TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Anne Schults 143046IAPB

Reliisi kvaliteedi hindamise meetrika analüüs ja prototüübi arendus ettevõtte näitel

Bakalaureusetöö

Juhendaja: Inna Švartsman
Magistrikraad

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: Anne Schults

18.05.2021

Annotatsioon

Lõputöö eesmärgiks on välja selgitada konkreetse ettevõtte ja selle projekti näitel, millised on reliisi halduse puhul korrektsed meetrikad, mida peaks jälgima, et tuua kõigile reliisidest huvituvatele osapooltele võimalikult palju väärtust. Pärast nõuete koondamist loodi veebirakendus, mis kuvab eelmainitud info ja annab reliisile ja sellele eelnenud tegevustele kvaliteedi hinnangu.

Töö tulemusena koostati nimekiri funktsionaalsetest, mittefunktsionaalsetest nõuetest ja loodi prototüüp lahendus, mis on eelkõige kasulik antud projekti tiimile. Samas võiks käesolev töö huvi pakkuda ka teistele tarkvaraarenduse suurettevõtetele või spetsialistidele, kes reliiside haldamisega peavad tegelema.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 24 leheküljel, 5 peatükki, 5 joonist, 2 tabelit.

Abstract

The Analysis and Implementation of Release Quality Evaluation Tool in a Software Company

The main goal of the thesis is to analyse and describe the requirements of new release quality evaluation tool in a software company and implement the prototype.

To find the best metrics for evaluating release quality, the author analysed most common practices, a widely known release management tool and the data from previous year's production releases statistics. In the second part of the thesis, author has implemented the prototype while describing the process and outcome.

As a result of this theses a list of functional and non-functional requirements were written and a prototype of this application was built.

The thesis is in Estonian and contains 24 pages of text, 5 chapters, 5 figures, 2 tables.

Lühendite ja mõistete sõnastik

API	*Application Program Interface*, rakendusliides, mille abil on võimalik erinevatel programmidel omavahel suhelda
Confluence (Atlassian)	Veebipõhine korporatiivne wiki
CORS	*Cross-origin resource sharing,* lähtekohtade vaheline ressursside jagamine, on mehhanism, mille abil on võimalik rakendustel omavahel andmeid jagada
Git	Vaba versioonihaldustarkvara
ITIL	*The Information Technology Infrastructure Library*, IT teenuste halduse levinuim praktika juhendite kogum
JavaScript	Objektorienteeritud programmeerimiskeel, mida kasutatakse peamiselt veebilehtede skriptimiseks
JIRA (Atlassian)	Ülesannete haldamise ja projektijuhtimise tarkvara
JSON	*JavaScript Object Notation*, lihtsustatud andmevahetusvorming, mis põhineb JavaScript programmeerimiskeele alamhulgal
Npm	*Node Package Manager*, JavaScripti programmeerimiskeele jaoks loodud paketihaldur
Reliis	*Release,* Riistvara, tarkvara, dokumentatsiooni, protsesside või muude komponentide kogum, mis on vajalik IT teenuse ühe või mitme autoriseeritud muudatuse tegemiseks. Iga reliisi sisu hallatakse, testitakse ja paigaldatakse kui ühte tervikut
Roll-out	Tarkvara kasutuselevõtt (nt uues harukontoris)
SPA	*Single-page application.* veebirakendus, mis kasutab ühte HTML dokumenti kogu rakenduse
Tarne	*Delivery,* protsess toote või teenuse arendamiseks ja avalikustamiseks.
Task	Ülesanne, mis luuakse JIRA keskkonda

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Bakalaureuse töös käsitletav ettevõte tegutseb ülemaailmsel turul transpordi ja logistika valdkonnas. Eestis olev IT harukontor tegeleb tehniliste lahenduste loomise ja oma regiooni süsteemide töös hoidmisega. Käesoleva töö juures vaatleme ühe konkreetse tarkvara projekti näitel reliiside haldamise töövoogu ja üritame seda parendada. Süsteem on pidevas muutumises, kehtestatud on teenustaseme lepped ja nõutav töökindluse tase on kõrge. Juhtkond eelistaks näha stabiilseid ja kõrge kvaliteediga uusi tarkvara versioone.

Tõstatatud on küsimus, milline on õigupoolest kvaliteetne reliis ja kuidas seda hinnatakse. Ainult tarkvara lähtekoodi hinnates saaksime statistilise hinnangu programmeerijate tööle, aga tarkvara elutsüklis on veel mitmeid etappe ja asjaosalisi, mis võivad sama tähtsat rolli mängida. Inspireerituna jaapanlaste *Kaizeni* filosoofiast [1], katsume olla kvaliteedihalduses orienteeritud protsessi parendamisele, võttes kogu protsessi tükkideks lahti ja hinnates iga protsessisammu edukust eraldi.

Antud töö põhieesmärkideks on:

- selgitada välja uuritava projekti jaoks olulisimad meetrikad reliiside jälgimisel;
- formuleerida funktsionaalsed ja mittefunktsionaalsed nõuded prototüübi ehitamiseks;
- töötada välja juhis, mille järgi arvutada iga tarkvara versiooni kohta reliisi koostamise ja tarne kvaliteedi hinnang;
- luua esialgne prototüüp, mis agregeeriks teises punktis välja valitud informatsiooni ja kuvaks kolmandas punktis välja arvutatud kvaliteedi taset.

Lõputöö tulemusel valmiv rakendus on oluliseks abivahendiks reliisi halduritele ja teistele huvitatud osapooltele, lihtsustades ja kiirendades andmete kogumist, raportite koostamist reliiside kohta. Lahendus peaks säästma aega raportite koostamiselt ning võimaldama kiiret tagasisidet kogu projektitiimi koostööle.

Esimeses peatükis antakse ülevaade antud töös käsitletavast ettevõttest projektist, organisatsioonist ja protsessidest.

Teises peatükis töötatakse läbi erinevaid soovitusi ja valmis lahendusi, mis tegelevad reliiside kvaliteedi uurimisega. Peatüki lõpus defineerib autor funktsionaalsed ja mittefunktsionaalsed nõuded antud töö realisatsiooniks.

Kolmandas peatükis kirjeldatakse autori poolt loodud veebirakenduse portotüübi realiseerimise samme ja tuuakse välja soovitused edasiarenduseks.

2 Taust ja probleem

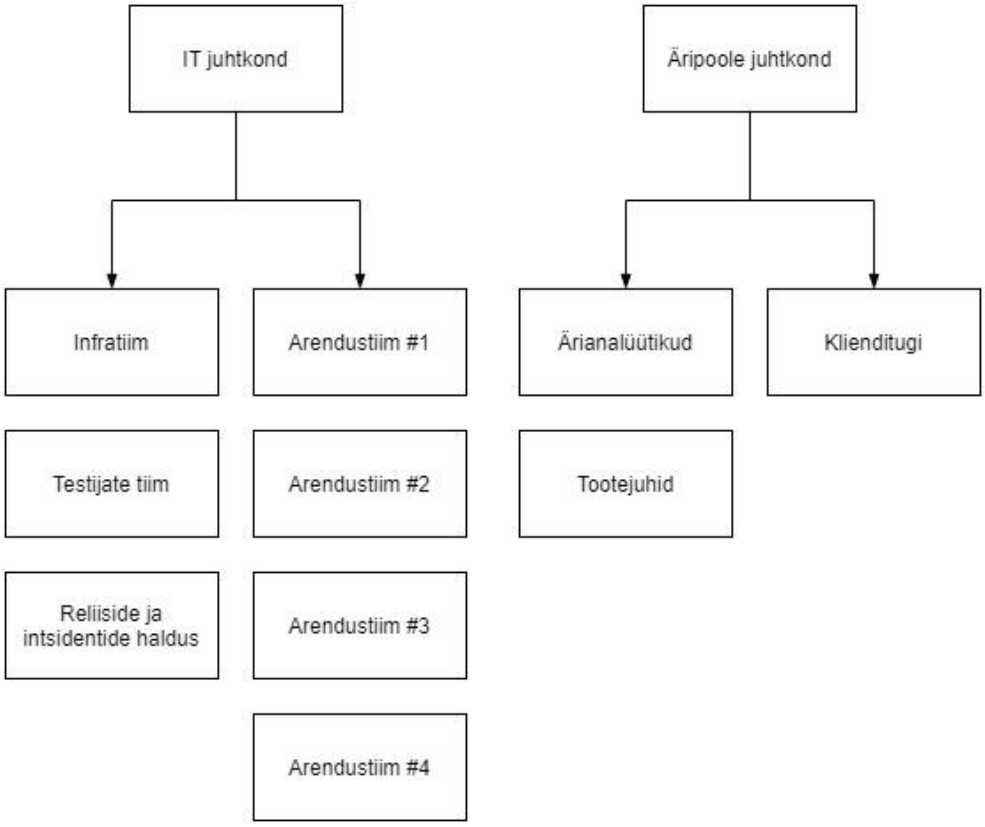
Käesoleva peatüki eesmärk on anda põgus ülevaade antud töös käsitletavast ettevõtte tarkvara tootest, selle IT organisatsioonist, protsessidest ja sealhulgas kirjeldada ära probleem.

2.1 Projekti tutvustus

Analüüsitava projekti näol on tegemist töölaua rakendusega, arendatud Java Swing raamistikus, mida kasutavad ettevõtte harukontorites töötavad logistikud selleks, et planeerida kaupade transporti maanteede kaudu. Rakendusel on ligikaudu 400 igapäevast kasutajat 30-s erinevas Euroopa harukontoris. Projekt on laienemise faasis ning iga paari kuu tagant juurutatakse antud rakendust uutes piirkondades [2].

