TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

OSKU HÄMÄLÄINEN

# OHJELMISTOJÄRJESTELMÄN SUORITUSKYVYN HALLINTA-PROSESSIN KEHITTÄMINEN

Diplomityö

Tarkastajat: professori Hannu Jaakkola ja apulaisprofessori (tenure track) Sami Hyrynsalmi
Tarkastajat ja aihe hyväksytty Talouden ja rakentamisen tiedekunnassa 24. helmikuuta 2017

# TIIVISTELMÄ

**OSKU HÄMÄLÄINEN**: Ohjelmistojärjestelmän suorituskyvyn hallintaprosessin kehittäminen
Tampereen teknillinen yliopisto
Diplomityö, 50 sivua, 4 liitesivua
Toukokuu 2017
Johtamisen ja tietotekniikan DI-tutkinto-ohjelma, Pori
Pääaine: Ohjelmistotuotanto ja tiedonhallinta
Tarkastajat: Professori Hannu Jaakkola ja apulaisprofessori (tenure track) Sami Hyrynsalmi

Avainsanat: suorituskyvyn hallinta, ohjelmiston suorituskyky

Ohjelmistojärjestelmän suorituskyky voi jäädä vähälle huomiolle ohjelmistokehityksessä, mikäli järjestelmän suorituskyky koetaan riittäväksi. Ohjelmistojärjestelmän suorituskykyä tulisi kuitenkin hallita läpi ohjelmiston elinkaaren aina uuden tuotteen arkkitehtuurisuunnittelusta ohjelmiston ylläpitoon. Suorituskyvyn hallinnan laiminlyönti voi johtaa siihen, että järjestelmän suorituskyvyn osatekijöitä ei täysin tunneta. Se puolestaan vaikeuttaa suorituskyvyn kehittämistä tarvittaessa.

Tutkielma käsittelee laajan jo käytössä olevan ohjelmistojärjestelmän suorituskyvyn hallintaa. Tutkielmassa pyrittiin löytämään uusia keinoja järjestelmän suorituskyvyn tunnistamiseen ohjelmistoprojektin aikaisemmassa vaiheessa. Tutkimusmetodina käytettiin laadullista toimintatutkimusta ja aineistonkeruumenetelminä osallistuvaa havainnointia ja haastatteluja. Järjestelmän suorituskyvyn hallinnan haasteita tarkasteltiin laaja-alaisesti kattaen ohjelmistokehitysorganisaation, sen toimintamallit ja suorituskykytehtävien käytännön haasteet. Tutkija toimi samalla järjestelmän suorituskyvystä vastaavana projektijohtajana.

Tutkielman keskeisin havainto oli se, että suorituskyvyn hallinnan puutteiden taustalla oli suorituskyvyn matala prioriteetti tuotekehitysorganisaatiossa ja heikko suorituskyvyn hallintaprosessin noudattaminen. Tämän seurauksena oli muodostunut käytäntö, jossa järjestelmän suorituskyky lähinnä todettiin järjestelmätasolla ohjelmistoprojektin lopussa. Järjestelmän suorituskykyä ei systemaattisesti suunniteltu ja kehitetty ohjelmistoprojektin aikana. Työn päätuloksena syntyi suorituskyvyn hallinnan prosessikonsepti, jonka perustana on suorituskyvyn jatkuva analyysi ohjelmistokomponenttitasolla. Jatkuvalla analyysillä tavoitellaan sitä, että ohjelmistokehittäjä saa lähes reaaliaikaisen tiedon komponentin suorituskyvystä ohjelmistokoodimuutoksen jälkeen.

# ABSTRACT

**OSKU HÄMÄLÄINEN**: Development of performance management process for SW system
Tampere University of Technology
Master of Science Thesis, 50 pages, 4 Appendix pages
May 2017
Master's Degree Programme in Management and Information Technology, MSc (Tech), Pori
Major: Software Engineering and Data Management
Examiner: Professor Hannu Jaakkola and Assistant Professor (tenure track) Sami Hyrynsalmi

Keywords: performance engineering, performance management, software performance

If the current performance of a software system already meets end user expectations, it might not get enough attention for long-term needs. However, the performance of a software system should remain on the agenda actively throughout the product development lifecycle, from the architectural design of a new system until the end of maintenance. Neglecting proper performance management may lead to a state where factors impacting software performance are not known. This can make performance optimization at a later stage very difficult and laborious.

This thesis deals with the performance engineering of a large software system. The objective of the thesis was to find new methods for identifying system performance at an earlier stage of the software project. The study was done by using participatory action research as a research method and by conducting a number of interviews among members of software development teams. Challenges with performance engineering were identified widely, covering the software development organization, its way of working, and the actual performance engineering tasks. The author of the thesis had overall execution responsibility of performance engineering in the organization during the time of study.

The main finding was that performance engineering problems were more related to the low priority of performance and to inadequate follow-up of the process than to the performance management process itself. The performance of the system tended to be just observed at the end of the software project in the final performance validation phase. System performance was not systematically designed and developed as an integral part of a software project. As the main result of the study, a new performance management process concept was created. Its foundation is continuous performance analysis on software component level. In an ideal situation, continuous performance analysis would be a part of continuous integration, providing almost real-time feedback about the performance of the software component to the designer directly after program code modification.

# ALKUSANAT

Tämän tutkielman tutkimus osuus on syntynyt osana päivittäistä ohjelmistokehitystyötä tutkijan oman vastuualueen puitteissa. Tutkielman tekoa ohjasi tietotekniikan professori Hannu Jaakkola.

Haluan kiittää professori Hannu Jaakkolaa mainiosta työn ohjauksesta työn eri vaiheissa. Diplomityöni tarkastuksesta haluan osoittaa kiitokset professori Jaakkolalle ja apulaisprofessori Sami Hyrynsalmelle. Tutkielman viimeistelyn osalta osoitan kiitokset kollegoilleni Kirsi Korhoselle, Maria Lahdelle ja Markus Nenoselle sekä oikoluvun osalta Virpi Kontiselle.

Kiitos Karoliinalle ja lapsilleni tuesta ja kärsivällisyydestä tutkielman kirjoitustyön aikana.

Tampereella, 10.4.2017

Osku Hämäläinen

# SISÄLLYSLUETTELO

# 1. JOHDANTO

Työssä kehitettiin olemassa olevan ohjelmistojärjestelmän suorituskyvyn hallintaprosessia. Päätavoite oli löytää keinoja, joilla tuotteen suorituskyky tunnistetaan ja hallitaan ohjelmistokehityksen aikana. Aiemmin järjestelmän suorituskyky oli pääosin tyydytty todentamaan järjestelmätason testauksella ohjelmistokehityksen loppuvaiheessa. Tästä aiheutuu riski ohjelmistoprojektien valmistumiselle, koska järjestelmätestauksessa löydettyjen vikojen korjaus on kallista ja ne voivat samalla vaarantaa ohjelmistoprojektin valmistumisen aikataulussaan

Tarve järjestelmän paremmalle suorituskyvyn hallinnalle oli seurausta asiakkaiden käytössä olevien järjestelmien kasvavasta kuormituksesta sekä vaatimuksista kasvattaa edelleen niiden tarjoamaa kapasiteettia. Järjestelmän suorituskyvyn kehittämisen edellytyksenä on ymmärrys järjestelmän suorituskykyyn vaikuttavista tekijöistä. Tunnettaessa järjestelmän yksittäisten osien suorituskyky ja niiden vaikutus koko järjestelmän suorituskykyyn voidaan parantaa niitä osia, jotka rajoittavat järjestelmän suorituskykyä. Toisin sanoen, poistamalla suorituskyvyn pullonkaulat saadaan kasvatettua koko järjestelmän suorituskykyä. Järjestelmien kasvanut kuormitus on paljastanut myös piileviä ohjelmistovikoja, jotka esiintyvät vain ison kuormituksen alla.

Työssä kohteena olevan ohjelmistojärjestelmän suorituskyvyn hallinnan vaikeutta lisäävät järjestelmän laajuus, sen arkkitehtuuri sekä maantieteellisesti hajaantunut suuri tuotekehitysorganisaatio. Ohjelmistojärjestelmä koostuu erilaisista ohjelmistokomponenteista, joita ei voi verrata suoraan keskenään suorituskyvyn näkökulmasta. Tämän vuoksi työssä pyrittiin löytämään keinoja ja menetelmiä, jotka soveltuvat mahdollisimman monen komponentin kehitykseen ja tukevat järjestelmätason suorituskyvyn hallintaa.

Työssä esitellään aluksi ylätasolla tutkimuksen kohteena oleva järjestelmä, sitä kehittävä organisaatio sekä sen käytössä oleva suorituskyvynhallinta prosessi. Työn viitekehyksenä on suorituskyvyn hallinta, jonka teoria esitellään erityisesti ohjelmiston evoluution ja ohjelmistokehitysprosessien näkökulmasta. Varsinainen työ toteutettiin toimintatutkimuksena kahdella tutkimuskierroksella. Molempien tutkimuskierrosten pohjalta luotiin uusia käytäntöjä, joiden onnistumista arvioitiin erikseen. Työn voi nähdä myös organisaation oppimistarinana suorituskyvyn hallinnan maailmaan. Organisaation oppiminen jatkuu kiihtyvällä vauhdilla tämän työn havaintojen, haastattelujen ja päätelmien pohjalta.

## 1.1 Tutkimusongelma ja -kysymykset

Tutkielman lähtökohtana oli olemassa oleva suorituskyvyn hallintaprosessi, joka oli erittäin testauspainotteinen. Tutkittava organisaatio oli havainnut jo tutkimusta edeltävänä aikana, että suorituskykyä ei juuri huomioitu ohjelmistokehityksen aikana. Organisaatiossa tiedostettiin yleisesti, että suorituskyky on laiteresurssien, ohjelmistoarkkitehtuurin ja yksittäisien toteutusratkaisujen summa – ei testauksen lopputulos. Organisaation vakiintuneet käytännöt eivät kuitenkaan vastanneet tätä havaintoa. Toimintamallin seurauksena järjestelmätestausvaiheessa kohdattiin suorituskykyongelmia, joiden korjaus aiheutti muun muassa aikatauluhaasteita. Vastaavasti järjestelmän suorituskyvyn ja sen tarjoaman kapasiteetin parantaminen oli ongelmallista sen kehitystä tukevien vakiintuneiden käytäntöjen puuttuessa.

Tutkimusongelma johdettiin tästä edelleen seuraavaan muotoon: miten suorituskyky ja sen ongelmat voidaan tunnistaa aiemmin standardeihin pohjautuvilla ratkaisuilla huomioiden sekä järjestelmän arkkitehtuuri että kehitysorganisaation sovitut toimintatavat? Tutkimuskysymyksien avulla pyrittiin tarkemmin ymmärtämään tutkittavaa ongelmaa ja löytämään ratkaisuja itse ongelmaan. Millaisia ongelmia nykyprosessissa on, ja mitkä seikat mahdollisesti estävät sen noudattamista? Onko organisaatiolla tarvittavaa osaamista prosessin noudattamiseksi? Miten prosessia tulisi kehittää, jotta suorituskykyongelmat tunnistetaan ja ratkaistaan aiemmin?

## 1.2 Hypoteesi

Tutkimustavoitteen mukaisesti asetettu hypoteesi oli, että suorituskykyongelmat voidaan havaita aiemmin kehittämällä edelleen suorituskyvyn hallintaprosessia ja huolehtimalla sen noudattamisesta.

Hypoteesista johdettiin alihypoteesit seuraavasti:

- Yleisesti ottaen, ohjelmistokehitystiimeillä ei ole tarvittavaa osaamista suorituskyvyn hallintaprosessin noudattamiseen.
- Tuotekehitysorganisaation toimintatavoissa ei ymmärretä tarpeeksi hyvin suorituskykyvaatimusten ei-toiminnallista luonnetta, jolloin niitä on pyritty hallitsemaan samoin keinoin kuin toiminnallisia vaatimuksia.

Ensimmäinen alihypoteesi perustui arvioon, että ohjelmistokehittäjien osaaminen on todennäköisesti suurempi ongelma kuin olemassa oleva suorituskyvyn hallintaprosessi. Mikäli tämä olisi totta, varsinainen ongelma ei ratkeaisi itse prosessia muuttamalla. Toinen

alihypoteesi perustui arvioon, että ohjelmistojen suorituskyvyn luonnetta ei täysin ymmärretty. Hypoteesin ollessa tosi, suurin tarve kohdistuisi suorituskykyosaamisen kasvatukseen.

## 1.3 Tutkielman rakenne

Luvussa 2 esitetään tutkittava organisaatio ja sen käyttämä suorituskyvyn hallintaprosessi. Luvussa esitetään myös tutkimusmetodina käytetty toimintatutkimus ja sen soveltaminen tutkielmassa. Luvussa 3 esitetään sekä suorituskyvyn hallinnan teoria että teorian suhde tutkimustyöhön. Itse tutkimustyö on tehty kahdessa iteraatiokierroksessa, jotka esitetään luvuissa 4 ja 5. Näiden lukujen alakohtien rakenne noudattaa alakohdassa 2.1 esitetyn toimintatutkimuksen vaiheita. Luvussa 6 esitetään tulokset ja tutkijan loppupäätelmät.

# 2. TUTKIMUKSEN JA KOHTEEN KUVAUS

Tutkielmassa tarkastellaan tuotekehitysorganisaatiota ja sen suorituskyvyn hallintaprosessia kokonaisuutena. Organisaatio on erikoistunut kehittämään ja ylläpitämään yhtä ohjelmistojärjestelmätuotetta.

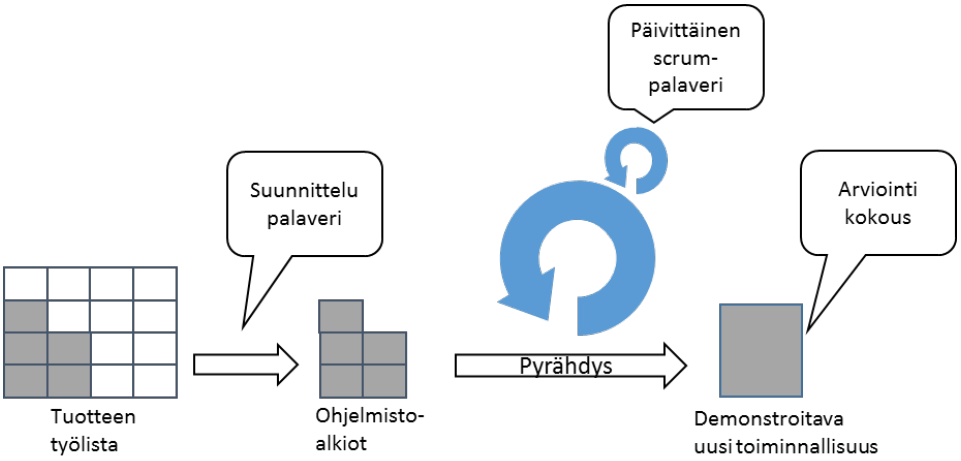
## 2.1 Ohjelmistokehitysorganisaatio ja sen toimintatavat

Organisaation ohjelmistokehitys perustuu ketteriin (engl. agile) kehitysmalleihin, joiden juuret ovat vuonna 2001 julkaistussa Agile-manifestissa. Tässä varsin yleisesti tunnetussa manifestissa korostetaan yksilöiden ja toimivan ohjelmiston roolia raskaiden prosessien ja dokumentaation sijasta. Toimivan ohjelmiston rinnalla asiakkaan tavoitteiden täyttäminen ja kyky muutoksiin ohjelmistoprojektin aikana ovat onnistuneen projektin edellytyksiä. (Agile 2001). Manifestissa on listattu kuitenkin periaatteita, joiden täydellinen toteuttaminen on käytännössä mahdotonta tutkittavassa organisaatiossa. Organisaatio on hajaantunut maantieteellisesti useisiin paikkoihin, ja tekijöiden ohjelmistokehitysosaaminen vaihtelee eri toimipisteiden välillä. Tällöin periaatteiden noudattaminen, kuten kehitysrooleissa olevien työntekijöiden tulee työskennellä yhdessä päivittäin ja tehokkain kommunikaatio on kehitystiimissä keskustelu kasvokkain, ei onnistu.

Organisaation ohjelmistokehitys on jaettu ohjelmistokehitystiimeihin, joista kukin vastaa tietyistä ohjelmistokomponenteista. Tiimin vastuu kattaa komponentin koko ohjelmistokehityksen ohjelmistomäärittelystä aina testaukseen asti mukaan luettuna komponentin sisäinen arkkitehtuuri. Ohjelmistokehitystiimien lisäksi on järjestelmätason tiimejä, joiden vastuulla on joko ohjelmiston toiminnallisia tai ei-toiminnallisia vaatimuksia. Näitä järjestelmätasontiimejä ovat esimerkiksi arkkitehtuuri, järjestelmän toiminnallinen verifiointi ja suorituskyky (engl. performance). Näistä jälkimmäistä tiimiä kutsutaan järjestelmän suorituskykytiimiksi. Tiimin tehtävä on järjestelmän kapasiteetti- ja suorituskykytavoitteiden asettaminen sekä niiden validoiminen. Organisaation toimintamallissa on kuitenkin sovittu, että ohjelmistokomponenttivastuu pitää sisällään myös komponentin suorituskyvyn (Nagy 2017). Ohjelmistokehityksen jakaminen ohjelmistokehitystiimeihin noudattelee hyvin Agile-manifestin periaatetta, jonka mukaan parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itseohjautuvissa tiimeissä (Agile 2001).

Organisaation ohjelmistokehitystä ohjataan ketteriin menetelmiin kuuluvalla Scrum-prosessilla (Kuva 1). Se on nykyisin laajasti käytetty ketterä menetelmä, jossa ohjelmistokehitystä tehdään lyhyissä pyrähdyksissä. Yksittäisen pyrähdyksen johtamiseen kuuluvat pyrähdyksen suunnittelukokous ja scrum-mestarin pitämä päivittäinen scrum-kokous

sekä pyrähdyksen lopussa katselmointikokous ja arviointikokous (Haikala ja Mikkonen 2011). Pyrähdyksessä toteutettavat ohjelmistoalkiot valitaan pyrähdyksen suunnittelupalaverissa tuotteenomistajan ja Scrum-tiimin yhteistyönä tuotteen työlistalta, jonka toteutettavien alkioiden priorisoinnista vastaa tuotteenomistaja (Haikala ja Mikkonen. 2011). Tutkitun organisaation pyrähdyksien kesto vaihtelee kahdesta kolmeen viikkoon, jotta pyrähdykset saadaan tahdistettua kalenterikuukausien kanssa. Ohjelmistokehitystiimit toimivat samassa pyrähdyssyklissä.




***Kuva 1.*** *Scrum-prosessi*

Komponenttitiimien yhteinen kehityssykli on tärkeää, koska ohjelmistopäivitykset toimitetaan järjestelmätason julkaisuina. Yleisesti julkaisujen välillä on useita pyrähdyksiä. Järjestelmätason testaus mukaan lukien suorituskyvyn validointi tapahtuu ennen uuden ohjelmistojulkaisun toimitusta asiakkaille. Järjestelmän suorituskykyä ei testata erikseen ohjelmistopyrähdyksille järjestelmätasolla. Jatkuvalla integroinnilla pyritään varmistamaan järjestelmän jatkuva toimivuus, jotta mahdolliset viat löydetään jo ennen järjestelmätestausvaihetta. Haikalan ja Mikkosen (2011) mukaan jatkuvassa integraatiossa ohjelmistomuutos tallennetaan versionhallintaa mahdollisimman usein ja ohjelmistosta rakennetaan uusia versioita jatkuvasti. Näin muutokset pysyvät pieninä ja toteuttajat saavat palautteet mahdollisimman nopeasti.

## 2.2 Suorituskyvyn hallintaprosessi tutkimuksen alussa

Ohjelmiston suorituskyvyn hallintaprosessi on määritelty yritystasolla, ja se on ohjelmistotuotteesta sekä ohjelmistokehitysmallista riippumaton. Prosessissa korostuvat suorituskykyvaatimusten määrittely ja suorituskykyvaatimusten validointi. Siinä ei oteta kantaa, miten suorituskykyä tulisi hallita ohjelmiston toteutusvaiheessa. Se ei myöskään kuvaa millään tavoin ohjelmistokomponentti- ja järjestelmätason vastuita ja rooleja. Prosessin mukaan suorituskykytestaus ja tuotteen vakaus (engl. stability) ovat julkaisun keskeisiä

laatukriteerejä. Ohjelmistokomponenttitasolla ei ollut vastaavia laatukriteerejä. (Kuivalainen 2014)

Tutkittavassa tuotteessa oli määritelty järjestelmätason kapasiteetti- ja suorituskykyvaatimukset, jotka verifioitiin järjestelmätason suorituskykytestauksessa. Järjestelmän suorituskykyvaatimukset oli luotu yhdessä tärkeimpien komponenttitiimien kanssa ja niitä ylläpidettiin järjestelmätason suorituskykytiimin voimin yli ohjelmistojulkaisujen. Komponenttitasolle oli asetettu ainoastaan vaatimus komponenttikohtaisesta suorituskyvyn testauksesta. Järjestelmätason suorituskykytiimi pyrki ohjaamaan, käytettävissä olevien henkilöresurssien sen salliessa, komponenttitiimien suorituskykytestausta katselmoimalla niiden testisuunnitelmat ja -raportit sekä tarjoamalla tarvittavat testausympäristöt. Testisuunnitelma ja -raportti oli standardoitu mallidokumentin ja ohjeiden avulla. Käytännössä komponenttikohtainen suorituskykytestaus perustui pitkälti samoihin menetelmiin ja työkaluihin kuin järjestelmätason testaus. (Durairaj 2013)

Järjestelmätason suorituskykytiimin arkkitehtien mukaan ohjelmistojärjestelmän suorituskyvyn ennakointi asiakaskohtaisessa toimintaympäristössä oli vaikeaa. Järjestelmien suuruuden ja käytössä olevien testausresurssien vuoksi kaikkia mahdollisia konfiguraatioita ja niiden suorituskykyä ei voitu todentaa testausympäristöissä. Asiakastoimituksissa riittävä kapasiteetti varmistettiin mitoittamalla laiteresurssit ja järjestelmän antama kapasiteetti siten, että järjestelmä varmasti selviytyy toimintaympäristössään. Toisaalta suurimman laitekonfiguraation kapasiteettiraja asetettiin merkittävästi todellista tasoa alemmaksi, jotta vältetään asiakkaan tuotantokäytössä olevan järjestelmän antaman kapasiteetin loppuminen.

