Tartu Ülikool

Arvutiteaduse instituut

Infotehnoloogia mitteinformaatikutele õppekava

Katre Siilivask

Scrum raamistik ja selle rakendamine Playtech Estonia OÜ osakonna näitel

Magistritöö (15 EAP)

Juhendajad: Anne Villems, MSc

Vambola Leping, MSc

Ivo Tamm

Tartu 2021

Scrum raamistik ja selle rakendamine Playtech Estonia OÜ osakonna näitel

Lühikokkuvõte:

Infotehnoloogia sektor on pidevas muutumises ning asutuse töövõtted ning kasutatavad raamistikud peaksid olema kooskõlas saavutatavate eesmärkidega. Mida selgemad on töötajate jaoks need raamistikud, seda lihtsam on ka töötajatel oma eesmärke täita ning jõuda ühise eesmärgini - tarnida kliendile kvaliteetne ning kliendi vajadustele vastav toode.

Magistritöö eesmärk on anda ülevaade agiilse arendusmetoodika Scrum raamistiku teoreetilisest taustast ning teha ülevaade, kuidas antud raamistikku on implementeeritud rahvusvahelises tarkvara ja IT tooteid arendavas ettevõttes, kus iga osakonna töökorraldus on erinev.

Scrum on agiilse arenduse raamistik, mis aitab meeskondades töid ja protsesse korraldada ja juhtida ning annab võimaluse igale meeskonnale, kes Scrum'i kasutab, muuta kasutatav raamistik omanäoliseks ja sobivaks vastavalt vajadusele.

Töö tulemustest selgus, et Scrum'i raamistiku kasutamine ettevõttes on suures ulatuses teooriaga kooskõlas ning see on muutnud tööprotsessid (näiteks sprindi planeerimine) palju efektiivsemaks ja täpsemaks. Leidus ka üksikuid kõrvalekaldeid teooriast (näiteks soovitatud meeskonna suurus), kuid samas antud olukord ei pärssinud osakonna tööprotsesse.

Võtmesõnad:

Scrum, toote kuhi, grooming, sprindi kuhi, sprindi planeerimine, *story points*, varjatud ajahinnangute andmine, sprindi ülevaade, retrospektiiv, Scrum Master, toote omanik, arendusmeeskond

CERCS: P175 Informaatika, süsteemiteooria

Scrum Framework and its Implementation on the Example of Playtech Estonia OÜ Department

Abstract:

The IT sector is constantly evolving and the businesses working methods and frameworks should be in line with the objectives to be achieved. The clearer these frameworks are for employees, the easier it will be for them to meet their goals and also reach a common goal - to deliver a high-quality product that meets the customer's needs.

The aim of the master's thesis is to give an overview of the theoretical background of the Agile development methodology's Scrum framework and to make an overview of how this framework has been implemented in an international software and IT product development company, where organizing the work in each department is different.

Scrum is an agile development framework that helps teams organize and manage work and processes, and allows each team that uses Scrum to make the framework unique and suitable according to the needs of the team.

The results showed that the use of the Scrum framework in the company is largely in line with the theory and using it has made work processes (such as sprint planning) much more efficient and accurate. There were also a few variations from the theory (for example, the recommended team size), but this situation did not inhibit the department's work processes.

Keywords:

Scrum, product backlog, grooming, sprint backlog, sprint planning, story points, planning poker, sprint review, retrospective, Scrum Master, product owner, development team

CERCS: P175 Informatics, systems theory

Sisukord

Sissejuhatus

Infotehnoloogia sektor on pidevas muutumises ning asutuse töövõtted ning kasutatavad raamistikud peaksid olema kooskõlas saavutatavate eesmärkidega. Mida selgemad on töötajate jaoks need raamistikud, seda lihtsam on ka töötajatel oma eesmärke täita ning jõuda ühise eesmärgini - tarnida kliendile kvaliteetne ning kliendi vajadustele vastav toode.

Ilma igasuguse tarkvarata ei kujuta me keegi enam oma igapäevast elu ette. Ka klientide soovid tarkvara osas on läinud järjest spetsiifilisemaks ning valmis tarkvaratooteid soovitakse saada järjest kiiremini, et omada konkurentsieelist üha muutuval turul. See on loonud vajaduse ka järjest kiiremaks ja paindlikumaks arenduseks ning eemale on liigutud traditsioonilise ja struktureeritud nn. koskmudeli arenduse juurest ja üle on võetud agiilseid metoodikaid ja raamistikke.

Digital.ai viib igal aastal läbi agiilsete meetodite kasutamise uuringuid ning 2020. aastal avaldatud uuringust [25] selgub (uuringu periood august – detsember 2019, vastajaid 40 000), et 95% vastanutest praktiseerivad igapäevaselt agiilseid metoodikaid. Scrum'i kasutamise populaarsus on tõusnud 2021.aastal tehtud uuringule toetudes 66%-ni (uuringu periood veebruar-aprill 2021, vastajaid 4182) [26], võrdluseks 40%, mis ilmnes esimesest taolisest uuringust. Lisaks on uuringus [25] välja toodud ka põhjused, miks on kasutusele võetud agiilsed meetodid. Kolm populaarsemat vastust olid kiirendada tarkvara tarnimist kliendile, suurendada võimet hallata muutuvaid prioriteete ning suurendada tootlikkust.

Üheks suureks tõukeks agiilsuse poole liikumiseks võib lugeda ka 2019. aastal alanud koroonapandeemiat, mis pani ettevõtteid mõtlema, kuidas muutustele reageerida oskuslikult ja efektiivselt, kui personal hakkas tegema kaugtööd oma kodust. Sellisel juhul on raskem kasutada traditsioonilisi meetodeid, kui meeskonnad ei ole enam koos ja lähestikku, vaid on hajutatud ja üksteisest kaugel. 2021.aasta uuringu [26] tulemused toetavad samuti uue tööviisina hübriid- ja kaugtööd. Uuringust lähtub, et vaid 3% vastanutest plaanib tagasi kontoritesse naasta, samas kui 25% vastanutest planeerivad jäädavalt kaugtööle jääda.

Scrum on agiilse arenduse raamistik, mis aitab meeskondades töid ja protsesse korraldada ja juhtida ning annab võimaluse igale meeskonnale, kes Scrum'i kasutab, muuta kasutatav raamistik omanäoliseks ja sobivaks vastavalt vajadusele. Scrum määrab ära raamistiku rollid, vastutusalad ja nende tööülesanded ning lisaks protsessid ja tegevused, mida peaks regulaarselt läbi viima, et meeskonna töö oleks efektiivne ja vastavalt vajadustele arenev.

Autori lõputöö on aktuaalne seetõttu, et kõik teoreetilise taustaga raamistikud ei pruugi olla igas ettevõttes üks ühele kasutatavad ning vastavalt ettevõtte taustale ja töökorraldusele oleks vaja ka kasutusele võetud raamistikku enda vajaduste ja eesmärkidega ühildada. Lisaks peab kasutusel olev raamistik toetama efektiivset töökorraldust muutuvas maailmas, kus kiirelt muutuvas olukorras on võimalik kõikide tööprotsesside läbiviimist jätkata ka kaugtööd tehes.

Autori lõputöö eesmärk on anda ülevaade agiilse arendusmetoodika ning Scrum raamistiku teoreetilisest taustast. Samuti teha ülevaade, kuidas antud raamistikku on implementeeritud rahvusvahelises tarkvara ja IT tooteid arendavas ettevõttes, kus iga osakonna töökorraldus on erinev. Lisaks on kirjeldatud muudatusi, mida tehti seoses 2019. aastal alanud koroonapandeemiaga, kui liiguti kaugtööle. Ülevaade antakse ühe osakonna näitel.

Käesolevas magistritöös on püstitatud järgmised uurimisülesanded:

- anda ülevaade Scrum raamistiku ajaloost;
- kirjeldada Scrum raamistikus olevaid rolle;
- kirjeldada Scrum raamistiku tegevusi ning nende omavahelisi seoseid;
- anda ülevaade Scrum raamistiku praktilisest kasutamisest IT ettevõttes.

Töö esimeses osas tehakse ülevaade agiilsest tarkvaraarendusest ning tarkvara elutsüklist. Lisaks tutvustatakse „Agiilse tarkvaraarenduse manifest”'i, kus on kirjas agiilse tarkvaraarenduse põhiväärtused. Töö teises osas antakse teoreetiline ülevaade Scrum'i raamistikust ning kirjeldatakse selle raamistiku rolle, artefakte ja tegevusi. Töö kolmandas osas kirjeldatakse Scrum'i raamistiku kasutamist Playtech Estonia OÜ *Data Services and Reporting* (edaspidi DSR) osakonna näitel ning analüüsitud sprindi planeerimise protsessi täpsust. Lisades on esitatud töövahendite näidised ning DSR osakonna eelnevate sprintide planeerimise andmed, mille alusel viidi läbi analüüs.

1. Kasutatavad mõisted ja terminid

Action item – tegevuspunkt või tegevuskava, mida tuleks teha.

Agile development – agiilne tarkvaraarenduse metoodika.

Burndown chart – maha põlemise, töö edenemise graafik, mis vertikaalteljel näitab järelejäänud töö hulka (kas tundides või toote kuhja üksuste ühikutes) aja jooksul, mida kuvatakse horisontaalteljel. Graafikult on näha tekkinud seisakuid või uute ülesannete juurde lisamisi sprindi käigus. Kuna aja jooksul peaks tööd järjest vähemaks jääma, on graafiku üldine suundumus põleda punktini, kus kogu töö on tehtud. [2]

Cross-functional team – ristfunktsionaalne meeskond. Ristfunktsionaalsed meeskonnad koosnevad erinevatest töövaldkondandest, kuid enam-vähem sama hierarhilise tasandi töötajatest, kes on kokku tulnud, et täita ülesanne [28].

Daily Scrum – lühike igapäevane koosolek, kus vaadatakse üle meeskonna edusammud, ülesanded ja seatakse eelseisvaks päevaks tööde prioriteedid; lisaks kasutatakse mõistet *stand-up* koosolek.

Demo – toote uut funktsionaalsuse tutvustamine, mis sai valmis tehtud eelneva sprindi jooksul.

Deployment – valmis saanud tarkvaratoote versioon installeeritakse klientide süsteemidesse.

DSR - *Data Services and Reporting* , Playtech Estonia OÜ osakonna lühend.

Epic – suur kasutajalugu, mis kirjeldab nõuet, mida hakatakse arendama. Kuna *epic* on liiga suur ühe sprindi jooksul arendamiseks, siis jagatakse see omakorda väiksemateks osadeks (kasutajalooks, ülesandeks). *Epic*'u näide: kliendina tahan ma osta kaupa e-poest.

Grooming – toote kuhja täpsustamine ja rafineerimine. Kasutatakse ka termineid *backlog refinement* ja *backlog management*.

Incremental development – inkrementaalne arendus, kus tarkvara arendatakse ja antakse kliendile üle väiksemate osade ehk inkrementide kaupa. Iga inkremendi arendamisel läbitakse kogu tarkvara elutsükkel.

Iterative development – iteratiivne arendus, kus arendatakse äärmiselt lühikeste iteratsioonide kaupa (näiteks 1 päev arendustööd) ning kliente kaasatakse arendusprotsessi pärast iga iteratsiooni lõppu. Võimaldab kiiremini reageerida muutustele, mis võivad projektis tekkida.

Kanban – agiilne lähenemisviis, mida kasutatakse lisaks olemasolevale protsessile, mis aitab visualiseerida töövoo liikumist süsteemis, piirata korraga töös olevaid üksuseid ning mõõta ja optimeerida töövoogu [2].

Kick-off – nn avalöök. Koosolek, mida tehakse enne arendamise algust, kus tutvustatakse arenduse sisu ja oodatavat tulemust.

POC – *proof of concept* ehk kontseptsiooni tõestus. Kasutatakse uute ideede testimisel, kui luuakse vastavalt ideele lihtne lahendus ja vaadatakse, kas see täidab ära suuremad vajadused, mis ülesandes on püstitatud.

Product backlog – toote kuhi; prioritiseeritud tegemata tööde nimekiri.

Product owner – toote omanik, lühendina kasutatakse ka PO.

Release – toote avalikustamine, välja laskmine.

Retrospective – tagasivaade viimasele lõppenud sprindile, mille käigus analüüsitakse, kuidas kulges viimane sprint ja millised olid murekohad ning kuidas neid järgmises sprindis vältida (vajadusel pannakse kirja *action item*'id).

Scrum – agiilse tarkvaraarenduse raamistik; kasutatakse ka meeskonna igapäevase koosoleku nimetusena (vt *daily scrum*).

Scrum of Scrums – lähenemisviis mitme Scrum'i meeskonna töö koordineerimiseks, kus igast Scrum'i meeskonnast üks või mitu liiget saavad kokku, et arutada ja lahendada meeskondade vahelisi sõltuvusprobleeme [2].

Sprint – arendustöö iteratsioon, mis on tavaliselt 2-4 nädala pikkune.

Sprint backlog – sprindi kuhi. Toote kuhjast valitud sprindi tööde nimekiri, mida soovitakse eelolevas sprindis valmis teha.

Sprint content – sprindi sisu ehk kõik ülesanded, mis on planeeritud sprinti.

Sprint planning – sprindi planeerimise tegevus, mille käigus valitakse eesolevasse sprinti need ülesanded, mida arendusmeeskond suudab valmis teha.

Sprint review – sprindi ülevaatamine sprindi lõpus, kus antakse ülevaade valminud arendustest.

Sprint status – lühikesed ülevaatlikud koosolekud käimasoleva sprindi hetkeseisu ülevaatamiseks ning probleemide tõstatamiseks ja lahenduste otsimiseks.

Stand-up – tavaliselt igal hommikul näost näkku toimuv lühike (kuni 10 minutiline) meeskonna koosolek, mis viiakse läbi püsti seistes.

Story points – numbriline väärtus, mis näitab ülesande/kasutajaloo suurust ja keerukust võrreldes teiste ülesannete/kasutajalugudega [13].

Task – ülesanne, mis tuleneb kasutajaloost ja on sellest veel täpsem ja väikesem. Nõuded, mida on võimalik arendada ühe sprindi jooksul. *Task* näide: kliendina tahan ma e-poes maksta LHV pangalingiga.

Umbrella feature – ehk vihmavari, mille alla on koondatud sarnasusel põhinevad featuurid. Kui arendusmeeskonna hallata on mitu toodet, siis koondatakse iga toote vihmavarju alla ainult selle konkreetse toote kasutajalood. Näiteks agiilsuse vihmavarju alla langevad agiilsed metoodikad, nagu näiteks Scrum, ekstreemprogrammeerimine, Kanban jne.

User Story – kasutatakse ka lihtsalt *story*. Kasutajalugu ehk kirjeldus nõudest, mida soovitakse saada. Võimalik arendada ühe sprindi jooksul. Kasutajaloo näide: kliendina tahan ma e-poes maksta pangalingiga.

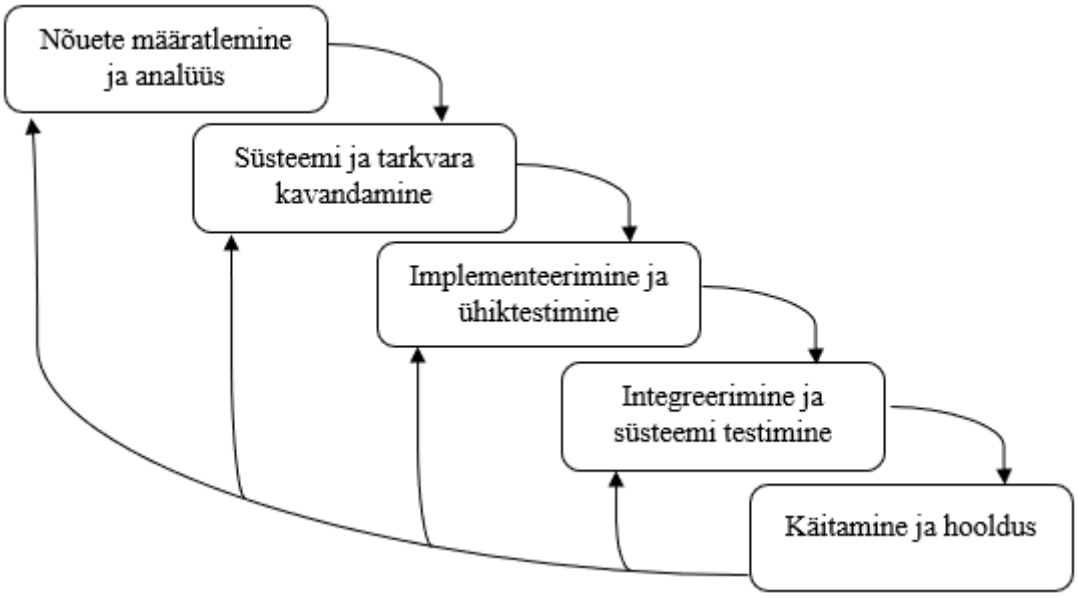
Velocity – ehk kiirus. Hinnang selle kohta, kui palju tegevusi toote kuhjast (vt. *product backlog*) suudab meeskond ühe sprindi jooksul ära teha [3].

2. Agiilse tarkvaraarenduse lähtepunktid

Tarkvaraarenduse metoodikaid ja raamistikke on aegade jooksul välja töötatud mitmeid erinevaid. Põhiliselt jagatakse need metoodikad kahte gruppi - traditsioonilised arendusmetoodikad, mille alla kuulub näiteks koskmudel (*waterfall model*) ning agiilsed arendusmetoodikad, mille alla kuulub ka Scrum. Traditsiooniliste metoodikate puhul tehakse valmis pikaajalised plaanid enne arendamise algust ning seejärel käib tarkvaratoote arendamine kindlate etappide kaupa vastavalt tehtud plaanidele. Uue etapi juurde liikuda ei saa enne, kui eelnev etapp on lõpetatud. Samas agiilse metoodika puhul tehakse lühiajalisemaid plaane ning arendusetapid on paindlikumad ja etappide vahel liikumine kiirem, mis tähendab ka kiiremini valmivaid tarkvaratooteid.

2.1. Tarkvara elutsükkel

Esimene 1970ndatel aastatel [3] kirjalikult avaldatud ja kasutusele võetud tarkvara arenduse protsessi mudel tuletati sõjatehnika tootearendusest, mis oma jadamisi ühest teise järgnevate faaside poolest on tuntud kui koskmudel või tarkvara elutsükkel, kuna see kirjeldab etappide kaupa tarkvara planeerimist alates selle sünnist kuni kasutuselevõtuni enne reaalselt arendama hakkamist.



Joonis 1. Koskmudel / tarkvara elutsükkel. Autori koostatud [3] põhjal.

Nagu nähtub ka jooniselt 1, siis etappide vahel tuleb liikuda järjest ning enne kui üks etapp ei ole lõpetatud, siis ei saa alustada ka järgmise etapiga ehk kui nõuded tarkvarale ei ole

määratletud, siis ei ole ka võimalik süsteemi ja tarkvara juba kujundada, kuna ei ole teada, millist funktsionaalsust peab antud süsteem täitma. Eelmise etapi juurde tagasi ei pöörduta ning tagasisidestamine toimub alles protsessi lõpus. Kui kliendi poolt peaksid tulema uued nõuded, siis neid saab analüüsida, arendada ja rakendada kogu protsessi uuesti läbides. Koskmudeli kasutamine eeldab äärmiselt head planeerimisoskust ja nõuete analüüsi, et vältida vigu ja probleeme hilisemates etappides.

Hakkas tekkima vajadus agiilse ja kiirema tarkvaraarenduse järgi ning järjest aktuaalsemaks muutus võime tulla kiiresti toime muutuvate protsesside ja nõuetega.

2.2. Inkrementaalne ja iteratiivne arendus

1980ndate aastate keskel tutvustas IBM [1] inkrementaalset arendust, kus tarkvara arendatakse ja antakse kliendile üle väiksemate osade ehk inkrementide kaupa. Sellest ajas saati hakati järjest tutvustama ka erinevaid iteratiivseid raamistikke, mis võimaldasid nõuete muutustele kiiresti reageerida ning arendust vastavalt sellele muuta. Srivastava [4] on oma artiklis oluliseks pidanud vahet teha, et inkrementaalse lähenemise puhul arendatakse tarkvara väiksemate inkrementide kaupa, kus iga inkremendi arendamisel läbitakse kogu tarkvara elutsükkel. Selline lähenemine annab kliendile jooksvalt võimaluse testida iga tarkvara inkrementi ja anda tagasisidet. Järgnev arendatav osa lisab uut funktsionaalsust, lisaks võib täiendada eelmist inkrementi ning kui kõik inkremendid on arendatud, siis on ka kogu tarkvaratoode valmis saanud. Iteratiivse lähenemise puhul on Srivastava [4] oluliste punktidena välja toonud võimaluse lisada muudatusi mistahes etapis, kuna arendatakse äärmiselt lühikeste iteratsioonide kaupa (näiteks 1 päev arendustööd) ning kaasata kliente arendusprotsessi sagedamini (pärast iga iteratsiooni lõppu). Selline lähenemine aitab vähendada vigu kliendi poolt soovitud funktsionaalsuses ning võimaldab kiiremini reageerida muutustele, mis võivad projektis tekkida.

1990ndate aastate lõpus [3] hakati järjest enam kasutama taolisi kiireid tarkvaraarenduse metoodikaid ning hakati arendama ideed agiilsetest meetoditest nagu Ekstreemprogrammeerimine (*extreme programming*), Scrum ja DSDM (*Dynamic System Development Method*).

2.3. Agiilse tarkvaraarenduse manifest

Erinevate agiilsete metoodikate arenedes ja kasutusele võttes hakati tundma ka vajadust üheselt määratleda ja kirja panna, millist metoodikat saab nimetada agiilseks. 2001. aasta veebruaris

said kokku 17 tarkvaraarenduse spetsialisti [5], kellest igaüks esindas erinevat tarkvaraarenduse metoodikat ning nad proovisid leida ühiselt erinevate metoodikate ühisosa. Sellest arutlusest sündis "Agiilse tarkvaraarenduse manifest", kus on kirjas agiilse tarkvaraarenduse põhiväärtused, mida järgitakse [5]:

a) hinnatakse rohkem inimesi ja nendevahelist suhtlust, kui protsesse ja arendusvahendeid;
b) hinnatakse eelkõige töötavat tarkvara, kui kõikehõlmavat dokumentatsiooni;
c) hinnatakse rohkem koostööd kliendiga arendusprotsessis, kui läbirääkimisi lepingute üle;
d) hinnatakse kiiret reageerimist muutunud oludele, kui algse plaani järgimist.

Oluline on lisada, et kõikidel välja toodud teguritel on arendusprotsessis oma väärtus, kuid nende kaalud ja prioriteedid on erinevad. Põhiväärtusi toetavad ka 12 eraldi kirja pandud põhimõtet, millest võib välja tuua järgmise [5]: "Mõistame muutuvaid olusid, isegi kui need ilmnevad arenduse lõppjärgus. Agiilsed meetodid pööravad sellised muutused meie kliendi konkurentsieeliseks."

"Agiilse tarkvaraarenduse manifesti" saab kokku võtta järgnevalt: keskendutakse töötava tarkvara loomisele lihtsaid põhimõtteid järgides ning samal ajal peetakse tähtsaks inimesi meeskonnas, kes seda teevad, mitte keerulist protsessi. Samas ei saa ka öelda, et protsesse ning dokumentatsiooni ei oleks üldse vaja, vastupidi, neid on siiski vaja, aga need ei ole esmatähtsad. Samuti hinnatakse kiiret reageerimist muutustele, mida võimaldavad erinevad agiilsed raamistikud, näiteks Scrum.

3. Scrum

Scrum'i kirjeldatakse kui kerget raamistikku, mis aitab inimestel, meeskondadel ja organisatsioonidel keeruliste probleemide jaoks adaptiivseid lahendusi kasutades väärtust luua [7]. Järgnevates peatükkides antakse ülevaade selle raamistiku kujunemisest ning ülesehitusest.

3.1. Scrum'i ajaloo algus

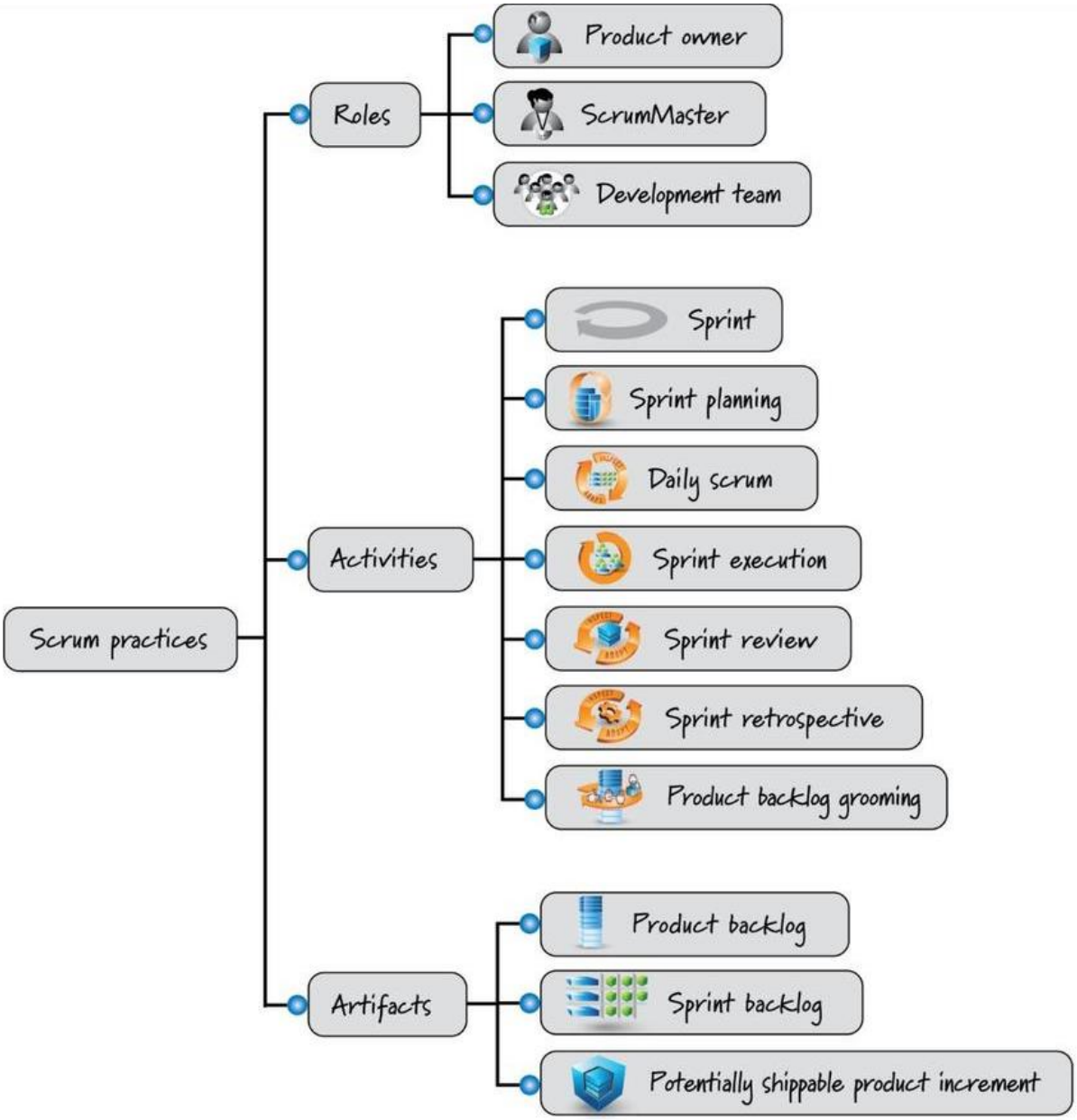
Scrum'i ajaloo alguseks on võimalik lugeda aastat 1986 [2], kui ilmus *Harvard Business Review* artikkel "*The New New Product Development Game*", kus Hirotaka Takeuchi ja Ikujiro Nonaka kirjeldasid, kuidas sellised ettevõtted nagu Honda, Canon ja Fuji-Xerox saavutasid maailmaklassi tulemusi kasutades tootearenduses mõõdetavat ning meeskonna põhist nö kõik-korraga (*all-at-once product development*) lähenemist. Samuti rõhutati artiklis iseorganiseeruvate meeskondade tähtsust ning kirjeldati juhtkonna rolli arendusprotsessis. Antud artikkel oli mõjukas ja andis tõuke erinevate kontseptsioonide ja mõistete kokku koondamisele ühise nimetaja Scrum alla [2], mis omakorda on laensõna ragbist, kus see viitab ragbimängu jätkamise viisile pärast juhuslikku rikkumist või kui pall on mängust välja läinud.

