Jenni Kallio

VAATIMUSTENHALLINTA JA SEN KEHITTÄMINEN OHJELMISTON ELINKAAREN NÄKÖKULMASTA

Tietojärjestelmätieteen
pro gradu -tutkielma
12.6.2008

Jyväskylän yliopisto
Tietojenkäsittelytieteiden laitos
Jyväskylä

TIIVISTELMÄ

Kallio, Jenni Katariina
Vaatimustenhallinta ja sen kehittäminen ohjelmiston elinkaaren näkökulmasta/ Jenni Kallio
Jyväskylä: Jyväskylän yliopisto, 2008.
131 s.

Tietojärjestelmätieteen pro gradu -tutkielma

Tutkielmassa tarkastellaan vaatimustenhallintaa ja sen kehittämistä ohjelmiston elinkaaren näkökulmasta. Vaatimusmäärittelyyn yleisesti ja vaatimustenhallintaan tarkemmin tutustutaan aluksi kirjallisuuden pohjalta. Tutkielmassa kuvaillaan vaatimustenhallinnan keskeisiä osa-alueita kuten muutostenhallintaa ja vaatimusten jäljittämistä sekä esitellään ohjelmiston ylläpidon aikaan painottuva elinkaarimalli. Kirjallisuuskatsauksen perusteella muodostetaan viitekehys, jossa käsitellään vaatimustenhallintaa muutospyyntöjen kautta. Empiirinen osa toteutetaan tapaustutkimuksena ja sen perustana käytetään muodostettua viitekehystä.

Empiirisessä osassa esitellään kohdeorganisaation vaatimustenhallinnan tilaa ja kypsyystasoa sekä rakennetaan muutospyyntöjen järjestelmällisen arvioinnin tueksi päätöspuita. Myös kohdeorganisaation eri elinkaaren vaiheissa olevia ohjelmistoja ja niiden vaatimustenhallintaa vertaillaan keskenään ja tutkitaan, missä määrin ohjelmiston elinkaaren vaihe vaikuttaa vaatimustenhallintaan.

Tapaustutkimuksen tuloksena havaittiin, että pelkkä kokonaisen ohjelmiston elinkaaren tarkastelu ei välttämättä anna riittävää kokonaiskuvaa. Ohjelmiston elinkaaren tarkastelussa on mahdollista ottaa huomioon esimerkiksi ohjelmistoversion tai -ominaisuuden elinkaari kattavamman kokonaiskuvan saamiseksi. Tutkimuksessa havaittiin myös elinkaaren näkökulman voivan auttaa ohjelmistoperhekokonaisuuksien hallinnassa.

Lisäksi tutkimuksessa havaittiin asiakkuuden olevan hyvin keskeisessä asemassa vaatimustenhallintaa tarkasteltaessa. Asiakkuuden on havaittu olevan kohdeorganisaatiossa useissa tapauksissa merkityksellisempi esimerkiksi muutospyynnön arvioinnissa kuin ohjelmiston elinkaaren vaiheen. Sen vuoksi empiirisen osan perusteella muokatun viitekehyksen elinkaaren näkökulmaan on lisätty keskeiseen osaan asiakkuuden elinkaari, joka saattaa sisältää useita ohjelmistoja ja niiden elinkaaria. On kuitenkin huomioitava, että tämän tutkimuksen tulokset eivät ole yleistettäviä. Tutkimuksen tarkoituksena on enemmänkin kiinnittää huomiota vaatimustenhallinnan kehittämiseen systemaattisemmaksi.

AVAINSANAT: vaatimustenhallinta, ohjelmiston elinkaari, elinkaarimallit, muutostenhallinta, vaatimusten jäljittäminen

ABSTRACT

Kallio, Jenni Katariina
Vaatimustenhallinta ja sen kehittäminen ohjelmiston elinkaaren näkökulmasta/ Jenni Kallio
Jyväskylä: University of Jyväskylä, 2008.
131 pages

Master's thesis in Information Systems Science

This study examines developing requirements management from the software life cycle's perspective. Requirements engineering generally and requirements management in more detail are first examined based on a literature review. The study describes the main parts of requirements management, for instance change management and requirements tracing. Also the life cycle model emphasizing maintenance is presented. Based on the literature review it is formed a framework which is utilized in the empirical part of the study. The research method of the empirical part is case study.

In the empirical part of the study the maturity of requirements management is evaluated in a case organization and decision trees are built up to make evaluation of change requests more systematic. In addition, a couple of softwares are compared in order to find out to what extent the phase of life cycle affects requirements management.

The results of the study include the following. Studying only the life cycle of the whole software is not enough. For example the life cycle of the software version and the life cycle of the software feature can also be considered to get a more comprehensive view. Life cycle perspective may help in managing software product families.

Customership has an important role in requirement management. It seems that in the case organization customership is in many cases more important than the phase of the software life cycle when assessing for example the change requests. That is why the life cycle of customership has been added to the central position in the framework of requirements management. However, it is important to notice that the results should not be generalized. Instead of generalization, the aim is to pay attention on developing requirements management to be more systematic.

KEYWORDS: requirements management, software life cycle, life cycle models, change management, requirements tracing

SISÄLLYSLUETTELO

KUVIOT

TAULUKOT

1 JOHDANTO

Käyttäjillä ja muilla sidosryhmillä on vaatimuksia, joihin ohjelmiston on kyettävä vastaamaan. Jotta ohjelmisto voi olla hyödyllinen, sen täytyy sopia tarkoitukseensa ja näin ollen täyttää sille asetetut vaatimukset. Vaatimukset ovat keskeisessä asemassa vaatimusmäärittelyprosessissa ja vaatimustenhallinnassa. Cheng ja Atlee (2007) määrittelevät vaatimusmäärittelyn (*requirements engineering*) prosessiksi, jossa vaatimukset päätetään. Heidän mukaansa vaatimustenhallinta (*requirements management*) on osa tätä vaatimusmäärittelyprosessia.

Vaatimusmäärittely on ohjelmistotekniikan ala, joka tutkii ohjelmiston toimintojen ja rajoitusten tavoitteita (Zave 1997). Vaatimusmäärittelyssä selvitetään tarkoitus, johon ohjelmistoa aiotaan käyttää. Tähän tarkoitukseen pyritään tunnistamalla sidosryhmät ja niiden tarpeet sekä dokumentoimalla ne sellaiseen muotoon, joka mahdollistaa analysoinnin, kommunikoinnin ja lopulta myös toteutuksen. (Nuseibeh & Easterbrook 2000)

Vaatimusmäärittelyprosessin keskeisiä vaiheita ovat vaatimusten tunnistaminen, mallintaminen, vaatimusanalyysi, vaatimusten vahvistaminen ja varmentaminen sekä vaatimustenhallinta (Cheng & Atlee 2007). Tässä tutkielmassa keskitytään vaatimustenhallintaan, jonka merkitys painottuu vaatimusmäärittelyprosessin loppuvaiheessa.

Vaatimustenhallinta on vaatimusten tunnistamiseen, polveutumiseen, analysointiin, koordinointiin, versiointiin ja jäljittämiseen liittyvän tiedon organisoimista ja hallintaa koko ohjelmiston elinkaaren ajan (Hoffmann, Kuhn, Weber & Bittner 2004). Vaatimustenhallinnan keskeisiä toimintoja ovatkin muutostenhallinta, versionhallinta sekä vaatimusten tilojen seuranta ja jäljittäminen (Wiegers 2003, 314).

Tutkielmassa tarkastellaan muutospyyntöjen kautta ohjelmiston elinkaaren vaiheiden merkitystä vaatimustenhallinnassa. Elinkaaresta korostuu erityisesti

ylläpito, joka on koko ohjelmiston elinkaareen suhteutettuna merkittävä sekä ajallisesti että kustannusten näkökulmasta. Esimerkiksi Sommervillen (1998, 673) mukaan ohjelmiston ylläpitokustannukset ylittävät yleensä ohjelmiston kehittämiskustannukset. Ylläpito on kallista sen vuoksi, että vaatimukset ja ympäristö muuttuvat (Pigoski 2002).

Tutkimusongelmana on selvittää, *miten elinkaariajattelua voidaan hyödyntää kehitettäessä vaatimustenhallintaa*. Tutkimuskysymyksinä ovat

1. Miten ohjelmiston elinkaaren vaihe vaikuttaa vaatimustenhallintaan?
2. Miten vaatimustenhallintaan (ja sen kypsyystasoon) ohjelmiston myöhemmissä vaiheissa vaikuttaa se, miten vaatimustenhallinta on hoidettu ohjelmiston elinkaaren alussa?
3. Miten vaatimustenhallintaa olisi mahdollista kehittää ohjelmiston elinkaaren näkökulmasta?