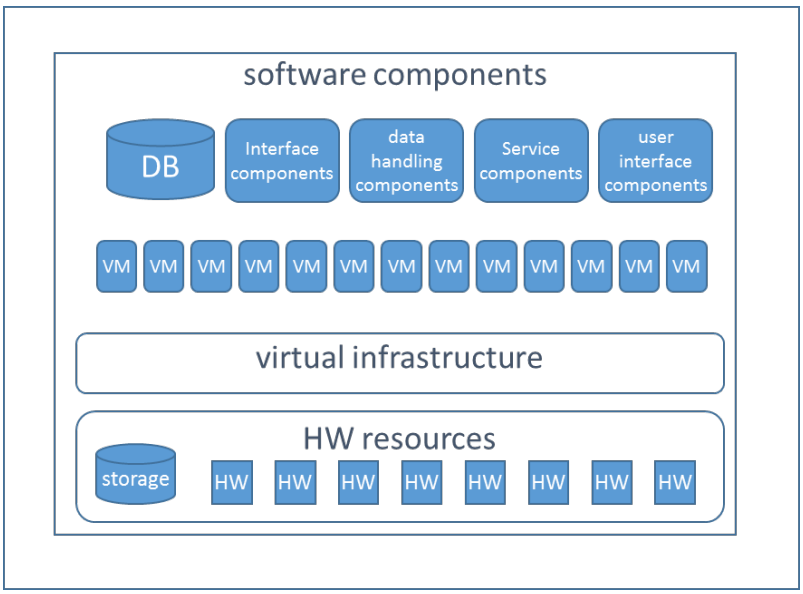
## 2.3 Tuotteen yleiskuvaus

Tutkittu ohjelmistojärjestelmä on osajärjestelmä laajasta tietoliikennejärjestelmästä, joka tyypillisesti kattaa laajan maantieteellisen alueen. Tutkitun ohjelmistojärjestelmän kuormitus on riippuvainen tietoliikenneverkon koosta, sen muista laitteista, sen välittämästä liikenteestä ja ylläpidon käyttäjäoperaatioista. Juuri toimintaympäristö asettaa vaatimukset palveluiden korkealle saavutettavuudelle (engl. high availability) sekä järjestelmän suorituskyvylle.

Ohjelmistolla on pitkä historia aina 1990-luvulta saakka. Noista ajoista järjestelmä on kasvanut valtavasti osana tietoliikenteen kasvua. Verkkojen kasvu on lisännyt myös verkon monimutkaisuutta, joka heijastuu myös tutkittavaan ohjelmistojärjestelmään. Järjestelmään on toteutettu uusia toiminnallisuuksia, mutta ohjelmiston ylätason arkkitehtuuriratkaisut periytyvät aina 1990-luvulta. Kun samaan aikaan järjestelmä- ja ohjelmistosuun-

nittelijat ovat vaihtuneet useaan kertaan, ohjelmiston kaikkien toteutusratkaisujen valintaperusteet eivät ole täysin selvillä. Samoista syistä ohjelmistokomponenttien välisien rajapintojen määrittely ja niiden käyttö eivät ole tarkasti tiedossa. Suorituskyvyn näkökulmasta tämä on erittäin ongelmallista, koska rajapintojen ja niiden kautta ohjelmistokomponentille tulevien tehtävien tunteminen on tärkeää suorituskyvyn mallinnuksessa ja analysoinnissa. Suorituskyvyn hallintaa esitellään tarkemmin luvussa 3.

Ohjelmistojärjestelmä koostuu ohjelmistoalustasta, joka tarjoaa keskitetyt peruspalvelut ohjelmistokomponenteille. Järjestelmä on erittäin tietointensiivinen, minkä vuoksi ohjelmistojärjestelmän tietokanta on kriittisessä roolissa. Käytännössä lähes kaikki ohjelmiston toteuttamat käyttötapaukset aiheuttavat tietokannan kirjoitus- tai lukuoperaation. Ohjelmistokomponentit jakautuvat kuvan 2 mukaisesti karkeasti jaotellen järjestelmän rajapintakomponentteihin, tietoa jalostaviin komponentteihin, palveluja toteuttaviin komponentteihin ja käyttöliittymäkomponentteihin. Tietoa jalostavat komponentit muokkaavat järjestelmärajapintojen kautta tulevaa tietoa ja tallentavat sen tietokantaan. Palveluja toteuttavat komponentit käyttävät tietokannan dataa ja muokkaavat sitä käyttäjän tarpeisiin. Tietokannassa on myös dynaamista tietoa kuten konfiguraatioparametreja, joiden käsittelyyn on erikoistunut osa palveluja toteuttavista komponenteista.



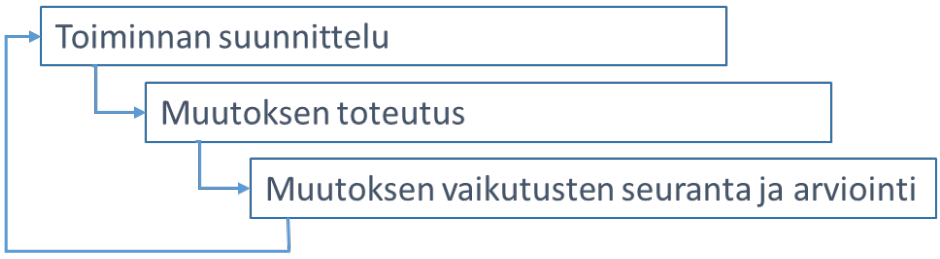

***Kuva 2.*** *Ohjelmistojärjestelmän yleiskuva*

Ohjelmiston suoritusympäristö on virtualisoitu, eli virtuaalinen infrastruktuuri peittää fyysiset laiteresurssit. Virtualisointi mahdollistaa laiteresurssien tehokkaamman käytön,

koska laiteresursseja ei tarvitse mitoittaa kiinteästi ohjelmistokomponenteille niiden suurimman arvioidun kuorman perusteella. Virtualisoinnin avulla komponenteille allokoidaan virtuaalisia laiteresursseja komponentin kunkin hetken todellisen tarpeen mukaan (VMware 2017). Ohjelmisto on myös hajautettu useille virtuaalikoneille. Yksittäinen virtuaalikone voi siten palvella useita ohjelmistokomponentteja tai yksittäinen käyttötapaus voi kuormittaa ohjelmistokomponentteja useissa virtuaalikoneissa. Haittapuolena on, että virtualisointi ja erilliset virtuaalikoneet vaikeuttavat virtuaalisten ja fyysisten resurssien kulutuksen analysointia.

## 2.4 Toimintatutkimus ja sen menetelmät

Tutkimusmetodina on käytetty toimintatutkimusta, joka luokitellaan laadullisen tutkimuksen alle. Toimintatutkimus metodina on menettelytapa, jota yleisesti käytetään parantamaan edellytyksiä ja käytäntöjä ihmisten keskinäisissä toiminnoissa. Toimintatutkimuksen vahvuus on sen keskittyminen todellisiin käytännön ongelmiin. Samalla se sitoo sitä hyödyntävät osapuolet tutkimukseen ja siinä toteutettaviin toimenpiteisiin. Toimintatutkimus soveltuu hyvin prosessien kehittämiseen, ja siinä tutkija on usein osa tutkittavaa organisaatiota. Toimintatutkimukselle on olennaista tutkimuksen toistuvuus, jolloin iteraatiokierroksen lopussa arvioidaan toimenpiteiden onnistumista ja jatketaan eteenpäin uuteen kierrokseen. Tällöin myös tutkimusongelma ja toimenpiteet tarkentuvat työn aikana ja tutkimus etenee epämääräisemmästä kohti tarkempaa. (Dick 1993)




***Kuva 3.*** *Toimintatutkimuksen vaiheet*

Toimintatutkimuksen iteraatiokierroksen vaiheet ovat toiminnan suunnittelu, muutoksen toteutus, muutosten vaikutusten seuranta ja arviointi (Kuva 3). Keskeisenä ajatuksena on pyrkimys muuttaa toimintaa ja sitä kautta ymmärtää ihmisten toimintaa sosiaalisessa tilanteessa. Suunnitteluvaiheessa suunnitellaan toimenpiteet, joilla muutos pyritään toteuttamaan. Vaikutusten seuranta- ja arviointivaiheessa tarkastellaan tapahtunutta muutosta,

josta voidaan jatkaa edelleen uudelle iteraatiokierrokselle. (Saaranen-Kauppinen ja Kauppinen 2006)

Toimintatutkimuksen laadullisen luonteen vuoksi tiedonkeruunmenetelminä voidaan käyttää muun muassa havainnointia, haastatteluja ja valmiita aineistoja ja dokumentteja. Verrattuna määrälliseen tutkimukseen laadullisessa tutkimuksessa on aina vaarana tutkijan oman näkemyksen sekoittuminen varsinaisen tutkimusaineiston kanssa. Osallistuvassa tutkimuksessa tämä ongelmaa korostuu, kun menetelminä käytetään havainnointia ja strukturoimatonta haastattelua. (Järvenpää 2006)

**Havainnoissa** tutkija toimii yksilöiden ja ryhmien kanssa yhteistyössä havainnoiden ilmiöitä ja käyttäytymistä. Tutkijan roolista riippuen hänen osallistumisensa aste vaihtelee täysin tutkimusryhmään osallistuvasta täysin ulkopuoliseen havainnoitsijaan. Täysin osallistuvan tutkijan tapauksessa on vaarana, että tutkija itse häiritsee tutkittavaa tilannetta. (Havainnointi voi olla strukturoitu tai ei-strukturoitu). **Haastattelu** on tutkijan ja haastateltavien välinen keskustelu, jonka järjestelmällisyys voi vaihdella hyvin järjestelmällisestä vähemmän järjestelmällisempään riippuen siitä kuinka paljon liikkumavaraa on tehtyjen kysymysten ja vastausmahdollisuuksien välillä. Haastattelu voi seurata tiukasti etukäteen valmisteltua lomaketta kysymyksineen ja vastauksineen tai se voi olla avoin keskustelu. **Valmiilla aineistoilla** tarkoitetaan alan kirjallisuutta, aiempia tutkimuksia ja niiden tuloksia sekä erilaisia asiakirjoja. Asiakirjoiksi luetaan myös tutkittavan organisaation dokumentit ja pöytäkirjat. (Saaranen-Kauppinen ja Kauppinen 2006)

## 2.5 Toimintatutkimuksen soveltaminen tässä tutkimuksessa

Työssä tarkasteltiin tutkitun organisaation suorituskyvyn hallintaprosessia sekä valittuja kehitystoimenpiteitä. Kehittämistoimenpiteet aiheuttivat muutoksia käytäntöihin ja rooleihin, jotka vaativat hyväksyntää tuotekehitysorganisaatiolta. Uusien käytäntöjen läpivienti vaati muutoksen johtamista, joka toteutui tutkittavan organisaation tavanomaisten johtamismenetelmien kautta (esim. kvartaalikohtaiset kehityssuunnitelmat). Työssä ei ole tarkasteltu erikseen muutosjohtamisen teoriaa ja sen haasteita.

Tutkimustyö toteutettiin kahdella toisiaan seuraavalla iteraatiokierroksella (Taulukko 1). Ensimmäisen iteraatiokierroksen päämääränä oli ymmärtää nykyisen prosessin ongelmakohdat ja tunnistaa avainkehityskohteet yhdessä suorituskyvystä vastaavan järjestelmätason tiimin kanssa. Toisella tutkimuskierroksella päämäärät pysyivät muutoin samoina, mutta tutkimus laajeni kattamaan koko tuotekehitysorganisaation. Työn edetessä myös tutkimusongelma tarkentui ymmärryksen kasvaessa.

***Taulukko 1.*** *Tutkimuksen päävaiheet, menetelmät ja päämetriikka*

| **Iteraatiokierros** (laajuus) | **Menetelmät tärkeysjärjestyksessä** | **Avainmittarit** | **Ajankohta** |
|---|---|---|---|
| **1.kierros** (suorituskykytiimi) | havainnointi, haastattelut, työpajat | suorituskykytestauksen laajuus | havainnot 9-12/2014<br>toimenpiteet 1-5/2015<br>arviointi 6/2015 |
| **2.kierros** (koko tuotekehitys) | haastattelut, havainnointi, auditointi | suorituskykytestauksen laajuus, suorituskykymittarit, prosessintuntemus | havainnot 8-10/2015<br>toimenpiteet 1-8/2016<br>arviointi 10/2016 |

Ensimmäisessä iteraatiokierroksessa käytettiin tutkimusmenetelmänä pääosin osallistuvaa havainnointia, avoimia keskusteluita sekä erillisiä työpajoja (engl. workshop). Tutkimustyö alkoi keskusteluilla suorituskyvystä vastaavan tiimin eri jäsenten kanssa. Keskustelujen kautta tutustuttiin tutkittavaan organisaatioon ja sen toimintatapoihin ilmaan määriteltyjä rajoja. Tutustumisvaiheen jälkeen työpajoja käytettiin ongelmien tarkempaan ymmärtämiseen sekä kehityskohteiden ideointiin. Rinnalla käytettiin kaiken aikaa tutkijan osallistuvaa havainnointia. Erityishuomio oli ymmärtää tuotteen historiaa ja tiedostaa, mitä kehitystoimenpiteitä oli jo aiemmin yritetty tehdä. Toimenpiteiden suunnittelu tehtiin yhteistyössä työpajoissa siten, että tutkijalla oli viimeinen sana toimenpiteiden valinnassa. Toimenpiteiden toteutukseen osallistuivat työpajoihin osallistuneet sekä useat muut suorituskykytiimin jäsenet. Valittujen toimenpiteiden onnistumista arvioitiin pääosin osallistuvan havainnoinnin mutta myös käytettyjen mittareiden kautta.

Toisessa iteraatiokierroksessa olivat haastattelut päätutkimusmenetelmänä. Haastatteluotos pyrittiin valitsemaan siten, että sillä saavutetaan mahdollisimman hyvä edustavuus komponenttitiimien osalta. Haastatteluissa pyrittiin puolistrukturoituihin ryhmähaastatteluihin etukäteen valmistelluilla tutkimuskysymyksillä. Tutkimuskysymykset oli määritelty tietoisesti keskustelun mahdollistajiksi ja suorituskyvyn hallinnan teemassa pysyttäjiksi. Useimmat haastattelut kääntyivät käytännössä enemmän teemahaastatteluiksi. Osallistuva havainnointi ja keskustelut olivat myös keinovalikoimassa. Toimenpiteet suunniteltiin yhteistyössä haastattelut toteuttaneen tiimin kanssa. Kuitenkin tutkijalla itsellään oli suurempi rooli toimenpiteiden valinnassa ja toteutuksessa kuin ensimmäisessä vaiheessa. Koska valittujen toimenpiteiden toteutus koski koko organisaatiota, toimenpiteiden onnistumista arvioitiin muodollisen TL9000:n sisäisen auditoinnin avulla.

Tutkimustyön tekijä oli samalla vastuussa tuotteen suorituskyvystä ja toimi projektijohtajana suorituskyvystä vastanneelle tiimille. Lisäksi tutkijalla oli kokonaisvastuu prosessin kehittämisestä ja vaadittujen toimenpiteiden toteuttamisesta päivittäisen vakiintuneen

suorituskykytyön johtamisen rinnalla. Tutkijan asema toisaalta varmisti tutkijan motivaation tutkimusaiheeseen ja toisaalta haastoi erottamaan omat mielipiteet ja tutkimushavainnot.

# 3. SUORITUSKYVYN HALLINNAN TEORIA

Ohjelmistojärjestelmien suorituskykyä luonnehditaan usein kyvyllä suorittaa määritelty joukko tehtäviä mahdollisimman nopeasti (läpäisykyky) ja nopealla vasteajalla (vastausaika) (Bondi 2015). Toisaalta suorituskykyä voidaan tarkastella minkä tahansa ohjelmiston ominaispiirteen osalta, jolle suoritusaika on mitattavissa (Smith ja Williams 2001).

Havaittuun suorituskykyyn vaikuttavat kaikki tehtävän käsittelyyn osallistuvat järjestelmän osat. Yksittäisen tehtävän suoritus muodostaa tietovirran ohjelmistojärjestelmässä. Tietovirran käsittelyyn osallistuvat järjestelmän osat ovat vuorovaikutuksessa keskenään ja vaikuttavat havaittuun suorituskykyyn. Bondin (2015) mukaan ohjelmistojärjestelmässä seuraavat osat ovat keskenään vuorovaikutuksessa:

- järjestelmäkomponenttien keskinäinen vuorovaikutus,
- laiteresurssien ja ohjelmiston vuorovaikutus,
- käyttäjärajapinnan vuorovaikutus järjestelmän muiden osien kanssa
- ja ohjelmistokomponenttien rajapintojen välinen vuorovaikutus.

Bondin (2015) mukaan järjestelmän suorituskyky on olennainen ohjelmistojärjestelmissä. Jos tuotteen suorituskyky on huono, se voi olla pahimmillaan kykenemätön suorittamaan sille annettuja tehtäviä. Huono suorituskyky voi vaikuttaa myös mahdollisiin ostajiin, jos suorituskyky ei vastaa käyttäjän odotuksia. Siten suorituskyky voi olla myös keskeinen kilpailutekijä. Suorituskyvyn tärkeyden vuoksi se on yksi keskeinen laatuominaisuus ohjelmistojärjestelmissä (QuEST Forum 2015).

Suorituskyvylle tulee määritellä tavoitteet, joita vasten valmis järjestelmä validoidaan (Smith ja Williams 2001). Ohjelmistojärjestelmien tavoitteet tulisi asettaa konkreettisesti mitattavilla vaatimuksilla. Suorituskykyvaatimukset luokitellaan ei-toiminnallisiin vaatimuksiin. Ei-toiminnallisien vaatimuksien alle luetaan ne ominaisuudet, joita ei voida määritellä suoraan ohjelmiston käyttäjille tarjoamina toimintoina (Sommerville 2011).

Suorituskykytavoitteiden saavuttaminen osana ohjelmistokehitystä vaatii suunnitelmallisuutta. Englannin kielessä tätä työtä kutsutaan ”performance engineering” -termillä, jolle ei ole suoraa suomenkielistä käännöstä. Tässä tutkielmassa sen on tulkittu käsittävän sekä suorituskyvyn hallintaa että kehitystä. Edellisellä voidaan tarkoittaa systemaattista prosessia suorituskykytavoitteista suorituskyvyn validoimiseen. Jälkimmäisellä voidaan tarkoittaa enemmänkin esimerkiksi menetelmiä ja matemaattisia malleja, joiden avulla pyritään mallintamaan ja parantamaan suorituskykyä. Tutkielman tavoitteiden vuoksi tutkimuksessa keskitytään nimenomaan suorituskyvyn hallintaan ja jätetään suorituskyvyn

kehitys vähemmälle huomiolle. Valittu määritelmä ei ole yksiselitteinen, koska esimerkiksi Tuovinen (2014) on kääntänyt ”performance engineering” -termin määritelmällä suorituskykyisten ohjelmistojen kehittäminen.