Antud firmas luuakse suur osa kasutatavatest infosüsteemidest ettevõtte siseselt, siinkohal ongi teenuse tellijaks sama firma äriline pool. Äripoole nõudeid koondavad lõppkasutajatelt ärianalüütikud. Nende ülesandeks on ka *roll-outide* planeerimine ja toote juurutamine. Kõik koondatud nõuded vormistatakse arendajatele sobilike suurustega *taskideks* ja prioritiseeritakse.

Arendustiime on projektis praegu neli, igal tiimil on kusjuures olemas *Scrum-masteri* rolli kandev arendaja ja tehniline arhitekt. Testijate tiim on eraldi arendustiimidest, põhiülesanneteks on automaattestide disain ja implementeerimine. Reliisi ja intsidentide halduse tiim vastutab reliiside planeerimise, läbi viimise ja ka intsidentide koordineerimise eest. Alltoodud joonisel on lihtsustatul kujul esitatud projekti organisatsioon.



Joonis 1. Projekti organisatsiooni struktuur lihtsustatul kujul.

2.2 Tarkvaraarenduse metoodika

Projekti planeerimine algas aastal 2014. Juba tollal teati, et rakendus saab olema sügavat äriloogikat sisaldav, turusituatsioonile kohandatav ja ajas palju muudatusi vaja minev. Tarkvaraarenduse kvaliteedi, efektiivsuse ja klientidele pakutava väärtuse tõstmiseks otsustati kasutada klassikalise koskmeetodi asemel väledaid metoodikaid.

2.2.1 Scrum

Scrum on levinuim iteratiivne ja inkrementaalne väleda tarkvaraarenduse raamistik. Scrumi metoodika väljatöötamist alustati 90-ndate alguses, esimene selleteemaline raamat ilmus aastal 1995, autoriteks Jeff Sutherland ja Ken Schwaber. Scrumi-i idee näeb ette töötamist nn. sprintidena, iga sprindi lõpus esitletakse kliendile tehtud tööd ja ollakse valmis seda tarnima, iga sprindi jooksul omakorda tuleb parendada tööprotsessi [3].

Käsitletava projekti puhul kasutati Scrumi algusest peale, sprindi pikkuseks valiti 2 nädalat, mille igakordseks tulemiks oli reliis, mis testiti ja valideeriti kliendi poolt. Selle

meetodi eelis oli, et kõik reliisid ja nende plaanid olid tiimile teada-tuntud. Arendajad ja tootejuhid oskasid oma tööd planeerida, et õigeaegselt ülesanded lõpetada ja olla kindel, et need muudatused reliisi kaastatud on. Esimene *roll-out* ehk tarne lõppkasutajatele toimus aastal 2017. Järgenvate aastate jooksul toimus veel mitmeid *roll-oute*, mis tõestasid projekti küpsust ning esialgne nn. projekt kasvas välja püsivaks tooteks.

2.2.2 Kanban

Kanbani idee tarkvaraarenduses on üle võetud tootmistööstusest. Tegemist ei ole otseselt tarkvara arenduse metoodikaga vaid muudatuste haldamise metoodikaga. Tähendab, et see on sobiv raamistik juba olemasoleva tarkvara igapäevaseks hoolduseks ja täiendamiseks. Idee on visualiseerida töövoog, piirata tööülesandeid selliselt, et töös oleks korraga mõistlik ja teostatav kogus tööd [4].

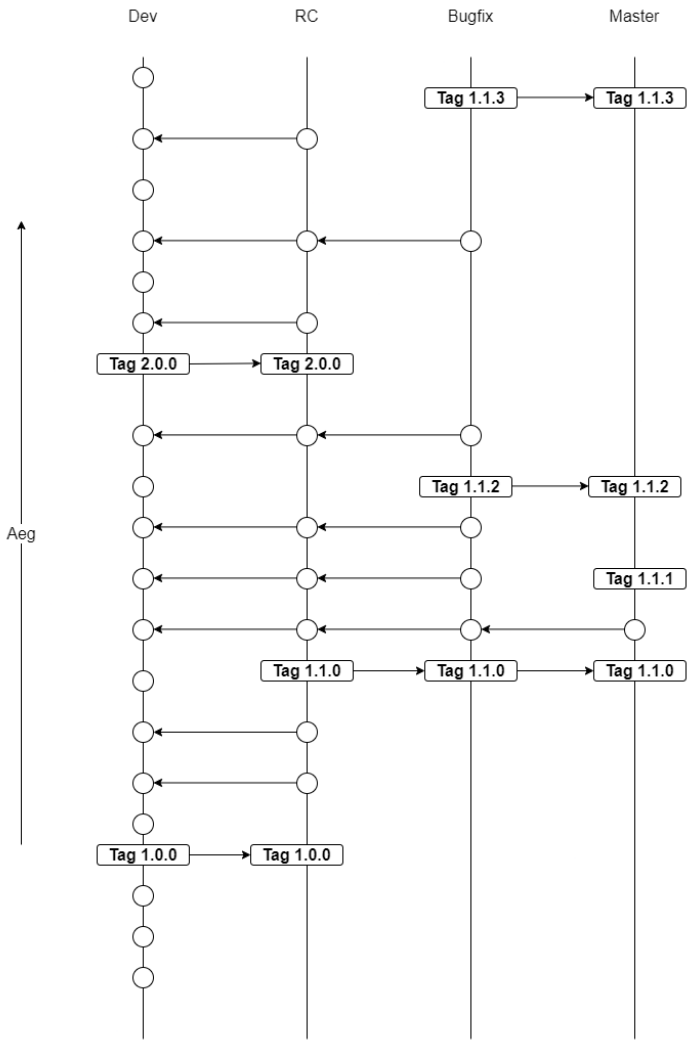
2019. aastal otsustatigi uue metoodika kasuks. Kanbani kasutusele võtmine ei vajanud äkilisi ega suuri muudatusi projekti üldises töökorralduses. Küll aga muutus reliiside tarnimine tavapärase kahenädalase tsükli asemel palju paindlikumaks. Reliise tehakse Kanbani metoodika järgi vajaduspõhiselt, vahel tähendab see 2-3 korda nädalas, suuremate funktsionaalsuste lisandumise korral jälle 1 kord kuus. Reliisi halduse tiimile tähendab see täiendavat koormust planeerimise ja kommunikeerimise vallas.

2.3 Reliisihalduse protsess

Muudatusi tarnitakse toodangukeskkonda igakuiselt ning lisaks reliisitakse vajaduspõhiselt kõrge ja kriitilise prioriteediga veaparandusi.

2.3.1 Git-i harude mudel

GitFlow-na tuntud harude mudel, välja töötatud Vincent Diressen-i poolt 2010.aastal, on projekti töövoo kujundamisel suurt rolli mänginud. [5] Antud mudeli eelised on mitme versiooni samaaegne edasiarendamine ja võimalus *roll-backide* tegemiseks. Algideed on sedavõrd kohandatud, et lisaks põhiharudele *develop* ja *master*, on põhiharudena kasutatakse ka *release-candidate* ja *bugfix* harusid. Neid kasutatakse nii tihti, et nende loomise ja kustutamise asemel neid pigem hoitakse elus ja uuendatakse vastavalt vajadusele. Alltoodud joonisel on näha, kuidas tarkvara versioon mööda harusid kulgeb.



Joonis 2. Harude mudel.

Dev haru kasutatakse peamise arendusharuna. *RC (ingl.k. Release Candidate)* ehk reliisi kandidaadi haru on reliisi stabiliseerimiseks. *Bugfix* haru kasutatakse toodangu veaparandusteks. *Master* haru peegeldab suure osa ajast toodangus olevat koodi, ainult kõige kõrgema prioriteediga veaparandusi tehakse selle haru peal, et need siis *hotfixina* tarnida.

2.3.2 Reliiside tüübid

Projektis tehtavaid reliise annab liigitada neljaks – *internal, major, bugfix, hotfix*.

Internal reliis on tarvara versioon, mida kindlasti ei plaanita toodangusse selles vormis saata, küll aga vajab vahetestimist mõnes testkeskkonnas.

Major reliis on tarkvara versioon, mis tehakse perioodiliselt *DEV* harust. See sisaldab uusi funktsionaalsuseid ja peab läbima stabiliseerimise faasi, kus teostatakse manuaaltestijate poolt ulatuslik regressioontestimine, kestvusega ligi nädal.

Bugfix reliis on tarkvara versioon, mis sisaldab enam kui ühte veaparandust toodangu jaoks. *Hotfix* on seevastu versioon, mis sisaldabki ainult ühte veaparandust ja tarnitakse tavaliselt vaid tundide jooksul.

2.3.3 Reliiside testimine

Automaattestidega kaetus on projektis küllaltki kõrge ning juba arenduse käigus käitatakse pideva integratsiooni serveris kokkupandud testikomplekt täies mahus peale igat rakendusse tehtud muudatust. Selline strateegia lihtsustab vigade leidmist varajases faasis, tagab arendusharude stabiilsuse ja võimaldab igal ajahetkel dokumenteerida tarkvara automaattestide läbivuse tulemust.

Pärast tarkvaraversiooni koostet ja tarnet testkeskkonda teostatakse manuaalsed vastuõtutestid, et verifitseerida tarkvara vastavust lisandunud funktsionaalsetele nõuetele ja kinnitada, et tehtud muudatused loovad kasutajale lisaväärtust. *Major* reliiside korral viiakse läbi ka regressioontestimine manuaaltestijate poolt, veaparanduste reliiside korral seda ei tehta.

2.3.4 Reliiside tarne

Pideva integratsiooni keskkonnana on kasutusel Jenkins, mis vastutab iga versioonihaldusesse jõudnud muudatuse korral kooste, automaattestide ja koodianalüüsi vahendite käitamise eest.