Tutkielman alkuosan kirjallisuuskatsauksen perusteella rakennetaan viitekehys, joka rakentuu tutkimusongelman ympärille. Tutkielman empiirinen osa toteutetaan kvalitatiivisena tapaustutkimuksena niin, että empiirinen osa, erityisesti haastattelut perustuvat viitekehyksen aihealueisiin. Empiirisen osan perusteella testataan kirjallisuuden pohjalta kootun viitekehyksen toimivuutta käytännössä ja lisäksi pyritään laajentamaan viitekehystä empiirisessä osassa saatujen tietojen perusteella.

Tutkielman tarkoituksena on tukea vaatimustenhallinnan kehittämistä systemaattisemmaksi. Tähän pyritään rakentamalla viitekehys, jota voidaan käyttää apuna, kun esimerkiksi ohjelmistoyrityksessä kehitetään vaatimustenhallintaa. Tutkielmassa esitettävän viitekehyksen ja tulosten ei ole tarkoituskaan olla kattavia, vaan keskeisintä on tukea vaatimustenhallinnan merkityksen hahmottamista sekä vaatimustenhallinnan kehittämistä järjestelmällisempään suuntaan.

Tutkielmassa käsitellään lyhyesti koko vaatimusmäärittelyprosessia, mutta pääpaino on vaatimustenhallinnassa, joka tapahtuu ylläpidon aikana. Näin ol-

len myös elinkaarimalleissa ja elinkaaren hallinnassa keskitytään erityisesti ylläpitoon.

Luvussa 2 käsitellään vaatimusmäärittelyä yleisellä tasolla. Luvussa havainnollistetaan sitä, miten vaatimustenhallinta liittyy vaatimusmäärittelyyn ja mitä muuta kuin vaatimustenhallinta kuuluu vaatimusmäärittelyyn. Ennen vaatimusmäärittelyn kuvailemista luvussa kuitenkin selitetään, mitä vaatimuksilla tarkoitetaan ja pohditaan sitä, millaisia piirteitä on laadukkailla vaatimuksilla.

Luvussa 3 keskitytään tarkemmin nimenomaan vaatimustenhallintaan. Luvussa esitellään vaatimustenhallinnan keskeisiä toimintoja, kuten muutostenhallintaa, version- ja konfiguraationhallintaa, vaatimusten jäljittämistä ja vaatimusten tilojen seurantaa. Luvun lopuksi käsitellään myös lyhyesti vaatimustenhallinnan työkaluja. Jo ennakoiden lukua 4, luvun 3 lopussa yhteenvedossa kootaan yhteen luvun keskeiset asiat ohjelmiston elinkaaren näkökulmasta.

Luvussa 4 tutustutaan elinkaarimalleihin ja elinkaaren hallintaan. Elinkaarimalleja ja elinkaaren hallintaa esitellään sen vuoksi, että tässä tutkielmassa käsitellään vaatimustenhallintaa tutkien sitä nimenomaan ohjelmiston elinkaaren näkökulmasta. Luvussa 4 kuvaillaan lyhyesti muutamia elinkaarimalleja, erityisesti Bennettin ja Rajlichin (2000a; 2000b) elinkaarimallia, jonka painopiste on ohjelmiston käyttöönoton jälkeisessä ajassa. Luvun loppuosassa kerrotaan myös lyhyesti ohjelmiston elinkaaren kustannusten arvioinnista.

Lukuun 5 kootaan viitekehys vaatimustenhallintaan käyttäen pohjana aiempia kirjallisuuskatsauslukuja. Viitekehystä käytetään myös empiirisen osan perustana ja sitä muokataan empiirisessä osassa tehtyjen havaintojen perusteella.

6. luvussa esitellään tapaustutkimuksena toteutetun empiirisen osan tiedonkeruutavat, tapaustutkimuksen tausta ja tavoitteet sekä kohdeorganisaatio ja sen rajaukset. Tutkimuksen kohteena on Digia Samstock -ohjelmistojen vaatimustenhallinta.

Luvussa 7 raportoidaan tutkimuksessa saadut tulokset. Luvussa analysoidaan muun muassa Digia Samstock -ohjelmistojen tilannetta SWOT-analyysin avulla sekä vertaillaan elinkaarellaan eri vaiheissa olevien ohjelmistojen vaatimustenhallintaa sekä rakennetaan päätöspuita muutospyyntöjä koskevan päätöksenteon tueksi. Luvussa esitetään myös erilaisia näkemyksiä ohjelmiston elinkaariajatteluun. Luvun lopussa esitellään empiirisen osan perusteella muokattu viitekehys.

Yhteenvetolukuun kootaan tiiviisti tutkimuksen keskeisimmät asiat ja arvioidaan, millaisia vastauksia tutkimuskysymyksiin on saatu. Yhteenvedossa esitellään myös jatkotutkimusaiheita.

2 VAATIMUSTENHALLINTA OSANA VAATIMUSMÄÄRITTELYPROSESSIA

Vaatimusmäärittelyn käsite on lähtökohtaisesti hyvin laaja (esim. Zave 1997). Cheng ja Atlee (2007) määrittelevät vaatimusmäärittelyn prosessiksi, jossa vaatimukset päätetään. Heidän mukaansa vaatimustenhallinta on osa tätä vaatimusmäärittelyprosessia. Myös tässä tutkielmassa vaatimustenhallintaa pidetään yhtenä vaatimusmäärittelyprosessin osana.

Vaatimusmäärittely on tärkeää, koska siinä päätetään tavoitteista, joita hankittavalta ohjelmistolta vaaditaan. Laadukkaiden vaatimusten määrittely on vaikeaa, mutta hyvin tärkeää (van Lamsweerde 2000). Vaatimusmäärittelyssä onnistuminen vaatiikin käyttäjien, asiakkaiden ja muiden sidosryhmien tarpeiden ymmärtämistä sekä ohjelmiston taustan ja ympäristön hahmottamista (Cheng & Atlee 2007).

Tässä luvussa esitellään vaatimuskäsitettä, vaatimusmäärittelyprosessia sekä sen osia ja haasteita, jotta on helpompi ymmärtää kokonaisuutta, johon vaatimustenhallinta liittyy. Luvun lopuksi kootaan yhteenveto luvun keskeisimmästä sisällöstä.

2.1 Vaatimukset ja vaatimustyypit

Käyttäjillä ja muilla sidosryhmillä on vaatimuksia, joihin hankittavan ohjelmiston on kyettävä vastaamaan. Jotta ohjelmisto voi olla hyödyllinen, täytyy sen sopia tarkoitukseensa ja täyttää sille asetettuja vaatimuksia. Esimerkiksi Nuseibehin ja Easterbrookin (2000) mukaan ohjelmiston tärkein menestyksen mittari onkin se, kuinka hyvin se sopii tarkoitukseen, johon se on hankittu.

Vaatimukset voivat aluksi olla toteamuksia ohjelmiston tavoitteista ja käyttäjien toiveita luonnollisella kielellä ilmaistuina, mutta toiveista ja toteamuksista on muokattava virallisempia määrityksiä (NATURE Team 1996). Vaatimuksia täy-

tyy valita, priorisoida, rajata ja niistä täytyy neuvotella (van Lamsweerde 2000). Ohjelmistokehitysprojekteissa on yleensä enemmän vaatimusehdotuksia kuin mitä aikaresurssit sallivat ja mistä asiakkaat ovat halukkaita maksamaan (esim. Karlsson 1996). Siksi vaatimusten priorisointi on keskeinen osa vaatimustenhallintaa.

Saiedian ja Dale (2000) perustelevat vaatimusten tärkeyttä ohjelmistoprojektissa sillä, että ilman tarkasti määriteltyjä vaatimuksia ohjelmiston kehittäjät eivät tiedä mitä tehdä, eivätkä käyttäjät tiedä mitä odottaa. Heidän mukaansa ilman tarkkoja vaatimuksia on myös mahdotonta arvioida, täyttääkö ohjelmisto käyttäjien tarpeet. Vaatimukset ovat siis keskeisessä asemassa sekä ohjelmiston toimittajan että asiakkaan näkökulmasta.

Vaatimuskäsitteelle on olemassa monia määritelmiä. IEEE:n sanastossa (IEEE Standard Glossary of Software Engineering Terminology 1990, 62) on määritelty termi vaatimus seuraavasti:

> 1. edellytykset, joita käyttäjä tarvitsee ratkaistakseen ongelman tai saavuttaakseen tavoitteen
>
> 2. edellytykset, joihin järjestelmän tai sen komponentin tulee kyetä vastaamaan täyttääkseen sopimuksen, standardin, määrittelyn tai muun virallisesti säädetyn dokumentin
>
> 3. kohdan 1 tai 2 mukaiset edellytykset, jotka on kuvattu dokumentoituna esityksenä.