## 3.1 Suorituskyvyn hallinta

Käyttäjien suorituskykytarpeiden saavuttamiseksi on tärkeää määritellä selvät ja mitattavat suorituskykyvaatimukset mahdollisimman aikaisessa ohjelmistonkehitysvaiheessa. Suorituskykyvaatimukset ovat potentiaalisia vaikuttimia ohjaamaan järjestelmäarkkitehtuurisuunnittelua ja teknologiavalintoja. Ne määritellään usein jo osana toimitussopimusta myyjän ja asiakkaan välillä tai ne voivat tulla suoraan viranomaisvaatimuksista. Suorituskykyvaatimusten määrittelyvaiheessa joudutaan tekemään valintoja järjestelmän nopeuden ja sen kustannusten välillä. Lisäkustannukset voivat syntyä esimerkiksi tehokkaammasta tietokonelaitteistosta tai monimutkaisemmasta toteutuksesta. Vastaavasti kuluja voidaan karsia laskemalla suorituskyvyn vaatimustasoa mutta silti täyttämällä käyttäjän tarve. Esimerkiksi sallimalla tehtävän suoritusajan kasvu yhdestä sekunnista kahteen sekuntiin voi pienentää merkittävästi järjestelmän tietokoneresurssien tarvetta, ja vasteaika voi olla samalla täysin hyväksyttävä käyttäjälle. (Bondi 2015)

Tietokonejärjestelmän suorituskyvyn arviointia varten tulisi määritellä suorituskykymalleja jo vaatimusmäärittelyvaiheen yhteydessä. Suorituskykymallien avulla pyritään ennakoimaan järjestelmän kapasiteettia ja suoritusviiveitä. Niiden avulla voidaan ennakoida myös järjestelmän käytön muutosten vaikutuksia suorituskykyyn. Suorituskykyyn vaikuttavia muutoksia voivat olla esimerkiksi käyttötapahtumien määrän, laitekonfiguraation ja ajastussääntöjen muutos. (Bondi 2015). Ohjelmistokehityksen alkuvaiheessa malleilla voidaan arvioida esimerkiksi valittujen tapahtumien määrää valitussa aikayksikössä laitteistoresurssien tarpeen arvioimiseksi tai ylipäätään vaatimuksen yleistä toteutettavuutta. Suorituskykymalli voidaan jalostaa edelleen yksityiskohtaisemmaksi sitä mukaa, kun ohjelmiston yksityiskohdista saadaan lisää ja tarkempaa tietoa. Suorituskykymallin tulisi kuitenkin olla mahdollisimman yksinkertainen. (Smith ja Williams 2001)

Smith ja Williams (2001) jakavat suorituskykymallit edelleen ohjelma- ja järjestelmäsuoritusmalleihin. Ohjelmasuoritusmallilla tarkastellaan yksittäisen käyttötapauksen vasteaikaa ja sen vaatimia resursseja ilman muiden käyttötapausten vaikutusta järjestelmään. Koska muut käyttötapaukset eivät vaikuta resurssien käyttöön, ohjelmasuoritusmallin antama tulos on optimistinen. Mikäli suorituskyky on heikko jo tämän mallin mukaan, mallia ei ole tarvetta kehittää enempää, koska muiden tekijöiden huomioiminen vain edelleen heikentää suorituskykyä. Ohjelmistosuoritusmalleista saadaan tietoa dynaamisiin järjestelmätietomalleihin. Järjestelmäsuoritusmallissa huomioidaan muut käyttötapaukset ja

niiden vaatimat resurssit, jolloin suorituskykyarvio on realistisempi. Mallin avulla voidaan arvioida työkuorman muutoksen vaikutusta suorituskykyyn, ohjelmiston skaalautumista tulevaisuuden tarpeisiin, uuden ohjelmiston vaikutusta toisiin järjestelmiin ja järjestelmän pullonkauloja. Samalla sillä saadaan vertailukelpoista tietoa arvioitaessa laitteiston tai ohjelmistopäivitysten vaikutuksia suorituskykyyn. Mikäli järjestelmäsuoritusmalli ennakoi suorituskykyongelmia, ne tulisi ratkaista ennen ohjelmistokehityksen jatkamista. Muutoin mallia voidaan edelleen tarkentaa tiedon lisääntyessä. (Smith ja Williams 2001)

Suorituskykytestauksen tehtävä on varmistaa suorituskykyvaatimusten toteutuminen ennen ohjelmiston toimitusta asiakkaalle. Suorituskykymallit ohjaavat osaltaan testitapauksien valintaa, koska niiden tulisi edustaa suorituskyvylle merkityksellisiä käyttötapauksia. Suorituskykytestauksessa tulisi myös varmistaa, että järjestelmä käyttäytyy suorituskykymallien mukaisesti eri läpäisykyvyn arvoilla. Halutun läpäisykyvyn testaaminen vaatii puolestaan ohjelmistopohjaisia kuormageneraattoreita, joiden avulla luodaan määrä syötteitä järjestelmän työkuorman (joukon käyttötapauksia) simuloimiseksi. Kuormittamalla järjestelmää eri läpäisykyvyn arvoilla voidaan lisäksi tutkia järjestelmän kapasiteettirajoja, sen toimintaa ylikuormitustilanteessa tai vakautta jatkuvan kuormituksen alla. Todetut kapasiteettirajat antavat arvokasta tietoa järjestelmän pullonkauloista järjestelmäarkkitehdeille ja -suunnittelijoille. (Bondi 2015)

Suorituskykytestauksen tuloksien vertailussa suorituskykymallilla saatuihin arvioihin tulee kiinnittää huomiota mallin oikeellisuuden lisäksi resurssien käytön lineaarisuuteen. Vaikka järjestelmä vaikuttaisi toimivan normaalisti ja täyttävän suorituskykyvaatimukset, poikkeamat suorituskykymallista ennakoivat Bondin mukaan ongelmia liittyen kapasiteettiin, käytettävyyteen tai jopa luotettavuuteen. Tällaisia ilmiöitä voivat olla kumulatiivinen muistivarauksen kasvu tai suorituksen hetkellinen jumiutuminen. (Bondi 2015)

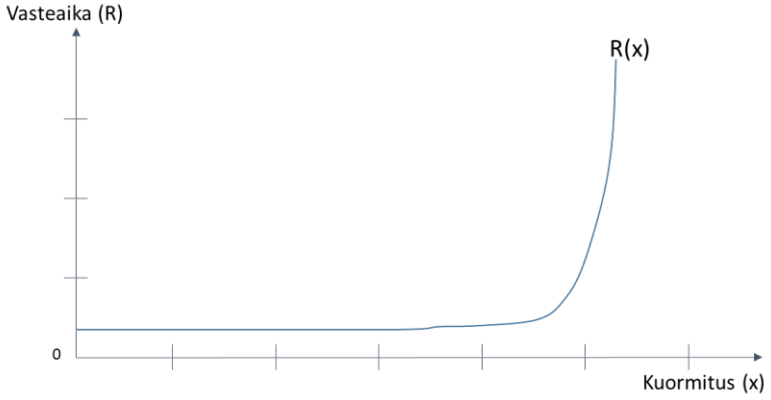
Testauksessa käytettävien työkuormien tulisi puolestaan edustaa mahdollisimman hyvin järjestelmän todellista käyttöä. Työkuorma voi olla tasaista tai olla muuttuvaa esimerkiksi vuorokauden ajan suhteen. Koska testauksessa ei voi käytännössä kattaa kaikkia mahdollisia työkuormien kombinaatioita, voidaan valita yksi tai useampi referenssityökuorma. Sillä saatuja mittaustuloksia voidaan käyttää myös asiakkaan vakuuttamisessa järjestelmän suorituskyvyn riittävyydestä asiakkaan tarpeisiin. (Bondi 2015)

## 3.2 Suorituskyvyn hallinta osana ohjelmiston evoluutiota

Ohjelmistojärjestelmän muuttuessa suorituskykymalleja tarvitaan arvioitaessa muutosten vaikutusta järjestelmän suorituskykyyn. Järjestelmän muutos voi olla seurausta esimer-

kiksi tarpeesta toteuttaa uusia käyttötapauksia, ohjelmistokomponentin päivityksestä (esimerkiksi ohjelmistoalustan toiminnallisuus) tai työkuorman kasvusta. Järjestelmän muutoksilla voi olla negatiivinen vaikutus havaittuun suorituskykyyn, jolloin aiemman suorituskykytason ylläpitäminen voi vaatia muutoksia järjestelmään. Esimerkiksi suuremman työkuorman käsittely suorituskykyvaatimuksien puitteissa voi vaatia lisää laiteresursseja. Suorituskykymallin avulla voidaan ennustaa näiden muutoksien vaikutuksia ja resurssitarpeita ennen muutoksen toteutusta. Muutosten vaikutusten ennakoinnista käytetään termiä kapasiteetin suunnittelu. (Smith ja Williams 2001)

Suorituskyky ja kapasiteetti ovat kiinteästi yhteydessä toisiinsa. Guntherin (2015) mukaan suorituskyvyllä tarkoitetaan yleisesti tehtävien suoritukseen menevää aikaa ja kapasiteetilla niiden suorittamiseen tarvittavia resursseja. Tehtävien (työkuorman) lisääntyessä tehtävien suoritukseen menevä aika (vasteaika) alkaa kasvaa, ja tietyssä pisteessä vasteajat ylittävät vaaditun suorituskykytason järjestelmän käytössä olevilla resursseilla (Kuva 4). Järjestelmän kyvystä tyydyttää läpäisykyky- ja vasteaikatavoitteet tehtävien määrän (kysynnän) kasvaessa käytössä olevilla resursseilla käytetään määritelmää järjestelmän skaalautuvuus. (Smith ja Williams 2001). Kapasiteetin suunnittelua vaikeuttaa se, että suorituskyvyn ja kapasiteetin suhde ei käyttäydy lineaarisesti. Esimerkiksi laiteresurssien lisäys ei välttämättä tarkoita järjestelmän kapasiteetin kasvua, jos ohjelmisto ei kykene hyödyntämään lisäresursseja eli skaalautumaan (Gunther 2015)




***Kuva 4.*** *Vasteajan kasvu suhteessa kuorman kasvuun*

Järjestelmän skaalautuvuus eli sen kyky sopeutua työkuorman muutoksiin tehokkaasti on tavoiteltu ominaisuus ohjelmistojärjestelmille. Skaalautuvuuden merkitys nähdään tärkeänä järjestelmän pitkäaikaiselle menestykselle. Skaalautuvuus tuo myös kustannustehokkuutta, jos käytössä olevat resurssit pystytään mitoittamaan sen hetkisen työkuorman, olio- tai prosessimäärän mukaisesti. Bondin mukaan skaalautuvuudelle ei ole yksiselitteistä yleisesti hyväksyä määritelmää eikä siten yksiselitteisiä mittareita. Bondin näkemyksen mukaan ohjelmistojärjestelmän skaalautuvuuden lajeja ovat rakenteellinen-,

kuormitus-, tila- ja tila-aika -skaalautuvuus. **Rakenteellinen** skaalautuvuus on ohjelmiston toteutuksesta riippuvainen, ja se voi asettaa kiinteitä rajoja, joiden ulkopuolelle ei voida mennä. Tällaisia rajoja voivat olla esimerkiksi standardoitu rajattu numeroavaruus tai ohjelmiston toteutuksessa rajoitettu säikeiden määrä. **Kuormitus**skaalaus on ohjelmiston kykyä sopeutua sulavasti työkuorman muutoksiin. **Tila**skaalaus on kykyä hallita muistinkäyttöä tehokkaasti tarpeen mukaan. **Tila-aika** -skaalaus on kykyä hyödyntää muistia tehokkaasti tietorakenteiden ja algoritmien avulla nopeasti kasvavan kuormitustilanteen mukaan. Tila-aika -skaalauksen esimerkki on esim. tiedonkäsittely tasapainotetussa puurakenteessa lineaarisen tietorakenteen sijasta. (Bondi 2015)

## 3.3 Suorituskyvyn hallinta osana ohjelmistokehitysprosessia

Tutkittavan kohdeyrityksen sisäisen ohjeistuksen mukaan suorituskyvyn hallintaa ei voi erottaa omaksi prosessiksi muusta ohjelmistokehityksestä (Kuivalainen 2014). Bondin (2015) mukaan suorituskyvyn hallinta on prosessi, joka koskettaa ohjelmistokehitystä kaikista sen näkökulmista: aina konseptin hahmottelusta ja vaatimuksista testaukseen ja toimitukseen asti. Smithin ja Williamsin (2001) mukaan suorituskyvyn hallinta voidaan helposti liittää osaksi ohjelmistokehitysprosessia määrittelemällä suorituskyvynhallinnalle olennaiset etapit ja vaihetuotteet, jotka ovat organisaatiolle ja sen projekteille tarkoituksenmukaisia. Suorituskyvyn hallinnan sisällyttäminen kaikkiin ohjelmistokehityksen vaiheisiin pienentää merkittävästi suorituskykyongelmien riskiä laajoissa ja monimutkaisissa järjestelmissä (Bondi 2015).

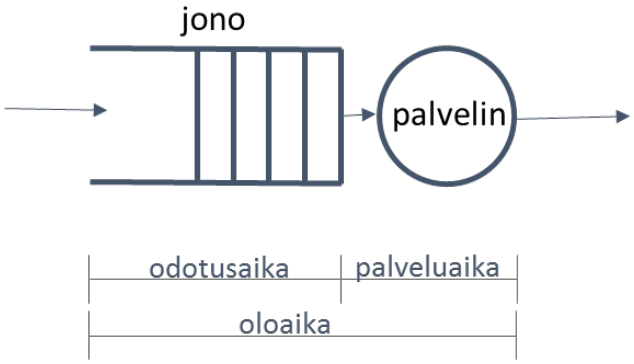
Suorituskyvyn hallinta osana ketterää ohjelmistokehitystä vaatii kuria ja suunnitelmallisuutta, jotta suorituskyky kyetään mittaamaan ja analysoimaan pyrähdyksen aikana. Suurimman haasteen aiheuttaa suorituskyvyn testausympäristö. Testausympäristö on joukko ohjelmistotyökaluja, jotka vaativat ohjelmistokehitystä siinä missä itse tuotekin. Koska ketterässä ohjelmistokehityksessä myös suorituskykyvaatimukset voivat muuttua pyrähdyksien välillä, ne voivat vaatia ohjelmistomuutoksia myös testausympäristöön. Etuna on se, että aikainen ja jatkuva suorituskykytestaus kasvattaa todennäköisyyttä havaita suorituskykyongelmat ohjelmiston aiemmissa kehitysvaiheissa eikä vasta tuotteen ollessa jo melkein valmis. Aikainen havaitseminen taas mahdollistaa refaktoroinnin tai uudelleen suunnittelun tulevissa pyrähdyksissä ohjelmistokehityksen aikana. (Bondi 2015)

Suorituskykytavoitteiden saavuttaminen projektin aikana ei ole aina yksinkertaista. Samat suorituskykyvaatimukset eivät välttämättä sovellu kaikille tuotevariaatioille tai asiakassegmenteille. Vastaavasti asiakas ei välttämättä pysty kertomaan tarkkoja tavoitearvoja, tai niiden määrittely voi olla erittäin vaikeaa. Määrittelyä voidaan helpottaa muun muassa rajaamalla vaatimukset tiettyihin edustaviin referenssityökuormiin esimerkiksi perustuen asiakassegmentin yleiseen todettuun työkuormaan. Suorituskyvyn verifiointi

on aina mielekästä, vaikka suorituskykyvaatimuksia ei olisi edes määritelty. Suorituskykytestauksen aikana voi paljastua muistivuotoja, rinnakkaisten vikojen vaikutuksia ja huonot toteutusratkaisut. Hyvin suunniteltuna suorituskykytestauksella voi todentaa suorituskyvyn rajat. Vaatimusten verifiointiin tuo oman haasteen se, että suorituskykytestaus vaatii lähtökohtaisesti toimivan ohjelmiston. Siksi toiminnallinen testaus tulisi olla suoritettuna ensin. (Bondi 2015)

## 3.4 Suorituskyvyn mittaus ja mallinnus

Suorituskyvyn mallinnuksen perusteena on jonoteoria. Kleinrockin (1975) mukaan se on matemaattinen menetelmä, jolla pyritään ennustamaan muun muassa jonon pituutta sekä odotusaikaa jonossa. Bondin (2015) mukaan ohjelmistojen tapauksessa jonot syntyvät tehtävien odottaessa ajovuoroaan palvelimella (engl. server) kuvan 5 mukaisesti. Tässä teoria esimerkissä palvelin tarkoittaa esimerkiksi yksittäistä tietokoneprosessoria tai IO-laitetta. Suoritettava tehtävä saapuu ensin palvelimen työjonoon, josta se siirtyy palvelimen prosessointiin ja poistuu valmistuessaan. Mikäli uusia tehtäviä saapuu enemmän kuin palvelin pystyy välittömästi suorittamaan, prosessointia odottavien tehtävien määrä jonossa alkaa kasvaa. Koska ohjelmiston yksittäinen käyttötapahtuma synnyttää joukon suoritettavia tehtäviä ja niiden suoritukseen osallistuu yleisesti eri laiteresursseja, ohjelmistojärjestelmä voidaan käsittää joukkona abstrakteja resurssijonoja (Bondi 2015).




***Kuva 5.*** *Töiden suoritus yhden palvelimen järjestelmässä*

Jonon suorituskykymittareina käytetään keskimääräistä oloaikaa (engl. residence time), käyttösuhdetta (engl. utilization rate), läpäisykykyä (engl. throughput) ja jonon pituutta (engl. queue lenght). Oloaika sisältää sekä odotus- että palveluajan. Odotusaika on suoritusvuoron odotukseen kulunut aika, ja palveluaika on tehtävän suoritukseen kulunut aika. Käyttösuhde on keskiarvo ajasta, jonka palvelin on varattuna suhteessa tarkasteluaikaan.

Läpäisykyky puolestaan kertoo, kuinka monta tehtävää palvelin suorittaa tarkasteluaikana. Jonon pituudella tarkoitetaan suoritusvuoroaan odottavien tehtävien määrää jonossa. (Smith ja Williams 2001)

Jonon suorituskykymittarit pystytään laskemaan kaavojen (3-1), (3-2) ja (3-3) avulla neljän alla mainitun perusmittauksen avulla (Smith ja Williams 2001). Perusmittaukset ovat:

- tarkasteluaika *T* (engl. measurement period),
- saapuneiden tehtävien määrä *A* (engl. number of arrivals),
- valmistuneiden tehtävien määrä *C* (engl. number of completions), ja
- tehtävien suoritukseen kulunut aika eli valmistuneiden tehtävien yhteenlaskettu palveluaika tarkasteluaikana *B* (engl. busy time).

Perusmittauksista voidaan laskea edelleen seuraavat yleiset suorituskykymittarit:

$$käyttösuhde,\ U = \frac{B}{T} \quad ,(utilization) \tag{3-1}$$

$$läpäisykyky,\ X = \frac{C}{T} \quad ,(throughput) \tag{3-2}$$

$$keskimääräinen\ palveluaika,\ S = \frac{B}{C} \quad ,(mean\ service\ time) \tag{3-3}$$

Koska suorituskyvyn mallinnuksen tavoite on arvioida tuotteen suorituskyky ennen tuotteen valmistumista, mallinnus ei voi perustua mitattuihin arvoihin. Suunniteltaessa järjestelmälle tulisi asettaa tavoitearvoina tehtävien saapumisnopeus $\lambda$ (engl. arrival rate) ja palveluaikavaatimukset *S* (engl. service requirements). Näistä pystytään edelleen johtamaan järjestelmän tavoitesuorituskykyarvot kaavojen (3-4), (3-5), (3-6) ja (3-7) avulla. Mallinnuksessa täytyy käyttää oletusta, että järjestelmä kykenee käsittelemään kaikki saapuvat tehtävät tarkasteluaikavälillä. Ilman tätä oletusta järjestelmän tavoitearvoja oloajalle ei voitaisi täyttää, koska tehtävien odotusaika jonossa kasvaisi kumulatiivisesti odottavien töiden määrän kasvaessa jatkuvasti. (Smith ja Williams 2001)

Tavoite suorituskykyarvot voidaan laskea Smithin ja Williamsin (2001) mukaan järjestelmän tehtävien saapumisnopeus ja palveluaikavaatimus tavoitearvoista seuraavasti:

$$läpäisykyky,\ X = \lambda \tag{3-4}$$

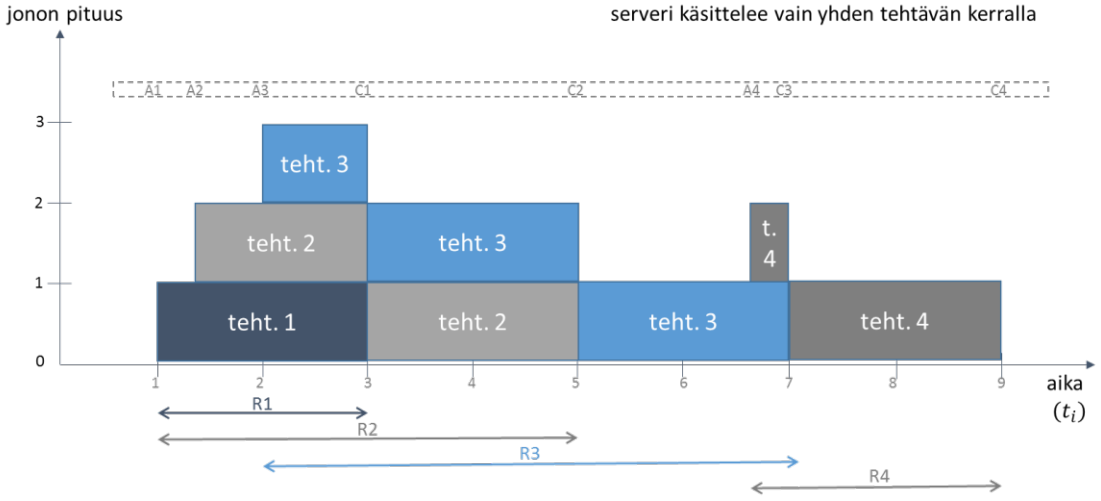
$$käyttösuhde,\ U = X \times S \tag{3-5}$$

$$oloaika,\ R = \frac{S}{1-U} \quad ,(residence\ time) \tag{3-6}$$

$$jonon\ pituus,\ N = X \times R \quad ,(queue\ lenght) \tag{3-7}$$

Edellä mainittu oloajan laskenta perustuu jonon suorituskyvyn mallinnuksen perustana olevaan Littlen lakiin (Smith ja Williams 2001). Littlen lain mukaan $N = X \times R$, jossa $N$ on jonon keskimääräinen pituus, $X$ on keskimääräinen läpäisykyky ja $R$ on keskimääräinen vastausaika. Littlen laki yhdistää jonon pituuden, läpäisykyvyn ja oloajan toisiinsa. Lain avulla pystytään laskemaan esimerkiksi arvioitu jononpituus, kun jonon läpäisykyky ja oloaika ovat tunnettuja. Kaikki Littlen lain muuttujat ovat keskiarvoja, koska käytännössä tehtävien määrä, niiden saapumisaika työjonoon ja niiden suoritusaika vaihtelevat tarkasteluaikavälillä. Arkielämässä Littlen lakia voi havainnollistaa lentoaseman lähtöselvitystiskin toiminnalla. Henkilöt saapuvat lähtöselvitykseen toisistaan riippumattomasti, ja henkilöä kohti käytetty palveluaika vaihtelee. Henkilöiden saapumismäärään voi vaikuttaa esimerkiksi suositeltu tuloaika kentälle ja palveluaikaan ruumaan menevien matkatavaroiden määrä. Yhden henkilön näkökulmasta jonossa oloaika voi vaihdella paljon, mutta keskiarvo kertoo hyvin lähtöselvitystiskin suorituskyvyn henkilöiden käsittelymäärän näkökulmasta valitulla aikavälillä. (Smith ja Williams 2001)

