Tallinna Ülikool
Digitehnoloogiate instituut

Tarkvarakomponendi nõuete analüüs SFOSi dokumendimooduli näitel

Bakalaureusetöö

Autor: Martti Naaber
Juhendaja: Romil Rõbtšenkov

Autor ..... ,, ... ,,2018
Juhendaja..... ,, ... ,,2018
Instituudi direktor..... ,, ... ,,2018

Tallinn 2018

Autorideklaratsioon

Deklareerin, et käesolev bakalaureusetöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

.....	.....
(kuupäev)	(autor)

Sisukord

Sissejuhatus

Tarkvaraarenduse elutsüklit saab defineerida kui protsessi, mille käigus selgitatakse välja, kuidas üks infosüsteem saab toetada organisatsiooni ärilisi vajadusi, pärast mida antud süsteem disainitakse, ehitatakse ning viiakse kasutajani (Dennis, Wixom & Roth, 2012, lk 6). Käesoleva bakalaureusetöö fookuses on konkreetselt analüüsifaas; analüüs kui termin viidates terviku osadeks lõhkumisele eesmärgiga mõista osade loomust, funktsiooni ja omavahelist seost (Dennis et al., 2012, lk 102).

Üheks selleks 'osaks' on nõue, mis on vajaduse praktiline väljendus (BABOK[1], 2015, lk 15), nt mida peab tarkvarasüsteem tegema, kui kasutaja vajutab suurt punast nuppu. Vastuse antud küsimusele leiab analüütik koostöös kliendiga, kuid see ei pruugi olla kerge, sest Blais (2013, lk 131) järgi ei pruugi klient teada, mis see probleem üldse on (*ala* 'miks seda punast nuppu üldse vaja on') või ei oska ta seda defineerida.

Nõuete defineerimine on väidetavalt üks põhilisi probleeme infosüsteemide alastes projektides (Paul & Tan, 2015); ometigi omab see kriitilist tähtsust, sest nõuete väljaselgitamise kui faasi kvaliteet mõjutab sellele järgnevate faaside ning sellest tulenevalt ka terve tarkvaratoote kvaliteeti (Alshazly et al., 2014, lk 513).

Käesoleva töö fookuses olevaks tarkvaratooteks on SFOS (ingl *Structural Funds Operational System*) ehk eurotoetuste taotlus- ja menetlussüsteem, mille arendusmeeskonnas täidab töö autor programmeerija rolli. Kuna antud süsteemi puhul eksisteerib pidev vajadus selgitada välja lisanduvate tarkvarakomponentide nõudeid ja kuna töö autoril on isiklik huvi analüütiku töö vastu, siis ühe uue mooduli (kui kliendi vajaduse tarkvaralise realiseeringu) najal tekkis autoril võimalus ka ise analüüsiprotsessist osa võtta.

Antud tööl on seega kaks eesmärki: esimeseks eesmärgiks on ära defineerida SFOSi dokumendi-mooduli ning olemasoleva tarkvarakomponendi uued nõuded vastavalt kliendi vajadustele; töö praktiliseks väärtuseks ja tulemuseks olles tarkvara spetsifikatsioon, mis on Wiess, Holzmann ja Frank (2012, lk 1102) järgi üks kriitiline edufaktor tehnoloogilistes projektides. Teiseks eesmärgiks on mõista nõuete väljaselgitamise protsessi kui teoreetilist kontseptsiooni.

[1] Ingl *Business Analysis Body of Knowledge*, mis on globaalselt tunnustatud ärianalüüsi alaste teadmuste kogum.

Töö koosneb neljast peatükist: esimeses peatükis uuritakse erialast kirjandust, et omandada arusaamine tarkvaraarenduse analüüsifaasist ning rakendada antud teadmisi ka praktikas, st reaalses nõuete väljaselgitamises. Teises peatükis kirjeldatakse analüüsiprotsessi konkreetselt SFOSi projektis, sest igasuguse analüüsi puhul tuleb arvestada kehtivate praktikatega. Samamoodi tuleb analüüsi puhul arvestada ka terve süsteemiga, selle võimaluste ja piirangutega, seega antakse kolmandas peatükis ülevaade rakendusest ja olemasolevatest seotud komponentidest.

Neljandas peatükis toimub nõuete väljaselgitamise protsess, st töö autor võtab ette kliendi probleemi või vajaduse, analüüsib selle läbi ning pakub välja võimalikud lahendused – nii iga vajaduse kohta iteratiivselt. Kõige lõpus tuuakse välja töö praktiline tulemus ehk nõuete spetsifikatsioon.

Käesoleva töö autorina tänan oma juhendajat Romil Rõbtšenkov’t suurepärase, toetava ning efektiivse koostöö eest.

1. Analüüsifaas ja nõuete väljaselgitamise protsess

Analüüsifaasi eesmärgiks on saada vastused küsimustele, et kes süsteemi kasutab, mida süsteem teeb, ning kus ja kuna seda kasutatakse (Dennis et al., 2012, lk 13). Selles faasis uuritakse olemasolevat lahendust või situatsiooni, identifitseeritakse täiendused ning defineeritakse uue süsteemi nõuded (Dennis et al., 2012, lk 102) – antud protsessi ning tegelikult terve tarkvaraarenduse elutsükli keskmes on süsteemianalüütik (Dennis et al., 2012, lk 7).

Kasutusel on ka terminid nagu ärianalüütik ja –analüüs. Ärianalüüs on praktika, mis võimaldab organisatsioonis läbi viia muutusi – seda tehakse niimoodi, et defineeritakse vajadused ning pakutakse välja lahendused, mis loovad väärtust osanikele[2]; ärianalüüsi saab läbi viia mitmes erinevas võtmes, nt äriintelligentsi või infotehnoloogia perspektiivis (BABOK, 2015, lk 2). Fookuses on seega ärilise eesmärgi saavutamine, sõltumata tulemuse formaadist, olles selleks nt kas organisatsiooniline muutus või tarkvaraprogramm (Rubens, 2007, lk 122).

Blais (2012, lk 94) kirjutab, et ärianalüütikut võib pidada probleemi poole süsteemianalüütikuks ning süsteemianalüütikut võib pidada tehnilise (lahenduse) poole ärianalüütikuks. See, mida viimane täpselt teeb, ei ole selgelt ära defineeritud, kuid tema tööülesandeid iseloomustavad märksõnad nagu süsteem, äristrateegia, organisatsioon jne (Rubens, 2007, lk 121-122). Äri-analüütik on seega iga inimene, kes teostab ärianalüüsi alaseid ülesandeid sõltumata isiku töötiitlist või rollist organisatsioonis, olles selleks nt tooteomanik, süsteemianalüütik või nõuete väljaselgitamise insener (BABOK, 2015, lk 2-3).

Analüütiku esimene ülesanne on ära defineerida reaalne probleem, mille lahendamist organisatsioon vajab – see võib tähendada esialgse probleemi eiramist ning uurimistööde alustamist (Blais, 2012, lk 136). Kui probleem on defineerinud ja valideeritud, on aeg välja selgitada lahendus või tuua välja nõuded, mis moodustavad lahenduse – paljudele analüütikutele moodustab see suurema osa nende tööst (Blais, 2012, lk 197).

Mis on nõue? Nõue on väide selle kohta, mida süsteem peab tegema või mis omadused peavad sellel olema (Dennis et al., 2012, lk 104), fookuses olles arusaamine, et millist väärtust on võimalik luua, kui nõue täidetakse (BABOK, 2015, lk 15). Nõudeid võib paigutada mitmesse kategooriasse:

[2] Süsteemiga (projektiga) seotud füüsilised või juriidilised isikud, kes võivad süsteemi edust või ebaedust võita või kaotada (Kaur & Singh, 2010, lk 28).

ärinõuded, osanike nõuded, lahenduse nõuded, funktsionaalsed nõuded, mittefunktsionaalsed nõuded jne (BABOK, 2015, lk 15). Dennis et al. (2012, lk 107) toovad aga välja, et on keeruline tõmmata eristavat joont erinevate kategooriate vahele, ning et mõned ettevõtted kasutavad antud termineid ühes ja samas tähenduses.

Nõuete väljaselgitamine on kõige kriitilisem aspekt terves tarkvaraarenduse elutsüklis (Dennis et al., 2012, lk 104), väärate või puuduvate nõuete tulemuseks võib olla vigane või lõpetamata toode, sõltumata sellest, kui oskuslikult on edasistes faasides tegutsetud (Alshazly et al., 2014, lk 513). Nt Standish Group (2014, viidatud Curcio et al., 2018, lk 32 kaudu) poolt läbi viidud uuringus leiti, et suurem osa põhjustest, miks tarkvaraprojektid katkestatakse, on mingil moel seotud nõuetega. Seega on oluline, et täielik nõuete komplekt oleks identifitseeritud suhteliselt varakult (Wiess, Holzmann & Frank, 2012, lk 1101), sest kui hiljem leitakse, et mingid nõuded on väärad või puudulikud, võib tööde ümbertegemine nõuda mahtude suurendamist, mis omakorda suurendab projekti aega ja maksumust (Dennis et al., 2012, lk 104).