Vaatimusmäärittelyn tarkoituksena on saada selville nimenomaan laadukkaita vaatimuksia. Mannion ja Keepence (1995) soveltavat niin sanottua SMART-muistisääntöä myös ohjelmiston vaatimuksiin sopiviksi. Heidän mukaansa vaatimusten tulee olla täsmällisesti kuvattuja (*specific*), mitattavia (*measurable*), saavutettavissa olevia (*attainable*), toteutettavissa olevia (*realisable*) ja jäljitettäviä (*traceable*). Mannion ja Keepence (1995) myöntävät kuitenkin, että mainitut kri-

teerit eivät takaa sitä, että vaatimukset on määritelty todellisia tarpeita vastaaviksi. Vaikka SMART-vaatimukset eivät takaakaan vaatimusten sisällöllistä oikeellisuutta, niiden avulla voidaan tarkistaa, että vaatimukset on ilmaistu laadukkaalla tavalla.

Vaatimuksia voidaan lajitella erilaisiin ryhmiin. Wiegers (2003, 8–9) jaottelee vaatimukset kolmeen erilliseen tasoon: liiketoimintavaatimuksiin, käyttäjävaatimuksiin ja toiminnallisiin vaatimuksiin. Lisäksi jokaisella järjestelmällä on myös ei-toiminnallisia vaatimuksia. Liiketoimintavaatimukset ovat järjestelmän hankkivan yrityksen korkean tason tavoitteita. Liiketoimintavaatimukset kertovat, miksi yritys on ottamassa käyttöön ohjelmistoa, eli ne kertovat tavoitteet, jotka yritys toivoo saavuttavansa. (Wiegers 2003, 8–9)

Käyttäjävaatimukset kuvailevat käyttäjien tavoitteita tai tehtäviä, joista käyttäjien tulee kyetä suoriutumaan hankittavan ohjelmiston avulla. Käyttäjävaatimuksia voidaan esittää esimerkiksi käyttötapausten ja skenaariokuvausten avulla. Käyttäjävaatimukset kuvaavat, mitä käyttäjä pystyy tekemään ohjelmistolla. (Wiegers 2003, 9)

Toiminnalliset vaatimukset täsmentävät ohjelmiston toiminnallisuudet, jotka kehittäjien täytyy rakentaa ohjelmistoon, jotta käyttäjät voivat suorittaa tehtävänsä näin ollen täyttäen liiketoimintatavoitteet. Toiminnallisia vaatimuksia kutsutaan joskus myös käyttäytymisvaatimuksiksi. Toiminnallisten vaatimusten tehtävänä on kertoa, mitä ohjelmiston tulee tehdä. (Wiegers 2003, 10)

Toiminnalliset vaatimukset dokumentoidaan ohjelmiston vaatimusmäärittelydokumenttiin. Dokumentti määrittelee ohjelmiston vaaditun toiminnan niin täydellisesti kuin on tarpeen. Toiminnallisten vaatimusten lisäksi dokumentti sisältää myös ei-toiminnallisia vaatimuksia. Ei-toiminnallisia vaatimuksia ovat esimerkiksi suorituskykyyn ja käytettävyyteen liittyvät vaatimukset. (Wiegers 2003, 10)

2.2 Vaatimusmäärittelyn haasteet

Vaatimusmäärittely on prosessi, jossa tunnistetaan, arvioidaan ja määritellään ohjelmiston vaatimuksia ja hallitaan niihin tulevia muutoksia. Menestyksekäs vaatimusmäärittely edellyttää käyttäjien, asiakkaan ja muiden sidosryhmien tarpeiden ymmärtämistä (Cheng & Atlee 2007). Se onkin yksi keskeisimmistä syistä siihen, että vaatimusmäärittely on haastavaa.

Vaatimusmäärittelyllä voidaan edistää ohjelmiston tarkoituksenmukaisuutta tunnistamalla sidosryhmät ja niiden tarpeet sekä dokumentoimalla ne (Nuseibeh & Easterbrook 2000). Vaatimusmäärittely on haastavaa eikä sen onnistuminen ole itsestään selvää. Chengin ja Atleen (2007) mukaan vaatimusmäärittely on vaikeaa, koska vaatimusmäärittelyn alussa vaatimukset ovat usein huonosti määriteltyjä. Hooks (2000) väittää, että viime aikoina vaatimusten määrittely on mennyt jopa huonompaan suuntaan. Hän perustelee väitettään esimerkiksi sillä, että ennen suunnitteluun ja valmistautumiseen käytettiin enemmän aikaa ja ylin johto tutustui vaatimuksiin ja kantoi vastuun hyväksymistään vaatimuksista. Hooks (2000) kuitenkin myöntää, että ennen myös esimerkiksi käyttöliittymät olivat yksinkertaisempia ja vaatimusten määrittelykin sitä kautta helpompaa.

Haasteita aiheuttaa myös se, että eri sidosryhmien edustajia (esimerkiksi maksajat, käyttäjät, kehittäjät) voi olla paljon ja ne voivat olla hajallaan. Sidosryhmien edustajien tavoitteet ja vaatimukset voivat poiketa toisistaan huomattavastikin. On myös mahdollista, että sidosryhmien tavoitteet voivat olla paitsi keskenään erilaisia ja ristiriitaisia, myös monitulkintaisia. Tavoitteiden saavuttamiseen voi vieläpä vaikuttaa useita tekijöitä, jotka eivät ole sidosryhmien hallinnassa. (Nuseibeh & Easterbrook 2000)

Järjestelmän ympäristö voi olla yhdistelmä laitteistoa, ihmisten käyttäytymistä, todellisen maailman ilmiöitä ja muita ohjelmistokomponentteja. On myös otettava huomioon, mitä ympäristön erikoistilanteet, kuten onnettomuudet, voivat

aiheuttaa järjestelmässä. Tämä puolestaan lisää monimutkaisuutta ja samalla vaikeuttaa vaatimusmäärittelyä. (Cheng & Atlee 2007)

Vaatimusmäärittelyn haastavuutta vielä lisää se, että kaikilla sidosryhmien edustajilla ei ole teknistä taustaa ja ajattelutapaa. Tämä voi aiheuttaa tulkintaongelmia asiakkaan ja ohjelmiston kehittäjän välillä. Vaatimukset tulisi kuitenkin muokata yksiselitteisiksi, yksityiskohtaisiksi sekä tarpeeksi tarkoiksi ja teknisiksi, jotta kehittäjät voivat suunnitella ja toteuttaa ohjelmiston niiden pohjalta (Cheng & Atlee 2007).

Saiedian ja Dale (2000) huomauttavat, että kehittäjät ovat asiantuntijoita ohjelmiston teknisessä suunnittelussa ja toteutuksessa, kun taas käyttäjät ovat puolestaan asiantuntijoita ohjelmiston kohdealueen tuntemisessa. Kehittäjillä ja asiakkailla on siis usein hyvin erilaiset osaamisalueet, joten yhteisen ja molempien osapuolien mielestä ymmärrettävän sekä tarpeeksi yksiselitteisen kommunikointitavan kehittäminen voi olla haastavaa.

2.3 Vaatimusmäärittelyprosessin osat

Cheng ja Atlee (2007) jaottelevat vaatimusmäärittelyprosessin viiteen vaiheeseen. Taulukkoon 1 on koottu lyhyesti vaatimusmäärittelyprosessin vaiheiden keskeisimpiä tehtäviä ja tavoitteita. Vaiheet pohjautuvat Chengin ja Atleen (2007) jaotteluun, jonka he ovat tehneet aiheeseen liittyvän kirjallisuuden pohjalta. Taulukkoon 1 on otettu mukaan myös muissa lähteissä esitettyjä huomioita vaatimusmäärittelyn eri vaiheiden tehtävistä ja tavoitteista. Taulukkoon ei ole kuitenkaan merkitty lähdeviitteitä, vaan lähteet käyvät ilmi, kun vaiheita käydään yksityiskohtaisemmin läpi taulukon jälkeen. Taulukossa on korostettu harmaalla taustavärillä vaatimustenhallinta, jota käsitellään tarkemmin seuraavassa luvussa.