Littlen lain toimintaa voi havainnollistaa hypoteettisella esimerkillä (Kuva 6). Kuvassa palvelin suorittaa saapuvat tehtävät (teht.) oloajassa ($R_i$). Tehtävät saapuvat satunnaisesti, ja kaikilla on sama keskimääräinen palveluaika järjestelmässä. Tarkasteluvälinä on kulunut aikaväli $[0, T]$. Kuvasta voidaan päätellä jonon pituus kullakin ajanhetkellä $t$. Laskemalla jonon pituuden keskiarvo hetkiltä $t_i$ saadaan keskimääräinen jonon pituus. Yksinkertaisuuden vuoksi esimerkin palvelin suorittaa vain yhden tehtävän kerrallaan, jolloin saapuneiden (A) ja valmistuneiden (C) tehtävien määrä tarkasteluaikavälillä on neljä. Tällöin edelleen läpäisykyvyksi (X) saadaan 0,5, koska $\frac{C}{T} = \frac{4}{9-1}$. (Bondi 2015)




***Kuva 6.*** *Jonon pituus tarkasteluvälillä*

Kuvan 6 mallinnus käsittää yksittäisen jonon. Ohjelmistojärjestelmätason mallinnuksessa yksittäiset jonot yhdistetään jonoverkkomallilla. Jonoverkkomallinnuksessa lasketaan yhteen yksittäisien jonojen vaikutus käyttäen edellä mainittuja teorioita. Jonoverkkomalli voi olla joko avoin tai suljettu. Avoimessa mallissa syötteet saapuvat järjestelmään toisistaan riippumatta ja poistuva tehtävä ei synnytä suoraan uutta tehtävää. Suljetussa mallissa poistuvaa tehtävää seuraa uusi tehtävä. Suljetusta järjestelmästä on esimerkkinä interaktiivinen palvelu, jossa järjestelmä jää odottamaan uutta käyttäjäsyötettä käyttäjän saaman vasteen jälkeen. Suljettu malli poikkeaa avoimesta sillä, että suljetussa mallissa huomioidaan viive, joka syntyy jonoverkkomallin jäädessä odottamaan uutta tehtävää. Tätä suljetun verkkojonomallin odotusaikaa kuvaavaa suorituskykymittaria kutsutaan keskimääräiseksi miettimisajaksi (engl. think time). (Smith ja Williams 2001)

## 3.5 Teorian suhde tutkimustyöhön

Organisaation johtamisen apuna käytetään yleisesti selkeitä tavoitteita, joiden saavuttamista mitataan ja analysoidaan säännöllisesti. Vastaavasti suorituskyvyn johtaminen edellyttää suorituskykymittareita tavoitteiden saavuttamisen arvioimiseksi. Bondin (2015) sanoin: “Ilman mittareita järjestelmän suorituskykyvaatimuksista voidaan keskustella vain epämääräisesti, ja vaatimuksia ei voida määrittää, testata tai panna käytäntöön”. Tilanteessa, jossa olemassa olevalla järjestelmällä ei ole suorituskykyvaatimuksia tai ne on huonosti määritelty, nykyinen suorituskyky selvitetään kuormittamalla järjestelmää eri työkuormilla. Tavoitteena on selvittää skaalautumisrajat mittaamalla läpäisykyky ja vasteaika eri kuormilla. (Bondi 2015). Toisin sanoen suorituskyvyn tunteminen ja suorituskykymittarit ovat suorituskyvyn hallinnan perusedellytyksiä. Tässä työssä vaikutti tämä periaate merkittävästi kehitystoimenpiteitä valittaessa. Siksi teoriaosassa on käsitelty myös mittaamista ja mallinnusta ohjelmistokehityksessä yleisesti tunnettujen vaiheiden, kuten vaatimusmäärittely, toteutus ja testaus, lisäksi.

# 4. HAVAINTOIHIN POHJAUTUVA ITERAATIO-KIERROS

Tutkimuksen alussa suorituskykytyö oli käytännössä suorituskyvyn testaamista ohjelmistokehityksen loppuvaiheessa, mutta varsinaisesta suorituskyvyn hallinnasta ei voida puhua. Muodollisesti yrityksen prosessin mukainen suorituskyvyn hallinta oli toteutettu, koska suorituskykyvaatimukset asetettiin ja suorituskyky validoitiin. Käytännön toteutus kuitenkin ontui, koska suurta osaa suorituskykyvaatimuksista ei koskaan kyetty validoimaan.

Tässä luvussa esitellään tutkijan omat havainnot käytännön suorituskyvyn hallinnan ongelmista ja valituista toimenpiteistä niiden korjaamiseksi. Tärkeimpänä havainnointikeinona ovat olleet keskustelut järjestelmän suorituskykytiimin jäsenien ja heidän esimiestensä kanssa. Keskusteluissa tutkija on pyrkinyt ymmärtämään myös alueen aiempaa historiaa sekä aiemmin toteutettuja toimenpiteitä, jotta vältettäisiin vanhojen virheiden toistaminen uusia toimenpiteitä valittaessa. Kohdassa 4.1 esitellään ensin tavoite todellisesta suorituskyvyn hallinnasta puhtaan testauksen sijasta. Havaitut suorituskyvyn hallinnan ongelmat on kuvattu kohdassa 4.2, ongelmien korjaamiseksi valitut toimenpiteet kohdassa 4.3 ja toimenpiteiden käyttöönotto kohdassa 4.4. Lopuksi kohdassa 4.5 arvioidaan muutoksen onnistumista.

## 4.1 Tavoitteena suorituskyvyn hallinta testauksen lisäksi

Tavoitteeksi asetettiin parantaa tuotteen suorituskyvyn hallintaa ohjelmistokehityksen aiemmissa vaiheissa. Tavoite sovittiin suorituskyvyn hallintatiimin johtoryhmän kesken tutkimuksen alussa järjestetyssä työpajassa. Kokemuksen perusteella tiimi tiesi, että suorituskyvyn hallinta vaatii enemmän työtä ohjelmistokomponenteilta jo ohjelmiston toteutusvaiheessa. Tiimi pystyi tekemään päätöksen, koska ohjelmistojärjestelmän suorituskyvyn hallinta on sovittu tiimin tehtäväksi organisaation toimintamallissa (alakohta 2.1). Viitekehyksenä toimi yritystason tuoteriippumaton suorituskyvyn hallintaprosessi.

Suorituskyvyn hallinnan merkitys oli kasvanut sekä uuden järjestelmäsukupolven myötä että kasvaneen kapasiteettitarpeen vuoksi. Uusi sukupolvi toi mukanaan tietokoneyksiköiden virtualisoinnin. Virtualisointi mahdollisti ohjelmistokomponenttien hajautuksen useammille virtuaalitietokoneyksiköille, mikä toi joustavuutta tietokoneresurssien hallintaan. Samalla kuitenkin perinteinen laiteresurssien käyttöön perustuva suorituskyvyn mittaus vaikeutui. Perinteisesti laiteresurssien käyttöä on tutkittu seuraamalla prosessorin ja

muistin käyttöastetta prosessitasolla. Ohjelmiston hajautus entistä pienemmille tietokoneyksiköille vaikeutti mittausta, koska käyttötapaukset hajaantuivat useammille tietokoneyksiköille. Lisäksi virtualisointi itsessään vaikeutti mittaamista, koska virtuaalisen käyttöjärjestelmän näkemä käyttöaste ei ole sama kuin fyysisen prosessorin käyttöaste (Dynatrace 2016). Kapasiteetin kasvanut tarve oli puolestaan seurausta asiakkaiden järjestelmien kasvusta. Kuten kohdassa 2.2 mainitaan, järjestelmiä jouduttiin ylimitoittamaan riittävän kapasiteetin varmistamiseksi. Kapasiteetin kasvatus muodostui kuitenkin ongelmalliseksi, koska kapasiteettia ei pystytty skaalaamaan suoraan virtuaaliresursseja lisäämällä. Vaikka skaalaus olisi onnistunutkin, se olisi perustunut enemmän arvaamiseen kuin tietopohjaiseen mitoittamiseen ja entisestään kasvavaan ylimitoitukseen.

## 4.2 Havainnot suorituskyvyn hallinnan ongelmista

Tutkijan havainnon mukaan keskeisin syy suorituskyvyn hallinnan puutteisiin oli suorituskyvyn hallintakulttuurin puuttuminen. Muutamia poikkeuksia lukuun ottamatta erityisesti komponenttitiimeissä motivaatio suorituskykyvaatimusten määrittelyyn ja vastaavasti niiden validointiin oli olematon. Tämä näkyi siten, että järjestelmätason tiimin täytyi erikseen ohjata suorituskykytyön tekemistä ja silti niiden tarvetta kyseenalaistettiin komponenttitiimien toimesta. Toisen avainsyy oli, että suorituskyvyn hallinta nähtiin komponenttitiimien näkökulmasta mahdolliseksi ulkoistaa järjestelmätason tiimille. Kuitenkin johtuen ei-toiminnallisten vaatimusten luonteesta järjestelmätason tiimin mahdollisuudet vaikuttaa tuotteen suorituskyyn tuotekehityksen aikana ovat yleisesti ottaen hyvin vähäisiä.

Tuotteen ulkoisia rajapintoja ei käsitelty tässä tutkielmassa, mutta niiden vaikutus täytyi huomioida mittaushaasteen osalta. Tuotteelle tuleva kuormitus ulkoisista rajapinnoista vaihtelee paljon eri asiakkaiden välillä. Sen vuoksi oli ollut mahdoton tunnistaa järjestelmälle referenssimittausmallia, jonka voisi sanoa edustavan merkittävää määrää tuotteen käyttötapauksista asiakkaiden käytössä. Vastaavasti kuorman simulointi, testimenetelmät ja testauskattavuus on jätetty tarkastelun ulkopuolelle.

Tutkijan avainhavainnot on koostettu taulukkoon 2, joka sisältää yleiskuvauksen havainnosta sekä sen havainnon kriittisyydestä ensimmäisen tutkimuskierroksen tavoitteen näkökulmasta.

***Taulukko 2.*** *Ensimmäisen tutkimuskierroksen havainnot*

| Piori-teetti | Havainto | Kuvaus | Tärkeys ja perus-telu |
|---|---|---|---|
| 1 | Suorituskyvyn tärkeyden muutos | Järjestelmän kapasiteetti oli muodostunut pullonkaulaksi ja vaati kasvatusta. Kapasiteetin kasvatus edellyttää suorituskyvyn kasvatusta kohdan 3.2 mukaisesti. | Erittäin tärkeä asiakastyytyväisyyden vuoksi. |
| 2 | Komponentit eivät noudata nykyprosessia | Komponenttitiimien vastuulle on määritelty komponentin suorituskyky ja kuormitustestaus, mutta niiden noudattamisessa on ongelmia. | Melko tärkeä, mutta ei juurisyy ongelmille. |
| 3 | Suorituskykytyön takapainoisuus | Prosessissa korostetaan testausta ja etenkin järjestelmän vakauden testausta varsinaisen suorituskyvyn mittauksen sijasta. | Tärkeä tutkimusongelman mukaisesti. |
| 4 | Suorituskyvyn mittaamisen vaikeus | Näkyvyys yhden komponentin suorituskykyyn osana järjestelmää on heikko. Tämä vaikeuttaa suorituskyvyn kehittämistä, koska ongelma ei ole helposti kohdennettavissa tiettyyn ohjelmistokomponenttiin. | Tärkeä, koska todettu suorituskyky on perustelu itse kehitystarpeelle. |
| 5 | Tuotteen monimutkaisuus | Järjestelmätason ohjelmistoarkkitehtuurin monimutkaisuus vaikeuttaa merkittävästi järjestelmän suorituskyvyn mallinnusta. | Tärkeä, mutta tätä ei ratkaista tämän tutkimuksen aikana. Suorituskyky yksin ei aina ole riittävä syy arkkitehtuurimuutoksille. |
| 6 | Vajavainen suorituskyvyn testikattavuus | Suorituskykytestaus keskittyy pääasiassa vakauden testaamiseen peruskuormalla. Esim. tuotteen suorituskykyrajoja tai ylikuormanhallintaa ei testata systemaattisesti. | Rajattu ulos tutkimuksesta. |

Taulukon havaintoja tarkastellaan yksityiskohtaisesti alakohdissa 4.2.1–4.2.5. Alakohdat on järjestetty taulukon mukaiseen järjestykseen. Havaintoa tuotteen monimutkaisuudesta käsitellään osana suorituskyvyn mittaamisen vaikeutta.

### 4.2.1 Suorituskyvyn tärkeyden muutos

Suorituskyvyn hallinta oli ollut melko näkymätöntä valtaosalle ohjelmistokehitystiimejä. Suorituskykytestauksesta vastaavan arkkitehdin mukaan juurisyynä oli ollut sekä tuotteen riittävä suorituskyky että tietokoneiden jatkuvasti kasvanut kapasiteetti. Siten yksittäisen ohjelmistokomponentin suorituskyky ei ollut muodostunut ongelmaksi eikä sitä ollut ollut tarve erikseen mitata. Vaikka tuotteen juuret ovat jo 1990-luvulla, vaade systemaattiselle komponenttitason suorituskykytestaukselle on esitetty ensimmäisen kerran vasta vuonna 2013 (Durairaj 2013).

Asiakkaiden liikennemäärien kasvu oli kuitenkin lisännyt painetta kasvattaa järjestelmän kapasiteettia. Rajoitukseksi oli noussut järjestelmän kapasiteetista vastaavan tuoteomistajan mukaan nykyinen ohjelmistoarkkitehtuuri ja ohjelmistokomponenttien skaalautumattomuus. Kapasiteettia ei pystytty kasvattamaan enää pelkillä laiteresursseilla kustannustehokkaasti. Tarve kapasiteetin kasvulle aiheutti siten myös tarpeen ymmärtää ohjelmistokomponenttien suorituskykyä. Tuoteomistajan mukaan yksittäisiä suorituskykyoptimointeja oli toki tehty yksittäisien pullonkaulaksi muodostuneiden komponenttien osalta. Ne eivät olleet kuitenkaan valtavirtaa ja eivät olleet edellyttäneet muutoksia suorituskyvyn hallintaan koko tuotteen laajuudella.

Järjestelmätiimissä oli myös huomattu se, että komponenttitiimien suorituskykytehtävät kilpailivat resursseista ohjelmiston toiminnallisten vaatimuksien toteutuksen kanssa. Komponenttitiimit olivat lähtökohtaisesti ylikuormittuneita, jolloin uuden ominaisuuden toteutus meni tiimien prioriteeteissa suorituskykytehtävien edelle. Aiempi vähäinen tarve suorituskyvyn hallinnalle ja tiimien ylikuormitus oli myrkyllinen yhdistelmä niillekin tiimeille, joissa oli ohjelmiston suorituskyvystä kiinnostuneita henkilöitä.

### 4.2.2 Komponenttitiimit eivät noudata nykyprosessia

Vaade komponenttikohtaisesta kuormitustestauksesta oli asetettu vuotta aiemmin ja samalla oli luotu mallidokumentit testisuunnitelmalle ja -raportille. Vaikka liiketoiminta-alueen johtoryhmä oli hankkeen takana, käytännön noudattamisessa oli ongelmia. Suorituskykytiimin näkemyksen mukaan syynä oli, että komponenttitiimit eivät ehdi tekemään suorituskykytestausta. Komponenttitiimin näkökulmasta heidän ohjelmistokehitykseensä vain lisättiin yksi paljon aikaa vaativa työvaihe lisää. Voidaan siten tulkita, että komponenttikohtaisen suorituskykytestauksen merkitystä ei nähty mielekkäänä komponenttitiimeissä – olihan tähän asti pärjätty sitä ilman.

Ongelmaksi muodostui myös se, että useat komponenttitiimit näkivät suorituskyvyn puhtaasti järjestelmätason asiana. Sitä varten oli olemassa järjestelmän suorituskykytiimi, ja

sen kuului hoitaa suorituskykytehtävät. Osin tämä ajattelumalli on ymmärrettävä, koska harva ohjelmiston käyttötapaus toteutettiin vain yhden komponentin toimesta. Koska yksittäisen vaatimuksen toteuttavat useat eri komponentit, järjestelmätason vaatimus täytyisi ositella yksittäisille komponenteille. Esimerkiksi käyttötapauksen vasteaika tulisi jakaa operaation toteuttavien komponenttien kesken. Komponenttitiimit eivät kokeneet, että oman osuuden hallinnalla olisi suurta merkitystä. Tämä asenne näkyi myös ratkottaessa suorituskykyongelmia. Komponenttitiimit olivat haluttomia tekemään muutoksia omaan komponenttiinsa suorituskyvyn parantamiseksi, koska ei ollut kyetty osoittamaan vain kyseistä komponenttia koskevaa suorituskykyvaatimusta.

### 4.2.3 Suorituskykytehtävien takapainoisuus

Ohjelmistokehityksessä looginen järjestys on määritellä ensin vaatimukset, jotka ohjaavat toteutusta ja joita vasten tuote loppujen lopuksi validoidaan. Tätä vaatii myös yrityksen noudattama standardi TL9000, jonka mukaan tuotekehitykselle tulee antaa syötteeksi sekä toiminnalliset että ei-toiminnalliset vaatimukset mukaan lukien suorituskykyvaatimukset (QuEST Forum 2016).

Järjestelmätasolla tuotteelle oli asetettu suorituskykyvaatimukset, mikä täyttää niin standardin kuin yrityksen sisäisen suorituskyvyn hallintaprosessin vaatimuksen. Suorituskykyarkkitehdin mukaan ohjelmistokomponenttikohtaisia suorituskykymittareita oli pyritty määrittelemään useaan otteeseen ennen tätä tutkimusvaihetta. Teknisen jaottelun vaikeuden lisäksi ongelmaksi oli muodostunut vaatimusten omistajuus. Se puolestaan johti siihen, että harvat komponenttitiimit edes pyrkivät määrittelemään niitä puhumattakaan niiden kunnioittamisesta ohjelmistototeutusta ohjaavina vaatimuksina.

Vaatimusten määrittämisongelma oli johtanut siihen, että suorituskyvyn ohjaaminen tuotekehityksen aikana oli hyvin vaikeaa suorituskykytiimin toimesta. Tämän seurauksena suorituskyvyn hallinta on keskittynyt järjestelmän testaamiseen. Komponenttitiimeille oli asetettu vaade suorituskykytestauksesta osana ohjelmistokehitysprosessia. Käytännössä suorituskykytestaus oli jäänyt taka-alalle huonosti komponentteihin kohdistuvien vaatimuksien takia. Sen sijaan komponentin suorituskykytestauksessa oli keskitytty komponentin vakaustestaukseen. Sama ilmiö heijastui myös järjestelmätason testaukseen. Tosin järjestelmätasolla syynä oli enemmän julkaisun rajallinen testausaika, mikä pakotti löytämään perusviat suorituskyvyn mittaamisen kustannuksella. Lisäksi komponenttitiimien vastahakoisuus suorituskykyvikojen korjaamiseen ei edesauttanut suorituskyvyn mittaamista. Siten myöskään standardin edellyttämä suorituskyvyn validointi ei täysimääräisesti täyttynyt.

### 4.2.4 Mittaamisen vaikeus ja tuotteen monimutkaisuus

On vaikea todentaa, oliko suorituskyvyn mittaamisen vaikeus enemmän seurausta suorituskykyvaatimuksien puutteista vai tuotteen arkkitehtuurista. Se oli kuitenkin havaittavissa, että tuotteen komponenttitason suorituskyvyn mittaamista ei ollut suunniteltu ohjelmiston määrittely- ja toteutusvaiheissa. Järjestelmässä ei ollut määriteltynä komponenttikohtaisia suorituskyvyn mittareita. Suorituskykytiimin arkkitehtien mukaan arkkitehtuurin monimutkaisuuden vuoksi ei ole tunnistettu yksinkertaisia ja yhdenmukaisia komponenttikohtaisia suorituskykymittareita ja -indikaattoreita. Useat ohjelmistokomponentit osallistuvat saman operaation toteutukseen toteuttaen eri tehtäviä kuten esimerkiksi tulevan datan muokkaus tai datan tietokantaan tallennus. Vaikeus tulee siitä, että samat komponentit toteuttavat erilaisia ja eri määriä rinnakkaisia operaatioita, mikä vaikeuttaa yksittäisen käyttötapauksen suorituskyvyn analysointia.

Suorituskykymittareiden reaaliaikaiseen monitorointiin oli olemassa työkalut, joita käytettiin järjestelmätason suoristuskykymittauksessa ja jotka sopivat myös komponenttitiimien käyttöön. Komponenttikohtaisten mittareiden puutteen vuoksi komponenttien suorituskyvyn mittauksessa oli keskitytty pitkälti samoihin mittareihin kuin järjestelmätason suorituskyvyn mittauksessa. Erona on vain se, että tarkastelu keskittyi komponentin suorituskykyyn tai oikeamminkin komponentin käyttämän virtuaaliyksikön resurssien kulutukseen. Komponenttikohtaiset mittaukset eivät olleet täysin hyödyttömiä, koska ne osaltaan vähensivät järjestelmätasolla havaittujen suorituskykyongelmien määrää. Komponenttitiimien tuottamaa komponenttikohtaista mittausdataa ei kuitenkaan pystytty uudelleen käyttämään järjestelmätasolla, koska järjestelmän muu yhdenaikainen kuormitus puuttui. Oman haasteensa lisäsi järjestelmätasolla käytetyt noin 1 000 yksittäistä resurssien kulutusmittaria. Yksittäisen mittarin perusteella ei pystynyt käytännössä päättelemään järjestelmän suorituskykyä.