Nõuete väljaselgitamise protsess (ingl *requirements engineering*) tähistab oma definitsioonilt rida tehnilisi otsuseid, mis viivad lahendamist vajava probleemi tunnistamisest selle detailse spetsifikatsioonini (Kaur & Singh, 2010, lk 27), keskpunktis olles tarkvarasüsteemi funktsioonid ja piirangud, mis aitavad saavutada reaalse maailma eesmärke (Zave, 1997, viidatud Belfo, 2012, lk 310 kaudu). Protsess ise koosneb mitmetest etappidest:

1. Protsess algab andmete kogumisega, kus osanike abiga kaardistatakse ära süsteemi nõuded ja piirangud (Curcio et al., 2018, lk 33). Enimlevinud meetodid on intervjuud, küsitlused, dokumentatsioonianalüüsid ja vaatlused (Dennis et al., 2012, lk 112).
2. Kui andmed on kogutud, algab analüüsi- ja läbirääkimiste protsess, et mõista tervet süsteemi ning kontrollida, kas kaardistatud nõuded on terviklikud, täielikud ja teostatavad (Curcio et al., 2018, lk 33).
3. Dokumenteerimise etapis pannakse kirja kõik nõuded (kui üldine informatsioon), mille alusel täpsustatakse ära funktsionaalsed ja mittefunktsionaalsed nõuded[3] (Curcio et al., 2018, lk 33). Nõuete kirjapanek on sealhulgas vajalik nii nende valideerimiseks kui ka konfliktide lahendamiseks osanike vahel (Kaur & Singh, 2010, lk 30).

[3] Analüüsifaasis defineeritud nõuded lähevad edasi disainifaasi, kus nad omandavad tehnilisema vormi, i.e. süsteeminõuded (Dennis et al., 2012, lk 107). BABOK juhendis (2015, lk 19) tuuakse välja, et kui nõuded peegeldavad vajadust, siis disain on fokuseeritud lahendusele, kuid erinevus nende kahe vahel ei ole alati selge.

4. Kõige lõpus toimub valideermine ehk kontrollitakse, et kas nõuded on täielikud ja et kas need rahuldavad kliendi vajadusi (Curcio et al., 2018, lk 33).

Eelmainitud etapid moodustavad kõigest ühe võimaliku (teoreetilise) lähenemise; etappide arv ning konkreetne sisu võib sõltuda konkreetsest raamistikust. Nt BABOK juhendis (2015, lk 134) tuuakse välja tegevused nagu (1) nõuete määratlemine ja modelleerimine, (2) kinnitamine, (3) valideerimine, (4) nõuete arhitektuuri defineerimine, (5) lahenduse alternatiivide leidmine ning (6) potentsiaalse väärtuse ja lahenduse väljapakkumine.

Antud tegevuste toetamiseks tuleks rakendada ka nõuete halduse protsessi, et hallata muutusi kõikides erinevates faasides[4] (Curcio et al., 2018, lk 33). Tüüpilised muudatused spetsifikatsioonides on nõuete lisamised, eemaldamised või paranduste tegemised; kuid haldamine kui selline ei tähenda ainult dokumendihaldust, vaid ka seda, et ollakse valmis tegelema muutuvate nõuetega pidevas andmete kogumise faasis (Kaur & Singh, 2010, lk 30).

Mis puudutab andmete kogumist ja üleüldse tervet tarkavaraarendusprotsessi, siis olulisel kohal on siin osanike osalus, sest Wiegers (2000, viidatud Belfo, 2012, lk 312 kaudu) järgi ollakse alles siis võimelised looma süsteemi, mis vastab osanike vajadustele. Ehk kui osanikud kaasatakse projekti, siis annab nõuete väljaselgitamise protsess paremaid tulemusi (Kujula et al., 2005, viidatud Wiess et al., 2012, lk 1103 kaudu) ning ühtlasi suurendab see ka süsteemi edukust (Tessem, 2017, lk 245).

Oluline ei ole samas mitte ainult osalus, vaid selle osaluse kvaliteet. Nt Appan ja Browne (2012) toovadki konkreetsemalt välja, et nõuete väljaselgitamise protsessi edukus sõltub kasutajate ja osanike võimekusel kommunikeerida õigeid lahendusi. Ja nõuete õigsus sõltub analüütiku ja kasutaja vahelise kommunikatsiooni efektiivsusest (Hanssen & Faegre, 2008, viidatud Appan & Browne, 2012 kaudu).

Kuidas aga kommunikeerida ‘õiget’ lahendust? Antud asjaolu viibki järgmise teemani, milleks on probleemid või siis väljakutsed, mis võivad nõuete väljaselgitamise etapis esineda. Probleemid on üldiselt kas inimese-, protsessi- või dokumentatsioonipõhised, nt osaniku puudulikud teadmised äristrateegiast, vääritimõistmised omavahelises suhtluses või kirjavead dokumentatsioonis (Alshazly et al., 2014, lk 517-518).

[4] Traditsioonilise koskmudeli puhul viiakse nõuete kogumine läbi konkreetses tarkvaraarenduse faasis (Sommerville, 2001, viidatud Curcio et al., 2018, lk 33 kaudu). Agiilse mudeli puhul ei ole tegevused järjestikused, vaid iteratiivsed ning viiakse läbi igas arendustsüklis (Lucia & Qusef, 2010).

Paljud analüütikud väidavad, et nõuete kogumine on nende peamine tööülesanne, mis on aga vale, sest tegelikult kasutajatel ei ole neid (Blais, 2012, lk 201-203), st et kasutajad ei ole üldiselt võimelised artikuleerima nõudeid, mis peegeldavad nende ärilisi vajadusi (Przybylek, 2013; Kaur & Singh, 2010, lk 28), neil võivad puududa teadmised või vastavad kogemused (Blais, 2012, lk 199). Uurimise eesmärgiks ei olegi seega osanikelt nõuete väljapärimine, vaid informatsiooni kogumine probleemi või lahenduse kohta, mida seejärel analüüsida, et lahendus ise välja pakkuda ning nõuded identifitseerida (Blais, 2012, lk 199).

Omaette probleem on kogutud informatsiooni tõlgendamine. Analüütikutel ja osanikel on erinevad taustad, ühel on teadmised IT-süsteemidest ning teisel äridomeenidest (Appan & Browne, 2012), seega erinevad mentaalsed mudelid ning eesmärgid võivad põhjustada üksteisest valesti arusaamist ning informatsiooni valesti tõlgendamist (Bednar & Welch, 2008, viidatud Appan & Browne, 2012 kaudu).

Appan ja Browne (2012) toovad veel välja, et nõuete väljaselgitamise protsess on ühtlasi haavatav ka tahtlikult edastatud väärinformatsiooni osas, sest antud protsess on suures osas analüütikute kontrolli all. See, kuidas kasutajad oma sõnumeid formuleerivad, sõltub nende võimekusest tuletada meelde olulist ja asjakohast informatsiooni ning see sõltub omakorda sellest, kuidas analüütik vestlust juhib (Appan & Browne, 2012). Analüütikud suunavad kasutajate tähelepanu olulistele ja kriitilistele probleemidele (Alvarez, 2002, viidatud Appan & Browne, 2012 kaudu) ja seega on täiesti võimalik, et nad võivad tahtlikult või tahtmatult esitada väärat või eksitavat informatsiooni, millest tulenevalt võivad omakorda kasutajad esitada ebatäpseid nõudeid[5] ning need võivad jõuda ka nõuete dokumenti ning lõpuks ka süsteemi disaini (Appan & Browne, 2012).

Kasutajate ja analüütikute ‘vale’ kommunikatsiooni kõrval võib probleemiks kujuneda ka kontekst, kus see kõik aset leiab. Przybylek (2013) kirjutab nt, et äriprotsesside ja neid toetavate infosüsteemide joondamisel (ühildamisel) on kaks suurt takistust: (1) süsteeme ei arendata välja konkreetse arusaamisega sellest ärist, mida need peavad toetama, ja (2) äriprotsessid ei ole süsteemi nõuetega seotud ja seega arenevad need üksteisest eraldi. Nõuete väljaselgitamisel keskendutakse Hochmüller (1997, viidatud Belfo, 2012, lk 311 kaudu) järgi tavaliselt objektidele, funktsioonidele ja seisunditele, mis ei ole aga piisav ehk arvesse tuleb võtta ka sellised faktoreid nagu majandus,

[5] Pigem informatsiooni, mille alusel toimub nõuete formuleerimine.

aeg, kasutajate sooritus (Belfo, 2012, lk 311) ja terve see organisatsiooniline ja sotsiaalne kontekst, kus süsteem lõpuks kasutusele võetakse (Kaur & Singh, 2010, lk 31).

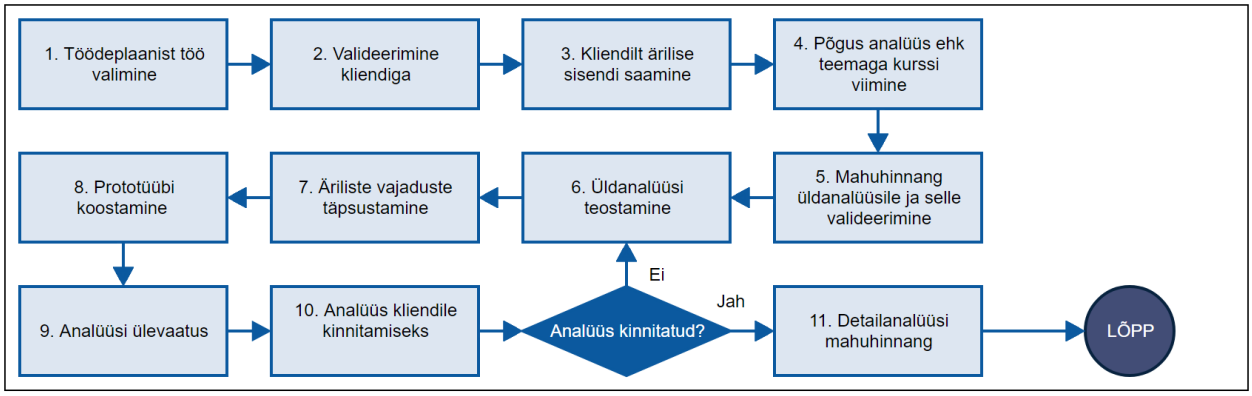
Teiste probleemide kõrval, kui nõuded on identifitseeritud, võib olla ka väljakutse selekteerida nõuete kandidaatide hulgast välja need ‘õiged’, arvestades seejuures kõikide huvidega, tehniliste piirangutega, osanike eelistustega ning ärilise väärtusega (Ruhe, Eberlein & Pfahl, 2002, viidatud Lehtola et al., 2009, lk 114 kaudu). Bohem (2003, viidatud Lehtola et al., 2009, lk 114 kaudu) kirjutab, et paljud tarkvaraarenduse praktikad viiakse läbi neutraalses keskkonnas, kus iga nõuet koheldakse võrdväärselt.