TAULUKKO 1. Vaatimusmäärittelyprosessin vaiheet

Vaiheen nimi suomeksi	**Vaiheen nimi englanniksi**	**Vaiheen tärkeimmät tehtävät ja tavoitteet**
Vaatimusten tunnistaminen	Elicitation	– Ymmärtää hankittavan ohjelmiston keskeisimmät päämäärät ja tavoitteet (myöhemmissä vaiheissa tarkennetaan ensimmäisen vaiheen tuloksia) – Määritellä järjestelmän rajat (ainakin karkealla tasolla) – Tunnistaa sidosryhmät – Auttaa käyttäjiä ymmärtämään paremmin omat tarpeensa => auttaa heitä arvioimaan eri vaihtoehtoja ja ymmärtämään päätösten seurauksia – Auttaa kehittäjiä ymmärtämään ydintarkoitus, johon ohjelmiston odotetaan vastaavan
Mallintaminen	Modeling	– Ilmaista tavoitteet mallien avulla – Tarkentaa ensimmäisessä vaiheessa saatuja malleja ja saada selville yksityiskohtaisempaa tietoa – Suorittaa esimerkiksi yrityksen, tiedon, käyttäytymisen, kohdealueen ja ei-toiminnallisten vaatimusten mallintaminen
Vaatimusanalyysi	Requirements analysis	– Arvioida vaatimusten laatua (analyysissa voi paljastua vaatimuksissa esim. ristiriitaisuuksia ja monitulkintaisuuksia) – Ymmärtää vaatimusten suhteita ja vaikutuksia tekniikoiden avulla – Tarkastella vaatimuksia objektiivisesti
Vahvistaminen ja varmentaminen	Validation and verification	– Varmistaa, että mallit todella kuvaavat sidosryhmien tarpeita – Tarkastella vaatimuksia subjektiivisemmin kuin vaatimusanalyysissä
Vaatimustenhallinta	Requirements management	– Hallita vaatimuksia ajan kuluessa (vaatimuksia lisätään, poistetaan ja virheitä korjataan) – Sisältää esimerkiksi muutostenhallintaa, versionhallintaa, vaatimusten dokumentointia ja tilojen seurantaa sekä vaatimusten jäljittämistä

Vaatimusten tunnistamien tarkoittaa toimenpiteitä, joiden tarkoituksena on ymmärtää hankittavan ohjelmiston päämäärä ja tavoitteet (Cheng & Atlee 2007). Nuseibehin ja Easterbrookin (2000) mukaan se on usein ensimmäinen vaihe ja he korostavatkin, että vaiheen aikana ei ole välttämättä mahdollista saada koottua valmiita vaatimuksia, vaan tarvitaan myös jatkovaiheita tarkentamaan ja jalostamaan ensimmäisessä vaiheessa saatuja tuloksia. Heidän mukaansa ensimmäisessä vaiheessa on syytä määritellä ainakin karkealla tasolla järjestelmän rajat, tunnistaa sidosryhmät ja päättää tavoitteet.

Saiedian ja Dale (2000) huomauttavat, että vaiheen aikana myös käyttäjät alkavat ymmärtää paremmin omia tarpeitaan ja se puolestaan auttaa heitä arvioimaan tehokkaammin eri vaihtoehtoja ja ymmärtämään päätöstensä seurauksia. Toisaalta ohjelmiston kehittäjille muodostuu vaiheen aikana käsitys siitä, mikä on ydinongelma tai -asia, joka ohjelmiston avulla pyritään ratkaisemaan (Saiedian & Dale 2000).

Mallintamisessa tavoitteet ilmaistaan mallien avulla. Tämän vaiheen aikana pyritään saamaan aikaan tarkempia ja yksiselitteisempiä malleja kuin mitä ensimmäisessä vaiheessa on ollut mahdollista tehdä. Mallintaminen auttaa saamaan selville yksityiskohtia, jotka saattoivat puuttua vielä kokonaan ensimmäisessä vaiheessa. (Cheng & Atlee 2007)

Mallintamisen voi jakaa useampaan osaan riippuen siitä, mihin mallintamisessa keskitytään. Yritysmallintamisessa pyritään ymmärtämään muun muassa yrityksen rakennetta, liiketoiminnan lainalaisuuksia, tavoitteita ja sitä, millaista tietoa yrityksessä tarvitaan ja käsitellään. Yritysmallintamisesta pyritään saamaan selville hankittavan järjestelmän tarkoitus. Tietomallinnuksessa pyritään ymmärtämään, mitä tietoa järjestelmään tallennetaan tai järjestelmässä muokataan ja hallitaan. Tähän mallintamiseen voidaan käyttää esimerkiksi kohde-suhde-mallintamista (ER-mallit) tai oliopohjaisia tekniikoita. Käyttäytymisen mallintamisessa puolestaan on tarkoitus ensin mallintaa, miten työ tehdään täl-

lä hetkellä. Sen jälkeen analysoidaan nykytilannetta, jotta saadaan selville keskeisimmät toiminnot ja lopuksi rakennetaan malli siitä, miten uuden järjestelmän tulisi toimia. Kohdealueen mallintamisessa kuvataan alue, johon järjestelmä tehdään. (Nuseibeh & Easterbrook 2000)

Vaatimusanalyysissa keskitytään arvioimaan kirjattujen vaatimusten laatua. Vaatimusanalyysi paljastaa mahdollisia väärinymmärryksiä ja kohtia, joita ei ole aiemmissa vaiheissa määritelty tarpeeksi tarkalla tasolla. Vaatimusanalyysissä voi ilmetä esimerkiksi, että vaatimus on monitulkintainen tai ristiriitainen. Myös vaatimusten välillä voi paljastua aiemmin tuntemattomia vuorovaikutussuhteita. (Cheng & Atlee 2007)

Vaatimusanalyysiin kuuluu esimerkiksi sellaisia tekniikoita kuin riski- ja vaikutusanalyysi (vaikutusanalyysista kerrotaan tarkemmin muutostenhallinnan yhteydessä kohdan 3.2 lopussa). Tekniikoiden avulla pyritään ymmärtämään vaatimuksia sekä niiden suhteita ja vaikutuksia paremmin. Lisäksi priorisointi-, visualisointi- ja analyysitekniikat auttavat valitsemaan mahdollisimman hyvän joukon vaatimuksia toteutettavaksi. (Cheng & Atlee 2007)

Vahvistamisessa ja varmentamisessa pyritään varmistamaan, että mallit ja dokumentit todella kuvaavat sidosryhmien tarpeita. Kun vaatimusanalyysissa tarkastellaan vaatimuksia objektiivisesti, niin vaatimusten varmentamisessa tarkastellaan asiaa subjektiivisemmin. Sen vuoksi sidosryhmien edustajat ovatkin keskeisessä asemassa vaatimusten vahvistamisessa ja varmentamisessa. (Cheng & Atlee 2007)

Hofmannin ja Lehnerin (2001) mukaan vaatimusten vahvistamiseen ja varmentamiseen on olemassa melko niukasti menetelmiä. He mainitsevat kuitenkin vahvistamiseen ja varmentamiseen käytetyiksi menetelmiksi esimerkiksi vertaisarvioinnit, katselmoinnit ja skenaariot, mutta painottavat myös päätösten ja niiden perustelujen kirjaamisen tärkeyttä.

Vaatimustenhallinta kattaa useita tehtäviä, jotka sisältävät vaatimusten muutosten hallintaa ajan kuluessa ja tuoteperheiden välillä (Cheng & Atlee 2007). Tyypillisesti muutokset vaatimuksissa ovat uusien vaatimusten lisäämistä, vanhojen poistamista sekä virheiden korjaamista (Nuseibeh & Easterbrook 2000). Wiegers (2003, 314) jakaa vaatimustenhallinnan muutostenhallintaan, versionhallintaan, vaatimusten tilojen seurantaan ja vaatimusten jäljittämiseen. Vaatimustenhallinnasta ja sen osista kerrotaan tarkemmin seuraavassa luvussa.

2.4 Yhteenveto

Jotta ohjelmisto voi olla hyödyllinen, sen tulee täyttää sille asetetut vaatimukset. Vaatimusten määritteleminen on tärkeää, jotta ohjelmiston kehittäjät tietävät, mitä tehdä ja asiakkaat tietävät, mitä odottaa. Vaatimukset voivat ohjelmistoprojektin alussa olla epätarkasti ilmaistuja toiveita ohjelmiston tavoitteista, mutta vaatimusmäärittelyn aikana vaatimuksia tarkennetaan ja niistä pyritään tekemään selkeitä, yksiselitteisiä ja ristiriidattomia. Vaatimuksia voidaan luokitella. On olemassa esimerkiksi liiketoimintavaatimuksia, käyttäjävaatimuksia, toiminnallisia vaatimuksia ja ei-toiminnallisia vaatimuksia.

Vaatimusmäärittely on haastavaa, koska usein vaatimuksia esittäviä sidosryhmiä on useita ja sidosryhmien vaatimukset voivat olla keskenään erilaisia ja ristiriitaisia. Vaatimuksista täytyykin neuvotella, ja niitä täytyy laittaa tärkeysjärjestykseen, jotta niistä saadaan hyvin määritelty ja toteutettavissa oleva kokonaisuus. Vaatimusmäärittelyyn tuo haasteita myös se, että vaatimusmäärittelyyn osallistuvien sidosryhmien edustajien taustat voivat olla hyvin erilaisia. Asiakkaan edustajilla on usein kohdealueen tietämystä ja kehittäjillä ohjelmistokehityksen osaamista. Toisaalta taas kehittäjiltä voi puuttua kohdealueen tuntemusta ja asiakkaalta tietoteknistä osaamista. Tällöin kommunikointiin tulee kiinnittää erityistä huomiota.

