TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Rainer Mulk 183189IABM

Nõuete haldamise protsessi väljatöötamine konkreetse ettevõtte näitel

Magistritöö

Juhendaja: Jaak Tepandi
Professor

Tallinn 2021

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Rainer Mulk

04.01.2021

Annotatsioon

Käesoleva magistritöö eesmärgiks on nõuete halduse protsessi tõhustamine vaadeldavas ettevõttes. Nõuete haldus peab olema kooskõlas SAFe (*Scaled Agile Framework*) tarkvaraarendusraamistikuga ja toetama agiilsest tarkvaraarendusest tulenevat mõtteviisi terves organisatsioonis. Lisaks peab ühtne nõuete halduse protsess arvestama tarkvaarasüsteemide kogu elutsüklit ning lihtsustama uute süsteemide arendamist ja olemasolevate süsteemide muutmist.

Kvalitatiivse uuringu käigus kaardistatakse esmalt passiivselt hetkeolukord nõuete kirjeldamisel erinevates projektides kaasamata arendusmeeskondi. Kaardistusele lisaks kogutakse arendusmeeskondade liikmetelt intervjuude abil täiendavat infot nõuete halduse tänaste praktikate kohta ning palutakse võimalusel välja tuua teadaolevad kitsaskohad.

Kogutud infole ja probleemkohtadele ning tänapäevaste agiilse arendusprotsessi arengusuundadele tuginedes viiakse läbi analüüs lahenduse leidmiseks. Analüüsi tulemusena pakutakse arendusmeeskondade liikmetele hindamiseks ja valideerimiseks välja neli konkreetset lahendusvarianti nõuete haldusprotsessi toetamiseks. Arendusmeeskondadelt saadud tagasiside põhjal selgitatakse välja sobivaim lahendus, mille põhjal kirjeldatakse terviklik nõuete halduse protsess. Välja töötatud protsess määrab kindla reeglistiku nõuete kirjeldamiseks vastavalt nõude tüübile ja arendusprojekti iseloomule ning rahuldab algselt nõuete protsessile seatud tingimused.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 68 leheküljel, 10 peatükki, 7 joonist, 1 tabel.

Abstract

Development of a requirement management process in the context of a specific company

The purpose of this master's thesis is to make requirements engineering more efficient in a given company. Requirements engineering has to be in accordance with the SAFe (Scaled Agile Framework) framework and support the agile software development mindset throughout the organisation. In addition, requirements engineering have to support the whole life cycle of the software system and help to simplify the modification of existing systems and development of new systems.

First, the current situation is passively mapped through qualitative research. This is done without consulting the development teams. In addition to the mapping, development teams are interviewed to aquire additional information on the company’s current requirement engineering practices and to seek out any known bottlenecks.

Based on the information gathered and modern agile developement processes, an analysis is performed to find a solution. As a result of the analysis, the development teams are offered four solutions to support the requirements engineering process. The developement teams will evaluate and validate these solutions. Based on the feedback of the developement teams, the most suitable solution is picked and based on the solution, a full requirements engineering process is described. The developed process determines a specific set of rules for describing the requirements according to the type of requirement and the nature of the development project and satisfies the conditions initially set for the requirements process.

The thesis is in Estonian and contains 68 pages of text, 10 chapters, 7 figures, 1 table.

Lühendite ja mõistete sõnastik

SAFe	Scaled Agile Framework on suurtele ettevõtetele suunatud agiilset tarkvaraarendusprotsessi toetav raamistik.
Time-To-Market	Aeg, mis kulub tarkvara arendamisele kuni valmislahendus on kliendile üle antud.
DevOps	Arenduse ja halduse (*Development and Operations*) meeskonnad, kes tegelevad nii süsteemide arendamisega kui ka nende süsteemide hilisema haldamisega.
Status-check	Sprindi jooksul peetavad koosolekud arendustööde hetkeolukorrast ja võimalikest probleemidest ülevaate saamiseks.
Sprint	Kindla pikkusega arendustsükkel, millesse planeeritakse konkreetsed arendustööd.
Sprint-planning	Järgmisse sprinti võetavate tööde planeerimise koosolek.
COVID	COVID-19 on kergesti nakkuv ja ohtlik koronaviirus, mis põhjustas 2020 aasta algusest ülemaailmselt palju erinevaid piiranguid inimeste liikumistele ja kogunemistele.
Reliis	Tegevus, mille käigus üks või mitu arendatud funktsionaalsust paigaldatakse *live* keskkonda.
Reliisitsükkel	Kokkulepitud sagedus *reliisi* teostamiseks.
SKAIS2	Sotsiaalkindlustusameti infosüsteem, mille arendamise maksumus kujunes olulisel määral suuremaks esialgselt plaanitust.
Backlog	Arendustööde nimekiri, mida ei ole sprinti planeeritud.
Use-case	Kasutusjuht on nimekiri sündmustest või tegevustest kasutaja rolli ja süsteemi vahel, mis viivad oodatus tulemuseni.
User-story	Kasutuslugu on loomulikus keeles kirjutatud kirjeldus süsteemi omaduste kirjeldamiseks.
URL	*Uniform Resource Locator* Internetiaadress

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Iga infosüsteemi loomine algab ideest või vajadusest ja mingil kujul kirjelduse koostamisest, mida me soovime saavutada. Selliseks kirjelduseks on võimalik kasutada mistahes sobivat formuleeringut. Me võime koostada skeeme, teha jooniseid, visandada prototüüpe, kirjutada dokumente jne. Kõik need kirjeldused täidavad ühte eesmärki – nad esitavad nõudeid, millele loodav infosüsteem peab vastama. Seega saame öelda, et iga infosüsteemi üks peamine alustala on süsteemile seatavad nõuded.

Enamasti sõltub nõuete kirjeldamise detailsus süsteemi keerukusest ja ka kriitilisusest. Mida kriitilisema tähtsusega süsteemiga on tegemist, seda detailsemalt tuleb nõudeid kirjeldada.

Lisaks süsteemi omaduste kirjeldamisele on süsteemi nõuetel veel mitmeid erinevaid kasulikke kasutusvõimalusi. Süsteemi loomise algfaasis saab nõuete põhjal hinnata loodava infosüsteemi arendamiseks kuluvat nii ajalist- kui ka materiaalset ressursivajadust. See omakorda aitab otsustada, kas loodav lahendus on majanduslikult otstarbekas.

Infosüsteemi käitumist kirjeldavatest nõuetest ei saa me rääkida ainult uute loodavate infosüsteemide kontekstis. Sama olulist rolli mängivad nõuded ka juba olemasolevate infosüsteemide juures ja seda eriti olukorras, kus infosüsteemi või mõnda selle osa on vaja mingil põhjusel muuta. Siin aitavad kirjeldatud nõuded hinnata muudatuste võimalikkust ja keerukust. Muudatustega võivad kaasneda uued nõuded, mis peavad olema kooskõlas olemasolevate nõuetega ega tohi minna omavahel vastuollu.

Tänapäeval esitatakse nõuete haldamise protsessile veel üks oluline väljakutse. Nimelt liigub enamik tarkvaraarendusprotsesse agiilse mõtelaadi suunas, mis tähendab loodava süsteemi pidevat ja suhteliselt kiiret muutumist. See omakorda eeldab ka vajadusel nõuete kiiret muutmisvõimalust. Senised nõuete haldussüsteemid, mis toetavad traditsioonilisi arendusprotsesse, ei ole mõeldud agiilselt arendatavate süsteemide nõuete haldamiseks. Eriti keeruliseks muutub olukord agiilselt arendatavate suurte infosüsteemide kontekstis.

Nõuded ei ole mitte üksnes arendusfaasis süsteemi kirjeldamiseks vajalikud, vaid on olulised kogu arendusprotsessi jooksul. Tarkvaraarendusprotsessi üks oluline etapp on valminud süsteemide testimine. Nõuete põhjal saab koostada sobivad testid, et kontrollida süsteemi tööd. Vastavalt kirjeldatud nõuetele saab üleüldse hinnata, et kas arendatud on see funktsionaalsus, milles algselt kokku lepiti ning kui hästi on kokkulepetest kinni peetud.

Võib öelda, et nõuded on iga infosüsteemi lahutamatu osa. Selleks aga, et nõudeid oleks võimalik efektiivselt kirjeldada, on vaja nõuete haldamise protsessi ja seda protsessi toetavat infosüsteemi. Lisaks tuleb nõuete haldusprotsess üles ehitada selliselt, et see toetaks agiilset arendusprotsessi ja oleks kasutatav nii üksikute arendustiimide ning üksikute arendusprojektide vaates, kui ka erinevate süsteemide üleselt tervikpildi loomiseks.

1.1 Taust

Käesoleva magistritöö raames võtame vaatluse alla ühe konkreetse ettevõtte, milles puudub keskse nõuete haldamise protsess ja seda toetav keskne infosüsteem. Eeltoodust tulenevalt on ettevõttes rida kitsaskohti, millele tuleks keskenduda ja mida saaks parandada:

- Erinevates osakondades/meeskondades/projektides kirjeldatakse nõudeid erinevalt.
- Erinevates süsteemides kirjeldatakse nõudeid erinevalt.
- Kasutatakse erinevaid tarkvaralisi lahendusi nõuete kirjeldamiseks.
- Puudub ühtne protsess ja reeglistik nõuete kirjeldamiseks ja haldamiseks.
- Ebaühtlase ja erineva kvaliteediga nõuete kirjeldus ei võimalda samade nõuete ristkasutust üle terve ettevõtte.

Eespool mainitud süsteemide all peetakse siinkohal silmas ettevõttesiseseid infosüsteeme, mis on vajalikud ettevõtte majandustegevuse toetamiseks. Tegemist ei ole tarkvara-tootmisega tegeleva ettevõttega. Kogu loodav tarkvara on kasutuses ettevõtte-sisestes süsteemides ja klientidega suhtlemisel.

1.2 Probleem

Keskse nõuete haldusvahendi puudumine võib ettevõttele kaasa tuua reaalset majanduslikku kahju. Kui uute lahenduste ehitamisel ei ole majandusliku kahju tekkimise oht ehk väga suur, siis olemasolevate süsteemide ringi tegemisel või täiendavate funktsionaalsuste lisamisel on risk olemas. Puudulikult või halvasti kirjeldatud nõuete korral ei ole võimalik objektiivselt hinnata, et kui laiaulatuslik mõju võib olla süsteemi funktsionaalsuse muutmisel. Kui nõuete kirjeldamise protsessi ei ole kokku lepitud ja nõuded on erinevate süsteemide arendamisel erinevatesse keskkondadesse kirjeldatud, siis nõuab nõuete ülesleidmine täiendavat ajalist ressurssi. Halvemal juhul ei ole nõudeid üldse kirjeldatud või on seda tehtud puudulikult ning juba olemasolevatele süsteemidele tuleb nõuded tagantjärele tekitada. Kõik see võib põhjustada ettevõttele suuremat rahalise

ressursi vajadust süsteemide ehitamisel, mida oleks võimalik vähendada ühtse kokkulepitud nõuete haldusprotsessi olemasolul.

1.2.1 Ettevõtte taust

Konkreetse ettevõtte puhul on tegemist keskmisest suurema ettevõttega, milles on mitmeid osakondi ning suur hulk erinevaid infosüsteeme. Nagu tänapäevastes suurtes infosüsteemides tavaks, on väiksemad süsteemid üksteisega tihedalt integreeritud ning moodustavad kogu ettevõtte jaoks ühtse terviku.

Erinevad osakonnad ettevõttes vastutavad erinevate süsteemide arendamise ja haldamise eest. Kui rakenduste halduse eest vastutab enamasti ettevõttesisene ressurss, siis süsteemide arendamise osas kasutatakse sageli ka ettevõttevälist ressurssi. Väliste partnerite kaasamine erinevatesse arendusprojektidesse on erinev ja võib sõltuda mitmetest asjaoludest (sisemise ressursi hõivatus, projekti ajakriitilisus jne). Ettevõttevälist ressurssi kaasatakse vajadusel erinevatesse arendusprotsessi rollidesse. Põhiliselt on välise ressursi näol tegemist arendajatega, kuid vajadusel kaasatakse väliste partneritena ka projektijuhte, analüütikuid ja testijaid.

1.2.2 Nõuete kirjeldamise kvaliteedi mõju

Süsteemide arendusvajaduste põhjuseid olemasoleva funktsionaalsuse muutmiseks või uue funktsionaalsuse lisamiseks on mitmeid:

- Kliendi soovidest/vajadustest tulenevad.
- Äriprotsesside täiendamise/muutmise/automatiseerimise vajadus.
- Seadusandlusest tingitud.
- Jne.

Arendusvajaduse põhjus määrab üldjuhul ka arenduse kiiruse. Seadusandlusest tingitud arendused näiteks peavad olema valmis seaduses sätestatud ajaks ja see seab ka konkreetse ajalise piiri. Nii moodustuvad ka arendusmeeskonnad erinevalt vastavalt sellele, kui kiire on arendustega ehk millised on prioriteedid ja millist ressurssi saab hetkel kasutada (ettevõtte sisene *vs.* ettevõtte väline). Eelmainitud põhjustel võib tekkida

olukordi, kus analüüsi ja arendusega peab tegelema meeskond, kes varem ei ole selle süsteemiga kokku puutunud.

Kui süsteem on varem korrektselt ja järjepidevalt dokumenteeritud ning nõuded kirjeldatud, siis toimub uue meeskonna sisseelamisprotsess suuremate tõrgeteta. Kui aga nõuete kirjeldamine on puudulik, siis peab nõudeid hakkama uuesti defineerima ja valideerima. Kuna süsteemid on omavahel integreeritud, siis lisaks enda arendatavale süsteemile peavad teada olema ka seotud süsteemidele kehtestatud nõuded, mis puudutavad arendatavat süsteemi.

Samade nõuete uuesti kirjeldamine ja juba kehtivate nõuete otsimine erinevate süsteemide spetsifikatsioonidest toob endaga kaasa suure ajakulu. Lisanduvat aega ja sellega kaasnevat rahalist ressurssi on raske hinnata, sest ebaühtlase tasemega nõuete kirjeldamine lisab määramatust ning võib suurendada olulisel määral lahenduse keerukust. Iga algselt planeerimata lisanduv ajakulu suurendab projekti eelarvet ja võib mõjutada ka projekti valmimise tähtaega. Kui siia juurde lisada veel võimalus, et mõni oluline nõue on jäänud tuvastamata, siis võib selle nõude hilisem realiseerimine osutuda väga kulukaks.

Ettevõtte keskse nõuete halduprotsessi olemasolu aitaks kaasa kiiremale ja kvaliteetsemale mõjuanalüüsi teostamisele. Kui on teada, millistes süsteemides, millistes tarkvarakomponentides ja millistes rakendustes on nõue kasutusel, siis nõude muutumise või täienemise mõju ja eeldatavat töö mahtu on lihtsam hinnata.

Viimase eelisena võib välja tuua testimise parema kvaliteedi, mille tagavad korrektselt kirjeldatud nõuded. Testid koostatakse vastavalt süsteemile kehtestatud nõuetele. Testimise käigus kontrollitakse, kas arendatud on varem kokku lepitud funktsionaalsust ja kas süsteem vastab seatud nõuetele. Järelikult on testide läbiviimise edukus otseses sõltuvuses kvaliteetsetest ja adekvaatsetest nõuetest.

Kõike eelnevat kokku võttes ei ole hetkeolukord ettevõttes nõuete halduse osas kuluefektiivne ja vajab kindlasti parandamist. Ühtne nõuete halduse protsess parandab nõuete kirjeldamise kvaliteeti, toetab analüüsi teostamist ja arenduse läbiviimist ning lihtsustab testimist.

2 Töö eesmärk ja tulem

Kaasaegne tarkvaraarendus on tänapäeval enamasti mingil kujul agiilne protsess. Pikalt ja põhjalikult etteplaneeritud tarkvaraarendust, kus kogu lahendus on detailideni kokku lepitud ning kõrvalekalded planeeritust ei ole soovitatavad, tänapäevases kiirelt muutuvas maailmas enam üldiselt ei kohta. Nõuete haldusvahend ja nõuete kirjeldamise protsess peavad toetama agiilset lähenemist tarkvaraarendusele. Ettevõttes tuleb kokku leppida, millises mahus nõuete kirjeldamine on kohustuslik ja milliste nõuete kirjeldamine on kohustuslik. Erinevalt klassikalistest metoodikatest, kus planeerimise käigus koostatud dokumentatsioon oli kogu arendustegevuse aluseks, pannakse agiilses maailmas dokumentide koostamisele oluliselt vähem rõhku.

Käesoleva magistritöö eesmärk on välja töötada ja kirjeldada nõuete halduse protsess. See tähendab nõude teket, võimalikku muutumist ja arengut kogu süsteemi elutsükli jooksul:

- Kuidas nõue tekib?
- Kuidas nõue võib ajas muutuda/täieneda?
- Kuidas nõue muutub kehtetuks?

Ühtset nõuete haldusprotsessi toetab nõuete haldusvahendi olemasolu. Keskse haldusvahendi peamine eesmärk on see, et kogu ettevõttes kehtivad nõuded oleksid kirjeldatud ühes süsteemis.

- Nõude kirjeldamiseks vajalik andmestik:
    - Unikaalne ID
    - Nimetus
    - Kirjeldus
    - Seotud süsteemid

    - Seotud komponendid

- Nõude muutmine:
    - Nõude sisu täiendamine/muutmine
    - Nõuete kokku ühendamine – mitu sarnast nõuet soovitakse kokku tõsta
    - Nõuete lahti ühendamine – ühes nõudes on sisuliselt mitu nõuet, mida on mõistlik vaadelda eraldi nõuetena
- Nõuete sulgemine ehk tühistamine
    - Nõue ei ole enam vajalik ja otsustatakse sellest nõudest loobuda

Ettevõtte erinevates osakondades ja erinevates projektides kasutatakse varem välja kujunenud praktikaid ning ühtset reeglistikku nõuete kirjeldamiseks ei ole. Käesolevas töös tuuakse välja haldusvahendi nõutav funktsionaalsus, kuid haldusvahendi enda realiseerimine ei ole magistritöö skoobis.

Välja töötatud nõuete haldusprotsess on vajalik uute- ja olemasolevate süsteemide arendusprojektide toetamiseks:

- Aitab kaasa parema tervikpildi tekkimisele.
- Parandab arendustegevuse planeerimist ja hindamist.
- Tõstab testimise kvaliteeti.
- Hoiab kokku kulusid.