### 4.2.5 Suorituskykytestauksen kattavuus

Tutkimustyön aikana tuli myös ilmi, että testikattavuus ei täyttänyt kaikkia TL9000-standardin vaatimuksia. Testikattavuus rajattiin kuitenkin ulos tutkimuksesta, koska testaus sinällään noudattaa olemassa olevaa prosessia ja sen puutteita kehitetään joka tapauksessa tutkimuksen rinnalla osana kehitystavoitteita (Hämäläinen 2015d). Tosin testikattavuuden osalta oli havaittavissa yhteys suorituskykyvaatimuksien ja -mallien puutteeseen. Mikäli ne olisivat olleet hallinnassa, olisi pystytty arvioimaan myös todellinen suorituskykytestauksen testikattavuus.

## 4.3 Suunnitelma suorituskyvyn hallinnan kehittämiseksi

Aiemmin kehitetty komponenttikohtainen suorituskykytestaus ja komponenttikohtaiset suorituskykyvaatimukset nähtiin lähtökohtaisesti oikeina ratkaisuina edetä nykytilanteessa. Samalla nämä täyttivät myös TL9000-standardin vaatimukset suorituskykyvaatimusten ja -testauksen osalta. Testauksen valintaa edesauttoi se, että komponenttikohtainen suorituskyvyn testaus oli jo määritelty osaksi normaalia ohjelmistokehitystä, kuten esitellään alakohdassa 2.2. Suorituskykytestauksen lisäksi lähdettiin korostamaan komponenttikohtaisten suorituskykyvaatimuksien tarvetta. Suorituskykytiimi oli jo aiemmin tiedostanut, että järjestelmätason suorituskykyyn vaikuttavat tekijät voitiin ymmärtää vain ymmärtämällä komponenttien suorituskyky.

Vaatimusten ja testauksen rinnalla päädyttiin vaatimaan myös komponenttikohtaisia suorituskykymittareita. Ohjeistus komponenttitiimeille tulisi rakentaa nimenomaan suorituskykymittareiden näkökulmasta. Suorituskykymittareita ei nähty ylivoimaisena lisätyönä komponenttitiimeille, koska ne testasivat komponenttinsa suorituskykyä joka tapauksessa. Lisäksi suorituskyvyn merkitystä ja komponenttitiimien roolia tulisi korostaa kehitysorganisaatiolle. Komponenttitiimeillä odotettiin myös olevan perusmotivaation kunnossa, koska asiakkaiden raportoimat suorituskykyyn liittyvät viat kuormittivat vikakorjauksien ja ylimääräisen hallinnon kautta useita komponenttitiimejä.

## 4.4 Valitut kehitystoimenpiteet

Toimenpiteet sovittiin useiden valmistelevien keskustelujen jälkeen suorituskykytiimin sisäisessä työpajassa joulukuussa 2014 ja esiteltiin koko organisaatiolle tammikuussa 2015 osana suorituskykytiimin toimenpidesuunnitelmaa (Hämäläinen 2015d). Kehitystoimenpiteet on käsitelty omissa alakohdissaan. Yleiskuva toimenpiteistä ja niiden arviointikriteereistä on esitelty taulukossa 3. Kehitystoimenpiteet toteutettiin vuoden 2015 ensimmäisen puoliskon aikana ja niiden onnistumista arvioitiin vuoden jälkimmäisellä puoliskolla. Toimenpiteiden toteutuksesta vastasivat järjestelmän suorituskyvystä vastaava projektijohtaja ja järjestelmän suorituskykytiimin arkkitehdit. Lisäksi tiimin muut jäsenet osallistuivat erityisesti toimenpiteiden seurantaan.

***Taulukko 3.*** *Ensimmäisen tutkimuskierroksen kehitystoimenpiteet ja niiden arviointimittarit*

| Tärkeys | Havainto | Kehitystoimenpide | Arviointimittari |
|---|---|---|---|
| erittäin tärkeä | suorituskyvyn tärkeyden muutos | viikoittainen tuotteen suorituskykyjulkaisu | julkaisutahti ja saatu palaute |
| melko tärkeä | komponenttitiimit eivät noudata prosessia | komponenttitason kuormitustestauksen kattavuuden seuranta | testauksen tekevien komponenttitiimien määrä |
| tärkeä | Suorituskykytehtävien takapainoisuus | komponenttikohtaiset suorituskykyvaatimukset | vaatimukset määritelleiden komponenttitiimien määrä |
| tärkeä | suorituskyvyn mittaamisen vaikeus | raporttipohja komponenttien kuormitusmittauksille | toimenpide on toteutettu |
| | | ohjeistus komponenttikohtaiselle suorituskykymetriikalle | toimenpide on toteutettu |

Taulukon 3 kehitystoimenpiteet esitetään yksityiskohtaisesti alakohdissa 4.4.1–4.4.5 taulukon mukaisessa järjestyksessä. Alakohdissa arvioidaan myös toimenpiteen onnistuminen.

### 4.4.1 Suorituskykyjulkaisu

Suorituskyvyn näkyvyyttä ja merkitystä tuotiin tuotekehitysorganisaation tietoon julkaisuilla, joita kutsuimme ”Performance News” -nimellä. Keskityimme julkaisuissa kertomaan, mitä suorituskyky on ja miten sitä tulisi hallita. Julkaisun avulla välitimme myös muut toimenpiteet organisaatiolle. Keskeisin julkaisu, jolla viestittiin koostetusti ohjelmistokomponenttikohtaisista suorituskykyvaatimuksista ja -mittareista koko organisaatiolle, julkaistiin huhtikuussa 2015 (Hämäläinen 2015c). Suorituskykyjulkaisujen haasteena oli se, että mielenkiintoisen julkaisun tekeminen viikoittain vaati paljon työtä. Toisaalta sovittu julkaisutahti pakotti myös miettimään, mitä suorituskyky oikein on ja miten asiat voi sanoa mahdollisimman tiiviisti. Asioiden tiivistäminen oli tärkeää, jotta ihmiset ylipäätään jaksavat lukea julkaisun. Alussa suorituskykytiimin jäsenet kyseenalaistivat julkaisujen hyödyn, mutta parin kuukauden jälkeen tiimin jäsenet alkoivat työstää uusia julkaisuja omasta halustaan.

Palautteen perusteella tuotteen suorituskyvyn hallinta ja sen osa-alueet tulivat ensimmäistä kertaa laajemmin näkyville. Palaute julkaisuista oli pääosin positiivista, ja järjestelmän suorituskykytiimi sai kiitosta kiinnostavista julkaisuista. Myös suorituskykytiimin yläpuolella olevat johtajat alkoivat seurata julkaisua ja kysellä lisätietoja yksityiskohdista.

Yksittäinen julkaisu, joka liittyi tuotteen työkuormaan, johti jopa laajempaan keskusteluun, ja sen seurauksena komponenttitiimien rooleja täsmennettiin. Valtaosa palautteessa liittyi kuitenkin julkaisujen erääseen yksityiskohtaan. Julkaisuissa käytettiin tehokeinona vaihtuvia kuvia, jotka havainnollistivat julkaisun aihetta. Kuvissa oli esimerkiksi ruuhkautunut liikenne (pullonkaula), valtava monitasoristeys (jaettu resurssi) ja ylikuormitettu kuorma-auto (kuormitusrajat). Järjestelmän kehitysorganisaation jäsenet odottivat mielenkiinnolla seuraavaa julkaisua nimenomaan osuvien kuvien takia. Kuvat olivat syy poimia suorituskykyjulkaisu viestien tulvasta.

### 4.4.2 Komponentin kuormitustestauksen kattavuuden seuranta

Ohjelmistokomponenttitason suorituskyvyn testauksen ohjeistus pidettiin alkuperäisenä. Suorituskykytestauksen merkitystä kuitenkin korostettiin sekä komponenttitiimien vastuuhenkilöille että tiimien suorituskykytestaajille. Järjestelmätason suorituskykytiimistä nimettiin henkilö, joka seurasi etenemää ja muistutti jatkuvasti komponenttitiimejä testauksen suorittamisesta ohjelmistojulkaisun aikana. Jatkuvalla seurannalla oli valtava merkitys itse testauksen suorittamiseen. Useat komponenttitiimit alkoivat suunnitella testausta vasta useiden muistutuksien jälkeen, vaikka oletusarvona oli sen suunnittelu osaksi normaalia ohjelmistoprojektia.

Ohjauksen seurauksena suorituskykytestausta tekevien komponenttitiimien määrä kasvoi, ja kasvu jatkui vielä varsinaisen toimenpidejakson jälkeen. Taulukossa 4 näkyy, kuinka tutkimusvaiheen aikana testattujen komponenttien määrä kasvoi 6% yksiköllä aina arvoon 67%. Positiivisin havainto asenteiden muutoksesta oli se, että itse testausta ei juurikaan kyseenalaistettu.

***Taulukko 4.*** *Kuormitustestausprosessin noudattaminen*

| Aikaväli | 2H/2014 havaintojakso | **1H/2015 toimenpidejakso** | 2H/2015 seurantajakso |
|---|---|---|---|
| komponentin kuormitustestaus tehty | 20 | 22 | 29 |
| **testikattavuus** | 61% | **67%** | 88% |
| komponentteja yhteensä | 33 | 33 | 33 |

Suorituskykytestausta tekemättömien komponenttitiimien perustelu oli joko ohjelmistomuutosten pieni määrä tai resurssipula. Osa komponenttitiimeistä perusteli kuormitustestauksen tarpeettomuutta vedoten vähäisiin muutoksiin komponentin lähdekoodissa. Jälkimmäisien komponenttien osalta ei siis nähty tarvetta suorittaa edes regressiotestausta suorituskyvyn osalta, mikä osaltaan viestii välinpitämättömyyttä suorituskyvyn hallinnassa. Osaa komponenttitiimeistä rajoittivat käytettävissä olevat henkilö- ja laiteresurssit.

Resurssipulaa selittää osaltaan alakohdassa 2.2 mainittu seikka, jonka mukaan komponenttitiimit käytännössä toistavat järjestelmätasolla tehtävää suorituskykytestausta. Sen seurauksena komponentin suorituskykytestaus vaatii käytännössä järjestelmätason ympäristöt, joiden käyttö vaatii omaa erikoisosaamista. Koska komponenttitiimit toistivat testausta aika ajoin, aikaa kului esimerkiksi kuormitussimulaattoreiden käytön uudelleen mieleen palauttamiseen.

### 4.4.3 Komponenttikohtaiset suorituskykyvaatimukset

Komponenttitiimeille asetettiin vaatimus määritellä ohjelmistokomponenttikohtaiset suorituskykyvaatimukset. Tämä ei vaatinut erillisiä perusteluja, koska organisaation toimintamallin mukaisesti komponenttitiimeillä on kokonaisvastuu omasta komponentista pitäen sisällään myös ei-toiminnalliset vaatimukset (kohta 2.1). Suorituskykyvaatimuksien määrittelyä ja kirjaamista varten tehtiin ohje (Heinonen 2016) suorituskykytiimin toimesta. Ohjeessa esiteltiin myös käytännön esimerkein, miten järjestelmä- ja komponenttitason vaatimukset eroavat toisistaan. Ohjeessa selvennettiin myös, kuinka toimitaan täysin uusien ja aiemmin määriteltyjen suorituskykyvaatimuksien kanssa. Olennaista on, että aiemmin saavutettu suorituskyky kyetään vähintään saavuttamaan myös uudemmissa ohjelmistojulkaisuissa. Verrattuna alakohdan 4.4.2 suorituskykytestauksen edistymiseen komponenttitiimeissä vaatimusten määrittely oli täysin uusi vaade komponenttitiimeille. Varsinaisen toimenpidejakson aikana muutos saatiinkin alkamaan vain kriittisimmissä komponenteissa (Taulukko 5).

***Taulukko 5.*** *Komponenttikohtaisien vaatimusten asettaminen*

| Aikaväli | 2H/2014 havaintojakso | **1H/2015 toimenpidejakso** | 2H/2015 seurantajakso |
|---|---|---|---|
| vaatimukset määritelty työkalussa | 0 | 3 | 18 |
| **vaatimuskattavuus** | **0%** | **9%** | **55%** |
| komponentteja yhteensä | 33 | 33 | 33 |

Alkuhitauden jälkeen seurantajaksolla vaatimusten määrittely eteni huomattavasti paremmin. Valitettavasti vaatimusten laadullista paranemista ei ollut juuri nähtävissä. Komponenttitiimit tyytyivät pääosin kopioimaan järjestelmätason suorituskykytiimin aiemmin määrittelemiä järjestelmätason vaatimuksia, vaikka ne eivät suoraan soveltuneet komponenttivaatimuksiksi. Toisin sanoen, tiimit hoitivat velvoitteen mahdollisimman pienellä työllä. Tätä toimenpidettä ohjattiin käytännössä vain ohjeistuksen avulla – erillistä ohjaajaa ei nimetty.

### 4.4.4 Raporttipohja komponenttikohtaiselle kuormitusmittaukselle

Aiemman komponentin suorituskykytestauksen raportin rinnalle luotiin uusi taulukkopohjainen raporttipohja. Vakiomuotoinen raporttipohja luotiin mittaustulosten yhdenmukaistamiseksi eri komponenttien välillä. Raporttipohjalla pyrittiin myös helpottamaan suorituskyvyn vertailua eri komponenttiversioiden välillä. Tätä käytännön hyötyseikkaa korostettiin motivoidessa komponenttitiimejä raportin käyttöön. Itse raporttipohja välitettiin osana testisuunnitteluohjeistusta sekä erikseen suorituskykyjulkaisussa.

Uusi raporttipohja osoittautui kuitenkin huonoksi toimenpiteeksi. Komponenttitiimit eivät nähneet siitä mitään hyötyä itselleen, ja järjestelmätason suorituskykytiimi ei pystynyt hyödyntämään kerättyä dataa. Virhe oli siinä, että raportti ei itse asiassa kertonut komponentin suorituskykyä. Sen sijaan se kertoi virtuaaliresurssien kulutuksen tietyllä työkuormalla. Kuten kohdassa 4.2 ja sen alakohdissa kuvataan, useat komponentit osallistuvat yksittäisiin käyttötapauksiin. Vastaavasti yksittäinen komponentti osallistuu useisiin käyttötapauksiin. Täten yksittäisen käyttötapauksen virtuaaliresurssien käyttö ei ole kovinkaan kiinnostava, etenkään kun komponenttitiimien mittausympäristöissä käyttämät laiteresurssit eivät olleen täysin yhteneväisiä. Toisin sanoen raportin jatkokäyttöä ei ollut mietitty ennen sen julkaisua.

Virhe raporttipohjassa osoitti myös suorituskykyyn liittyvän erityisosaamisen merkityksen, johon myös viitataan tässä työssä käytetyissä teorialähteissä (Bondi 2015, Smith ja Williams 2001). Raporttipohja tehtiin yhteistyössä tiimin kanssa, joka tuki kehitystiimejä ja asiakkaita erilaisissa järjestelmätason ongelmissa. Tiimin jäsenien järjestelmäosaaminen oli selvästi keskivertoa parempi, mutta heiltä puuttui tietotaito suorituskyvyn mittaamisesta ja mallintamisesta. Koska lisäksi järjestelmätason suorituskykytiimin arkkitehdit kyseenalaistivat uutta pohjaa jo sen luontivaiheessa, olisi se ollut parempi jättää kokonaan julkaisematta.

### 4.4.5 Ohjeistus komponenttikohtaiselle suorituskykymetriikalle

Ohjelmistokomponenttien suorituskyvyn vaatimuksien ja mittareiden määrittelyä varten julkaistiin ohjeistus. Julkaisua edelsi ohjeiden toteutus, mikä tehtiin tämän työn tutkijan ja arkkitehtien yhteistyönä. Suorituskykymittareiden määrittelyä varten tehtiin havainnollinen esimerkein varustettu ohje suorituskykymittareista (Häkkinen 2015). Lisäksi useiden avainkomponenteista vastaavien tiimien kanssa pidettiin erillisiä palavereja, jossa ohjetta esiteltiin ja siitä keskusteltiin. Arviona voidaan kuitenkin todeta, että valitettavasti

edellä kuvattu virtuaaliresurssien kulutukseen keskittyvä raporttipohja ja käyttötapauksiin keskittyvä suorituskykymittariohje eivät tukeneet toisiaan.

## 4.5 Arvio kehitystoimenpiteiden onnistumisesta

Tuotekehitysorganisaation kasvanut kiinnostus suorituskykyasioihin ja valittujen toimenpiteiden toteuttaminen viestii toimenpiteiden osittaisesta onnistumisesta. Tätä on syytä arvostaa, koska komponenttitiimit toteuttivat vaadittuja toimenpiteitä omien resurssiensa puitteissa ja toimenpiteet vaativat muutoksia tiimien vakiintuneisiin käytäntöihin. Muutos myös jatkui varsinaisen tutkimuskierroksen jälkeen, kuten todetaan taulukossa 4 ja 5.

Perusongelmana säilyi pinttynyt ajattelumalli, jonka mukaan järjestelmätason suorituskykytiimi vastaa tuotteen suorituskyvystä. Viesti komponenttikohtaisesta suorituskyvyn hallinnasta ei siis mennyt vielä täysin läpi kehitysorganisaatiolle. Komponenttitiimit eivät todennäköisesti edelleenkään nähneet suorituskykyvaatimusten ja -mittareiden palvelevan ohjelmistokomponentin kehitystä ja suorituskykyongelmien vähentämistä. Toisaalta mikään valitsemamme toimenpide ei ollut kohdistettu toteutusvaiheeseen eikä korostanut toteutusvaiheen ratkaisuja tuotteen suorituskykyyn. Ohjelmiston suorituskyky kuitenkin syntyy valtaosin toiminnallisien vaatimusten toteutusratkaisujen seurauksena. Tämä havainto olikin yksi keskeinen lähtökohta tutkimuksen toiselle iteraatiokierrokselle.

Ratkaisua tutkimusongelmaan ei siis saavutettu tällä ensimmäisellä iteraatiokierroksella. Vaikka komponenttikohtaisia vaatimuksia ja mittaustuloksia alettiin määritellä ja raportoida, suorituskyvyn todentaminen komponenttitasolla tapahtui edelleen hyvin myöhäisessä vaiheessa. Käytännössä se tehtiin vasta järjestelmätason suorituskykytestauksen rinnalla kuten tutkimuksen lähtötilanteessa. Toimenpiteiden toteutus myös vaati vahvaa ulkoista ohjausta järjestelmätason suorituskykytiimiltä, mikä viestii osaltaan motivaation puutteesta komponenttitiimeissä.

Järjestelmätason suorituskykytiimi myös teki virheitä toimenpiteiden valinnassa ja toteutuksessa. Erityisesti komponenttitiimien edellytykset ja tarvittavat toimenpiteet suorituskyvyn hallinnan tarpeisiin olisi tullut miettiä paremmin etukäteen. Edellä alakohdassa 4.4.4 mainitun raporttipohjan osalta vaikutus saattoi olla jopa negatiivinen, koska se ei tukenut pääviestiä ja oli ristiriidassa tavoiteltujen komponenttimittareiden kanssa. Oman haasteensa tuo myös komponenttien erilaisuus. Yhdelle komponentille sopiva ohje tai malli ei ole yleisesti kaikille sopiva.

# 5. HAASTATTELUIHIN POHJAUTUVA ITERAATIOKIERROS

Tutkimuksen toinen iteraatiokierros käynnistyi syksyllä 2015. Toisen kierroksen tutkimusasetelma asetettiin perustuen sekä ensimmäisen vaiheen oppeihin että sidosryhmän antamaan uuteen tavoitteeseen. Ohjelmistojärjestelmän johtoryhmän vaatimus oli luoda suorituskyvyn varmistusprosessi (engl. performance assurance process) estämään sekä suorituskyky että vakausvikojen (engl. stability faults) päätyminen asiakkaalle. Johtoryhmän ajatuksen taustalla oli odotusarvo paremmista järjestelmätason kuormasimulaattoreista ja testiympäristöistä. Tämä kävi ilmi pyydetyn laadunvarmistusprosessin yksityiskohdista. Toisin sanoen suorituskyky ja sen hallinta nähtiin edelleen voimakkaasti järjestelmätasolta ohjattavana ja testausnäkökulmasta. Samalla se kertoo siitä, että myös johtoryhmän täytyisi muuttaa ajatusmallia. Ilman johtoryhmän tukea on vaikea saavuttaa tavoitetta ohjelmistokomponenttikohtaisesta suorituskyvyn hallinnasta komponenttitiimien toimesta.

Toisella tutkimuskierroksella tutkimukseen osallistuivat myös komponenttitiimit. Tätä kautta saatiin parempi kuva suorituskyvyn hallinnan ongelmista komponenttitiimien näkökulmasta. Tutkimusongelma tarkentui siihen, millä keinoin tuotteen suorituskyky komponenttitasolla voidaan havaita aiemmin.