1993. aastal lõi Jeff Sutherland [2] koostöös oma meeskonnaga tarkvaraarenduses kasutamiseks mõeldud Scrum'i protsessi, kombineerides selle jaoks kokku kontseptsioonid 1986. aastal ilmunud artiklist, objektorienteeritud arendusest, empiirilisest protsessijuhtimisest, inkrementaalsest ja iteratiivsest aredusest, tarkvaraprotsessi ja tootlikkuse uuringutest ning keerulistest adaptiivsetest süsteemidest. Kaks aastat hiljem, 1995. aastal, avaldasid Jeff Sutherland ja Ken Schwaber ühiselt esimese kirjutise „*The SCRUM Development Process*", mis Scrum'i laiemalt tutvustas [29]. Alates sellest ajast on järjest avaldatud mitmeid artikleid ja publikatsioone, mis kirjeldavad Scrum'i olemust ja selle kasutusvõimalusi. Suure panuse on andnud ka Mike Cohn, arendades kasutajalugusid, kui tööriista kliendile orienteeritud tööeesmärkide kirjeldamiseks ning arendades *story point*'i, kui võimalust mõõta töö hulka ja meeskondade kiirust (*velocity*) [29].

Scrum'i ei peeta standardiseeritud metoodiliseks protsessiks vaid raamistikuks, mis aitab tööd meeskonnas korraldada ja juhtida [2]. See annab võimaluse igale meeskonnale, kes Scrum'i kasutab, muuta kasutatav raamistik omanäoliseks ja sobivaks vastavalt vajadusele.

3.2. Scrum’i ülevaade

Järgnev joonis 2 annab ülevaate Scrum’i väljakujunenud raamistikust (*Scrum practices*), mis on koondatud 3 suuremasse grupeeringusse: Scrum’i meeskonna rollid, tegevused ja artefaktid.



Joonis 2. Scrum’i väljakujunenud raamistik [2].

Rollidena on Scrum’i raamistikus kasutusel toote omanik, Scrum Master ja arendusmeeskond. Scrum’i tegevused on sprint, sprindi planeerimine, igapäevane Scrum, sprindi teostamine, sprindi ülevaatamine, sprindi retrospektiiv ja toote kuhja groomimine ehk hindamine ja rafineerimine. Scrum’i artefaktidena kasutatakse toote kuhja, sprindi kuhja ja toote inkrementi, mida on võimalik kliendile tarnida.

3.3. Scrum’i rollid

Joonisel 2 lk 13 on välja toodud 3 põhilist rolli, mida Scrum’i raamistikus ühes meeskonnas kasutatakse:

a) toote omanik

b) Scrum Master

c) arendusmeeskond

Schwaber ja Sutherland [7] täpsustavad Scrum’i juhendis, et Scrum’i meeskonnas on üks toote omanik, üks Scrum Master ja arendusmeeskond ning Scrum’i meeskonnas ei ole alagruppe ega hierarhiaid ja meeskond on ühtne spetsialistide üksus, mis keskendub korraga ühele eesmärgile.

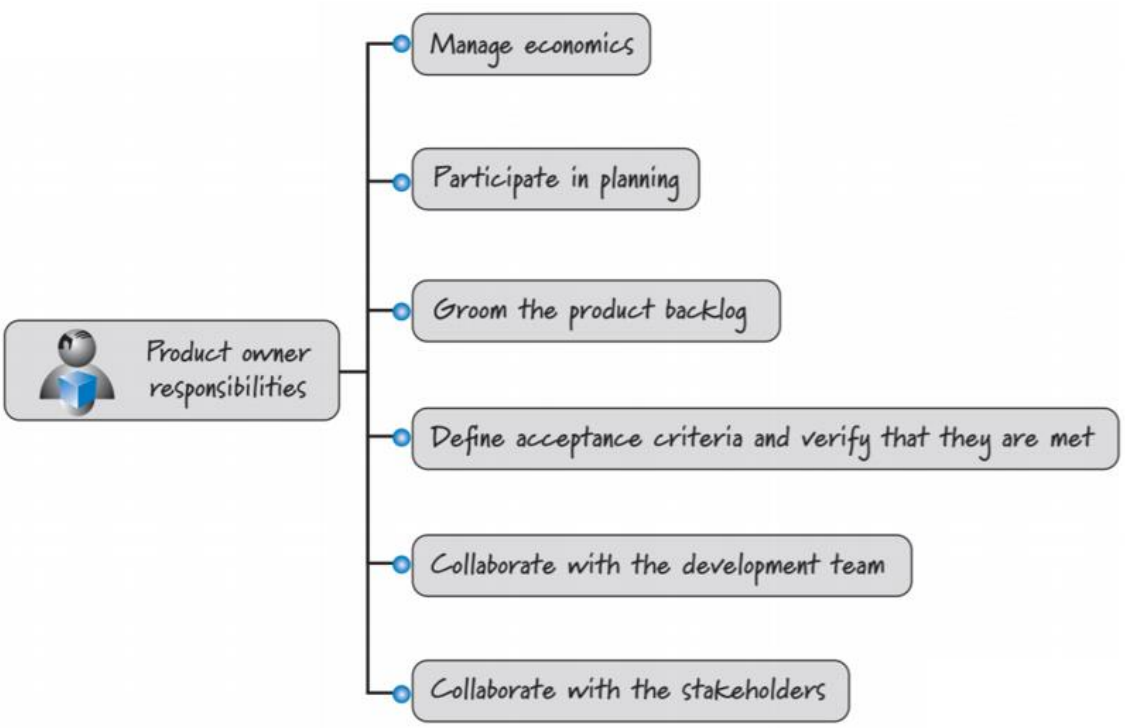
3.3.1. Toote omanik

Toote omaniku roll on vastutada selle eest, mida arendatakse ja millises järjekorras seda tegema peaks. Sommerville [3] lisab, et toote omaniku ülesanne on toote nõuete väljaselgitamine, prioritiseerimine ja toote kuhja (*product backlog*) pidev ülevaatamine, et tagada projekti vastamine kriitilistele ärivajadustele. Rubin [2] täpsustab, et toote omanik on kui tootejuhtimise keskpunkt, kelle ülesanne on otsustada, milliseid funktsionaalsuseid ja mis järjekorras arendatakse ning kommunikeerida selge nägemus sellest, mida Scrum’i meeskond üritab saavutada ka kõigile teistele osapooltele. Rubini arvates peaks toote omanik vastutama kogu arendatava või hooldatava lahenduse üldise edukuse eest, tegema aktiivset koostööd Scrum Master’i ja arendusmeeskonnaga ning peaks olema igal ajal valmis vastama kõikidele tekkivatele küsimustele.

Kui seda käsitlust kokkuvõtlikult vaadata, siis on tooteomaniku roll kahe suunaga. Esimese suunana peab toote omanik omama ülevaadet ettevõtte ärilistest eesmärkidest [2] ning tarkvara toote tellija nõuetest ning teise suunana peab toote omanik panema paika eesmärgid Scrum’i meeskonnale ning prioritiseerima arendusi vastavalt nendele paika pandud ärilistele eesmärkidele ja klientide ootustele. Põhilised toote omaniku kohustused on välja toodud joonisel 3 lk 15.

Kuigi jooniselt lähtub, et toote omaniku õlule langeb äärmiselt palju kohustusi ning neid võiks jagada mitme inimese vahel, siis Scrum’i teoorias on algselt nähtud selles rollis ikkagi ühte vastutavat inimest. Samas on Rubin [2] öelnud, et teatud tingimustel võivad olla otstarbekad ka omaette toote omanike meeskonnad või toote omaniku volituste edasi andmine meeskonna

teistele liikmetele, kuid ka sellisel juhul peab olema üks toote omanik, kelle võimuses ja vastutusalas on lõplike otsuste tegemine.



Joonis 3. Toote omaniku põhikohustused [2].

Joonisel 3 olev esimene põhiülesanne on hallata majanduslikku poolt ehk vastutada heade majanduslike otsuste langetamise ees nii toote avalikustamise (*release, deployment*), sprindi kui ka toote kuhja tasemel [2]. Tal peab olema võime analüüsida seoseid majandusliku seisukorra ning toote arendusse võtmise prioriteetide vahel ning neid vastavalt sellele muuta. Kui toote omanik näeb, et majanduslikult ei ole kasulik enam arendust X teha, sest selle toote nõudlus on muutunud ning loodetud kasumi numbrit ei saavutata, siis on tema võimuses selle arenduse prioriteedi vähendamine ning see toote kuhja lõppu tõsta.

Toote omanik peab osalema ka planeerimise protsessides. Tal on võtmeroll nii portfoolio-, toote-, avalikustamiste- ja sprindi planeerimise tegevustes [2], kus ta peab koostööd tegema nii ettevõtte siseste huvirühmadega kui ka arendusmeeskonnaga, et paika panna need suunad kuhu ja kuidas tootega liigutakse. Sprindi planeerimise protsessi on pikemalt kirjeldatud peatükis “Sprindi planeerimine”.

Toote omanik on vastutav toote kuhja korrastamise (*grooming*) eest, mis hõlmab endas nii toote kuhja arendusülesannete loomist ja täpsustamist, hindamist kui ka prioritiseerimist [2]. Tasub tähele panna, et toote omanik on vastutaja rollis ning ei pea üksi neid kõiki tegevusi tegema –

tegevused on tehtud koostöös või jagatud erinevate osapooltega. Korrastamise protsesside olemust on pikemalt kirjeldatud peatükis “Toote kuhi” ja „Toote kuhja hindamine“.

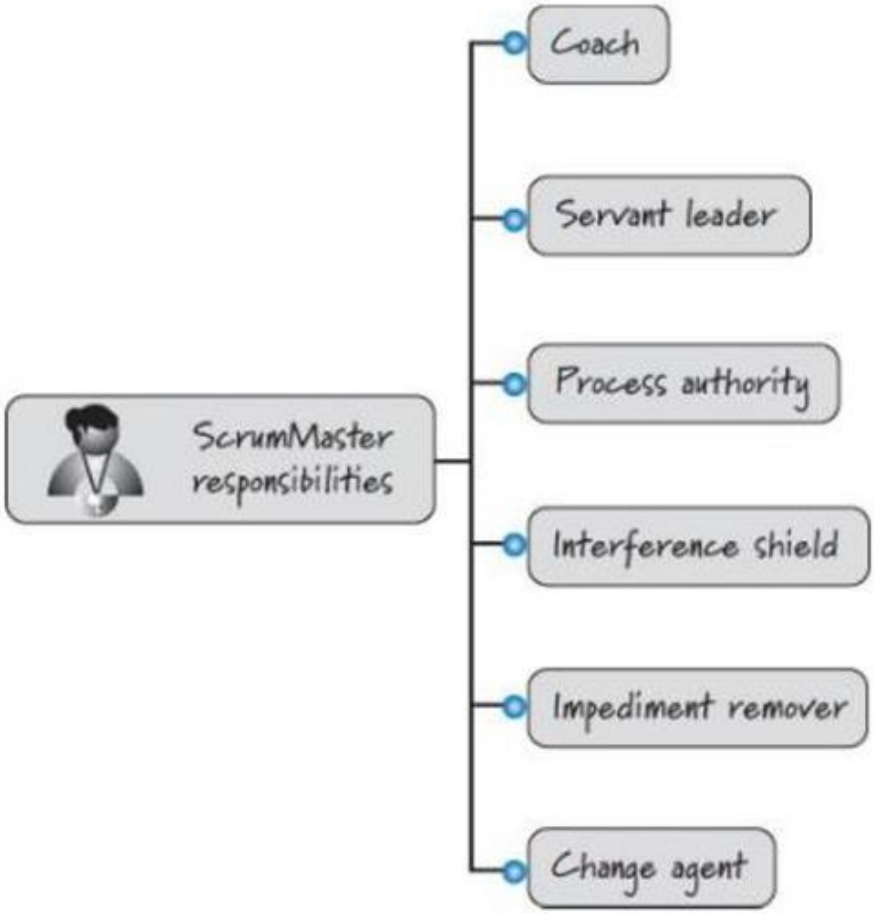
Lisaks vastutab toote omanik iga toote kuhja ülesande vastuvõetavuse kriteeriumi määratlemise eest ehk millised on need tingimused, mille korral toote omanik oleks funktsionaalsete ja mittefunkstionaalsete nõuete täitmise puhul rahul [2]. Toote omanik peab tagama vastuvõetavuse kriteeriumite olemasolu enne seda, kui antud arendusülesanne sprindi planeerimisel kaalumisele võetakse, et vältida arendusmeeskonna puudulikku arusaamist antud ülesandest ning toote omanik peaks olema lõplik otsustaja selle üle, kas pärast arenduse lõppu ka see ülesanne vastab algselt määratud ootustele [2].

Joonisel 3 lk 15 kahe viimase ülesandena on nimetatud koostööd nii arendusmeeskonnaga kui ka erinevate huvirühmadega. Toote omaniku roll on olla igapäevaselt kaasatud, anda arendusmeeskonnale ajakohast tagasisidet, et saavutataks oodatavad tulemused [2]. Et toote omanik oleks oma töös edukas peaks Schwaberi ja Sutherlani [7] arvates kogu organisatsioon tema otsuseid, mis on seotud toote kuhja prioritiseerimisega, austama, kuna toote omanik võib esindada prioritiseerimisel paljude osapoolte vajadusi. Seega need, kes soovivad toote kuhja muuta ning oma soovidele toote osas kõrgemat prioriteeti saada, saavad seda teha eeskätt toote omanikku veendes ning põhjendades, miks peaks seda arendust teistest kõrgemale tõstma [7].

3.3.2. Scrum Master

Teine oluline roll Scrum’i raamistiku meeskonnas on Scrum Master. Tema põhiline roll on aidata kõikidel osapooltel mõista ja omaks võtta Scrum’i väärtusi, põhimõtteid ja tavasid [2]. Teisisõnu on tema roll olla toetav üksus nii arendusmeeskonna kui ka toote omaniku vahel, olla abiks oma teadmistega ja aidata meeskonnal täita Scrum’i raamistikus välja toodud põhimõtteid. Samas tuleb tähele panna, et Scrum Masteril ei ole volitusi otseselt meeskonda juhtida, seega ei saa seda rolli võrrelda traditsioonilise projektijuhi või arendusjuhi rolliga [2].

Joonisel 4 lk 17 on välja toodud Scrum Masteri põhilised kohustused, mille on Rubin [2] oma raamatus välja toonud ja ka üksikasjalikult iga punkti lahti seletanud.



Joonis 4. Scrum Master'i põhilised kohustused [2].

Esimese punktina on joonisel mainitud juhendaja ehk *coach*'i rolli. Rubin on [2] Scrum Master'it kirjeldanud nö agiilse *coach*'ina, kes, juhendades ülejäänud kahte Scrum'i meeskonna rolli (toote omanikku ja arendusmeeskonda), aitab kõrvaldada takistused nende kahe meeskonna rolli vahel ja seeläbi võimaldab toote omanikul otseselt juhtida arendust. Samas tasub tähele panna, et Scrum Master'ile ei saa omandada üks ühele agiilse *coach'*i nimetust ja rolli, kuna nende kahe rolli puhul on sisulisi erinevusi. Artiklis [6] "*Differences Between Agile Coach and Scrum Master*" nenditakse samuti fakti, et need kaks rolli on omavahel sarnased - mõlemal juhul on põhikohustus aidata meeskondadel arendada agiilset mõtteviisi ja nad võivad kasutada selleks ka sarnaseid tehnikaid, kuid erinevus seisneb nende rollide ulatuses. Erinevused [6] tulevad esile juhendatavate meeskondade arvu juures - Scrum Master töötab tavaliselt konkreetse meeskonnaga, samas kui agiilne *coach* töötab toote tasemel, kaasates mitmeid erinevaid meeskondi ning kasutatavate raamistike juures - Scrum Master kasutab oma töös Scrum'i raamistikku, mida ta võib täiendada ka Kanban'iga, samas kui agiilne *coach* võib anda nõu mitmete erinevate agiilsete tehnikate ja vahendite kasutamisel.

Seega eeldatakse, et agiilsel *coach*'il [6] on süvateadmised mitmetest erinevatest metoodikatest, ta on sõltumatu, iseseisev juhendaja ja vastutab erinevate meeskondade või ka juhtkonna juhendamise eest. Rubin lisab [2], et Scrum Master juhendajana jälgib, kuidas meeskond kasutab Scrum'i ning kui peaksid tekkima probleemid, siis aitab meeskonnal leida neile lahendusi, mitte ei lahenda neid meeskonna eest ise. Scrum'i protsessi juhina (*process authority*) aitab Scrum Master meeskonnal pidevalt protsesse täiustada, et maksimeerida väärtust, mida luuakse ettevõttele [2].

Lisaks juhendajale on Rubin [2] Scrum Master'it iseloomustanud ka kui meeskonna teenrit (*servant leader*), kes tagab meeskonna esmatähtsate vajaduste rahuldamise ning ei küsi meeskonnalt "Mida te täna minu jaoks teete?", vaid "Mida mina saan täna teha, et aidata teil ja kogu meeskonnal olla veel tõhusam?". Siia haakuvad ka järgmised kohustused olla kaitsekilbiks erinevate sekkumiste eest (*interference shield*) ja aidata eemaldada meeskonnal tekkida võivaid takistusi (*impediment remover*). Igasugused välised ja sisemised sekkumised võivad tulla erinevatest allikatest [2], näiteks juhtidelt, kes võivad sprindi keskel soovida meeskonnaliikmeid suunata tegelema teistelt meeskondadelt tulenevate küsimuste ja ülesannetega ning Scrum Master peaks selles olukorras toimima vahelülina, et meeskond saaks jätkuvalt keskenduda oma eesmärkide täitmisele. Lisaks sellele peaks Scrum Master eemaldama ka meeskonna produktiivsust pärssivaid takistusi [2], kui meeskonnaliikmed ise sellega hakkama ei saa - näiteks kui meeskonna vastutusalas ei ole serverite ebastabiilsusega tegelemine, aga see pärsib nende tööd, siis Scrum Master saab siinkohal võtta enda ülesandeks tegeleda sellega koos serverite töö eest vastutavate inimestega.

Viimase kohustusena on Rubin [2] välja toonud Scrum Master'i võtmeisiku ja muutuste elluviija (*change agent*) rolli, mis ei eelda ainult eelmises punktis välja toodud füüsiliste takistustega tegelemist, vaid ka mõtteviiside muutmist ehk Scrum Master aitab mõista muutuste vajalikkust, Scrum'i mõjusid väljaspool Scrum'i meeskonda ja laiaulatuslikku kasu, mida Scrum aitab saavutada. Lisaks peaks Scrum Master tagama [2], et tõhusad muutused toimuvad organisatsiooni kõikidel tasanditel, võimaldades sellega mitte ainult lühiajalist edu Scrum'i kasutamisest, vaid veelgi olulisemalt ka pikaajalist kasu.

Scrum Master'i roll meeskonnas ei pruugi olla täiskohaga töö, sest meeskonnad, kes on pikemat aega koos töötanud ja edukalt Scrum'i raamistiku põhimõtteid rakendanud vajavad aja jooksul järjest vähem suunamist ja juhendamist Scrum Master'i pool, seega võib seda rolli kombineerida ka teiste rollide või kohustustega [2]. Eelkõige soovitab Rubin [2] rolli laiendada teiste meeskondade Scrum Master'iks olemisele ja teadmiste jagamisele, kui võtta lisakohustus

näiteks arendusmeeskonna liikmena, kuna sellisel juhul võib tekkida huvide konflikt. Ta toob näiteks olukorra, kui Scrum Master'il on käsil olulised Scrum'i raamistiku tegevused (näiteks meeskonna takistuste eemaldamine) ja samal ajal arendusmeeskonna arendajana kriitilise defekti parandamine. Mõlemad ülesanded on olulised ning ühe eelistamine teisele võib pärssida arendusmeeskonna tööd.

3.3.3. Arendusmeeskond

Traditsioonilises tarkvara arenduses [2] mõeldakse arendusmeeskonnast rääkides erinevaid ametipositsioone nagu näiteks arhitektid, arendajad, testijad, andmebaasi administraatorid, UI (*user interface*) ehk kasutajaliidese disainerid ja nii edasi, kuid Scrum määratleb arendusmeeskonna rolli, mis on oma olemuselt mitmekesine ristfunktsionaalne iseorganiseeruv kogum inimesi, kes vastutavad soovitud tarkvaratoote kujundamise, arendamise ja testimise eest. Rubin on arendusmeeskonna kirjeldamiseks välja toonud järgnevad punktid [2]:

a) iseorganiseeruv meeskond, kes peaks ise leidma parima viisi toote omaniku poolt määratud eesmärgi saavutamiseks;

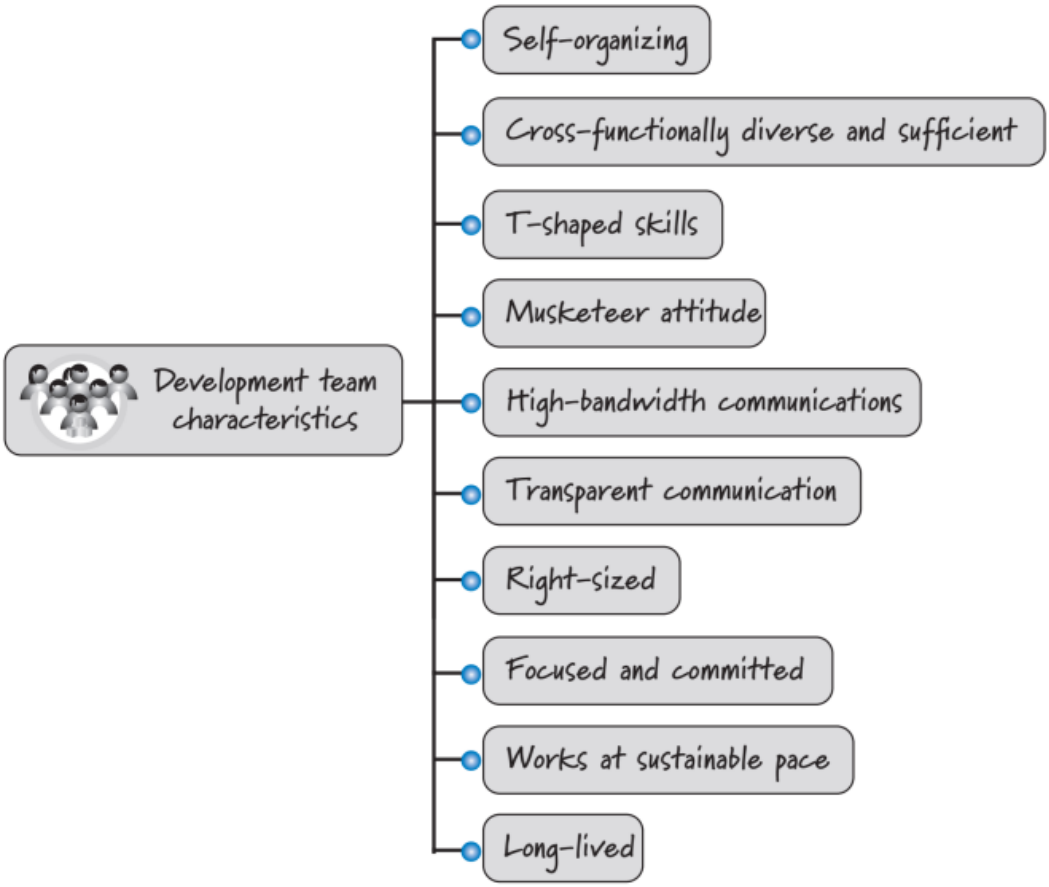
b) tüüpiliselt 5-9 liikmeline meeskond;

c) liikmetel peavad ühiselt olema vajalikud oskused kvaliteetse ja töötava tarkvara loomiseks.

Kui võrrelda Scrum'i arendusmeeskonda traditsiooniliste organisatsioonide meeskondadega, siis on neis kahes üsna palju erinevusi. Rubini sõnul [2] on paljud organisatsioonid harjunud erinevaid rolle jagama rolli spetsiifilistesse meeskondadesse. Ehk teisisõnu võib organisatsioonis olla eraldi meeskond arhitekte, eraldi meeskond analüütikuid, eraldi meeskond arendajaid, testijaid ja nii edasi. See omakorda tähendab, et meeskonnad toimivad üksteisest sõltumatult ja kui ühes meeskonnas saab tööülesanne lõpetatud siis antakse see edasi järgmisele meeskonnale (näiteks analüüs tehtud ja arendusse antud või arendus lõpetatud ja testimisse saadetud) ning ise hakatakse tegelema järgmise projektiga [2].

Seevastu Scrum'i meeskonnas peab arendusmeeskond tegema igas sprindis ära kogu töö - nii disaini, funktsionaalsuse arendamise, integreerimise kui ka testimise, et sprindi lõpus saaks üle anda töötava funktsionaalsuse ning selline töökorraldus eeldab kogemustega meeskonnaliikmeid igast valdkonnast [2].

Joonisel 5 on välja toodud hulk omadusi, mis ühel Scrum'i arendusmeeskonnal olema peaksid - meeskond peaks olema iseorganiseeruv ning ristfunktsionaalselt mitmekesine, mis tähendab et nad on võimelised olemasolevate liikmetega ära tegema kogu töö otsast lõpuni. Meeskonna liikmetel peaks olema ka nii öelda T-kujulised oskused, mida Rubin [2] on kirjeldanud järgnevalt - T tähe katus kui laius, mis hõlmab endas inimese võimet töötada laialdaselt väljaspool oma tuumikala ja T tähe post kui sügavus, mis kirjeldab inimese funktsionaalse ala süvateadmisi. Eesmärk on meeskond üles ehitada suures osas liikmetest, kellel on olemas oskused tuumikala katmiseks, kuid kellel oleks samas ka mingid kattuvad oskused enda tuumikalast väljaspool olevate liikmetega ning vajadusel lisada meeskonna koosseisu ka mõni spetsialist [2].