Projektis ei ole veel kasutusele võetud pidevavalmidust ja pidevtarne metoodikaid, seda mitmetel erinevatel põhjustel, nagu näiteks – rakenduse arhitektuurist tulenevalt ei ole võimalik teha versiooniuuendusi seisuajata; vastuvõtutestimine on manuaalne ja tehakse sõltuvalt reliisi koosseisust erinevate andmestikega testkeskkondades; automaattestide arendus käib paralleelselt või pärast arenduse lõppemist, mitte enne; äripoole ja lõppkasutajate vastuseis, kuna see tähendaks ka nende jaoks protsessi muudatusi jmt.

Vastavalt kokkulepitud reliisi graafikule luuakse koodist paigaldatav üksus, mida saab kasutada erinevatesse keskkondadesse paigaldamiseks. Paigaldamiseks kasutatav skript on kõigi keskkondade jaoks sama ning kasutab vastavalt keskkonnale vajalikku

konfiguratsioonifaili. Reliisi protsessi osadeks on ka reliisi kommunikeerimine ja suitsutestimine, mille jooksul täidetakse alamhulk kõigist testidest selgitamaks, kas põhilised funktsioonid töötavad. [6]

2.3.5 Reliisijärgne aruandlus

Pärast reliisi tarnimist toodangukeskkonda tuleb reliisihalduril koostada aruanne Confluence'i wikisse. Selle sisuks on: loetelu muudatustest ja nende sõltuvustest; kuupäevalased tegevused kooste ja tarnete osas; testimiste tulemid nii testijate tiimi kui ka tootejuhtide poolt; tarnele järgnenud intsidendid ja nende raportid.

Aruande valmimisel tuleks ülevaatlik e- mail saata ka kõigile huvi tundvatele osapooltele – arendustiimidele, tootejuhtidele, ärianalüütikutele, testijatele, klienditoele ja juhtkonnale.

Kvartaalselt tuleb esitada koondnimistu kõikidest toimunud reliisidest. Neid esitletakse enamjaolt illustreerivate graafikutena, et lugeja saaks kiire ülevaate. Tähtsaim info on lisatud uute funktsionaalsuste arv, veaparanduse arv, intsidendid ja nende lahendamisele kulunud aja vastavus teenustaseme lepingule.

Nimetatud protsessid on lihtsakoelised ent ajakulukad. Seesuguse töö automatiseerimine ja protsessi asendamine loodava rakenduse poolt pakutava veebiliidese näol lihtsustaks reliisi halduri tööd ja lubaks seevõrra süveneda kõrgema prioriteediga ülesannetesse.

3 Tarkvaranõuete analüüs

Tarkvara nõuete kogumiseks on erinevaid võimalusi, mis kõik aitavad erineval moel huvitatud osapoolte vajadusi välja selgitada ja neid kaardistada. Enimtuntud ja kasutatud analüüsimeetodid on näiteks: intervjuu kasutajatega, töötoa läbiviimine, ajurünnak, küsimustik, vaatlus, prototüüpimine, olemasoleva süsteemi analüüs jne. Autor lähtus analüüsi meetodi valimisel meetodi kasutamisele kuluvast ajast, sisendi saamiseks vajalikust inimeste arvust ja analoogse tööriista olemasolust. Eelnevat arvestades osutusid valituks järgnevad meetodid: ITILi ehk enimlevinud IT-teenuste halduse praktika juhendiga tutvumine, alternatiivse rakenduse analüüs ja olemasoleva informatsiooni kõrvutamine.

3.1 ITILi soovitatavad võtmetegurid

ITIL *(The Information Technology Infrastructure Library)* on IT-teenuste halduse parimate praktika raamistik. Esimene versioon kogumikust töötati välja 1990ndatel. Kuigi 2019. aastal anti ilmus ITIL versioon 4, ei ole seda praktikas veel käesolevas projektis kasutusele võetud ning lähtutakse ITIL versioonist 3, mille koolitused ja juhendid on töötajad saanud.

ITIL defineerib reliisihalduse protsessi eesmärgid järgmiselt [7]:

- reliisi plaanide koostamine ja kokku leppimine;
- tarkvara versioonide kooste, testimine, dokumentatsioon ja virtuaalsetes hoidlates hoiustamise kindlustamine;
- reliiside tarne test- ja toodangukeskkondadesse pärast kõikide kohustuslike sammude läbimist lähtudes plaanidest ja ajakavast;
- tagada uute või muudetud teenuste vastavus kokkulepitud teenustasemele;
- tagada minimaalne ettearvamatu mõju toodangu teenustele ja nende tööle, dokumenteerida tulemid ja/või probleemid;
- tagada klientide, kasutajate ja teenuse halduse meeskonna rahulolu teenuste ülemineku faasis, sealhulgas vajalikud kasutusjuhendid ja koolitused lõppkasutajatele.

ITILi järgi jaotatakse protsess nelja faasi:

- reliisi ja tarne planeerimine;
- reliisi kooste ja testimine;
- paigaldamine;
- läbivaatus ja sulgemine.

Lõputöö eesmärk on otseselt parendada ja automatiseerida viimase faasi tulemit, seega vaatleme lähemalt, milliseid tegevusi ja milliste meetrikate jälgimist ITIL selles järgus ette näeb.

- Koguda lõppkasutajatelt tagasisidet tarnest, nt läbi küsitluste;
- välja tuua kvaliteedi kriteeriumid, mida ei saavutatud;
- kontrollida, et vajalikud tegevused, parandused ja muudatused oleksid korda saadetud;
- läbi vaadata veel avatud muudatuste nõuded ja tagada, et nende muudatuste jaoks on uued plaanid ja ressursid planeeritud;
- kontrollida, et pärast tarnet ei esineks teenuse sooritusvõime, ligipääsetavuse probleeme;
- tagada, et kõik teadaolevad probleemid ja vead on dokumenteeritud ja äripoole poolt aktsepteeritud;
- kontrollida, et teenus on valmis sissetöötamistoest edasi liikuma teenuste käitluse etappi.

Järgnev tabel toob välja ITIL näited kriitilistest eduteguritest (CSF) ja võtmeteguritest (KPI) kõrvutades need autori kommentaaridega ja analüüsi tulemiga, kas vastav meetrika kaasatakse uude tööriista või mitte.

Tabel 1. ITILi näited KPI-dest ja autori kommentaarid.

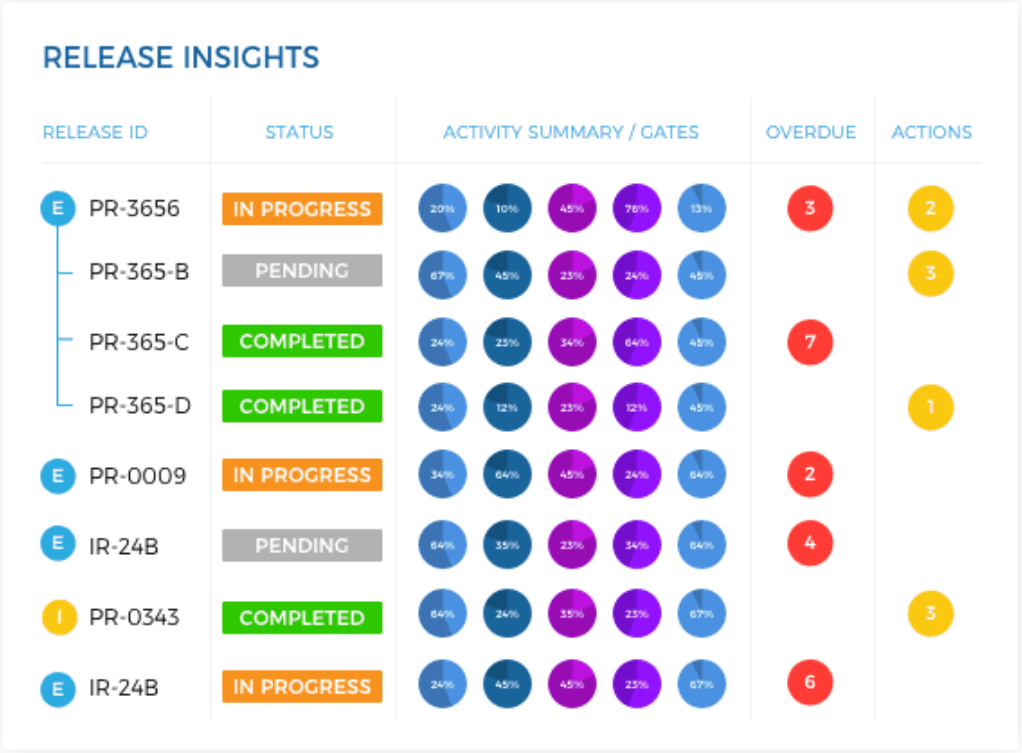
ITILis välja pakutud KPI	**Autori kommentaar**	**Otsus**
Suurenenud reliiside koguarv, mis teostati protsesse järgides.	Loodavas prototüübis peaks loendatud olema kõik reliisid ja peavaatel kuvatud ka reliisi kooste kuupäevad, mis hõlbustab reliiside koondvaate saamist. Küll aga ei ole esialgu plaanis tuua välja reliiside koguarvu perioodi kohta.	Jah
Suurenenud reliiside koguarv, mis vastab tellija ja lõppkasutajate ootustele.	Reliisid, mis ei saa tellija heakskiitu ei jõua toodangusse ja seega nende elutsükkel ei ole terviklik, mis saab olema kajastatud kvaliteedi arvutuses.	Jah
Reliisi järgne konfiguratsioonihalduse süsteemi ja tarkvara püsikogu auditil leitud vigade kogus.	Raske või pea võimatu automeerida.	Ei
Vähenenud reliiside arv, mil tarnet teostatakse muul viisil kui tarkvara püsikogust pärit ressurssidest.	Seda tüüpi probleemi ei esine piisavalt sageli, et seda kaasata hindamismudelisse.	Ei
Vähenenud intsidentide arv, mis tuleneks valede komponentide tarnest.	Kuigi seda on varasemalt juhtunud, ei ole see piisavalt sage nähtus, et seda eraldi kontrollida. Saab olema kajastatud üldiste intsidentide all.	Jah
Vähenenud teenuse sooritusvõime.	Serverite ülekoormusest tingitud intsidente hallatakse taaskord üldiste intsidentide all, sest ei ole kuigi tihti esinev probleem. Küll aga tuleks seda meetrikat rakendada edasiarenduste faasis, sest projekti kasutajaskonda on plaanitud mitmekordistada.	Jah