Kohdassa 5.2 ja sen alakohdissa esitetään haastatteluiden löydökset ja kohdassa 5.3 löydöksistä johdettu suunnitelma uudistetusta suorituskyvyn hallintaprosessista. Uudistettu hallintaprosessi ja sen käyttöönotto esitellään kohdassa 5.4. Luvun lopussa kohdassa 5.5 arvioidaan iteraatiokierroksen onnistumista. Löydökset on yleistetty ja ryhmitelty tutkijan toimesta viiteen kategoriaan. Ne ovat suosituskykyosaaminen, suorituskyvyn prioriteetti, suorituskyvyn testattavuus, yhteistyö toisien komponenttitiimien kanssa sekä itse suorituskyvyn hallintaprosessi. Kukin kategoria käsitellään omassa alakohdassa.

## 5.1 Tarve uudelle suorituskyvyn hallintaprosessille

Tarve toiselle tutkimusvaiheelle syntyi erityisesti johdon paineesta. Samalla se mahdollisti johdon täyden tuen tutkimustyölle ja komponenttitiimien osallistumisen itse tutkimukseen. Ensimmäisen tutkimuskierroksen oppien jälkeen tutkija näki paremmaksi osallistuttaa komponenttitiimit haastattelemalla heidät. Samalla johtoryhmä lupasi tutkimusresursseja tuotekehityksen eri toimipisteistä.

Ensimmäisen tutkimuskierroksen jälkeen ymmärrettiin se, että huolellisempi nykyisen prosessin noudattaminen ei ratkaise itse tutkimusongelmaa. Tuote on ja pysyy monimutkaisena, ja suorituskykyvaatimusten sekä -mittareiden tunnistaminen tulee olemaan vaikeana. Siksi yksittäisien toimenpiteiden sijaan itse prosessia täytyisi edelleen kehittää. Tutkimustyötä varten nimettiin oma tutkimustiimi, joka hoiti komponenttitiimien haastattelut ja suorituskyvyn hallintaprosessin kehityksen.

## 5.2 Haastatellun organisaation näkemys ongelmakohdista

Havainnot kerättiin systemaattisesti haastattelujen kautta (Hämäläinen 2015a), jotka hoidettiin tätä tehtävää varten muodostetun tiimin toimesta. Tutkijan lisäksi kaksi jäsentä oli järjestelmän suorituskykytiimistä ja viisi komponenttitiimeistä. Tutkija analysoi havainnot yhdessä tiimin kanssa, mutta ei osallistunut itse haastatteluihin. Yhteensä 33 komponenttitiimistä haastateltiin 13 tiimiä, jotka oli valittu haastattelijoiden kesken (Hämäläinen 2015b). Haastattelukysymykset on esitetty liitteessä 1. Valinnassa pyrittiin saamaan kattava otos huomioimalla komponentin tyyppi sekä komponentin kriittisyys järjestelmän toiminnalle. Tiimin sijainti vaikutti sikäli, että jokaisesta toimipisteestä valittiin vähintään yksi haastateltava tiimi. Taulukossa 6 on mainittu tiimien jaottelu, joka perustuu kohdassa 2.3 esitettyyn jaotteluun.

***Taulukko 6.*** *Haastateltujen tiimien määrä huomioiden komponentin tyyppi ja tiimin sijainti*

| **Komponenttityyppi / toimipiste** | **Palveluja toteuttava** | **Konfigurointiparametreja hallitseva** | **Peruspalveluja toteuttava** | **Tietoa jalostava** | **Ulkoisen rajapinnan toteuttava** |
|---|---|---|---|---|---|
| toimipiste A | 2 | 3 | | | |
| toimipiste B | | | | | 4 |
| toimipiste C | | 1 | | | |
| toimipiste D | | | 1 | 1 | |
| toimipiste E | | | | 1 | |

Yksittäiset löydökset käytiin läpi ja rinnakkaiset löydökset yhdistettiin. Yhdistämisen jälkeen löydöksiä oli 20 kappaletta, jotka esitellään liitteessä 2. Löydökset jaoteltiin edelleen viiteen pääluokkaan, jotka olivat suorituskykyosaaminen, suorituskyvyn yleinen prioriteetti, yhteistyö muiden komponenttitiimien kanssa, suorituskyvyn testauskyvykkyys ja suorituskyvyn hallintaprosessi. Kukin näistä luokista on käsitelty erikseen tämän luvun alakohdissa. Haastattelujen perusteella on yleisesti todettavissa, että komponenttitiimit

eivät näe merkittäviä ongelmia oman suorituskykynsä suhteen. Komponenttitiimeissä ongelmat nähdään pikemminkin järjestelmätason suorituskyvyn hallinnan ongelmina.

### 5.2.1 Suorituskykyosaaminen

Haastattelut aloitettiin kysymyksellä, miten haastateltavat ymmärtävät suorituskyky-käsitteen. Vastauksien perusteella suorituskyky ymmärrettiin yleisesti operaatioiden määränä määrätyssä ajassa tai operaatioiden kestona. Myös ylikuorman hallinta tuli esille yksittäisessä vastauksessa. Kuitenkaan suorituskykyyn liittyviä muita osa-alueita, kuten luotettavuus ja skaalautuvuus, ei vastauksissa tullut esille.

Kysymykseen, kuinka suunnittelette ja hallitsette suorituskykyä, saatiin hyvin eritasoisia vastauksia. Tiimeissä, joissa komponentin suorituskykyä on jouduttu parantamaan käytännön ongelmien takia, oli selvästi parempi osaaminen. Näissä tiimeissä uuden ohjelmisto-ominaisuuden vaikutusta pyrittiin analysoimaan ennen varsinaista toteutusvaihetta. Tiimien pyrkimyksenä oli saavuttaa vähintään aiemman ohjelmistojulkaisun suorituskykytaso uusista toteutetuista ominaisuuksista huolimatta. Heikoin osaaminen oli tiimeissä, jotka kehittävät ylemmän tason sovelluksia. Nämä sovellukset hyödyntävät järjestelmän muiden komponenttien tuottamia perusominaisuuksia, jolloin sovellusten suorituskyky on käytännössä riippuvainen muiden komponenttien suorituskyvystä. Kuitenkin nämäkin sovellukset voivat aiheuttaa järjestelmälle merkittäviä suorituskykyongelmia, kuten esimerkiksi erittäin laaja ja kuormittava kysely järjestelmän tietokannasta.

Kokonaisuutena on havaittavissa, että suorituskyvyn hallintaa ei täysin ymmärretä komponenttitiimeissä. Haastatteluissa tuotiin esille, että suorituskyvyn hallintaan liittyvät ohjeet olivat puutteellisia. Komponenttitiimit toivoivat myös enemmän järjestelmätason tiimin tukea komponentin suorituskyvyn hallintaan. Suorituskykyä ei esimerkiksi ollut huomioitu komponenttivaatimuksissa tai valmistumiskriteereissä (engl. definition of done). Tämän seurauksena esimerkiksi ylikuormanhallintaa ei välttämättä ollut suunniteltu ja ylikuormasuojaus oli jouduttu rakentamaan vasta asiakkaan kohtaamaan ongelman jälkeen. Toisaalta komponenttitiimeissä ei välttämättä edes tiedetty operaattorin järjestelmän todellista työkuormaa, mikä osaltaan vaikeutti kuormitusrajojen määrittelyä.

### 5.2.2 Suorituskyvyn yleinen prioriteetti

Vaikka suorituskyky ja sen merkitys olisi ymmärretty, halutun suorituskyvyn saavuttaminen vaatii työtä. Tällöin esiin nousee usein haaste käytettävistä resursseista suhteessa työn määrään. Siksi haastatteluissa kysyttiin, mikä on suorituskykyvaatimusten prioriteetti tiiminne työjonossa.

Vastauksien perusteella suorituskyvyn hallinnan yleinen prioriteetti oli selvästi alhaisempi kuin uusien ohjelmisto-ominaisuuksien toteutuksen prioriteetti. Suorituskyky sai korkeamman prioriteetin yleensä vain, jos suorituskyvyn puute muodostui konkreettiseksi ongelmaksi. Poikkeuksena oli kaksi tiimiä, joissa suorituskyvyn sanottiin olevan olennainen osa toteutusta ja siihen kiinnitettiin erikseen huomiota. Alhainen prioriteetti käytännössä estää tai ainakin vaikeuttaa suorituskyvyn kehittämistä, jolloin suorituskyvyn eteen tehdään vain aivan välttämätön. Tätä näkemystä tukivat myös vastaukset kysymykseen, mitä suorituskykytestauksen kehitystavoitteita tai ehdotuksia tiimillä on. Matala prioriteetti vaikutti myös suoristuskykyosaamisen kasvatukseen. Esimerkiksi ensimmäisessä tutkimusvaiheessa toteutettu suorituskykymittareiden ohje oli osalle edelleen tuntematon. Lisäksi positiivisena havaintona voidaan sanoa, että tiimit tiedostavat puutteita yksittäisten järjestelmätason käyttötapauksien suorituskykytestauksen osalta.

### 5.2.3 Yhteistyö muiden komponenttitiimien kanssa

Tuotteen arkkitehtuurin ja erityisesti ei-avoimien sisäisten rajapintojen vuoksi komponentin suorituskykyä ei pystytä hallitsemaan muista komponenteista riippumattomasti. Tämä edellyttää kokonaisuuden ymmärtämistä ja yhteistyötä komponenttitiimien kesken. Haastatteluvastauksien perusteella toimivaa yhteistyötä ei juurikaan ollut. Sitä ei myöskään koettu olleen yhdessä järjestelmätason suorituskykytiimin kanssa. Rohkaisevaa on kuitenkin se, että haastatelluissa tiimeissä tiedostettiin suorituskyvyn tiukka kytkös muihin komponentteihin.

Samalla vastauksissa todetaan vähäinen ymmärrys oman ja käyttötapaukseen liittyvien komponenttien yhteistoiminnasta erityisesti suorituskyvyn näkökulmasta. Omalta osaltaan komponenttitiimit näkivät riittävänä julkaista oman komponentin suorituskyvyn testiraportin. Vastaukset kertovat suorituskyvyn omistajuuden epäselvyydestä samaa, mikä mainitaan yhtenä perusteena prosessin takapainoisuuteen myös ensimmäisen tutkimusvaiheen osalta alakohdassa 4.2.3. Siten voidaan todeta, että järjestelmän arkkitehtuuri on yksi keskeisimmistä, joka tulisi huomioida suorituskyvyn hallintaprosessin kehityksessä.

Vastauksien perustella myös järjestelmätason suorituskyvyn tiimin rooli nähtiin epäselvänä. Komponenttitiimien mukaan järjestelmätiimi vaati komponenttien suorituskykytestisuunnitelmat ja -raportit, mutta ei kyennyt antamaan arvokasta palautetta näiden dokumenttien osalta. Samoin ensimmäisen iteraatiokierroksen toimenpide komponenttien kuormitusmittausdatan keruusta (alakohta 4.4.4) nähtiin tarpeettomana. Järjestelmätason tiimiltä odotettiin siis paljon vahvempaa roolia suorituskyvyn käytännön hallinnassa, mikä on tietysti ristiriidassa itse tutkimustavoitteen kanssa. Tutkimustavoite oli nimen-

omaan havaita suorituskyky ja sen ongelmat aikaisemmin. Vastausten perusteella on kuitenkin selvää, että järjestelmä- ja komponenttitiimien roolit ja vastuut eivät olleet selviä koko organisaation laajuudella.

Haastattelulöydösten perusteella voidaan päätellä, että tuotteen arkkitehtuuri ja organisaatiomalli ovat avainasioita, jotka täytyy huomioida niin tämän tutkimuksen toimenpiteistä kuin yleisemmin suorituskyvyn hallintaprosessia määritettäessä. Koska tutkitun järjestelmän yksittäisen komponentin suorituskyky on riippuvainen myös toisista komponenteista, suorituskykyparannukset vaativat läheistä yhteistyötä eri komponenttien välillä.

### 5.2.4 Suorituskyvyn testauskyvykkyys

Kyvykkyys suorituskykytestaukseen nousi esille testaukseen liittyvissä vastauksissa. Kyvykkyydellä tarkoitetaan edellytyksiä toteuttaa tavoitteiden mukainen testaus, jonka edellytyksiä ovat muun muassa testausympäristöt, simuloidut rajapinnat ja analysointityökalut.

Vastauksissa korostui erityisesti suorituskykytestauksen hitaus. Yhtä poikkeusta lukuun ottamatta yhteen testikierrokseen meni aikaa yhdestä kahteen kuukauteen. Tämän seurauksena testitulokset tulivat myöhässä tai olivat jopa hyödyttömiä. Erityisesti toimipisteen A mukaan testauksen tuloksien ei nähty palvelevan komponentti- eikä järjestelmätasoa. Valtaosa ajasta kului testiympäristön valmisteluun, mikä kertoo osaltaan suorituskykytestaukseen vaaditun testijärjestelmän työläydestä.

Kuten kohdassa 2.2 todetaan, komponenttikohtainen suorituskykytestaus tapahtuu pääosin vastaavalla testiympäristöllä mitä käytetään järjestelmätestauksessa. Tämän seurauksena myös testaustyökalut ovat vastaavat. Lisäksi komponenttitiimeillä ei välttämättä ole käytännön osaamista järjestelmätasosta ja testaustyökaluista, mikä hidastaa testausympäristön valmistelua. Vastaavasti testitilanteen hallinta ja tulosten analysointi ovat vaikeampia, koska muun järjestelmän vaikutusta ei voida sulkea ulos. Testijärjestelmän automaation puute sekä kuormasimulaattoreiden rajallisuus koettiin vastauksen perusteella myös ongelmiksi. Automaatioasteen haluttiin olevan korkeampi, jolloin testin suoritus ja testidatan keruu tapahtuisi automaattisesti. Vastaavasti toivottiin uusia simulaattoreita, joita ei ollut vielä toteutettu järjestelmätason testeihin ja joilla pystyttäisin simuloimaan laajemmin komponenttiin liittyviä käyttötapauksia.

Yleisenä havaintona vastauksista voidaan sanoa, että komponenttitiimit halusivat avaimet käteen -testausjärjestelmän, jolla voisi tehdä vaaditun komponenttikohtaisen suorituskyky-

kytestauksen. Suorituskyvyn analysoinnin vaikeus ei tullut odotuksista huolimatta komponenttitiimien edustajien vastauksissa esille. Ainoastaan yksi tiimi kykeni toteuttamaan jatkuvan suorituskykytestauksen, ja se tehtiin öisin komponentin uusimmalle versiolle.

### 5.2.5 Suorituskyvyn hallintaprosessi

Vastauksien perusteella suorituskyky huomioitiin läpi ohjelmistokehityksen kuten mikä tahansa ei-toiminnallinen ominaisuus. Komponenttitiimit analysoivat uudet ominaisuudet ja niiden mahdolliset vaikutukset komponentin suorituskykyyn. Suorituskykyratkaisuja ohjasivat myös olemassa olevat järjestelmätason suorituskykyvaatimukset, aiemman julkaisun suorituskyky sekä kehittäjien kokemus. Kehittäjien kokemus tarkoittaa lähinnä tietotaitoa aiempien ratkaisujen suorituskykyvaikutuksista. Analyysivaiheessa harkittiin myös tarve suorituskykytestaukselle ja tarvittaessa tehtiin myös toteutuksen prototyyppi vaikutuksien ymmärtämiseksi.

Vastaukset olivat linjassa yleisien ohjelmistokehitysperiaatteiden mukaan. Erikoiseksi tilanteen teki se, että komponenttitiimit eivät kuitenkaan ole tyytyväisiä itse vaatimuksiin ja niiden testaukseen, kuten todettiin testauskyvykkyyden ja yhteistyön osalta alakohdissa 5.2.3 ja 5.2.4. Havaintona voidaan sanoa, että suorituskykyanalyysi ja testaus oli pyritty tekemään komponenttitiimeissä, mutta se ei ollut järjestelmällistä eikä järjestelmällisesti dokumentoitua. Hyvä esimerkki tästä oli yhden tiimin havainto merkittävästi parantuneesta komponentin suorituskyvystä (osana koko järjestelmää), mutta parannuksen syytä ei ollut kyetty ymmärtämään.

Vastauksissa on havaittavissa myös eroja toimipisteiden ja komponenttitiimien kesken. Erityisesti Aasiassa sijaitsevan toimipisteen D vastauksissa korostui prioriteetin merkitys. Ohjelmistokomponenttien suorituskykyongelmat tunnistettiin tässä toimipisteessä, mutta käytännön komponenttitason keinoja ei ole käytännössä edes mietitty tai ratkaisua odotettiin muualta. Tämä löydös nosti esille kulttuuritaustan yhtenä lisätekijänä suorituskyvyn hallinnalle alakohdan 5.2.3 järjestelmän arkkitehtuurin ja organisaatiomallin rinnalle.

## 5.3 Suunnitelma uudesta suorituskyvyn hallintaprosessista

Toisen tutkimuskierroksen tavoitteeksi oli asetettu kohdan 5.1 mukaisesti tuotteen suorituskyvyn havaitseminen aikaisemmassa vaiheessa, mikä edellyttää suorituskyvyn hallintaprosessin parantamista. Ratkaisun haastetta lisäsi se, että haastattelulöydösten perusteella ongelmat suorituskyvyn hallinnassa olivat paljon laajemmat kuin oli alun perin ymmärretty. Haastattelut myös vahvistivat käsitystä, että suorituskyvyn hallinnan ongelmat eivät ratkea entistä kattavammalla suorituskykytestauksella. Taustalla oli selvästi suorituskyvyn hallinnan kulttuurin puuttuminen, minkä seurauksena kukin komponenttitiimi

oli kohdistanut voimavarojaan suorituskyvyn hallintaan ja sen osaamisen kasvatukseen omien intressiensä mukaisesti. Yleisesti toiminnalliset vaatimukset veivät voiton ei-toiminnallista vaatimuksista. Tärkeimpänä tekijänä yhteisen suorituskykykulttuurin luomiseen tutkimustiimi näki suorituskykytehtävien tekeminen näkyvämmäksi, mikä edellyttää toimivia sääntöjä ja on linjassa toisen kierroksen tutkimustavoitteen kanssa. Johdon toive (kohta 5) uudesta testauspainotteista laadunvarmistusprosessista hylättiin lopullisesti tässä vaiheessa. Ensimmäisestä tutkimuskierroksesta viisastuneena pyrittiin mahdollisimman pieneen määrään toimenpiteitä todellisen muutoksen mahdollistamiseksi.

Tutkimustiimi valitsi toimenpiteeksi yksinkertaisen ohjemateriaalin luomisen. Materiaali selventää koko sisäiselle organisaatiolle, mistä suorituskyvyn hallinnassa on kysymys. Materiaali sisältää viittauksia pääosin jo olemassa oleviin ohjeisiin ja menetelmiin. Tavoitteeksi ei asetettu yksityiskohtaisia prosessikuvauksia. Tutkimustiimin aiempien kokemusten perusteella ei koettu mielekkääksi määrittää yhtä kaikille komponenttitiimeille soveltuvaa prosessia, koska komponentit ovat monimuotoisia ja organisaation toimintamalli salli komponenttikohtaiset toimintatavat. Aluksi materiaali suunniteltiin jaettavaksi yleisien ohjelmistokehitysmallien mukaisiin osa-alueisiin, kuten ohjelmistomäärittely, -toteutus ja -testaus. Tämä toimenpide valittiin haastattelulöydösten perusteella, joita olivat suorituskyvyn hallintaprosessi ja suorituskykyosaaminen sekä yhteistyö muiden komponenttitiimien kanssa. Kehitystoimenpidettä valitessa ehdotukset testiympäristöjen lisäämisestä sekä testaus ja mittaus työkalujen kehittämisestä jätettiin pois suunnitelmasta.

## 5.4 Uuden prosessin kehitys ja julkaisu

Prosessinkehitys aloitettiin jakamalla luvun 5 alussa esitelty tutkimustiimi ohjelmistokehitysvaiheiden mukaisiin virtuaalisiin pienryhmiin. Ryhmät olivat vaatimus-, toteutus- ja testausryhmä. Kunkin ryhmän tavoite oli kuvata suorituskyvyn hallinta kyseisen ohjelmistokehitysvaiheen osalta. Tutkimustiimin jäsenet valittiin heidän vastuu- ja osaamisalueensa perusteella. Pienryhmien tuli pohtia kyseisen vaiheen ongelmakohtia, joihin ohjeen tulisi erityisesti antaa vastauksia. Ohjeeseen tulisi sisällyttää tieto olemassa olevista työkaluista sekä komponenttitiimien hyväksi havaitsemat käytännöt. Tutkimustiimin viikoittaisissa kokouksissa seurattiin pienryhmien työn etenemistä ja keskusteltiin siihen mennessä toteutetusta materiaalista.