Joonis 5. Arendusmeeskonna peamised omadused [2].

Lisaks peaks meeskonna liikmetel olema musketäride suhtumine, mida me kõik kirjeldame lausega "Üks kõigi ja kõik ühe eest", mis tähendab, et kogu meeskond peaks panustama ühise eesmärgi nimel - saada valmis töötav ja kvaliteetne tarkvaratoode. Kui mõni meeskonnaliige

peaks olema vastupidise suhtumisega, siis pärsiks see ka lõpp eesmärgi saavutamist kogu ülejäänud meeskonnal.

Tähtsal kohal meeskonnas on ka kommunikatsioon - nii selle kiirus ja efektiivsus (*high-bandwidth*) kui ka läbipaistvus (*transparent*). Väärtuslikku informatsiooni tuleb vahetada üksteisega meeskonna sees kui ka toote omaniku ja Scrum Master'iga ning seda tuleks teha viisil, mis on kiire ja tõhus, sest see suurendab nii teabe jagamise sagedust kui ka kvaliteeti [2]. Arvestades neid põhimõtteid saab öelda, et mida kiiremini liigub informatsioon (näiteks nõuete muutustest analüüsis), seda kiiremini saab reageerida arendusmeeskond nendele muutustele ning kohandada vastavalt sellele oma tööülesandeid ja liikuda tarkvara tootega kliendi soovitud suunas. Rubin [2] toob välja ka meetmed, mis aitavad efektiivset kommunikatsiooni saavutada:

a) eelistada tuleks näost näkku suhtlust;

b) kui vähegi võimalik, peaksid meeskonnaliikmed asuma füüsiliselt lähestikku;

c) hajutatud meeskondade puhul peaks olema võimaldatud tehnoloogiline tugi kommunikatsiooni parandamiseks, näiteks erinevad telekonverentsi seadmed;

d) eelistada tuleks ristfunktsionaalseid meeskondi, kuna neil on tõhusamad suhtluskanalid juba seetõttu, et nad teavad neid meeskonnaliikmeid, keda on vaja teatud tööülesannete jaoks ning vähendatud on tööülesannete üleandmine teistele meeskondadele;

e) ühes meeskonnas olemine vähendab ametlikke tööülesannete üleandmise sagedust ja formaalsust, mis omakorda parandab suhtlemise kiirust;

f) vähendada erinevatele tseremoniaalsetele tegevustele kulutatud aega, kus meeskonna liikmed peavad läbima protsesse ja tavasid, mis lisavad vähe väärtust - näiteks kui meeskonna liikmed peavad enne tegeliku kliendi või kasutajaga rääkimist läbima kolm eraldatuse taset, võib seda "kliendiga rääkimise" tseremoniaalsust pidada efektiivse kommunikatsiooni takistuseks;

g) väiksemates meeskondades on vähem suhtluskanaleid, mis parandavad kommunikatsiooni kvaliteeti, kuna suhtluskanalite arv ei suurene lineaarselt vaid valemi N(N-1)/2 järgi, kus N on meeskonna liikmete arv (näiteks 5 liikmelises meeskonnas on suhtluskanaleid 10, aga 10 liikmelises meeskonnas juba 45).

Läbipaistev suhtlus lisab selge arusaamise sellest, mis tegelikult toimub ning aitab seeläbi vältida üllatusi ja aitab meeskonna liikmete vahel usaldust luua [2].

Samuti on tähtis õige suurusega arendusmeeskond, sest Scrum toimib kõige paremini just väikestes meeskondades. Üldreeglina on 5-9 liikmeline meeskonda peetud kõige paremaks [2]. Ka Scrum Guide [7] toob välja selle, et arendusmeeskond võiks olla keskmiselt 10 või vähema liikmeline, sest meeskond peab olema võimalikult väike, et olla jätkuvalt oma töös väle ning samas piisavalt suur, et teha ühe sprindi jooksul ära märkimisväärne töö. Rubin ise [2] peab oma 25 aastase Scrum'iga töötamise kogemuse pealt kõige optimaalsemaks 5-7 liikmelist meeskonda.

Mike Cohn [8] ei tahaks meeskonna suurust ära määrata konkreetse numbriga ning eelistab kasutada Deutschmani [9] kirjeldatud lähenemist, mida hakati kasutama Amazonis, kui inimesed hakkasid kurtma, et meeskondades peaks olema parem kommunikatsioon - Jeff Bezos mõtles välja mõiste "kahe pitsa meeskond", mis tähendab seda, et kui kahe pitsaga ei saa meeskonda ära toita, siis on meeskond liiga suur. Optimaalne meeskonna suurus selle mõiste järgi oleks siis sõltuvalt inimeste isust 5-7 inimest [9].

Cohn [8] on lisaks välja toonud eelised, miks võiks hoida meeskonna väikesena:

a) vähem sotsiaalset logelemist - inimesed pingutavad vähem, sest nad arvavad, et suures meeskonnas on ikka keegi, kes võtab seisvad ülesanded üle ja teeb need lõpuni;

b) konstruktiivne koostöö toimub pigem väiksemates meeskondades, kuna enam kui 10-12 liikmelises meeskonnas on inimestel keeruline luua usalduslikku ning vastastikkuse vastutuse tunnet;

c) vähem aega kulutatakse koordineerimisele ja kogu meeskonna osalusega koosolekutele aja leidmisele ja planeerimisele;

d) keegi ei saa jääda tahaplaanile ja meeskonnaliikmete osavõtt suureneb - suurtes meeskondades on tavaliselt madalam kõikide meeskonna liikmete osavõtt grupi tegevustes ja aruteludes;

e) väikesed meeskonnad on liikmetele meeldivamad ja pakuvad rohkem rahuldust, kuna väikese meeskonna puhul on ühe inimese panus nähtavam ja sisukam;

f) demoraliseeriv üle spetsialiseerumine on vähem tõenäoline ning suurema projekti puhul võtavad inimesed suurema tõenäosusega erinevaid rolle.

Rubin [2] lisab samas, et see, et Scrum'i kasutamine soosib väiksemaid meeskondi, ei tähenda, et Scrum'i ei saaks kasutada suuremate arendustegevuste jaoks, mis hõlmab endas rohkem inimesi - näiteks kui ühe projekti kallal on tööd tegemas 36 liikmeline arendusmeeskond, siis

selle asemel, et juhtida seda üüratut meeskonda, võiks kõik liikmed jagada väiksemateks Scrum'i arendusmeeskondadeks, kus oleks 9 või vähem liiget ning igal meeskonnal oma ülesanded. Mitu meeskonda saavad omavahel ülesandeid kooskõlastada mitmel erineval viisil, Rubin [2] on ühe levinud viisina toonud välja *scrum of scrums* meetodi, mis pakub võimalust ühendada mitu meeskonda koostöö tegemiseks. Sommerville [3] toob välja mitme meeskonnaga Scrum'i peamised omadused, mis võiks olla täidetud:

a) igal eraldiseisval meeskonnal peaks olema oma toote omanik ja Scrum Master ning kogu tervikprojekti jaoks võiks olemas olla ka peamine toote omanik ja Scrum Master;

b) iga meeskond peaks valima oma toote arhitekti ja iga meeskonna toote arhitektid peaksid tegema koostööd tarkvara toote ja/või süsteemi üldise arhitektuuri kujundamiseks ja arendamiseks;

c) tarkvara toodete väljalaskmine/avalikustamine peaks olema sünkroniseeritud iga meeskonnaga selleks, et saaks välja anda täieliku ja töötava toote/süsteemi;

d) teha igapäevaseid *scrum of scrums* koosolekuid, kus iga meeskonna esindajad saavad arutada tehtud edusamme, tuvastada probleeme ja planeerida selleks päevaks tehtavat tööd.

Meeskonna sees on suure tähtsusega ka keskendunud ja pühendunud olek meeskonna eesmärkide saavutamiseks. Rubin [2] täpsustab, et keskendunud olek tähendab iga meeskonnaliikme jaoks tegelemist, keskendumist ja oma tähelepanu suunamist meeskonna eesmärgile ning pühendunud olek tähendab meeskonna eesmärkidele pühendumist nii headel kui ka halbadel aegadel.

Lisaks peaks kogu meeskond töötama jätkusuutlikus tempos. Rubini [2] arvates peaks see vähendama võimalust, et meeskonna liikmed peaksid mõnes etapis tegema palju ületunde ning oleksid seeläbi läbi põlenud.

Arendusmeeskond peaks olema ka pika elueaga. Rubin [2] on välja toonud Ralph Katz'i 1982. aastal avaldatud uurimuse "*The Effects of Group Longevity on Project Communication and Performance*" tulemuse, et pikaealised meeskonnad on oma töös produktiivsemad kui äsja moodustatud rühmad. See on ka üsna loogiline, kuna pikaealise meeskonna liikmed on juba varasemalt koostööd teinud ning tunnevad oma kaaslaste oskusi ja ülesannete lahendamise võimekust - nad teavad, kelle poole pöörduda erinevate teemadega, kes omab teatud toote osa arendamisel sügavaid teadmisi (ehk eelpool kirjeldatud T-kujuliste oskuste posti sügavuse osa) ning kes suudab efektiivselt aidata meeskonda probleemide esinemisel. Äsja moodustatud

meeskondade puhul läheb aega üksteise tundmaõppimisele, usaldamisele ning kohanemisele, enne kui hakkab toimima edukas ja produktiivne koostöö ning individuaalsete ülesannete lahendamise asemel muutub meeskond terviklikuks ja üheskoos saavutatakse püstitatud eesmärgid.

3.4. Scrum raamistiku tegevused ja artefaktid

Järgnevates peatükkides selgitatakse lähemalt joonisel 2 lk 13 toodud Scrum'i tegevusi ja artefakte. Need kõik on omavahel seotud Scrum'i sprindis. Scrum'i raamistiku selgrooks loetakse sprinti ja kõiki sinna juurde kuuluvaid tegevusi, mis ulatuvad toote kuhja ülevaatamisest sprindi planeerimise, teostamise ja tagasiulatuvate ülevaadete tegemiseni.

3.4.1. Sprint

Sprindiks nimetatakse fikseeritud (tavaliselt 2-4 nädalat) pikkusega iteratsiooni, mille käigus valmib tarkvara toode/inkrement, mis peaks kliendile ja/või kasutajale looma mingisugust reaalset väärtust või uut funktsionaalsust.

Sprindid on [2]:

a) Scrum'i raamistiku skelett;

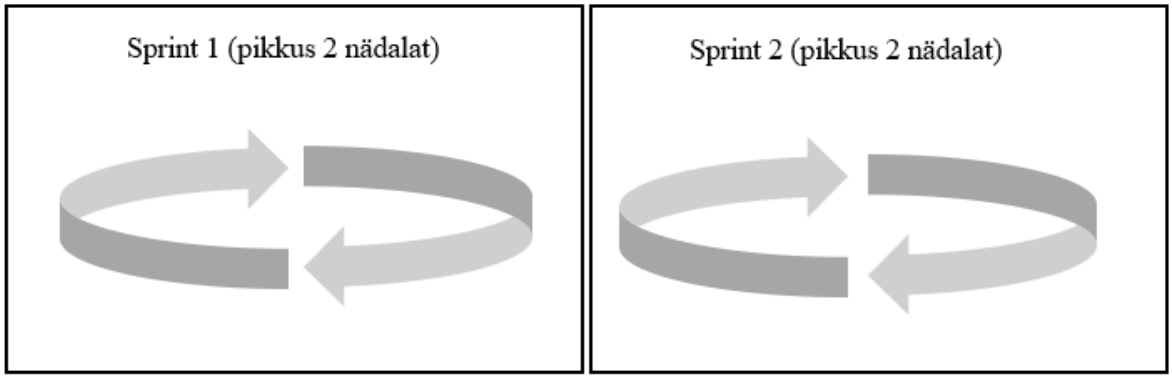
b) ajaraamistikuga kindlaks määratud (*timeboxed*);

c) ühtlase pikkusega;

d) kindla algus- ja lõpu ajaga;

e) lühikese kestusega (1 nädal kuni 1 kuu).

Seega sprindid on üksteisele järgnevad tegevuste ajaaknad, mis on reeglina sama pikad ja millel on kindel algus- ja lõpu aeg ning kui üks sprint on lõppenud, siis järgneb sellele kohe uus. Joonisel 6 lk 25 on kujutatud sprintide põhimõtet, kuidas üksteisele järgnevad sama pikad tegevuste aknad.



Joonis 6. Sprintide iseloomustus ja järgnevus. Autori koostatud Rubin [2] põhjal.

Scrum Guide järgi sprindi ajal [7]:

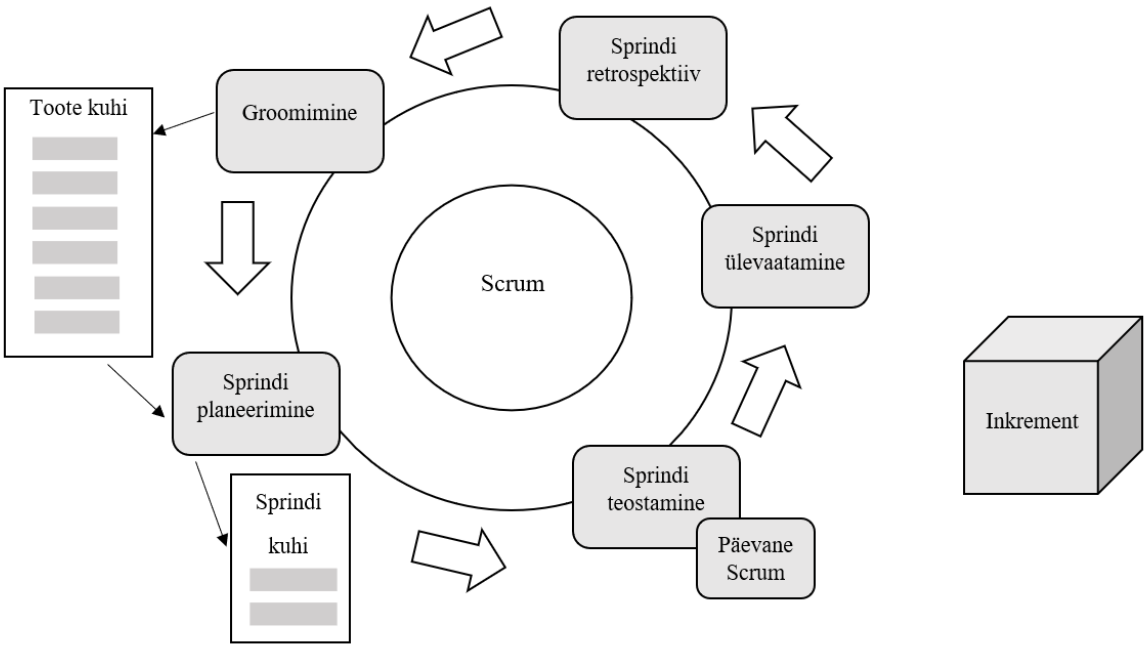
a) ei tehta muudatusi, mis võiksid ohtu seada sprindi eesmärgi saavutamise;

b) kvaliteet ei lange;

c) toote kuhi täpsustub vastavalt vajadustele;

d) skoopi võidakse toote omanikuga koostöös selgitada ja uuesti läbi rääkida, kui vahepeal on selgunud uut infot.

Reeglina ei ole sprindi ajal erinevad muudatused nii eesmärgi kui ka personali osas lubatud, kuid igas olukorras tuleb lähtuda sellest, et ärilised vajadused ei kannataks ja seega ei ole selle reegli järgimine alati 100 % võimalik [2].



Joonis 7. Scrum'i raamistiku tegevused ja artefaktid. Autori koostatud.

Joonisel 7 lk 25 on kujutatud kogu sprindi protsess, mille planeerimine algab toote kuhjast, kust valitakse välja need ülesanded, mida soovitakse järgmise sprindi jooksul valmis arendada. Valikud toimuvad tavaliselt toote omaniku poolt määratud prioriteetide alusel. Valitud ülesanded liigutatakse sprindi planeerimise käigus sprindi kuhja. Sprindi ajal toimub nende ülesannete arendamine ja implementeerimine ning sprindi ülevaatamine (*sprint review*), et oleks võimalik kiirelt ennetada ja lahendada tekkivaid probleeme ning seisakuid arendustegevustes. Arendusmeeskond viib igapäevaselt läbi lühikesi Scrum'i koosolekuid (*daily scrum*) kogu sprindi vältel. Kui sprint on lõppenud peaks valmis olema ka arendusülesanded sellises osas, et neid saab üle anda kliendile/kasutajale – toimub inkremendi versiooni üleandmine kliendile (*release, deployment*). Pärast sprindi lõppemist toimub retrospektiivi koosolek, kus vaadatakse tagasi lõppenud sprindile ning analüüsitakse nii hästi lahendatud ülesandeid kui ka tekkinud probleeme, mis võisid takistada meeskonnaliikmete tööd. Koosoleku käigus võib luua tegevuskava *(action items*), mis aitaksid vältida järgmiste sprintide korral samu probleeme või protsesse parandada.

Seega kokkuvõtlikult on sprindi planeerimiseks esimese asjana vaja prioritiseeritud toote kuhja.

3.4.2. Toote kuhi

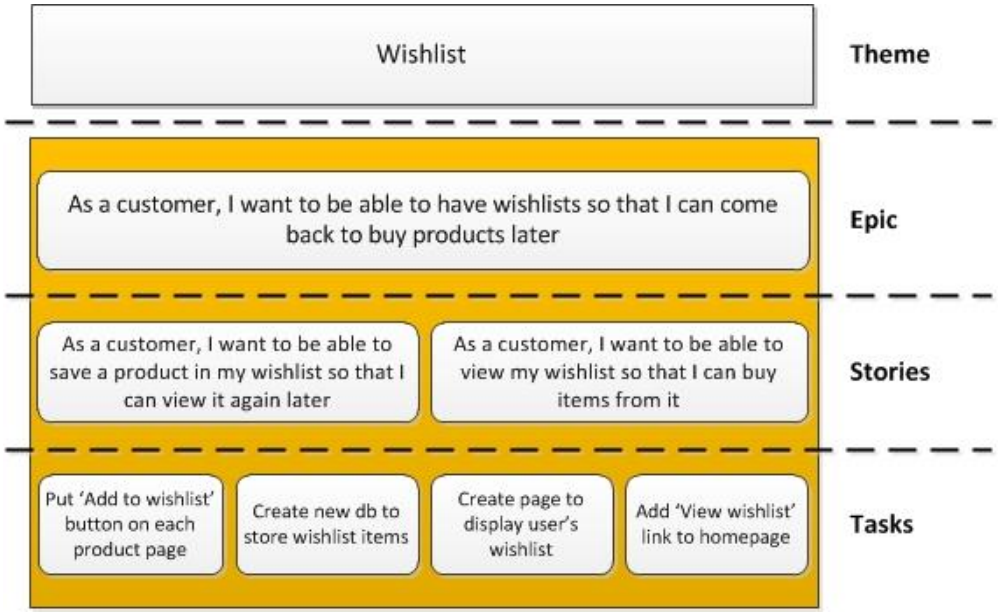
Toote kuhjas on ära kirjeldatud kogu toote funktsionaalsus, mis on jaotatud väiksemateks ülesanneteks ja on prioritiseeritud. Prioritiseerimisega tegeleb toote omanik. Täpsemalt on toote kuhjas reastatud klientidele ja kasutajatele mõeldud featuurid ja funktsionaalsused, mis tihtilugu on kirjeldatud kasutajalugudena (*user stories*) ning samuti parandamist vajavad defektid, tehnilised täiustused, uute teadmiste omandamised ja muud ülesanded, mida toote omanik peab tähtsaks [2].

Toote kuhjal on teatud omadused, mis seda lihtsast tegemist vajavate ülesannete loendist eristavad [30]:

a) toote kuhja lisatud kasutajalugu loob alati kliendile, ettevõtte omanikule ja lõppkasutajale ärilist või tehnilist väärtust – väärtuseta kasutajalood on ressursside raiskamine ja tuleks toote kuhjast eemaldada;
b) toote kuhjas olevad kasutajalood on prioritiseeritud ning järjestatud kõrgemast madalamani – tänu prioriteetide määramisele saab toote omanik otsustada, mida arendusmeeskond peaks järgnevalt arendama;

c) kasutajaloo üksikasjade tase sõltub selle positsioonist toote kuhjas – täpsemalt tuleks määratleda ainult need nõuded mis arendatakse järgmise paari sprindi jooksul;
d) kasutajalood on hinnatud – hinnangud mõjutavad hiljem sprindi planeerimise protsessi, sprindi kuhja ja toodete väljaandmise plaane;
e) toote kuhi on nö elav dokument, kuna see muutub ja areneb pidevalt – vajadusel lisatakse toote kuhja uusi kasutajalugusid, võidakse muuta ja täpsemalt määratleda olemasolevaid kasutajalugusid või neid isegi kustutada;
f) toote kuhjas ei ole tegevuspunkte (*action item*) ega madala tasemega ülesandeid.

Et mõista, millised on seosed ülesannete, kasutajalugude ja *epic*'ute vahel ning mida iga mõiste tähendab, on joonisel 8 toodud näide.



Joonis 8. Seosed *epic*'u, kasutajaloo ja ülesannete vahel [23].

Teema (*theme*) võtab kokku üleüldise arendust vajava valdkonna, milleks antud joonisel on „*Wishlist*" ehk sooviloend. Teema sünonüümina võib kasutada vihmavarju mõistet (*umbrella feature*), mis koondab enda alla konkreetse featuuri ülesanded ja kasutuslood. Selle alla on tehtud *epic*, mis täpsustab kliendi vajadusi antud arendusvaldkonnas – kliendina soovin ma luua sooviloendeid, et saaksin hiljem tagasi tulla ja endale meeldinud tooteid osta. Kuna *epic* on ka üsna üldine ja ei anna täit ülevaadet, mida täpselt peaks arendama, luuakse selle alla omakorda kasutajalood (*stories*), mis täpsustavad, mida tahab klient sooviloendi puhul teha. Kasutajalugu saab vaadata juba konkreetse nõudena, mida klient soovib ning mida oleks võimalik sprindi jooksul arendada. Antud joonisel on üheks kasutajaloo nõudeks – kliendina tahan, et saaksin oma sooviloendit salvestada, et saaksin seda hiljem uuesti vaadata.

Arendusülesandena saab siit välja lugeda sooviloendi salvestamise võimalust, kuid seda saab omakorda täpsustada pisemate ja konkreetsemate ülesannetega (*task*). Näiteks tuleks lisada kõigepealt nupp „Lisa sooviloendisse“, et klient saaks tooteid oma loendisse salvestada. Selleks tuleb teha ka andmebaasis andmetabelite muudatusi, et oleks olemas salvestatud loend ning selleks on ülesanne – loo uus andmetabel, et talletada klientide sooviloendi üksused.

Seega kokkuvõtlikult võib kuuluda ühe teema ja *epic*'u alla üsna palju kasutajalugusid, mis võtavad kokku kliendi vajadused ning iga kasutajalugu võib olla jagatud väiksemateks arendusülesanneteks.

Kui Scrum'i meeskonnas on loodud suur hulk üksusi, siis paratamatult tuleb suuta neid hallata ning otsustada, kas hoida neid kõiki ühes üldises toote kuhjas või luua mitu väiksemat toote kuhja. Rubin [2] kasutab selleks „üks toode, üks toote kuhi“ reeglit, täpsustades, et igal tootel peaks olema oma toote kuhi, mis võimaldab kirjeldada kogu toodet ja arendada seda prioriteetide järjekorras. Samas peab Rubini sõnul antud reegli rakendamisel olema ettevaatlik, et ei kaoks praktiline ja toimiv toote kuhja struktuur ning kasutataks enda vajadustele vastavat süsteemi.

Mike Cohn ja Roman Pichler on hea toote kuhja iseloomustamiseks kasutusele võtnud akronüümi DEEP, kus iga tähe taga peitub oma mõte [8]:

a) ***D****etailed appropriately* ehk üksikasjalikult defineeritud - need toote kuhja kasutajalood, mida esmajärjekorras arendama hakatakse peavad olema piisavalt detailsed ja mõistetavad, et need saaks järgmisel sprindil lõpuni viia. Rubin [2] lisab, et need kasutuslood, millega saab hakata järgmises sprindis tööle, peavad olema väikesed ja detailselt kirjeldatud, toote kuhja lõpuosas olevad kasutuslood võivad olla suuremad ja üldisemalt kirjeldatud ning kui jõutakse lähemale nende töösse võtmisele, siis jagatakse see väiksemateks ja detailsemateks ülesanneteks;

b) ***E****mergent* ehk arenev - toote kuhi ei ole staatiline, vaid on ajas pidevas arengus ja muutumises. Kui aja jooksul kerkib esile uusi detaile, siis toote kuhja kasutuslugusid lisatakse, eemaldatakse ja prioritiseeritakse ümber;

c) ***E****stimated* ehk hinnanguline - toote kuhi on rohkem kui ainult nimekiri tööülesannetest, see on ka kasulik planeerimisvahend. Toote kuhja alumises otsas asuvad ülesanded ei pruugi olla veel täiesti selged, seega ei pruugi neile antud hinnangud olla ka nii täpsed, kui need hinnangud, mis on antud toote kuhja esimese otsa ülesannetele. Rubin [2] täiendab seda mõtet sellega, et toote omanik saab igale ülesandele antud hinnangut

kasutada ka sisendina, et leida iga ülesande prioriteet toote kuhjas - enamike ülesannete suurus hinnangud on antud kasutusloo punktidena (*story points*) või ideaalsete päevade arvuga (*ideal days*);

d) ***P**rioritized* ehk prioritiseeritud - toote kuhi peaks olema sorteeritud nii, et kõige suuremat väärtust loovad ülesanded oleksid esimeste seas ja kõige vähem väärtust loovad ülesanded toote kuhja lõpuosas. Töötades alati prioriteetide järjekorras suudab meeskond maksimeerida arendatava toote või süsteemi väärtust. Lisaks on Rubin [2] öelnud, et kõiki ülesandeid ei ole mõistlik prioriteetide järjekorda panna, kuna toote kuhja lõpuosa ülesandeid on raskem hinnata väikse detailsuse astme tõttu ja seega oleks kohustuslikus korras prioritiseerimine pigem aja raiskamine.

Et hallata toote kuhja ja seal olevaid ülesandeid vastavalt DEEP meetodile tuleb seda pidevalt korrastada. Neid tegevusi tehakse toote kuhja täiustamise, rafineerimise ja hindamise käigus.

3.4.3. Toote kuhja hindamine

Toote kuhja hindamine ehk groomimine hõlmab endas kolme põhitegevust - toote kuhja ülesannete loomist ja täpsustamist ning üksikasjade lisamist, toote kuhja ülesannetele hinnangute andmist ja toote kuhja ülesannete prioritiseerimist [2]. Kõik need tegevused muudavad toote kuhja, kuna võidakse lisada uusi väiksemaid ja kõrgema prioriteediga ülesandeid, jagada olemasolevaid suuremad ülesanded detailsemateks osadeks või koguni võidakse mõned ülesanded toote kuhjast eemaldada.