Curcio et al. (2018, lk 41-42) toovad veel lisaks välja probleemid nagu motivatsiooniküsimused (nt motivatsioon hoida dokumentatsioon pidevalt ajakohasena), osaniku kättesaadavus, puudulik dokumentatsioon agiilses protsessis jne.

Lõpetuseks, nõuete väljaselgitamise protsessi tulemuseks on tarkvara nõuete spetsifikatsiooni dokument (Herlea, 2002, viidatud Wiess et al., 2012, lk 1101 kaudu). Sõltuvalt selle kasutamise eesmärkidest võib selle ametlikkuse määr olla erinev (st formaalsest nõuete kirjeldusest tavakeeleni välja) (Bjarnasona et al., 2016, lk 61), kuid kvalitatiivne nõuete spetsifikatsiooni dokument on teostatav, oluline, selge, kõikehõlmav ja kõigi osanike poolt aktsepteeritud (Wiess et al., 2012, lk 1101). Antud teoreetilisi teadmisi saab käesoleva töö autor kasutada taustainformatsioonina analüüsiprotsessis, sh arvestada erinevate võimalike probleemidega, mis võivad nõuete väljaselgitamisel esineda.

2. Analüüsiprotsess SFOSi projektis

Käesolevas peatükis antakse ülevaade sellest, kuidas valmib nõuete spetsifikatsiooni dokument SFOSi projektis. Protsess algab üldanalüüsiga (vt Joonis 1), kus luuakse ja piiritletakse üldine visioon, kuidas kliendi ärilist probleemi lõppkokkuvõttes tarkvaraliselt lahendada. Analüütik saab kliendilt sisendi, töötab selle läbi ja saab temaga seejärel kokku (füüsiliselt või virtuaalselt), et täpsustada olemasolevaid ja uusi detaile. Antud protsess toimub tsüklis, kus iga iteratsiooniga minnakse järjest sügavamale, kuni kõik asjaolud on kaardistatud ja kirja pandud.



Joonis 1. Üldanalüüsi protsess.

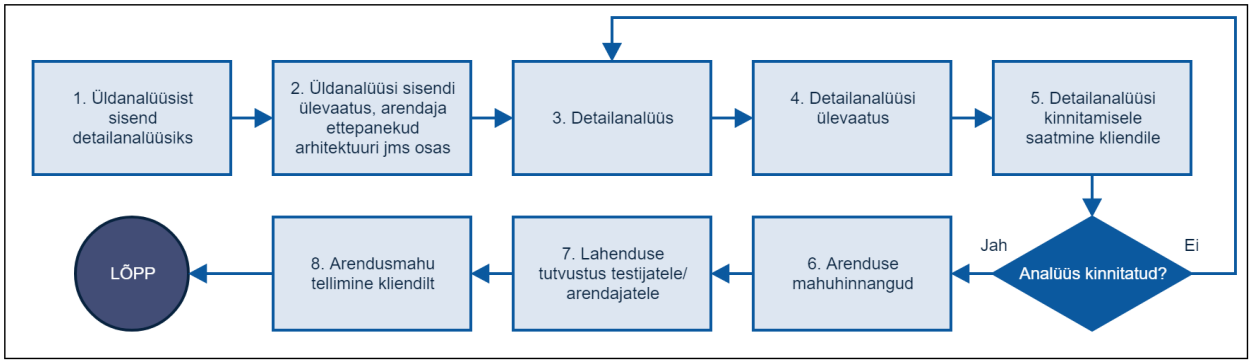
Üldanalüüsi eesmärk ei ole laskuda detailidesse – nii võib töömahtusid valesti hinnata, sest probleemile saab üldjuhul läheneda mitmel erineval moel ning tavaliselt tuleb töö käigus ka erinevaid nüänsse välja. Nt kui detailanalüüsis on konkreetselt kirjas, et ‘kustuta’ nupu vajutamisel kontrollitakse üht ja teist tingimust, siis üldanalüüsis on välja toodud, et antud vormil või kuval[6] peab olema kirjete kustutamise võimalus

Kui üldanalüüs on valmis, enamasti prototüübi ja lahenduse üldisema kirjelduse näol, siis kooskõlastatakse see kliendi ning vajadusel ka lõppkasutajatega. Kui on vaja sisse viia parandusi või täiendusi, siis korratakse protsessi, vastasel juhul saab selle aga kinnitada ning edasi liikuda detailanalüüsi juurde.

Detailanalüüsi puhul on protsess suhteliselt sarnane (vt Joonis 2). Alusmaterjaliks võetakse üldanalüüsi tulemused, vajadusel päritakse kliendilt lisainformatsiooni, ning hakatakse antud punkte analüüsima sügavamalt, st süsteemi tasemel. Kui analüütik on detailianalüüsi lõpetanud

[6] Kasutajale kuvatud vaade rakenduses.

ning klient on andnud tööle positiivse ja kinnitava tagasiside, siis on see kui nõuete spetsifikatsiooni dokument valmis.



Joonis 2. Detailanalüüsi protsess.

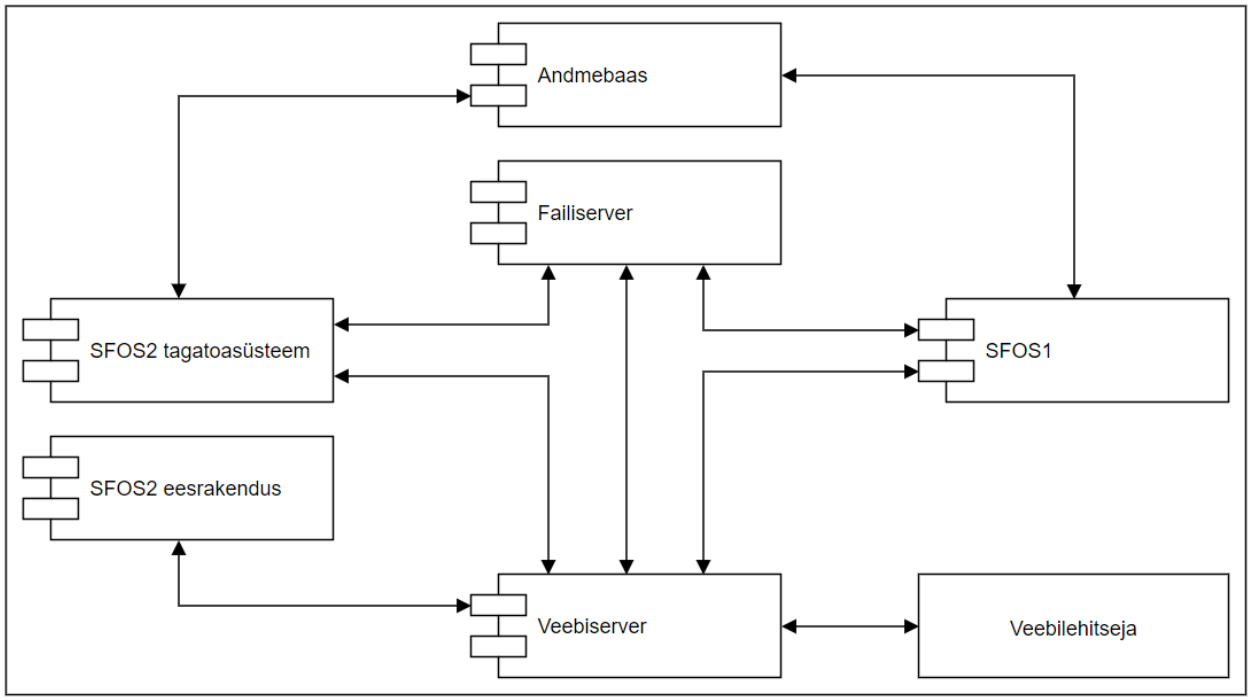
Analüüsiprotsess ei ole aga üld- ja detailanalüüsiga veel lõppenud, sest tavaliselt tekib arendajatel ning testijatel mitmeid küsimusi (nt arenduse käigus tuleb erinevaid nüänsse välja) ja analüütik peab seda protsessi toetama, teatuid asjaolusid täpsustama. Ehk analüüs lõppeb tegelikult alles siis, kui arendus on tehtud ja tootestatud. Selleks, et alustada konkreetse tarkvaramooduli nõuete analüüsiga, tuleb aru saada rakenduse ülesehitusest ja antud mooduliga soetud komponentidest, ning sellest annab ülevaate järgmine peatükk.