3 Metoodika

Käesolevas peatükis kirjeldatakse magistritöö eesmärkide saavutamise metoodikat ehk milliseid tegevusi tehes on plaan jõuda soovitud tulemuseni.

Peaasjalikult lähtutakse töö läbiviimisel kvalitatiivsest uurimismetoodikast, mille abil soovitakse uuritavatele küsimustele vastuseid leida. Kvalitatiivse uuringu teostamiseks kasutatakse informatsiooni kogumiseks süsteemi/objekti vaatlust ja viiakse läbi intervjuusid. Oluline roll on sõnalisel suhtlusel ja sõnalisel kirjeldusel. Erinevalt kvantitatiivsest uurimismetoodikast ei ole arvulised näitajad väga olulised [1].

Kvalitatiivne uurimismetoodika üritab leida vastused küsimustele miks ja kuidas. Keskendutakse rohkem kogutud informatsiooni tähendusele ja tõlgendamisele. Süvenetakse uurimistemaatika detailidesse ning ei vaadelda üldist trendi. Kvalitatiivse uurimismetoodika puhul on oluline informatsioonist arusaamine ja põhjus-tagajärg analüüsist järelduste tegemine [1].

3.1 Töö käik

Järgnevalt on kirjeldatud töö teostamiseks läbiviidud etapid, mis on valitud lähtuvalt metoodikast.

3.1.1 Andmete kogumine ehk hetkeolukorra kaardistamine

Kvalitatiivse uurimismetoodika korral kasutatakse informatsiooni kogumise esmase allikana süsteemide jälgimist. Sellise lähenemise eesmärgiks on mitte tekitada kunstlikke ehk laboratoorseid tingimusi. Nii saadakse võimalikult objektiivne hinnang süsteemi hetkeolukorrale.

Käesolevas töös teostas autor iseseisvalt erinevate projektide ja arendusmeeskondade tänase olukorra kaardistuse nõuete kirjeldamise vaates. Sellesse protsessi ei olnud kaasatud arendusmeeskondade liikmeid ja saadud informatsioon peegeldas ilmekalt iga meeskonna tööharjumusi ja nõuete kirjeldamise mustreid.

3.1.2 Intervjuude läbiviimine

Teise informatsiooni kogumise allikana kasutati töös intervjuude läbiviimist projektimeeskondade liikmetega. Selle etapi läbiviimise eesmärgiks oli eelnevas punktis kogutud informatsiooni täiendamine. Intervjuude läbiviimise käigus oli soov aru saada eelmises punktis kogutud informatsiooni sisust ning põhjendada seniste praktikate tagamaid.

Lisaks küsiti intervjuude kaudu arvamusi seni välja kujunenud nõuete kirjeldamise ja süsteemide dokumenteerimise heade ning halbade omaduste osas. Millised praktikad on tulnud süsteemide arendusprotsessis kasuks, millised praktikad võiks parandada ja milliseid praktikaid tulevikus kindlasti vältida? Kui toodi välja nõuete kirjeldamise puudujääke, siis püüti aru saada, kas on kavas tulevikus olukorda muuta ja miks seda seni tehtud ei ole.

Intervjuud puudutasid ka küsimust ettepanekute osas. Meeskondadel paluti võimalusel välja tuua ideid, mis aitaksid nõuete kirjeldamist muuta efektiivsemaks. Uuriti, kas on konkreetseid ettepanekuid ühtse nõuete halduse protsessi väljatöötamiseks. Hinnati üldist meelestatust muudatuste osas. Kas ollakse hetkeolukorraga pigem rahul või ollakse valmis muutusteks?

3.1.3 Kogutud andmete ja informatsiooni analüüs

Kahes eelmises etapis saadud info alusel viidi läbi sisuline analüüs. Uuriti meeskondade ühtseid käitumismustreid. Võrreldi erinevate meeskondade nõuete kirjeldamise aspekte, ja üritati leida ühisosad nii positiivsete kui ka negatiivsete kogemuste osas. Sarnaste mustrite korral üritati leida sarnasui ka meeskondade vahel, et leida aru saada sarnasuste põhjustest. Arvestati edasisi tegevusplaane ja arengusuundasid ning ettepanekute sobivust üle arendusmeeskondade.

3.1.4 Analüüsi tulemustest järelduste tegemine

Analüüsi tulemusena selgitati välja konkreetsed ideed nõuete kirjeldamise ja haldamise reeglistiku koostamiseks, mis oleksid kasutatavad üle kõikide arendusmeeskondade ja arendusprojektide. Saadud järeldused formuleeriti konkreetseteks ettepanekuteks, mille alusel küsiti meeskondadelt tagasisidet.

3.1.5 Tagasiside küsimine ja lahenduse valideerimine

Analüüsi tulemusena valminud ettepanekuid lasti hinnata nii neil meeskonnaliikmetel, kellega viidi läbi intervjuud, kuid kaasati ka teisi meeskonnaliikmeid, keda algselt ei intervjueeritud. See aitas saada tagasisidet muudatuste suuruse võimalikkuse kohta ehk kui palju oleksid meeskonnad valmis oma seniseid nõuete kirjeldamise tavasid vajadusel muutma.

3.1.6 Tulemi kirjeldamine

Magistritöö viimase etapina kontsentreeriti eelmistes sammudes kogutud informatsiooni ja selle analüüsimisel saadud järeldused formuleeriti konkreetseks tulemiks. Saadud tulemus annab maksimaalse võimaliku kasu, mis saadakse ühtsel nõuete kirjeldamisel üle ettevõtte. Teisalt arvestab saadud tulemus ka võimalikult minimaalsete või lihtsate muudatusvajadustega hetkeolukorraga võrreldes.

3.2 Analüüs

Käesoleva magistritöö raames töötas autor läbi rea kaasaegseid teadusartikleid, mis puudutavad nõuete haldust agiilses arendusprotsessis. Saadud teadmised kombineeriti arendusmeeskondadelt kogutud informatsiooniga ja selle informatsiooni analüüsi tulemustega. Kaasaegsete lahenduste väljatöötamine eeldab loomulikult tänapäevasete arengusuundade ja soovituste kasutamist. Samas tuleb arvestada ka vaatluse all oleva ettevõtte vajadustega.

4 Ettevõtte arendusprotsess ja kasutatav raamistik

Ettevõttes on juurutamisel uus arendusprotsessi raamistik, mida jooksvalt parendatakse ja täiustatakse vastavalt vajadustele. Arendusprotsessi toetamiseks on kasutusel erinevad tarkvaralised lahendused.

4.1 SAFe – Scaled Agile Framework

SAFe[1] raamistik on mõeldud agiilse arendusprotsessi arendamiseks ja agiilse lähenemise skaleerimiseks üle kogu ettevõtte (Joonis 1). SAFe on suunatud suurte tarkvaraprojektide arenduse agiilseks elluviimiseks. Erinevalt Scrum raamistikust, mis on välja töötatud peamiselt arendusmeeskonna põhise metoodikana, on SAFe puhul tegemist kogu ettevõtet hõlmavate agiilsete protsesside raamistikuga. SAFe annab juhiseid, kuidas agiilne lähenemine annab kasu kõigis protsessides, mitte ainult kitsalt tarkvaraarenduse protsessis.

SAFe on kasutusel paljudes suurtes ja globaalsetes ettevõtetes.

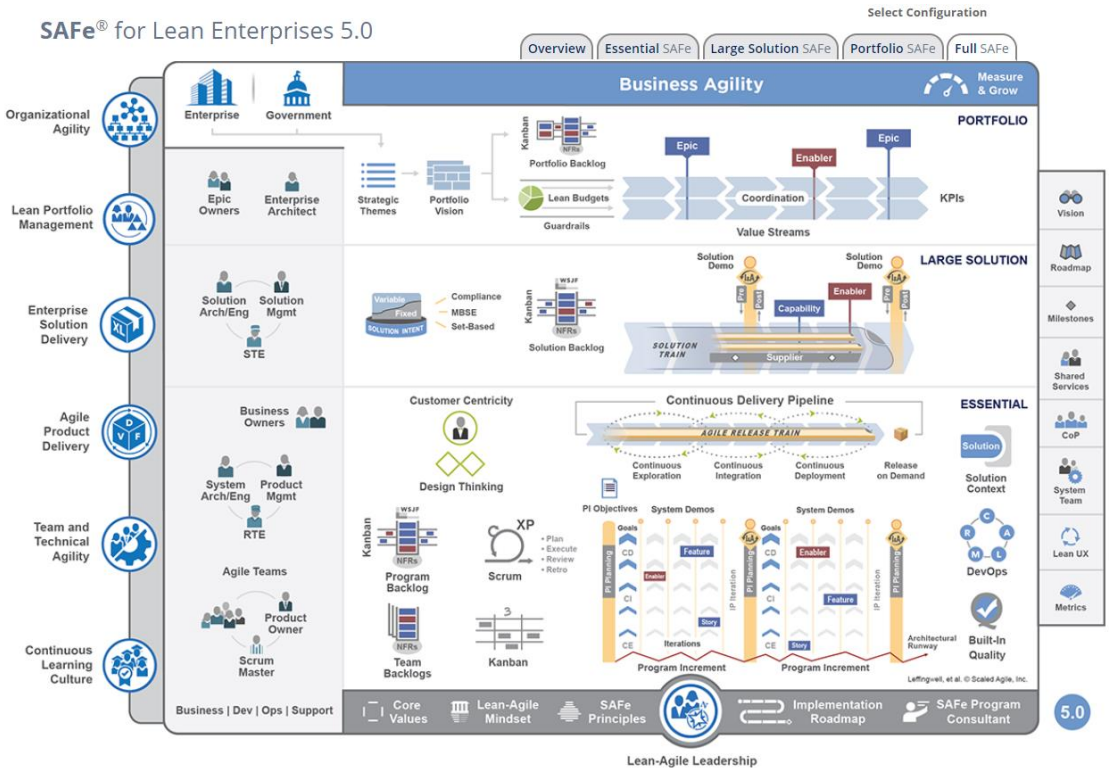
4.1.1 SAFe raamistiku kasutamine

Peamiste kasulike omadustena on SAFe raamistiku kirjelduses toodud välja järgmised punktid:

- Parandab arenduse valmimise kiirust (*Time-To-Market*).
- Suurendab arendustiimide produktiivsust.
- Parandab arenduskvaliteeti.
- Tõstab töötajate pühendumust.

[1] https://www.scaledagile.com/
https://www.scaledagileframework.com/

SAFe aitab kaasa loodava tarkvara pidevale tarnevoole (*Continous Delivery*) ning samal ajal tagab tellijale regulaarse ja ennustatava ülevaate valmivast funktsionaalsusest



Joonis 1. SAFe tarkvaraarendusprotsess.

Ettevõttes ei ole eesmärgiks seatud SAFe raamistiku täielik sajaprotsendiline kasutuselevõtmine. Praktikad on näidanud, et enamasti ei sobitu ükski raamistik täielikult ühtegi ettevõttesse, osakonda või projekti. Väga detailne ja põhjalik raamistiku järgimine võib hakata segama projekti elluviimist ning kasu asemel hoopis kahju tuua. Raamistiku juurutamise käigus lähtutakse SAFe agiilse arendusprotsessi põhivoost. Üksikute etappide läbiviimisel on jäetud tiimide enda otsustada, et kui täpselt nad SAFe reegleid järgivad. Tiimid on erinevad ja seetõttu kasutatakse ka erinevaid lähenemisi. Tiimide sees toimub pidev protsessi parendamine leidmaks just neid lahendusi, mis konkreetsele tiimile kõige paremini sobivad. Edu tagab see, kui kasutatavast raamistikust leitakse üles just need suunised ja käitumismallid, millest konkreetsel meeskonnal ka päriselt kasu on.

Tarkvaraarendusmeeskondi on nii ettevõtte siseseid, kui ka koostöös väliste tarkvaraarendusettevõtetega. Erinevatel meeskondadel on lubatud kasutada erinevaid agiilseid tarkvaraarendusprotsesse (Extreme Programming (XP), Scrum, the Dynamic Systems Development Method (DSDM), Agile Modeling (AM), Adaptive Software Development (ASD), Lean Development (LD), Crystal Methods and Features-Driven

Development (FDD)). Konkreetse arendusprotsessi valik on iga tiimi enda otsustada ja selles osas tiimideülest ühtset arendusprotsessi kasutamise kohustust ei ole.

4.1.2 Arendusmeeskondade koosseis ja SAFe rollid

Meeskondades on määratud kindlad rollid ja vastutusalad. Kogu protsessi vaates on SAFe raamistikus rolle loomulikult rohkem, kuid need ei oma käesoleva töö raames suuremat tähtsust. Alljärgnevalt on toodud nimekiri rollidest, kes otseselt osalevad tarkvaraarendusprotsessis:

- *Process Manager* (PM) – protsessijuht tagab äriprotsessi eesmärkide ja tulemuslikkuse täitmise.
- *Process Owner* (PO) – protsessi omanik vastutab äriprotsessile püstitatud nõuete täitmise eest, vastava äriprotsessi info omanik, juhib riske ja hoolitseb efektiivse töökorralduse eest.
- *IT Analyst* – analüütik teeb süsteemianalüüsi loodava tarkvaralise lahenduse kohta ja koostab kirjelduse arendajale ning testijale.
- *Developer* – arendaja kirjutab tegelikku tarkvarakoodi vastavalt analüütiku spetsifikatsioonile. Sõltuvalt meeskondadest võivad olla eraldi *Back-end Developer* ja *Front-end Developer*.
- *Solution Architect* – lahenduse arhitekt vastutab tehnilise terviklahenduse eest üle süsteemide ja lähtub protsessi/teenuse vaatest.
- *System Architect* – süsteemi arhitekt vastutab konkreetses süsteemi arhitektuuri ja toimimise eest, kehtestab ka arendusreeglid.
- *Tester* – testija koostab testid ja testib valminud arendused.

Lisaks on veel *DevOps* meeskonnad, kes enamasti toetavad samaaegselt mitut arendusmeeskonda ja paigaldavad valmis rakendused *live*-keskkondadesse. *DevOps* meeskondade põhitegevus on hoolitseda süsteemide igapäevase tõrgeteta toimimise eest.

4.1.3 Arendusprotsess

Ühtse arendusprotsessi toimimiseks on kokku lepitud kindlad etapid, millest arendusmeeskonnad peavad lähtuma. See tagab parema arendusprotsessi planeerimise meeskondade sees ja lihtsustab nii arendusmeeskondade omavaheliste kokkulepete kui ka äritellijate ja arendusmeeskondade vaheliste kokkulepete tegemist.

Tiimide üleselt on ühised arendusprotsessi osad järgmised:

- PI (*Program Increment*) *planning* – kõikide tiimidega samaaegselt toimuv tööde planeerimine järgmiseks arendustsükliks. Prioriteetide põhjal koostatud tööde nimekirja vaatab meeskond üle, hinnatakse umbkaudne maht ja jagatakse sprintidesse. Vajadusel lepitakse kokku tiimidevahelised sõltuvused, mis võivad mõjutada loodava funktsionaalsuse valmimist. Pannakse kirja ka võimalikud riskid, mis võivad takistada tööde valmimist või põhjustada valmimise edasilükkumist.

- *Sprint* – arendustsükkel, mille käigus toimub reaalne arendustöö.

- *Sprint demo* – tellijale esitatav ülevaate, kus kõik tiimid demonstreerivad ja kirjeldavad sprindi jooksul valminud töid.

Sprintide käigus sõltub infovahetuse sagedus vastavalt meeskonnas kokkulepitule. Kes soovib teha igapäevaseid kohtumisi ja kellele piisab näiteks paarist korrast nädalas tööde hetkeseisu ülevaatuseks. Lisaks on eraldi sprindi planeerimise ja retrospektiivi ehk sprindi kokkuvõtte tegemise koosolekud.

4.2 Kasutatavad töövahendid

Järgnevalt on välja toodud peamised tarkvaralised töövahendid, mis on kasutusel kõikides arendusmeeskondades. Kindlasti on töövahendeid kasutusel veel, kuid nende kõikide üleslugemine ei ole käesolevas töös oluline. Vaatleme vaid neid rakendusi mille kasutamine on kohustuslik ja kuhu tuleb kõik süsteemid ja arendustööd kirjeldada.

4.2.1 Enterprise Architect

Tarkvaratootja Sparxsystemi[1] poolt toodetud Enterprise Architect (EA) on väga võimekas modelleerimise, disainimise ja dokumenteerimise vahend erinevate protsesside, süsteemide ja nendevaheliste seoste kirjeldamiseks. Võimaldab jooniste koostamisel kasutada väga suurel hulgal erinevaid notatsioone. Jooniste elementidele saab lisada suurel hulgal erinevat metaandmestikku.

EA suur võimekus seab teisest küljest aga ka teatavad piirangud. Keerukamate jooniste koostamisel peavad arendusmeeskondade liikmed oskama EAd kasutada ja EAs koostatud joonistest aru saama. See hõlmab endas nii notatsiooni tundmist, kui ka metaandmetest aru saamist.

4.2.2 Atlassian tooteperekond

Atlassian[2] tootevalikusse kuulub palju erinevaid tarkvaraarendusprotsessi abistavaid tarkvaralisi pakette. Nimetame järgnevalt kahte olulisemat, mis on seotud nõuete kirjeldamisega ja süsteemide dokumenteerimisega.

- Jira – konkreetsete analüüsi- ja arendusülesannete haldusvahend, mis võimaldab jälgida ka suuremate arendusetappide valmimist. Lisaks kasutatakse ka tööajaarvestuseks.

- Wiki – dokumentatsiooni kirjeldamise ja hoidmise vahend.

Lisaks nendele kahele kasutatakse veel mõningaid tehnilisemaid Atlassiani tooteid, kuid need puudutavad koodihoidlaid, arenduste paigaldusvahendeid jms ning ei ole käesoleva magistritöö vaates olulised.

[1] https://www.sparxsystems.com/

[2] https://www.atlassian.com/software/

4.2.3 Slack

Operatiivseks infovahetuseks on igal meeskonnal kasutusel oma Slacki[1] kanal. Lisaks meeskonnapõhistele kanalitele luuakse vahel ka erinevate süsteemide üleseid kanaleid, kui tekib vajadus vastavaid süsteeme hõlmava arenduse teostamiseks.