Groomingud toimuvad regulaarselt, neid juhib toote omanik ja sellest võtavad osa erinevad huvigrupid, Scrum Master ja arendusmeeskond. Groomingud toimuvad kõikide osapoolte koostöös, kuid lõplik otsustaja on toote omanik [2]. Rubin lisab [2], et head toote omanikud mõistavad, et koostöö soodustab olulisi dialooge erinevate osapoolte vahel ning aitab kasutada nö mitmekesise isikute grupi kollektiivset intelligentsust ja väljavaateid, mis omakorda võivad esile tuua olulist teavet, mis muidu võiks kahe silma vahele jääda. Lisaks annab selline ühine arutlemine iga konkreetse ülesande juures kõigile osapooltele ka selgema pildi, mida tuleb teha ja milline on oodatav tulemus ning seeläbi vähendab tekkida võivaid arusaamatusi töö käigus.

Scrum'i raamistikus ei ole ära määratud, kuna toote kuhja groomimised peaksid aset leidma, vaid on lihtsalt viidatud, et see on pidev protsess, mis oma olemuselt on planeerimata tegevus väljaspool esmaste tegevuste voogu [2]. Rubin on välja toonud erinevaid olukordi ja osapooli, kuna ja kellega võidakse groomida toote kuhja [2]:

a) esmane grooming võib toimuda ühe osana toote avalikustamise (*release planning*) planeerimise ajal;

b) toote arendamise ajal võib toote omanik erinevate huvigruppidega groominguid teha vabalt valitud sagedusega;

c) arendusmeeskonnaga töötades võib toote omanik planeerida regulaarsed koosolekud (näiteks üks kord nädalas, üks kord sprindi jooksul) toote kuhja täiustamiseks, mis tagab korrapärase ajakava ja võimaldab meeskonnal sprindi planeerimisel sellega arvestada;

d) mõnikord eelistavad meeskonnad hajutada groomingu tegevusi üle kogu sprindi, selle asemel, et teha seda konkreetses ajavahemikus ja võivad inkrementaalset groomingut teha ka iga päev pärast oma igapäevase Scrum'i (*daily scrum*) tegevusi;

e) mõned meeskonnad leiavad, et groomingut võiks teha ka ühe osana sprindi ülevaatamise (*sprint review*) ajal, kuna kõik asjaosalised saavad siis parema ülevaate sellest, kus etapis oma tootega ollakse ja kus soovitakse olla (lisades juurde toote kuhja ülesandeid või muutes juba olemasolevaid).

Et groomingu ajal antavaid hinnanguid kirja panna, on vaja ka kokkulepitud ühikuid, mida kasutada.

3.4.4. Hinnangu ühikud

Selleks, et hinnata ülesande raskust ja komplekssust kasutatakse kahte erinevat lähenemist: kasutajalugude punkte ehk *story point*'e ja ideaalse päeva mõõdet [2]. Mõlemaid lähenemisi mõõdetakse numbriliste ühikutena ja iga meeskond saab ise määratleda neid oma parima äranägemise põhjal - üks meeskond võib määratleda ülesandeid ideaalse tööpäeva ühikuna (päev ilma igasuguste katkestusteta, koosolekuteta, e-mailidele vastamisteta jne), samas teine meeskond võib määratleda seda ideaalse töönädala ühikuna ning kolmas meeskond võib *story point*'i ühikuna määratleda kombinatsiooni kasutajaloo keerukusest ja suurusest [11]. Cohn [13] eelistab ülesande hindamiseks kasutada ideaalse tööpäeva mõõdet, mis annab numbrilise hinnangu, kui mitu päeva mõne ülesande tegemiseks kulub. Tema sõnul on ülesannet ideaalse päevana lihtsam hinnata, kui kasutada kogu kuluva aja mõõdet, kuhu on ka sisse arvestatud erinevad koosolekud, vahele tulevad ülesanded, e-mailidele vastamised jne. Rubini [2] kogemuste põhjal kasutavad 70% ettevõtetest, kellega tema on töötanud, *story point*'e, mis võtavad arvesse nii ülesande keerukust kui ka suurust ning ühendavad need üheks suhtelise

suuruse mõõduks, mille abil ülesandeid hinnata ja omavahel võrrelda. Tema arvates on ideaalse aja mõõtme kasutamise puhul risk vääraks tõlgendamiseks ehk neid ideaalseid päevi hakatakse kaardistama kui kalendripäevi.

3.4.5. Planning Poker ehk varjatud ajahinnangute andmine

Varjatud ajahinnangute andmine on tehnika, mida kasutatakse ülesannete hindamisel ja suhtelise suuruse ühiku määramisel. Seda konsensusel põhinevat tehnikat tutvustas esmakordselt James Grenning [15] ja laiemalt on seda populariseerinud Mike Cohn [2]. Grenning [15] on antud tehnikaga lahendanud kaks probleemi, mis ülesannete hindamisel tihti ette tulevad - hinnangute andmine võtab väga kaua aega ja kogu meeskond ei pruugi hinnangu andmisel osaleda. Antud tehnika lahendab need probleemid järgnevalt [15]:

a) kõigepealt loetakse ette ülesanne/kasutajalugu kliendi poolt ning vajaduselt toimub arutelu, mis täpsustab seda ülesannet/kasutajalugu vastavalt vajadusele;

b) seejärel kirjutab iga arendaja ja/või meeskonnaliige oma prognoosi märkmekaardile ilma seda teistega arutamata;

c) kui kõik on oma hinnangu kirja pannud, pööratakse kaardid ümber ja hinnangud avalikustatakse;

d) kui hinnangud ühtivad ja ollakse konsensusel, siis antud ülesande/kasutajaloo hinnang registreeritakse ja liigutakse edasi järgmise ülesande/kasutajaloo hindamise juurde;

e) kui hinnangud ei ühtinud, saab meeskond arutada lahknevaid hinnanguid, avaldada arvamust oma hinnangu suhtes ja jõuda ühiselt konsensusele, milline võiks antud ülesande/kasutajaloo hinnang olla.

Et seda tehnikat kasutada, tuleb kokku leppida ka hindamisskaala, mida kasutatakse. Grenningi [15] loodud skaalat 1,2,3,5,7,10 ja ∞ (lõpmatus) saab kasutada nii *story point'*ide kui ka ideaalsete päevadena hindamisel ning see aitab hinnangu suuruse hoida alla 2 nädala ning kui ülesanne/kasutajalugu hinnatakse pikemaks, kasutatakse lõpmatuse kaarti. Kõige sagedamini kasutatakse Mike Cohni poolt välja pakutud skaalat [2], mis põhineb osaliselt modifitseeritud Fibonacci järjestusel 1,2,3,5,8,13,20,40,100 ning erisümbolitena kasutatakse samuti lõpmatust (tähistab väga suurt hinnangut, mida ei ole mõtet numbriliselt hinnata), ? (näitab, et meeskonna liige ei saa ülesandest aru ja palub toote omanikul anda täiendavaid selgitusi) ja π (pii - viide inglise keelsele väljendile *pie* = pirukas; kasutatakse siis, kui meeskonna liige on väsinud ja näljane ning vajab pausi). Alternatiivselt kasutatakse ka ruutu tõstetud skaalat 1,2,4,8,16,32.

Scrum'i meeskonnas teostatakse planeerimist sarnaselt Grenningi tutvustatud tehnikale [2] paari muudatusega:

a) toote omanik on see, kes valib ülesande ja loeb selle ette arendusmeeskonnale, vajadusel annab lisa selgitusi ning vastab arendusmeeskonna poolt esitatud küsimustele;

b) Scrum Master toetab meeskonda ning vajadusel aitab neid paremini kaasata hindamisprotsessi;

Antud protsessi kasutamine paneb kõiki meeskonnaliikmeid arutlustel mõttega kohal olema ja kaasa töötama, kuna igaüks peab oskama põhjendada ka oma hinnangut juhul, kui need ei ühti üldsuse poolt antud hinnangutega. Lisaks peaksid kõik tekkivad küsimused saama ka koheselt vastuse ning seeläbi on selgem visioon, mida ülesandes/kasutajaloos arendajalt oodatakse.

Microsoftis läbi viidud juhtumiuuringu [31] tulemustest nähtub, et kui tavaliselt on ülesannete hindamine projekti alguses üsna keeruline ja suure hindamisvea protsendiga, kuna esineb teadmatust, ebakindlust ja projekti edenedes ning uusi teadmisi saades muutuvad antud hinnangud täpsemaks, siis antud tehnikat kasutades on suudetud hindamisel tehtud vigade protsent hoida üsna madalal juba projekti algusest saati. Juhtumiuuringus osalenud meeskonnad põhjendasid seda põhjalikuma eeltööga, nagu näiteks funktsionaalsuse nõuete väiksemateks ja detailsemateks osadeks jagamisega, et neid oleks võimalik valmis teha ühe iteratsiooni jooksul.

Kokkuvõtlikult ei ole oluline, kuna toimub toote kuhja grooming vaid tähtis on, et ülesanded oleks piisavalt hästi kirjeldatud ja arusaadavad ning grooming toimuks hõlmates erinevaid osapooli ja huvigruppe, kes ülesannetele hinnanguid annavad. Lisaks peab nendel osapooltel olema selge visioon, mille alusel ülesandeid hinnatakse (ideaalne päev vs *story point*) ja milline on kasutatav hindamisskaala. Pärast hindamisi võib toote kuhjas lõpuks olla mitmete nädalate ja ka kuude jagu ülesandeid ning neid hakatakse prioriteetide järjekorras määrama eelseisvatesse sprintidesse. Neid tegevusi tehakse sprindi planeerimise käigus.

3.4.6. Sprindi planeerimine

Sprindi planeerimine hõlmab endas toote kuhjast ülesannete valimist, mida arendusmeeskond võiks järgmise sprindi käigus valmis teha ning kliendile üle anda. Sprindi planeerimise koosolek toimub perioodiliselt ning kestab ligikaudu 1 tund iga sprindi nädala kohta [11]. Koosolekul osalevad toote omanik, Scrum Master ja kogu Scrum'i arendusmeeskond ning

meeskonna kutsel võivad osaleda ka välised huvigrupid, ehkki see on üsna haruldane [10]. Planeerimise käigus jagab toote omanik sprindi esialgset eesmärki, tutvustab prioritiseeritud toote kuhja ja vastab küsimustele, mis meeskonnal võivad tekkida toote kuhja osas [2]. Arendusmeeskond seevastu töötab usinalt selle nimel, et pakkuda välja omapoolne realistlik nägemus sellest, mida nad on võimelised uue sprindi jooksul ära tegema [2]. Scrum Master'i roll on planeerimise koosolekul olla pigem *coach*, jälgida planeerimise tegevusi, küsida uurivaid küsimusi ning hõlbustada eduka tulemuse tagamist nii planeerimise tegevusel kui ka arendusmeeskonna välja pakutud plaani realistlikkuse tagamisel [2].

Scrum Guide [7] täpsustab, et planeerimine toimub kogu Scrum'i meeskonna koostöös ning toote omanik peab tagama, et kõik kohalviibijad on ette valmistunud ja valmis arutama, millised on toote kuhja kõige tähtsamad ülesanded ning kuidas need on seotud toote eesmärkidega. Lisaks peab toote omanik olema valmis selgitama meeskonnale kõige kõrgema prioriteediga ülesandeid ja kasutajalugusid ning heaks suuniseks loetakse seda, kui toote omanik on valmis tutvustama vähemalt kahe sprindi jagu prioriteete ja ülesandeid [10]. Siinkohal eeldab Huether [11], et toote omanik on planeerimise koosolekuks valmistunud, määrates ära kõige suurema väärtusega ülesanded ja töötab selle nimel, et need oleksid valmis meeskonnale tutvustamiseks. Lisaks on ta välja toonud punktid, mida võiks toote omanik iga ülesande juures enne planeerimise koosolekut üle vaadata [11]:

a) määrata kasutajaloo ligikaudne *story point* väärtus;

b) eemaldada sõltuvused teiste ülesannetega/kasutajalugudega;

c) luua testitavad näited;

d) defineerida vastuvõtukriteeriumid (*acceptance criteria*);

e) viia ülesanne vastavusse INVEST kriteeriumitega.

Lühendit INVEST kasutatakse kasutajalugude kvaliteedi hindamiseks [12] ning iga täht selles sõnas viitab ühele kriteeriumile. Mike Cohn [13] on pikemalt lahti seletanud Bill Wake [14] poolt 2003. aastal kirjeldatud tähed ja lühendid hea kasutusloo jaoks:

a) ***I**ndependent* ehk sõltumatu - tuleks vältida erinevate sõltuvuste tekitamisi erinevate kasutajalugude vahel. Kasutajalugude vahelised sõltuvused toovad endaga kaasa prioriteetide seadmise ja planeerimise probleeme ning lisaks on ka nende kasutajalugude hindamine raskem. Sõltuvuste korral on võimalik neid lahendada kahte

moodi - kas ühendada sõltuvad kasutajalood üheks suuremaks, kuid iseseisvaks kasutajalooks või leida kasutajalugude tükeldamiseks erinev viis/dimensioon.

b) ***N****egotiable* ehk läbiräägitav - kasutajalood ei ole kirjalikud lepingud, mida arendatav tarkvara peab rakendama, vaid need on pigem jutukaardid (*story cards*) funktsioonide lühikirjeldustega, mille üksikasjad tuleb läbi rääkida kliendi ja arendusmeeskonna vahel. Jutukaartide puhul on tegemist pigem meeldetuletusega kliendiga detaile arutada, mitte üksikasjalike nõuete kirjeldusega ning seega peaks jutukaart sisaldama ühte-kahte fraasi, mis tuletavad meelde vestluse teemat ning märkuseid vestluse käigus lahendatavate probleemide kohta.

c) ***V****aluable* ehk väärtuslik kliendile ja kasutajatele - igaüks hindab kasutajalugu enda jaoks erinevalt, seega peab hea kasutajalugu olema väärtuslik nii kliendile kui ka kasutajale, mitte ainult arendusmeeskonnale. Arendusmeeskonnale kirjutatud kasutajalood keskenduvad tihti tehnoloogilistele aspektidele, mis ei pruugi olla kasulikud ja arusaadavad klienditele ja kasutajatele, et nemad omakorda oskaksid prioritiseerida neid kasutuslugusid oma areduskavasse. Ehk selle asemel, et kirjutada "Kogu veakäsitlus toimub ühiste klasside komplekti kaudu" võiks parem kasutajalugu olla "Kõik vead esitatakse kasutajale järjepidevalt". Parim viis tagamaks, et iga kasutajalugu oleks kliendile või kasutajale väärtuslik, oleks lasta kasutuslugu kliendil endal kirjutada.

d) ***E****stimable* ehk hinnatav - on oluline, et arendusmeeskond oskaks hinnata kasutajaloo suurust ja aega, mis kulub selle kasutajaloo muutmiseks töötavaks inkremendiks. See eeldab, et arendusmeeskonnal on selleks vajalikud teadmised (nii valdkonna- kui ka tehnilised teadmised) ning kasutajalugu ei ole liiga suur. Suured kasutajalood võib kirja panna ühe *epic*'u alla, mis on kohatäide (*placeholder*) või meeldetuletus süsteemi suurte osade kohta, mida tuleks kliendiga arutada.

e) ***S****mall* ehk väike - kasutajaloo suurus on oluline, sest kui kasutajalugu on kas liiga suur või liiga väike, siis ei saa neid planeerimisel kasutada. Samamoodi on raske töötada *epic*'utega, kuna need võivad sageli sisaldada mitut kasutajalugu. Kasutajalugude suuruse muutmiseks võiks liiga väikesed kasutajalood liita kokku üheks suuremaks kasutajalooks ning vastupidi *epic*'u puhul võiks kasutajalood jagada väiksemateks osadeks.

f) ***T**estable* ehk testitav - kasutajalood tuleb kirja panna nii, et need oleksid ka testitavad. Testide edukas läbimine tõestab, et arendusmeeskond on kasutuajaloo edukalt valmis saanud. Võimaluse korral peaksid testid olema automatiseeritud.

Seega peaks toote omanik enne planeerimist kindlaks tegema, et planeeritavad ülesanded looks väärtust, neil ei oleks sõltuvusi mõne teise ülesandega, need oleks piisavalt väikesed ja hinnatavad ning neid oleks võimalik testida vastavalt vastuvõtukriteeriumitele. Kui mõni ülesanne on liiga suur, nii et seda ei ole võimalik ühe sprindi jooksul valmis saada, peaks selle planeerimisest kõrvale jätma ja/või selle väiksemateks osadeks tegema.

Põhilised 3 teemat, mida planeerimise käigus arutatakse on [7]:

1. Miks see sprint on väärtuslik

Toote omanik teeb ettepanekud, kuidas toode saaks järgnevas sprindis väärtust ja kasulikkust suurendada. Seejärel teeb kogu Scrum'i meeskond koostööd sprindi eesmärgi määratlemiseks. Sprindi eesmärk on lühike, 1-2 lause pikkune kirjeldus sellest, mida meeskond soovib järgmise sprindiga saavutada [10]. Tänu sellele on võimalik anda kiire ülevaade erinevatele huvigruppidele, mis väärtust antud sprint loob. Sprindile seatud eesmärk tuleb lõplikult vormistada enne sprindi planeerimise lõppu.

2. Mida selles sprindis on võimalik ära teha

Aruteludes toote omaniku ja arendusmeeskonna vahel valitakse toote kuhjast ülesanded, mida kaasata eelseisvasse sprinti. Scrum'i meeskond võib neid ülesandeid selle protsessi juures veel ka täpsustada, mis omakorda suurendab arusaamist, mida selles ülesandes oodatakse. Sprindi jooksul võimalike tehtavate ülesannete mahu valimine võib olla keeruline protsess, kuid mida rohkem arendusmeeskond võtab arvesse varasemaid kogemusi ja suutlikkust eelnevates sprintides, seda lihtsam on neil hinnata oma võimekust järgnevaks sprindiks ja seda täpsemalt prognoosida. Selleks, et hinnata juba küpse meeskonna võimalikku panust sprindis, saab kasutada kombinatsiooni meeskonna liikmete saadavalolekust (*availability*) ja kiirusest (*velocity*), samas kui uued meeskonnad ei pruugi teada oma kiirust, peaks nende puhul prognoose tegema nende suutlikkuse (*capacity*) põhjal [11]. Meeskonna kiirust mõõdetakse üle pikema ajaperioodi (üle mitme sprindi) ning seda tehakse ülesannetele antud hinnangute (*story points*) ja vastavalt sprindi jooksul valmis saadud ülesannete hinnangute summa leidmisel ehk kui sprindis X tehti valmis 2 ülesannet, mille hinnangud olid vastavalt 5 ja 13 *story point*'i, siis loetakse meeskonna kiiruseks (*velocity*) 18. Sellest lähtuvalt saab ka uute sprinti planeerida

ülesandeid, mis summaarselt annavad kokku 18 (eeldusel, et kõik samad meeskonnaliikmed on olemas, kelle töö põhjal eelnev kiiruse ühik on saadud).

3. Kuidas valitud tööd saavad tehtud

Igale toote kuhjast valitud ülesandele kavandab arendusmeeskond tööülesanded, mis oleks vaja teha, et see inkrement saaks sprindi jooksul valmis. Seda tehakse sageli toote kuhja ülesande väiksemateks osadeks jagamise teel, mille tegemine võtaks arendusmeeskonnal aega kuni 1 päev. Kuidas see väiksemateks osadeks jagamine käib on arendusmeeskonna pädevuses. Rubin [2] toob välja kaks reeglit, mille alusel võiks toote kuhjast ülesandeid valida - esimene lähenemine oleks sprindi eesmärgist lähtuvalt ehk valitakse neid ülesandeid, mis on kooskõlas eesmärgiga ning teine lähenemine (kui puudub sprindi eesmärk) oleks valida ülesandeid toote kuhja ülaosast ehk kõige kõrgema prioriteediga ülesanded.

3.4.7. Sprindi kuhi

Sprindi kuhi on sarnane toote kuhjaga. Sprindi kuhja valitakse ülesanded toote kuhjast sprindi planeerimise käigus vastavalt ülesannete prioriteetidele ja groomingu käigus antud hinnangutele ning sprindi kuhjas olevad ülesanded tuleb valmis saada ühe sprindi jooksul. Rubin [2] lisab, et toote kuhjast ei tohiks sprindi kuhja valida selliseid ülesandeid, mida ei ole võimalik sprindi jooksul valmis saada ehk ei alustata seda, mida ei suudeta lõpule viia. Siinkohal jõuame uuesti INVEST kriteeriumi S täheni, mis tähistas väikest ülesannet ehk kui valitud ülesanne on liiga suur, aga seda soovitakse ikkagi sprinti võtta, tuleks proovida seda tükeldada väiksemateks ja iseseisvateks osadeks, mida on võimalik arendusmeeskonnal valmis saada.

Kui sprint on alanud, siis tohib ainult arendusmeeskond sprindi kuhja ülesandeid juurde lisada [13], kuna nemad teavad kõige paremini, millal nad on oma töödega planeeritud ajast ette jõudnud ja suudaksid mõne ülesande lisaks võtta.

3.4.8. Sprindi teostamine

Sprindi teostamine algab pärast sprindi planeerimist ja teostamiseks loetakse kogu tööd, mida Scrum'i meeskond planeeritud sprindi ajal teeb ning mille lõpuks peaks olema loodud väärtust vastavalt nendele inkrementidele, mis on valmis saadud. Siinkohal mängivad suurt rolli eelpool kirjeldatud arendusmeeskonna omadused, eelkõige olla iseorganiseeruvad ja omada piisavaid oskusi, et otsustada kuidas ja mis vahenditega täita sprindi jaoks seatud eesmärgid ja sprindi lõpuks valmis saada töötavad inkremendid. Scrum Master saab arendusmeeskonda vajadusel

juhendada ja abistada tekkivate takistuse eemaldamisel. Toote omanik peab sprindi teostamise ajal olema kättesaadav selgitavatele küsimustele vastamiseks, vahepealse töö ülevaatamiseks ja arendusmeeskonnale tagasiside andmiseks, sprindi eesmärgi korrigeerimise arutamiseks, kui muutuvad tingimused seda nõuavad ja veendumaks, et toote kuhja ülesannete vastuvõtukriteeriumid on täidetud [2].

Sprindi eesmärgi täitmiseks on arendusmeeskonna kohustus hallata töö voogu ning langetada otsuseid, kui mitut ülesannet peaks paralleelselt töös olema, millal peaks mõne konkreetse ülesandega alustama, milliseid töid tuleb teha ja kes seda tegema peaks [2]. Samuti ei tohiks siin unustada eelpool mainitud musketäride suhtumist ehk tööülesandeid jagatakse küll arendusmeeskonna sees, kuid lõppeesmärk peaks kõigil olema ühine - sprindi eesmärgi täitmine.

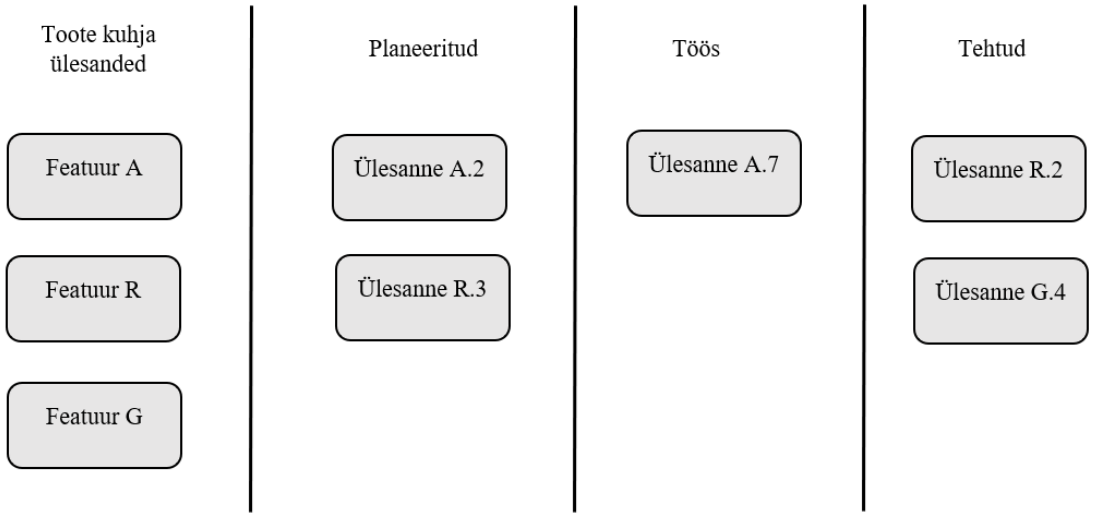
Selleks, et kõigil arendusmeeskonna liikmetel oleks ülevaade, kuidas eesmärgi poole liigutakse, on hea läbi viia lühikesi igapäeva koosolekuid ehk *daily scrum*'e. Neid koosolekuid peetakse tavaliselt võimalikult vara hommikul (kell 9:00 või 9:30), kuid siiski pärast seda, kui kogu meeskond on tööle saabunud ning osalema peavad kõik arendusmeeskonna liikmed, kaasa arvatud toote omanik ja Scrum Master [13]. Koosolekud hoitakse võimalikult lühikesed, tavaliselt 10-15 minutit, aga mitte rohkem kui 30 minutit ja selleks, et koosolekud ei veniks pikemaks, peetakse neid koosolekuid tihti ka seistes [13]. Sellest ka alternatiivne nimetus *stand-up meeting* ehk püstijala koosolek. Koosoleku ajal vastab iga meeskonnaliige kolmele küsimusele [13]:

1. Mida sa tegid eile?
2. Mis on sul täna plaanis teha?
3. Millised takistused segavad sul töö tegemist?

Sellistele küsimustele vastamine annab väga kiire ja lihtsa ülevaate iga meeskonnaliikme tööst ja plaanidest eesmärgi täitmiseks. Samuti saab tõstatada takistusi ja probleeme, mis töö tegemist takistavad. Kuna Scrum Master on ka nendel koosolekutel kohal, saab ta võtta siin juhtiva rolli nende takistuste eemaldamiseks.

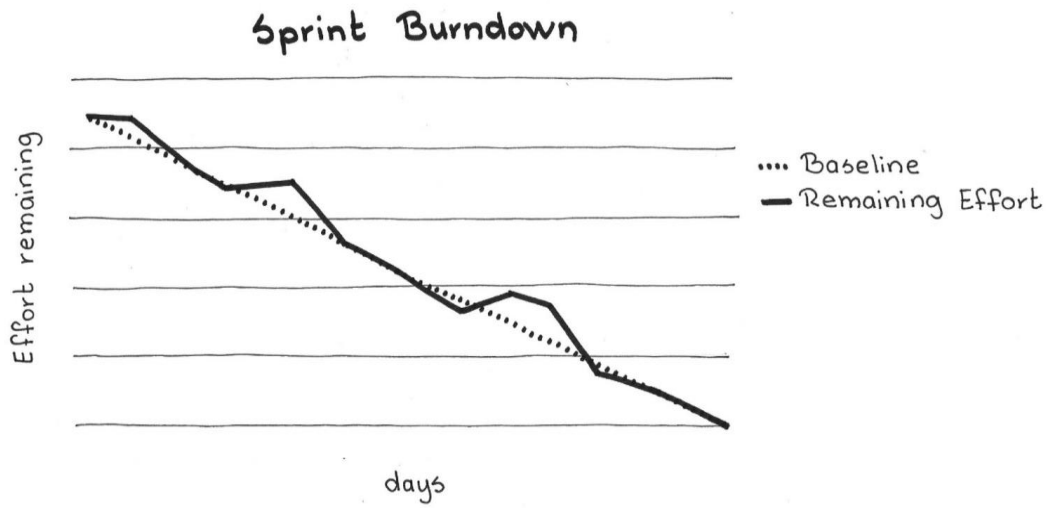
Lisaks igapäevastele koosolekutele peaks sprindi progressi kohta omama ka visuaalset ülevaadet [2]. Selleks võib kasutada erinevaid graafikuid ja ülesannete tahvleid (*task board*) nagu on toodud joonisel 9 lk 38. Visuaalselt annab selline tahvel kiire ülevaate, mis seisus ülesanded on ja millega oleks veel vaja tegeleda. Kõik sprinti planeeritud ülesanded on esialgu tegemist vajavate ülesannete tulbas ning ülesande seisu muudab arendusmeeskonna liige, kes

sellega tegelema hakkab. Ehk kui ta alustab tööd ülesandega, siis liigutab ta selle "Töös" tulpa ja kui ülesanne vastab vastuvõtukriteeriumitele, siis liigutab ta selle vastavalt "Tehtud" tulpa. Iga arendusmeeskond võib antud tahvlit luua vastavalt oma vajadustele ning detailsuse astmele.



