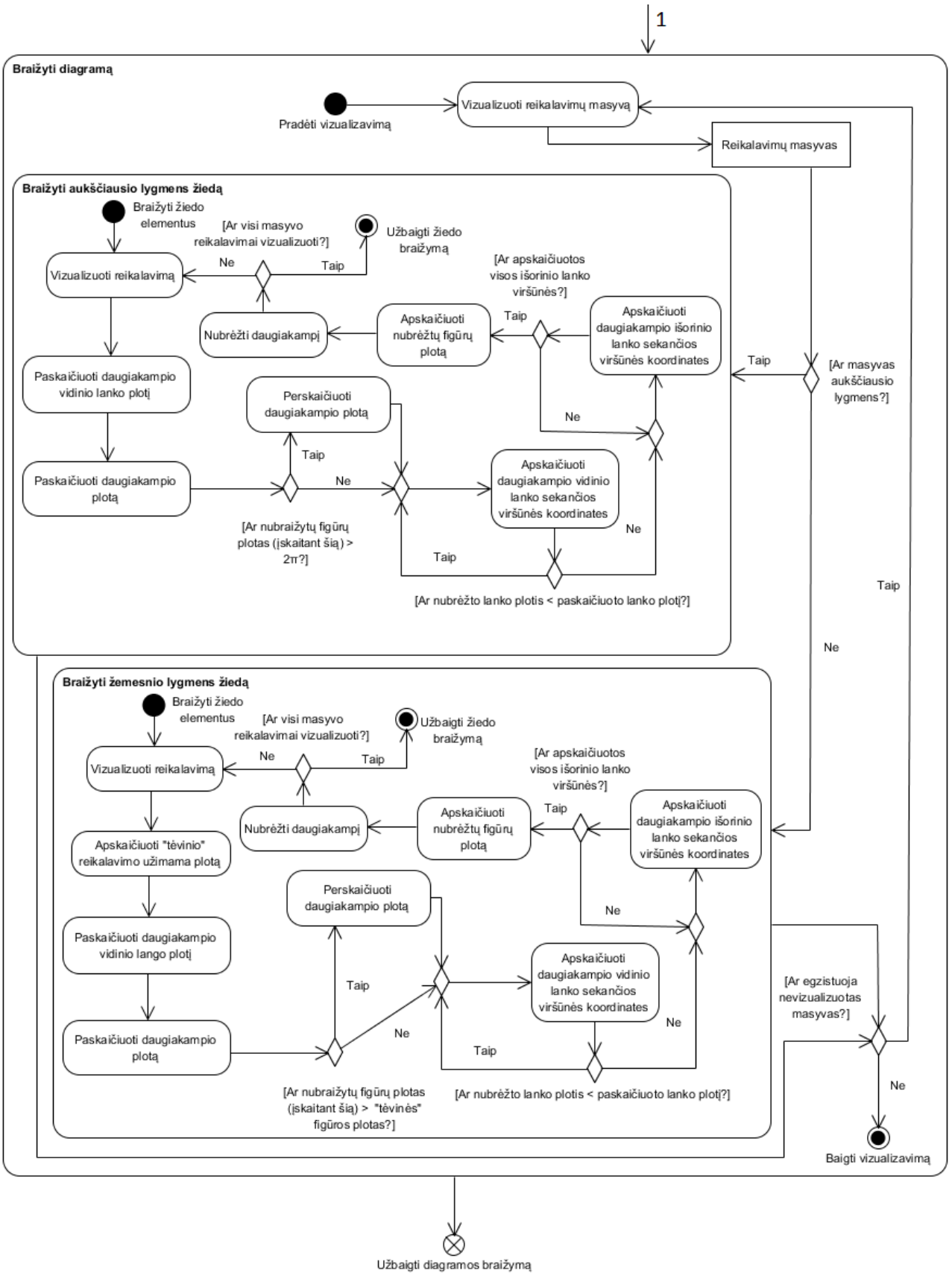
Kuriamo prototipo klasių diagrama



3 PRIEDAS

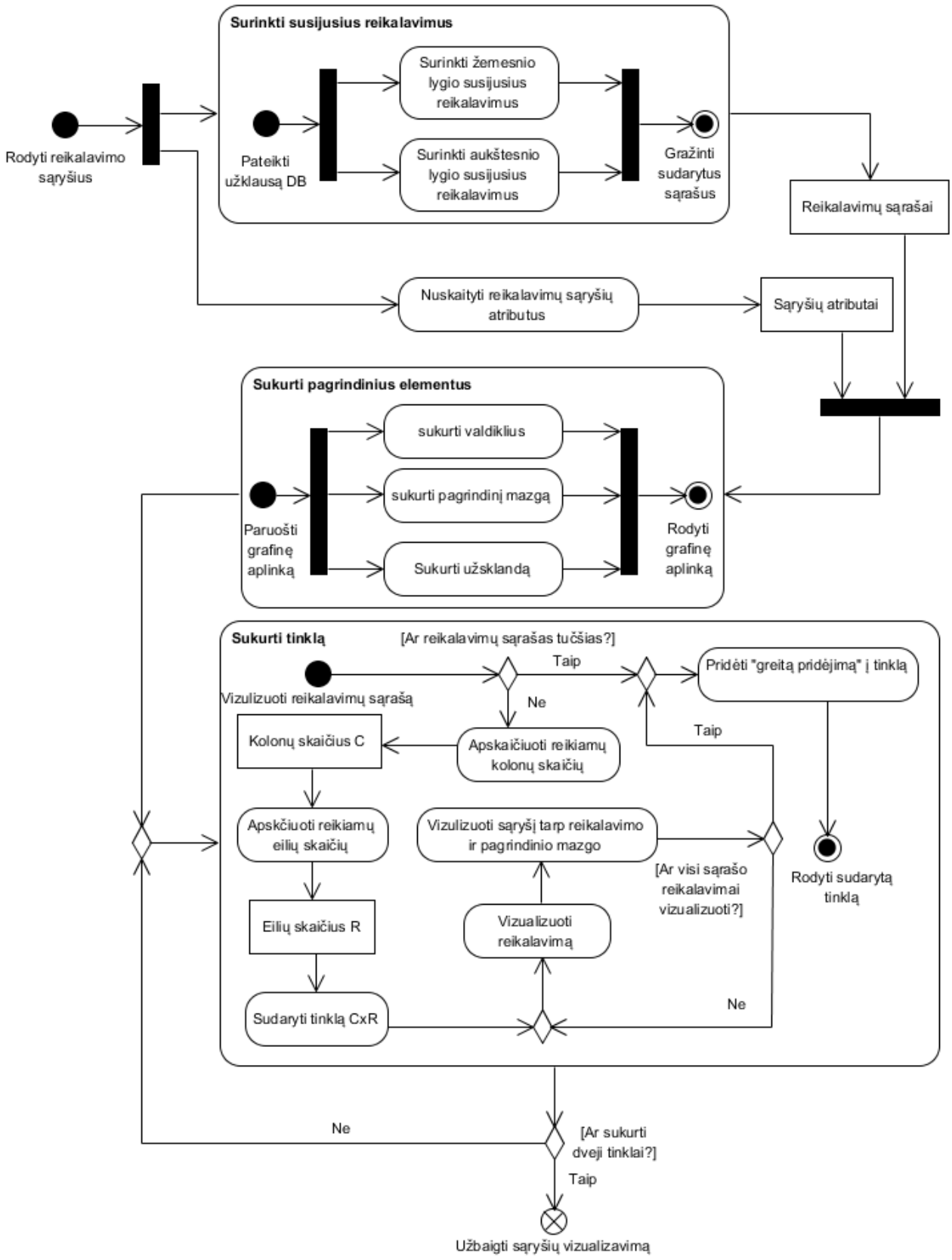
Sunburst algoritmo diagrama




1
Braižyti diagramą
Pradėti vizualizavimą
Vizualizuoti reikalavimų masyvą
Reikalavimų masyvas
Braižyti aukščiausio lygmens žiedą
Braižyti žiedo elementus
[Ar visi masyvo reikalavimai vizualizuoti?]
Užbaigti žiedo braižymą
Ne
Taip
Vizualizuoti reikalavimą
Nubrėžti daugiakampį
Apskaičiuoti nubrėžtų figūrų plotą
[Ar apskaičiuotos visos išorinio lanko viršūnės?]
Taip
Apskaičiuoti daugiakampio išorinio lanko sekančios viršūnės koordinates
Taip
[Ar masyvas aukščiausio lygmens?]
Paskaičiuoti daugiakampio vidinio lanko plotį
Perskaičiuoti daugiakampio plotą
Ne
Taip
Paskaičiuoti daugiakampio plotą
Ne
Apskaičiuoti daugiakampio vidinio lanko sekančios viršūnės koordinates
Ne
[Ar nubraižytų figūrų plotas (įskaitant šią) > 2π?]
Taip
[Ar nubrėžto lanko plotis < paskaičiuoto lanko plotį?]
Taip
Ne
Braižyti žemesnio lygmens žiedą
Braižyti žiedo elementus