Intsidentide arv pärast tarnet.	Lihtsasti jälgitav, sest intsidentidest raporteeritakse JIRA ülesannete kaudu, millel on kõrge prioriteet ja eristuv märgend.	Jah
Kasutajate rahulolu muutused.	Seni ei ole seda tüüpi küsitlusi läbi viidud ja lähiajal ei ole ka plaanis, idee on hea ja ilmselt mingil hetkel rakendatakse.	Ei
Vähenenud ressursside ja kulude vajadus, et tuvastada ja lahendada intsidente.	Suur osa tehnilistest ja infrastruktuuri puudutavatest uuendustest tehakse IT-teenuse reliisitsüklist väljaspool, mistõttu on neid keeruline omavahel siduda.	Ei
Vähenenud arv lõppkasutajate poolt raporteeritud intsidente, mille põhjuseks on puudulik kasutusjuhendi tundmine.	Selle valdkonnaga eest vastutab antud projekti juures teine tiim.	Ei
Suurenenud intsidentide arv, mis vajavad esimese või teise taseme tuge.	Ettevõtte definitsioonide kohaselt antud toote puhul esimese taseme tuge ei eksisteerigi, sest tegemist ei ole veel globaalselt kasutatava tootega. Teise taseme alla klassifitseeruvad kõik intsidendid, mis vajavad *hotfix* tüüpi reliisi, seega need on kajastatud intsidentidena.	Jah
Suurenenud rahulolu nii kasutajate, tellija kui ka tiimi silmis kogu reliisihalduse protsessi jooksul.	Praegu ei mõõdeta aga lähitulevikus võiks sedagi rakendada.	Ei

3.2 Olemasolevates rakendustes kasutatavad meetrikad

Isegi tänapäevases moodsal infoajastul leiavad reliisihalduses kasutust just väga lihtsakoelised rakendused, nagu näiteks Microsoft Excel, Sharepoint ja Conluence wiki. Mitmed ettevõtted proovivad tühimikku täita ja pakkuda terviklikku teenust, mis reliisihaldus protsessi lihtsustaksid. Tuntuimaid neist on näiteks Plutora, CloudBees,

Circle CI. Esmasel vaatlusel selgus, et Plutora poolt pakutav lahendus sarnaneb kõige enam sellele (vt. joonis 3), mida antud bakalaureuse töö autor enda töös üritab realiseerida. Seetõttu siinkohal vaatleme täpsemalt, missuguseid funktsionaalsusi, mõõdikuid tööriist pakub ja millised neist on mõeldav üle võtta.



Joonis 3. Plutora rakenduse põhivaade reliisidest [8].

Allolev tabel toob välja Plutora rakenduse juures kasutatavad reliisihalduse funktsioonid [9] kõrvutades need autori kommentaaridega ja analüüsi tulemiga, kas vastav meetrika kaasatakse uude tööriista.

Tabel 2. Olemasoleva tarkvara meetrikad ja autori kommentaarid.

KPI-d olemasoleva tarkvara põhjal	**Autori kommentaar**	**Otsus**
Reliiside sageduse visualiseerimine.	On tähtis jälgida, sest reliiside suur number on positiivne.	Jah
Lähtekoodi kaetus ühiktestidega.	Võetakse arvesse üldise koodianalüüsi käigus ja kajastub seal.	Jah

Kasutajate, tellija rahulolu.	Nagu varasemalt mainitud, seda praegusel hetkel ei mõõdeta.	Ei
Äriväärtuse loome, ROI.	Sellist infot haldab ärianalüüsi valdkond.	Ei
Lõpetatud *taskide* arv reliisis.	Saab olema tööriistas kuvatud aga kvaliteedi arvutamisel peaks autori arvates võrdlema pigem planeeritud ja tegelikult lõpetatud *taskide* arvu erinevust.	Jah
Reliisi elutsükli läbimiseks kulunud aeg.	Võiks olla asjakohane meetrika, kui projektis kasutataks pidevvalmiduse ja pideva tarne metoodikaid. Käesoleva projekti puhul tekivad pudelikaela efekti pikad manuaalsed vastuvõtu- ja regressioontestid.	Ei
Testkeskkondade kasutatus ja konfiguratsiooni haldus.	Iga tarkvara versioon peab jõudma vähemalt ühte, heal juhul rohkematesse testkeskkonda, seega on oluline protsessi osa ja saab olema arvestatud kvaliteediarvutuses.	Jah
Regressioon testimise tulemus, vastuvõtu testimise tulemus.	Iga reliis saab vastava hinnangu ja on oluline need omavahel siduda.	Jah

Tasub ära märkida, et Plutora tarkvara omab veel väga palju erinevaid funktsionaalsuseid, mida siinkohal välja toodud ei ole ning ületavad meie vaadeldava käsitlusala.

3.3 Olemasoleva informatsiooni kõrvutamine

Käesolevas peatükis vaatleme 2020. aastal toodangukeskkonda jõudnud reliiside andmestikku ja teeme sellest mõned järeldused. Lisas 2 on ära toodud tabel, mille põhjal järgnev kokkuvõte on kirjutatud.

Kogu 2020 aasta vältel tarniti toodangukeskkonda 77 tarkvara versiooni, kusjuures kokku sisaldasid need 510 uut funktsionaalsust ja 416 veaparandust. Sellest numbrist näeme juba ära, et veaparanduste hulk ühe uue funktsionaalsuse kohta on 0.81, mida olenevalt tarkvara vanusele ja elutsüklile võib pidada kas heaks või halvaks tulemuseks.

571 *taski* ehk 60% koguarvust on tarnitud *major* reliiside raames. Tähendab, et suur osakaal, 40%, laskub ka veaparandus reliisidele.

Aasta jooksul tehti 9 *major* tüüpi reliisi, kusjuures 6 neist, ehk 66% vajasid samal või järgmisel päeval *hotfixi*. Kogu aasta jooksul leidis aset 22 *hotfixi* vajavat intsidenti.

Reliiside planeerimisel on valdavalt tekkinud null kuni kolm viga, see tähendab, et reliisi halduri, tooteomanike ja arendajate vaheline kommunikatsioon on reliisi planeerimise hetkel olnud piisav, et tagada kokkulepitud reliisi koosseis õigeks ajaks.

Andmestiku põhjal võib väita, et tarkvara versioonide automatiseeritud testide tase on läbi aja stabiilne, mediaan ja aritmeetiline keskmine on mõlemad väärtusega 97%. Hea tulemuse taga on mitu asja – arendajad testivad oma muudatusi põhjalikult enne põhiharudesse integreerimist; testijad haldavad ja uuendavad pidevalt automaatteste; reliisi haldurid peavad silmas automaattestide tulemusi, enne kui reliisi koostama hakatakse; automatiseeritud testide koguarv on nii suur, et näha üheprotsendilist vahet, peaks katki olema märkimisväärne kogus automaatteste.

Tarne tagajärjel tekkinud seisuaja mediaan on 19 minutit, neljal juhul on seisuaeg ületanud 25-te minutit, millest võib eeldada, et tarneprotsessil toimus midagi ebastandardset.

Kõige paremaks *hotfixi* ette ennustavaks võtmeteguriks võib märkida kasutajatoe poolt avatud *taskide* arvu. Mida suurem on avatud *taskide* arv, seda suurem on tõenäosus *hotfixi* vajaduse järele.

3.4 Funktsionaalsed nõuete loetelu

Loodava veebiteenuse funktsionaalsed nõuded on koostatud tuginedes eelnevale analüüsile ning autori tööalastele kogemustele.

Rakendus peab kuvama pealehel peamise vaatena järgmise info:

1. JIRA API kaudu:
    a) suletud staatuses olevate reliiside nimistu;
    b) iga reliisi juurde kuuluvad *taskid*, mis on nähtaval alles pärast reliisi-nimelise akordionlingi avamist;
    c) reliisi sisu erinevus võrreldes planeeritud sisuga;
    d) reliisijärgselt kasutajatoe avatud *taskide* arv;
    e) reliisijärgsete intsidentide arv.
2. Jenkinsi API kaudu:
    f) reliisi kooste tegemise tulemus;
    g) testkeskkonda tarne tegemise tulemus;
    h) proovikeskkonda tarne tegemise tulemus;
    i) toodangukeskkonda tarne tegemise tulemus.
3. Confluence'i API kaudu:
    j) automatiseeritud testide tulemus;
    k) koodianalüüsi tulemus;
    l) manuaaltestide tulemus.