3. Rakenduse kirjeldus

Struktuuritoetuse registri operatiivsüsteem ehk lühidalt SFOS on elektrooniline keskkond, mis võimaldab andmesisestust Euroopa Liidu struktuuritoetusest abi taotlemisel. Süsteem koosneb kahest poolest: (1) ametnike pool, kus Rahandusministeeriumi ja teiste täitevasutuste ametnikud sisestavad ja töötlevad taotlusi, makseid jne; (2) taotlejate pool, kus viimased saavad ise sisestada taotlusi ning paranduspalveid (RIHA, 2018). Käesolevas peatükis antakse ülevaade süsteemi arhitektuurist, kirjeldatakse dokumendimoodulit ja failide haldamisega seotud komponente, ning tuuakse välja mõningad piirangud, millega peab süsteemi arendamisel arvestama.

3.1 Süsteemi arhitektuur

Tehnoloogilises mõttes koosneb süsteem erinevatest moodulitest, mis omavahel suhtlevad (vt Joonis 3). Moodul SFOS1 on 2003. aastal loodud SFOSi tagatoasüsteem, millel on ühine andmebaas SFOS2-ga (andmete talletamiseks ja pärimiseks). SFOS2 tähistab 2015. aastal loodud eesrakendust ning tagatoasüsteemi[7], mis pakub esimesele veebiserveri kaudu REST-teenuseid.



Joonis 3. Rakenduse moodulid (lihtsustatud mudel, st teave serverite, autentimise jne kohta on välja jäetud).

[7] Eesmärk on SFOS1 aja jooksul täielikult SFOS2 vastu välja vahetada; hetkel töötavad mõlemad rakendused.

Veebiserveri vahendusel suhtlevad kliendiga SFOS1, SFOS2 eesrakendus ning failiserver kui väline teenus. Viimase puhul on tegemist 2017. aastal loodud veebirakendusega (arendajaks kolmas osapool), kus hoitakse infosüsteemi laetud dokumendi- ja binaarfaile. Lisaks kasutatakse süsteemi poolt ka teisi väliseid teenuseid, nt äri- ning riigihangete registrit andmete pärimiseks.

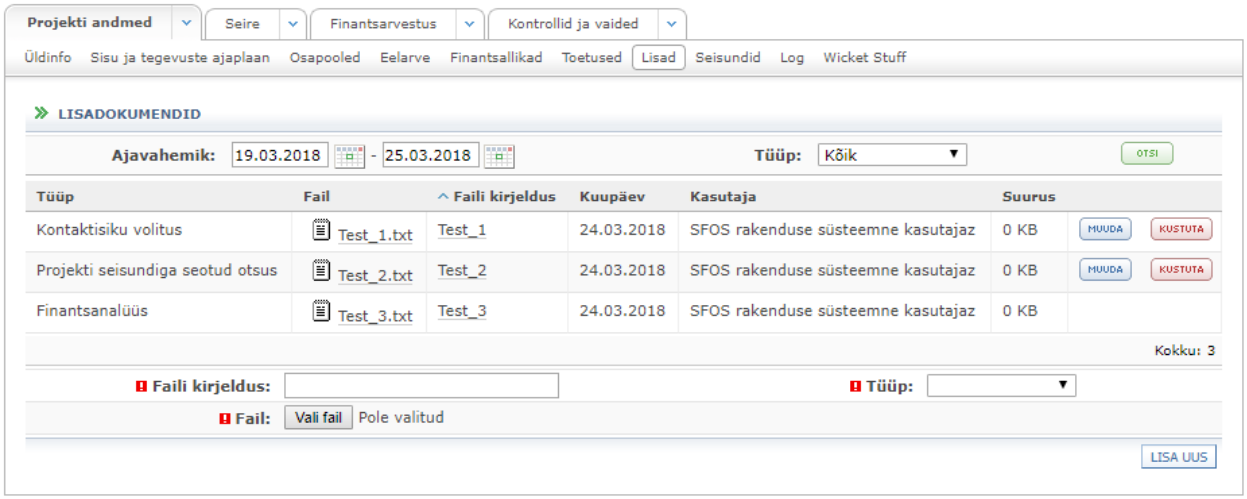
Tervet keskkonda saab mooduliteks jagada ka äriloogilises võtmes, st süsteem koosneb eraldiseisvatest, ent omavahel seotud äriloogilistest osadest, kus iga osa täidab konkreetset eesmärki ja funktsiooni. Nt projekti moodul tähistab tervet taotluse sisestamise protsessi, mille juurde kuuluvad alammoodulid nagu üldandmed, eelarve, hanked ja lepingud, rahastajad jne. Teiste moodulitena võib veel välja tuua nt projektide hindamise, taotlusvoorude haldamise, riigihangete kontrollimise jne. Selline modulaarne ülesehitus ja lähenemine väljendub konkreetselt tarkvara koodi struktureerituses ja üleüldises haldamises.

3.2 Dokumendimoodul ja failide haldamine

Käesoleva bakalaureusetöö fookuses on dokumendimoodul kui kuva, kus on välja toodud kõik projektiga seotud dokumendid. Antud mooduli arendustöödega on seotud ka failide haldamise loogika (üles- ja allalaadimine, muutmine, kuvamine) kui üldkasutatav tarkvarakomponent, mida kasutavad ka teised süsteemi moodulid. Järgnevalt kirjeldataksegi neid komponente, mis on seotud antud töö nõuete väljaselgitamise protsessiga (fookuses on SFOS2).

3.2.1 Projekti lisadokumendid

Projekti lisadokumendid on dokumendimoodul SFOS1-s (vt Joonis 4); SFOS2-s see puudub.



Joonis 4. Projekti lisadokumentide kuva SFOS1-s.

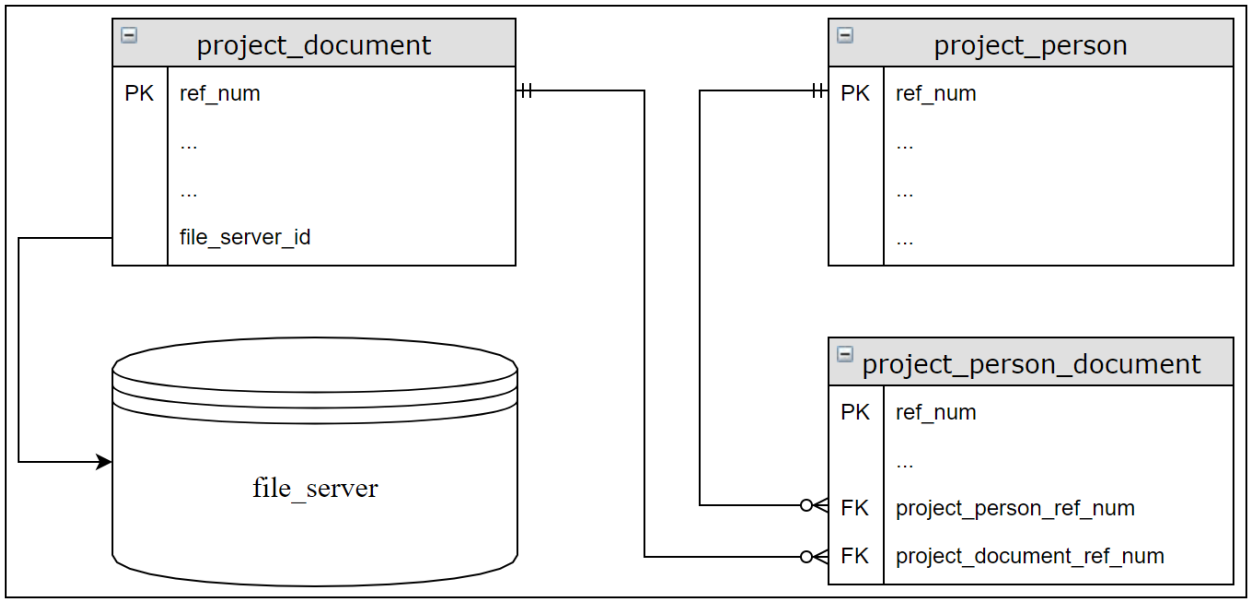
Antud moodulis olevas tabelis on iga faili kohta välja toodud tüüp, nimetus, kirjeldus, lisamise kuupäev, lisaja nimi ning faili suurus. Sõltuvalt tingimustest saab faile kustutada ning muuta nende kirjeldust ja tüüpi. Tabeli kirjeid on võimalik lisamise kuupäeva ning tüübi järgi filtreerida.

3.2.2 Faili salvestamise tehniline pool

Projekti tasemel on faili salvestamisega seotud kolm andmebaasitabelit, mida käesolevalt võib illustreerida mooduli ‘projekti kontaktisikud’ näitel:

1. projekti failid (project_document);
2. projekti alammoodul (project_person);
3. projekti alammooduliga seotud failid (project_person_document).

Projektiga seotud failide metainformatsioon (nimetus, kirjeldus jne) on salvestatud maha project_document tabelisse. Iga fail kuulub konkreetse mooduli juurde ning selle identifitseerimine käib läbi vahetabeli, kus on viited project_document ning antud mooduli andmemudeli primaarvõtmetele (vt Joonis 5).



Joonis 5. Andmebaasitabelite vahelised seosed.