Joonis 9. Näidis ülesannete tahvel. Autori koostatud.

Lisaks kasutatakse tihti sprindi progressi visualiseerimiseks ka töö edenemise ehk *burndown* graafikut [2], mis eeldab, et sprindi teostamise ajal värskendavad arendusmeeskonna liikmed iga päev hinnangut selle kohta, millises mahus on jäänud ülesannet veel täita, enne kui selle lõpetada saab. Kui ülesanne on lõpetatud, väheneb ka kogu hinnangu maht selle võrra. Joonisel 10 on toodud näide antud graafikust.



Joonis 10. Sprindi töö edenemise graafik [16].

Iga sprindi alguses on teada, milline on planeeritud ülesannete kogu maht, mida kujutatakse vertikaalsel teljel (*effort remaining*)ja horisontaalsel teljel on sprindi pikkus päevades (*days*). Kuna töö hulk peaks pidevalt vähenema, siis ideaalsel juhul peaks saama sprindi viimase päeva lõppedes ka kõik ülesanded valmis. Seda näitab joonisel katkendlik trendi joon (*baseline*) ning sellele lisatakse reaalne sprindi hetkeseis, mis näitab, millises tempos on saadud valmis ülesandeid (allapoole liikumine) ning kas on tekkinud seisakuid (horisontaalne liikumine) või kõrvalekaldeid uute ülesannete lisamise näol (tagasi ülespoole liikumine). Analoogselt kasutatakse ka sprindi *burnup* graafikut, kus eesmärk on vertikaalsel teljel ja tehtud tööde graafiku joon liigub ülespoole eesmärgi suunas.

Sprindi lõppemise lähedal viib meeskond läbi kaks olulist nö kontrolli ja kohanda tegevust - sprindi ülevaate (keskendub tootele) ja sprindi retrospektiivi (keskendub kasutatavale protsessile) [2]. Sprindi teostamise lõppedes peaks olema täidetud sprindi eesmärk.

3.4.9. Sprindi ülevaade

Sprindi ülevaade toimub iga sprindi tsükli lõpu lähedal ja tavaliselt enne sprindi retrospektiivi nagu on kujutatud joonisel 7 lk 25. Seda peetakse üheks olulisemaks õppimiskohaks Scrum'i raamistikus, kuna see annab kõigile arenduse osapooltele võimaluse kontrollida ja kohandada seni valmis tehtud toodet ning teha märkuseid, ettepanekuid ja diskuteerida, kuidas toote arendusega edasi liikuda [2]. Sprindi ülevaate koosolekul võivad osaleda lisaks toote omanikule, Scrum Master'ile ja arendusmeeskonnale ka kõik teised huvitatud osapooled (näiteks juhtkond, kliendid) [2]. Sprindi ülevaate koosolek on informaalne ning tavaliselt on selle koosoleku üheks osaks uute funktsioonide näitamine ehk demo [13]. See tähendab, et arendusmeeskonna liikmed tutvustavad sprindi jooksul tehtud funktsionaalsust ning näitavad, kuidas see ka reaalselt töötab. Rubini [2] sõnul ei ole sprindi ülevaade pelgalt sprindi demo, milleks seda koosolekut sageli sildistatakse, vaid kõige olulisem aspekt on sügavuti minev arutelu ja koostöö osalejate vahel. Samas annab arendusmeeskonna poolt demonstreeritud lahendus hea pinna küsimuste küsimiseks, täpsustuste tegemiseks ja pikemaks aruteluks.

Eeltöö sprindi ülevaate koosolekuks hõlmab endas õigete inimeste kutsumist, koosoleku aja kirja panemist, kinnitust, et sprinti planeeritud tööd on valmis ning demonstratsiooniks valmistumist ja kindlaks määramist, kes mida esitleb [2]. Õigete inimeste kutsumine annab kõige parema otsese tagasiside tehtud tööle ja uutele ettepanekutele. Sprindi ülevaate ajal demonstreeritakse ainult neid ülesandeid, mis on saanud valmis ehk mis vastavad varem kirjeldatud vastuvõtukriteeriumitele. Rubini [2] arvates peaks lõppkokkuvõttes olema toote

omanik see, kelle kohustus on enne sprindi ülevaadet õigel ajal kinnitada, kas ülesanne vastab nendele kriteeriumitele ja kas see funktsionaalsus on valmis demonstreerimiseks.

Radigan [17] lisab, et funktsionaalsuse demonstreerimine annab suurepärase võimaluse meeskonna saavutuse tähistamiseks ning ühise meeskonna tunde suurendamiseks. Positiivsust ja tunnustust lisab Gonçalvesi [18] sõnul ka toitude ja jookide pakkumine nendel koosolekutel, mille eest peaks hoolitsema Scrum Master. Selline korraldus on eelkõige mõeldav pikemate koosolekute puhul. Schwaber ja Sutherland [7] on Scrum'i raamistikus märkinud sprindi ülevaate koosoleku maksimum ajaks 4 tundi, kui tegemist on ühe kuu pikkuse sprindiga. Vastavalt sellele, kui lühike on sprint, muutub ka sprindi ülevaate koosoleku pikkus.

3.4.10. Sprindi retrospektiiv

Kui sprindi ülevaade keskendus tootele, siis pärast ülevaadet toimuv retrospektiiv keskendub protsessidele, mille kaasabil toodet tehakse. Schwaber ja Sutherland [7] ütlevad, et retrospektiivi eesmärk on kavandada võimalusi ja tegevusi sprindi kvaliteedi ja tõhususe suurendamiseks. Kaasaegsete retrospektiivide tutvustajaks peetakse Norm Kerthi [19], kes on välja andnud ka käsiraamatu projekti retrospektiivi koosoleku läbiviimiseks. Oma käsiraamatus kirjeldab Kerth [19], et parim aeg retrospektiivi jaoks on üks kuni kolm nädalat peale projekti lõppu ning kindlasti enne uue projekti algust. Seega saab seda vastavalt laiendada ka Scrum'i raamistikku ning retrospektiivi läbi viia peale sprindi lõppu. Schwaber ja Sutherland [7] on Scrum'i raamistikus märkinud sprindi retrospektiivi koosoleku maksimum ajaks 3 tundi, kui tegemist on ühe kuu pikkuse sprindiga. Vastavalt sellele, kui lühike on sprint, muutub ka retrospektiivi koosoleku pikkus, kuid rusikareegliks loetakse 45 minutit iga sprindi nädala kohta [20].

Rubini [2] arvates on retrospektiiv üks olulisemaid ja samas kõige vähem hinnatumaid tegevusi Scrum'i raamistikus. Tema sõnul ei pea retrospektiiv olema üldse raske tseremoniaalne protsess, vaid võib ka olla lihtne meeskonnaliikmete kokkusaamine ja selliste küsimuste arutelu nagu [2]:

a) Mis toimis selles sprindis hästi ja mille tegemist me tahaksime jätkata?

b) Mis ei toiminud selles sprindis hästi ja mille tegemist me tahame lõpetada?

c) Mida me võiksime tegema hakata või mida me peaksime parandama/täiustama?

Kuna retrospektiivi koosolekul osaleb kogu Scrum'i meeskond [2] ehk toote omanik, Scrum Master ja kõik arendusmeeskonna liikmed (välised osapooled peaksid osalema vaid Scrum'i

meeskonna kutsel), siis antud küsimused annavad hea pinna ühiseks aruteluks ja analüüsiks. Rubin [2] lisab, et retrospektiiv seisneb eelkõige ikkagi Scrum'i meeskonna protsesside parendamises, mitte inimeste süüdistamises või noomides individuaalset käitumist. Analüüsi tulemina saab kirja panna ka ühiselt kokku lepitud punktid (*action items*), mis tegevusi ja kelle poolt tuleks teha, et protsesse parendada ja uutes sprintides mitte enam vanu vigu korrata. Samuti peaks retrospektiivi ajal võtma aega ka eelmise retrospektiivi ajal kirja pandud parendustegevuste ülevaatamiseks, kas need on saanud täidetud [2].

Kokkuvõtlikult annab retrospektiiv ülevaate Scrum'i meeskonnale, kui hästi on suudetud Scrum'i raamistikku oma töös kasutada, kui hästi täidab iga meeskonna liige oma rolli ja kohustusi ning mis osas tuleks protsesse parendada.

4. Scrum’i raamistiku praktikas kasutamine

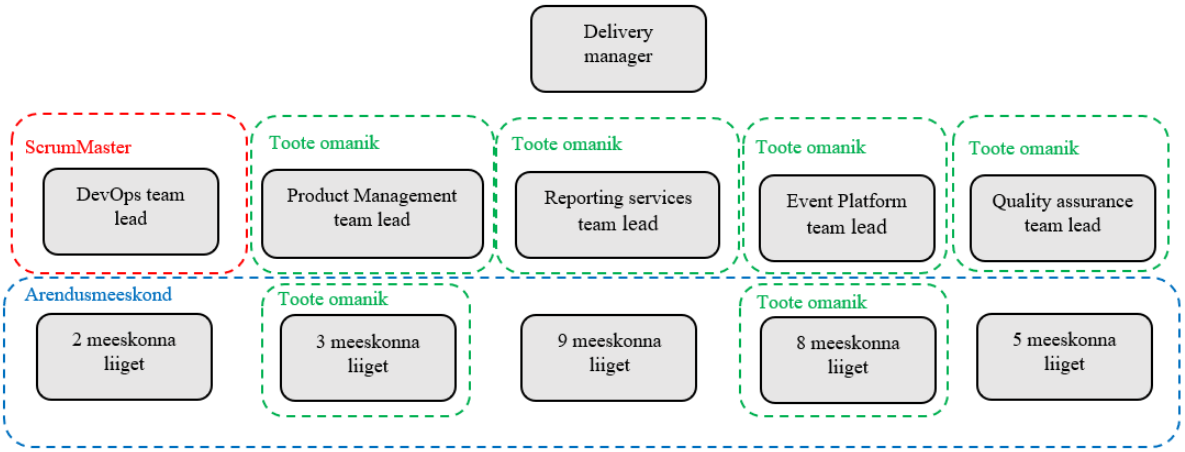
Playtech Estonia OÜ kuulub ühtsesse Playtech gruppi ja loob mängutarkvara, platvormi ja terviklahendusi maailma suurimatele mänguoperaatoritele. Playtechis töötab üle 6000 inimese 19 asukohas üle maailma. Playtechile pandi alus 1999. aastal Tartus ning kuni 2006. aastani oli Eesti üksus ettevõtte ainuke arenduskeskus, seega on Playtechi edulugu alguse saanud just Eestist. Playtechi Eesti üksus on tänaseni grupi üks suurimaid ja olulisemaid arenduskeskusi ning Tartu ja Tallinna kontorisse on koondunud grupi arendus-, tootmis- ja teenindusüksused. Üks Playtechi äriüksustest on IMS (Information Management System), mille alla kuulub ka *Data Services and Reporting* (DSR) osakond. [21]

Autor osales kõikides Scrum’i raamistikus märgitud tegevuste koosolekutel, mida antud osakonnas rakendatakse ning analüüsis dokumentatsiooni, mida osakonnas Scrum’i kasutuselevõtu käigus on talletatud, et saada ülevaade raamistiku rakendamise hetkeseisust ja tehtud tegevustest.

Järgnevates peatükkides antakse lühiülevaade DSR osakonnast ning pikem ülevaade Scrum’i raamistiku kasutamisest ja oma tööks kohandamisest DSR osakonna näitel.

4.1. Osakonna ülevaade

DSR osakond on üks väike osa kogu IMS osakonnast, kes tegelevad igapäevaselt Playtechi poolt pakutava platvormi arendamisega. Igapäevane töö toimub inglise keeles ning kasutatakse inglise keelseid termineid ja ametinimetusi. Ülevalt alla loetaval hierarhilisel joonisel 11 on välja toodud DSR osakonna alluvused, meeskonnad ning juurde lisatud Scrum’i rollid.



Joonis 11. Ülevaade DSR osakonnast ja rollidest. Autori koostatud.

Osakonda juhib *delivery manager*, kelle otseses alluvuses töötab viis meeskonna juhti (*team lead*) ning igal meeskonna juhile allub oma meeskond (joonisel kuvatud meeskonna juhi all tema meeskonna liikmete arv). Lisaks on joonisele 11 märgitud Scrum'i meeskonna rollid – punasega märgitud Scrum Master'i rolli kannab DevOps meeskonna juht; rohelisega märgitud toote omaniku rolli kannab *product management* meeskonna juht, *reporting services* meeskonna juht, *event platform* meeskonna juht, *quality assurance* meeskonna juht ning lisaks kaks meeskonna liiget; sinisega märgitud arendusmeeskonnaks loetakse kõiki meeskonna liikmeid.

Meeskonnad jagunevad vastavalt tööülesannetele analüüsiga seotud meeskonnaks (*product management*), arendusega seotud meeskonnaks (*reporting services, event platform*), testimisega seotud meeskonnaks (*quality assurance*) ja probleemide lahendamisega seotud meeskonnaks (*DevOps*). Kokku on igapäevaselt osakonnas 33 inimest, suveperioodidel on lisaks ka praktikandid ning osakonna liikmete arv 2021. aasta juulikuu seisuga on 39. Meeskonnad jagunevad Tartu ja Tallinna kontori vahel. E*vent platform* meeskond töötab täies osas Tallinna kontoris, mõne meeskonna puhul on vaid osad liikmed tööl erinevate linnade kontorites. Aja jooksul on meeskondade siseselt liikmetele lisandunud ka toote omanike rolle lisaks põhitööle meeskonnas.

Scrum'i raamistiku kasutamise poole on liigutud järk-järgult mitme aasta vältel ning Scrum'i tegevusi ja artefakte on võetud kasutusele vastavalt vajaduse tekkimisele. Konkreetsemalt hakati keskenduma Scrum'i raamistiku kasutamisele 2019. aasta suvel, kui osakonnaga liitus uus kolleeg, kellest sai *DevOps* meeskonna juht (meeskonnas 2 liiget) ja edaspidi ka Scrum Master. Peamine väljakutse, millega Scrum Master kokku puutus oli DSR-i 5 meeskonna töö planeerimine kõige tõhusamal viisil ning kuidas luua nähtavust ja läbipaistvust meeskondade tulemustest projektijuhtimise meeskonnale [27]. Peamiseks murekohaks Scrum Master'i jaoks oli planeerimise protsessi parendamine, kuna tihtilugu puudus ülevaade, kui palju meeskonnaliikmeid on uue sprindi ajal tööl, sprinti planeeriti rohkem tööd, kui jõuti valmis teha, paljud ülesanded lükati edasi järgmistesse sprintidesse ning seetõttu tuli plaane muuta ka projektijuhtimise meeskonnal. Kõik see kokku viis Scrum Master'i otsima lahendust, mida hakati nägema Scrum'i raamistiku kasutuselevõtus.

Scrum'i ei olnud võimalik kasutusele võtta 100% nii nagu teoorias on kirjutatud, kuna teooria näeb ette ühte kuni 9 liikmelist arendusmeeskonda, aga DSR-i osakond oli sellel hetkel juba 3-4 korda suurem, seega tuli hakata raamistikku kohandama vastavalt enda vajadustele. DSR-i meeskonnad on omavahelises sõltuvuses ning meeskonna arvu vähendamine või väiksemateks

osadeks jaotamine ei ole võimalik, kuna kõik meeskonnad sõltuvad testijate meesonna (*quality assurance*) olemasolust, kes on põhiline filter tarkvara toote kvaliteedi tagamisel ning kõikide arendusmeeskondade (*reporting services, event platform*) tehtud arenduste testimisel.

Igapäevaseks projektide jälgimiseks, erinevate töölaudade loomiseks, dokumentatsiooni haldamiseks ja muuks kasutatakse erinevaid Atlassiani tooteid, eelkõige Confluence ja Jira. Jiras luuakse kõik projektid, *epic*'ud, kasutajalood, ülesanded, lisaks hallatakse toote ja sprindi kuhja, luuakse töölauad ja salvestatakse groomingul antud hinnangud. Confluence kasutatakse eelkõige analüüside ja projektide dokumentatsiooni haldamiseks. Igal meeskonnal on ka omad töövahendid, kuid Scrum'i raamistiku järgimiseks ja elluviimiseks on need kaks põhilisemad.

Jira on võimekas tööriist, mis annab laia valiku funktsionaalsusi Scrum'i raamistiku rakendamiseks. Eelkõige aja planeerimiseks ning ülesannete hinnangute märkimiseks võeti kasutusele lahtrid *report notes* ja *story points*, millest on pikemalt juttu peatükis " Varjatud ajahinnangute andmine DSR-is" ning mille abil on võimalik luua erinevaid diagramme ja tahvleid (*dashboard*) sprindi planeerimise ja jooksva sprindi seisu jälgimiseks. Autor lisab, et edaspidi aitab taolise nähtava info ja statistika kogumine kindlasti kaasa ka planeerimise protsesside parendamisele, kui nähakse sprinti planeeritud töö mahtu olemasoleva inimressursi puhul ning kui palju planeeritud tööst sai sellises koosseisus tehtud.

Sprindid DSR-i osakonnas on 2 nädala pikkused. Erandina tehakse pikemaid sprinte jõulude-aastavahetuse ja suvise kollektiivpuhkuse ajal, kui suur osa meeskonnast puhkab ning 2 nädala jooksul ei ole võimalik mahukaid tulemusi saavutada. Tabelis 1 on toodud 2 nädala pikkuse sprindi jooksul tehtavad Scrum'i tegevused.

Tabel 1. Ühe sprindi kestuse ajal tehtavad tegevused.

Sprindi nädal	**Esmaspäev**	**Teisipäev**	**Kolmapäev**	**Neljapäev**	**Reede**
1		Eelmise sprindi retrospektiiv Sprindi demo*	Sprindi seis #1		
2	Sprindi seis #2	Toote X grooming	Toote Y grooming	Sprindi seis #3 Tartu grooming	Sprindi planeerimine

Groomingu koosolekuid on mitu, kuna 2 neist on toote spetsiifilised ja kolmas koosolek on üldine DSR-is planeeritud tootearenduste jaoks. Eelmise sprindi demo toimus lühikest aega uue sprindi alguses aga hetkel ei ole seda enam tehtud, kuna arendusmeeskonna liikmed tihtilugu kas ei soovinud avalikul koosolekul tehtud tööd tutvustada või tehtud tööd olid liiga

väikesed ja spetsiifilised, et seda oleks võimalik jagada nii, et ei näidataks vaid muudatusi koodi tasandil. Uusi arendusi ja funktsionaalsusi, mis väärivad laiemalt tutvustamist, lisatakse kogu IMS osakonna infokirja „*Release highlights*“, kuhu on lisatud kokkuvõte antud arendusest ning mida muudeti ja miks see oli vajalik.

4.2. Scrum’i rollid DSR-is

Scrum’i raamistikus välja toodud toote omaniku, Scrum Master’i ja arendusmeeskonna rollid on DSR osakonnas kõik esindatud.

Teooria ütleb, et Scrum Master’i roll ei pruugi olla täiskohaga töö ning lubab antud rolli puhul võtta lisaülesandeid. Nii on see ka DSR-i osakonnas, kus Scrum Master on ühtlasi ka DevOps meeskonna juht, tegeledes igapäevaselt ka antud meeskonna tööülesannetega. Autor näeb siin väikest vastuolu teoorias välja tooduga, kus soovitati lisaülesandeks pigem Scrum Master’i rolli laiendamist ja teadmiste jagamist teistes meeskondades, kui samas Scrum’i meeskonnas arendusmeeskonna liikme roll võtta, kuna see võib tekitada huvide konflikti kriitilistes olukordades. Samas on suudetud Scrum Master’i ja DevOps meeskonna juhi roll DSR-i osakonnas hoida heas harmoonias ning mõlema rolli ülesanded on saanud täidetud ning seega saab taolist teooriast erinevat lähenemist pidada edukaks.

Kindlasti on Scrum Master DSR-i osakonnas olnud teerajaja Scrum’i raamistiku suuremal kasutuselevõtul ning tutvustanud erinevaid Scrum’i tegevusi ja artefakte vastavalt Scrum’i raamistikule. Ta on ka muutuste elluviija, luues uusi tehnilisi lahendusi igapäevaste Scrum’i tegevuste lihtsustamiseks – näiteks sprindi planeerimise tegevuse jaoks loodud planeerimise tööriist (*planning tool*), millest tuleb juttu peatükis „Sprindi planeerimine DSR-is“. Samuti jälgib Scrum Master DSR-is käimasoleva sprindi kulgu ning korraldab selle jaoks iga 2 nädala pikkuse sprindi jooksul 3 jooksva seisu koosolekut (*status meeting*) nagu oli välja toodud tabelis 1 lk 44.

Toote omaniku rolli täitis algselt üks inimene, kelleks oli *product management* meeskonna juht. Ta oli eraldiseisev üksus DSR-i osakonnas, kuid kuna tema hallata oli lisaks ka osakonna analüütikute tööülesanded, siis lisati toote omaniku rollile ka meeskonna juhi roll ja koondati analüütikud ühte *product management* meeskonda. Vastavalt lisandunud rollile sai toote omanik suunata prioriteetsemad ülesanded õigel ajal analüüsi ning sealt edasi õiges järjekorras arendusse. Aja jooksul on toote omaniku rolli laiendatud ka teistele meeskonna juhtidele, lisaks mõnele arendusmeeskonna liikmele. Antud laiendus on tingitud DSR osakonna tooteportfelli suurusest ja eripäradest – algselt väikesed tooted on aja jooksul muutunud järjest mahukamaks

ning ühel peamisel toote omanikul on raske detailselt jälgida kõikide väikeste nüansside teostamist ja juhtida nende arengut. Teine põhjus toote omaniku rolli laiendada teistele osakonna töötajatele oli anda võimaluse võtta rohkem vastutust toote edenemise ja arengu eest. Selline käik andis võimaluse laiemalt mõjutada DSR-i osakonna tegevusi ning rikastada tööülesandeid. Lisaks loodetakse, et selline vastutuse ja võimaluse andmine tõstab omakorda töötajate rahulolu oma tööga ning suurendab osakonna jõudlust. Kuigi toote omanike rolle on jagatud osakonn mitmele liikmele, on vastutav tooteomanik jätkuvalt *product management team lead.*

Autori arvates on taoline rolli vastutuse jagamine antud suurusega arendusmeeskonna puhul õige käik, kuna iga väiksema toote omanik juhib selle toote arendusvajadusi ning seeläbi vähendab peamise toote omaniku töökohustusi, kuid samas hoiab teda kursis toote suuna käekäiguga. Lisaks oskab väiksema toote osa omanik näha oma toote tulevikusuundi ning neid vastutava toote omanikuga läbi rääkida ja vastavalt sellele arendusvajadusi registreerida, vajadusi põhjendada ja neile prioriteete saada. See tagab iga toote pideva arengu, millega ainult üks toote omanik ei suudaks alati 100% kursis olla, kui tema hallata on lai tooteportfell. Samas peab silmas pidama, et meeskonnas oleks üks toote omanik, kelle võimuses ja vastutusalas on lõplike otsuste tegemine.

Arendusmeeskonda kuuluvad nii analüütikud, arendajad kui ka testijad. Kuna platvormi arendus hõlmab mitmeid osapooli, osakondi ja meeskondi, siis kõiki vajalikke rolle konkreetses arendusmeeskonnas esindatud ei ole (näiteks arhitekt, disainer). Sellegipoolest on tegemist suuresti iseorganiseeruva meeskonnaga ning oskusi ja teadmisi on laialdaselt, et teha ära toote omaniku poolt prioritiseeritud ülesanded. Kui mõne ülesande puhul ei osata koheselt leida parimat lahenduse ideed, luuakse *research* ehk uurimisülesanded, mille puhul uuritakse ja kõrvutatakse erinevaid võimalikke teoreetilisi lahendusi. Kui uurimisülesanded on tehtud, siis järgmisena luuakse ülesanded POC-ide (*proof of concept*) tegemise jaoks, mis peaks andma reaalse pildi sellest, kuidas antud lahendus tehniliselt töötaks ja kas see kataks ära vajadused, mis arendusülesandes on ette nähtud. Autori arvates aitab selline uurimustöö tegemine ja erinevate võimaluste kaalumine kaasa arendusmeeskonna teadmiste laiendamisele ja süvendamisele ning koostööle, kus erinevad osapooled saavad välja käia ideid ja otsida ning läbi proovida erinevaid tehnilisi lahendusi arendusülesande lahendamiseks.

DSR-is on arendusmeeskond segu traditsioonilisest ja Scrum'i meeskonnast. Teooria põhjal tõi Rubin välja traditsioonilise meeskonna töövormi, kus ülesanne käis läbi erinevate rollispetsiifiliste meeskondade käest ehk kui analüütik lõpetab analüüsi, siis arendaja alustab

tööga ja kui arendaja lõpetab oma töö, siis testija alustab selle ülesande testimisega. Samas Scrum näeb ette kogu töö tegemist arendusmeeskonna sees. DSR osakond on vaikimisi jagatud erinevateks rollispetsiifilisteks meeskondadeks (analüütikud, arendajad, testijad), kuid kogu töö tehakse ühes Scrum'i arendusmeeskonnas ning ülesanne ei liigu tavaliselt väljapoole antud arendusmeeskonda, kuna vajalikud rollid on DSR-is sees esindatud. Erandiks on juhud, kui ülesande arendamise jaoks on vaja sisendit näiteks disainiga tegelevast meeskonnast, kes asub Kiievis või kui ülesanne on seotud ettevõtte siseselt erinevate meeskondadega. Sellisel juhul oodatakse, kui üks meeskond lõpetab oma osa ja annab teatepulga üle DSR-i arendusmeeskonnale.