Slacki kasutatakse jooksvalt tekkinud küsimustele kiirelt vastuste saamiseks ja operatiivseteks aruteludeks. Leitud lahendused dokumenteeritakse alati Jira, Wiki või EA keskkondadesse.

Olendevalt meeskondadest kasutatakse Slacki ka koosolekute läbiviimiseks (*status-check*, *sprint-planning*, *grooming*, jne), sest enamik meeskonnaliikmeid töötab COVID pandeemia tingimustes suuremal või vähemal määral kodukontorites. Kuna Slacki kanalid tekitatakse mingi konkreetse teema põhiselt (meeskond, projekt, süsteem), siis on ka koosolekud nendes kanalites vastava teema põhised.

4.2.4 Microsoft Teams

Microsoft Teams[2] on piisava turvalisusega, funktsionaalsusega ja jõudlusega rakendus koosolekute läbiviimiseks. Nimetatud kommunikatsioonirakendus on samuti väga aktiivses kasutuses, sest suure hulga inimeste füüsiline kokkukutsumine ühte koosolekuruumi ei ole hetkeolukorras soovitatav. Microsoft Teams kaudu peetakse sagedamini neid koosolekuid, mis ei ole ühe meeskonna põhised. Siia hulka kuuluvad nii osalise meeskonna osavõtul peetavad koosolekud (äritellijatega peetavad analüüsikoosolekud, arhitektuurikoosolekud, jms), kui ka üle meeskondade peetavad koosolekud (*PI planningu* koosolekud, *sprindi demo*, jne).

[1] https://slack.com/

[2] https://www.microsoft.com/en-us/microsoft-365/microsoft-teams/group-chat-software

5 Tarkvaraarendusmetoodikad ja nõuete haldus

Tarkvaraarendusprotsessid on viimaste aastakümnete jooksul läbi teinud pidevaid muutusi. Umbes 20 aastat tagasi põhines igasugune tootmisprotsess planeerimisel. Kirjeldati soovitud toode ehk dokumenteeriti üksikasjalik spetsifikatsioon, planeeriti ressurss, valmistati toode ja tarniti tellijale. Sarnaselt toimiti ka tarkvaraarendusprotsesside läbi viimisel.

Planeerimisel põhinevat tarkvaraarendusmetoodikat, mida kutsutakse ka traditsiooniliseks tarkvaraarendusprotsessiks, oli põhiline viis tarkvaraprojektide realiseerimiseks paarkümmend aastat tagasi. Plaanipõhine juhtimine pärines tööstusettevõtete juhtimise praktikate üle toomisest tarkvaraarendusse. Tarkvaraarendus kui uus tööstusharu oli oma erisuses veel avastamata ja muul moel ei osatud sellele läheneda. Planeeritud juhtimine tähendas suurel hulgal detailset ja põhjalikku dokumentatsiooni, mis oli aluseks plaani teostamisel.

Planeerimisel põhineval tarkvaraarendusprotsessi eeliseks peeti stabiilsust ja teatavat ennustatavust. Lõpptulemus oli ette kirjeldatud ja oodatav tulemus oli täpselt teada. Eeldati, et algselt kokku lepitud tingimused töö käigus ei muutu. [2].

Mida aeg edasi, seda selgemaks sai, et tarkvaraarendus ei ole võrreldav klassikalise tootmisega, kui soovitakse saada head lõpptulemust. Sageli ei olnud valminud lõpptulemus nende omadustega, mida sooviti. Halvemal juhul oli lõpptoode täiesti kasutuskõlbmatu ning kogu protsessi tuli otsast alata.

5.1 Koskmeetod

Klassikaline ehk koskmeetodil (*waterfall*) põhinev arendusprotsess algab nõuete kogumistest ja dokumenteerimisest (*Software Requirements Specification* – SRS). Loodud nõuete spetsifikatsiooni dokument on aluseks arendustööde planeerimisel ja teostamisel.

Koskmeetodil toimuva arendusprotsessi üks põhiline eeldus on nõuete staatilisus. Nõuded, mis on dokumenteeritud ja milles on kokku lepitud, hiljem ei muutu. Kui nõuded siiski muutuvad, siis väga vähesel määral. Nõuete muutmine, eriti arenduse hilisemas faasis, võib kaasa tuua oluliselt suurema ajalise ja/või rahalise ressursivajaduse.

5.2 Agiilne arendusmetoodika

Agiilne arendusprotsess vastupidiselt klassikalisele arendusprotsessile on just pidevas muutumises. Nõudeid ja vajadusi vaadatakse pidevalt üle ning staatilist dokumenti nõuete kirjeldamiseks ei koostata. Nõuete haldusprotsess peab toetama agiilset arendusprotsessi ja võimaldama nõuete pidevat muutumist ning täienemist.

Nõuete haldus on traditsiooniline meetod, mille eesmärk on tuvastada, analüüsida, dokumenteerida ja valideerida nõudeid. Selline lähenemine on omane traditsioonilisele arendusprotsessile. Võib tekkida küsimus, et kas nõuete haldus ja agiilne arendusmetoodika siis üldse kokku sobivad? Klassikaline arendusprotsess baseerub nõuete dokumendil ja tellija näeb lõpptulemust ehk valmis arendust. Agiilse metoodika fookus on vastupidiselt suunatud pidevale ja tihedale otsesuhtlusele tellija ja arendajate vahel [3].

Agiilse arendusprotsessi edukaks toimimiseks peavad olema täidetud mitmed tingimused. Agiilses maailmas on kõik pidevas muutumises ja muutuste dünaamika on pigem kiire. Arendustiimid on fokusseeritud konkreetsete ülesannete täitmisele. Üksikud arendusülesanded peavad olema piisavalt terviklikud ja lisama mõne uue konkreetse funktsionaalsuse valmivasse lahendusse. Pikka reliisitsüklit ei ole ja toimub pidev tarkvara tarne (*Continous Delivery*).

Kiire ja pidev muutumine tähendab seda, et samamoodi muutuvad pidevalt ka tarkvarale seatavad nõuded. Täiendavate nõuete lisandumine ja olemasolevate nõuete muutumine või ära langemine paneb olulise vastutuse nõuete halduse süsteemile.

Ühes Ericssoni tootearenduse osakonnas alustati 2012. aastal transformatsiooni agiilsele arendusprotsessile. Kui enne transformatsiooni algust olid nõuded ette koostatud ja reliisitsükli tihedus oli 2 reliisi aastas, siis aastaks 2017 jõuti tiheduseni üks reliis kuus.

Arendusprotsessis osales 30 väikest meeskonda, kes pidid hakkama kasutama Scrum metoodikat, mis tõi endaga kaasa väga suur muutuse ka nõuete halduse vaates.

Peamised väljakutsed olid järgmised.

- Nõuded muutuvad sagedasti, mis põhjustab meeskondade vahelisi üksteise tööde blokeerimisi.

- Nõuete kirjeldamise andmebaasist tehti palju erinevaid kloone. Nende erinevate andmebaaside pidev korrastamine, millised nõuded on realiseeritud ja millised nõuded on veel lõpetamata ning korrastatud andmebaasi klooni sünkroniseerimine põhiandmebaasiga, nõudis arvestatavaid jõupingutusi.

- Puudus otsene seos nõude ja *live*-keskkonda paigaldatud versiooni vahel. Ei olnud võimalik aru saada, et millal konkreetne featuur on lõppkasutajale tarnitud.

Kirjeldatud probleemide lahendamiseks võeti kasutusele nõuete haldusvahend T-Reqs. [4].

5.3 Agiilne arendusprotsess

Kaasaegsed tarkvaraarendusmetoodikad on enamasti suuremal või vähemal määral agiilsed. Klientide vajadused muutuvad kiiresti ja seda peab toetama ka tarkvaraarendus. Agiilsed arendusprotsessid on praktikas näidatud, et töötavad hästi väiksemates ja lihtsamates projektides. Väiksemates projektides on vähem dokumentatsiooni ja puuduvad pikaajalised ning enamasti jäigad kokkulepped. Soovid ja vajadused on agiilsetes protsessides pidevas muutumises. Arendaja ja tellija vahel on kogu arendusprotsessi vältel tihe omavaheline koostöö. Palju on näost-näkku kohtumisi. Iteratsioonid on lühikesed ja loodav funktsionaalsus areneb ning täieneb pidevalt. Vajalikud muudatused tehakse kohe. Selliselt välditakse olukordi, kus alles projekti hilisemates faasides ilmneb mõni oluline muudatusvajadus. Mida hiljem avastatakse oluline muudatusvajadus, seda aega nõudvamaks ning seetõttu ka kulukamaks kujuneb selle muudatuse realiseerimine.

Kuidas aga juurutada agiilset lähenemist suurtesse, keerukatesse ja missioonikriitilistesse tarkvaraprojektidesse? Kas seda saab teha sama lihtsalt kui väikestes projektides? Kas suurtes projektides on agiilsed meetodid sama edukad kui väikestes projektides?

Agiilse meetodi kohaselt toimub ärinõuete pidev kogumine, täienemine ja muutumine. Klientidele tarnitakse pidevalt uut või täiustatud/muudetud funktsionaalsust. Klientide kasutusse antava tarkvara granulaarsus peab olema valitud selliselt, et klient saaks seda kasutada ja selle põhjal anda kiirelt tagasisidet. Dokumentatsiooni koostatakse minimaalselt.

Agiilsed meetodid on tõestanud ennast edukalt väikestes projektides, kuid üha enam leiab agiilne mõtteviis kasutamist ka tõeliselt suurtes tarkvaraarendusprojektides, kus meeskonnad on globaalsed.

Viimase 8-9 aasta jooksul on tutvustatud mitmeid Agiilse lähenemise raamistikke, mis on mõeldud kasutamiseks suurte ja keerukate tarkvarasüsteemide arenduste juhtimiseks. Scaled Agile Framework (SAFe), Large Scale Scrum (LeSS), Disciplined Agile Delivery (DAD), Scrum of Scrums (SoS), Nexus, and Recipes for Agile Governance in the Enterprise (RAGE).

Ükski metoodika iseenesest ei ole „hõbekuul“, mis lahendaks kõik tarkvaraarendusega seoses tekkivad probleemid. Lisaks agiilsele metoodikale peab kogu organisatsioon olema agiilse mõtteviisiga. Ainult selliselt võib loota edu saavutamisele [5].

Süsteemi suurust on sageli raske objektiivselt hinnata. Milliste parameetrite järgi võime otsustada, et tegemist on suure süsteemiga ja saame öelda, et agiilset arendust on keeruline teostada? Milliste tingimuste olemasolul võib kindlalt väita, et süsteem ei ole liiga keerukas ja agiilne arendus on ainuõige valik? Sellistele küsimustele on praktiliselt võimatu üheselt vastata.

Suurte projektide korral, mille pikkuseks võib olla planeeritud mitmeid aastaid, on klassikalise arendusprotsessi valikul peamiseks ohuks see, et lõpptulemus ei vasta algsele ootusele. Kõige halvemal juhul osutub loodud süsteem täiesti kasutuskõlbmatuks (SKAIS2). Projekti lõpuks oleme kulutanud väga suure hulga ressurssi, kuid vastu ei ole saanud midagi, mida oleks võimalik kasutada.

Kui sama projekti juhtida agiilselt, siis on tellija pidevalt kaasatud ja saab üsna varakult aimu, et kas liigutakse õiges suunas ehk soovitud tulemuse poole. Kui selgub, et vajadused või soovid muutuvad, siis saab need muudatused lihtsamini ellu viia.

Siiski on ka agiilsel lähenemisel omad ohud. Kui näiteks juba arendatud funktsionaalsust pidevalt parendatakse ja/või ringi tehakse, siis see kulutab samuti ressurssi ja tervikliku lõpptulemuse saavutamine lükkub kaugemasse tulevikku.

5.4 Komponendi/süsteemi- *vs*. featuuripõhine lähenemine

Võime tarkvaraarendusmeeskonnad laias vaates jagada kaheks. Komponendi- või süsteemipõhised meeskonnad ja featuuripõhised meeskonnad. Nendel meeskondadel on põhimõtteliselt erinevad huvid tarkvarapakettide arendamisel. Komponendipõhised ja ka süsteemipõhised arendusmeeskonnad on huvitatud eelkõige sellest, et nende süsteemid töötaksid laitmatult. Featuuripõhistes meeskondades on fookuses kliendile pakutava väärtuse kasvatamine. Edukaks tulemuseks on vaja, et mõlemad meeskonnad töötaksid koos hästi. Ühelt poolt peame pakkuma klientidele vajalikku funktsionaalsust ja teiselt poolt peavad süsteemi erinevad osad vastama etteantud nõuetele.

Kiiresti arenevad komponendid ja süsteemid on pidevas muutumises. Põhjalikku ja detailset dokumentatsiooni nõuete kohta ei koostata ja kui ka koostatakse, siis seda ei suudeta hoida pidevalt piisavalt värskena. Sageli ei ole võimalik aru saada, et millistele nõuetele täpselt süsteem vaadeldaval ajahetkel vastab. Täpse ja konkreetse ülevaate puudumine segab ka testimist.

Agiilses maailmas räägitakse enamasti tarkvara omadustest, mis aitavad kliendi probleeme lahendada. Vähem rõhku pannakse konkreetsetele süsteemi nõuetele. Siin tekib paratamatult väljakutse, et kuidas neid kahte maailma ühendada. Süsteem peab pakkuma kliendile mingit kasu, kuid detailsed süsteeminõuded on selleks liiga spetsiifilised.

Raamistikud nagu Scaled Agile Framework ja Large-Scale Scrum[1] tegelevad kirjeldatud probleemi lahendamisega, kuid mitte alati piisavalt. [5]

[1] https://less.works/

5.5 Nõuetega seotud teadmise haldus

Featuuripõhised meeskonnad omavad teadmist selle kohta, mida klient tahab loodava tarkvaraga saavutada ehk milleks klient seda täpselt kasutab. Süsteemipõhised meeskonnad seevastu omavad spetsiifilist teadmist süsteemi nõuete kohta, millele nende süsteem täpselt vastab.

Featuuripõhist teadmist nimetatakse lühiajaliseks teadmiseks. Kliendi vajadused ja soovid võivad kiirelt muutuda ja nii ka loodud tarkvara funktsionaalsus.

Süsteemipõhist teadmist nimetatakse pikaajaliseks teadmiseks. Nõuded, millele konkreetne süsteem vastama peab, on stabiilsemad ja ei ole nii kiires muutumises kui kliendile pakutavad featuurid.

Arendusprotsessi jooksul erinevate meeskondade loodav dokumentatsioon on eelevat silmas pidades samuti erinev. Featuuripõhine meeskond kirjeldab nn kasutajapõhiseid nõudeid ja süsteemipõhine meeskond süsteemispetsiifilisi funktsionaalseid nõudeid.

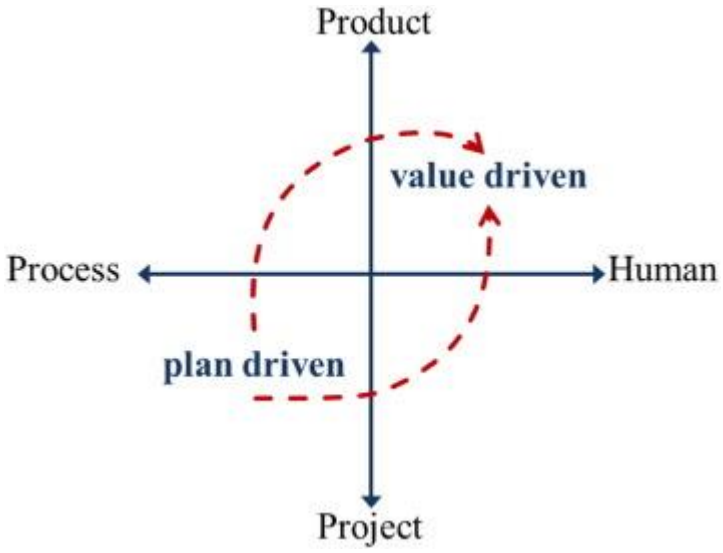
Detailsusaste, millele kasutajapõhised ja süsteemipõhised nõuete dokumendid peavad vastama, on erinev. Tuleks lähtuda arusaamast, et kirjeldatud nõuded baseeruvad autonoomsete agiilsete meeskondade langetatud otsustele. Kasutajapõhised nõuded on üldisemad ja süsteemipõhised nõuded on spetsiifilisemad. Autonoomsetel meeskondadel peab olema piisav kompetents, et nad oskaksid hinnata langetatud otsuste ja nende otsuste realisatsiooni mõju kogu süsteemile. [5].

Tiimisiseste, nn lokaalsete lahenduste osas on lihtne kokkuleppeid saavutada, nõudeid kirjeldada ja hiljem seda ka dokumenteerida. Keerukam on olukordades, kus kokkulepped tuleb saavutada üle mitme süsteemi või komponendi ja kus kokkulepe tuleb saavutada mitme meeskonna vahel. See esitab meeskondadele keeruka väljakutse nõuetega seotud teadmise haldamiseks. Samas on tegemist väga kriitilise teemaga kogu süsteemi vaates. Eriti keerukas on selle küsimuse lahendamine suurte agiilsete süsteemide arenduse juures. [5].

Kes vastutab nõuete halduse eest, kui need puudutavad mitmeid komponente või süsteeme? Kuidas sünkroniseerida teadmist erinevate tiimide vahel ja erinevate süsteemide vahel? Kui igal süsteemil on näiteks vastutavaks üks inimene, siis kas see ei

või kujuneda pudelikaelaks arendusprotsessis, kui süsteemil on palju tarbijaid ja tuleb kõigiga sünkroniseerida kehtestatavaid nõudeid?

Need on küsimused, millega tegelevad suurte ja keerukate süsteemide agiilsed arendusraamistikud.



Joonis 2. Agiilses arendusprotsessis seatakse fookus tulemusele.

Agiilses arendusprotsessis on osalejate fookus inimestele kasu andmises läbi loodava väljundi (Joonis 2) [6].

Huvipooled on pidevalt seotud tarkvara valmimisega. Seda parem saab lõpptulemus, mida tihedam on koostöö. Pole mingit kindlat perioodi arendustsüklis, et millal lastakse tellija uut süsteemi kasutama. Mida varem tekib tarkvara lõppkasutajal aimdust loodavast lahendusest, seda varem saab ta kaasa rääkida probleemidest ja kitsaskohtadest süsteemi kasutamisel.