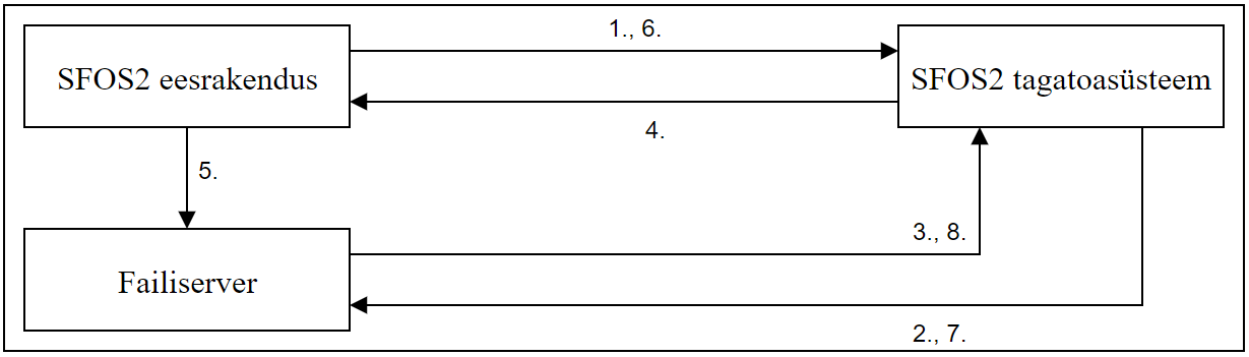
Faili üleslaadimisel ja salvestamisel lisatakse kirjed nii project_document tabelisse kui ka vahetabelisse; faili kustutamisel need kirjed üldjuhul[8] eemaldatakse. Siinkohal on oluline välja

[8] Sõltuvalt äriloogikast ei ‘kustutata’ faile alati läbi kirjete eemaldamise, vaid ka vastavate kirjete muutmise läbi (nt lõppkuupäeva määramise abil), milles tulenevalt eksisteerib antud juhtudel ka failide taastamise võimalus.

tuua, et iga andmebaasitabeli kohta on olemas ka eraldi ajalootabel (nt project_document_history), kus säilitatakse informatsioon kõikide (sh kustutatud ja muudetud) kirjete kohta.

Tabelis project_document on atribuut file_server_id, mis viitab failiserveris olevale failile – antud fail paikneb väljaspool süsteemi täiesti tavalisel kettal ja sellele ligipääsemiseks kasutatakse failiserveri poolt pakutavaid avalikke ja privaatseid teenuseid. Privaatsed teenused pole kättesaadavad lõppkasutajale ning neid kasutatakse ainult suhtluseks rakenduse ja failiserveri vahel. Avalike teenuseid kasutab lõppkasutaja faili üles- ja allalaadimiseks.

Faili üleslaadimise ja salvestamise protsess on järgmine (vt Joonis 6):



Joonis 6. Faili üleslaadimise ja salvestamise protsess.

(1) Klient saadab SFOS2 rakendusele päringu faili üleslaadimiseks. (2) SFOS2 tagatoasüsteem kutsub seejärel välja privaatse failiserveri teenuse, (3) mis tagastab unikaalse räsi failiserverisse faili üleslaadimiseks ning (4) see saadetakse omakorda tagasi kliendile. (5) Kasutades genereeritud avalikku URLi laadib klient faili üles. (6) Kui kuva salvestatakse, siis saadab klient SFOS2 rakendusele eelnevalt failiserverisse üles laetud faili räsi. (7) Kliendi poolt saadetud räsi abil otsib SFOS2 tagatoasüsteem failiserverist vastava faili idendifikaatori ning (8) salvestab selle endale andmebaasi.

3.2.3 Faili üleslaadimine eesrakenduses

Faili üleslaadimiseks kasutatakse üle terve eesrakenduse üht taaskasutatavat komponenti (vt Joonis 7); samas on süsteemis mooduleid, kus on tulenevalt äriloogikast kasutusel ka erilahendused. Iga faili kohta on välja toodud nimetus, tüüp (sõltuvalt tingimustest), kirjeldus, lisamise kuupäev, lisaja nimi ning faili suurus. Tüübi ja kirjelduse määrab lõppkasutaja ise, ülejäänud väljad täidetakse süsteemi poolt automaatselt. Sõltuvalt mooduli ärilistest tingimustest saab faile muuta ja kustutada.

Joonis 7. Failide lisamise, muutmise ja kustutamise tarkvarakomponent.

3.3 Piirangud

Igasugusele arendusele võib sätestada teatud piiranguid nii äriloogika kui ka tehnoloogilised võimalused. SFOSi projektis tuleb arvestada nt sellega, et eraldiseisvad moodulid ei tohi üksteist mõjutada, nt andmete salvestamine projekti eelarve alammoodulis ei tohi muuta andmeid taotlusvooru tasandil – peab säilima modulaarsus, mis on vajalik terve süsteemi paremaks haldamiseks tehnoloogilisel tasemel.

Piiranguid panevad peale ka tehnoloogilised raamistikud, nt kui eesrakenduses kasutada kuval dünaamilist tabelit, kus jälgitakse[9] muutusi igas tabeli kirjes, siis tuleb arvestada asjaoluga, et kümme tuhat kirjet võib põhjustada süsteemile liiga suurt koormust – sellistel juhtudel tuleb esitada ka äriloogiline küsimus, et kas on reaalne võimalus, et andmete hulk võib üldse nii suur olla.

Tihti võib probleemi allikaks olla ka veebilehitseja. SFOSi projektis toetatakse ametlikult kolme erinevad lehitsejat ning kui üks neist teatud lahendust ei toeta (mingi funktsioon ei tööta viisil nagu see peaks töötama), siis tuleb leida alternatiiv.

Mõnes mõttes võib piirangute alla paigutada ka ‘ühtse stiili’ jälgimise nõude. Nt kui ‘salvesta’ nupp peab olema rohelist värvi ning paiknema all paremas nurgas, siis peab ka igasugune tehnoloogiline lahendus ka sellega arvestama. Sõltuvalt vajadustest rakendatakse vahel ka erijuhtumeid, kuid kui nende maht kasvab liiga suureks, siis võib rakenduse ühtne voog ja struktureeritus lõpuks kaotsi minna.

Piiranguid võib välja tuua veel ja veel, nt tarbitavad teenused, mis on väljaspool SFOSi projekti (ei ole võimalik kätte saada vajalikke andmeid vajalikus formaadis), kuid igasugune piirang sõltub lõpuks eelkõige konkreetsest lahendusest ja arendusest.

[9] JavaScript-i raamistiku AngularJS-i funktsionaalsus.

4. Nõuete väljaselgitamine

Dokumendimooduli arenduse eesmärgiks on (1) realiseerida projekti lisadokumentide kuva SFOS2-s ning (2) muuta kasutaja jaoks failide haldus mugavamaks ja minimeerida olukordi, kus üht ja sama faili laaditakse üles mitmeid kordi. Käesolevas peatükis antakse ülevaade kliendi vajadustest, teostatakse nõuete väljaselgitamise analüüs ning kirjutatakse välja analüüsi tulemuste kokkuvõte ehk nõuete spetsifikatsioon.

Kliendi vajadused kaardistati ära 2017. aasta novembris SFOSi projekti analüütiku poolt (käesoleva töö autor selles protsessis ei osalenud) ning need on välja toodud töö lisades (vt Lisa 1). Fookuses on dokumendimoodul ning failide haldamise loogika, kus olemasolev funktsionaalsus jääb kõik alles, kuid vaja on juurde lisada uusi võimalusi. Konkreetselt tõi klient välja, et ühele failile peab saama määrata mitut tüüpi; uue faili lisamisel peab olema võimalus laadida üles täiesti uus fail või valida üks olemasolevate seast; iga faili juurde peaks tekkima ajalookomponent (versioneerimine); ning lisaks tuleks uurida ka failide tuvastamise funktsionaalsust, selle võimalusi ja piiranguid. Ühtlasi soovib klient SFOS2-te uut moodulit, kus oleks kuvatud kõik projektiga seotud failid.

4.1 Nõuete analüüs

Nõuete väljaselgitamise protsess on järgmine: töö autor võtab ette kliendi vajaduse, analüüsib selle läbi ning pakub välja võimalikud lahendused. Seejärel arutatakse iga probleem läbi SFOSi analüütikuga ning tulemuseks on punkt nõuete spetsifikatsioonis. Antud protsess toimub iteratiivselt iga vajaduse/nõude kohta.

4.1.1 Ühele failile mitme tüübi määramine

Igale projektiga seotud failile saab vajadusel määrata konkreetse tüübi[10], kuid kõigest ühe, ning klient soovib, et saaks määrata mitu. Tehnoloogilisel tasemel märgitakse faili tüüp ära project_document tabelis (vahetabelites on ainult seosed), mis tähendab seda, et praegune andmemudel sätestab teatud piirangu. Selleks, et ühe failiga saaks ühe project_document kirje

[10] Nt hindamise moodulis võib hindamise lähtematerjali kui dokumendi tüübiks olla sh ‘Hindamise juhend’ või ‘Hindamisleht’.

lõikes siduda rohkem tüüpe, oleks üheks võimalikuks lahenduseks uue vahetabeli loomine, nt project_document_type, mille andmemudel oleks järgmine (vt Tabel 1):

Tabel 1. Faili tüübi vahetabel.