4. Reliisi kvaliteedi hinnang, kusjuures see arvutatakse 10 meetrika põhjal, mis on kirjeldatud ülaltoodud punktides c) kuni l). Arvutuses on 10 teguri osakaalud võrdsed kuid iga teguri maksimaalsed punktisummad defineeritakse koodis tulenevalt viimase aasta reliisi keskmistele väärtustele. Näiteks, kui kasutajatoe poolt avatakse reliisijärgselt keskpäraselt 10 veaparanduse *taski*, siis see võetakse heaks tulemuse määraks, ehk 10%. Kui tegelik avatud *taskide* arv on suurem, siis sellevõrra alandatakse teguri eest arvestatavat protsenti.

   Nagu eelnevast kirjeldatud, on autori soov panna väärtused sõltuma eelnevate tulemuste üldistest keskmistest. Seda seetõttu, et iga infosüsteem ajas areneb ning muutub, ning automaatselt kohanduv meetrika on täpsem. Niisiis ei kirjutata siin välja rangeid numbrilisi definitsioone iga teguri jaoks.

3.5 Mittefunktsionaalsed nõuete loetelu

Loodava veebiteenuse mittefunktsionaalsed nõuded on järgmised:

1. Rakendus peab olema kasutatav Google Chrome ja Mozilla Firefoxi brauseri värskeimas versioonis.

2. Rakenduse andmepäringukiirus ei tohi ületada viite sekundit.

3. Rakenduse on realiseeritud SPA-na (Single Page Application) ehk üheleherakendusena.

4. Rakendust peab olema lihtne ümber konfigureerida teiste projektide JIRA, Jenkinsi, Confluence'i jaoks.

5. Rakendus peab olema iseuuenev, iga minuti järel tehakse andmete uus päring.

4 Prototüübi realisatsioon

Käesolevas peatükis räägitakse töövahendite valikust ja arenduse käigust. Seejärel antakse ülevaade valminud rakendusest, esinenud probleemidest ja võimalikest edasiarenduse ideedest.

4.1 Kasutatud tehnoloogiad

Lähtudes püstitatud probleemist ja ettevõtte nõuetest, mille tarbeks rakenduse prototüüp arendatakse, valiti välja järgmised tehnoloogiad: TypeScript, Npm, AngularJS. Antud peatükis vaadeldakse välja toodud tehnoloogiaid detailsemalt ja põhjendatakse nende valikut.

Angular on populaare ja võimekas JavaScripti raamistik, arendatud Google poolt. Angular võimaldab lihtsasti jagada rakenduse erinevad funktsionaalsused eraldi komponentidesse ning neid hierarhiliselt siduda [10]. Angulari arendatakse veebi viimaste standardite järgi, mis tähendab, et see toetab kõigi brauserite värskemaid versioone [11]. Vue.js ja React.js ees osutus Angular valituks, sest ettevõtte kasutab just seda teeki kõige enam ning autoril on sellega ka rohkem kogemust.

Veebirakenduses kasutatud kuvaelemendid, tabel ja selle vormindus pärinevad disainiraamistikust Angular Material [12]. Alternatiivina võiks kasutada ka PrimeNG, NGBootstrap või NgZorro teeke. Autor on valinud Angular Materiali seetõttu, et see on lõputöö kirjutamise hetkel kõige populaarsem kasutajaliidese arendamiseks sobilik teek, on Angulari enda tiimi poolt arendatud ja disainivalikute poolest minimalistlik [13].

Funktsionaalsuse poolest on teegil juba sisseehitatud tabeli rea laiendamise võimalus, mis autori arvates pakub loogilist, ning selget andmete kuvamise viisi. Kuna tabelis võib olla ridu palju, siis hoiab see ka ruumi kokku.

TypeScript on Microsofti poolt välja töötatud programmeerimiskeel ja põhineb JavaScriptil. TypeScripti kood tõlgitakse kompileerimisel alati JavaScriptiks ümber [14]. TypeScripti eeliseks on JavaScripti ees selle tüübikindlus, tänu millele on koodi lihtsam hallata ja vigu enne kompileerimist avastada. Negatiivsena võib välja tuua kompileerimise suurema ajakulu [15].

Node.js on avatud lähtekoodiga käituskeskkond, mis võimaldab serveripoolseid rakendusi kirjutada JavaScriptis. Selliselt on võimalik arendada nii eesliides kui ka tagaliides samas programmeerimiskeeles [16].

Npm (Node Package Manager) on korraga repositoorium avatud lähtekoodiga Node.js projektide jaoks ja pakihaldussüsteem, mis võimaldab paigaldada, hallata pakette ja nende versioone [17].

4.2 Arenduse protsess

Veebirakenduse arendus algas kasutajaliidese loomisest kasutades Angulari raamistikku. Projekti ja arenduskeskkonna seadistamiseks kasutati Npm paketihaldurit. Angular Materiali abil loodud tabel täideti esialgu sisse kirjutatud andmetega.

Järgmise sammuna katsetati JIRAst ja Jenkinsist päringute tegemist Postmaniga. Vastused salvestati JSONina Typsecript faili, muutujatesse, mis seejärel tabeli andmeallikatena ühendati ning sel viisil sai tabelis juba kuvada kuvada infot õigete struktuuriga objektidelt ja nende väljadelt.

Kui rakendus oskas õige, kuid koodi sisse kirjutatud, andmemudeli järgi tabelit kuvada, tuli alustada väliste süsteemide ühendamisega, et andmeid päritaks konkreetsete teenuste käest. Selleks kasutab autor HttpClient [18] klassi ja RxJS-i [19], millest esimest kasutatakse HTTP päringute tegemiseks ja viimast päringu vastuse haldamiseks. Näiteks lehe laadides alustatakse RxJS forkJoin [20] käsuga HttpClient päringud erinevate teenuste pihta, tagastades päringute lõppemisel korraga ühe vastuse objekti, milles on kõikide päringute tulemused. Selliselt saab olla kindel, et kõik päringud on korraga lõppenud, ning võib andmed saata edasisele töötlemisele (vt Joonis 4).

```
ngOnInit() {
  const versions = this.http.get( url: `/external/api/versions`);
  const uat = this.http.get( url: `/external/api/builds?buildName=UAT_environment`);
  const prod = this.http.get( url: `/external/api/builds?buildName=production_deployment`);
  const release = this.http.get( url: `/external/api/builds?buildName=release_management`);
  const preProd = this.http.get( url: `/external/api/builds?buildName=preprod_deployment`);
  const wiki = this.http.get( url: `/external/api/wiki`);

  forkJoin( sources: [versions, uat, prod, release, preProd, wiki]).subscribe( next: (results: any) => {
    const versions = results[0];
    const uat = results[1];
    const prod = results[2];
    const release = results[3];
    const preProd = results[4];
    const wiki = results[5];

    this.dataSource.data = this.prepareData(versions, uat['builds'], prod['builds'], release['builds'], preProd['builds'], wiki);
  });
}
```

Joonis 4. HttpClient ja RxJS koodinäide.

Antud rakendus kasutab kasutaja autentimiseks JIRA, Jenkinsiga ja Confluence'ga suhtlemisel Basic-tüüpi meetodit, mis HTTP päringu tegemisel lisab päisesse välja, mis sisaldab kooloniga eraldatud ja Base64 alusel enkodeeritud kasutajanime ja parooli [21]. Alternatiividena oleks võinud kasutada ka OAuth või JWT meetodeid, mille põhimõte seisneb ligipääsulubade kasutamisel [22], [23]. Viimased kaks oleksid turvalisemad, kuna ei eelda kasutaja info edastamist protokolli sees [24]. Küll aga on Basic-tüüpi autentimist lihtsam kasutada ja Jenkinsi ühendust pakkuv teek teisi viise töö kirjutamise hetkel ei toeta.

Brauserid rakendavad vaikimisi domeenisisese ressursikasutuse printsiipi (same-origin policy), mis piirab eesrakendusest tehtud päringud erinevas domeenis oleva tagarakendusega suhtlemisel. Selliselt ei saa võimalik pahatahtlik kood teha kasutaja brauseri nimel päringuid teistesse tagarakendustesse [25].

Selleks, et eelnevalt kirjeldatud piirangust mööda saada, CORS (*Cross-origin resource sharing*) veateavitust saamata, otsustas töö autor eesrakenduse päringu suunata läbi *proxy* tagarakenduse serverisse [26]. Veebiserveri loomiseks kasutati Express veebiraamistikku, mis võimaldab väikese vaevaga seadistada serveri [27].

Niisiis on eesrakendus käimas Angulari vaikimisi seatud pordil 4200, Expressi server käib pordil 3000. Rakenduse käitamisel tehakse näiteks päring aadressile /jira/issues, mis suunatakse ümber localhost:3000/jira/issues peale. Tagarakendus saab päringu kätte ja teeb omakorda uue päringu, konkreetse näite puhul Jira poole.