5.6 Nõuete haldus agiilses arendusprotsessis

Nõuded mängivad tarkvaraarenduses keskset rolli. Agiilses maailmas ei ole traditsioonilist nõuete dokumenti. Kasutatakse prioritiseeritud nõuete listi ehk toote *backlogi*. Sellest listist võetakse tööd arendusse vastavalt seatud prioriteetidele.

Nõuded muutuvad pidevalt kogu arendusprotsessi vältel. Ei ole kindlat kokku lepitud nõuete dokumenti projekti alguses, millele lõpptulemus vastama peaks, mille alusel süsteemi testitakse ja mille põhjal otsustatakse kas projekti võib lugeda lõpetatuks.

Tänapäevasest teaduskirjandusest võib palju leida viiteid nõuete ja nende haldamise kohta. Infosüsteemidele seatavad nõuded on üks fundamentaalsetest alustaladest, ilma milleta ei ole võimalik infosüsteeme arendada. Nõuetega seotud temaatikaga on tegeletud juba esimestest infosüsteemide loomisest alates. Samas on see valdkond aktiivses muutumises käsikäes arendusmetoodikate muutumisega ning aktuaalne ka tänapäeval.

Alates 1993. aastast korraldatakse igal aastal rahvusvaheline konverents, mis on pühendatud ainult nõuete haldusele: „International Requirements Engineering Conference“ . Alates 2003. aastast on konverentsi sponsoreerinud organisatsioon „Institute of Electrical and Electronics Engineers“ (IEEE). Konverentsi materjalides on palju viiteid viimastele trendidele ja arengusuundadele nõuete halduse teemadel.

Kogu maailm meie ümber on pidevas muutumises. Muutused toovad aga endaga kaasa ka nõuete muutumise ja muutmise vajaduse. Kui varem olid muutused aeglased ja jõuti nendega harjuda, siis tänapäeval üha agiilsemaks muutuvad arendusmetoodikad nõuavad kiiret reageerimist. Ühest küljest on sellised muutused häirivad. Teisalt aga on tegu muutustega, mis võimaldavad kiiret arengut.

Uued agiilsed arendused ja erinevate tehnoloogiate laiaulatuslik kasv (avatud lähtekoodiga süsteemid, mobiilirakendused, asjade Internet jne) nõuavad teistsugust lähenemist ka nõuete halduses. Üheks põhiliseks omaduseks peab saama nõuete jälgitavus, et relevantseid nõudeid saaks kiiresti leida [7].

Nagu mainitud, siis agiilset arendusprotsessi toetava nõuete haldusprotsessi puhul on oluline, et nõuded oleksid jälgitavad ja lihtsalt üles leitavad. Hästi jälgitavad nõuded on seotud süsteemide või rakenduste või tarkvarakomponentidega piisava detailsusastmeni (sõltub nõudest, kas on üldisem või spetsiifilisem nõue).

Varasemad praktikad on näidanud, et inimene ei ole väga järjepidev nõuetele seosete kirjeldamisel. Selle asemel on erinevaid tehnoloogiaid, mis genereerivad seoseid automaatselt. Isegi siis, kui seosed on piisavalt heal tasemel kirjeldatud, ei ole soovitud nõuete leidmine lihtne. Sama fraas võib süsteemis olla kasutusel mitmes tähenduses ning otsingutulemused ei anna oodatud vastuseid. Just mitmetähenduslike terminite otsing on käesolevas kontekstis keeruline ülesanne. Kasutada saab erinevaid tehnoloogiaid ja otsingualgoritme, et mitmetähenduslike terminite otsing annaks paremaid tulemusi [8].

Üha enam kasutatakse infosüsteemide arendamiseks rahvusvahelisi meeskondi, kes töötavad sama projektiga samaaegselt erinevateest riikidest. Piiriülene arendustegevus toob endaga kaasa mitmeid aspekte, millega tuleb arvestada ja mida tuleb teadvustada. Erinevates ajavööndites töötamisel ei saa korraldada koosolekuid ja infovahetus on aeglane, riigiti on erinev töökultuur jne.

Globaalsete meeskondadega tehtavas tarkvaraarenduses on nõuete haldusel väga oluline roll. Eriti suur väljakutse on nõuete muudatuste haldus ja sellega kaasnevad probleemid. Eduka tulemuse tagab nõuete halduse, nõuete muudatuste halduse, projektijuhtimise ning nende erinevate protsesside omavaheline tihe koostöö [9].

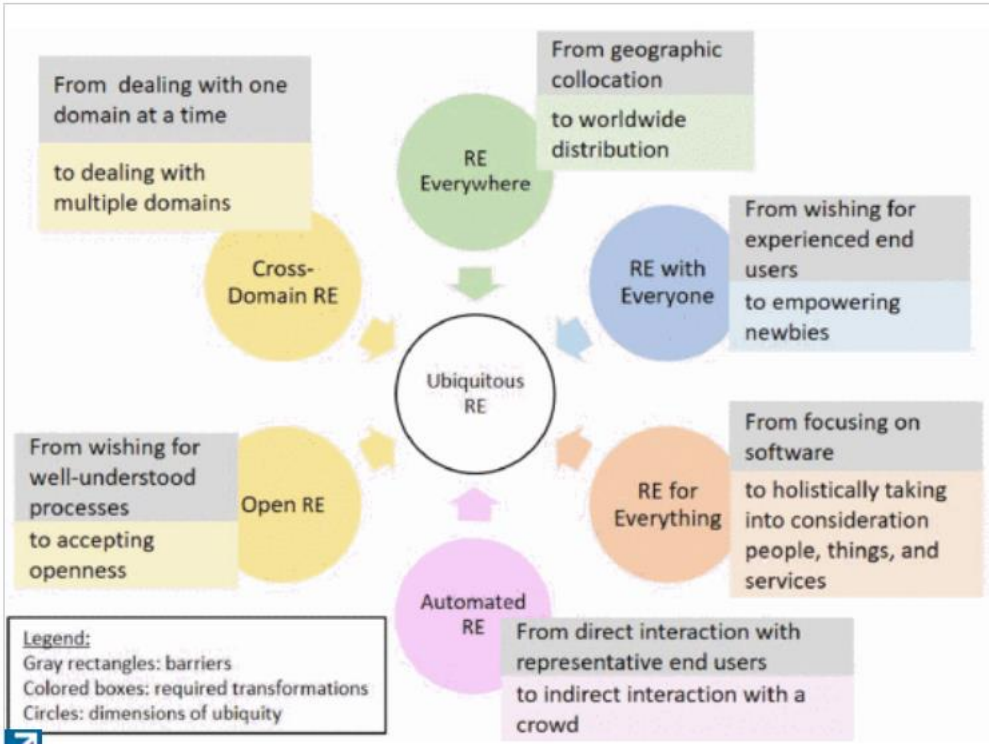
Kui nõuete muudatuste haldus on keerukas agiilsetes arendusprojektides ja kohalike arendusmeeskondade korral, siis seda keerukam on see globaalsetes arendusprojektides. Erinevate probleemidega, nagu näiteks keeleline barjäär, suurtest vahemaadest tingitud kommunikatsiooni keerukus jms, peab arvestama ka nõuete muudatuste halduse protsessi väljatöötamisel.

Üheks võimaluseks on kasutada raamistikku, mida teatakse nime all „AZ-mudel“. Mudel koosneb kolmest suuremast etapist:

- koordineerimise või kooskõlastuse faas – pannakse kirja muudatus;
- analüüsi faas – tehakse otsus, et kas muudatus implementeeritakse;
- arenduse ja implementeerimise faas – disainitakse, arendatakse ja implementeeritakse muudatus.

„AZ-mudel“ ja projektijuhtimise erinevad faasid on üksteist toetavad [10].

Me elame digitaalse transformatsiooni ajastul. Nõuete haldus jaotub erinevatesse dimensioonidesse tarkvaraarenduses, mis on põhjustatud digitaalsest transformatsioonist (Joonis 3) [11].



Joonis 3. Nõuete halduse kuus dimensiooni [11].

Sageli on kirja pandud nõuded madala kvaliteediga, mis omakorda põhjustab halva kvaliteediga lõpptulemuse. Üks sagedane põhjus ebakvaliteetsete nõuete tekkimisel on huvirühmade leige suhtumine nõuete kirjeldamisel.

Huvigruppide initsiatiivi elavdamiseks võib kasutada erinevaid tehnikaid. Üks võimalus on nõuete kirjeldamisse sisse tuua klassikalise mänguteooria elemente [12].

Nõuetele prioriteetide määramine aitab planeerida arendusprotsessi. Prioriteetsemad nõuded on ettevõttele olulisemad ja toovad rohkem kasu kui väiksema prioriteediga nõuded. Agiilses arendusprotsessis saab tellija määrata nõudeid koostades, milline on süsteemi baasfunktsionaalsus ning mis peab süsteemis kindlasti arendatud olema. Madalama prioriteediga nõuded võib arendada hiljem.

Parima tulemuse saavutame siis, kui saame määrata nõuetele prioriteedid automaatselt mingi süsteemi poolt. Kui nõudele määrab prioriteedi inimene, siis ei pruugi see alati olla objektiivne valik. Selliseid automaatseid prioriteedi määramise algoritme on koostatud [13].

5.7 Testimine ja vigade tuvastamine

Süsteemi mittevastavus nõuetele põhjustab vigu. Samuti põhjustab vigu süsteemis mõne nõude puudumine. Mida hilisemas projekti faasis viga avastatakse, seda kulukam on selle likvideerimine.

Mudelipõhine valideerimine läbi kasutuslugude vaadete aitab vigade tuvastamisel [14].

Testimine aitab hinnata, kas loodav süsteem vastab ootustele ja kas süsteemi töös ei ilmne vigasid. Testimiseks peab teadma, millised on süsteemile seatud nõuded. Vastasel korral ei ole võimalik tuvastada, kas loodud süsteem vastab sellele, milles kokku on lepitud.

Kirjeldatud nõuded ja testimine peavad töötama sünkroonis, et saavutada parim tulemus [15].

Suurte süsteemide testimine on ajamahukas ja nõuab palju ressurssi. Olulisemat funktsionaalsust, mille puudumine või vigaselt töötamine põhjustab süsteemis suuri probleeme, tuleb testida põhjalikumalt, kui vähemolulisi aspekte.

Nõuetele seatavad prioriteedid ning testide koostamine vastavalt nõuete prioriteetidele aitab määrata testimise põhjalikkust [16].

6 Nõuete kirjeldamise hetkeolukord vaadeldavas ettevõttes

Magistritöö järgmises punktis kirjeldame vaadeldava ettevõtte tarkvaraarendusprotsessi. Vaatleme millised on kehtivad kokkulepped arendustööde läbiviimisel ja kuidas neid kokkuleppeid rakendatakse. Sealhulgas kirjeldame ka nõuete halduse ja nõuete kirjeldamise mustreid erinevates tarkvaraarendusmeeskondades.

6.1 Arendustööde ja arendusülesannete kirjeldamine

Konkreetsete tööülesannete kirjeldamiseks arendusmeeskonnale või konkreetsele arendajale kasutatakse Jira pileteid. Piletitele määratakse tüüp. Töökorralduslikult on kokku lepitud, millist tüüpi tegevusele konkreetset tüüpi pilet vastab.

6.1.1 EPIC

Sellist tüüpi Jira pilet kirjeldab mingit suuremat ja terviklikku funktsionaalsust, mis lõpptarbijale omab konkreetset väärtust. EPIC luuakse alati ühte kindlasse projektiruumi ja määratakse vastutaja ning arendusmeeskond. Tegemist on teatud mõttes nn „katusega“ üksikutele töödele, mida on vaja teha tervikliku funktsionaalsuse valmimiseks.

6.1.2 STORY

Enamik arendusülesandeid, mis antakse arendajatele töösse, on STORY tüüpi piletid. Üldjuhul kuulub STORY mingi EPICu alla ja on üks osa terviklikust funktsionaalsusest. Kui on tegemist mingi väikese eraldiseisva arendustööga, siis võib STORY olla ka „ilma katuseta“ ja mitte kuuluda EPICu alla.

6.1.3 SUB-TASK

Osad meeskonnad kasutavad suuremate STORYde tükeldamiseks SUB-TASKe. See lihtsustab tööde staatuse jälgimist ja aitab saada ülevaadet, et milline osa STORYst on tehtud ja mis on veel tegemata. SUB-TASK kuulub alati konkreetse STORY alla ja ei saa eksisteerida eraldiseisvana.

6.1.4 TASK

Kui tegemist on konkreetse ja suhteliselt lihtsa ülesandega, mis ei eelda väga palju analüüsi ning kirjeldust, siis sellised tööd kirjeldatakse TASK tüüpi piletitena. Enamasti kuuluvad siia alla näiteks andmete migreerimised ühest süsteemist teise, seadistuste tegemine süsteemis vms. Osad meeskonnad kasutavad TASK tüüpi pileteid ka analüüsitööde puhul.

6.1.5 BUG

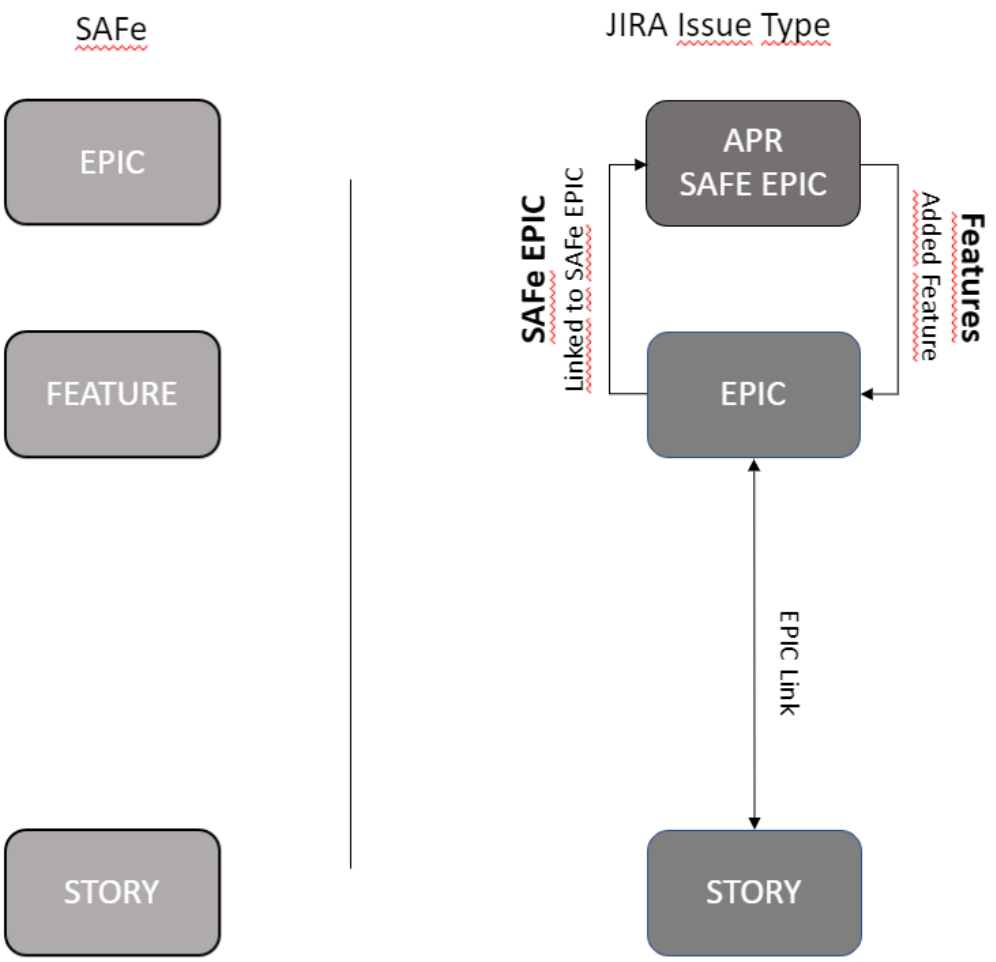
Testimise käigus või ka *live*-keskkonnas tekkivad vead kirjeldatakse BUG tüüpi piletitena. Sõltuvalt vea kriitilisusest saab määrata piletile prioriteedi. Vastavalt määratud prioriteedile saab meeskond otsustada, kas viga tuleb parandada kohe või saab sellega hiljem tegeleda.

6.2 SAFe metoodika ja Jira töödehaldusvahend

SAFe raamistikus kasutatakse tööde kirjeldamiseks sarnast notatsiooni nagu seda on Jira piletite tüübid, kuid sarnaste väljendite taga mõistetakse pisut erinevaid kirjeldusi. SAFe ja Jira notatsioonide üldine vastavus on toodud joonisel 1 "SAFe raamistiku ja JIRA vastavus“ (Joonis 4).

Joonisel kasutatav mõiste SAFE EPIC on kasutusel ka Jira pileti tüübina. Eraldi seda kirjelduses välja tuua ei tundu otstarbekas, sest kasutatakse äritellija poolt suuremate arendussoovide kirjeldamiseks. Arendusprotsessis sellise tüübiga Jira pileteid kasutusel ei ole.

Mõiste APR on üldine nimetaja kogu ettevõtte arendusportfellile.



Joonis 4. SAFe ja Jira notatsiooni vastavus.