Atribuut	Kirjeldus
REF_NUM	Objekti unikaalne identifikaator (primaarvõti)
TYPE	Dokumendi tüüp, seotud klassifikaatoriga
PROJECT_DOCUMENT_REF_NUM	Viide projekti failidele (objekt PROJECT_DOCUMENT)

Faili metaandmete hoidmiseks kasutatakse üle terve rakenduse, nii eesrakenduses kui ka tagatoasüsteemis, ka teisi andmeobjekte, nende töötlemiseks mitmeid erinevaid klasse ja meetodeid – eksisteerivad geneerilised komponendid, kuid teatud moodulid nõuavad ka erilahendusi. Olemasolev tehniline lahendus on koodi mõttes üles ehitatud sellele, et iga faili kohta saab maksimaalselt määrata ainult ühe tüübi. Uue andmemudeli lisamine nõuaks suurejoonilisemate muudatuste tegemist nii eesrakenduses kui ka tagatoasüsteemis, sh ka SFOS1-s, sest andmebaas on mõlemal süsteemil üks ja see sama. Ühtlasi tuleks läbi viia ka andmete migreerimine.

Visuaalsed muudatused (see, mida lõppkasutaja näeb) puudutaksid ainult faili tüübi määramist, kus kasutajal oleks võimalus valida mitut erinevat tüüpi – tehniline funktsionaalsus on selleks geneeriliste komponentide näol juba olemas.

Teine lahendus, kuidas ühele failile mitut tüüpi määrata, seisneb topeltkirjete tekitamises, st lisatakse project_document tabelisse ühe faili kohta nii mitu kirjet kui on tüüpe – metaandmed oleksid üldjoontes samad, sh failiserveris oleva faili identifikaator, ainsaks erinevuseks olekski tüüp. Antud lahendus ei nõuaks suurejoonilisemate muudatuste tegemist ja rakenduses saaks jätkata olemasoleva loogika kasutamist – projekti lisadokumentide vaates ning faili kuvamise komponendis tuleks ainult ühed ja samad failid tüüpide lõikes kokku grupeerida.

Analüütikuga antud lahendusi läbi rääkides sai selgeks, et eesmärk ei ole siin mitte ühes moodulis ühele failile mitme tüübi määramine, vaid et erinevates moodulites oleks võimalik kasutada üht ja seda sama failiserveris olevat faili. Põhimõtteliselt ei lähe siinkohal käiku kumbki lahendus, sest topeltkirjeid kui selliseid tegelikult ei tekigi. Tehnoloogilises võtmes on siin eesmärk luua võimalus lisada project_document tabelisse kirjeid selliste failiserveri identifikaatoritega, mis on varem juba

süsteemi sisestatud – antud võimaluse realiseerimine leiab aset kliendi vajaduse ‘uue faili lisamisel peab olema võimalus laadida üles täiesti uus fail või valida üks olemasolevate seast’ juures.

4.1.2 Faili üleslaadimise funktsionaalsuse täiendamine

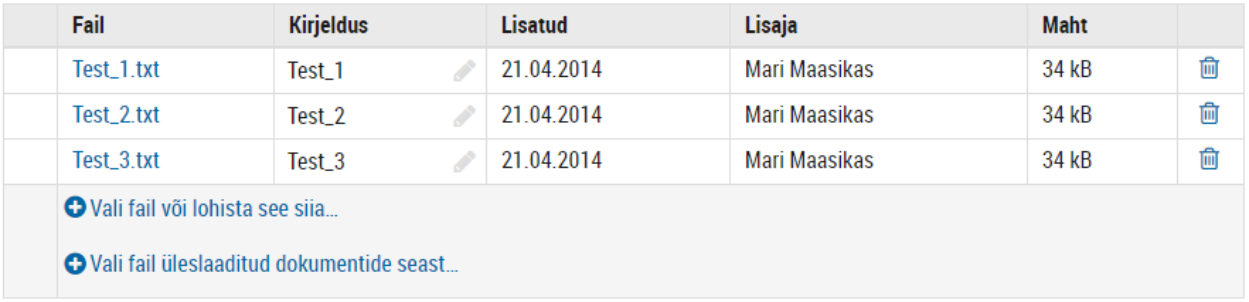
Eesmärgiks on luua olukord, kus kasutajal oleks võimalik valida, et kas ta soovib üleslaaditava faili[11] valida olemasolevate seast või laadida üles täiesti uue faili – hetkel on rakenduses realiseeritud ainult viimane funktsionaalsus, st uue faili lisamisel kuvatakse kasutajale veebilehitseja dialoogakent, kus viimane saab enda süsteemist (arvutist) vajaliku faili välja valida. Selleks, et kasutaja saaks faili valida olemasolevate seast, tuleks luua eraldi kuva või modaalne aken, kus oleks tabeli formaadis välja toodud kõik projektiga seotud failid. Iga faili kohta oleks välja toodud:

1. tüüp;
2. nimetus;
3. kirjeldus;
4. lisamise aeg;
5. faili maht/suurus;
6. selekteerimise kast.

Selekteeritud failid lisatakse failide üleslaadimise tarkvarakomponendi tabelisse (vt Joonis 7, lk 17) (üheskoos uute failidega), kus kasutajal on võimalik modifitseerida nende metaandmeid, nt muuta tüüpi või kirjeldust. Kasutaja arvutist valitud failid salvestatakse maha kasutades olemasolevat lahendust. Olemasolevate failide salvestamiseks ‘uutena’ (st failiserverisse uut faili ei lisata, kuid tekitatakse kirjed project_document ja vastava mooduli failide vahetabelisse) tuleb luua uus tehniline lahendus, mis mõjutab komponente nii eesrakenduses kui ka tagatoasüsteemis.

Analüütikuga antud lahendust läbi rääkides tekkis küsimus ainult versioneerimise osas – kui ühel konkreetsel failil on olemas ajalugu, siis peab olemasolevate failide tabelis olema kuvatud selle faili viimane versioon, ja kui tekitatakse uus entiteet (st uus kirje project_document tabelis), siis ei tohi ‘vana’ faili ajalugu ‘uuega’ kaasa minna. Lahenduse visuaalse poole (vt Joonis 8) eest vastutas ettevõtte UX-meeskond – võrreldes varasema versiooniga (vt Joonis 7, lk 17) lisandusid andmekirjete tabeli alla lingid failide üleslaadimiseks.

[11] Fail, mida soovitakse siduda konkreetse mooduliga.



Joonis 8. Faili lisamise prototüüp.

Ehk lõplikus lahenduses on kasutajale kuvatud kaks linki: 'Vali fail või lohista see siia' ja 'Vali fail üleslaaditud dokumentide seast'. Viimase variandi puhul viiakse kasutaja uuele kuvale, kus paikneb tabel kõikidest projektiga seotud failidest.

4.1.3 Faili versioneerimine ja kustutamine

Äriline probleem on järgmine: taotluse sisestamise protsessis lisab kasutaja taotlusele faili (versioon 1); taotluse menetleja laeb selle alla, täiendab seda (versioon 2) ja saadab selle taotlejale tagasi. Taotleja nüüd omakorda täiendab seda sama faili (versioon 3), laeb selle keskkonda üles, kuid kustutab selle eelmised versioonid – klient aga soovib, et faili ajalugu säiliks. Oluline on siinkohal märkida, et faili versioon kui selline eksisteerib hetkel ainult kasutajate peas, tehnilises mõttes ei ole seda tegelikult olemas, sest praegune andmestruktuur ei luba faile omavahel siduda. Versioneerimise eesmärgiks on seega luua olukord, kus oleks võimalik vaadata ühe konkreetse faili vanemaid versioone.

Üheks võimalikuks lahenduseks oleks sätestada teatud moodulites failide lisamisele piirangud, st lubada kasutajal hoiustada ühe mooduli lõikes ainult üht faili. Kui üks fail on üles laetud ning kasutaja lisab uue, siis olemasolev 'eemaldatakse', st määratakse sellele andmebaasis lõppkuupäev. Kui antud mooduli kohta päritakse eesrakenduses andmeid, siis tuuakse kohale kõik sellega seotud failid – fail, millel puudub lõppkuupäev, on see 'õige' fail (ehk kõige uuem versioon) ning kõik muud failid on selle vanemad versioonid, sõltumata nimetusest, reaalsest sisust, kirjeldusest vms.

Teine lahendus, mis ei seaks failide lisamisele selliseid arvulisi piiranguid, oleks failide omavahel sidumine – üks variant oleks lisada project_document tabelisse uus atribuut nimetusega parent_document, mille abil oleks võimalik konkreetne fail siduda teisega (parent_document tähistaks alati originaalfaili). Antud lahenduse teine variant oleks luua omaette vahetabel, kus oleks välja toodud esialgne dokument ning tema järglased (vt Tabel 2):

Tabel 2. Failide sidumise vahetabel.