Chengin ja Atleen (2007) mukaan vaatimusmäärittely jaetaan vaatimusten tunnistamiseen, mallintamiseen, vaatimusanalyysiin, vaatimusten varmentamiseen

sekä vaatimustenhallintaan. Vaatimusmäärittelyn alussa painotetaan sitä, että asiakas hahmottaa omat tarpeensa ja toisaalta ohjelmiston kehittäjät ymmärtävät tarkoituksen, johon ohjelmistoa ollaan kehittämässä. Kun ohjelmiston keskeisimmät tavoitteet on saatu selville, niitä voidaan tarkentaa erilaisilla mallintamistavoilla. Vaatimuksia analysoitaessa pyritään poistamaan vaatimusten välisiä ristiriitaisuuksia sekä monitulkintaisuuksia ja näin ollen parantamaan vaatimusten laatua. Myös vaatimusten vaikutusten arviointi on keskeisessä asemassa. Vaatimuksia varmennettaessa tutkitaan, että aiemmissa vaatimusmäärittelyn vaiheissa tunnistetut vaatimukset ja mallit todella vastaavat sidosryhmien tarpeita. Vaatimustenhallinnassa puolestaan otetaan huomioon ajan myötä muuttuvien vaatimusten vaikutukset.

Suurin osa vaatimusmäärittelyprosessin vaiheista sijoittuu ohjelmistokehitysprojektin alkuvaiheeseen, mutta jatkossa tässä tutkielmassa kiinnitetään huomiota siihen, mitä vaatimuksille tapahtuu ja miten niitä hallitaan ajan kuluessa ohjelmiston siirryttyä elinkaarellaan kehityksestä käyttöön. Silloin varsinainen vaatimusmäärittely on jo tehty, mutta ympäristö ja tarpeet muuttuvat vielä kehitysprojektin päätyttyäkin aiheuttaen sen, että myös vaatimukset muuttuvat ja kehittyvät.

3 VAATIMUSTENHALLINNAN KESKEISET OSA-ALUEET

Tässä luvussa käsitellään vaatimustenhallintaa. Aluksi määritellään vaatimustenhallinnan käsite. Sen jälkeen esitellään vaatimustenhallinnan osa-alueita: muutostenhallintaa, versionhallintaa, vaatimusten tilojen seurantaa ja vaatimusten jäljittämistä. Kutakin osa-aluetta käsitellään omassa kohdassaan. Tässä luvussa kerrotaan lyhyesti myös vaatimustenhallinnan työkaluista ja luvun lopussa kootaan yhteenvetoon luvun keskeisin sisältö.

3.1 Vaatimustenhallinnan määrittely

Suuri osa vaatimusmäärittelyprosessista sijoittuu ohjelmistoprojektin alkuvaiheeseen (Nuseibeh & Easterbrook 2000). Vaatimukset kuitenkin muuttuvat myös ohjelmiston käyttöönoton jälkeen. Tällöin tarvitaan vaatimustenhallintaa. Taloudellisesti on edullisinta pyrkiä siihen, että mahdolliset väärinymmärretyt ja virheellisesti määritellyt vaatimukset korjataan mahdollisimman aikaisessa vaiheessa elinkaarta. Vaatimusten korjaaminen on sitä kalliimpaa mitä myöhemmin virheet havaitaan ja korjataan (esim. Nakajo & Kume 1991). Aina vaatimusten korjaaminen ei ole kuitenkaan mahdollista varhaisessa vaiheessa, sillä vaatimukset voivat muuttua ajan kuluessa esimerkiksi ohjelmistoa käyttävän yrityksen liiketoiminnan muuttuessa.

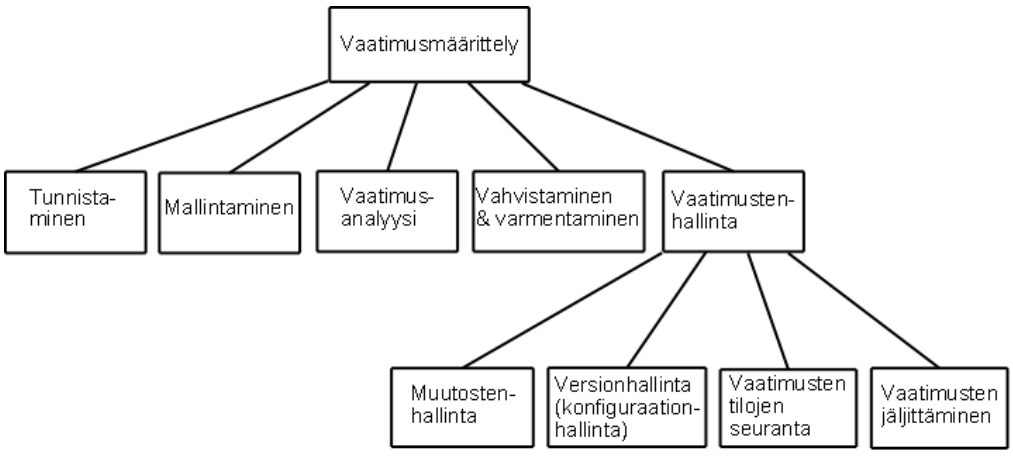
Paetsch, Eberlein ja Maurer (2003) määrittelevät vaatimustenhallinnan tavoitteeksi informaation tallentamisen, varastoimisen, välittämisen ja hallitsemisen. Chengin ja Atleen (2007) mukaan vaatimustenhallinta on eräänlainen kattotoiminto, joka sisältää monenlaisia vaatimusten hallintaan liittyviä tehtäviä mukaan lukien myös vaatimusten muuttumisen ja kehittymisen ajan ja eri tuoteperheiden myötä. Asiakkuuden aikana asiakkaan tarpeet ja sen myötä vaatimukset saattavat muuttua niin, että alkuperäisten vaatimusten mukainen tuote ei enää täytäkään asiakkaan tarpeita. Tällöin vaatimustenhallinnan avulla voi-

daan päätyä asiakkaalle tarjoamaan esimerkiksi saman tuoteperheen jotain toista tuotetta.

Tyypillisesti muutokset vaatimuksissa ovat uusien vaatimusten lisäämistä, vanhojen poistamista sekä virheiden korjaamista. Uusia vaatimuksia lisätään luonnollisesti siitä syystä, että ne ovat jääneet jostain syystä pois alun perin, mutta uusia vaatimuksia lisätään myös silloin, kun sidosryhmien tarpeet muuttuvat. Vaatimuksia poistetaan yleensä vain kehitystyön aikana. Syynä vaatimusten poistamiseen on usein pyrkimys kustannus- ja aikataulusäästöihin. (Nuseibeh & Easterbrook 2000)

Muuttuvien vaatimusten hallinta ei ole vain dokumenttien hallintaa, vaan sen vaikutus on paljon laajempi. Tarvitaan vaatimusten tunnistamista, riskien uudelleenarvioimista ja järjestelmän arvioimista operatiivisessa ympäristössään. Jokainen aiottu muutos tulee arvioida olemassa olevien vaatimusten ja arkkitehtuurin ehdoilla, jotta sen kustannus-hyötysuhde saadaan selville. (Nuseibeh & Easterbrook 2000)

Kuviossa 1 on esitetty luvun 2 pohjalta vaatimusmäärittelyn osat Chengin ja Atleen (2007) mukaan. Kuvioon 1 on merkitty myös Wiegersin (2003, 314) mukaan vaatimustenhallinnan osa-alueet, joita käsitellään tarkemmin tässä luvussa. Kuvio havainnollistaa vaatimustenhallinnan suhdetta vaatimusmäärittelyyn.



KUVIO 1. Vaatimusmäärittelyn ja vaatimustenhallinnan osat

Wiegers (2003, 314) jakaa vaatimustenhallinnan neljään osa-alueeseen, jotka ovat muutostenhallinta, versionhallinta, vaatimusten tilojen seuranta ja vaatimusten jäljittäminen. Seuraavassa neljässä kohdassa käsitellään kutakin vaatimustenhallinnan osa-aluetta erikseen. Osa-alueiden sisältöä on kuitenkin laajennettu muiden lähteiden pohjalta, esimerkiksi versionhallinnan yhteydessä tarkastellaan myös laajemmin konfiguraationhallintaa.

3.2 Muutostenhallinta

Vaatimustenhallintaan kuuluu luonnollisena osana muutostenhallinta. Niin kuin Wiegerskin (2003, 328) toteaa, muutos ei ole itsessään huono asia, vaan luonnollinen osa ohjelmiston kehitystä. Kaikkia vaatimuksia ei ole edes mahdollista määritellä etukäteen ja sen vuoksi ohjelmiston elinkaaren aikana kehittyy uusia vaatimuksia. Muutoksia ei voi välttää, joten niitä pitää pystyä hallitsemaan.