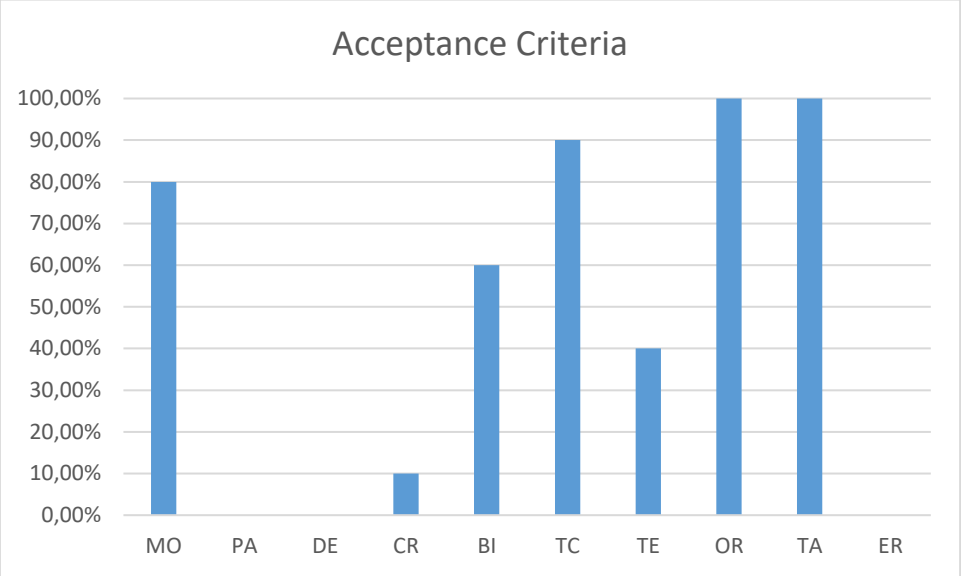
6.3 Dokumenteerimise ülevaade

Jira pileteid koostavad ja kirjeldavad meeskonnad erinevalt. Hetkeseisust ülevaate saamiseks võttis töö autor aluseks 10 suvalist projekti ja vaatles neis projektides kirjeldatud STORY tüüpi Jira pileteid. EPIC tüüpi piletite võrdlust eraldi ei teinud, sest konkreetsed arendusülesanded on alati STORY tüüpi piletites. Samuti jäeti välja TASK ja BUG tüüpi piletid. Nende mõlema tüübi puhul on üldjuhul koostatud konkreetne kirjeldus, et mida selle pileti raames tuleb teha.

6.3.1 Acceptance Criteria

Konkreetse võrdluse saamiseks tegin kokkuvõtte *Acceptance Criteria* (AC) andmevälja kasutuse kohta Jira piletites (Joonis 5). AC on Jira pileti üks paljudest andmeväljadest ja on mõeldud kirjeldamaks tingimusi, millele vastava Jira raames tehtud arendused

vastama peavad. Graafik näitab selgelt, et ühtlane lähenemine tööde kirjeldamisele puudub. Nagu näha, siis osades projektides kasutatakse AC välja täitmist alati või peaaegu alati ja teistes projektides kasutatakse välja täitmist vähem või ei kasutata üldse. Siit saab järeldada, et kasutamine *vs.* mittekasutamine on projekti meeskonna kokkuleppeline otsus. Kui vaadata Jira piletite kirjelduse (*Description*) andmevälja nendes projektides, kus AC välja ei kasutata, siis AC tingimused on tegelikult enamikes kirjeldustes siiski olemas.



Joonis 5. Acceptance Criteria kasutusstatistika STORY'des.

6.3.2 Wiki ja Enterprise Architect

Järgmise kahe parameetrina võrdlesin STORY'de juures viitamist Wiki dokumentatsioonile ja viiteid EA joonistele (Joonis 6). STORY lugesin dokumenteerituks Wiki keskkonnas juhul, kui Jira pileti mõnes kirjelduses (*Description* või *Link To Documentation* andmeväli) oli viide Wiki lehele. EA kasutamiseks lugesin kahte võimalust. Esiteks STORYs olevat otselinki EA joonisele. Teiseks STORYs olevat viidet Wiki dokumentatsioonile ja Wikis olevat linki EA joonisele.

Wikis tööde dokumenteerimine on samuti erinev ja sõltub konkreetsest projektist/meeskonnast. Võib järeldada, et kui meeskond tegeleb mingi ühe konkreetse funktsionaalsuse arendamisega ja kasutatakse pidevalt samu Wiki lehti, siis ei vaevuta iga Jira pileti juurde Wiki viidet lisama. Samas aga leidus ka pileteid, mille tööde kirjeldus oleks olnud mõistlik ka Wikis kirjeldada. Üldjuhul võib siiski öelda, et süsteemide kirjeldused on Wikis dokumenteeritud. Kui jätta kõrvale erandina üks projekt, mille kohta

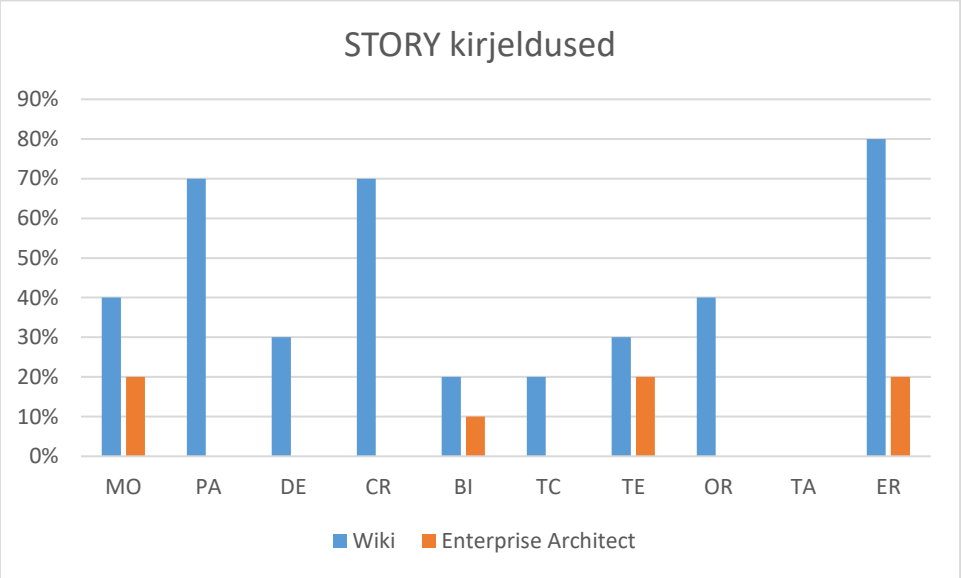
Wiki dokumentatsioon puudus, siis keskmiselt on Wiki dokumentatsiooni kattuvus 44% Jira piletitest üle vaadeldavate projektide.

EA kasutuse vaates on Jira piletites EA viiteid vähem kui Wiki viiteid. Seda võib seletada mitme erineva aspektiga.

- Wiki keskkond on aktiivses kasutuses olnud pikema perioodi vältel kui EA.
- Wiki dokumentatsiooni koostamine on lihtsam kui EA ja võrreldav tavalise tekstidokumendi kirjutamisega.
- EA kasutamine nõuab rohkem baasteadmisi erinevatest süsteemi komponentidest ja EA jooniste koostamise loogikast.
- EA on täna kasutusel peamiselt süsteemide arhitektuurilise ülesehituse kirjeldamiseks.
    - Süsteemi komponentide omavahelised seosed.
    - Erinevate süsteemide omavaheline integratsioon.
    - Süsteemide integratsioonide spetsifikatsioonid.
- Süsteemide arhitektuurilise vaatega ja üldisemate süsteemide vaheliste integratsioonikirjeldustega on seotud proportsionaalselt vähem Jira pileteid.
- EA teisi erinevaid diagramme ja süsteemide kirjeldamise võimalusi kasutatakse vähem.

EA võimekus spetsiifiliste jooniste ja diagrammide koostamises on väga suur. Süsteemide arhitektuuriline vaade on vaid üks väga väike osa EA tegelikust suutlikkusest infosüsteemide kirjeldamisel. EA diagramme kasutatakse veel ka andmebaasi struktuuride kirjeldamisel ning protsessidiagrammide koostamisel. Ettevõttes on eesmärk EA kasutamist laiemalt juurutada ja selles suunas ka liigutakse. Meeskonnad teevad omavahelist koostööd ning toimuvad arutelud, millisel määral ja millise sisuga infot oleks mõistlik EA-s talletada. Koolitatakse EA kasutajaid, et saavutada ühtlane tase EA jooniste koostamisel. Tõenäoliselt tõuseb EA propageerimisel tulevikus kasutus nii Wikis kui ka Jira piletites.

Projektil koondnimetusega ’TA’ ei leidunud ühtegi viidet Wiki kirjeldusele ega ka EA joonistele. Selle põhjuseks võib olla projekti iseloom. Tegemist oli tehnilist tüüpi töödega, mis olid koondatud eraldi projekti alla. Projekti raames vahetati välja ja uuendati konkreetse süsteemi alusplatvormi ning tehti koodi refaktooringut. Võib arvata, et meeskond ei pidanud vajalikuks Wiki dokumentatsiooni lisada, sest süsteem ise oma funktsionaalsuselt ei muutunud.



Joonis 6. Wiki ja Enterprise Architect kasutusstatistika STORY'des.

6.4 Kasutajatelt kogutud informatsioon

Jira piletite kirjelduste võrdlemisele lisaks küsitleti ka erinevate meeskondade liikmeid, et saada paremat ülevaadet, millisel viisil erinevates projektides nõuete kirjeldamine käib. Konkreetset küsimustikku intervjuude läbiviimiseks ja informatsiooni kogumisel ei koostatud, kuid põhimõtteliselt keskenduti järgmistele punktidele:

- Kuidas kirjeldate erinevaid nõudeid süsteemidele ja süsteemi komponentidele?
- Milliseid vahendeid nõuete kirjeldamisel eelistatult kasutate (Jira, Wiki, EA, tavaline tekstidokument, mõni muu vahend)?
- Millisel kujul nõudeid kirjeldate (*Use-case*, *User-story*, tekstiline kirjeldus, mõni muu lahendus)?
- Kuidas seote nõuded süsteemide ja konkreetsete süsteemikomponentidega?

- Kui on vaja mingi olemasoleva süsteemi kohta kehtivaid nõudeid üles leida, siis kuidas seda teha saab?

- Kas ja kuidas nõudeid versioneeritakse?

Küsitluste ja vestluste läbiviimine andis palju kasulikku tagasisidet hetkeolukorrast ja ka tähelepanekuid parandusvõimaluste kohta.

6.4.1 Nõuete kirjeldamine üldiselt

Üle ettevõtte kehtivat ühtset nõuete kirjeldamise protsessi ei ole kokku lepitud. Seetõttu on ka kirjeldamise tase väga erinev. Tuleb kahjuks tunnistada, et kirjeldamine on väga eklektiline. Paljudel juhtudel kirjeldavad nõudeid äritellijad oma arendussoovides. Hilisemas analüüsi faasis ja arendusprotsessi käigus eraldi nõudeid juurde ei lisata. Põhjus võib olla selles, et mitmed meeskonnad tunnevad oma süsteeme liiga hästi ning äritellimuse põhjalt on juba aru saada, et mida soovitakse saavutada. Spetsiifiliste nõuete kirjeldamine tundub sellisel juhul asjatu lisatööna.

Iga projekt on nõuete kirjeldamise vaatest „analüütiku nägu“. Ühtsed kokkulepped puuduvad ja kõik kasutavad endale sissejuurdunud harjumusi. Enamasti keskendutakse tulemusele ja võib-olla vähem korrektsele nõuete kirjeldamisele, mille tegelik kasu võib tekkida alles hiljem, kui süsteemi peaks olema vaja muuta või täiendada. Vahel võib juhtuda, et projekti võtab üle teine meeskond, kus on harjutud töötama veidi teistsuguse metoodika järgi. Sellisel juhul kohandutakse kiirema lõpptulemuse saavutamiseks samuti pigem meeskonna tööharjumuste järgi.

6.4.2 Keskkondade kasutus

Arendussoovid algavad enamasti äritellija vajadusest. Harvem on tegu tehnilise vajadusega platvormi asendamiseks vms. Esmastel kohtumistel ja aruteludel äritellijatega koostatakse plaane või kavandeid erinevate tarkvaraliste lahenduste abil. Kasutatakse tavalist tekstidokumendi koostamist kirjeldamiseks ja teemapunktide kirja panemiseks, visandatakse jooniseid jne. Sageli ei jõuta aruteludest kaugemale ehk arendustellimust ei tekigi ja siis ei ole mõttekas kohe hakata Wikisse dokumentatsiooni koostama. Kui projekt saab kooskõlastuse ja rahastuse, siis on põhjust alustada projekti kirjeldamist Wiki keskkonnas.

Nagu Jira piletite analüüsist selgus, siis enamasti on Wikis iga projekti kohta mingi dokumentatsioon olemas (Joonis 6). Kui Wikis on dokumentatsiooni vähem, siis selle võrra on rohkem infot Jira piletite kirjeldustes. Põhjus miks üks keskkond domineerib, võib tulla juba äritellijalt. Kui äritellijal on piisavalt selge ettekujutus arendustöödest, mida on vaja arendada, siis koostab äritellija juba ise suurema hulga Jira pileteid ja Wikisse jõuab vähem infot. Neid kahte keskkonda kasutatakse paralleelselt ja väga tihti kasutatakse ka omavahelist viitamist, et vältida dubleerimist.

Info dubleerimine ja sellega kaasnevad probleemid tulid mitmetest vestlustest välja. Kui sama info on kirjas mitmes keskkonnas, siis seda peab hoidma sünkroonis, mis omakorda lisab keerukust ja vajab täiendavat ajalist ressurssi. Kui sünkroniseerimist ei tehta järjepidevalt kogu projekti arendustsükli jooksul, siis vähemalt projekti lõppedes peaks info olema kõikides keskkondades sama. Vastasel juhul ei ole hiljem võimalik vasturääkivuste korral otsustada, milline info on tõene.

Konkreetselt nõuete endi kirjeldamine varieerub samuti. Kasutatakse Jira pileti AC andmevälja või lihtsalt kirjutatakse nõuded pileti kirjelduse andmevälja algusesse. Teise variandina kirjeldatakse nõuded Wiki lehel ja Jira piletisse lisatakse viide Wiki dokumendile. Mõlemal variandil on oma plussid ja miinused.

- Kui nõuded on Wiki lehel ja neid on koos mitmelt Jira piletilt, siis Jira piletit arendades on arendajal keeruline aru saada, milliseid nõudeid selle Jira pileti raames tuleb realiseerida. Samamoodi võib segane olukord tekkida ka Jira pileti testimisel. Kasu võiks olla Wiki lehel Jira piletile viitamisest. Positiivse poolena on jälle süsteemi vaates nõuete hoidmine ühes kohas ja nende hilisem ülesleidmine lihtne.

- Kui nõuded on Jira pileti küljes, siis on olukord täpselt vastupidine. Jira piletit on lihtsam arendada ja testida, kuid hilisem nõuete otsing on keerukam.

Wiki kasutamisega kerkis esile veel üks probleem. Nimelt ei ole Wikis head võimalust dokumente struktureerida erinevatest vaatenurkadest lähtuvalt. Wikis on dokumendid võimalik struktureerida klassikalisse puustruktuuri. Enamatel juhtudel on arendusprojektil oma Wiki leht koos struktureeritud alamlehtedega ja projektide lehed kuuluvad arendusprojektide puusse. See aga ei võimalda saada head ülevaadet näiteks süsteemide vaates erinevatest projektidest. Süsteemivaade aga oleks vajalik mõjude

hindamisel näiteks halduse vaates, milliseid süsteeme konkreetne arendusprojekt mõjutab ja kas mõjutatud võivad olla ka seotud süsteemid.

Süsteemivaatele aitab kaasa EA, kus vähemalt arhitektuur ja integratsiooni joonised peavad olema kirjeldatud. Enne arendusotsusele kooskõlastuse saamist peavad EA joonised läbima arhitektuurinõukogu, kus hinnatakse lahenduse sobivust süsteemide tervikpildi vaates.

Süsteemivaadet aitab toetada ka Jira, kus piletite külge pannakse seos süsteemiga või süsteemi komponendiga. See võimaldab teha päringut üle Jira piletite, mis on konkreetse süsteemiga seotud.

EA kasutus muudes valdkondades kui arhitektuuriline vaade pigem tagasihoidlik. Kasutatakse ka andmebaasi struktuuride kirjeldamiseks, kuid seda enamasti päris uute süsteemide korral, millel ei ole andmebaasi veel olemas. See on ka mõistetav, sest EA suudab joonise põhjal andmebaasi koostamise laused ise valmis genereerida. Olemasolevate süsteemide andmebaasistruktuure on EAs vähem.

Mõned analüütikud ja äritellijad kasutavad EAd ka protsessijooniste ja muude mudelite koostamisel. See annab küll hea ülevaate analüütikule endale ja võimaldab ka teistel osapooltel paremini lahendustest aru saada. Visuaalne pilt on sageli paremini ja lihtsamini mõistetav kui pikk teksti kujul koostatud kirjeldus. Samas võtab nende jooniste haldus ja muude dokumentidega sünkroniseerimine omajagu aega. Lisaks peab EA jooniseid koostades silmas pidama, et kasutataks üldtuntud standarditele vastavat notatsiooni, et erinevad osapooled joonistest samamoodi aru saaksid.

6.4.3 Nõuete kogumine

Tarkvara arendustsükkel algab esmalt äritellija vajadusest lahendada mingi probleem. Tellija peab koostama piisava tasemega üldise kirjelduse, mille põhjal saaks otsustada ligikaudse tööde mahu. Umbmäärast ressursivajadust on vaja projekti otstarbekuse ja kasumlikkuse hindamiseks.

Enamasti suhtleb äritellija juba algse idee formuleerimiseks vastava süsteemi analüütikute ja arhitektidega. Loodavale süsteemile tekivad esmased ärinõuded, mis hilisemas analüüsi ja arendus etapis muutuvad järjest detailsemateks ja konkreetsemateks süsteeminõueteks.

Äritellijatega peetavate arutelude käigus tekib palju infot, mis võib olla väga killustatud. Koosolekutelt tekivad memod, vahetatakse e-kirju, koostatakse dokumente, suheldakse *online*-keskkondades jne. Kogutud infot salvestavad analüütikud erinevalt. Parema ülevaate saamiseks ja lahenduse visualiseerimiseks kasutatakse FreeMind, MindNode või mõnda muud vastavat rakendust. Kasutatakse ka EA jooniseid, kuid näiteks kahte eelnimetatud rakendust on sageli lihtsamad kasutada. Suurt spetsifikatsiooni dokumentatsiooni enamasti ei koostata. Vajadusel koostatakse lihtsam teemapunktide nimekiri koos täiendavate selgitustega. Selle koostamisel kasutatakse mõnd primitiivset tekstiredaktorit või ka Microsoft Office perekonda kuuluvat Excelit või OneNote rakendust.