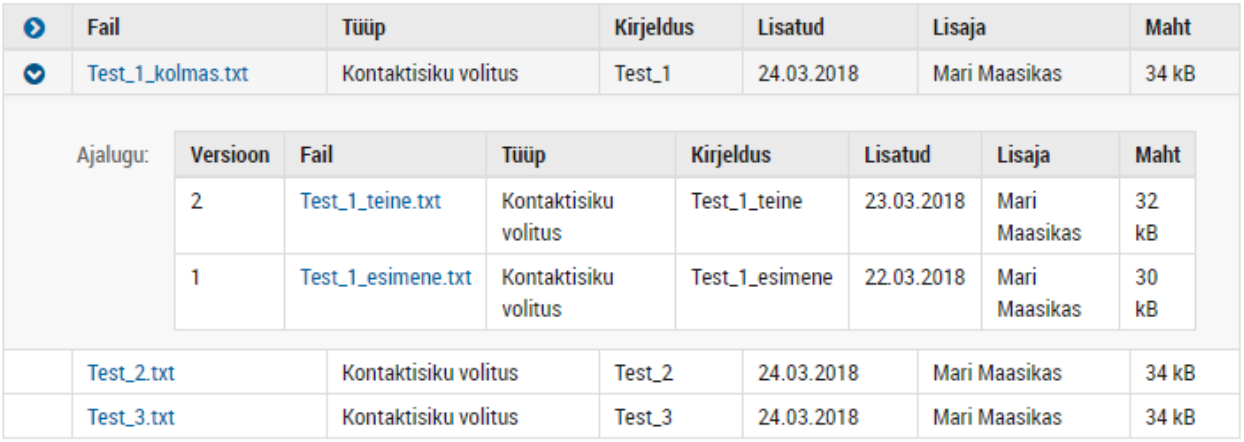
Atribuut	Kirjeldus
REF_NUM	Objekti unikaalne identifikaator (primaarvõti)
DOCUMENT_REF_NUM	Viide projekti dokumentidele (objekt PROJECT_DOCUMENT)
PARENT_DOCUMENT_REF_NUM	Viide projekti dokumentidele (objekt PROJECT_DOCUMENT)

Failide üleslaadimise komponendis kuvatakse eesrakenduses iga kirje puhul eraldi välja võimalust lisada antud failist uuem versioon (vt Joonis 9), mille tulemusel antud kuval olev vana fail asendatakse uuega ning vana fail seega ‘eemaldatakse’.

Joonis 9. Failist uue versiooni lisamise prototüüp.

Kui kasutaja katkestab protsessi, siis taastub endine seis. Kui kasutaja salvestab kuva, siis salvestatakse andmed andmebaasi. Kui kasutaja otsustab faili reaalselt kustutada, siis kustutakse (st eemaldatakse kirjed andmebaasist) ka selle vanemad versioonid ehk kõik seotud failid.

Projekti mooduli ühe kirje/faili kohta saab kõik selle erinevad versioonid kokku grupeerida ja lõppkasutajale kuvada (vt Joonis 10):



Joonis 10. Failide vaatamise komponendi prototüüp.

Antud lahendus nõuaks suurejoonelisemate muudatuse tegemist nii eesrakenduses kui ka tagatoasüsteemis. Oluline on märkida, et see lahendaks ainult versioneerimise probleemi, st et kasutajal oleks ikka võimalik lisada failist uus versioon ilma vana versiooniga sidumata ja seega

vana fail ära kustutada. Selle probleemi saaks lahendada nii, et piirata kasutaja võimalust teatud seisundites või tingimustes faile reaalselt kustutada.

On olemas ka kolmas lahendus. Nt Dropboxis[12] toimub versioneerimine automaatselt, st et kui üleslaetud faili nimi ja tüüp on identne mõne olemasoleva failiga, siis see olemasolev fail asendatakse uuega; vanemaid versioone on võimalik taastada, mille tulemuseks on taas uus versioon failist. Sarnaselt töötavad ka Google Drive[13] ja Microsoft OneDrive[14] failide hoiustamise lahendused. Antud lahendus oleks põhimõtteliselt automatiseeritud versioon käesoleva probleemi teisest lahendusest.

Analüütik jõudis samasugusele järeldusele nagu on teises lahenduses välja toodud – failide omavahel sidumine käib project_document uue atribuudi parent_document kaudu. Lisaks, kustutada saab faile, kuid kustutada ei saa failide vanemaid versioone. Ning samuti ei saa neid taastada – kasutajal on samas võimalus vanem versioon alla tõmmata ning see uue versioonina taas üles laadida.

4.1.4 Faili tuvastamine

Klient sooviks failide tuvastamise funktsionaalsust, st et kui kasutaja laeb faili üles, siis süsteem oskaks seda teiste failidega võrrelda ja väljastada tulemuse, et kas see on juba failiserverisse lisatud. Failide reaalne üles- ja allalaadimine toimub kliendi (eesrakenduse) poole peal, mis tähendab seda, et failide tuvatamine ei saa toimuda SFOSi rakenduses (sest faili sisu tagatoasüsteemi ei jõuagi). Failide tuvastamine saab seega toimuda ainult failiserveris, kus sellised teenused aga hetkel puuduvad – kuna selle arendajaks on kolmas osapool, siis on see ka väljaspool SFOSi skoopi.

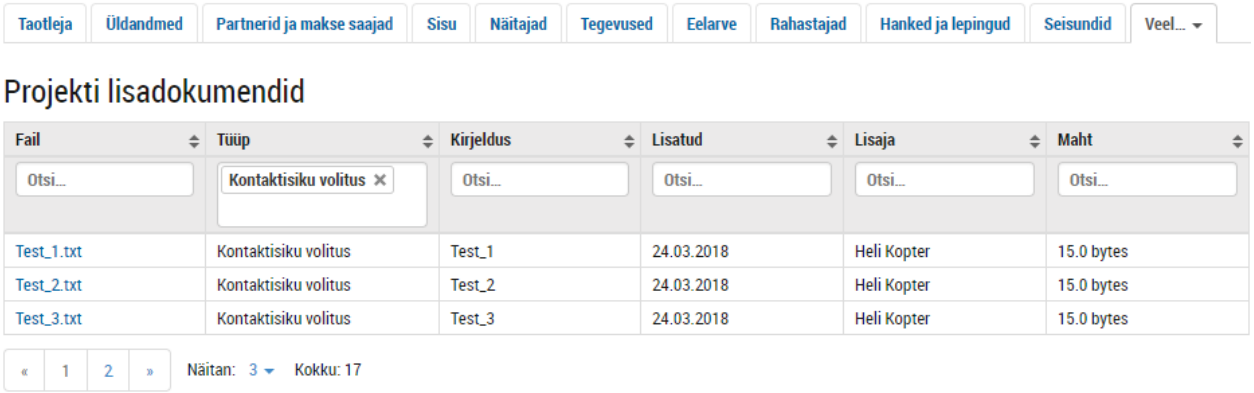
Kui selline tehniline funktsionaalsus oleks failiserveris olemas, siis oleks failide tuvastamise käitumisloogika nt järgmine: kui kasutaja laeb üles faili, mis on juba failiserveris olemas, siis uut faili serverisse ei salvestata, vaid võetakse olemasoleva faili identifikaator. Vajadusel võib ka kasutajat informeerida, et selline fail on juba süsteemis olemas. Kasutajale ei ole aga vajalik anda selles osas valikut, et kas ta kasutab olemasolevat faili või soovib failiserverisse lisada täiesti uue faili, sest (1) failiserverisse jäävad failid alati alles (st kui projekti moodulis fail ära kustutakse, siis vastavad kirjed eemaldatakse küll SFOSi poole pealt, kuid failiserverisse jääb fail alati alles) ning

[12] https://www.dropbox.com/
[13] https://www.google.com/drive/
[14] https://onedrive.live.com/

(2) mahasalvestamised project_document ja projekti komponendi dokumentide tabelisse toimuvad igal juhul.

4.1.5 Projekti lisadokumendid

Eesmärgiks on realiseerida SFOS1-s olev dokumendimooduli lahendus ka SFOS2-s, st luua kuva (vt Joonis 11), kus on välja toodud kõik projektiga seotud failid. Lõppkokkuvõttes on tegemist filtreeritava ja sorteeritava tabeliga, kuhu ei tohi kasutaja saada lisada uusi kirjeid ega muuta või kustutada olemasolevaid.



Joonis 11. Projekti lisadokumentide kuva prototüüp.

Analüütikuga antud prototüübi üle arutades tõi ta veel lisaks välja, et kuna ühte faili (kui failiserveris olevat entiteeti) võib projektis olla kasutusel mitu korda, siis ei tohiks seda kuvada topelt, vaid tüübi ja kirjelduse lõikes grupeeritult.

4.2 Nõuete spetsifikatsioon

Siinkohal tuuakse välja nõuete väljaselgitamise protsessi tulemused ehk nõuded, mille alusel on võimalik läbi viia reaalseid arendustöid. Oluline on märkida, et nõuete spetsifikatsiooni dokument on eelkõige arusaadav ja loetav konkreetses kontekstis – kirjas ei ole projekti üldloogikat, küll aga eksisteerivad lühikirjeldused olemasolevate lahenduste kohta (vajadusel, sõltuvalt äriloogika keerulisuse astmest) ning viited seotud objektidele, sh prototüübile.

4.2.1 Faili üleslaadimine

1. Edaspidi on kasutajal võimalus valida, et kas ta soovib üleslaaditava faili valida olemasolevate failide seast või laadida üles täiesti uue faili; kasutajale kuvatakse kaks linki: ‘Vali fail või lohista see siia’ ja ‘Vali fail üleslaaditud dokumentide seast’.

2. Kui kasutaja valib võimaluse ‘Vali fail või lohista see siia’, siis kuvatakse talle veebilehitseja poolt dialoogaken, et valida enda arvutist õige fail, või siis kasutaja lohistab faili konkreetsele lingile või selle lingiga seotud alale.
3. Kui kasutaja valib võimaluse ‘Vali fail üleslaaditud dokumentide seast’, siis kuvatakse talle vaade, kus paikneb tabel kõikidest projektiga seotud failidest, mille hulgast saab kasutaja vajalikud välja valida.

4.2.2 Failist uue versiooni üles laadimine

1. Kui kasutaja valib võimaluse ‘Lisa uus versioon failist’, siis kuvatakse talle veebilehitseja poolt dialoogaken, et valida enda arvutist õige fail.
2. Failide vaatamise tabelis kuvatakse kasutajale alati kõige viimast versiooni failist – selle varasemad versioonid kuvatakse tabeli rea peidetud informatsioonina.
3. Kui kustutatakse faili viimane versioon, siis kustutatakse ka failiga seotud ajalugu.
4. Kustutada ei saa faili varasemaid versioone.