Autori kogemuse põhjal on DSR-is olemas musketäride suhtumine ja ühiselt pingutatakse, et sprindi jooksul planeeritud ülesanded saaksid ka tehtud. Suhtumist, kus „Minu osa on tehtud, edasi vaadake ise" ei kohta ning erinevates rollides olevad meeskonnaliikmed on kättesaadavad küsimustele vastamiseks või tehtud lahenduste arutamiseks. Efektiivse suhtluse toimimiseks kasutatakse suhtlusrakendust MS Teams või Zoom ning kontoris viibides on võimalus ka näost näkku suhtluseks. Näost näkku suhtlus on kindlasti efektiivsem ja kiirem, kuid sellele on oma jälje jätnud 2020. aasta koroonapandeemia, kui arendusmeeskonnad hakkasid alates märtsist kaugtööd tegema kodukontoritest ning suhtlus muutus täielikult virtuaalseks. Üleminek nõudis harjumist ning õppimist ennast jagama korraga erinevate vestlusakende vahel ja oskust lülituda kiirelt ühest teemast teise.

Suhtluskanalite arv DSR-i meeskonnas on äärmiselt suur (39*(39-1)/2 = 741), kuna arendusmeeskonnas on kokku 39 liiget, kuigi teooria näeb parimaks liikmete arvuks umbes 5-9 liiget. Antud meeskonna puhul ei saa ka kasutada Jeff Bezose mõistet "kahe pitsa meeskond", kuna sellisel juhul saaks iga liige vaid imepisikese tüki pitsast. Nii suure liikmete ja suhtluskanalite arvuga meeskonna puhul tuleb ette probleemseid olukordi, kus vajalik info ei pruugi jõuda kõigi osapoolteni või mõned kokkulepped tehakse üksikute inimeste vahel ja ei ole kirjalikult talletatud. Kasutatakse küll üldiseid info jagamise vestlusi MS Teamsis, kuid siiski võib juhtuda, et mõni liige jääb sellest infost ilma. Seega ei ole antud arendusmeeskonna suurus just kõige mõistlikum ja nõuaks efektiivsema kommunikatsiooni nimel tööd.

Autori arvates on selline suhtluskanalite arv liiga suur ning vaeva tuleb kindlasti näha efektiivsema kommunikatsiooni nimel. Kuna osapooli, kes peaksid olema infoga kursis on palju, võiks eelistada info vahetamisel ja talletamisel kirjalikku varianti. Koosoleku vormis info vahetamine on küll kiirem, aga koosolekute kokkukutsumine eeldab pikka planeerimisaega, et leida ühiselt sobiv ajaaken ning lisaks võib seal jagatud info ununeda.

Sellisel juhul peaks koosolekul olema inimene, kes kirjutab üles koosolekul arutatud teemad ja kokku lepitud tegevused ning saadab selle peale koosolekut laiali kõigile osalistele, et hiljem oleks võimalik seda infot üle vaadata ja sellele oma töös toetuda. Samuti on operatiivseks suhtluseks võimalik kasutada suhtlusrakendust MS Teams, kus saab mitme osapoolega korraga kirjaliku vestluse vormis infot vahetada ja kokkuleppeid sõlmida ning mille ajalugu jääb alles ning selle juurde on võimalik alati tagasi pöörduda.

DSR-is on hakatud looma ka vestlusi iga suurema arendusülesande jaoks eraldi, kuhu lisatakse kõik selle ülesandega seotud osapooled, et arendusprotsessis osalejad oleksid kogu info ja tehtavate muudatusotsustega kursis. Kindlasti on nendes vestlustes ka analüütikud, kes vastavalt kokkulepitud muudatustele jooksvalt uuendavad ka analüüsi dokumenti, et analüüsile tuginev arendus oleks korrektselt tehtud ja dokumenteeritud.

4.3. Toote kuhi DSR-is

Vastavalt teoorias mainitud reeglile, et ühel tootel peaks olema üks toote kuhi, DSR osakond ei toimi, kuna kasutatakse ühte üldist toote kuhja. Samas on teoorias lubatud kasutada ka ühte toote kuhja, kui selle mitmeks kuhjaks jagamine muudab protsessi keerukaks.

DSR-i algselt arendatavast tootest on aja jooksul välja kasvanud uued tooted, mille haldamiseks ei ole siiamaani vajadust omaette toote kuhja loomiseks nähtud. Hetkel asuvad nende toodete ülesanded ühises toote kuhjas, aga nende groomimine toimub eraldi koosolekuna. Lisaks on osakonnas jagatud toote omaniku rolli meeskonna liikmetele, kes vastutavad konkreetse toote eest, kuid vastutavaid toote omanikke on üks.

Autor nõustub ühise toote kuhja kasutamise otstarbekusega, kuna osakonna tooted on üksteisest välja kasvanud või omavahel sõltuvad. Kui toote kuhjasid oleks mitu, peaks olema võimalik neid üheselt kuhjade vahel jaotada. Kui üks kasutajalugu on sisendiks mitmele tootele, siis tekib probleem, kuhu kuhja see kasutajalugu lisada ning millise prioriteedi see antud kuhjas saab. Ühise toote kuhja puhul on sellistele kasutajalugudele lihtsam prioriteete anda ja neid hallata.

Toote kuhja prioritiseerimise eest hoolitseb vastutav toote omanik, kes on ühtlasi *product management team lead*. Toote kuhja hallatakse Jiras, kus on üks üleüldine toote *backlog* kõikidest ülesannetest ja kasutajalugudest, mida oleks vaja arendada. Toote kuhjas on siltide abil märgistatud alamtoodete (näiteks toode X, toode Y), meeskondade (näiteks Tartu meeskond, Tallinna meeskond) ülesanded, mida on Jira funktsionaalsuse abil võimalik kergelt

välja filtreerida, et toote omanik saaks kerge vaevaga näha nende toodete/meeskondade ülesannete prioriteetide järjekorda.

Vastavalt tähtsusele valib toote omanik need ülesanded, mis tõstetakse toote kuhja etteotsa ja mis liiguvad edasi analüüsi etappi. Väiksema prioriteediga ülesanded toote kuhjas ei pruugi kunagi arendusse jõuda, sest need kas aeguvad (toode liigub hoopis uute suunda), lahendatakse jooksvalt mõne teise arenduse käigus või ootavad seda hetke, kui on „vaba aega“ nendega tegelemiseks (näiteks erinevad toote parendamise ülesanded). DSR-i toote kuhjas on hetkel umbkaudu 600 ülesannet, millest vanimad on registreeritud rohkem kui 2 aastat tagasi ning reaalsus on näidanud, et „vaba aega“ nendega tegelemiseks ei pruugi tulla, sest uued lisanduvad nõuded on prioriteetsemad.

Autor leidis, et toote kuhjas olev Jira ülesanne koos lisatud kirjeldusega ei vasta üksi teoorias kirjeldatud DEEP D tähe nõudele ehk ülesanne ei ole üksikasjalikult defineeritud. Teooria kohaselt peavad esmajärjekorras arendusse minevad ülesanded olema piisavalt detailsed ja mõistetavad. DSR osakonnas lisatakse vastav detailsus Confluences loodud analüüsi lehele, mis lingitakse Jira ülesandega.

Kui ülesanne on võetud arendusplaani ning analüütik on oma töö selle ülesande kallal lõpetanud, liigutab ta selle „*Ready for grooming*“ nimekirja ehk ülesanne on valmis minema hindamisele. Hindamisele ei saadeta ülesandeid, millel ei ole täpsustavat ja selget informatsiooni, et arendusmeeskond oskaks seda hinnata ja ka kohe selle kallal tööle asuda. Hindamiseks minevate ülesannete nimekirja saavad lisada ülesandeid ka toote omanikud ja teised arendusmeeskonna liikmed. Eraldi on „*Ready for grooming*“ nimekirjad tehtud ka kolmele suuremale tootele ning neid hinnatakse eraldi toote groomingu koosolekutel, mida on tutvustatud tabelis 1 lk 44.

4.4. Groomingu koosolekud DSR-is

Üldised groomingu koosolekud toimuvad Tartu kontori arendusmeeskonna liikmete ning kõikide toote omanike osalusel. Kuna Tallinna kontoris asuv arendusmeeskond tegeleb spetsiifiliste toote osadega, siis nemad tavaliselt üldistel groomingu koosolekutel ei osale. Toote spetsiifilistel groomingutel osalevad vaid need arendusmeeskonna liikmed, kes on seotud selle toote arendusega ja oskavad seda hinnata. Selline groomingu koosolekute hulk ja osalejaskond on DSR-i osakonnas kompromiss ette tulevate tootearenduste kogu osakonnale tutvustamise ja kaasarääkimise võimaldamisel ühelt poolt ning inimeste koosolekutega üle

koormamisest hoidumise vahel teiselt poolt. Osakonnas kaalutakse liikumist toote põhiste alammeeskondade ja tseremooniate suunas.

Groomingu koosolekuid viib läbi toote omanik (üldiselt on selleks *product management team lead*, eraldi toodete puhul konkreetse tootega tegeleva meeskonna toote omanik), kes planeerib selle jaoks sobiva aja ja saadab enne koosolekut ka nimekirja ülesannetest, mis hindamisele tulevad, et osalejad saaksid nendega juba eelnevalt tutvuda. Koosolekud toimuvad regulaarselt 1 kord sprindi jooksul, ajaliselt planeeritakse selleks maksimaalselt 1,5h. Mõnikord, kui kogu osakonna sisend ei ole vajalik, teevad oma väiksemaid meeskonna siseseid hindamisi ka arendajad või testijad.

Kuna teooria ei ole rangelt määratlenud, kuna ja kui palju võiks või peaks neid koosolekuid tegema, siis taoline lähenemine meeskonna siseste hindamiste jaoks on autori arvastes hea lahendus, mida kasutatakse, et mitte raisata kogu osakonna väärtuslikku tööaega selliste ülesannete arutelu peale, milles nende poolse sisendi vajadus puudub. On tähtis, et need koosolekud siiski toimuksid ja vajalikud ülesanded saaksid võimalikult täpselt hinnatud, et neid saaks planeerida eelseisvatesse sprintidesse.

4.4.1. Story points ühikud DSR-is

DSR-is kasutatakse ülesannete hindamiseks *story points* (edaspidi SP) lähenemist. Scrum'i raamistikku kasutusele võttes oli algselt arendusmeeskonnal segadus, kuidas SP peaks hindama, kas pigem tundides või aja ja keerukuse kombinatsioonis. Raske oli jõuda ühise arusaamise ja lähenemiseni, kuna konkreetsed kokkulepped puudusid ja meeskonna liikmed hindasid SP vastavalt oma kriteeriumitele. Selleks, et protsessi parandada ja paika panna kõigi jaoks ühised lähtekohad, analüüsis Scrum Master kollektiivse hinnangu andmiseks kasutatavat süsteemi ja võimalikke kitsaskohti.

Kui SP puhul kasutatakse kombinatsiooni ajast ja keerukuses, sai protsessi eelisena välja tuua ülesannete hindamise kiirust ilma konkreetset ajahinnangut andmata. Lisaks võimaldab taoline hindamine tabada ka ebamäärasust, kuna kasutatakse Fibonacci jadal põhinevaid ühikuid, mis määravad ülesandele kuluva aja. Samas on SP-d piisavalt täpsed, et need võimaldaksid planeerida tulevasi sprinte ning hallata projektiplaane ja prognoose.

Kitsaskohana toodi välja SP-de võrdsustamine ainult ajaga, kuna see vähendab suhtelise hindamise ühiku kasulikkust. See tähendab, et kui meeskond annab hinnangu 1 SP, mida omakeskis võetakse kui 1 tund, siis tuleks töö ära teha ka 1 tunniga. Samas ei tea meist keegi

kunagi täpselt ette, kui palju aega kuluda võib. Siit kaasneb ka teine puudus, et inimesed on tundides hindamises ebatäpsed ning palju täpsemad suhtelise suuruse hindamisel.

Üldise kollektiivse hinnangu leidmise protsessi probleemiks peeti ka SP-de keskmistamist, mis tähendab seda, et kui hinnangute andmisel antakse mitme meeskonna liikme poolt võrdselt näiteks SP=3 ja SP=5, siis ülesande hinnanguks märgitakse (5+3)/2= 4.

Autor nõustub Scrum Master'i analüüsiga, et selline lähenemine on vale, kuna osa meeskonnast on arvanud ülesande keerukuseks SP=5 ja kui nemad peaksid selle ülesande, mille hinnanguks on SP=4, uues sprindis valmis tegema, siis on võimalik, et seda ei pruugita valmis saada. Selline keskmistamine annab valesid hinnanguid ülesannetele sprindi planeerimiseks. Pigem tuleks läbi arutada, miks sellised suuremad hinnangud on antud ning leida, kas hirmudel on ka alust või pigem on võimalik ülesanne valmis teha ka kiiremini, teades rohkem fakte.

Lisaks on protsessi puudusena välja toodud SP hinnangu muutmine käimasoleva sprindi ajal ehk siis kui arendusmeeskond on hakanud lõpuks ülesannet reaalselt lahendama. Seda ei tohiks teha ka siis, kui arendusmeeskond on saanud aru, et nad on andnud ülesandele vale või ebatäpse hinnangu. Kui antud SP hinnang on olnud ebatäpne, on see ikkagi üks osa sprindi ja meeskonna kiirusest (*velocity*) ning antud teave jääb alles ja näitab ajaloolist kiirust, mida saab kasutada tulevaste sprintide paremaks planeerimiseks. Selliseid valesti antud hinnanguid tuleks arutada ka sprindi retrospektiivi koosolekul, kuid tihtilugu unustatakse seda teha, mis on ka üheks miinuseks, mille kallal tuleks tööd teha, kui kasutada SP lähenemist.

Seega lähtuvalt analüüsitud süsteemist ja kitsaskohtadest on DSR-i meeskonnas kasutusel SP lähenemine, kus hinnatakse ülesande aja ja keerukuse kombinatsiooni. Iga meeskond on pannud enda jaoks paika SP tähenduse vastavalt aja ja keerukuse kombinatsioonile. Osakonnas püütakse üle minna ühtlasema hindamismudeli kasutamisele üle kõikide meeskondade. Tabelis 2 lk 52 on välja toodud praegu kasutusel olevad SP hinnangud ning ühtlustatud SP hinnangud, mida soovitakse kasutusele võtta. toodud tundide all on arvesse võetud lisaks ka keerukust. Keerukuse all peetakse silmas nii dokumentatsiooni tegemist, võimalike vigade parandamist kui ka ülesannet ennast. Lisaks manuaalse töö osakaalu ning kas peaks juurde lisama ka automatiseerimist (näiteks automaattestide kirjutamist).

Tabel 2. SP hinnangute tähendused DSR-i osakonnas.

	Hetkel kasutatavad SP hinnangud		**Ühtlustatud hindamismudel**	
SP	Aeg ja keerukus	Hinnang päevades	Aeg ja keerukus	Hinnang päevades
1	kuni 1h tööd	0,125	kuni 4h tööd	0,5
2	kuni 4h tööd	0,5	kuni 1 päev tööd	1
3	4h kuni 1 päev tööd	1,5	1-2 päeva tööd	1,5
5	kuni 3 päeva tööd	3	2-3 päeva tööd	2,5
8	kuni 4 päeva tööd	4	Kuni 4 päeva tööd	4
13	5 päeva tööd	6	5 päeva tööd	6
21	2 nädalat tööd	10	2 nädalat tööd	10

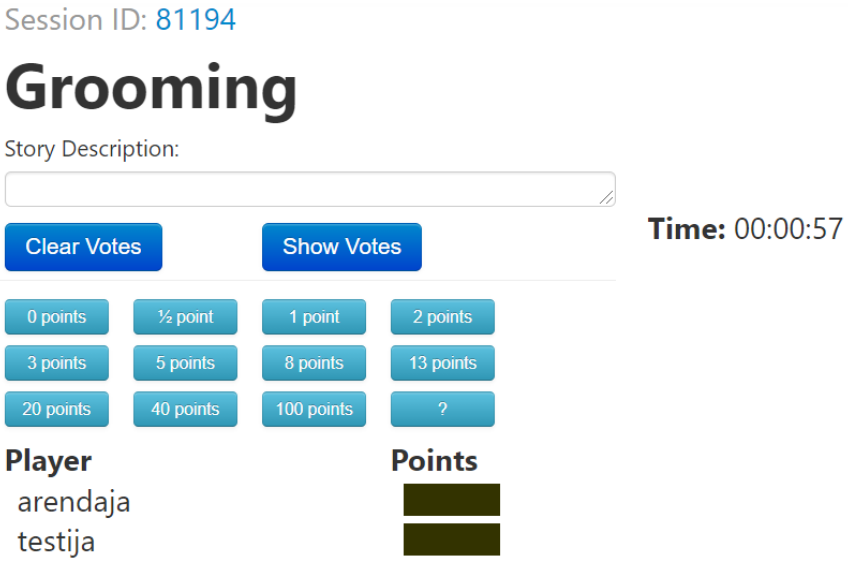
Hetkel kasutusel oleva SP süsteemi ühtlustamine on käsile võetud seetõttu, et seosed SP väärtuste vahel ei ole kooskõlas reaalsete ülesannetega. Eelkõige puudutab ühtlustamise protsess SP-sid 1-5. Näiteks on ebamõistlik kasutada 1 SP jaoks keerukust kuni 1h tööd, kuna neid juhtumeid, kus taolisi ülesandeid esineb, on DSR-is harva. Taolise SP kasutamine lisab ka lubatud ajaraamistikus ülesande täitmise riski ehk kui hakatakse nägema, et ülesande täitmine võib võtta rohkem aega, hakatakse kiirustama ja seetõttu antakse järgi kvaliteedis. Lisaks soovitakse saada paremat sisendit sprindi planeerimiseks.

4.4.2. Varjatud ajahinnangute andmine DSR-is

DSR-i osakonnas kasutatakse ülesannete hindamiseks Grenningi poolt loodud varjatud ajahinnangute andmise ideed. Antud idee kasutamise eesmärk on tagada hinnangute andmise hetkel sõltumatus teiste antud hinnangutest, mis omakorda suurendab hinnangute erinevusi (ei kopeerita tahtlikult teisi). Erinevad hinnangud tekitavad arutelusid töö sisu kohta, mis viib täpsemate hinnangute andmiseni ja parimate praktikate levimiseni meeskonna liikmete vahel.

Toote omaniku poolt planeeritud koosolekul on kohal kogu osakond (kui ei ole tegemist eraldi väiksema toote X või Y groomimisega), kuid samas kõik liikmed ei osale hindamisel (näiteks analüütikud, kes on kohal lisaküsimustele vastamiseks). Varasemalt toimusid koosolekud kontoris ning kasutati agiilse planeerimise kaardipakke. Igas pakis oli olemas SP numbrilise ühiku kaart ning lisaks ∞, ? ja kohvitass (vastab samale ideele, mis oli kirjeldatud π puhul), kuid 2020. aasta koroonapandeemia tõttu kodukontoritesse kolimisel ei saa selliseid koosolekuid enam teha. Alternatiivina toimuvad koosolekud virtuaalselt ning kasutatakse vabavaralist *Pointing Poker* [22] sessiooni, mida on kujutatud joonisel 12. Lisaks on Scrum

Master'i poolt välja töötamisel planeerimise tööriista (*planning tool*) funktsionaalsus varjatud ajahinnangute andmise rakenduse lisamiseks, mida oleks võimalik integreerida Jiraga ning automatiseerida antud hinnangute lisamist otse Jira ülesandele.



Joonis 12. *Pointing Poker* sessiooni näide

Joonisel 12 lk 52 näidatud sessiooni ID-d kasutades saavad kõik osapooled ennast kirja panna ja hääletada. Võimalik on kasutada vaikimisi antud numbrilisi ühikuid nagu joonisel 12 või luua oma ühikute süsteem. Antud SP hinnangud on nähtavad alles siis, kui kõik on oma hinnangu andnud ning toote omanik on need avalikuks teinud (*show votes*). Autor peab heaks funktsionaalsuseks ka taimeri olemasolu, sest kui mõne ülesande arutelu läheb liiga pikaks ning segadust tekitavaid küsimusi on palju, siis ei ole ilmselt antud ülesanne valmis hindamiseks ning seda peaks täpsustama ja järgmisel korral uuesti hindama.

Toote omanik valib järjest hindamisele tulevad ülesanded. Kui teooria on öelnud, et toote omanik loeb seejärel ette hindamisele tuleva ülesande, siis reaalsuses see DSR-is nii ei ole. Pigem eeldatakse, et kõik on eelnevalt tutvunud hindamisele tuleva ülesandega iseseisvalt, kuna nimekiri ülesannetest saadetakse kõigile enne koosoleku toimumist. Keerulisemate ülesannete puhul antakse sõna analüütikule, kes kirjeldab ülesannet ning selle sisu, misjärel saab meeskond küsida täpsustavaid küsimusi ja ülesannet hinnata. Siinkohal saab autor öelda, et toimitakse teooria järgi, isegi kui toote omaniku asemel annab ülevaate vastutav analüütik, sest kui meeskonnal on arusaam ülesande sisust, saab igaüks anda oma hinnangu. Hinnanguid annavad vaid arendajad ja testijad. Eraldi arvestust peetakse arendajate antud hinnangute ning testijate antud hinnangute puhul. Kui hinnangud ühtivad, pannakse toote omaniku poolt kirja arendajate poolt antud hinnang, testijate poolt antud hinnang ning siis nende kahe osapoole

summa, millest lähtutakse sprindi planeerimisel ehk siis näiteks *software engineer*: 5 ja *quality assurance*: 8, mis annab ülesande SP hinnanguks SP=13. Hinnangud märgitakse Jira ülesande juurde – eelpool mainitud *report notes* lahtrisse pannakse kirja iga meeskonna antud hinnang ning *story points* lahtrisse saadud summaarne hinnang.

Sprindi planeerimisel lähtutakse ülesande suuruse arvestamisel summaarsest hinnangust *story points* lahtris, mille andsid arendajad ja testijad.

4.5. Sprindi planeerimine DSR-is

Kui ülesanded toote kuhjas on hinnatud, tuleb hakata planeerima uut sprinti. Sprindi planeerimise koosolekud toimuvad toote omaniku eestvedamisel (*product management team lead*) 1 kord sprindi jooksul ja selleks on planeeritud 1,5h. Kui teooria ütleb, et sprindi planeerimisel peaksid osalema kõik osapooled (toote omanik, Scrum Master ja arendusmeeskond), siis DSR-is osalevad planeerimisel ainult meeskondade juhid (*team leads*) ja Scrum Master. Seega teoreetiline lähenemine, kus planeerimise käigus tutvustab toote omanik uue sprindi eesmärki ning arendusmeeskond saab anda omapoolset hinnangut ja realistlikku nägemust, mida nad suudaksid uues sprindis teha, ei ole DSR-is kasutusel. Planeerimise koosolekul on vaid meeskondade juhid, kes esindavad oma meeskonda ja planeerivad nende tööülesanded.

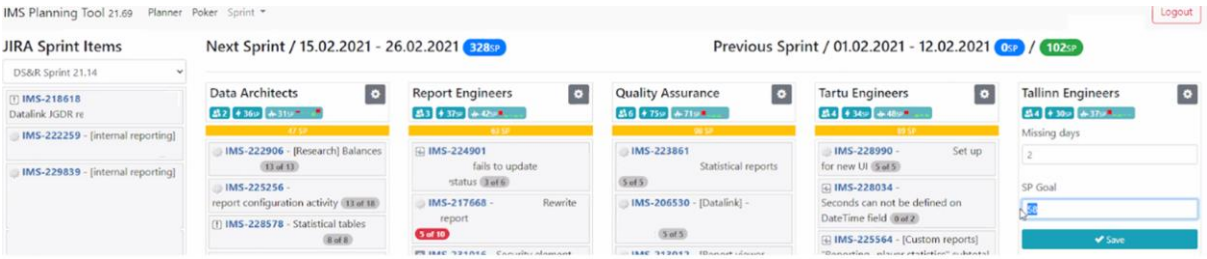
Autori arvates on taoline teooriast kõrvalekalle põhjendatud, kuna kogu arendusmeeskonna kaasamine (39 liiget) muudaks planeerimise koosoleku äärmiselt mahukaks ja aeganõudvaks. Kuna meeskondade juhid on lisaks toote omanikud ning nende juhtida on ülesannete prioriteedid, siis suudavad nad anda head ülevaadet oma meeskonna võimekusest ning nende tööd realistlikult planeerida. Enne planeerimise koosolekut arutatakse võimalikud sprinti planeeritavad ülesanded läbi meeskonna siseselt ning planeerimise koosolekul antakse edasi kogu meeskonna nägemus.

Sprindi planeerimise protsessi on aja jooksul parendatud, et see oleks kiirem ja efektiivsem ning täpsem. Algselt kasutati ainult Jira toote kuhja, kust valiti kõrgema prioriteediga ja groomingul hinnatud ülesanded, mida võiks eelolevas sprindis arendada. Seejärel vaadati üle inimressursi olemasolu ja planeeriti vastavalt sellele võimalikud ülesanded. Sellise metoodika puhul kerkis esile mitmeid probleeme, millele tuli leida lahendus, et sprindid oleks paremini planeeritud. Järgnevalt on välja toodud 4 probleemi ja nende lahendust:

- hindamisel ei võeta arvesse kõiki aspekte

    - tuleks mõelda laiemalt ja kaaluda ka potentsiaalseid defekte, automaatteste, dokumentatsiooni uuendamist, mis võivad hindamisel SP numbrit suurendada;
- sprinti planeeritakse liiga palju ülesandeid ning aeg-ajalt kantakse neid üle järgmisesse sprinti, puuduv puhver aeg
    - võimalike lahendustena tuleks igasse sprinti lisada rohkem puhver aega ootamatuste või aeganõudvamate tööde jaoks; lisaks tuleks iga ülesande juures mõelda ka potentsiaalsete defektide ja nendele kuluva aja peale, mis võivad esineda; analüütik peaks koos arendajate ja testijatega jagama suured arendused osadeks, mis oleks piisavalt väiksed, et olla teostatavad ühe sprindi raames;
- liiga palju defekte arenduse kohta
    - kui arendust on hinnatud näiteks 8 SP vääriliselt, aga ilmnenud defektide arvelt võib see suureneda, siis kannatavad selle arvelt ka teised sprindi ülesanded, seega tuleks enne arendust teha nn *kick-off* koosolekud, kus analüütik tutvustab ülesande sisu ning oodatavat tulemust, et vältida arendaja poolt tehtavat lisatööd ja muudatusi arenduse käigus;
- liiga palju hinnatud ülesandeid, liiga suur toote kuhi
    - kui toote kuhi on liiga suur, on ka raske teha valikuid, millised peaksid olema prioriteedid. Samuti on probleemiks kui hinnatud ülesannete nimekiri muutub liiga pikaks ning need kõik ei mahu eelolevasse sprinti, mis tähendab, et kui ülesanne kunagi lõpuks sprinti jõuab, siis ei pruugi antud hinnang enam täpne olla ja põhjustab probleeme planeerimisel. Lahendustena tuleks toote kuhi iga toote omaniku poolt üle vaadata vähemalt kord kvartalis ja eemaldada ülesanded, mis ei ole enam asjakohased ning hindamisele võtta ainult neid ülesandeid, mida kavandatakse järgnevatesse sprintidesse.

Selleks, et planeerimise tegevust järjest parendada ning kiiremaks ja lihtsamaks muuta, on Scrum Master arendanud PHP-l põhineva planeerimisvahendi „*Planning tool*“. Antud lahendus on integreeritud Jiraga, et kõik sprinti planeeritavad ülesanded saaksid arvesse võetud koos neile antud SP hinnangutega. Lisaks on välja toodud eelneva sprindi statistika, kui palju SP-sid planeeriti ja kas need said ka edukalt lõpetatud.