Esimerkiksi Lehmanin laeissa otetaan kantaa muutosten väistämättömyyteen. Lehmanin ensimmäisen lain mukaan ohjelmistoa tulee jatkuvasti mukauttaa muutoksiin tai siitä tulee ajan mittaan käyttökelvoton. Ensimmäistä lakia kutsutaankin jatkuvan muutoksen laiksi. Myös Lehmanin toinen laki, lisääntyvän monimutkaisuuden laki, liittyy muutoksiin ja niiden hallintaan. Toisen lain mukaan ohjelmiston monimutkaisuus kasvaa, kun ohjelmisto ajan mittaan kehittyy, ellei monimutkaisuuden lisääntymisen estämiseksi tehdä työtä. Myös Lehmanin kuudes laki, jatkuvan kasvun laki, viittaa muutosten väistämättömyyteen. Kuudennen lain mukaan ohjelmiston toiminnallisuuksien tulee lisääntyä jatkuvasti, jotta käyttäjät pysyvät tyytyväisinä. (Lehman 1996)

Muutostenhallinta sisältää muutosten ehdottamisen jälkeen analysoinnin muutoksen aiheuttamista vaikutuksista. Analyysin perusteella tehdään päätös, toteutetaanko vaatimusta. Jos muutos toteutetaan, myös suunnitelmiin ja vaatimuksia käsitteleviin dokumentteihin on tehtävä tarvittavat muutokset. Myös vaatimusten mahdollista epävakautta tulee arvioida. (Wiegers 2003, 314)

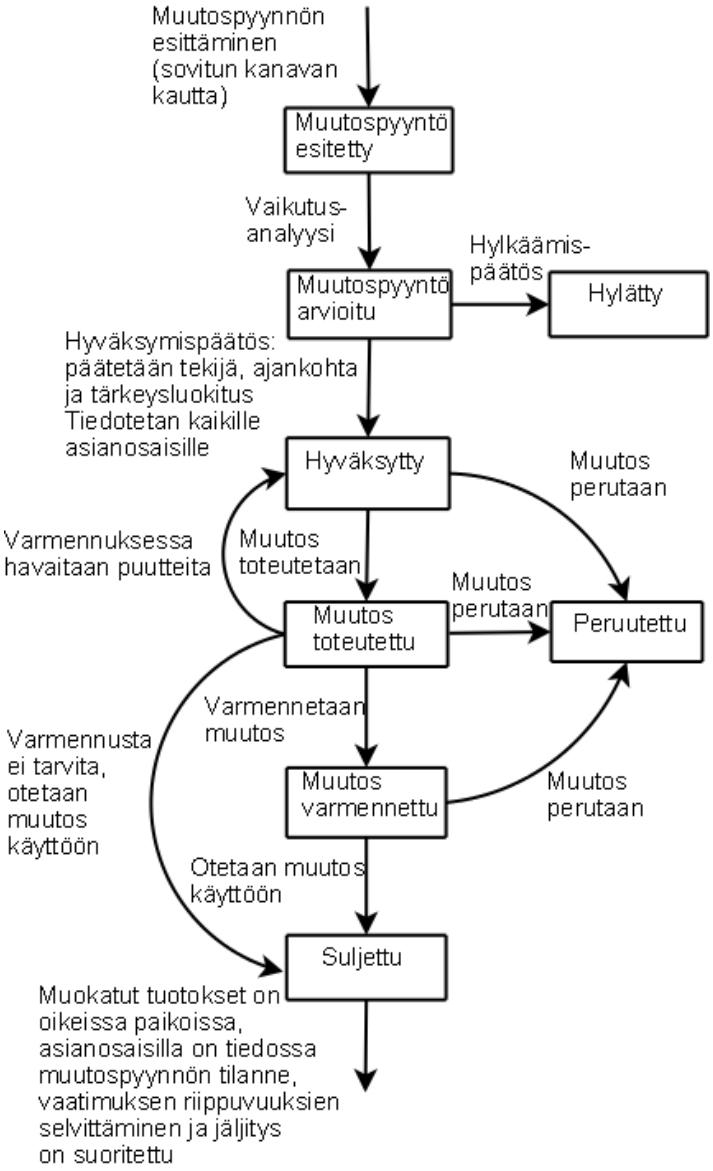
Wiegers (2003, 328) on listannut muutostenhallinnan menestystekijöitä. Hänen mukaansa ehdotetut muutospyynnöt tulee arvioida huolellisesti ennen toteuttamista. Kun tulee tarve muuttaa vaatimuksia, on syytä aloittaa korkeimmalta tasolta, jota muutos koskee. Tarvittaessa on siis lähdettävä pohtimaan asiaa liiketoimintapäätöksistä asti. Hyväksytyistä muutoksista tulee tiedottaa kaikille, joihin muutos vaikuttaa. Lisäksi vaatimusmuutoksia tulee hallita yhdenmukaisella tavalla. (Wiegers 2003, 328)

Muutosehdotuksia tai -pyyntöjä esitetään usein ilman tarkempaa pohdintaa ja muutoksen vaikutusten analysointia. Muutospyyntö esitetään myös monissa tapauksissa kohdealueen termejä käyttäen eikä se ole siis välttämättä kovin teknisessä muodossa (Bennett & Rajlich 2000a). Bennett ja Rajlich (2000a) toteavatkin, että kun muutospyyntö esitetään, suuri osa muutospyyntöön käytettävästä ajasta kuluu siihen, että pyritään ymmärtämään ohjelmiston toimintaa muutospyynnön näkökulmasta. Siis jos muutospyyntö on esitetty ohjelmiston kohdealueen termejä käyttäen, voi olla hidasta selvittää, mitä kohtaa ohjelmakoodista muutos koskee.

Jotta muutospyyntöjen merkitys ja vaikutukset tulisi arvioitua yhdenmukaisella tavalla, suosittelee esimerkiksi Wiegers (2003, 331) noudattamaan muutospyynnöissä yhtenäistä ja selkeää prosessia. Hänen mukaansa kaikkien muutospyyntöjen tulisi noudattaa määriteltyä prosessia ja muutoin kuin prosessin mukaisesti tulleita muutospyyntöjä ei tulisi huomioida. Tämä kannustaa myös asiakasta ja muita sidosryhmiä noudattamaan prosessia, sillä jos he eivät noudata sovittuja menettelytapoja, ei muutospyyntöä oteta käsittelyyn. Toki on otettava huomioon, että muutospyyntöjen laajuuksissa on eroja. Siksi muutosprosessissa on hyvä olla myös hyvin pienille ja riskittömille muutospyynnöille kevyemmät menettelytavat kuin laajavaikutuksisemmille muutoksille. (Wiegers 2003, 331)

Wiegers (2003, 333–338) esittelee prosessimallin, jonka mukaan muutosprosessi voi mennä. Jotta prosessin alkuehdot täyttyvät, tulee muutospyynnön tulla sovitulla tavalla ja sovitun kanavan, esimerkiksi lomakkeen tai sähköpostin kautta. Kun muutospyyntö on täyttänyt alkuehdot, arvioidaan seuraavaksi pyynnön hyödyllisyys, kustannukset, riskit ja vaikutukset sekä tutkitaan, onko pyyntö hyväksyttävissä liiketoimintavaatimusten ja resurssirajoitusten puitteissa. Kun arviointi on tehty, päätöksentekoon oikeutetut henkilöt päättävät, hyväksytäänkö vai hylätäänkö muutospyyntö. Jos pyyntö hyväksytään, sille määritellään tekijä, tärkeysluokitus ja toteutusajankohta. Vaatimuksen tila päivitetään ajantasaiseksi ja samalla uudesta vaatimuksesta tiedotetaan kaikille, joiden työhön se vaikuttaa. Muutos voi vaikuttaa esimerkiksi vaatimusmäärittelydokumenttiin, käyttöliittymäkomponentteihin, ohjelmakoodiin, suunnittelu- ja testausdokumentteihin ja käyttöoppaisiin. Kun muutos on tehty, se tulee varmentaa. Varmentaminen voidaan tehdä esimerkiksi vertaisarvioinnin avulla ja siinä varmistetaan, että muutetut määritykset, käyttötapaukset ja mallit vastaavat muutoksen kaikkia osia. Vaatimusten jäljittämisen (ks. kohta 3.5) avulla voidaan lisäksi selvittää, mihin kaikkeen tehty muutos vaikuttaa. Muutosprosessin päätepiste saavutetaan, kun tietyt ehdot ovat täyttyneet. Ensinnäkin vaatimuksen tila tulee olla joko hylätty, suljettu tai peruutettu. Lisäksi kaikkien muokattujen tuotosten tulee olla laitettuina oikeisiin paikkoihin. Muutokseen tai muutospyyntöön osallistuneilla tulee olla tiedossa muutoksen yksityiskohdat ja muutospyynnön tilanne. Vaatimuksen riippuvuuksien selvittäminen ja jäljitys tulee myös olla suoritettuna. Wiegers (2003, 333–338)

Kuviossa 2 on havainnollistettu edellä esiteltyä muutosprosessia. Kuvio perustuu Wiegersin (2003, 335) esittämään eräänlaiseen tilakaavioon, mutta kuvioon on lisätty tekstikohtia, jotka selittävät tarkemmin, mitä keskeisiä asioita eri vaiheiden aikana käydään läpi.