4.2.3 Projekti lisadokumentide nimekiri

1. Nimekiri koondab kokku kõik projektiga seotud kustutamata failid.
2. Faile ei saa juurde lisada ning olemasolevaid ei saa muuta ega kustutada.
3. Antud nimekiri on sorteeritav ja filtreeritav.
4. Kui üht faili on lisatud mitmest erinevast vaatest, siis ei kuvata seda topelt, vaid tüübi ja kirjelduse lõikes grupeeritult.

Lõpetuseks toob käesoleva töö autor välja, et terves nõuete väljaselgitamise protsessis, sh ka nõuete spetsifikatsiooni kirjutamisel oli suureks abiks ja toeks SFOSi projekti analüütik ning tema tehtud märkmed.

Kokkuvõte

Käesoleva bakalaureusetöö aluseks oli vajadus arendada välja uus tarkvarakomponent, millest tulenevalt tekkis töö autoril võimalus võtta osa selle analüüsiprotsessist. Töö eesmärgiks oli seega ära defineerida SFOSi dokumendimooduli ning olemasoleva failide haldamise komponendi uued nõuded vastavalt kliendi vajadustele, töö tulemuseks olles nõuete spetsifikatsioon.

Nõue on väide selle kohta, mida süsteem peab tegema või mis omadused peavad sellel olema; nõuete väljaselgitamise protsess koosneb andmete kogumisest (*ala* ‘mis on lahendamist vajav äriline probleem’), analüüsimisest, dokumenteerimisest ja valideerimisest. Terve protsessi juures tuleb arvestada erinevate probleemidega, nt kliendi võimekusel kommunikeerida õigeid vajadusi ning analüütiku oskusel saadud informatsiooni õigesti tõlgendada.

SFOSi projektis algab nõuete väljaselgitamise protsess üldanalüüsiga, st analüüsitakse ja täpsustatakse kliendilt saadud sisendeid nii mitu korda, kuni ära on kaardistatud ja defineeritud kõik vajadused ja nõuded. Sellele järgneb detailanalüüs, mis on oma loomult üldanalüüsi sarnane, kuid mida tehakse üksikasjalikumalt, süsteemi tasemel.

Analüüsiprotsessis on olulisel kohal rakenduse ülesehitus, sh selle võimaluste ja piirangutega arvestamine. SFOS on elektrooniline keskkond, mis võimaldab andmesisestust Euroopa Liidu struktuuritoetusest abi taotlemisel. Süsteem koosneb mitmetest komponentidest, antud töö fookuses olles dokumendimoodul kui kuva, kus on välja toodud kõik projektiga seotud failid, ning failide haldamise tarkvarakomponent.

Nimetatud komponentide nõuete väljaselgitamiseks võttis töö autor ette kliendi vajadused ning pakkus iga punkti kohta välja võimalikud lahendused. Olemasolevatele kuvadele ja funktsionaalsustele lisandusid uued võimalused failide üleslaadimiseks ning versioneerimiseks ja ühtlasi valmis prototüüp ning kirjeldus uuest dokumendimoodulist. Antud töö praktilist tulemust ehk nõuete spetsifikatsiooni saab kasutada reaalsete arendustööde läbiviimiseks.

Antud töö andis autorile nägemuse ja kogemuse nõuete väljaselgitamise protsessist, selle juurde kuuluvatest võimalikest probleemidest ja väljakutsetest. Ühtlasi rõhutas see vajadust arendada oskust näha seoseid süsteemi erinevate komponentide vahel – tihti on neid seoseid rohkem ja tihti on need seosed keerulisemad, kui pealtnäha paistab.

Kasutatud kirjandus

Alshazly, A. A. & A. M. Elfatatry & M. S. Abougabal (2014). Detecting defects in software requirements specification. *Alexandria Engineering Journal*, 53, 513-527.

Appan, R. & G. J. Browne (2012). The impact of analyst-induced misinformation on the requirements elicitation process. *MIS Quarterly*, 36(1), 85-106.

Belfo, F. (2012). People, organizational and technological dimensions of software requirements specification. *Procedia Technology*, 5, 310-318.

Bjarnason, E. & M. Unterkalmsteiner & M. Borg & E. Engström (2016). A multi-case study of agile requirements engineering and the use of test cases as requirements. *Information and Software Technology*, 77, 61-79.

Blais, S. P. (2012). *Business analysis: best practices for success*. United States of America: John Wiley & Sons, Inc.

Curcio, K. & T. Navarro & A. Malucelli & S. Reinehr (2018). Requirements engineering: a systematic mapping study in agile software development. *The Journal of Systems and Software*, 139, 32-50.

Dennis, A. & B. H. Wixom & R. M. Roth (2012). *System analysis and design*. United States of America: John Wiley & Sons, Inc.

International Institute of Business Analysis (2015). *A guide to the business analysis body of knowledge*. Canada.

Kaur, R. & T. Singh (2010). Analysis and need of requirements engineering. *International Journal of Computer Applications*, 7(14), 27-32.

Lehtola, L. & M. Kauppinen & J. Vähäniitty & M. Komssi (2009). Linking business and requirements engineering: is solution planning a missing activity in software product companies? *Requirements Eng*, 14, 113-128.

Paul, D. & Y. L. Tan (2015). An investigation of the role of business analysts in IS development. Ettekanne. *Twenty-Third European Conference on Information Systems (ECIS)*. Münster, Germany.

Przybylek, A. (2013). Bridging the gap between business process models and use-case models. Ettekanne. *IEEE International Workshop on Requirements Engineering*. Constantine.

RIHA, URL (kasutatud aprill 2018) https://www.riha.ee/api/v1/systems/sfos/files/c705b79e-8c9b-98a3-2e8a-0e374be71d36.

Rubens, J. (2007). Business analysis and requirements engineering: the same, only different? *Requirements Eng*, 12, 121-123.

Tessem, B. (2017). The customer effect in agile system development projects. A process tracing case study. *Procedia Computer Science*, 121, 244-251.

Wiess, R. & V. Holzmann & M. Frank (2012). The factors affecting the quality of the software requirements specifications in technological projects: a report on a research in progress. Ettekanne. *IFAC sümpoosium – Information Control Problems in Manufacturing*. Bucharest, Romania.

Summary

Requirement analysis of a software component: the case of a document module in SFOS

The purpose of this bachelor thesis 'Requirement analysis of a software component: the case of a document module in SFOS' is to perform a business and system level analysis of the module in question, more specifically, to document its requirements in the form of a requirements specification.

A requirement is simply a statement of what the system must do or what characteristic it must have (Dennis et al., 2012, lk 104); requirements engineering is the process of collecting data from the users after which this data is analysed, documented and validated. In the SFOS project, the requirements engineering process consists both of general and detailed analysis, latter one being more system level oriented.

In order to perform an adequate analysis it is important to understand the system itself. SFOS (Structural Funds Operational System) is a management system for granting European Structural and Investment Funds to applicants in Estonia. The system consists of multiple components, the current thesis focusing on the document module and on the software component that manages file uploads and downloads.

Requirements engineering was carried out in the following fashion: the author selected an item from the list of problems or needs the client had, analysed it and then proposed possible solutions – this was done in iteration. In the requirements specification multiple new functionalities were outlined, i.e. file version control and different options on how to upload a new file to the system; a prototype of a new document module was also presented.

The current thesis gave the author an insight into the world of software component analysis, the problems and challenges it may contain. In addition, it emphasized the importance and need to continuously develop the ability to see the complex relationships between different software modules.

LISAD

Lisa 1: Kliendi vajadused

Kliendi vajadusi sõnastab kliendi esindaja, kes on SFOSi projektiga olnud seotud ca 15 aastat.

- Ühele failile peaks saama määrata rohkem kui üht tüüpi.
- Kas meil on võimalik tuvastada, et tegemist on failiga, mis meil on juba olemas kui uut faili üles laaditakse? Juhud: Laadin üles faili naide.docx ja nüüd muudan vaid failid pealkirja naide1.docx, siis süsteem suudab tuvastada, et tegemist on sisuliselt sama failiga. Või kõige lihtsam näide - ma laadin üles faili naide.docx ja laadin uuesti sama faili, siis süsteem tuvastaks seda. Kui on teostatav, siis analüüsida käitumisloogika.
- Peab tekkima uut faili lisades võimalus määrata, kas ma soovin faili valida olemasolevate failide seast või laadida üles uus fail.
- Faile ei tohi saada kustutada ‘Projekti lisad’ vaatest.
- SFOS kui ka eSFOS poole pealt ei tohiks saada algatada uue faili lisamist ‘Projekti lisad’ vaatest. Antud punkt on SFOS vaatest vajalik läbi rääkida RÜ-dega. Täna on võimalik täiendavaid faile lasta üles laadida lisaväljade abil.
- Failide kustutamine üldises vaates läbi analüüsida ja leida lahendus. Nimelt täna on probleem, kus taotleja lisab faili, RÜ vaatab üle ja täiendab laadib üles faili ning suunab taotluse tagasi taotlejale. Taotleja kustutab eelneva versiooni(d) ja lisab enda faili uuesti. Tegelikult RÜ-d soovivad faili ajalugu säilitada. Üks variant nt. taotleja saab vaid praegu täiendamisel lisatud faile kustutada ja pärast seda kui on korra need failid andnud ülevaatuseks (taotluse uuesti esitanud), siis enam neid faile pole võimalik kustutad.