Tutkimustiimin suurimmaksi ongelmaksi muodostuivat näkemyserot toteutettavien ohjeiden tarkkuudesta ja ylipäätään niiden tarpeellisuudesta. Toiset pitivät tarkasteltavia asioita itsestäänselvyytenä ja osana yleistä ohjelmistokehitysosaamista. Nämä henkilöt halusivat pitää materiaalin teoreettisena. Toiset puolestaan pyrkivät hakemaan hyvin tarkkaa kuvausta, jolloin myös uusilla ohjelmistokehittäjillä olisi selvät toimintaohjeet. Jäl-

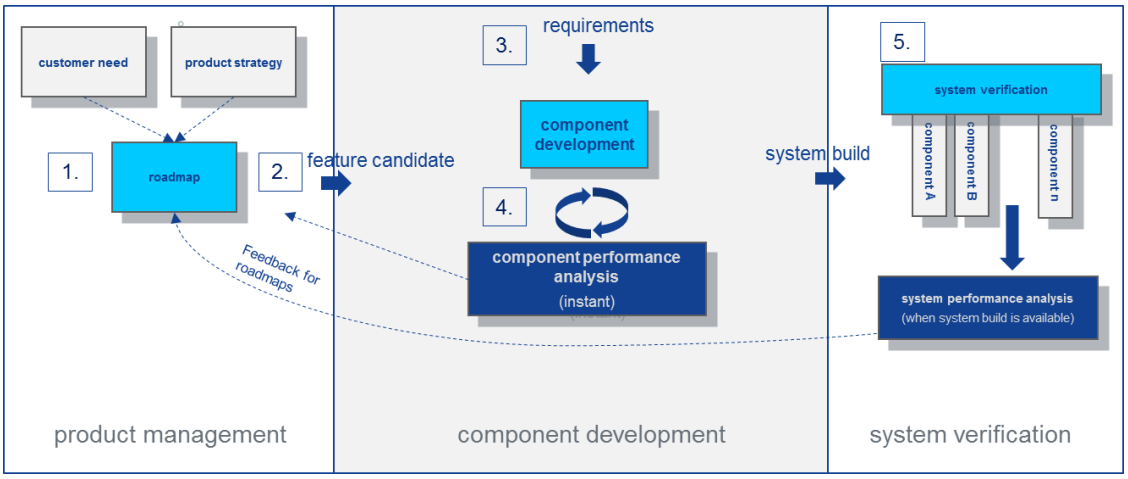
kimmäinen näkökulma oli tietenkin ristiriidassa aiemman havainnon kanssa, jonka mukaan ohjelmistokomponentit olivat hyvin erilaisia ja yksi keino ei sovi kaikille komponenteille. Esimerkiksi suorituskyvyn näkökulmasta käyttäjän suorittama hetkellinen tietokantaoperaatio poikkeaa olennaisesti jatkuvasta tietovirran kirjoituksesta tietokantaan. Toisena merkittävänä ongelmana oli pienryhmien tiukka pitäytyminen kiinni olemassa olevissa toimintatavoissa. Ryhmät dokumentoivat pikemminkin vakiintuneita toimintatapoja kuin pyrkivät löytämään ratkaisuja havaittuihin ongelmiin, vaikka toimintatavat todettiin haastattelujen perusteella riittämättömiksi komponenttitiimien yhteistyön ja suorituskyvyn testattavuuden näkökulmasta. Tutkijan havaintona voi todeta, että tutkimustiimin ajattelu heijasti vahvasti organisaation suorituskyvyn hallinnan toimintatapoja, vaikka sen ongelmat tunnettiin hyvin tutkimustiimissä.

Työn edetessä ymmärrettiin käytetyn prosessin (kohta 2.2) perusongelma; suorituskyvyn hallintaprosessi ei käytännössä tukenut ohjelmiston toteutusvaihetta. Avainongelma oli seuraus siitä, että yksittäisen ohjelmakoodimuutoksen ja sen suorituskykytestauksen välillä oli merkittävä viive. Sen seurauksena ohjelmistokehittäjä sai palautteen koodimuutoksen suorituskykyvaikutuksesta liian myöhään. Palaute tuli vasta, kun muutos oli jo sisällytetty järjestelmätason koontiversioon (engl. build). Viivettä lisäsi se, että jokaiselle järjestelmätason versiolle ei tehty laajaa suorituskykytestausta. Kriittisen korjauksen tapauksessa viivettä ei yleensä ollut, koska korjausta osattiin odottaa. Kriittinen korjaus otettiin suorituskykytestaukseen mahdollisimman nopeasti joko kokonaan uutena järjestelmäversiona tai erillisenä korjauksena. Viive tuli esille tapauksissa, joissa ohjelmistomuutoksen suorituskykyvaikutusta ei osattu odottaa. Ongelman juurisyytä ei useimmiten pystytty välittömästi rajaamaan yksittäiseen ohjelmistomuutokseen, ja ongelman selvitys työllisti useita henkilöitä. Pahimmassa tapauksessa ohjelmistomuutoksen alkuperäinen tekijä oli jo ehtinyt siirtyä uuden toteutustehtävän pariin, mikä edelleen hidasti juurisyyn löytämistä. Myös suorituskyvyn hallintakulttuurin näkökulmasta ohjelmiston toteutusvaiheen tuen puute oli ongelmallista. Vaikka suorituskyky tiedettiin tärkeäksi, tuotteen suorituskyky tuli organisaatiolle näkyväksi lähinnä ongelmien kautta. Ohjelmistokehittäjille ei ollut tarjolla perusmenetelmiä, joilla suorituskykyvaikutus olisi voitu todeta välittömästi toiminnallisten vaatimuksien toteutuksen rinnalla.

Havainto toteutusvaiheen tuen puutteesta ohjasi pohtimaan, miten puuttuva tuki saadaan mukaan suorituskyvyn hallintaprosessiin. Tämän seurauksena tutkimustiimin pienryhmäjaottelu purettiin ja prosessia tarkasteltiin uudelleen kokonaisuutena. Vaiheita yhdistävänä tekijänä nähtiin suorituskykyanalyysi, joka muodostui jatkossa kuvatun uuden prosessin keskeiseksi tekijäksi.

### 5.4.1 Konsepti uudesta suorituskyvyn hallintaprosessista

Uudistetun prosessikonseptin (Kuva 7) mukaan suorituskykyanalyysi on suorituskyvyn hallinnan keskeinen tekijä. Suorituskykyä tulee analysoida jatkuvasti ohjelmistokehityksen eri vaiheissa, ja mitatun suorituskykytiedon takaisinkytkentä edellisiin vaiheisiin on keskeinen analyysin oikeellisuuden varmistamiseksi. Suorituskyvyn analysointimenetelmät riippuvat ohjelmiston kehitysvaiheesta aina uuden ominaisuuden suorituskykyvaikutusten arvioinnista mitatun suorituskykytiedon analyysiin. Suorituskykymittauksen osalta tulee pyrkiä siihen, että sama mittaustapa on toistettavissa eri testausympäristöissä. Jatkuva suorituskykyanalyysi luo edellytyksen suorituskyvyn huomioimiseksi jo osana uuden toiminnallisuuden toteutusta. Ohjelmistomuutoksen suorituskykyvaikutus on nähtävillä välittömästi sen toteutuksen aikana. Tällöin komponenttiin tehdyn ohjelmistomuutoksen aiheuttamat suorituskykyongelmat voidaan ratkaista välittömästi ja estää niiden johtuminen järjestelmätestausvaiheeseen. Samalla se luo edellytykset ohjelmistokehityskulttuurin luomiseksi, jossa yksittäisen toteutuksen suorituskykytehokkuus on osa toteutuksen valmistumiskriteeriä.




***Kuva 7.*** *Uusi suorituskyvyn hallintaprosessikonsepti*

Prosessikonsepti on yhteensopiva kohdassa 2.2 mainitun yrityksen suorituskyvyn hallintaprosessin kanssa. Konsepti on jaettu kolmeen päävaiheeseen: järjestelmän suunnittelu, ohjelmistokomponentin kehitys ja järjestelmän verifiointi. Järjestelmän suunnittelu sisältää vakiintuneet menetelmät ohjelmisto-ominaisuuksien esitutkimuksesta osana julkaisun sisällönhallintaa. Esitutkimus sisältää myös suorituskykyvaikutusten arvioinnin. Komponentin kehitys noudattaa täysin komponenttitiimien käyttämää ohjelmistokehitysprosessia, johon uutena tehtävänä liitetään jatkuva suorituskykyanalyysi. Menetelmää jatkuvan suorituskykyanalyysin luomiseen ei määrätä, jolloin menetelmän valinnassa voidaan huomioida komponentin erityispiirteet. Luontevinta on kuitenkin sisällyttää se osaksi

komponenttikohtaista automatisoitua jatkuvaa integrointia. Järjestelmän verifiointi sisältää järjestelmätason vakiintuneet suorituskyvyn verifiointikäytännöt. Sen lisäksi järjestelmätason verifiointiympäristön merkitys referenssiympäristönä kasvaa, koska se mahdollistaa komponenttikohtaisen analyysin osana koko järjestelmää. Suurin ero järjestelmän alkuperäiseen suorituskyvyn hallintaprosessiin on se, että erillisen komponenttikohtaisen suorituskykytestauksen merkitys pienenee tai jopa poistuu, mikäli jatkuva suorituskykyanalyysi kyetään toteuttamaan komponenttitasolla.

Kuvan 7 prosessikonseptissa on viisi keskeistä tehtävää, joilla hallitaan tuotteen suorituskykyä osana ohjelmistokehitystä. Näiden tehtävien osalta on huomioitava, että ne on valittu tutkittavan tuotteen lähtökohdista. Keskeisimmät valintaperusteet olivat jo toteutettu laaja järjestelmä ja sen suorituskyvyn heikko tuntemus.

Nämä tehtävät ovat:

1. asiakkaan kapasiteetti ja suorituskykytavoitteiden/-vaatimuksien huomioiminen osana uusia ohjelmisto-ominaisuuksia,
2. ohjelmistokomponentin nykyisen kapasiteetin ja suorituskyvyn tuntemus,
3. suorituskykyvaatimuksien määrittely ja hallinta,
4. ohjelmistokomponenttikohtainen jatkuva suorituskykyanalyysi ja
5. järjestelmätason suorituskykyverifiointi.

Järjestelmän suunnitteluvaiheessa **on tärkeä analysoida uusien ohjelmisto-ominaisuuksien suorituskykyvaatimukset (1)**. Koska valtaosalla uusia ominaisuuksista ei tavoitella suurempaa järjestelmän suorituskykyä tai kapasiteettia, jää suorituskykyvaikutus helposti analysoimatta ilman erillistä määriteltyä tehtävää. On tärkeä ymmärtää uuden ominaisuuden suorituskykyvaikutus ennen komponenttitiimin sitoutumista uuden ominaisuuden toteutukseen. Tiimin täytyy analysoida, onko komponentin suorituskykyä parannettava uuden ominaisuuden vaatimuksien saavuttamiseksi ja mikä on siihen vaadittava työmäärä. Analyysin edellytyksenä on **tieto komponentin nykyisestä kapasiteetista ja suorituskyvystä (2)** ja siihen vaikuttavista tekijöistä, joiden olemassaolo ei ole itsestäänselvyys tutkittavassa tuotteessa. Mikäli tietoa ei ole, komponenttitiimin täytyy ensin pyrkiä tunnistamaan ja toteuttamaan komponenttikohtaiset suorituskykymittarit, joista johdetaan tarvittaessa myös mahdollisesti puuttuvat suorituskykyvaatimukset. **Suorituskykyvaatimukset toimivat perustana (3)**, johon komponentin uusien versioiden suorituskykyä verrataan. Suorituskykyvaatimusten tavoitearvoja voidaan tiukentaa, mikäli asiakastarve sitä edellyttää.

Ohjelmistokomponentin kehitysvaiheessa keskeisin tehtävä on **jatkuva suorituskykyvaikutusten analyysi (4)** ohjelmisto-ominaisuuden toteutuksen rinnalla. Jatkuva ana-

lyysi on käytännössä mahdollista vain automatisoimalla, jonka edellytyksenä on koneellisesti mitattavissa olevat suorituskykymittaukset. Lisäksi mittauksista laskettavalle suorituskykymittareille tulee olla suorituskykyvaatimuksesta asetettu tavoitearvo (3). Jatkuva analyysi tarjoaa ensimmäisen palautteen järjestelmäsuunnittelulle. Palaute mahdollistaa esitutkimusvaiheessa tehdyn suorituskykyarvion vertailun toteutuneeseen. Tämä mahdollistaa aikaisen reagoinnin suorituskykyongelmaan ja kasvattaa todennäköisyyttä ongelman korjaamiseksi jo komponentin kehityksessä ilman aikatauluviiveitä ohjelmistojulkaisuun.

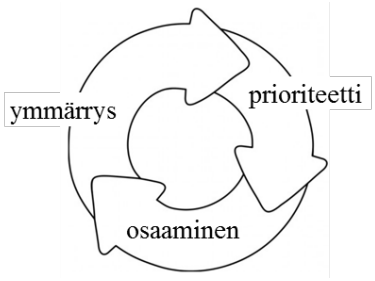
Järjestelmätason **suorituskyvyn verifioinnin ensisijaisena roolina on (5)** varmistaa, että ohjelmistokomponenttien yhteisvaikutus suorituskykyyn vastaa tavoitteita. Järjestelmätasolla tulee käyttää samoja komponenttikohtaisia suorituskykymittareita ja -indikaattoreita. Siten komponenttikehittäjät saavat vertailukelpoista palautetta komponentin suorituskyvystä komponenteille yhtenäisestä referenssiympäristöstä. Referenssiympäristö on tärkeä, koska se mahdollistaa järjestelmätasolla komponenttien suorituskykyvaikutusten keskinäisen arvioinnin. Samalla se luo vapauksia ja säästöjä komponenttien testiympäristöihin ja käytäntöihin, koska komponenttikohtaisien testiympäristöjen ei tarvitse olla yhdenmukaisia suorituskyvyltään.

### 5.4.2 Uuden konseptin käyttöönotto

Uusi prosessikonsepti esiteltiin järjestelmän johtoryhmälle helmikuussa 2016, ja se sai välittömästi sen täyden tuen. Keskeinen viesti oli, että suorituskyvyn hallinta saadaan kuntoon vain nostamalla sen prioriteettia tuotekehityksessä. Riittävä prioriteetti mahdollistaa suorituskykyparannukset sekä panostuksen suorituskykyosaamisen kasvatukseen. Osaamisen kasvu luo puolestaan edellytyksiä ymmärtää nykyinen suorituskyky, mikä on edellytys suorituskyvyn parannukseen. Tutkijan omien kokemuksien perusteella mikään edellä olevista ei yksinään pysty muuttamaan vakiintuneita ajattelumalleja suorituskyvystä. Yhdessä ne kuitenkin voivat muodostaa positiivisen kierteen, joka alkaa ruokkia kuvan 8 mukaisesti itse itseään.

Järjestelmän nykyisen suorituskyvyn ymmärtäminen oli lähtökohta, jotta suorituskyky saadaan hallintaan. Se vaatii komponenttikohtaisien mittareiden tuottamista sekä niiden analysointia. Tuotehallinta vahvisti suorituskyvyn prioriteettia epäsuorasti nostamalla järjestelmän kapasiteettitavoitteiden prioriteettia suhteessa uusiin toiminnallisiin ominaisuuksiin. Kapasiteetin kasvatus edellyttää nykyisen suorituskyvyn tuntemista, jotta muutokset eivät heikennä suorituskykyä. Samalla komponentin suorituskyvyn tutkiminen tukee tekemällä oppimista. Komponenttitiimien käytännön oppiminen on erityisen tärkeää,

koska sama menetelmä ei sovi kaikille komponenteille niiden erilaisuuden vuoksi. Järjestelmätason tiimin täytyy tukea oppimista rinnalla yleisellä ohjeistuksella suorituskyvyn hallinnan periaatteista ja menetelmistä.




***Kuva 8.*** *Suorituskyvyn hallinnan positiivinen kierre*

Jotta varsinainen tavoite komponenttikohtaisesta suorituskyvyn hallinnasta saavutetaan, tarvitaan komponenttikohtainen jatkuva suorituskyvyn analyysi. Käytännössä tämä tarkoittaa myös työmenetelmien kehitystä, jotta vähintään testi ja mittaus saadaan mahdollisimman pitkälle automatisoitua. Tämän tutkimuksen puitteissa ei ollut resursseja käytettäväksi itse menetelmien kehitykseen. Tämä tosiasia rajasi toimenpiteet etupäässä ajatusmalleihin vaikuttamiseen. Suorituskyvyn hallintaprosessikonseptin esittelyn lisäksi komponenttitiimejä vaadittiin jatkamaan jo ensimmäisellä iteraatiokierroksella aloitettua komponenttikohtaisien mittareiden määrittelyä. Uutena vaadittiin komponenttitiimin analyysi oman komponentin suorituskyvyn pullonkauloista ja niiden korjaustavoitteista tavoitekapasiteetin saavuttamiseksi. Lisäksi käynnistettiin järjestelmätason vetämä suorituskyky-yhteisö, jolla yhdistetään järjestelmätason tiimin ja komponenttitiimien suorituskyvystä vastaavat henkilöt. Yhteisön päätavoitteeksi asetettiin tiedonjako.

Jotta muutos ajattelumalleista tapahtuisi, konseptin esittelyssä korostettiin muutoksella tavoiteltavaa hyötyä. Oman komponentin suorituskyvyn rajojen tunteminen edesauttaa komponentin skaalautuvuuden (kohta 3.2) parantamista ja varautumista ylikuormatilanteiden hallintaan. Jatkuva analyysi itsessään vaatii jatkuvan testauksen, joka mahdollistaa ensinnäkin lähes välittömän palautteen ohjelmistonkehittäjälle. Toiseksi, jatkuva komponenttikohtainen testaus vähentää järjestelmätason testauksessa havaittuja vikoja. Lisäksi se kasvattaa koko järjestelmän testikattavuutta, mikä edelleen vähentää asiakkaille päätyvien vikojen määrää. Täysin toteutuessaan komponenttikohtainen jatkuva analyysi nopeuttaa ohjelmistojen julkaisunopeutta. Tutkijan arvion mukaan järjestelmätason suorituskykytestaus voidaan jättää jopa suorittamatta joka julkaisulle, jos komponenttikohtaisien mittaustuloksien pohjalta voidaan selvästi osoittaa järjestelmän suorituskyvyn olevan muuttumaton ohjelmistomuutoksien jälkeenkin. Julkaisujen testauspäätöstä ei voi kuitenkaan tehdä satunnaisesti vaan se pitää perustua arvioon ohjelmistomuutoksen vaikutuksesta koko järjestelmään.

## 5.5 Arvio suorituskyvyn hallinnan muutoksesta

Kuvan 7 uusi prosessikonsepti selkeyttää ohjelmistokomponentin roolia järjestelmän suorituskyvyn hallinnassa, mutta se ei yksin riitä. Tutkijan mielestä vakiintuneiden ajatusmallien ja tapojen muutos vaatii aikaa. Osittainen muutoksen arvioiminen on hankalaa, koska muutos kohdistuu nimenomaan ajatusmalleihin. Suorituskykymittareiden ja jatkuvan analyysin toteuttaminen vaativat työtä, ja tämä työ joutuu kilpailemaan resursseista toiminnallisten vaatimusten toteutuksen kanssa. Muutoksen nopeus ei siis ole välttämättä kiinni yksin motivaatiosta vaan myös suorituskyvylle ja sen hallinnalle annetusta prioriteetista. Riittävän prioriteetin saaminen on kiinni johtoryhmän ja tuotehallinnan edustajien asettamasta painoarvosta suorituskyvyn hallinnan ja muiden ohjelmistokehitystehtävien välillä. Jos suorituskyky koetaan tärkeäksi, myös tavoiteltu suorituskyvyn hallinnan muutos on todennäköisempi.

Muutoksen arvioimiseksi laatuorganisaatio, joka on yksi keskeisistä tuotekehityksen sidosryhmistä, toteutti laatuauditoinnin syksyllä 2016. Koska auditointi toteutettiin osana TL9000-vaatimuksen 8.8.2 vaatimaan yrityksen sisäistä säännöllistä auditointia (QuEST Forum 2016), auditointi tehtiin yritystason suorituskykyprosessia vasten (Kuivalainen 2014). Auditoinnin fokusalueeksi oli valittu suorituskyky, koska se osaltaan tuki suorituskyvyn hallinnan muutosta vakiintuneesta erittäin testauspainotteisesta järjestelmätason tehtävästä kohti komponenttikohtaista suorituskyvyn hallintaa. Vaikka tässä auditoinnissa ei vielä auditoitu organisaatiota uutta prosessikonseptia vasten, auditoinnista saatiin samalla tietoa mahdollisesta muutoksesta kehitysorganisaation suorituskyvyn hallinnan käytännöissä. Osana auditointia haastateltiin niin järjestelmä- kuin komponenttitiimien henkilöitä. Haastatteluihin oli valittu kahdeksan komponenttitiimiä, ja yhteensä 31 henkilöä haastateltiin. Tutkija nimettiin auditoinnin omistajaksi, koska hänellä oli kokonaisvastuu järjestelmän suorituskyvystä. Haastatteluihin tutkija ei osallistunut muutoin kuin yhtenä haastateltavana. Auditoinnin loppuraportissa ei yksilöity toimipistettä tai henkilöä, mistä tieto oli saatu. Se oli laatuorganisaation tietoinen valinta ja se kerrottiin auditoitaville heti haastattelun aluksi. Myös tutkija sai tietoonsa vain loppuraportin, ei yksityiskohtia. (Kuustie ja Kylmäkoski 2016)

Kylmäkosken (2017) mukaan uuden prosessikonseptin mukainen ajattelu ei ollut mennyt vielä täysin läpi organisaatiolle. Esille tuli kuitenkin esimerkkejä siitä, että muutos oli alkanut tapahtua. Suurin osa haastatelluista komponenttitiimeistä oli jollain tasolla alkanut kehittää edellytyksiä komponentin suorituskyvyn ymmärtämiseksi. Tiimeillä oli suunnitelmia suorituskykytestauksen kehittämisestä, uusista suorituskykymittareista ja pullonkaulojen analysoinnista. Osalla tiimeistä oli jo myös käytännön toteutuksia edellä mainituista. Hänen mukaansa auditoinnissa tuli myös esille, että testaus oli edelleen val-

litseva tapa suorituskyvyn analysoinnissa. Joissain tiimeissä oli pyrkimyksiä suorituskykytestauksen automatisointiin. Tämän perusteella näissä tiimeissä oli olemassa halua suorituskyvyn aikaisempaan hallintaan, joka on mahdollistaja jatkuvalle suorituskykyanalyysille. Edelleen Kylmäkosken mukaan parhaimmat komponenttitiimit ovat ottaneet oppia uusista työtavoista. Siihen on kuitenkin matkaa, että valtaosa organisaatiosta toteuttaisi niitä.