Joonis 13. DSR-is kasutatav planeerimise tööriista *Planning Tool* näide [27]

Joonisel 13 on toodud väike näide kasutatavast planeerimise tööriistast (vaata suuremat joonist lisas 1). Vasakul tulbas märgitakse ära sprint, mida planeeritakse ning kuvatakse nimekiri ülesannetest, mida on võimalik eelseisvasse sprinti planeerida. Järgnevalt on tulbad igale tiimile – andmebaasi arhitektidele, raportite arendajatele, testijatele ning eraldi Tartu kontori tarkvaraarendajatele ja Tallinna kontori tarkvaraarendajatele. Vastavalt teoorias soovitatule on iga grupi juures ära märgitud ka liikmete arv, nende võimekus SP-des eelseisvaks sprindiks ning eelmise 5 sprindi statistika põhjal keskmine võimekus SP-des. Kollasel ribal on hetkel planeeritud ülesannete SP-de summa. Kui riba on kollane, siis on planeeritud rohkem SP-sid, kui meeskond lõpuni viia jõuaks. Sinise riba puhul on planeeritud võrdselt või vähem SP-sid, kui meeskonna võimekus lubaks. Igat gruppi on võimalik modifitseerida vastavalt inimeste olemasolule ja puhkepäevadele ja lisada SP eesmärk eelolevaks sprindiks. Teooria poolt soovitatud 1-2 lauselist sprindi eesmärki kirja ei panda ning eesmärgiks on ainult SP maht.

Autori arvates on raske sõnastada sprindi eesmärki, kuna DSR-i osakonna toote portfell on äärmiselt lai ning sprindi jooksul arendatakse funktsionaalsusi vastavalt kliendi nõuetele, vastavalt sisemistele toote eesmärkidele ning töötatakse erinevate uurimisülesannete kallal. Selline lai sprindi tööde teema võib muuta sprindi eesmärgi sõnastuse pikaks ja kohmakaks ning hetkel kasutatav SP-de maht täidab antud eesmärki DSR-i jaoks hästi.

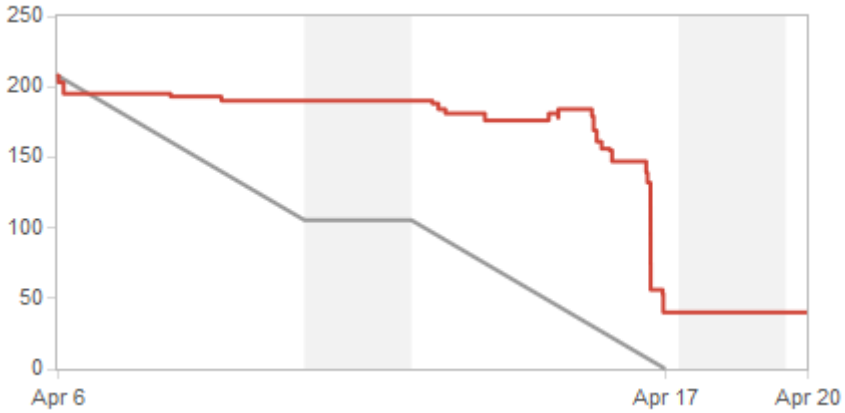
Sprindi planeerimist alustatakse meeskonna liikmete ressursi ülevaatamisega ehk täidetakse ära töölt eemaloleku osa (*missing days*). Iga inimese kohta, kes on tööl 10 tööpäeva ehk terve 2 nädalase sprindi, arvestatakse võimekuseks 21 SP. Seega kui üks inimene on tööl ainult 8 tööpäeva või töötab osakoormusega, siis vastavalt sellele muudetakse ka tema võimekuse SP arvu. Lisaks vähendatakse inimese SP arvu, kui tal on lisaks toetavad tegevused (*support activities*), mida ta sprindi ajal peab tegema. Seda arvestust peetakse iga meeskonna grupi kohta eraldi. Lisaks vaadatakse üle, kas järgmise sprindi jooksul on tulemas ka vabu päevi või riigipühasid, mis vähendavad tööaega. Vastavalt neile tegevustele saadakse kokku sprindi eesmärk, mida arendusmeeskond suudab lõpuni viia ja mida väljendatakse SP-des.

Seejärel vaadatakse üle lõppenud sprint ning kas seal oli ülesandeid, mida ei suudetud lõpetada ja arutatakse, kas need peaks üle tooma uute sprinti. Kui ülesanne tuleb üle tuua, vaadatakse üle ülesandele antud SP hinnang ning vastavalt tehtud tööle korrigeeritakse seda uue sprindi jaoks. Näiteks kui ülesande SP oli 13 (arendaja oli hinnanud ülesannet SP=8 ja testija SP=5) ning arendaja oli oma osa lõpetanud, kuid testimiseni ei jõutud, siis uues sprindis on see ülesanne hinnanguga SP=5. Ülesannete üle toomine vähendab vastava meeskonna uue sprindi SP võimekust vastavalt üle toodud SP-de arvule. Kasutatakse ka LATE silti ülesannete juures, kui on juba ette teada, et ülesanne eeldab arendust terve sprint (21 SP) ja testimisega saab alustada alles uuel sprindil. Selline lähenemine annab parema ülevaate olukorrast, kus ülesande arendamisele kulub aega mitu sprinti ning planeerimise käigus arvestatakse selle ülesandega ka ülejärgmise sprindi planeerimisel.

Pärast seda vaadatakse üle vaba ressurss uute ülesannete planeerimiseks, nimekiri sprindi ülesannetest ning hakatakse meeskondadele ülesandeid lisama. Ülesande lisamisel muutub planeerimise tööriistas automaatselt ka SP-de arv, mis on meeskonnale planeeritud ja kas see jääb meeskonna võimekuse piiridesse või ületab seda. Vastavalt sellele hakatakse SP-sid korrigeerima ülesannete lisamisega/eemaldamisega. Kui SP-d on vaid veidi üle lubatud võimekuse, antakse otsus meeskonna juhile, kas selline olukord oleks tema poolt lubatav või tuleks ülesandeid vähendada. Vähendamise puhul eemaldatakse kas suure SP arvuga ülesanne või väiksema prioriteediga ülesanne, mis tõstetakse edasi tuleviku sprinti, et uuel planeerimisel oleks see kohe nähtav.

Pärast planeerimise koosolekut saadavad nii arendajate kui ka testijate meeskonnajuhid välja ülevaatliku kirja kogu osakonnale, kus on ära märgitud meeskonnaliikmete puhkused ja ülesanded, millega iga meeskond uuel sprindil tegelema hakkab ja kes tegeleb toetavate tegevustega.

Üksikutel juhtudel saatis sarnase kirja välja ka osakonnajuht või Scrum Master, kus esialgu toodi välja eelneva sprindi kokkuvõte töö edenemise graafiku näol, mis on toodud joonisel 14 lk 58 ja näitab punase joonega planeeritud SP-de mahtu ning mis tulemuseni jõuti sprindi lõpus. Hall joon näitab ideaalset olukorda, kuidas planeeritud SP-d võiksid sprindi jooksul järjest väheneda ning jõuda olukorda, kus kogu töö on tehtud. Hallid alad joonisel märgivad nädalavahetuse päevi, kui tööd ei tehta (seda toetab ka ideaalse sprindi joon, mis hallil alal liigub külgsuunaliselt, mitte alla).



Joonis 14. Näide DSR-i sprindi töö edenemise graafikust

Lisaks toodi kirjas ära nimekiri eelmises sprindis lõpetatud ülesannetest, ülesannetest, mis liigutati uute sprinti ja nimekiri uuest planeeritud sprindi ülesannetest.

4.6. Sprindi kuhi DSR-is

Kui sprindi planeerimine on lõppenud, on tekkinud sprindi kuhi nende ülesannetega, millega eelseisvas sprindis tegelema hakatakse. Sprindi kuhja sprindi jooksul reeglina ei muudeta, kuid jooksvalt võetakse sisse kiireloomulist lahendamist nõudvaid toodangus avastatud defektide parandustöid. Suuremate arenduste SP hinnangutesse on arvestatud puhver lisa arenduste ja potentsiaalsete defektide parandamise jaoks, mis tuleb lahendada sama sprindi koosseisus. Selle tegevuse eesmärk on sprindi lõpus alati saavutada kvaliteetne tulemus, mitte defekte sisaldav uus funktsionaalsus.

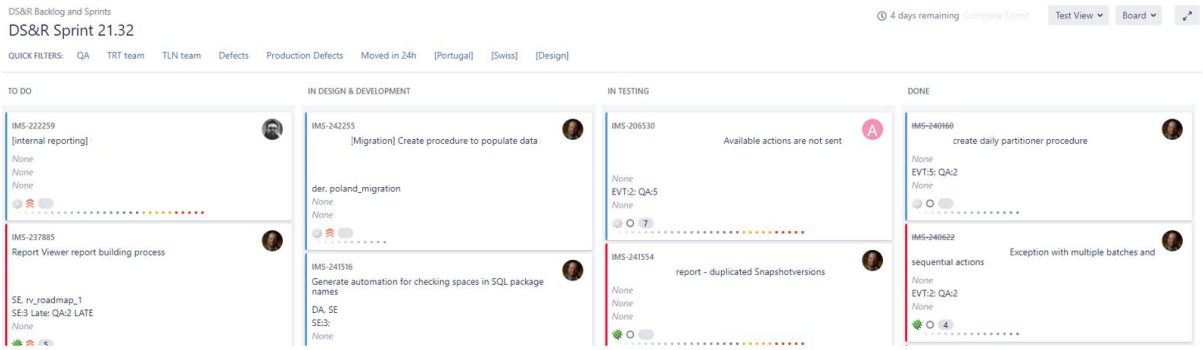
Nii nagu ütleb teooria, et käimasolevas sprindis tohib vaid arendusmeeskond kuhja ülesandeid juurde lisada, kuna nemad oskavad hinnata ülesande suurust, DSR-is ei ole. Ehk kui arenduse käigus on ilmnenud lisaülesanne, mis on sõltuv arendamisel olevast ülesandest, saab arendaja otsustada, kas see on piisavalt väike, et see ülesanne lisada käimasolevasse sprinti või nõuab see suuremat arendustööd, mida peaks eelevalt analüüsima ja hindama ning seetõttu ei ole võimalik seda sprindi kuhja koheselt lisada. Lisaks muudavad sprindi kuhja DSR-is ka toote omanik ja arendusmeeskonna juht ning mõnikord ei suuda arendusmeeskond taolisi lisatud ülesandeid sprindi jooksul valmis saada, kuna nendega ei oldud planeerimise käigus arvestatud.

4.7. Sprindi teostamine DSR-is

Sprindi teostamiseks on aega 2 nädalat ning selle aja jooksul teevad kõik meeskonna grupid iseseisvalt tööd. Tekkivate küsimuste jaoks on alati kättesaadavad analüütikud ning toote omanikud, kes selgitavad arendusmeeskonnale vajadusel ülesande sisu ja oodatavat tulemust.

Kogu arendusmeeskonda hõlmavat *daily scrum*'i või *stand-up* koosolekut DSR-is läbi ei viida, kuid neid võimalusi ei ole keelatud kasutada igal meeskonna juhil oma meeskonna põhiselt ja kasutada teoorias välja toodud 3 põhiküsimust, mis annavad kiire ülevaate tööde seisust. Näiteks toimub igal esmaspäeval 30 minuti pikkune analüütikute koosolek *product management team lead'*i eestvedamisel, kus iga analüütik tutvustab, mis tegevusi tegi ta eelmisel nädalal, mis tegevusi planeerib selleks nädalaks ja millised on probleemid, mis tema tööd takistavad. Arendusmeeskonnad peavad igapäevaseid 20 minutilisi koosolekuid *reporting services team lead*'i eestvedamisel hommikul kell 9:10, kus arutatakse läbi, mida meeskonnaliige tegi eile, mida planeerib täna ja mis on probleemid. Testijate meeskond *quality assurance team lead*'i eestvedamisel viib läbi igapäevase 30 minutilise koosoleku, et olla kursis iga liikme ülesannete seisu ja probleemidega.

Sprindi kestel on lisaks sprindi hetkeolukorra koosolekud (*sprint status meeting*), mida viib läbi Scrum Master ja kus osalevad lisaks meeskondade juhid, toote omanikud ja analüütikud. Antud koosolekuid toimub sprindi jooksul 3 tükki, nagu oli kirjeldatud tabelis 1 lk 44 ning need on 30 minutit pikad. Antud koosoleku ajal vaadatakse üle käimasoleva sprindi tahvel Jiras, mida oli kirjeldatud joonisel 9 lk 38 ning on näitlikustatud joonisega 15 , kus on toodud DSR-is kasutatavad tulbad (vaata suuremat joonist lisas 2).



Joonis 15. DSR-i sprindi tahvel.

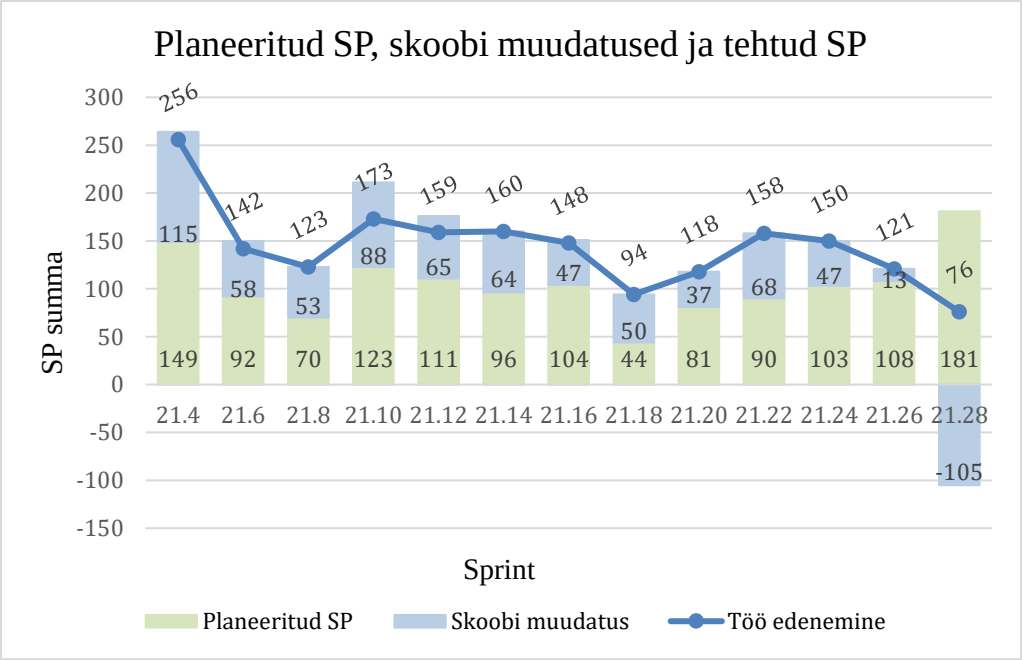
Esimeses „*To Do*" veerus on ülesanded, mida ei ole veel alustatud. „*In Design & Development*" veerus on ülesanded, mis on töös arendusmeeskondades ning „*In Testing*" veerus need ülesanded, millega tegelevad juba testijad või on testimise ootel. Kõik ülesanded, mis on lõpetatud, liiguvad „*Done*" veergu. Koosolekut alustatakse „*In Testing*" tulba ülevaatamisest ning iga ülesande juures annab testijate meeskonna juht ülevaate selle ülesande hetkeolukorrast ning murekohtadest ja seisakutest. „*In Design & Development*" tulba kohta annab tavaliselt ülevaate arendajate meeskonna juht. Joonisel paremal üleval nurgas on meeldetuletuseks näha ka päevade arv, mis sprindi lõpuni on jäänud.

Kui tegemist on „*Sprint status #1*“ koosolekuga, siis väga suurt tähelepanu ei pöörata „*To Do*“ veerule, kuna sprint on alles nii alguses, et ei ole olnud võimalik kõikide töödega korraga veel alustada. Kui on jõutud juba sprindi lõpu poole ja tegemist on „*Sprint status #3*“ koosolekuga, eeldatakse, et kõik „*To Do*“ veeru ülesanded on töös ja saavad lõpetatud sprindi lõpuks. Kui siiski on seal veerus ülesandeid, siis arutatakse, miks ei ole jõutud nendega alustamiseni ja kui suur on võimalus, et selleni üldse ei jõuta ning vastavalt sellele liigutatakse antud ülesanne vajadusel sprindist välja.

Sprindi jooksul jälgib Scrum Master ka töö edenemise graafikuid (*burndown chart*) ja raporteid ning vastavalt sellele statistikale korrigeeritakse ka sprindi planeerimise protsessi.

Antud graafikuid ja raporteid analüüsis ka magistritöö autor ning koostas nende põhjal lisa 3, kus on toodud andmed 2021. aasta sprintide teostamise kohta. Lisas 3 on välja toodud SP-de summad, mida sprintidesse planeeriti, kui palju oli sprindi jooksul skoobi muudatusi (näiteks avastatud defektid, mis tuleks koheselt parandada, hinnangute muutused) ning kui palju SP-sid sprindi lõpuks reaalselt teostati ehk töö edenemise SP summa.

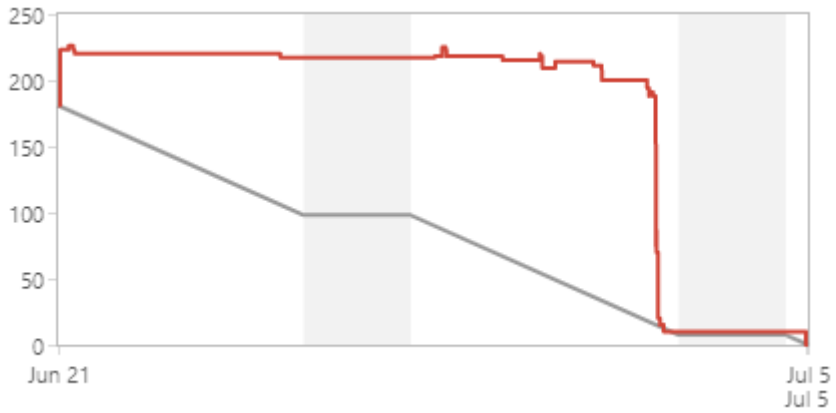
Joonisel 16 on välja toodud rohelistes tulpades planeeritud SP-d ning väärtus ühikute summana iga sprindi kohta, sinistes tulpades on skoobi muudatuste maht SP-des (ülesannete lisamine sprinti, hinnangute muutmine, ülesannete eemaldamine sprindist). Lisaks on joonisele joonega kantud reaalselt teostatud SP-de ühikute summa ehk töö edenemine sprindis, mille alla loetakse nii ülesannete lõpetamine kui ka uuesti avamine (mis vähendab töö edenemist).



Joonis 16. 2021. aastal sprintidesse planeeritud SP, skoobi muudatused ja teostatud SP.

Antud jooniselt nähtub, et sprintide planeerimine on olnud üsna täpne. Uusi ülesandeid on sprinti planeeritud parajas mahus, et jätta puhver sprinti planeeritud arendustega kaasnevate defektide lahendamise ja hinnangute muutumise jaoks ning kiiret lahendust vajavate ülesannete jaoks, mis sprinti võivad lisanduda. Suuri kõikumisi kogu summaarse sprindi skoobi ja töö edenemise vahel on tulnud ette minimaalselt. Lisa 3 koostamise jaoks andmeid analüüsides ei ilmnenud töö edenemise graafikutelt ka olukordi, kus enne sprindi lõppu oleks töö otsa lõppenud.

Analüüsides skoobi muudatusi ilmnes suuri kõikumisi sprindis 21.10, 21.26 ja 21.28. Antud sprintide puhul on selliseid olukordi põhjustanud algse hinnangu muutmine töö käigus, kuna ilmnes aspekte, mis muutsid ülesande teostamise keerulisemaks. Lisaks on põhjuseks olnud ka mahukad keskkondade seadistamise ülesanded, mille puhul on vaja kooskõlastusi teiste osakondadega ning ettenägematut ooteaega ei olnud sprindi ajapuhvrisse planeeritud. 21.26 ja 21.28 sprint erinevad selle poolest, et meeskonnaga liitusid suvepraktikandid, kellele antud arendusülesanded vältavad kogu praktika aja (2 kuud) ning nende hinnanguline maht on osadel ülesannetel lausa 42 SP. Taolisi ülesandeid ei ole võimalik valmis saada ühe sprindi jooksul, kuna süsteemid, mida osakonnas kasutatakse on üsna keerukad. 21.28 sprinti lisati skoobi muudatusena 42 SP suurune ülesanne (lisaks teisi väiksemaid ülesandeid) ning eemaldati suur osa sprindi skoobist (ka algselt planeeritud ülesannetest) sprindi lõpus, kaasa arvatud 42 SP ülesanne, kuna oli näha, et nendega ei jõuta tegeleda. Teostati ainult 76 SP väärtuses ülesandeid, kuigi esialgne plaan ilma skoobi muudatusteta nägi ette 181 SP. Arvulise poole pealt vaadatuna planeeriti sprinti 45 ülesannet ja teostati 44 ülesannet (25 neist planeeritud ülesanded ja 19 skoopi lisandunud ülesanded). Antud tegevused on näitlikustamiseks joonisel 17, kus on kuvatud 21.28 sprindi töö edenemise graafik.



Joonis 17. Sprint 21.28 töö edenemise graafik.

Jooniselt on selgelt näha sprindi alguses lisatud SP-d ning sprindi lõpus eemaldatud SP-d ning sprindi kestel horisontaalne liikumine. Seda põhjustasid antud sprindis mitmed tegurid – mõned ülesanded võtsid rohkem aega, kui esialgne hinnang ette nägi, sprinti lisandusid prioriteetsed kohest parandamist vajavad ülesanded, millest arusaamine ja lahendamine võtsid äärmiselt palju ressurssi, lükates tahaplaanile sprinti planeeritud muud arendused, mis tuli skoobist välja tõsta.

Kokkuvõttes toob autor välja, et planeerimise tööriista kasutamine ja järjest paraneva sprindi planeerimise protsessi mõju on näha meeskonna võimekuse (töö edenemise) pealt valmis saada ülesandeid, mida sprinti planeeritakse ja mis jooksvalt skoopi lisandub. Ettenägematuid olukordi esineb, kuid sel juhul tuleb efektiivselt sprindist eemaldada ülesandeid ja antud arendustegevuste edasilükkamist kommunikeerida kõikidele osapooltele.

4.8. Sprindi ülevaade ja demo DSR-is

Osakonna eesmärk on sprindi ülevaate koosolekute puhul jõuda sinna, et toote omanik annab ülevaate, mis ülesanded on tehtud ja valmis ja mis ei ole ning arendusmeeskond saab võimaluse arutleda, mis sprindi jooksul nende jaoks on hästi läinud, mis takistusi neil töös ette tuli ja kuidas need probleemid lahendati ja seejärel demonstreerib ülesandeid, mis said valmis.

Teoorias tutvustatud sprindi ülevaatamise koosolekuid, DSR-is hetkel ei peeta, kuna antud suurusega osakonna puhul oleks need ebamõistlikult pikad, kui arutada läbi kõik sprindis valmis saanud erineva suurusega ülesanded ja neid kõiki ka demonstreerida. Lisaks puudub osakonna liikmete hulgas huvi taoliste ülesannete demonstreerimiste osas.

Eelnevalt on DSR-is läbi viidud ainult demole keskenduvaid koosolekuid pärast iga sprindi lõppu. Neid koosolekuid korraldas Scrum Master, kes enne koosolekut küsis igast meeskonnast, kas neil on ülesandeid, mis vääriksid tutvustamist ja kes oleks valmis seda tegema. Algselt oli see mõeldud ainult oma osakonna siseselt läbi viimiseks, kuid huvi ilmnes ka teiste IMS-i osakondade poolt ning edaspidi saadeti koosoleku kutset ka laiemalt DSR-i osakonnast väljapoole. Koos koosoleku kutsega saadeti ka nimekiri ülesannetest, mis tulevad tutvustamisele. Demo käigus tutvustati koostöös analüütiku, arendaja ja testijaga igat ülesannet ning näidati valmis tehtud lahendust. Vastati ka täpsustavatele küsimustele, mis osalejate poolt võisid tulla. Kui algselt tehti demo peale iga sprindi lõppu, siis vahepeal tehti seda korra paari-kolme sprindi tagant, kuna arendatavad ülesanded olid kas liiga suured ning kestsid mitu sprinti

ja demo ajaks ei olnud midagi näidatavat valmis või olid ülesanded vastupidi liiga väikesed, et nende jaoks eraldi lühikest demo koosoleku korraldada.

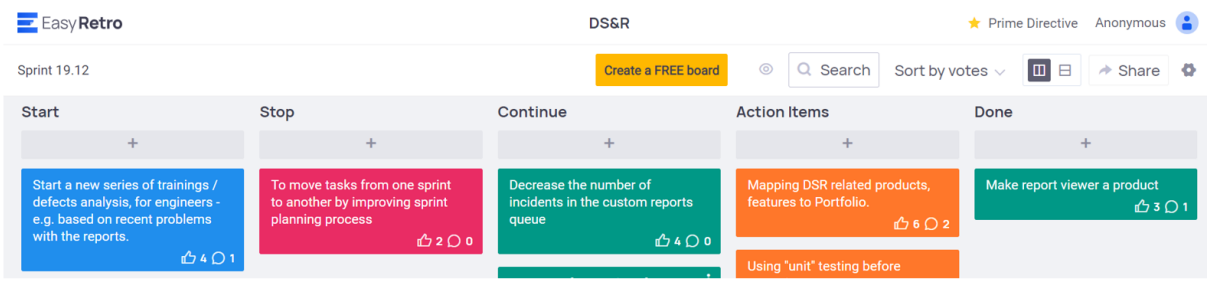
Kuna paljudel meeskonna liikmetel puudus ka huvi sellisel suure osalejate arvuga koosolekul oma tehtud tööd tutvustada ja sunduslikus korras ei soovitud seda peale suruda, siis aja jooksul muutusid demo koosolekute vahed juba väga pikaks ning hetkel neid enam ei tehta üldse. Uusi arendusi ja funktsionaalsusi, tutvustatakse IMS osakonna infokirjas „*Release highlights*“ ning avalikus dokumendis "*Release Notes*".

Autori arvates võiks läbi viia ka demosid, kuna paberil antud kirjeldus ei pruugi edasi anda funktsionaalsuse kasutamisoskust. Selleks oleks vaja laiemale üldsusele näidata, mis funktsionaalsust arendati, kuhu see täpselt tootes lisati ja kuidas on võimalik seda kasutada, kus andmebaasitabelis hoitakse konfiguratsioonide väärtusi jne. Lisaks annab taoline funktsionaalsuse tutvustamine autori arvates võimaluse aruteluks, mille käigus saavad ka osakonnast väljaspool olevad huvigrupid küsida täpsustavaid küsimusi või teha parendusettepanekuid, mis aitaks kaasa funktsionaalsuse ja toote edasisele arengule.

4.9. Sprindi retrospektiiv DSR-is

Iga sprindi lõppedes toimub DSR-is osakonnas ka sprindi retrospektiivi koosolek, mis annab ülevaate möödunud sprindi protsessidest ja tulemustest ning probleemidest. Seda tehakse selleks, et tuua teoorias väljatoodud punktid, kavandada võimalusi ja tegevusi sprindi kvaliteedi ja tõhususe suurendamiseks, reaalsusesse. Retrospektiivi koosoleku tähtsus ja formaat on DSR-is aja jooksul muutunud.