[Ar visi masyvo reikalavimai vizualizuoti?]
Užbaigti žiedo braižymą
Ne
Taip
Vizualizuoti reikalavimą
Nubrėžti daugiakampį
Apskaičiuoti nubrėžtų figūrų plotą
[Ar apskaičiuotos visos išorinio lanko viršūnės?]
Taip
Apskaičiuoti daugiakampio išorinio lanko sekančios viršūnės koordinates
Apskaičiuoti "tėvinio" reikalavimo užimamą plotą
Perskaičiuoti daugiakampio plotą
Ne
Paskaičiuoti daugiakampio vidinio lango plotį
Taip
Apskaičiuoti daugiakampio vidinio lanko sekančios viršūnės koordinates
[Ar egzistuoja nevizualizuotas masyvas?]
Paskaičiuoti daugiakampio plotą
Ne
Ne
Taip
[Ar nubraižytų figūrų plotas (įskaitant šią) > "tėvinės" figūros plotas?]
[Ar nubrėžto lanko plotis < paskaičiuoto lanko plotį?]
Ne
Baigti vizualizavimą
Užbaigti diagramos braižymą

4 PRIEDAS

Netmap algoritmo diagrama



5 PRIEDAS

Kompaktinis diskas

Kompaktinio disko turinys:

- Baigimasis magistrinis darbas .pdf formatu
- Sukurto programinės įrangos prototipo realizacija
- Įvairios, darbe pateiktos, diagramos