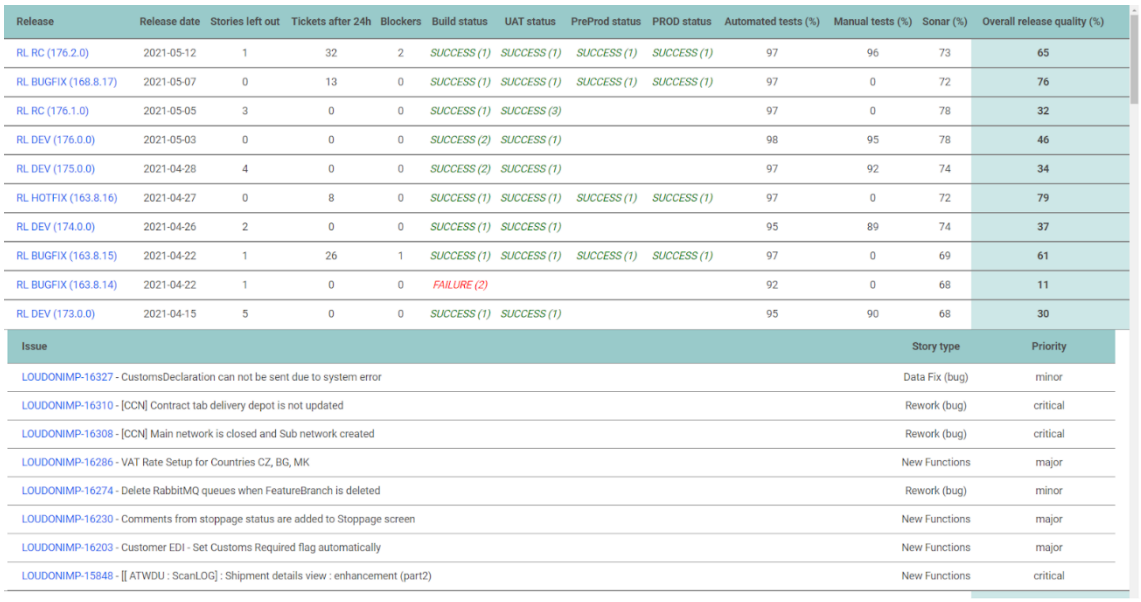
JIRA API-st andmete pärimise dokumentatsioon on Atlassiani poolt hästi dokumenteeritud [28], ent internetis leidub mitmeid abistavaid programme, mis lisavad APIle abstraktsioonikihi, kus päringute tegemiseks ei pea ise Jira APIga ühendust looma ega *endpointe* implementeerima. Tuleb vaid koodis vastavaid funktsioone välja kutsuda. Autori poolt osutus valituks jira.js-nimeline teek [29]. Selle toel on autentimine ja andmete pärimine lihtsustatud. JIRAst on tabelisse integreeritud kõik projekti versioonid, kuupäevaliselt kahanevas järjekorras. Versiooni nimi on hüpiklingina, mis viib peale klikates antud Jira lehele. Lisaks tehakse versiooni reale klikkides lisapäring, mis leiab kõik versiooniga seotud *taskid*. JIRAst leitakse ka iga versiooni planeeritud *taskide* arv, tarnimise järel kasutajatoe poolt avatud *taskide* arv ja intsidendi käivitanud *taskide* arv, kõik kolm otsingut tehakse JIRA välja „*label*“ järgi, mida antud projekti puhul kasutatakse.

Jenkinsi API-ga suhtlemiseks osutus valituks teek node-jenkins [30]. Teegi eelisteks teiste sarnaste ees on see, et see on lihtsasti kasutatav, hea dokumentatsiooni ja laia kasutajaskonnaga. Jenkinsist päritakse reliisi kooste ja kolme erineva keskkonna tarne tulemused. Juhul, kui mõne versiooni puhul on *pipeline* käivitatud mitmel korral, arvestatakse see üldises kvaliteedi arvutuses maha, vastav käivituste koguarv on näha tabelis sulgudes.

Confluence’i wiki API-ga suhtluseks valiti välja teek confluence-api [31]. Kuna projektis käivitatavad automaattestide ja koodianalüüsi testide tulemused laetakse üles ka wikisse, otsustas autor, et seda tüüpi infot on kõige lihtsam pärida Confluence’st. Manuaalset regressiooni, nagu varasemalt mainitud, viiakse läbi vaid *major* reliiside puhul ja veaparanduse reliiside puhul see tegur kvaliteediarvestuses nullitakse.

4.3 Tulemus

Arendustöö tulemusena on valminud prototüüp (vt. joonis 5), mis vastab välja toodud funktsionaalsetele nõuetele. Mittefunktsionaalsetest nõuetest jäi üks tahaplaanile, nimelt ei ole praegu kuigi lihtne seadistada sama prototüüpi teiste projektide jaoks, sest vastav konfiguratsioon on koodi sisse kirjutatud. Edasiarenduste raames on võimalik seda parandada.



Release	Release date	Stories left out	Tickets after 24h	Blockers	Build status	UAT status	PreProd status	PROD status	Automated tests (%)	Manual tests (%)	Sonar (%)	Overall release quality (%)
RL RC (176.2.0)	2021-05-12	1	32	2	SUCCESS (1)	SUCCESS (1)	SUCCESS (1)	SUCCESS (1)	97	96	73	65
RL BUGFIX (168.8.17)	2021-05-07	0	13	0	SUCCESS (1)	SUCCESS (1)	SUCCESS (1)	SUCCESS (1)	97	0	72	76
RL RC (176.1.0)	2021-05-05	3	0	0	SUCCESS (1)	SUCCESS (3)			97	0	78	32
RL DEV (176.0.0)	2021-05-03	0	0	0	SUCCESS (2)	SUCCESS (1)			98	95	78	46
RL DEV (175.0.0)	2021-04-28	4	0	0	SUCCESS (2)	SUCCESS (1)			97	92	74	34
RL HOTFIX (163.8.16)	2021-04-27	0	8	0	SUCCESS (1)	SUCCESS (1)	SUCCESS (1)	SUCCESS (1)	97	0	72	79
RL DEV (174.0.0)	2021-04-26	2	0	0	SUCCESS (1)	SUCCESS (1)			95	89	74	37
RL BUGFIX (163.8.15)	2021-04-22	1	26	1	SUCCESS (1)	SUCCESS (1)	SUCCESS (1)	SUCCESS (1)	97	0	69	61
RL BUGFIX (163.8.14)	2021-04-22	1	0	0	FAILURE (2)				92	0	68	11
RL DEV (173.0.0)	2021-04-15	5	0	0	SUCCESS (1)	SUCCESS (1)			95	90	68	30

Issue	Story type	Priority
LOUDONIMP-16327 - CustomsDeclaration can not be sent due to system error	Data Fix (bug)	minor
LOUDONIMP-16310 - [CCN] Contract tab delivery depot is not updated	Rework (bug)	critical
LOUDONIMP-16308 - [CCN] Main network is closed and Sub network created	Rework (bug)	critical
LOUDONIMP-16286 - VAT Rate Setup for Countries CZ, BG, MK	New Functions	major
LOUDONIMP-16274 - Delete RabbitMQ queues when FeatureBranch is deleted	Rework (bug)	minor
LOUDONIMP-16230 - Comments from stoppage status are added to Stoppage screen	New Functions	major
LOUDONIMP-16203 - Customer EDI - Set Customs Required flag automatically	New Functions	major
LOUDONIMP-15848 - [[ATWDU : ScanLOG] : Shipment details view : enhancement (part2)	New Functions	critical

Joonis 5. Kuvatõmmis valminud prototüübist.

Positiivse külje pealt võib välja tuua liidese kasutajamugavuse. Kuigi info pärimist on palju, tehakse need esimesel laadimisel ligi kolme sekundi jooksul ära ning edasine sirvimine on muretu. Reliisi identifikaator ja reliisis sisalduvad *taskid* on tehtud lingitavaks, et lubada kiiret ligipääsu JIRAsse, kui selleks peaks vajadus olema.

Negatiivse külje pealt tuleks ära mainida prototüübi kujundus, mis esmasel kasutamisel võib kasutajas segadust tekitada. Üldise kvaliteedi arvutus toimub automaatselt taustal ning kasutajale ei näidata arvutuskäiku, tulevikus võiks mõelda värvide lisamise peale, nt positiivse tulemuse saanud reliisid ja reliisietapid värvitaks roheliseks ja teised vastavalt kollaseks, punaseks.

4.4 Edasiarenduse ideed

Valminud veebirakendusel on potentsiaali, et automatiseerida veelgi reliisi halduse protsessi ja prototüüpi kasutatakse edaspidi põhjana, et lisada rohkem funktsionaalsust.

Automatiseeritud raportite koostamine

Bakalaureuse töö koostamise käigus tutvus autor lähemalt erinevate APIde võimalustega ning leidis, et Confluence'i APIt on edukalt võimalik kasutada ka uute lehtede

koostamiseks. Tulevikus võiks reliisi kvaliteedi hinnangud Confluence'i üle kanda, et säiliks ajalugu.

Automatiseeritud emailide saatmine

Reliiside kogu elutsükli vältel käib pidev kommunikatsioon arendajate, tootejuhtide ja reliisihalduse tiimi vahel. Seesugused kirjavahetused peaks olema lihtne automatiseerida.

Linkide lisamine infokildude juurde

Praeguses prototüübis on reliisi versioonid ja *taskid* lingitud, et tagada kiire juurdepääs info allikani. Lisaks võiks kõigile Jenkisist pärinevatele tulemustele lisada hüperlingid. See soodustaks veebirakenduse potentsiaali olla kolleegide eelistatuim veebileht kiire info otsimiseks.

Planeeritud ja protsessis olevate reliiside lisamine

Praeguses formaadis on keskendutud ainult reliisidele, mis on juba oma elutsükli lõpetanud ja jõudnud lõppkasutajani. Tegelikult võiks sama veebirakendust kasutada ka reliiside elutsükli ajal ja planeerimisel.

Reliisikalendri vaate lisamine

Reliisikalender hallatakse praegu Confluence'is ja sündmused sisestatakse käsitsi. Rakenduse abil võiks kas automatiseerida sündmuste lisamise kalendrisse või luua kalender lisa vaatena.

Teiste projektide integreerimine

Praegune rakendus on seadistatud ainult ühe projekti jaoks. Samas on teada, et Jira, Confluence ja Jenkins on ettevõtteülesed tööriistad, mida kõik kasutavad. Rakenduse kood võiks tulevikus olla lihtsasti seadistatav ka teiste projektide jaoks.