Tutkijan omien havaintojen perusteella erityisesti komponenttitiimit, jotka kehittävät palveluja tarjoavia komponentteja ja arvioidaan muiden komponenttien näkökulmasta pullonkauloiksi, ovat edistyneimpiä muutoksen toteuttamisessa. Heidän intressinsä on tehdä nähtäväksi palvelevan komponentin työkuorma ja osoittaa oman komponentin vaikutus koko järjestelmän suorituskyvylle. Vastaavasti palvelua hyödyntäviä komponentteja kehittävät tiimit eivät välttämättä tunnista todellista tarvetta suorituskyvyn hallinnalle. Lisäksi edelleen löytyy myös komponenttitiimejä, jotka näkevät suorituskyvyn hallinnan puhtaasti järjestelmätason asiana. Näiden havaintojen perusteella täytyy edelleen parantaa ohjeistuksen argumentointia, miksi komponenttikohtainen suorituskyvyn hallinta on avain koko järjestelmän suorituskyvyn hallintaan. Vertauskuvallisesti voisi sanoa, että Ferrari vaatii Ferrarin osat. Ferrarin suorituskyky ei saa kaatua siihen, että auton voimansiirto ei kestä rasitusta tai auton hallittavuus ei täytä käyttäjän odotuksia. Yksittäiset osat eli komponentit ovat avainroolissa koko järjestelmän suorituskyvyn näkökulmasta.

Iteraatiokierroksen onnistumisiin voi laskea komponenttitiimien kanssa syntyneen vuoropuhelun. Esimerkiksi jatkuvan analyysin osalta menetelmät on paras ratkaista komponenttitiimien toimesta. Järjestelmätason tiimi pystyy auttamaan menetelmäkehityksessä oman kokemusperäisen tietonsa pohjalta. Vastaavasti viikoittain kokoontuva suorituskyky-yhteisö ruokkii edelleen muutosta ja pystyy jakamaan tietoa hyväksi havaituista ratkaisuista komponenttitiimien ja järjestelmätiimin kesken. Keskustelun tärkeyttä korostaa seuraava havainto uuden suorituskyvyn hallintakonseptin kehityksen aikana. Keskustelussa tutkijan ja komponenttitiimin jäsenen kesken syntyi havainto siitä, että suorituskyvyn hallinta noudatti edelleen täysin ohjelmistokehityksen vesiputouskehitysmallia, vaikka organisaation muutoin käytti ketterän kehityksen menetelmiä. Sinällään havainto on itsestäänselvyys kohdassa 2.2 kuvatun nykyisen prosessin valossa, mutta järjestelmätiimi ei kyennyt näkemään sitä aiemmin. Tämä havainto jo tutkimuksen alkuvaiheessa olisi auttanut suuntaamaan kehitysajatukset suoraan tähän ongelmakohtaan.

# 6. YHTEENVETO

Tutkimuksen ongelmasta johdetun **päähypoteesin** mukaan suorituskykyongelmat voidaan havaita aiemmin kehittämällä edelleen suorituskyvyn hallintaprosessia ja huolehtimalla sen noudattamisesta. Tämä hypoteesi tuli osoitettua toteen. Tutkimuksen aikana tuli selvästi esille, että suorituskykytestaus noudatti puhtaasti ohjelmistokehityksen vesiputousmallia niin komponentti- kuin järjestelmätasolla. Käytännössä suorituskykytestaus tehtiin komponenttikehityksen loppuvaiheessa kerran, jolloin suunnittelija sai palautteen toteutuneesta suorituskyvystä hyvin myöhään. Tutkimuksen tuloksena toteutettu uusi suorituskyvyn hallintakonsepti, jonka avainkohtana on jatkuva suorituskykyanalyysi, mahdollistaa lähes välittömän palautteen toteuttajalle. Uusi konsepti oli samalla vastaus yhteen tutkimuskysymykseen: ”Miten prosessia tulisi kehittää, jotta suorituskykyongelmat tunnistetaan ja ratkaistaan aiemmin?”.

Toinen tutkimuskysymys kuului: ”Onko organisaatiolla tarvittavaa osaamista prosessin noudattamiseksi?”. Tutkimuskysymyksestä oli johdettu edelleen **ensimmäisen alihypoteesi**, jonka mukaan ”tuotekehitysorganisaatiolla ei ole tarvittavaa osaamista suorituskyvyn hallintaprosessin noudattamiseen”. Tämä tuli osoitettua toteen sekä tutkijan omien havaintojen että haastattelujen kautta. Asian korjaamiseksi todettiin tarve korkeammalle suorituskyvyn prioriteetille, osaamisen kasvatukselle ja nykyisen suorituskyvyn mittaamiselle. Tämä muutos aloitettiin tutkimuksen loppuvaiheessa ja jatkuu tutkimuksen jälkeen. **Toisen alihypoteesin** mukaan ”tuotekehitysorganisaation toimintatavoissa ei ymmärretä tarpeeksi hyvin suorituskykyvaatimusten ei-toiminnallista luonnetta, jolloin niitä on pyritty hallitsemaan samoin keinoin kuin toiminnallisia vaatimuksia”. Tämä alihypoteesi ei pitänyt paikkaansa. Suorituskykyä ei yritetty hallita kuten toiminnallisia vaatimuksia. Tosin sen hallinta muutenkin oli heikoissa kantimissa, kuten myös haastattelutulokset osoittivat.

Kolmannen tutkimusongelman avulla haettiin vastausta siihen, ”Millaisia ongelmia nykyprosessissa on, ja mitkä seikat mahdollisesti estävät sen noudattamista”. Tutkimuksessa tuli selväksi se, että suurimmat ongelmat ovat prioriteetin, osaamisen ja prosessin noudattamisen suhteen. Lukuun ottamatta noudatettua vesiputousmallia, joka hidasti palautteen antamista ohjelmistokehittäjälle, prosessit tai työkalut itsessään eivät olleet ongelma. Suorituskyvyn hallinnan säännönmukainen toteuttaminen aina vaatimuksista testaukseen on avainasemassa suorituskykyongelmien estämiseksi. Kun suorituskykyä ei hallita systemaattisesti, sen pitkäjänteinen kehitys kärsii. Tällöin myös menetelmät, kuten

suorituskyvyn systemaattinen mittaus, voivat jäädä puutteelliseksi. Tämä taas johtaa kierteeseen, jossa käsitys suorituskyvystä karkaa entistä kauemmas järjestelmän laajentuessa ja monimutkaistuessa. Siten myös ei-toiminnallisen suorituskyvyn osalta voidaan puhua teknisestä velasta (engl. technical gap).

Kriittisesti katsoen tutkimuksen alussa olisi tullut määrittää tarkemmat mittarit, miten kokonaismuutosta mitataan. Jatkossa tähän tulee kiinnittää huomiota, koska mitattava muutos toisaalta motivoi tekijöitä ja sidosryhmiä, ja toisaalta helpottaa tarvittavien menetelmäkehityksen investointien toteuttamista. Samoin perustein voi kritisoida kehitettyä suorituskyvyn hallintaprosessikonseptia. Vaikkakin jatkuva ja nopea suorituskykyanalyysi on epäilemättä tavoiteltava ominaisuus, ei ole yksiselitteistä, tulisiko suorituskyky mitata komponentti- vai järjestelmätiimin toimesta. On toki mahdollista nopeuttaa suorituskykypalautetta nopeuttamalla itse ohjelmistokehityssykliä muodikkaiden DevOps periaatteiden mukaisesti. DevOpsille ei ole olemassa yksiselitteistä määritelmää, mutta usein siitä puhuttaessa nimenomaan korostuu ohjelmiston muutoksien nopea julkaisutahti.

Työn lähtökohtana oli tavoite järjestelmän paremmasta suorituskyvyn hallinnasta. Koska suorituskyvyn tunteminen on edellytys sen hallinnalle, keskeiseksi keinoksi valikoitui suorituskykymittarit ja niiden mittaus. Teorian näkökulmasta tämä oli järkevä valinta. Kirjoitustyön viimeistelyvaiheessa ja osallistuvan havainnoinnin edelleen jatkuessa heräsi kuitenkin tutkijan oma epäily, oliko valittu näkökulma paras mahdollinen itse suorituskykyhaasteiden ratkaisemiseen. Vaihtoehtoinen näkökulma olisi voinut olla keskittyminen järjestelmän ja sen ohjelmistokomponenttien skaalautumiseen. Vaikka skaalautuminen ei takaa optimaalista läpäisykykyä ja nopeita vasteaikoja, skaalautuminen on konkreettisempi asia komponenttitiimeille. Toiseksi skaalautuminen pienentää riskiä isoihin suorituskykyongelmiin, koska virtualisointi-infrastruktuuri kykenee lisäämään virtuaaliresursseja kuormituksen kasvaessa. Kolmanneksi suorituskykymittareiden tunnistaminen oli vaikeaa komponenttitiimeille – suorituskyky ja erityisesti sen mallinnus vaativat erikoisosaamista. Epäilys heräsi siksi, että virtualisointi on muuttunut viime vuosina jokapäiväiseksi ja puhtaasti suorituskyvyn hallintaan liittyvien uusien julkaisujen määrä on ollut vähäinen. Varmasti muuallakin pohditaan, onko investointi yksityiskohtaiseen suorituskyvyn hallintaan varmasti rahan arvoista. Investointipäätöstä tehtäessä kannattaa tarkastella ainakin järjestelmän aikakriittisiä käyttötapauksia ja niiden määrää. Tätä kannattaa myös tutkielman lukijan pohtia oman ohjelmistojärjestelmänsä osalta ennen syöksymistä suorituskykymittareiden syövereihin!

# LÄHDELUETTELO

**Julkiset lähteet**

Agile. (2001). Agile Manifesto. Saatavilla: http://agilemanifesto.org.

Bondi, A. (2015). Foundations of software and system performance engineering. Addison Wesley.

Dick, B. (1993). You want to do an action research thesis? Saatavilla: http://www.aral.com.au/resources/arthesis.html

Dynatrace. (2016). Virtualization's Impact on Performance Management. Dynatrace LLC. Saatavilla: https://www.dynatrace.com/resources/ebooks/javabook/why-virtualization-has-impact-on-performance-management/. Viitattu: 20.2.2017.

Gunther, N. (2015). Performance Analysis vs. Capacity Planning. The Pith of Performance- Blog. Saatavilla: http://perfdynamics.blogspot.fi/2015/03/performance-analysis-vs-capacity.html. Viitattu: 25.2.2017.

Haikala, I., Mikkonen, T. (2011). Ohjelmistotuotannon käytännöt. Talentum.

Järvenpää, E. (2006). Laadullinen tutkimus. Teknillinen korkeakoulu. saatavilla: http://www.cs.tut.fi/~ihtesem/k2007/materiaali/luento4.pdf. Viitattu: 30.11.2016.

Kleinrock, L. (1975). Queueing Systems, Volume 1: Theory. Wiley

QuEST Forum. (2016). Quality Management System, Requirements Handbook. TL 9000:2016 (R6). Quality Excellence for Suppliers of Telecommunications Forum.

Saaranen-Kauppinen, A., Puusniekka, A. (2006). KvaliMOTV - Menetelmäopetuksen tietovaranto [verkkojulkaisu]. Tampere: Yhteiskuntatieteellinen tietoarkisto [ylläpitäjä ja tuottaja]. Saatavilla: http://www.fsd.uta.fi/menetelmaopetus/kvali/L6_4.html. Viitattu: 20.3.2017.

Smith, C., Williams, L. (2001). Performance solutions, A practical guide to creating responsive, scalable software. Addison Wesley.

Sommerville, I. (2011). Software Engineering. Pearson.

Tuovinen, A-P. (2014). Ohjelmistojen suorituskyky, Kurssin tavoitteet. Helsingin yliopisto, tietojenkäsittelytieteen laitos. Saatavilla: https://www.cs.helsinki.fi/u/aptuovin/spe/k14/luento_1.pdf. Viitattu: 10.1.2017.

VMware. (2017). Virtualization overview. WMware. Saatavilla: http://www.vmware.com/uk/solutions/virtualization.html. Viitattu: 20.2.2017.

**Yrityksen sisäiset lähteet**

Durairaj, A. (2013). Performance requirements - Communication_material_2013. Kohdeyritys. Saatavilla: Yrityksen intranet.

Heinonen, J. (2015). Performance requirements in A360. Kohdeyritys. Saatavilla: Yrityksen intranet.

Häkkinen, J. (2015). Performance Indicator Presentation. Kohdeyritys. Saatavilla: Yrityksen intranet.

Hämäläinen, O. (2014). Workshop Q1/2015 targets 17122014. Kohdeyritys. Saatavilla: Yrityksen intranet.

Hämäläinen, O. (2015a). Interviews. Kohdeyritys. Saatavilla: Yrityksen intranet.

Hämäläinen, O. (2015b). Interviews_schedule. Kohdeyritys. Saatavilla: Yrityksen intranet.

Hämäläinen, O. (2015c). Performance News: What on earth is happening? Kohdeyritys. Saatavilla: Yrityksen intranet.

Hämäläinen, O. (2015d). Performance Action Plan 2015. Kohdeyritys. Saatavilla: Yrityksen intranet.

Kuivalainen, M. (2014). Performance management process. Kohdeyritys. Saatavilla: Yrityksen intranet.

Kylmäkoski, R. (2017). Riku Kylmäkoski, laatupäällikkö, kohdeyritys. Haastattelu 24.3.2017.

Kuustie, R. Kylmäkoski, R. (2016). Performance management audit - final audit report. Kohdeyritys. Saatavilla: Yrityksen intranet.

Nagy, P. (2017). Way of working. Kohdeyritys. Saatavilla: Yrityksen intranet.

# LIITE 1 HAASTATTELUKYSYMYKSET

## Questions which could be announced beforehand:

- What does performance as term is telling for you?
- How the performance or availability related functionality get inside of your component/product?
- How you do performance design and plan it?
- Do you use performance requirements in that process?
- What you do to ensure your product will not cause overload, downtime, unavailability or downtime of a system?
- How do you ensure your product can handle proper amount of data?
- How do you know that performance of your product is good enough?
- How do you know the conditions under which your product shall operate?

## Questions for actual interview:

Prioritization:

Questions in bold black are priority 1. Questions in black are priority 2.

### Role: Tester/developer (implementation & testing)

- What does performance as term is telling for you?
- How do you do performance design and how do you plan it?
    - How the performance or availability related functionality get inside of your component/product?
    - How do you get performance requirements for your component?
        - Are the performance requirements scalable?
        - Do you think your team has qualifications to design good, clean code which also performs well? Do you know a person in your development site you can always discuss such design-related topics?
    - How is it guaranteed that overload situation doesn't cause outages?
        - How the overload situations ware handled in the past
- Do you do formal test planning and run actual tests for performance, availability, overload areas?
    - What is the priority of performance improvements in your component development process?
    - What is a cost of performance tests (how many people are involved in performance testing and for how long)?
    - Where do you get simulators for your testing? How much time it cost to maintain that
    - How do you get test lab for your testing?
- What type of performance testing improvements you are planning or like to propose?
- What are your best practices you like to share for all?
- What you think is missing to make proper performance work?

### Role: Architect (system design)

- What does performance as term is telling for you?
- How do you do performance design and how do you plan it?
    - How the performance or availability related functionality get inside of your component/product?
    - How is it guaranteed that overload situation doesn't cause outages?

    - How the overload situations are considered?
- Do you use performance/availability requirements in that process?
    - How do you define performance requirements for your component?
- Are the performance characteristics of depended components known to you? Are you able to provide performance characteristics of own component to other teams?
- Are the performance requirements scalable?
- What are your best practices you like to share for all?
- What you think is missing to make proper performance work?

## Role: Component team leader & product owner (product management & execution)

- How the performance or availability related functionality get inside of your product?
    - Are the performance aspects part of your DoD for the developed features?
    - Do you have time allocated for performance improvements of your components?
    - What is the priority of performance and high availability improvements in your backlog?
    - Is performance testing part of master test plan?
    - What are your best practices you like to share for all?
    - What you think is missing to make proper performance work?
- Do you use performance requirements/availability in that process?
    - How do you define performance requirements for your component?

# LIITE 2 HAASTATTELULÖYDÖKSET

| Area | Category | Finding | Actions | Responsible | Votes |
|---|---|---|---|---|---|
| PO & Architecture & requirements<br>PO = product owner | Co-operation | Co-operation and collaboration between sites is not properly working or properly supporting each other, e.g. due to lack of understanding of other components/functionalities needs/behaviours. | •Joint meetings to clarify responsibilities between teams shall be organized.<br>•Understand of interfacing between functionalities shall be increased.<br>•Co-operation and collaboration possibilities shall be identified. | Component project leader & development teams | 4 |
| PO & Architecture & requirements | Co-operation | Multiple functionalities may require services from a common service which basically requires deeper understanding of inter-component requirements. | •Awareness of the shared functionality usage shall be increased.<br>•Performance (throughput and processing capacity related) requirements shall be revisited. | POs & Architects & development teams | 1 |
| Design & implementation | Co-operation | Role of system performance team is not clear for teams. system performance team collects irrelevant data from team perspective. Component teams expect stress test to be performed and operation times & stability to be measured. system performance team not providing valuable comments for component performance test plan and reports. | •Clarify system performance team role and competences.<br>•Individual meetings with value streams to clarify their expectations.<br>•Transfer competences from applications to System performance team.<br>•Dedicated person for application. | System performance team | 0 |
| Design & implementation | Co-operation | Interaction between components (different components are not seen as customers). | No actions defined for these items at this point. | System performance team | |
| Testing | Co-operation | Communication on performance issues between component teams have been (more or less) missing. | No actions defined for these items at this point. | | 0 |
| Testing | Knowledge, Priority | It's not fully understood what performance engineering means for components.<br><br>Lack of competence and time to improve performance testing competence.<br><br>It's not fully understood what should be tested and how those tests should be done.<br><br>Missing guides & support for component performance engineering. | •Increase performance engineering knowledge<br>•Clarify responsibilities, objectives & scope of the Performance engineering. System performance team vs. component teams & between component teams.<br>•Arrange training (Material & Training sessions)<br>•Follow up<br><br>•Prepare basic guidelines for component teams how to cope with system wide services & features. | System performance team | 7 |
| Design & implementation | Knowledge, Priority | No performance aspects in User Stories' Conditions of Satisfaction and in requirements. Limited knowledge about how the customer uses application and how the system will be loaded.<br><br>Overload protection mechanisms are implemented after pronto. Usually developers don't plan such mechanisms or believe that overload protection already exists. Usually developer conceders performance only in terms of speed. | •Performance should be included in DoD.<br>•Performance analysis should be included in FC pre-study.<br>•Improve study quality.<br>•Study should be standardized. | PO & Architect | 4 |

| | | | | | |
|---|---|---|---|---|---|
| Design & implementation | Priority | Priority for performance improvements is low. No time for competence build up in performance area (materials available, but no common knowledge about it). | •Invest time for performance improvements.<br>•Treat performance as a feature. (http://blog.codinghorror.com/performance-is-a-feature/)<br>"Make Performance a Point of (Public) Pride".<br>•In each sprint team can use 10% of time for improvements, Add another 10% only for performance improvements. | PO and Component project leader | 3 |
| Design & implementation | Priority | System architectural changes are needed. Rigid architecture forces developers to use inefficient APIs and technologies. Different components have different needs. | •Propose architectural renewal keeping modularity and independence in mind.<br>•Part of this requests might be results that available resources are misused or developers don't know how to optimize available resources. There are missing DB experts and application server domains. We should build strong competences in these areas on each site, so experts are available locally. | System and component level architects | 3 |
| PO & Architecture & requirements | Process | Some component specific requirements have been dropped or forgotten over to last couple of years, or even have not been existing at all on certain areas. | • Requirement workshops shall be arranged and results reviewed with relevant stakeholders.<br>•Unambiguous requirements shall be removed. | Quality & POs & Architects & development teams | 2 |
| PO & Architecture & requirements | Process | In some areas, proper performance or availability related features candidates are not existing or are ignored (e.g. overload protecting or scalability related FCs). | • Proper performance, availability, scalability etc. feature component creation shall be taken in place.<br>•Pre-studies shall cover also non-functional requirements properly. | Quality & POs & Architects & development teams | 2 |
| Design & implementation | Test capability | Late or no feedback about performance issues, because tests are executed rarely. Usually there is no component performance tests only e2e. No automation (no automatic environment setup, test execution, report generation). | •Invest time in performance test automation, and integration with CI.<br>•Test performance on different test levels.<br>•Prepare guideline how to test performance in small scale (on unit or module test level).<br>•Define basic performance KPI's for component, and track them through releases. | PO, Development team and Architect | 3 |
| Testing | Test capability | Test preparation, test Analysis, test reporting on customer like environment is currently time consuming | •Development of the easy (one click) test environment building system which can produce customer like system (or whatever is needed)<br>•Following parts would be needed<br>•Lab reservation & product installation<br>•Simulator installation<br>•Test tool (Data collection & monitoring) installation<br>•Analysing & reporting | System performance team | 2 |
| Testing | Test capability | No comprehensive and scalable simulators available for end to end functionalities<br><br>Common test tools for performance PET topics are missing. | •Improve simulator road mapping as part of the requirement management. As result of simulator roadmap, we should know what simulators are needed and when those should be available.<br>•In parallel of simulator road mapping we should plan data collection/monitoring tool content. | System performance team | 1 |
| Testing | Test capability | | •Develop light weight simulators to simulate e.g. certain e-2-e functionality for testing purposes | Component teams & system performance team | 0 |
| Testing | Test capability | Test environment (labs, simulator, monitoring tools) stability is a concern. | No actions defined for these items at this point. | | |