KUVIO 2. Muutosprosessi vaatimustenhallinnassa (muokattu Wiegersin 2003, 335 pohjalta)

Vaikutusanalyysi on keskeinen osa muutostenhallintaa ja muutosprosessia. Queille, Voidrot, Wilde ja Munro (1994) ovat määritelleet, että vaikutusanalyysin tehtävänä on arvioida vaikutukset sille, että ohjelmistoon tehdään joukko muutoksia. Vaikutusanalyysin tavoitteena on minimoida muutoksen odottamattomat sivuvaikutukset (Queille ym. 1994).

Kuten muutosprosessin kuvauksessa kävi ilmi, niin vaikutusanalyysi voidaan tehdä muutospyynnön toteuttamista arvioitaessa. Vaikutusanalyysi auttaa

ymmärtämään mahdollisen muutoksen seurauksia ja se puolestaan auttaa tekemään perusteltuja päätöksiä siitä, hyväksytäänkö muutospyyntö vai ei. Analyysissa tutkitaan ehdotettua muutosta tunnistamalla komponentit, joita pitäisi tehdä, muokata tai poistaa, jos muutos toteutetaan. Samalla arvioidaan myös muutostöiden työmäärää. (Wiegers 2003, 345)

Pieneltä tuntuvallakin muutoksella voi olla laajat vaikutukset (esim. Wiegers 2003, 344). Yau, Collofello ja MacGregor ovat jo vuonna 1978 kiinnittäneet huomiota muutosten vaikutusten laajuuden arviointiin ja haastavuuteen, he kutsuvat tutkimuksessaan muutoksen aikaansaamia vaikutuksia heijastusvaikutuksiksi (*ripple effect*).

Vuosituhannen vaihde konkretisoi vaikutusanalyysien merkityksen hyvin laajasti miltei kaikille aloille suunnatuissa ohjelmistoissa. Bohner (1996) mainitsee, että vuosituhannen vaihteen vuosilukuihin liittyvistä muutoksista tehtiin vaikutusanalyyseja, jotka osoittivat, että kohtuullisen yksinkertaiselta vaikuttavan muutoksen vaikutukset olivat hyvin laajat. Myös euron käyttöönotto vaati laajaa vaikutusanalyysia (esim. De Lucia, Pannella, Pompella & Stefanucci 2002).

3.3 Konfiguraationhallinta ja versionhallinta

Versionhallinnan tarkoituksena on estää vanhentuneiden ja vaihtelevien vaatimusten aiheuttamat ongelmat. Jokainen versio vaatimusdokumenteista tulee olla yksilöllisesti tunnistettavissa ja muutokset tulee dokumentoida selkeästi. Yksinkertaisimmillaan se voi tarkoittaa esimerkiksi muutoshistorian ylläpitämistä dokumentin yhteydessä. Muutoshistoriasta tulee käydä ilmi, mikä muutos on tehty, milloin se on tehty, kuka sen on tehnyt ja mistä syystä. (Wiegers 2003, 317–319)

Wiegers (2003, 317–319) keskittyy versionhallintaa kuvaillessaan erityisesti vaatimusmäärittelydokumentteihin eikä korosta varsinaisen lähdekoodin version-

hallintaa. Kuitenkin yleisesti (mm. Estublier 2000) versionhallinta käsitetään laajemmin sisältäen esimerkiksi lähdekoodin.

Tässä kohdassa esitellään lyhyesti myös konfiguraationhallintaa, koska konfiguraationhallinnan kuvaileminen voi auttaa laajentamaan versionhallintaan liittyviä asioita. Estublier (2000) määrittelee konfiguraationhallinnan alaksi, joka mahdollistaa sen, että jatkuvasti kehittyvät ohjelmistot pysyvät hallinnassa edistäen samalla laatua. Hän mainitsee myös yleisemmin, että konfiguraationhallinta on tapa kontrolloida monimutkaisten järjestelmien kehittymistä. Määritelmä on hyvin laaja ja sen mukaan miltei kaikki elinkaaren ja vaatimusten hallintaan liittyvät asiat voisi käsittää kuuluvaksi konfiguraationhallintaan.

Konfiguraationhallinta on ohjelmiston elinkaaren tukemista. Se edistää projektinhallintaa, laadunvarmistusta, ohjelmiston kehittämistä ja ylläpitoa sekä hyödyttää asiakkaita ja ohjelmiston käyttäjiä. Konfiguraationhallinnassa kontrolloidaan ohjelmiston eheyttä ja kehittymistä tunnistamalla sen osat, hallitsemalla muutoksia sekä kirjaamalla, varmentamalla ja raportoimalla konfiguraatioon liittyvää tietoa. Ohjelmistokehittäjän näkökulmasta konfiguraationhallinta helpottaa kehittämiseen ja muutostenhallintaan liittyviä toimintoja. (SWEBOK 2004, 106)

Dart (1991) on määritellyt IEEE:n (1987) standardiin pohjautuen konfiguraationhallinnan keskeisiä toimintoja. Yksi toiminnoista on tunnistaminen, jossa määritellään ohjelmistotuotteen rakenne sekä yksilöidään siihen kuuluvat komponentit ja niiden tyypit (Dart 1991). Toisin sanoen tunnistetaan kontrolloitavat kohteet (SWEBOK 204, 108).

Konfiguraationhallintaan liittyvä kontrollointitoiminto tarkoittaa ohjelmistoversioiden ja -julkaisuiden sekä niihin tehtyjen muutosten hallintaa ohjelmiston elinkaaren aikana (Dart 1991). Konfiguraation kontrolloinnissa keskitytään muutostenhallintaan elinkaaren aikana, siihen liittyy ohjelmiston muutospyyntöjen ehdottaminen, arvioiminen ja hyväksyminen (SWEBOK 2004, 108–112).

Keskeinen toiminto on myös tilatietojen ylläpito ja kirjaaminen. Se on komponenttien ja niihin tulevien muutospyyntöjen tilatietojen tallentamista ja raportointia (Dart 1991). Tilatietojen ylläpidossa siis tallennetaan ja raportoidaan tehokkaaseen ohjelmiston konfiguraatioiden hallintaan tarvittavaa tietoa (SWEBOK 2004, 113).

Auditointi ja arviointi liittyvät myös konfiguraationhallintaan ja tarkoittavat sitä, että varmistetaan ohjelmiston valmius ja komponenttien välinen yhtenäisyys (Dart 1991). Ohjelmiston konfiguraation auditoinnit voidaan jakaa toiminnalliseen ja fyysiseen auditointiin. Ohjelmiston elinkaaren aikana pidetään virallisten lisäksi myös epävirallisia auditointeja tarpeen mukaan. (SWEBOK 2004, 108–114)

Konfiguraationhallintaan kuuluu myös ohjelmiston julkaisujen hallinta ja toimitus. Näihin toimintoihin liittyy ohjelmistopaketin tarpeellisten osien tunnistaminen ja kokoaminen sekä ohjelmiston toimittaminen asiakkaalle. Lisäksi esimerkiksi julkaisun ajankohdan suunnittelu kuuluu julkaisujen hallintaan. Toimitettaviin osiin kuuluu varsinaisen ohjelmiston ohella myös muuta. Esimerkiksi dokumentaatio, tiedot julkaisussa olevista muutoksista sekä konfiguraatiotiedot yleensäkin, ovat osa toimitusta. (SWEBOK 2004, 114–115)

Varsinaisten konfiguraationhallinnan toimintojen lisäksi myös prosessin hallinta on keskeinen osa konfiguraationhallintaa. Prosessin hallinnassa suunnitellaan konfiguraationhallintaa, päätetään muun muassa vastuista, resursseista ja aikatauluista sekä valitaan ja otetaan käyttöön tarvittavat työkalut. Konfiguraationhallintaprosessista tehdään kirjallinen suunnitelma, joka käydään läpi ja tarkastetaan yleensä palvelun laatuun liittyvän sopimuksen yhteydessä. (SWEBOK 2004, 108–109)

Edellä mainituissa konfiguraationhallinnan keskeisissä toiminnoissa on havaittavissa yhteneväisyyksiä vaatimustenhallinnan muihinkin osiin kuin versionhallintaan. Konfiguraationhallinnassa on samoja piirteitä kuin kohdassa 3.1 kä-

sitellyssä muutostenhallinnassa ja esimerkiksi tilatietojen ylläpidossa on havaittavissa samankaltaisuutta vaatimusten tilojen seurannan kanssa. Vaatimusten tilojen seurannasta kerrotaan tarkemmin seuraavassa kohdassa.