Retrospektiivi koosolekutega alustamisel kutsuti sinna algselt vaid meeskondade juhid ning hiljem lisati juurde ka analüütikud. Koosoleku läbiviimiseks kasutati vabavaralist tahvlit *EasyRetro* [24], mis on toodud joonisel 18 (vaata suuremat joonist lisas 4).

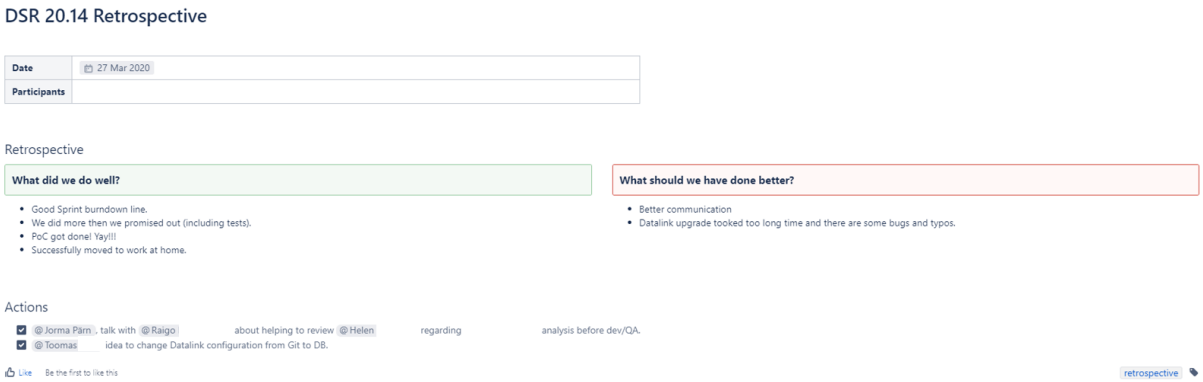


Joonis 18. Näide DSR-is kasutatud tahvlist.

Iga koosoleku alguses anti aega 10-15 minutit, et kõik saaksid vastavasse tulpa kirja panna mõtteid, mida arutada võiks ehk mis tegevusi võiks hakata tegema, mida jätkata ja mis

tegevused võiks lõpetada. Kui mõtted olid kirja saanud, anti igale osalejale 5-6 häält, mida sai kasutada selle mõtte poolt hääletamiseks (pöial üles märk mõtte juures), mida iga osaleja arvates võiks arutada. Kui hääled olid antud, asuti arutama neid mõtteid, mis said kõige rohkem hääli. Arutluse käigus tekkinud arvamused, ettepanekud ja tegevused pandi kirja kommentaaride alla või lisati tegevuskavasse. Uuel koosolekul vaadati need üle, tõsteti valmis saanud tegevused „Tehtud“ tulpa ja hakati uusi mõtteid kirja panema. Kuna antud formaati kasutades muudeti kogu aeg olemasolevat tahvlit ning kogu info kuhjus ühele tahvlile, siis muutus see üsna kiirelt mahukaks.

Uue Scrum Master’i liitumisega ning üldiselt Scrum’i raamistiku kasutusele võtmisega muutus ka retrospektiivi koosoleku vorm. Hetkel viib retrospektiivi koosolekut läbi Scrum Master, kes kutsub koosolekule vaid meeskondade juhid ja iga möödunud sprindi kohta loob ta uue alamlehe Confluence, kus on ära toodud kuupäev, osalejad ning vastatud küsimustele, mis läks hästi ja mis läks halvasti. Lisaks *action item*’id, mida peaks muutma. Joonisel 19 on toodud näide esialgselt kasutusele võetud Confluence lehest (vaata suuremat joonist lisas 5).



Joonis 19. Näide DSR-i retrospektiivi lehest.

Antud lehekülg annab kiire ülevaate hästi läinud ja probleemsetest olukordadest, kuid üsna pea liiguti edasi ülesande põhise retrospektiivi tegemise poole, mis tähendab, et retrospektiivi lehele lisati juurde sektsioon „*Sprint content*“ ehk kõik ülesanded, mis sprindis olid koos lisa veergudega arendajate ja testijate kommentaaride jaoks. Selline lähenemine aitab läbi analüüsida probleemsete ülesannete spetsiifilised murekohad, nagu näiteks konfiguratsiooni muudatused võtsid rohkem aega, kui planeeritud või analüüsi muudeti, seega tuli testimise töö uuesti üle teha. Antud lehele saavad kommentaare lisada kõik arendusmeeskonna liikmed enne retrospektiivi koosolekut. Retrospektiivi koosoleku ajal võetakse esmalt arutlusele need ülesanded, millele on juurde lisatud mingi kommentaar ning seejärel arutletakse ülesandeid

vastavalt vajadusele või ettepanekutele. Kui ülesannete juures midagi rohkemat arutlusele ei tule, vastatakse küsimustele, mis läks hästi, mis läks halvasti ning lisatakse *action item*'id.

5. Kokkuvõte

Järjest kiirenev tarkvara inkrementide tarne klientidele on pannud ettevõtteid kasutama agiilseid arendusmetoodikaid, mis peaks toetama efektiivset töökorraldust muutuvas maailmas, kus kiirelt muutuvas olukorras on võimalik kõikide tööprotsesside läbiviimist jätkata ka kaugtööd tehes.

2020.aastal avaldatud agiilsete meetodite kasutamise uuringust selgub, et 95% vastanutest praktiseerivad igapäevaselt agiilseid metoodikaid. Scrum'i kasutamise populaarsus on tõusnud 2021.aastal tehtud uuringule toetudes 66%-ni, võrdluseks 40%, mis ilmnes esimesest taolisest läbiviidud uuringust. Populaarsemad põhjused, miks on kasutusele võetud agiilsed meetodid, on kiirendada tarkvara tarnimist kliendile, suurendada võimet hallata muutuvaid prioriteete ning suurendada tootlikkust.

Magistritöö on aktuaalne seetõttu, et kõik teoreetilise taustaga raamistikud ei pruugi olla igas ettevõttes üks ühele kasutatavad ning vastavalt ettevõtte taustale ja töökorraldusele oleks vaja ka kasutusele võetud raamistikku enda vajaduste ja eesmärkidega ühildada.

Magistritöö eesmärk, anda ülevaade agiilsest arendusmetoodikast ja Scrum'i raamistiku teoreetilisest taustast koos antud raamistiku kasutamise näitega rahvusvahelises tarkvara ja IT tooteid arendavas ettevõttes sai täidetud. Lisaks kirjeldati muudatusi, mida tehti seoses 2020. aastal laiemalt levima hakanud koroonapandeemiaga.

Töö esimeses osas kirjeldati agiilsete arendusmetoodikate olemust ja tutvustati agiilse tarkvaraarenduse manifesti põhimõtteid.

Töö teises osas kirjeldati Scrum'i raamistikku, mis aitab meeskondades töid ja protsesse korraldada ja juhtida ning annab võimaluse igale meeskonnale, kes Scrum'i kasutab, muuta kasutatav raamistik omanäoliseks ja sobivaks vastavalt vajadusele. Ülevaade tehti Scrum'i rollidest ja artefaktidest ning kirjeldati, mis on Scrum'i tegevused ja kuidas need on omavahel seotud sprindi protsessides.

Töö kolmandas osas anti teoreetilisele osale tuginedes ülevaade Scrum'i raamistiku kasutamisest DSR-i osakonnas. Selleks osales magistritöö autor kõikides tegevustes, mida Scrum'i raamistikule tuginedes osakonnas läbi viiakse. Lisaks analüüsis autor dokumentatsiooni, mida on raamistikku parendades loodud ning Jira töövahendis 2021. aasta sprintide plaane, skoobi muudatusi ja töö edenemise graafikuid.

Järeldustena saab välja tuua, et kõiki teoorias toodud lähtekohti DSR-i osakonnas ei täideta. Suurima kõrvalekaldena teooriast saab välja tuua Scrum'i arendusmeeskonna rolli, mille efektiivsemaiks suuruseks määratakse erinevatele allikatele tuginedes 5-9 liiget. DSR-i osakond ületab seda numbrit rohkem kui 4 korda, olles 39 liikmeline. Lisaks ei viida DSR-is osakonnas läbi sprindi ülevaate koosolekuid, kuid ka sellise liikmete arvu ja osade tegevuste ära jätmisel on suudetud rakendada Scrum'i raamistiku efektiivselt.

2020. aastal levinud koroonapandeemia muutis DSR-i osakonna töökorraldust – tehakse kaugtööd kodukontorites kasutades MS Teams ja Zoom töövahendeid, meeskonnad on rohkem hajutatud ning näost-näkku suhtlus varasemaga võrreldes peaaegu puudub. Koosolekud on muudetud virtuaalseks ning vastavalt sellele on kohandatud ka töövahendeid – suurimad muutused on virtuaalsete koosolekute pidamised ja varjatud ajahinnangute andmise kaartide asendamine *pointing poker* virtuaalruumiga. Antud muudatustega on suudetud kohaneda ja jätkuvalt töötada protsesside, näiteks täpsema sprindi planeerimise, parendamise nimel. Seda toetab ka 2021. aasta planeeritud sprintide analüüs, millest lähtus, et sprintide mahtu planeeritakse üsna täpselt vastavalt meeskonna võimekusele. Mõne sprindi puhul esines ka tegemata töid sprindi lõpus, mida ei suudetud planeerimise protsessis ette näha, kuid taolised olukorrad õpetavad tulevikus neid vigu mitte kordama.

Scrum raamistik on paindlik ning teoorias kirjeldatud lähtekohad annavad võimalusi neid vastavalt oma vajadustele mugandada. DSR osakond on teinud seda üsna efektiivselt ning iga lõppenud sprindiga saadakse uusi teadmisi, mis aitavad protsesse jätkuvalt parandada ning püstitatud eesmärke saavutada.

6. Viidatud kirjandus

[1] **Sommerville, I**. Software Engineering. 9th Edition. U.S: Pearson Education, Inc. 2011

[2] **Rubin, K. S.** Essential Scrum. A practical guide to the most popular agile process.Sixth printing. U.S: Pearson Education, Inc. 2015.

[3] **Sommerville, I.** Software Engineering. 10th Edition. U.S: Pearson Education, Inc. 2016

https://mycourses.aalto.fi/pluginfile.php/1177979/mod_resource/content/1/Sommerville-Software-Engineering-10ed.pdf

[4] **Srivastava, A**. Incremental Vs Iterative Methodologies. 2019. https://businessanalyst.techcanvass.com/incremental-vs-iterative-methodologies-in-software-development/ (24.02.2021)

[5] History: The Agile Manifesto. s.a. https://agilemanifesto.org/history.html (24.02.2021)

[6] Differences Between Agile Coach and Scrum Master. s.a. https://www.knowledgehut.com/blog/agile/agile-coach-vs-scrum-master (29.04.2021)

[7] **Schwaber, K., Sutherland, J.** The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game. 2020.

[8] **Cohn, M.** Succeeding with Agile. Software development using Scrum. Addison-Wesley Professional. 2009.

[9] **Deutschman, A.** Inside the Mind of Jeff Bezos. Fast Company magazine. Issue 85. 2004. https://www.fastcompany.com/50541/inside-mind-jeff-bezos-4 (25.05.2021)

[10] Sprint Planning Meeting. s.a. https://www.mountaingoatsoftware.com/agile/scrum/meetings/sprint-planning-meeting (25.05.2021)

[11] **Huether, D.** Simple Cheat Sheet to Sprint Planning Meeting. 2012. https://www.leadingagile.com/2012/08/simple-cheat-sheet-to-sprint-planning-meeting/ (25.05.2021)

[12] What does INVEST Stand For? s.a. https://www.agilealliance.org/glossary/invest/ (25.05.2021)

[13] **Cohn, M.** User Stories Applied. For Agile Software Development. U.S: Pearson Education, Inc. 2004.

[14] **Wake, B.** INVEST in Good Stories, and SMART Tasks. 2003. https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/ (25.05.2021)

[15] **Grenning, J.** Planning Poker or How to avoid analysis paralysis while release planning. 2002. https://wingman-sw.com/papers/PlanningPoker-v1.1.pdf (25.05.2021)

[16] **Nettleton, C.** How to Use a Sprint Burndown Chart. 2010. https://appliedframeworks.com/how-to-use-a-sprint-burndown-chart/ (25.05.2021)

[17] **Radigan, D.** Agile Sprint Review. Three steps for better sprint reviews with your agile team. s.a. https://www.atlassian.com/agile/scrum/sprint-reviews (29.05.2021)

[18] **Gonçalves, L.** What Is Sprint Review Meeting and How To Hold Fantastic Ones. 2021. https://adaptmethodology.com/sprint-review-meeting/ (29.05.2021)

[19] **Kerth, N**. Project Retrospectives: A Handbook for Team Reviews. Dorset House Publishing co., Inc. 2001. https://ptgmedia.pearsoncmg.com/images/9780133488579/samplepages/0133488578.pdf

[20] What is Sprint Retrospective Meeting in Scrum. s.a. https://www.visual-paradigm.com/scrum/what-is-sprint-retrospective-meeting/ (29.05.2021)

[21] Playtech – Meist. s.a. https://www.playtech.ee/meist (29.05.2021)

[22] Pointing Poker. s.a. https://www.pointingpoker.com (05.06.2021)

[23] **Lowe, D.** Theme, Epic, Story, Task. 2014. https://scrumandkanban.co.uk/theme-epic-story-task/ (05.06.2021)

[24] EasyRetro. s.a. https://easyretro.io/ (05.06.2021)

[25] 14th Annual State of Agile Report. Digital.ai Software, Inc. 2020. https://explore.digital.ai/state-of-agile/14th-annual-state-of-agile-report (24.05.2021)

[26] 15th State of Agile Report. Digital.ai Software, Inc. 2021. https://explore.digital.ai/state-of-agile/15th-state-of-agile-report (22.07.2021)

[27] **Praakli, M.** Project Time Management Challenge: How We Wrangled Chaos Out of Planning. 2020. https://playtech.ee/blog/project-time-management-challenge-how-we-wrangled-chaos-out-of-planning (06.06.2021)

[28] **Laane-Thamdrup, A.** Paindliku ülesehitusega firmas suureneb tegutsemisvabadus. Äripäev. 2004 http://www.mastery.ee/paindliku-ulesehitusega-firmas-suureneb-tegutsemisvabadus/ (06.06.2021)

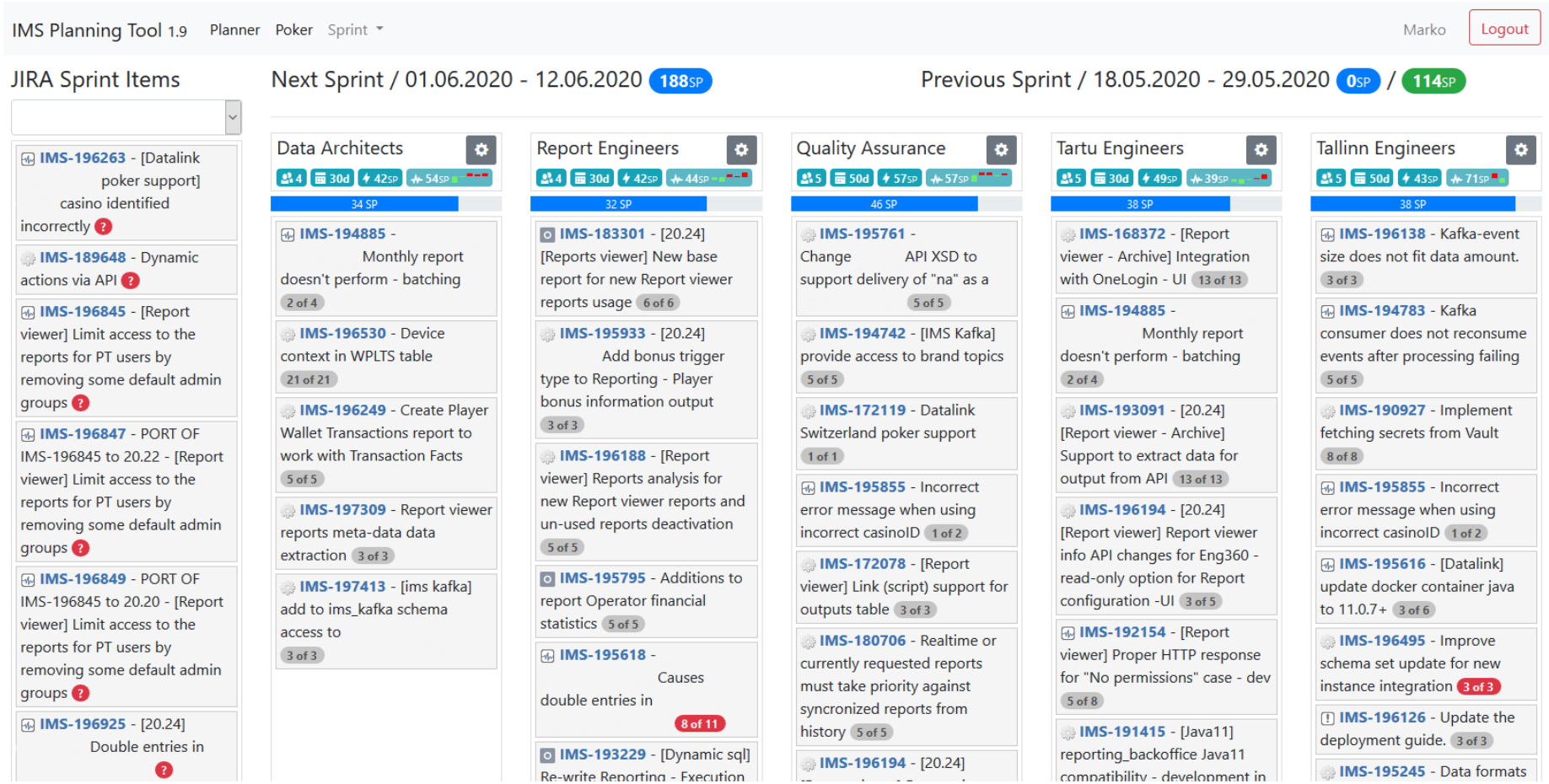
[29] **Kneafsey, S.** A Short History Of Scrum. s.a. https://www.thescrummaster.co.uk/scrum/short-history-scrum/ (22.06.2021)

[30] What Is The Scrum Product Backlog? This Might Surprise You! International Scrum Institute. s.a. https://www.scrum-institute.org/the-scrum-product-backlog-the-scrum-framework.php (22.06.2021)

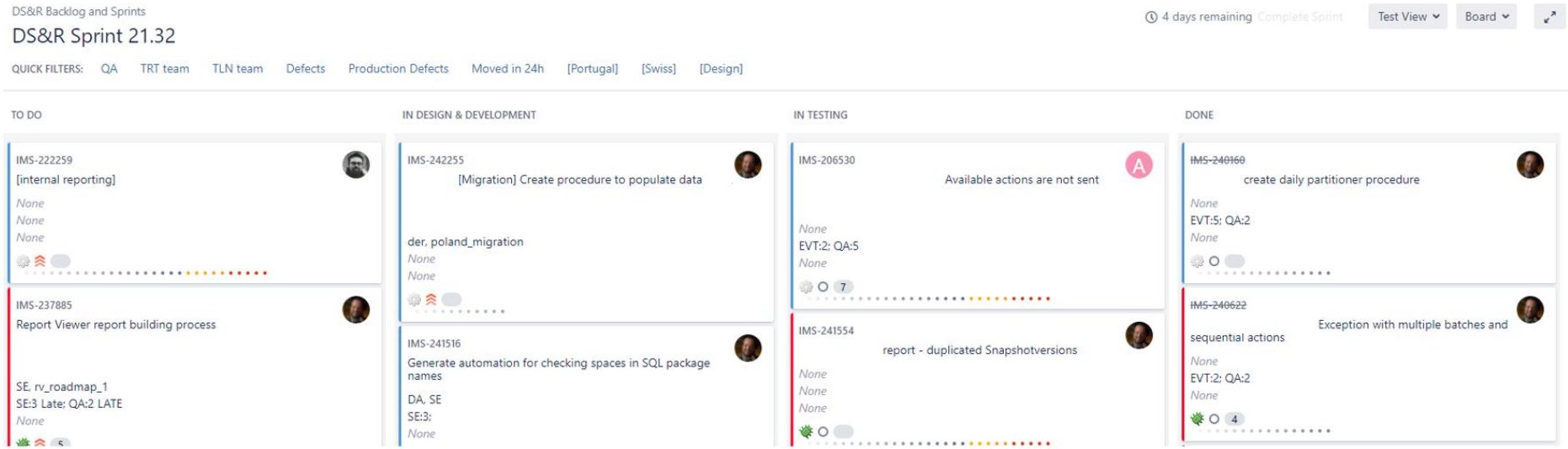
[31] **Brown, G., Meltzer, A., Nagappan, N., Williams, L.** Scrum + Engineering Practices: Experiences of Three Microsoft Teams. International Symposium on Empirical Software Engineering and Measurement. 2011.

Lisad

Lisa 1. Planeerimise tööriista tahvli vaade



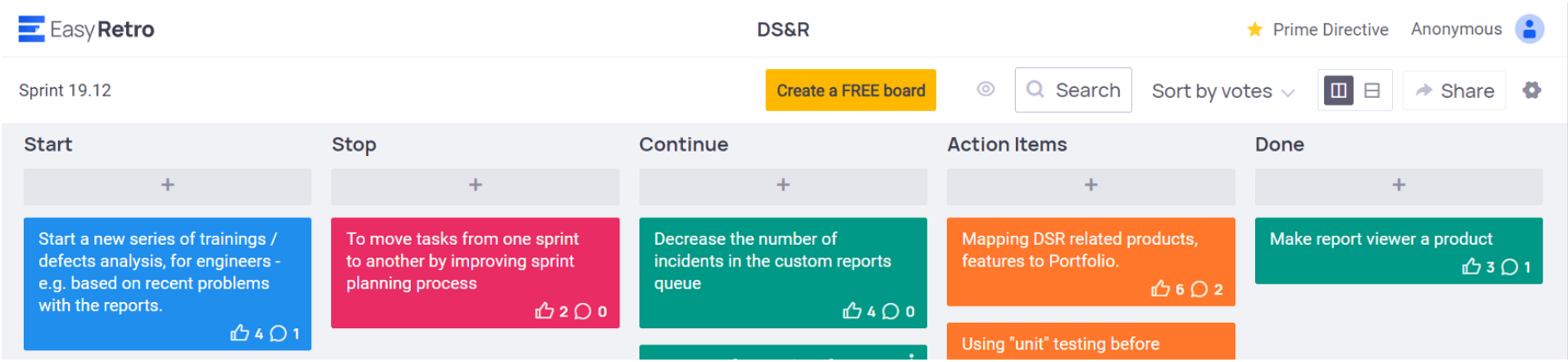
Lisa 2. Sprindi tahvli vaade



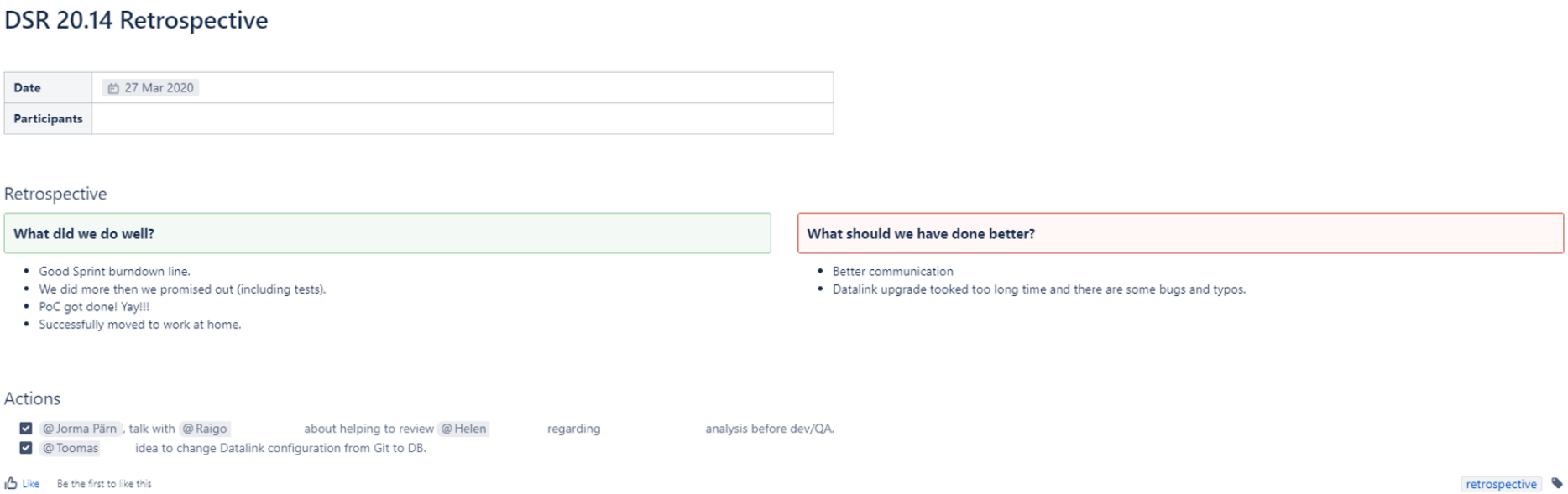
Lisa 3. Eelnevate sprintide planeeritud SP-d, skoobi muudatused ning sprindi lõpuks teostatud SP-d aastal 2021.

Sprint	Planeeritud SP	Lisatud SP	Eemaldatud SP	Kokku skoobi muudatus	Kokku sprindis SP	Töö edenemine (*burndown*)
21.4	149	115	0	115	264	256
21.6	92	62	4	58	154	142
21.8	70	57	4	53	123	123
21.10	123	91	3	88	211	173
21.12	111	67	2	65	176	159
21.14	96	73	9	64	160	160
21.16	104	48	1	47	151	148
21.18	44	51	1	50	94	94
21.20	81	43	6	37	118	118
21.22	90	73	5	68	158	158
21.24	103	55	8	47	150	150
21.26	108	57	44	13	121	121
21.28	181	87	192	-105	76	76

Lisa 4. EasyRetro tahvli vaade



Lisa 5. Retrospektiivi lehe vaade



DSR 20.14 Retrospective

Date	27 Mar 2020
Participants	

Retrospective

What did we do well?

- Good Sprint burndown line.
- We did more then we promised out (including tests).
- PoC got done! Yay!!!
- Successfully moved to work at home.

What should we have done better?

- Better communication
- Datalink upgrade tooked too long time and there are some bugs and typos.

Actions

- [x] @Jorma Pärn , talk with @Raigo about helping to review @Helen regarding analysis before dev/QA.
- [x] @Toomas idea to change Datalink configuration from Git to DB.

Like Be the first to like this

retrospective

Litsents

Mina, Katre Siilivask,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose „**Scrum raamistik ja selle rakendamine Playtech Estonia OÜ osakonna näitel“,** mille juhendajad on **Anne Villems**, **Vambola Leping** ja **Ivo Tamm** reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.
2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.
3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.
4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Katre Siilivask

04.08.2021