Kui projekt on saanud heakskiidu ja on otsus see arendusse võtta, siis võimalusel alustatakse Wikis dokumenteerimist või ka juba üldisemate Jira piletite loomist. See väldib liigset dubleerimist ja erinevates süsteemides andmete sünkroniseerimist. Püütakse vältida olukorda, kus oluline info on mõnes e-kirja tekstis või selle manuses, sest see killustab informatsiooni ja ülevaade kokkulepetest võib kaduda. Kui kasutada Wiki ja Jira keskkondi, siis saab suhtlemisel kasutada ainult viiteid nimetatud keskkondadesse ja dokumente endid ei pea e-kirja vms teel vahetama.

Wiki ja Jira kasutamisel on veel eeliseid.

- Dokumentatsioon asub kõigi kasutajate jaoks ühes kohas ja nii ei teki dokumentidest erinevaid versioone.
- Kõik kasutajad saavad ühes ja samas kohas paralleelselt dokumentatsiooni muuta.
- Dokumentatsiooni saab kommenteerida.
- Wiki tekitab iga lehemuudatusega uue versiooni ja vanemad versioonid on lihtsasti jälgitavad. Lisaks saab näha ka kes ja millal konkreetset lehte muutis ning mida lehel muudeti.
- Jira muudatuste ajaloost on näha kommentaaride ja teiste väljade muudatus, muudatuse tegija ja muutmise aeg.

6.4.4 Nõuete kirjeldamine

Eelistatud variant nõuete kirja panemiseks on kasutuslugude ehk *use-case*'ide kasutamine. Kui alati ei ole kasutusloo kirjeldamine võimalik, siis kasutatakse ka tavalist tekstilist kirjeldamist tekstipunktidena. Kasutuslugudele lisaks on kasutusel ka äritellija kirjeldatud ärireeglid süsteemi omaduste kohta. Vahel võivad kasutuslood või ärireeglid puududa, siis kasutatakse ühte nendest kirjeldustest, mis olemas on. Kindlat reeglistikku ei ole ja nõuete kirjeldused on ebaühtlased.

User-story kaudu nõuete kirjeldamist kasutatakse harvem.

6.4.5 Nõuete sidumine süsteemidega ja süsteemi komponentidega

Jira keskkonnas on loodud nimekiri erinevatest infosüsteemi komponentidest:

- Kasutajaliidesed
- Keskkonnad
- Rakendused
- Andmebaasid
- Veebilehed
- jne.

Jira piletitele märgitakse juurde seotud komponent või mitu komponenti. See võimaldab üldisemas vaates Jiras kirjeldatud nõuded komponendi tasemel omavahel ära siduda. Rohkem kindlaid kokkuleppeid nõuete ja süsteemide sidumise kohta tehtud ei ole. Jira pileti juures komponendi määramine on tegelikult seotud arenduste paigaldamisega. See võimaldab süsteemihalduritel üheselt aru saada, millistesse süsteemidesse ja mida on vaja paigaldada.

Sageli on nõuete seosed süsteemidega kirjeldatud Wikis tavadokumendina. See teeb keeruliseks nõuete süstematiseerimise erinevate süsteemide vaates.

6.4.6 Nõuete otsing

Konkreetse projekti analüüsi ja arendusprotsessi jooksul nõuete haldus ja vajadusel mõne nõude ülesleidmine ei valmista ühelegi projektimeeskonnale mingit probleemi.

- Analüütik teab, kuidas ja kuhu ta nõuded kirjeldas.
- Väiksemate projektide puhul on enamik nõudeid analüütikul või teistel meeskonnaliikmetel peas ja operatiivselt omavahel suheldes lihtsasti leitavad.
- Arendajad teavad, millised nõuded said süsteemi arendatud.
- Testijad teavad, kus on kirjas nõuded, mille põhjal süsteemi sai testitud.

Keeruliseks läheb nõuete leidmine siis, kui projekti valmimisest on juba mõnda aega mööda läinud. Vahetuvad meeskonnaliikmed, vahetuvad terved meeskonnad, võetakse kasutusele uusi tarkvaralisi keskkondi/süsteeme, milles nõudeid hallatakse jne.

Olemasoleva süsteemi kehtivate nõuete leidmise probleemidega on kõik meeskonnad kokku puutunud. Ühtset kohta, kust kiiresti vajalikku infot leida, ei ole. Kuna nõuete kirjeldamisel kasutatakse erinevaid lähenemisi ja dokumenteerimise tase on kaootiline, siis seda enam on nõuete leidmine hiljem väga keeruline või pea võimatu.

Otsinguks kasutatakse teemaga seotud võimalikke märksõnu, mille põhjal tehakse kas Wikis või Jiras otsinguid ja üritatakse soovitud nõudeid üles leida. Väga vanade süsteemide korral võivad nõuded olla ka tavadokumentidena, mis asuvad veel vanas dokumendihoidlas Livelink. Livelink lubab küll dokumendi sisust märksõnu otsida, kuid keeruline on hinnata, et kas otsitavate nõuete kohta üldse on Livelinki keskkonnas mõni dokument või seda ei leita üles või on see sootuks kustutatud. Praegu Livelinki keskkonda analüüsi dokumentatsiooni enam ei salvestata.

Teine suur probleem nõuete leidmisel on nõuete kirjeldamise kvaliteet. Kui isegi suudetakse leida kehtivad nõuded, siis ei saa alati kindel olla, et need nõuded ka sellisena realiseeritud on. Harvad ei ole juhud, kus nõuete dokumendid on koostatud projekti algusfaasis. Projekti arenedes on nõuded muutunud ning nõuete dokumentatsiooni ei ole uuendatud. Sellisel juhul ei jää tavaliselt muud üle, kui arendajal rakenduse koodi analüüsida ja üritada välja selgitada, millised nõuded on realiseeritud ja millised mitte.

Projekti dokumentatsiooni ajakohastamisele ei aita kuidagi kaasa ka agiilsete arendusmetoodikate juurutamine. Vastupidi, agiilses arendusprotsessis sageli ei jää selleks piisvalt aega ja probleem võib pigem süveneda.

6.4.7 Nõuete versioneerimine

Sama nõude muutumist ajas ei jälgita üheski arendusmeeskonnas. Nagu nõuete otsingu probleemi puhul välja toodi, on nõuete enda üles leidmine juba keeruline ülesanne, sest sageli ei ole nõuete dokumentatsioon ajakohastatud. Seda naiivsem oleks loota, et nõuete erinevate versioonide dokumenteerimine oleks paremas olukorras.

Kui kõik nõuded oleksid kas Wiki dokumendis või Jira piletite juures kirjeldatud, siis on võimalus vaadata dokumendi muudatuste ajalugu. Wiki lehtede muudatused ja Jira piletite muudatused on mõlemad jälgitavad. Lihtne on näha, milline kasutaja on mida muutnud ja millal muudatus tehti. Praktikas võib see lahendus aga suurte ja mahukate dokumentide korral olla ebamugav.

6.4.8 Ettepanekud ja soovitused

Nõuete halduses valitseva kaootilisusega ja sellega kaasnevate probleemidega nõustusid kõik. Samuti tõdesid kõik, et lihtsat lahendust on keerukas välja pakkuda. Võiksid olla teatud kokkulepped nõuete kirjeldamisel ja lahenduste dokumenteerimisel, et tase oleks ühtlasem. Samas tekib aga kohe oht protsessi üle reguleerida ja seada liigseid piiranguid.

Tekkida võivad probleemid liigselt reglementeeritud ja keeruka nõuete haldusprotsessi korral.

- Tekitab liigset bürokraatiat.
- Kulutab liialt ressurssi. Peamiselt ajalist-, kuid see tähendab ühtlasi ka rahalist ressurssi.
- On keerukas kasutada.
- Vajab ümberõpet/ümberharjumist tänastelt meeskonnaliikmetelt.
- Tekitab liigset stressi.
- Võib olla keerukas omandada uutel meeskonnaliikmetel.

Hetkeolukorra plussiks on meeskondadele antud vabadus arendustööde teostamiseks. Kõik saavad suhteliselt vabalt valida, millises keskkonnas dokumenteerida ja ka mida dokumenteerida. Põhiline orienteeritus on võimalikult kiiresti ja kvaliteetselt jõuda valmislahenduseni. Sellest tingituna dokumenteeritakse süsteemide kirjeldusi paljuski põhimõttel, et nii vähe kui võimalik ja nii palju kui vajalik.

Veel toodi välja järgmisi olulisi põhimõtteid, mida peaks järgima.

- Nõuete haldamiseks peaks olema üks kindel koht, et ei tekiks erinevaid versioone dokumentatsioonist.
- Paralleelselt sama informatsiooni ei tohi kopeerida erinevatesse keskkondadesse, sest muudatuste korral peab infot sünkroniseerima ning see on lisatöö.
- Omavahelises suhtluses kasutatakse viiteid allikale (Wiki, Jira, või EA lingid) ja dokumendi enda sisu ei kopeerita meilidesse vms.
- Nõuete kirjeldamise keskkonnale peavad kõik meeskonna liikmed omama juurdepääsu.
- Erinevad osapooled peavad saama ise dokumentatsioonis muudatusi teha või kommenteerida.
- Kõik muudatused peaksid olema jälgitavad.

7 Nõuete haldusprotsessi võimalikud arendussuunad

Alljärgnevas punktis võtame kokku uurimustöö käigus kogutud hetkeolukorra kirjelduse ja intervjuude läbiviimise teel saadud informatsiooni. Lisades kogutud informatsioonile kaasaegsed nõuete halduse arengusuunad agiilses arendusprotsessis, saab välja pakkuda erinevad lahendused, mille kasutuselevõtmist on võimalik ettevõttes kaaluda.

Tuginedes teadusartiklitele ja nõuete halduse protsessi tänasele olukorrale vaadeldavas ettevõttes, saab üsna kindla veendumusega väita, et ühte konkreetset ja kõigile projektidele/meeskondadele sobivat lahendust ei ole võimalik leida. On erinevaid näiteid kus sarnane nõuete haldusprotsess ei ole ühesuguselt efektiivne sarnastes arendusprojektides ja sarnaste meeskondade struktuuri juures. Alati leiab nüansse, mis sobivad ühele meeskonnale paremini kui teisele meeskonnale ja vastupidi.

Autori poolt on toodud välja neli võimalikku valikut, mida valideeriti meeskonnaliikmetelt tagasisidet küsides . Erinevate variantide arv on taotluslikult mitte eriti suur, et vähendada liigseid omavahelisi kattuvusi lahenduse osas ja hoida piisavat konkreetsust valiku tegemiseks. Kaks esimest lahendust on tehnilisemad ja sisaldavad endas lisaks nõuete halduse protsessi väljatöötamisele ka täiendavat tarkvaralist tuge loodavale protsessile. Kaks viimast lahendust on pigem töökorralduslikud kokkulepped ühtse protsessi juurutamiseks ja loodavast protsessist parima võimaliku kasu saamise variandid.

Valiku tegemisel tuleb arvestada paljude aspektidega. Võib küll tekkida soov ideaalse nõuete haldusprotsessi loomiseks, kus oleks väga täpselt kirjeldatud, milline on protsess ja milline on reeglistik selle protsessi täitmiseks. Teisalt aga peab arvestama ka kasutajate soovide ja vajadustega seda ideaalset protsessi kasutusele võtta. Võib küll juhtuda, et loodud protsess õnnestub disainida täpselt selline, nagu kasutajatele on sobiv ning protsessi juurutamine sujub tõrgeteta. Paraku on aga piisavalt palju näiteid, kus ka hästi läbimõeldud protsess ei juurdu ja seda ei hakata kasutama.

Vähemoluline aspekt ei ole ka vajaliku ressursi mõõde protsessi väljatöötamisel. Kui tulemuse kasulikkuses ning edukuses ei olda piisavalt kindlad, siis ei ole mõttekas

kulutada ka ülemäära palju nii ajalist- kui ka rahalist ressurssi protsessi arendamisele ja juurutamisele.

Järgnevates punktides on lahti kirjutatud nelja võimaliku lahendusvariandi põhimõtted.

7.1 Eraldiseisev tarkvaraline rakendus nõuete halduseks

Üheks võimalikuks variandiks on nõudeid hallata täiesti eraldiseisvas tarkvaralises keskkonnas ja tänastest töövahenditest sõltumatult. Selleks võib luua ise ettevõtte siseselt sobiva tarkvaralise lahenduse, mis arvestaks täpselt kõigi spetsiifiliste vajadustega vaadeldava ettevõtte kontekstis. Võimalik on osta ka sobiv valmislahendus, mis toetaks nõuete haldusprotsessi vajaliku funktsionaalsusega.

Ainult nõuete haldusprotsessi toetamiseks loodud tarkvaral on rida eeliseid.

- Enda loodud nõuete haldussüsteemi saab disainida täpselt ettevõtte soovidest ja vajadustest lähtuvalt.
- Vajadusel saab süsteemi edasi arendada ja lisada puuduolevat funktsionaalsust või olemasolevat funktsionaalsust täiendada.
- Valmislahenduse saab valida võimalikult hästi sobiva vastavalt vajadustele.
- Valmislahendusi üldjuhul aja jooksul arendatakse edasi ja muudetakse paremaks.

Teisest küljest kaasnevad eraldiseisva tarkvara korral ka puudused.

- Enda loodud tarkvara väljatöötamine võib osutuda rahaliselt kulukaks.
- Täiendusvajaduste arendamine nõuab rahalist ressurssi.
- Valmislahenduse ostmine ja hilisem täiendamine eeldab rahalisi väljaminekuid.
- Eraldiseisev lahendus on efektiivne nõuete halduse vaates, kuid puudub seos teiste süsteemidega, mis täna toetavad arendusprotsessi.
- Keeruka lahenduse korral võib osutuda vajalikuks süsteemi kasutamise koolituste läbiviimine, millega kaasneb töötajate ajalise ressursi kulu.

- Töötajatel ei ole motivatsiooni uut süsteemi kasutama hakata, kui selle kasulikkuses ei olda kindlad.

7.2 Eraldiseisev tarkvaraline rakendus koos liidestusega olemasolevatesse süsteemidesse

Teise valikuvariandina on eraldiseisev tarkvaraline lahendus nõuete haldamiseks sarnaselt eelmise punktiga, kuid mis omab lisaks liidestusi teiste tarkvaraarendusprotsessi toetavate süsteemidega (Jira, Wiki, EA). Enda koostatud tarkvara korral on võimalik liidestused disainida ettevõtte vajadustele sobivalt, kuid piiranguks võivad osutuda liidestavad süsteemid ise. Käesoleva töö skoobis ei olnud Jira, Wiki ja EA liidestusvõimaluste põhjalikum uurimine ning seetõttu ei saa täpsemalt hinnata, kas liidestusi oleks võimalik disainida ja arendada piisavalt paindlikuks.

Turult võib leiab ka sarnaseid valmislahendusi, mis omavad liidestusi erinevatesse süsteemidesse. Kui kasutada liidestatud valmislahendust, siis peab rahulduma valmislahenduse funktsionaalsusega ja liidestusvõimalustega, mis on vastavas valmislahenduses implementeeritud.

Positiivsed ja negatiivsed aspektid on käesoleva valiku korral samad, mis on kirjeldatud eelmises punktis. Lisaks eelnevale tuleb arvestada ka liidestamisega kaasneva keerukusega, millel on ajaline ja rahaline kulu. Lisanduvad ka riskid, et sobivat liidestust ei õnnestu disainida. Enda loodud tarkvara korral võib seda piirata liidestavate süsteemide võimekus sobivaid liidestusvõimalusi pakkuda. Valmislahenduse korral tuleb esmalt valida omadustelt ettevõttele sobiv rakendus ja see omakorda peab võimaldama liidestust ettevõttele sobivate rakendustega. Võimalik, et sobivat lahendust ei õnnestu turult leida või tuleb teha kompromisse funktsionaalsuses või liidestuste omaduste osas. Täiendav positiivne omadus võrreldes eelmise punktiga seisneb loomulikult võimaluses tänaseid keskkondi (Jira, Wiki, EA) edasi kasutada.

7.3 Nõuete haldus ühes tänastest keskkondadest

Kolmanda lahendusena võib vaatluse alla võtta ühe tänastest arendusprotsessi toetavatest keskkondadest ja hallata nõudeid selles valitud keskkonnas. Nõuete halduse protsessi saab kirjeldada vastavalt valitud keskkonna omadusi silmas pidades. Arendusprotsessi

toetavateks keskkondadeks ja võimalikuks nõuete halduse keskkonnaks võib olla Jira, Wiki või EA.

Olemasoleva keskkonna valikul saab välja tuua järgnevad eelised.

- Vähene ressursivajadus – rakendused on juba kasutusel ja uue tarkvara arenduskulud või valmislahenduse soetuskulud puuduvad.
- Koolitusele ei pea kulutama aega – kasutajad oskavad keskkondasid kasutada ning täiendavat erikoolitust ei vaja
- Vähem stressi – sõltuvalt loodava protsessi keerukusest ei pea arendusmeeskonnad enda käitumisharjumusi väga palju muutma. Konkreetne muutmisvajadus sõltub muidugi sellest, kui suured on erinevused meeskonna tänaste harjumuste ja loodava protsessi vahel.

Samuti on ka käesoleva valiku puhul mõned ohud.

- Kõigis kolmes keskkonnas on nõuete haldamisvõimalused erinevad. Kui valida ainult üks keskkond, siis ei pruugi see valik katta kõikide arendusprojektide/arendusmeeskondade vajadusi.
- Liiga keerukalt disainitud protsess võib tekitada kasutajates vastumeelsust, ning protsessi juurutamine ei pruugi õnnestuda.

7.4 Nõuete haldus erinevates tänastes keskkondades kombineeritult

Viimane valitud lahendusvariant on nõuete haldusprotsess, mis kasutab kõiki kolme tänast keskkonda ja kus nõudeid kirjeldatakse samuti kõigis kolmes keskkonnas. Oluline on siin silmas pidada, et mitte kõiki nõudeid ei kirjeldata paralleelselt kõigis kolmes keskkonnas, vaid vastavalt näiteks projekti iseloomule valitakse ka keskkond, milles nõudeid hallatakse.

Eelmise punktiga võrreldes on eelised ja ohud samad. Lisanduvaks eeliseks on keskkondade vahel kombineeritud haldusprotsessil paindlikumad võimalused nõuete kirjeldamiseks. Samas suureneb ka oht keerukama protsessi kujunemiseks, sest valikuvõimalused nõuete kirja panekuks suurenevad.