5 Kokkuvõte

Bakalaureuse töös käsitletava ettevõtte reliisi halduse protsess sisaldab mitmeid rutiinseid samme. Suurenenud reliiside arvu tõttu on tõstatanud küsimus reliiside kvaliteedi kohta. Käesoleva töö eesmärgiks oli defineerida reliisi kvaliteedi meetrikad ja luua prototüüp rakendusele, mis koondaks vastava info.

Leidmaks parimaid meetrikaid uuris töö autor enamlevinuid praktikaid, üht valdkonna populaarset töövahendit ja ettevõtte eelmise aasta reliiside statistikat. Analüüsi tulemusena koostati kümnest meetrikast koosnev nimistu, mis peaks antud projekti reliisi kvaliteedi hindamisel esindatud olema ning mille põhjal võib arvutada ülevaatliku hinnangu kogu protsessi edukusele. Lisaks leidis autor analüüsi käigus kaks meetrikat, mille kohta praegu puudub ülevaade ning mida tasuks projekti arenedes hindamissüsteemi kaasata – lõppkasutajate rahulolu ja teenuse sooritusvõime muutused.

Prototüübi loomisel võeti sisendiks analüüsi etapis valminud funktsionaalsed ja mittefunktsionaalsed nõuded. Veebirakenduse realiseerimiseks kasutati TypeScripti ja AngularJS raamistikku. Töö käigus valmis esialgne rakendus, mis täidab valdavas osas ettekirjutatud nõuded. Veebirakendus kuvab pealehel kõigi tarkvara versioonide kohta käiva põhiinfo ja reliisi elutsükli jooksul läbitud etappide lisainfo. Rakendus arvutab reliisile üldise kvaliteedihinnangu.

Prototüüp ja edasiarenduse ideed on esitletud ettevõttele ning tagasiside on olnud positiivne. Tulevikus on oodata prototüübi majutamist, edasiarendust ning töösse rakendamist. Uus tööriist on uuenduslik, lisaväärtust loov, selle edaspidisel arendusel ja töös hoidmisel võib olla hea mõju projekti tiimi ajakasutusele ja reliiside kvaliteedile tänu vahetu tagasiside saamisele.

Kasutatud kirjandus

[1] „Kaizen Method and Philosophy,“ [Võrgumaterjal]. Available: https://www.spica.com/blog/kaizen-method. [Kasutatud 23 Märts 2021].

[2] Organisatsiooni siseveeb, 2021. [Võrgumaterjal].

[3] J. S. Ken Schwaber, „The Scrum Guide,“ [Võrgumaterjal]. Available: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf. [Kasutatud 15 Märts 2021].

[4] Kanbanize, „Kanban,“ [Võrgumaterjal]. Available: https://kanbanize.com/kanban-resources/getting-started/what-is-kanban. [Kasutatud 15 Märts 2021].

[5] V. Driessen, „A successful Git branching model,“ [Võrgumaterjal]. Available: https://nvie.com/posts/a-successful-git-branching-model/. [Kasutatud 25 Märts 2021].

[6] J. Tepandi, „Tarkvara protsessid ja kvaliteet,“ 18 Aprill 2020. [Võrgumaterjal]. Available: https://tepandi.ee/tks-loeng.pdf.

[7] S. Rance, „ITIL Service Transition,“ London, The Stationery Office, 2011, pp. 114-115.

[8] „Plutora Release Management,“ Plutora, [Võrgumaterjal]. Available: https://www.plutora.com/platform/release-management. [Kasutatud 02 Aprill 2021].

[9] Plutora, „Deliver High Quality Software - Faster,“ 2020.

[10] „Angular,“ [Võrgumaterjal]. Available: https://angular.io/. [Kasutatud 05 Mai 2021].

[11] „Browser support,“ Angular, [Võrgumaterjal]. Available: https://angular.io/guide/browser-support. [Kasutatud 06 Mai 2021].

[12] „Angular Material,“ [Võrgumaterjal]. Available: https://material.angular.io/. [Kasutatud 05 Mai 2021].

[13] „NPM trends,“ [Võrgumaterjal]. Available: https://www.npmtrends.com/@angular/material-vs-ng-zorro-antd-vs-primeng-vs-ng2-bootstrap. [Kasutatud 05 Mai 2021].

[14] „TypeScript documentation,“ Microsoft, [Võrgumaterjal]. Available: https://www.typescriptlang.org/docs/. [Kasutatud 05 Mai 2021].

[15] „The Benefits of Migrating from JavaScript to TypeScript,“ AppDynamics, [Võrgumaterjal]. Available: https://www.appdynamics.com/blog/engineering/the-benefits-of-migrating-from-javascript-to-typescript/. [Kasutatud 05 Mai 2021].

[16] „NodeJS,“ [Võrgumaterjal]. Available: https://nodejs.org/en/about/. [Kasutatud 05 Mai 2021].

[17] „NpmJS,“ [Võrgumaterjal]. Available: https://www.npmjs.com/. [Kasutatud 05 Mai 2021].

[18] „HttpClient,“ Google, [Võrgumaterjal]. Available: https://angular.io/api/common/http/HttpClient. [Kasutatud 02 Mai 2021].

[19] „The RxJS library,“ Google, [Võrgumaterjal]. Available: https://angular.io/guide/rx-library. [Kasutatud 07 Aprill 2021].

[20] „Learn RxJS: ForkJoin,“ [Võrgumaterjal]. Available: https://www.learnrxjs.io/learn-rxjs/operators/combination/forkjoin. [Kasutatud 07 Mai 2021].

[21] J. Reschke, „ The 'Basic' HTTP Authentication Scheme,“ [Võrgumaterjal]. Available: https://tools.ietf.org/html/rfc7617#section-2.1. [Kasutatud 05 Mai 2021].

[22] „OAuth for REST APIs,“ Atlassian, [Võrgumaterjal]. Available: https://developer.atlassian.com/cloud/jira/platform/jira-rest-api-oauth-authentication/. [Kasutatud 05 Mai 2021].

[23] „Introduction to JSON Web Tokens,“ [Võrgumaterjal]. Available: https://jwt.io/introduction. [Kasutatud 05 Mai 2021].

[24] „The Difference Between HTTP Auth, API Keys, and OAuth,“ [Võrgumaterjal]. Available: https://nordicapis.com/the-difference-between-http-auth-api-keys-and-oauth/. [Kasutatud 05 Mai 2021].

[25] „Cross Origin Resource Sharing,“ [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS. [Kasutatud 05 Mai 2021].

[26] „Fixing CORS errors with Angular CLI proxy,“ GitConnected, [Võrgumaterjal]. Available: https://levelup.gitconnected.com/fixing-cors-errors-with-angular-cli-proxy-e5e0ef143f85. [Kasutatud 06 Mai 2021].

[27] „ExpressJS,“ [Võrgumaterjal]. Available: http://expressjs.com/. [Kasutatud 05 Mai 2021].

[28] „JIRA API,“ [Võrgumaterjal]. Available: https://developer.atlassian.com/server/jira/platform/rest-apis/. [Kasutatud 05 Mai 2021].

[29] „Jira.js,“ GItHub, [Võrgumaterjal]. Available: https://github.com/MrRefactoring/jira.js. [Kasutatud 05 Mai 2021].

[30] „Jenkins,“ GItHub, [Võrgumaterjal]. Available: https://github.com/silas/node-jenkins. [Kasutatud 06 Mai 2021].

[31] „Confluence API,“ GitHub, [Võrgumaterjal]. Available: https://www.npmjs.com/package/confluence-api. [Kasutatud 05 Mai 2021].

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

Mina, Anne Schults

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „ Reliisi kvaliteedi hindamise meetrika analüüs ja prototüübi loomise ettepanekud ettevõtte näitel“, mille juhendaja on Inna Švartsman
   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.

18.05.2021

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.