8 Kasutajatelt kogutud tagasiside analüüs

Eelmises peatükis välja pakutud neljale võimalikule lahendusvariandile paluti kasutajate grupil anda tagasisidet. Eesmärgiks oli selgitada välja eelistatud arengusuunad, mille põhjal saaks otsustada edasised tegevused. Kasutajate kaasamine nõuete haldusprotsessi väljatöötamisse juba algses faasis on õigustatud käitumine. Tulevased protsessi kasutajad saavad aktiivselt kaasa mõelda ja teha ettepanekuid. Selliselt väldime olukorda, kus kasutajad näevad valmis disainitud protsessi alles peale selle valmimist. Saame siin tuua ühese paralleeli planeerimisel põhineva *vs* agiilise arendusmetoodika vahel. Lisaks aitab kasutajate varajane kaasamine vähendada uuendustega kaasnevat stressi, sest tulevaste muudatustega ollakse aegsasti kursis.

Palusime kasutajatel nelja välja pakutud lahendust hinnata ja moodustada pingerida, millised lahendused oleksid paremad ja millised lahendused ei pruugiks tänases arendusprotsessis väga edukad olla. Lisaks hinnangule soovisime ka põhjendusi, millele tuginedes nende arvamus kujunes.

Tagasiside tulemusi analüüsides saab väita, et suund tuleks vähemalt esialgu võtta kahe viimase versiooni poole ehk eelistatakse töökorralduslikku protsessi. Olemasolevad tarkvaralised töövahendid võiksid katta ära nõuete halduse vajadused ja pigem on probleem ühtsete reeglite puudumises ning ühtses protsessis kokku leppimises. Täiesti uue rakenduse kasutuselevõtt ei pruugi anda piisavalt lisandväärtust, et selle arendamisele ja juurutamisele oleks mõistlik aega ja raha kulutada. Ja kui ka tekib olukord, kus olemasolevad rakendused ei ole oma võimalustelt piisavad ning vajavad täiendamist, siis saab uue rakenduse arenduse teha alati tulevikus.

Erinevate lahendusvariantide poolt- ja vastuargumente on võimalik võrrelda koondtabelis (Tabel 1). Koostatud tabelist on lihtne näha, et eelistatav on valida tänaste keskkondade vahel ja vältida uue keskkonna lisandumist. Uue keskkonna lisamine toob endaga kaasa täiendava rahalise ja ajalise ressursivajaduse ning lisab juurutamise keerukust. Lisaks võib uue keskkonnaga kaasnev keerukus vähendada kasutajate motivatsiooni seda

kasutusele võtmast, sest keegi kasutajatest ei näinud, et uue keskkonna kasutuselevõtt oleks vajalik.

Tabel 1. Lahendusvariantide valikuargumentide koondtabel.

	Rahaline ressursivajadus	Ajaline ressursivajadus	Juurutamise keerukus	Kasutajate motiveeritus
Eraldiseisev rakendus	Suur	Suur	Suur	Väike
Eraldiseisev rakendus koos liidestusega	Suur	Suur	Suur	Keskmine
Üks tänastest keskkondadest	Väike	Keskmine	Keskmine	Keskmine
Tänaste keskkondade kombinatsioon	Väike	Väike	Väike	Suur

Edasi keskendumegi kahele võimalikule lahendusele põhjalikumalt ehk nõuete haldus ühes keskkonnas või nõuete haldus üle kõigi keskkondade kombineeritult. Järgnevate punktide kirjeldused nõuete halduse kohta erinevates keskkondades, on koostatud kasutajate poolt saadud informatsiooni põhjal.

8.1 Nõuete haldus Jiras

Konkreetsete arendusülesannete vaates on Jira piletite alla nõuete kirjeldamine kõige mugavam lahendus. Arendamisele minev Jira pilet on enamasti piisavalt konkreetne ja äritellijalt saadud sisendi põhjal saab analüütik või ka tooteomanik kirjeldada vajalikud nõuded. Arendaja saab luua lahenduse vastavalt kirjeldatud nõuetele. Testija saab testida valminud arendust vastavalt Jira piletis kirjeldatud nõuetele.

Jira pileti juures on nõuete kirjeldamine sobilik lahendada kahel erineval viisil:

- *Acceptance criteria* – kirjeldatakse konkreetsed tingimused, millele loodud arendus peab vastama ja mille põhjal saab hinnata, kas saavutati soovitud tulemus

- *Description* – pikema kirjelduse lisamiseks mõeldud andmeväli Jira pileti koosseisus

Kuigi nõudeid võib kirjeldada ka väljal *Description*, siis eelistatum on kasutada *Acceptance criteria* välja. See annab kindluse, et kõik vastava Jira pileti täitmiseks vajalikud nõuded on ühel konkreetsel väljal. *Description* väljal nõuete kirjeldamise korral on ühes kohas koos erinevat informatsiooni ja konkreetseid nõudeid võib olla tülikas üles leida.

Nagu varasemalt sai kirjeldatud, siis kasutame suurema tervikliku funktsionaalsuse arendamisel EPIC tüüpi Jira pileteid, mille alla koondatakse väiksemad STORY tüüpi Jira piletid konkreetsete arendusülesannete teostamiseks. Kui nõuded kirjeldada *Acceptance criteria* väljal, siis saab tekitada nõuete struktuuri. EPIC tüüpi pileti alla kirjeldatakse üldisema taseme nõuded, mis võivad olla koostatud juba äritellimuse vormistamise käigus. STORY tüüpi pileti alla kirjeldatakse konkreetsed ja detailsemad nõuded, mille kaudu realiseeritakse üldisemad nõuded.

Nõuete hilisemaks otsinguks saab kasutada Jira keskkonna enda otsingumootorit. Otsida saab Jira pileteid väga paljude erinevate parameetrite järgi ja päringu tulemustesse tagastatavad Jira pileti andmeväljad on samuti konfigureeritavad. Päringu tulemusi on võimalik eksportida erinevatesse vormingutesse, mille abil saame Jira piletites olevat informatsiooni töödelda väliste rakenduste abil.

Nõuete hilisemal otsingul annavad päringud seda paremaid tulemusi, mida ühtlasemalt ja süstemaatilisemalt Jira piletites täidetakse lisavälju. Täna kasutatakse aktiivsemalt kahte lisavälja:

- *Component/s* – kasutatakse käesoleval hetkel Jira sidumiseks konkreetse süsteemiga või konkreetsete süsteemidega. Täitmine on kohustuslik, sest selle välja järgi saab otsustada, millistesse süsteemidesse tuleb valmis arendused paigaldada.

- *Labels* – silt, mille abil saab viidata mingile temaatikale, millesse konkreetne Jira pilet võib kuuluda

Mõlemal eelpoolmainitud väljal võib olla üks või ka mitu väärtust. Lisatavad väärtused võib valida varem kasutatud väärtuste hulgast ja lisaks saavad kasutajad ka ise uusi valikuid lisada. Võimalike väärtuste hulk tuleks üle meeskondade kokku leppida. Uute väärtuste lisamine peaks olema koordineeritud keskselt ühes kohas. Kui uute siltide või komponentide võimalikke väärtuseid saavad kasutajad luua oma äranägemise järgi, siis tekib suurel hulgal tähenduselt sarnaseid silte, mis pikemas perspektiivis lisab segadust.

8.2 Nõuete haldus Wikis

Wiki oma eesmärgilt on mõeldud üldisema dokumentatsiooni hoidmiseks. Siia sobivad nii äritellijate poolt kirjeldatavad arendussoovid ja samuti ka analüütikute poolt koostatud detailsemad süsteemide spetsifikatsioonid. Wiki dokumentide koostamisvõimalused on väga mitmekesised. Tekstile saab lisada erinevaid tabeleid, nimekirju, pilte, diagramme jne. Saab kasutada linkimist teistele objektidele. Valikus palju vorminguid erineva sisuga objektide kuvamiseks, mis lihtsustab koostatud dokumentide lugemist ja sisust arusaamist.

Konkreetseid nõudeid saab samuti Wikis kirjeldada, kuid nende hilisem leidmine üldise dokumentatsiooni seest võib osutuda keerukaks. Otsingut on võimalik Wikis teha täisteksti otsinguna. Kui vaadelda otsinguvõimalusi konkreetselt nõuete kontekstis, siis peaks olema nõuded kuidagi eristatavad üldistest muudest kirjeldustest. Selleks tuleks sellisel juhul kehtestada konkreetne reeglistik nõuete kirja panemiseks. Samas oleks nende reeglite korrektset täitmist keerukas tagada ja kontrollida.

Wiki keskkonnas saab lehtedele lisada silte ehk märksõnu. Siltide kasutamine aitab leida sama temaatikat hõlmavaid Wiki lehti. Nagu ka Jira keskkonnas siltide kasutamist tuleks keskselt juhtida, siis sarnane on olukord ka Wiki keskkonnas. Kui keskset kontrolli ei ole, siis on oht, et silte tekib liiga palju ja siltide põhjal lehtede otsing ei anna loodetud kasu.

Wiki dokumendid jagatakse struktuuridesse, mis võimaldab koostada dokumentide jaotused teemade lõikes. Hästi läbi mõeldud Wiki lehtede struktuur lihtsustab info leidmist. Teisest küljest ei võimalda ühe struktuuri kasutamine grupeerida lehti mõne muu vaatenurga põhjal, kui seda on loodud struktuur. Kui näiteks struktuur luuakse tehnilisest

vaatest lähtuvalt, siis on mugav kasutada süsteemi vaadet ja näha hõlpsasti süsteemide vahelisi seoseid. Samas aga protsessi vaade, kus protsessi käigus osalevad mitmed erinevad süsteemid, eeldaks teistsuguse struktuuri loomist. Siin oleks üheks lahenduseks siltide kasutamine, mis aitaks Wiki lehti erinevates struktuuriharudes üheks loogiliseks koguks kokku liita.

Konkreetselt nõuete kirjeldamise ja otsingu vaates on Wiki siiski liialt kohmakas. Ka väga hästi dokumenteeritud ja struktuurselt korrektselt grupeeritud arendusprojekti korral ei ole Wikis võimalik lihtsasti ülesse leida ainult nõudeid, mis on süsteemile esitatud. Samuti ei saa me rääkida mingistki raporti koostamisest, milles oleksid kirjas näiteks ühe konkreetse funktsionaalsuse realiseerimiseks vajalikud nõuded.

8.3 Nõuete haldus EAs

EAs on nõuete halduseks suuremad võimalused võrreldes Jira ja Wiki keskkondadega. Nõude kirjeldamiseks on EAs eraldi nõude tüüpi komponent, millel on rida atribuute ning suuremat kasutamist võiksid leida nendest järgmised:

- Nimetus – nõude nimetus
- Kirjeldus – nõude pikem kirjeldus
- Tüüp – Funktsionaalsed- ja mittefunktsionaalsed nõuded.
- Staatus – ettepanek, valideeritud, kinnitatud, implementeeritud.
- Alias – saab panna nõudele varjunime.
- Võtmesõnad – märksõnad sammase valdkonda kuuluvatele nõuetele viitamiseks.
- Prioriteet – madal, keskmine, kõrge.
- Versioon – nõude muutumisel saab määrata millise versiooniga nõudest on tegemist.

Nõudekomponendi võib siduda EAs suvalise muu komponendiga suvalisel joonisel või diagrammil. Sama komponenti võib siduda mitme erineva muu komponendiga, mis tagab

selle, et üks nõue on alati ühekordselt kirjeldatud ja ei ole dubleeritud mitmes erinevas süsteemis ja erinevatel joonistel.

Eesmärgiks on seatud, et protsessijoonised oleksid kõik kirjeldatud EAs. Kõrgemate tasemete protsessijoonised koostatakse äritellija poolt, kuhu saab juba lisada üldisemaid nõudeid äriprotsessi sammudele ja tegevustele. Madalama taseme protsessijoonised koostatakse juba koostöös analüütikutega ja lisatakse spetsiifilisemad ja detailsemad nõuded, mida saab kasutada lahenduse realiseerimisel.

Lisaks protsessijoonistele on EAs ka süsteemide kirjeldused ja süsteemide omavahelised integratsioonidiagrammid. Mida detailsemalt on integratsioonidiagrammid ja liidestuste spetsifikatsioonid kirjeldatud, seda paremini saab kirjeldada konkreetsetele süsteemidele, süsteemikomponentidele ja liidestussõlmedele kehtivaid nõudeid.

Nõuete otsinguvõimalused on EAs head samal põhjusel, et kasutusel on eraldi nõuete komponent. Lihtne on leida nõuete seoseid teiste komponentidega ja vastupidi ehk. komponendile kehtivaid nõudeid.

Ettevõtte seisukohalt on väga kasulik nõude ühekordne kirjeldamine. Sama nõude kohta ei tohiks olla süsteemis ühtegi teist dubleerivat nõuet. Kui nõue on korrektselt seotud süsteemi komponentidega, mille osas vastav nõue kehtib, siis nõude muutumisel on lihtne näha muudatusega kaasnevat laiemat mõju. Selline mõjuanalüüs aitab välja tuua võimaliku ressursivajaduse ning hinnata muudatuse majanduslikku otstarbekust.

8.4 Kombineeritud nõuete haldus

Kui süveneda nõude mõistesse üldisemalt, siis võib nõuet tõlgendada kui ettekirjutust, reeglit, tingimust vms. Järelikult võivad nõuded olla seotud väga erinevate äriprotsesside või süsteemide komponentidega. Nõuded võivad olla seotud äriprotsessi sammudega, äriprotsessi tegevustega, süsteemidega, süsteemi suuremate- või väiksemate komponentidega süsteemi mittefunktsionaalsete parameetritega jne. Sellest tulenevalt on ka nõuete tüübid erinevad ja erineda võib ka nõuete kirjeldamise formaat. Saab püstitada küsimuse, et kas kõiki nõudeid üldse saab mugavalt kirjeldada ühes keskkonnas olenemata sellest, et kas vaatleme meie kasutada olevaid keskkondi või erinevaid võimalikke keskkondi laiemalt. Palju on näiteid, kus võimalikult universaalne lahendus disainitakse kas liiga primitiivseks, mille funktsionaalsus ei ole piisav või siis vastupidi

liiga keerukaks, et kõik vajadused oleksid kaetud. Viimase variandi puhul on oht, et süsteemi kasutamine on liiga ebamugav ja aeganõudev.

Kui me võtame aluseks põhimõtte, et nõuded võivad olla kirjeldatud erinevates süsteemides, siis saame vastavalt tüübile nõude kirjeldada sobivasse keskkonda. Saame ära kasutad neid eeliseid mida keskkonnad meile pakuvad ja võimalusel vältida erinevate keskkondade puuduseid või nende puuduste mõju vähendada.

Nagu ka varem on korduvalt mainitud, siis tänaste kokkulepete kohaselt on arendusülesannete täitmisel põhiliseks töövahendiks Jira. Seega on kõige detailsemate süsteemi nõuete kirjeldamise loogiliseks asukohaks needsamad Jira piletid ja *Acceptance criteria* andmeväli.

Arhitektuurilised süsteemikirjeldused ja integratsioonidiagrammid on kokkuleppeliselt kirjeldatud EAs. Siis on mõistlik EAs kirjeldada ka tehnilisema sisuga nõuded süsteemidele, süsteemi komponentidele ja süsteemide omavahelistele liidestustele.

Selleks, et vältida nõuete dubleerimist ja sellega kaasnevat hilisemat keskkondade omavahelist sünkroniseerimist, tuleb kasutada keskkondade vahel võimalikult palju viitamist. Harilikke otselinke saab lisada erinevatesse kirjeldustesse, kuid mõistlik on teha siingi teatavad kokkulepped, et erinevate süsteemide kirjeldused oleksid sarnase struktuuri järgi koostatud.

Jira piletite atribuutide struktuuris on andmeväli *Link To Documentation*. Siia saab lisada näiteks viite Wiki dokumentatsioonile.

Wiki lehel saab kasutada otseviiteid Jira piletitele. Viide oskab kuvada lisaks Jira pileti numbrile ja pealkirjale ka Jira pileti staatust. See võimaldab Wiki dokumenti lugedes aru saada, et kas viidatud Jira pileti raames arendatav funktsionaalsus on juba realiseeritud või on veel arendamisel.

EA on võimalik siduda nii Jira kui ka Wiki keskkondadega. EA komponendi ja Jira pileti vahele saab luua ühese seose. Kuna EA liidestusvõimalused väliste rakendustega arenevad pidevalt, siis selles osas võib kindlasti oodata koostöövõimaluste paranemist erinevate süsteemide vahel.

Jira ja Wiki poolt EA joonistele viitamine ja EA jooniste kasutamine on tehtud võimalikult mugavaks. Kuna EA enda rakendus on piisavate kogemusteta kasutaja jaoks liiga keeruline, siis saab jooniste lugemiseks kasutada EA veebiliidest. EA töökeskkonnas on võimalik genereerida URL, mis viitab EA vastavale joonisele EA veebikeskkonnas. Genereeritud URL sisaldab endas unikaalset viidet konkreetsele joonisele. Kui joonist muudetakse, siis viide joonisele jääb samaks ja genereeritud URL viitab alati õigele joonisele.

9 Järeldused ja soovitused

Täiesti uue eraldiseisva nõuete haldusvahendi või täiesti uue ja tänasest erineva nõuete haldusprotsessi kasutuselevõtt ei leidnud kasutajatelt piisavalt toetust. Teisalt ei ole ka tänane olukord nii lootusetult halb, et seda ei võiks üritada täiendada ja paremaks muuta. Uus lahendus oleks kindlasti võimalik disainida nõuete halduse ja agiilse arendusprotsessi vaates tänasest efektiivsemaks. Samas ei ole mingit garantiid, et saavutatav tulemus annaks sellisel määral kasu, mis õigustaks uuele lahendusele vajaliku ressursi kulutamise.

Hetkeolukorra kaardistamise ja kasutajatelt informatsiooni kogumise käigus joonistus selgelt välja muster tänaste kitsaskohtade osas. Välja toodud probleeme põhjalikumalt analüüsides on võimalik järeldada, et olukorra parandamiseks saab välja pakkuda rida lihtsamaid lahendusi, mida kirjeldame järgmistes punktides täpsemalt.

9.1 Ühtne nõuete kirjeldamise reeglistik

Nõuete kirjeldamise kvaliteedi parandamiseks peab üle erinevate arendusmeeskondade ja arendusprojektide kehtima ühtne nõuete haldusprotsess ning nõuete kirjeldamise reeglistik. Selliselt on tagatud nõuete kirjeldamise ühtlane struktuur olenemata arendusprojektist või süsteemist. Alljärgnevalt selgitame põhimõtteid, millest tuleb lähtuda ühtse nõuete kirjeldamise reeglistiku koostamisel.

9.1.1 Nõuete kirjeldamine vastavalt nõude tüübile ja äriprotsessi kirjeldusele

Klassikaliselt jagunevad nõuded kaheks – funktsionaalsed- ja mittefunktsionaalsed nõuded. Samas võib nõudeid jaotada ka teistest põhimõtetest lähtuvalt, näiteks millisele osale tarkvaralisest terviklahendusest nõue kuulub:

- Süsteem
- Süsteemi komponent
    - Kasutajaliides
    - Teenus

    - Andmebaas

- Süsteemide omavaheline integratsioon

- Arhitektuur.

Kui lähtuda vaatluse all oleva ettevõtte senisest tööpraktikast ja senistest kokkulepetest töö korraldamisel, siis kõige otstarbekam on nõude kirjeldamine selles süsteemis, millesse vastav nõue loogiliselt kuulub.

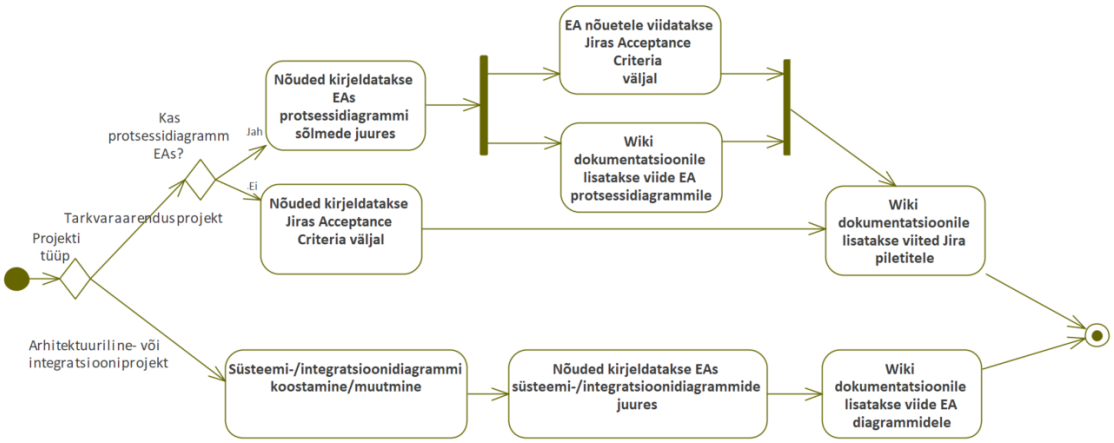
Süsteemide ja süsteemi komponentide arendamiseks koostatakse arendajatele Jira piletid. Seega on loogiliselt sobiv ka süsteemile või süsteemi komponendile kehtivad nõuded kirjeldada Jira piletite juures. Jira pileti sobiv andmeväli nõuete kirjeldamiseks on *Acceptance criteria*. Nõuete kirjeldamine eelpoolmainitud väljal lihtsustab olulisel määral kõigi arendusprotsessis osalevate meeskonnaliikmete tööd. Konkreetse töö juures on konkreetsel väljal kirjeldatud täpsed nõuded, mida antud töö tulemusel realiseeritakse. Äritellija koos analüütikuga kirjeldab vajadused *Acceptance criteria* väljale, arendaja saab vastavalt kirjeldatud nõuetele teha vajalikud arendused ning testija saab sama välja alusel testida valminud arenduse korrektsust.

Acceptance criteria väljale tuleb nõuded kirjeldada piisavalt konkreetsed ja terviklikud, et ilma täiendava informatsioonita oleks võimalik aru saada, millist funktsionaalsust vastava nõude realiseerimisel saavutatakse. Vältida tuleb olukordi, kus nõuded kirjeldatakse Jira pileti *Description* väljale ja jäetakse *Acceptance criteria* väljale kirjeldamata. *Description* väljale saab lisada Jira piletit puudtavat täiendavad informatsiooni ning vajadusel viidata *Acceptance criteria* väljal kirjeldatud nõuetele.

Süsteemide arhitektuuri diagrammid ja süsteemide vahelised integratsioonidiagrammid kirjeldatakse kokkuleppeliselt EAs. Seetõttu on vastavate nõuete kirjeldamise loogiliseks keskkonnaks samuti EA. Arhitektuurijoonised annavad suure tervikpildi süsteemidest ja nendevahelistest seostest ning joonistega seotud nõuded näitavad detailsemalt süsteemide ja integratsioonisõlmede omadusi.

Mitmed meeskonnad kasutavad EA keskkonna pakutavaid võimalusi ka äriprotsesside kirjeldamiseks. EA keerukusest tingituna ei ole see käitumismuster siiski veel valdav. Kui aga äriprotsess on kirjeldatud EAs, siis on äriprotsessist parema ülevaatlikkuse saamiseks otstarbekas kirjeldada nõuded samuti äriprotsessi joonise juures EAs. Sellisel juhul tuleb

arendusülesannete Jira piletite *Acceptance criteria* väljal viidata EAs protsessijoonisel kirjeldatud nõuetele.



Joonis 7. Nõuete kirjeldamise protsessidiagramm.

Vastavalt projekti tüübile ja äriprotsessi kirjelduse olemasolule EAs, saab koostada protsessidiagrammi, mis näitab millistel tingimustel ja millistes keskkondades tuleb nõuded kirjeldada. Vastav protsessidiagramm on toodud joonisel (Joonis 7). Kui tegemist on arhitektuurilise- või integratsiooni projektiga või kui on tegemist tarkvaraarendusprojektiga, kus äriprotsessidiagramm on kirjeldatud EAs, siis kirjeldatakse ka nõuded EAs, vastasel juhul kirjeldatakse nõuded Jira piletite juures *Acceptance ariteria* väljal. Tarkvaraarendusprojekti korral lisatakse EAs äriprotsessidiagrammi olemasolul Jira *Acceptance criteria* väljale viide EAs kirjeldatud nõuetele. Lisaks lisatakse Wiki dokumentatsioonile viited Jira piletitele. Arhitektuurilise- või integratsiooni projekti korral lisatakse Wiki dokumentatsioonile viited EAs koostatud diagrammidele.

Täna puuduvad ühtsed kokkulepped nõuete kirjeldamise keskkondade ja konkreetsete andmeväljade osas. Iga arendusmeeskond ja iga arendusprojekt võib seda ise otsustada. Nõudeid on nii Wikis, Jira piletites erinevatel andmeväljadel ja ka EAs. Selleks, et nõuded oleksid ühesuguse struktuuri järgi kirjeldatud, tuleb üle meeskondade ja projektide kokku leppida kindel protsess ja ühtne reeglistik. Kokkulepete olemasolul saab nõuda nendest kokkulepetest kinni pidamist ja kontrollida nende täitmist meeskonnaliikmete poolt.

Ühtse struktuuri järgi nõuete kirjeldamine on väga suureks abiks mõne süsteemi või süsteemi komponendi kohta kehtivate nõuete ülesleidmisel. Tänases olukorras, kus meeskondadel on vabadus otsustada dokumenteerimise ja nõuete kirjeldamise struktuuri osas, peame nõudeid otsima kõigist kolmest keskkonnast. Ja isegi siis, kui oleme kõigis kolmes keskkonnas kõik teadaolevad nõuded tuvastanud, ei saa me kindlad olla, et mõnes meeskonnas ei kasutata veel mõnd täiendavat keskkonda nõuete kirjeldamiseks ja millest ei ole nõudeid sünkroniseeritud meie kolme töökeskkonda. Ühtne kokkulepitud struktuur lihtsustab olulisel määral nõuete leidmist. Nõuete pärimiseks erinevates keskkondades saab kasutada nendesamade keskkondade päringumootoreid, kui me teame millises keskkonnas ning millise struktuuri põhjal nõuded on kirjeldatud.

9.1.2 Dubleerimise vältimine

Üks oluline reegel, millest nõuete kirjeldamisel tuleb rangelt kinni pidada, on ühe ja sama arenduse raames ühe ja sama nõude kirjeldamine ühes keskkonnas. Kui sama nõue on mitmes keskkonnas samaaegselt kirjeldatud, siis võib see tekitada palju probleeme.

- Nõude muutumisel tuleb nõude kirjeldust muuta mitmes keskkonnas.
- Mitmes keskkonnas samade nõuete kirjelduste sünkroniseerimine nõuab täiendavat ajalist ressurssi.
- Kui nõuete kirjelduste sünkroniseerimine ei ole automaatne, siis võib ette tulla olukordi, kus erinevates keskkondades on sama nõue kirjeldatud erinevalt. Sellisel juhul ei ole võimalik tuvastada, et milline kirjeldus on korrektne ja milline mitte.

Kõige rohkem probleeme tekitab just viimati mainitud olukord. Kui meil on mitu erineva sisuga nõuet, mis tegelikult kehtivad sama funktsionaalsuse kohta, siis korrektse nõude väljaselgitamine võib nõuda mitmete meeskonnaliikmete ajalise- ja seega ka rahalise ressursi kasutamist.

Dubleerimise vältimiseks tuleb kasutada erinevate keskkondade vahel viitamist. EA korral saab viidata konkreetsele nõude komponendile. Jira korral saab viidata konkreetsele Jira piletile. Viidete kasutamine aitab vältida dubleerimisega kaasneda võivat segadust.

Kui lähtuda nõuete kirja panekul protsessist, mida iseloomustav protsessidiagramm on kirjeldatud eelmises punktis (Joonis 7.), siis samu nõudeid ei dubleerita ja nõuete kirjeldusi ei ole vaja sünkroniseerida erinevate keskkondade vahel. Oluline on vaid jälgida, et õiged viited oleksid lisatud vastavatesse keskkondadesse sõltuvalt nõude enda kirjeldusest. Teine oluline aspekt, mida tuleb silmas pidada, on see, et juba kirjeldatud nõuet ei muudetaks ega asendataks kergekäeliselt mõne teise nõudega. Alati tuleks ennem veenduda et asendatavale nõudele ei ole viidatud mõnest teisest keskkonnast.

10 Kokkuvõte

Käesoleva magistritöö eesmärgiks oli nõuete haldamise protsessi välja töötamine ühele konkreetsele ettevõttele. Nõuete haldusprotsess peab toetama ettevõttes kasutusel olevat agiilset tarkvaraarendusprotsessi ja SAFe tarkvaraarendusraamistikku. Ühtne nõuete haldusprotsess peab tagama parema nõuete kirjeldamise kvaliteedi, mis omakorda lihtsustab erinevate süsteemide kohta kehtivate nõuete tuvastamise.

Tulemuse saavutamiseks viis autor läbi kvalitatiivse uuringu hetkeolukorra kõige problemaatilisemate kitsaskohtade väljaselgitamiseks. Uuringu läbiviimiseks vajaliku informatsiooni kogumiseks kasutati vaatlust, mille käigus kaardistati tänane nõuete haldusprotsess. Lisaks viidi läbi intervjuud tarkvaraarendusmeeskondade liikmetega. Tuginedes tänapäevastele teadusartiklitele, mis kirjeldavad nõuete haldusprotsesse kaasaegses agiilses tarkvaraarendusprotsessis võrreldes klassikalise tarkvaraarendusprotsessiga, teostati uuringu käigus kogutud informatsiooni analüüs. Saadud tulemuste põhjal koostati võimalikud lahendusvariandid, mida tutvustati tarkvaraarendusmeeskondade liikmetele. Ühtlasi valideeriti lahendusvariantide sobivust ning koguti erinevate lahendusvariantide kohta tagasisidet.

Välja pakutud lahendusvariandid keskendusid paljuski tarkvaralistele keskkondadele, milles ja kuidas nõudeid tuleks uue protsessi kohaselt kirjeldada. Vaatluse all oli mitmeid võimalusi: uute keskkondade loomine nõuete haldamiseks; olemasolevate keskkondade ja uute keskkondade kombineerimine; tänaste olemasolevate keskkondade kasutamine, kus oleks kokku lepitud kindel reeglistik nõuete kirjeldamiseks.

Kasutajatelt saadud tagasisidet analüüsides oli selgelt näha, et uus täiendav keskkond nõuete kirjeldamiseks ja haldamiseks ei ole eelistatud lahendus. Uue keskkonna lisandumine ei anna nõuete haldusprotsessi mõistes mingit täiendavat lisandväärtust võrreldes tänaste keskkondade kasutamisega. Kindlasti kaasnevad aga uue keskkonna loomiseks ja kasutamise juurutamiseks suuremad ressursivajadused ning suureneb nõuete haldamise keerukus. Kasutajate eelistus on läheneda probleemile töökorralduslikult ja kasutada tänaseid keskkondi nõuete haldamisel.

Autor pakub oma töö tulemusena välja konkreetse nõuete haldusprotsessi, mis vastab algselt seatud tingimustele. Toetatud on nii agiilse tarkvaraarendusprotsessi vajadused, kui ka SAFe tarkvaraarendusraamistik. Protsess baseerub tänastel arendusprotsessi toetavatel tarkvaralistel keskkondadel ning seab konkreetsed reeglid nõuete kirjeldamisele. Vastavalt arendusprojekti tüübile ja äriprotsessi kirjeldusele tuleb nõuete haldusprotsessi kohaselt nõuded kirjeldada kindlas keskkonnas. Lisaks määratakse kindlaks nõuetele viitamise reeglid erinevates keskkondades vältimaks nõuete dubleerimist. Ühtse nõuete haldusprotsessi kasutuselevõtt tagab parema nõuete kirjeldamise kvaliteedi ja nõuded on alati leitavad konkreetsetes keskkondades olenemata süsteemist või arendusprojektist.

Kasutatud kirjandus

[1] L. Õunapuu, Kvalitatiivne ja kvantitatiivne uurimisviis sotsiaalteadustes, Tartu Ülikool, 2014.

[2] Melaku Girma; Nuno M. Garcia; Mesfin Kifle, „Agile Scrum Scaling Practices for Large Scale Software Development," *2019 4th International Conference on Information Systems Engineering*, Shanghai, China, mai 2019.

[3] Richard Berntsson Svensson, Sigurdur Örn Birgisson, Christian Hedin, Björn Regnell, „Agile Requirements Abstraction Model – Requirements Engineering in a Scrum Environment," *Computer Science*, 2008.

[4] Eric Knauss; Grischa Liebel; Jennifer Horkoff; Rebekka Wohlrab; Rashidah Kasauli; Filip Lange; Pierre Gildert, „T-Reqs: Tool Support for Managing Requirements in Large-Scale Agile System Development," *2018 IEEE 26th International Requirements Engineering Conference*, 20-24 august 2018.

[5] E. Knauss, „The Missing Requirements Perspective in Large-Scale Agile System Development," *IEEE Software (Issue 3)*, mai-juuni 2019.

[6] Eva-Maria Schön; Jörg Thomaschewski; María José Escalona, „Agile Requirements Engineering: A systematic literature review," *Computer Standards & Interfaces (Volume 49)*, jaanuar 2017.

[7] J. Cleland-Huang, „Disruptive Change in Requirements Engineering Research," *2018 IEEE 26th International Requirements Engineering Conference*, Banff, AB, Canada, 20-24 august 2018.

[8] Wentao Wang; Nan Niu; Hui Liu; Zhendong Niu, „Enhancing Automated Requirements Traceability by Resolving Polysemy," *2018 IEEE 26th International Requirements Engineering Conference*, Banff, AB, Canada, 20-24 august 2018.

[9] Muhammad Shafiq; Qinghua Zhang; Muhammad Azeem Akbar; Arif Ali Khan; Shahid Hussain; Fazal-E Amin, „Effect of Project Management in Requirements Engineering and Requirements Change Management Processes for Global Software Development," *IEEE Access (Volume: 6), 11.05.2018.*

[10] Muhammad Azeem Akbar; Nasrullah; Muhammad Shafiq; Jawad Ahmad; Muhammad Mateen; Muhammad Tanveer Riaz, „AZ-Model of software requirements change management in globaal software development," *2018 International Conference on Computing, Electronic and Electrical Engineering (ICE Cube)*, Quetta, Pakistan, 12-13 november 2018.

[11] Karina Villela; Anne Hess; Matthias Koch; Rodrigo Falcao; Eduard C. Groen; Jörg Dörr; Carol Naranjo Valero; Achim Ebert, „Towards Ubiquitous RE: A Perspective on Requirements Engineering in the Era of Digital Transformation," *2018 IEEE 26th International Requirements Engineering Conference*, Banff, AB, Canada, 20-24 august 2018.

[12] Philipp Lombriser; Fabiano Dalpiaz; Garm Lucassen; Sjaak Brinkkemper, „Gamified Requirements Engineering: Model and Experimentation," *REFSQ 2016: Requirements Engineering: Foundation for Software Quality*, Gothenburg, Sweden, 4. märts 2016.

[13] A. Sapunkov; T. Afanasieva, „Software for Automation of User Requirements Prioritization," *ICGDA 2019 Proceedings of the 2019 2nd International Conference on Geoinformatics and Data Analysis*, Prague, Czech Republic, 15-17 märts 2019.

[14] Poranat Tianual; Amnart Pohthong, „Defects Detection Technique of Use Case Views during Requirements Engineering," *ICSCA '19 Proceedings of the 2019 8th International Conference on Software and Computer Applications*, Penang, Malaysia, 19-21 veebruar 2019.

[15] Elizabeth Bjarnason; Markus Borg, „Aligning Requirements and Testing: Working Together toward the Same Goal," *IEEE Software (Volume: 34 , Issue: 1 , Jan.-Feb. 2017).*

[16] Muhammad Abbas; Irum Inayat; Mehrdad Saadatmand; Naila Jan, „Requirements Dependencies-Based Test Case Prioritization for Extra-Functional Properties," *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops*, Xi'an, China, 22-23 aprill 2019.

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Rainer Mulk

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Nõuete haldamise protsessi väljatöötamine konkreetse ettevõtte näitel“, mille juhendaja on Jaak Tepandi
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

04.01.2021

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.