Lisa 2 – Reliiside statistika 2020 põhjal

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	142.3.0	major*	1/2/2020	1/15/2020	13		1	0	64	48	16	0	8	2	6	97%	22	22
2	142.3.2	hotfix	1/17/2020	1/17/2020	0		0	0	1	0	1	0		1	0	97%	17	8
3	142.3.3	bugfix	1/21/2020	1/22/2020	1		0	0	11	1	10	0	6	1	3	98%	18	10
4	142.3.4	bugfix	1/27/2020	1/28/2020	1		0	2	14	1	13	0	4	5	4	97%	18	11
5	142.3.5	bugfix	2/5/2020	2/6/2020	1		0	0	12	1	10	1	4	3	3	97%	19	12
6	142.3.6	bugfix	2/11/2020	2/11/2020	0		0	1	6	0	6	0	2	1	3	97%	18	18
7	142.3.8	hotfix	2/13/2020	2/13/2020	0		0	0	2	0	2	0	1	0	1	97%	18	2
8	142.3.11	bugfix	2/19/2020	2/20/2020	1		0	0	1	0	1	0	0	0	1	98%	18	4
9	143.4.0	major*	2/12/2020	2/26/2020	14		1	1	55	43	10	2	3	2	5	99%	25	36
10	143.5.2	hotfix	2/28/2020	2/28/2020	0		0	0	2	0	2	0	1	1	0	97%	18	10
11	143.5.3	bugfix	3/4/2020	3/4/2020	0	Roll-out	0	1	3	3	0	0	0	0	0	98%	18	15
12	143.5.4	hotfix	3/4/2020	3/4/2020	0	Roll-out	0	0	1	0	1	0	0	0	1	97%	17	8
13	143.5.5	bugfix	3/6/2020	3/6/2020	0	Roll-out	0	2	11	1	10	0	3	2	5	97%	18	7
14	143.5.6	bugfix	3/9/2020	3/10/2020	1		0	0	9	0	8	1	2	3	3	97%	19	13
15	143.5.7	bugfix	3/12/2020	3/12/2020	0		0	0	6	0	6	0	0	2	4	97%	20	20
16	145.1.0	major*	3/11/2020	3/25/2020	14		2	0	49	40	9	0	0	4	5	93%	22	19
17	145.1.1	hotfix	3/26/2020	3/26/2020	0		0	0	1	0	1	0	1	0	0	97%	21	22
18	145.1.3	hotfix	4/7/2020	4/7/2020	0		0	0	1	0	1	0	0	0	1	97%	22	10
19	145.1.5	bugfix	4/14/2020	4/15/2020	1		0	0	2	1	1	0	0	1	0	97%	36	15
20	145.1.6	hotfix	4/17/2020	4/17/2020	0		0	0	2	0	2	0	0	1	1	95%	20	26
21	145.1.7	hotfix	4/20/2020	4/20/2020	0		0	0	1	0	1	0	1	0	0	97%	21	5
22	146.2.0	major*	4/13/2020	4/27/2020	14		2	1	67	62	5	0	4	1	0	98%	25	42
23	146.2.1	hotfix	4/28/2020	4/28/2020	0		0	0	2	0	2	0	0	1	1	97%	18	19
24	146.2.2	hotfix	4/29/2020	4/29/2020	0		0	0	2	0	2	0	0	2	0	97%	18	12
25	146.2.3	hotfix	5/5/2020	5/5/2020	0		0	0	1	0	1	0	0	0	1	97%	19	23
26	146.2.4	hotfix	5/7/2020	5/7/2020	0		0	0	1	0	1	0	0	0	1	97%	20	12
27	146.2.5	bugfix	5/11/2020	5/12/2020	1		1	1	11	2	9	0	1	5	3	96%	20	19
28	146.2.6	hotfix	5/13/2020	5/13/2020	0		0	0	1	0	1	0	0	0	1	96%	20	14
29	146.2.7	bugfix	5/19/2020	5/20/2020	1		0	0	4	1	3	0	0	1	2	96%	19	17
30	146.2.9	bugfix	5/26/2020	5/27/2020	1		0	1	2	0	2	0	0	1	1	96%	19	11
31	146.2.10	bugfix	6/4/2020	6/4/2020	0		0	1	4	1	3	0	1	0	2	96%	18	14
32	150.3.0	major*	6/1/2020	6/11/2020	10		3	3	89	75	14	0	9	0	5	97%	23	30
33	150.4.1	bugfix	6/16/2020	6/17/2020	1		0	3	12	2	10	0	1	2	7	97%	18	17
34	150.4.2	bugfix	6/25/2020	6/26/2020	1		0	2	8	2	6	0	0	3	3	98%	19	21
35	150.4.3	hotfix	6/29/2020	6/29/2020	0		0	0	1	0	1	0	0	1	0	97%	18	15
36	150.4.4	bugfix	6/30/2020	6/30/2020	0		0	0	5	0	5	0	1	2	2	97%	20	14
37	150.4.5	bugfix	7/2/2020	7/2/2020	0		0	0	8	0	8	0	2	2	4	97%	23	9
38	150.4.6	bugfix	7/3/2020	7/3/2020	0		0	0	2	1	1	0	0	0	1	97%	19	12
39	150.4.7	bugfix	7/4/2020	7/4/2020	0		0	1	2	1	1	0	0	1	0	97%	20	7
40	150.4.9	bugfix	7/8/2020	7/8/2020	0		1	0	7	0	7	0	1	2	4	96%	18	16
41	150.4.10	hotfix	7/9/2020	7/9/2020	0		0	0	1	0	1	0	0	0	1	96%	18	13
42	150.4.11	bugfix	7/10/2020	7/10/2020	0		0	2	5	0	5	0	2	0	3	97%	19	12
43	150.4.12	bugfix	7/14/2020	7/14/2020	0		0	2	5	1	4	0	0	1	3	97%	20	22
44	150.4.13	bugfix	7/21/2020	7/22/2020	1		0	1	12	1	11	0	1	4	6	97%	17	24
45	150.4.14	hotfix	7/23/2020	7/23/2020	0		0	0	1	0	1	0	0	0	1	97%	18	14
46	151.1.0	major*	8/10/2020	8/24/2020	14		3	4	97	82	14	1	8	5	1	97%	25	36
47	151.1.1	hotfix	8/26/2020	8/26/2020	0		0	0	1	0	1	0	1	0	0	97%	20	27
48	151.1.2	hotfix	8/27/2020	8/27/2020	0		0	0	1	0	1	0	0	0	1	97%	21	18
49	151.1.3	bugfix	9/1/2020	9/2/2020	1	Roll-out	0	2	12	2	10	0	2	6	2	97%	20	16
50	151.1.4	bugfix	9/3/2020	9/3/2020	0	Roll-out	0	1	7	0	6	1	1	1	4	97%	35	19
51	151.1.5	bugfix	9/8/2020	9/9/2020	1		0	0	5	0	5	0	2	2	1	97%	18	14
52	151.1.6	bugfix	9/15/2020	9/16/2020	1		0	0	6	1	4	1	1	2	1	97%	22	15
53	153.0.0	major*	9/9/2020	9/23/2020	14		2	3	44	36	7	1	6	0	1	97%	18	34
54	153.3.1	bugfix	9/24/2020	9/24/2020	0		0	1	9	3	6	0	2	1	3	97%	20	24
55	153.3.2	bugfix	9/30/2020	10/1/2020	1	Roll-out	0	0	14	0	14	0	4	5	5	97%	19	16
56	153.3.3	bugfix	10/5/2020	10/5/2020	0	Roll-out	0	2	13	2	11	0	1	4	6	97%	20	17
57	153.3.6	bugfix	10/7/2020	10/7/2020	0	Roll-out	0	0	9	1	8	0	0	2	6	96%	24	18
58	153.3.7	bugfix	10/8/2020	10/8/2020	0	Roll-out	0	0	5	2	3	0	1	1	1	96%	20	29
59	153.3.8	bugfix	10/12/2020	10/12/2020	0	Roll-out	0	2	16	4	12	0	1	5	6	95%	20	24
60	153.3.9	bugfix	10/14/2020	10/14/2020	0	Roll-out	0	2	12	3	9	0	1	4	4	95%	19	15
61	153.3.11	bugfix	10/20/2020	10/20/2020	0	Roll-out	0	1	10	9	1	0	0	0	1	96%	20	42
62	153.3.12	hotfix	10/20/2020	10/20/2020	0	Roll-out	0	0	2	0	2	0	0	0	2	96%	20	26
63	153.3.13	bugfix	10/22/2020	10/22/2020	0	Roll-out	0	1	4	0	4	0	0	1	3	96%	19	36
64	153.3.14	hotfix	10/22/2020	10/22/2020	0	Roll-out	0	0	1	0	1	0	0	0	1	96%	18	21
65	154.0.0	major*	10/12/2020	10/28/2020	16		2	3	65	35	25	5	15	4	6	97%	23	48
66	154.0.3	bugfix	10/29/2020	10/29/2020	0		0	1	13	1	12	0	3	5	4	97%	19	23
67	154.0.4	bugfix	11/4/2020	11/4/2020	0		0	2	15	3	12	0	1	4	7	98%	19	26
68	154.0.5	hotfix	11/5/2020	11/5/2020	0		0	0	1	0	1	0	0	0	1	98%	20	14
69	154.0.6	bugfix	11/9/2020	11/10/2020	1		0	2	8	0	7	1	1	3	3	97%	19	13
70	154.0.7	bugfix	11/11/2020	11/11/2020	0	Roll-out	0	2	8	2	6	0	2	1	3	97%	19	26
71	154.0.11	hotfix	11/23/2020	11/23/2020	0	Roll-out	0	0	1	0	1	0	0	1	0	96%	18	9
72	154.0.12	bugfix	11/25/2020	11/25/2020	0	Roll-out	0	1	7	2	5	0	1	2	2	97%	17	14
73	154.0.15	bugfix	12/2/2020	12/2/2020	0		0	1	2	0	2	0	0	1	1	97%	18	13
74	155.6.0	major*	11/30/2020	12/10/2020	10		3	5	41	34	7	0	5	1	1	97%	24	34
75	155.6.1	hotfix	12/10/2020	12/10/2020	0		0	0	1	0	1	0	0	1	0	97%	36	25
76	155.6.2	hotfix	12/11/2020	12/11/2020	0		0	0	1	0	1	0	0	1	0	98%	18	14
77	155.6.5	bugfix	12/16/2020	12/17/2020	1		0	0	2	0	2	0	1	0	1	98%	18	11

Joonis 6. Reliiside statistika 2020 näitel.

A - Reliisi nimi
B - Reliisi tüüp
C - Kooste päev
D - Tarne päev
E - Testimisperioodi pikkus (päevades)
F - Sündmused
G - Sõltuvustega reliisid
H - Väljajäetud taskide arv
I - Taskide arv
J - Uute funktsionaalsuste arv
K - Veaparanduste arv
L - Tehnilised muudatused
M - Veapranduste esimene tüüp: regressioon
N - Veaparanduste teine tüüp: Uute funktsionaalsuste ebapiisav testimine
O - Veaparanduste kolmas tüüp: esialgsete funktsioonide muudatused pärast uute funktsionaalsuste tutvustamist
P - UI testide tulemus (%)
Q - Seisuaeg (min)
R - Kasutajatoele raporteeritud vigade arv 24h jooksul
major* - sedatüüpi reliisi juurde on arvestatud nii algne reliis kui ka stabiliseerimise ajal tehtud muudatused

Joonis 7. Eelneva tabeli juurde kuuluv veergude päiste legend.