3.4 Vaatimusten tilojen seuranta

Vaatimusten tilojen seurantaa pidetään yhtenä vaatimustenhallinnan toiminnoista (esim. Wiegers 2003, 314; Paetsch, Eberlein & Maurer 2003). Vaatimusten tiloja voivat olla esimerkiksi ehdotettu, hyväksytty, toteutettu, vahvistettu ja poistettu. Kun joku valtuutettu henkilö esittää vaatimuspyynnön, tulee vaatimuksen tilaksi ehdotettu. Kun vaatimus on analysoitu, sen vaikutukset on arvioitu ja keskeisimmät sidosryhmät sekä ohjelmiston suunnittelijat ovat hyväksyneet vaatimuksen toteutettavaksi, asetetaan vaatimuksen tilaksi hyväksytty. Tämän jälkeen vaatimus toteutetaan ja vaatimuksen tilaksi päivitetään toteutettu. Kun toteutetun vaatimuksen toiminta on todettu oikeanlaiseksi, päivitetään vaatimuksen tilaksi vahvistettu. (Wiegers 2003, 322–323)

Kaikkia vaatimuksia ei aina toteuteta, tällaisessa tilanteessa vaatimuksen tilaksi laitetaan esimerkiksi poistettu. Tällöin on syytä myös tallentaa tieto siitä, kuka teki päätöksen poistamisesta ja miksi vaatimus päätettiin poistaa. Myös hylätyt vaatimukset ja syy hylkäämiseen kannattaa kirjata, sillä hylätyt vaatimukset tulevat usein myöhemmin uudelleen esille. (Wiegers 2003, 322–323)

Wiegersin (2003, 321) mukaan vaatimuksille on hyödyllisempää määritellä edellä mainittuja tiloja kuin esimerkiksi valmiusprosentteja. Tilojen määritteleminen on valmiusprosentteja hyödyllisempää siksi, että valmiusprosentteja on vaikea arvioida ja prosenteilla arvioituna vaatimuksista annetaan usein liian optimistisia arvioita. Kun vaatimuksille asetetaan tilat ja selkeät edellytykset siihen, mitä kuhunkin tilaan vaaditaan, on mahdollista seurata vaatimusten tilojen jakaumaa ajan kuluessa. (Wiegers 2003, 321–323)

Wiegers (2003, 321–323) huomauttaa myös, että on tärkeää määritellä, kenellä on oikeus muuttaa vaatimuksen tilaa. Lisäksi hänen mukaansa on tärkeää, että vaatimusten tiloja muutetaan vasta, kun kaikki tilamuutoksen edellyttämät asiat on tehty.

3.5 Vaatimusten jäljittäminen

Vaatimusten jäljitettävyys on yksi keskeisimmistä vaatimustenhallinnan tehtävistä. Vaatimusten jäljitettävyyden avulla voidaan perustella vaatimuksia ja saada apua vaatimusmuutosten seurausten ja vaikutusten arviointiin (Nuseibeh & Easterbrook 2000). Vaatimusten jäljittäminen voi parantaa ohjelmiston laatua, koska sen avulla saadaan ihmisistä riippumatonta tietämystä ohjelmistosta (Heindl & Biffl 2005).

Vaatimusten jäljittämisessä määritellään vaatimusten linkit toisiin vaatimuksiin ja muihin järjestelmän osiin. Yksinkertaisilla vaatimusmuutoksilla on usein kauaskantoiset vaikutukset. On vaikeaa saada selville kaikki komponentit, joihin muuttuneen vaatimuksen seuraukset voivat vaikuttaa. Jäljitettävyystiedot tukevat vaikutusanalyysia auttamalla tunnistamaan, mitä kaikkea on muutettava, mikäli vaatimusmuutos toteutetaan. (Wiegers 2003, 353–354)

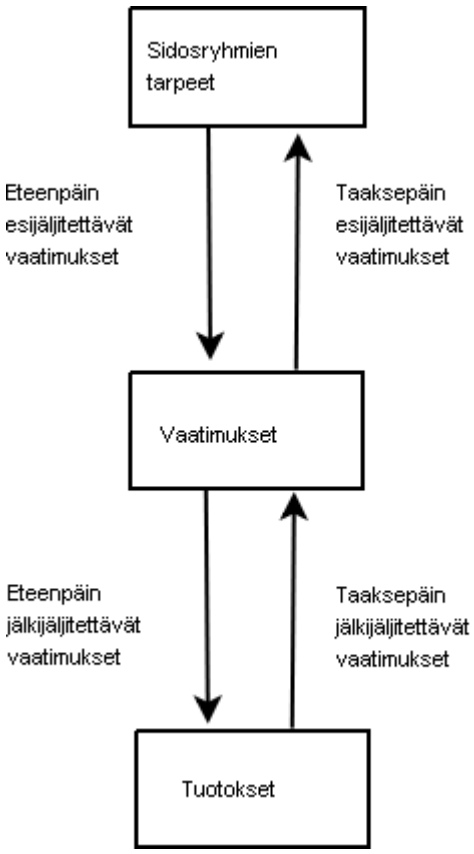
Jäljitettävyysketjujen avulla voi seurata vaatimuksen elinkaarta eteen- ja taaksepäin (Gotel & Finkelstein 1994). Jäljitettävyyden helpottamiseksi jokainen vaatimus tulisi nimetä yksilöllisesti, jotta siihen on mahdollista viitata helposti (Wiegers 2003, 354).

Jarke (1998) on Davisin (1990) pohjalta jaotellut vaatimusten jäljitettävyysketjut neljään erilaiseen tyyppiin. Ensinnäkin on olemassa kahta eri tyyppiä jälkijäljitettävyyttä. Jälkijäljitettävyydessä vaatimukset yhdistetään esimerkiksi suunnitteluun, toteutukseen tai vaatimuksen vaikutusanalyysiin. Eteenpäin suunnatussa jälkijäljitettävyydessä (*forward from requirements*) jäljitetään ketju yksittäisen vaatimuksen ja ohjelmiston komponentin välillä, eli yksittäisen vaatimuksen

tulee yhdistyä johonkin ohjelmiston osaan. Eteenpäin suuntautuvan jälkijäljitettävyyden avulla voidaan varmistaa, että kaikki vaatimukset on täytetty. Taaksepäin suuntautuvan jälkijäljitettävyyden (*backward to requirements*) avulla voidaan selvittää, miksi joku ohjelmiston osa on toteutettu. Taaksepäin suuntautuva jälkijäljitettävyys siis yhdistää ohjelmiston osan tiettyyn vaatimukseen. Se auttaa hahmottamaan, onko ohjelmistossa sellaisia osia tai ominaisuuksia, joita ei vaadita. Wiegers (2003, 355) kuitenkin huomauttaa, että useimmissa sovelluksissa on osia, joita ei voi suoraan yhdistää mihinkään tiettyyn vaatimukseen, mutta lähtökohtana on, että aina tulisi olla tiedossa, miksi joku sovelluksen osa on toteutettu. (Jarke 1998; Wiegers 2003, 354–355)

On olemassa myös eteenpäin suuntautuvaa esijäljitettävyyttä sekä taaksepäin suuntautuvaa esijäljitettävyyttä. Eteenpäin suuntatuvassa esijäljitettävyydessä (*forward to requirements*) voidaan arvioida, mihin vaatimuksiin muutokset sidosryhmien tarpeissa vaikuttavat. Taaksepäin suuntautuvan esijäljityksen (*backward from requirements*) avulla voidaan puolestaan jäljittää kunkin vaatimuksen alkuperä. Jarken (1998) mukaan jälkijäljitettävyys on paremmin ymmärretty kuin esijäljitettävyys, vaikka hänen mukaansa vain esijäljitettävyys havainnollistaa liiketoiminnan ja IT:n yhteyden. (Jarke 1998; Wiegers 2003, 354–355)

Kuviossa 3 on Wiegersin (2003, 354) kuvion pohjalta havainnollistettu vaatimusten jäljittämisen eri tyyppejä. Wiegersin kuviossa ylimpänä on asiakkaan tarpeet, mutta Kuvioon 3 on laitettu ylimmäksi sidosryhmien tarpeet, joihin myös Jarke (1998) viittaa esitellessään jäljitettävyystyyppejä. Ylimpään laatikkoon on laitettu sidosryhmien tarpeet asiakkaan tarpeiden sijasta, sillä se auttaa havainnollistamaan, että vaatimuksiin vaikuttaa useita erilaisia tahoja mahdollisesti ristiriitaisine tarpeineen ja mielipiteineen.



KUVIO 3. Vaatimusten jäljittämisen tyypit (muokattu Wiegersin 2003, 354 pohjalta)

Vaikka muun muassa Jarke (1998) ja Wiegers (2003, 353–366) esittävät vaatimusten jäljittämisen positiivisena ja lupaavana keinona hallita vaatimuksia, huomioivat Heindl ja Biffl (2005) myös sen, että todellisuudessa vaatimusten jäljittäminen ei ole yhtä hallittua kuin malleissa. Heidän mukaansa vaatimusketjujen tunnistaminen ja ylläpitäminen voi olla kallista vaatimusten määrän kasvaessa ja jäljityksen tarkkuuden lisääntyessä. Vaatimusten jäljittäminen ei ole useinkaan selkeä ja systemaattinen prosessi, vaan enemmänkin vaatimuksia jäljitetään tilannekohtaisesti ja ilman tarkempaa suunnittelua. Tällöin myös siitä aiheutuvat kustannukset jäävät usein piileviksi. (Heindl & Biffl 2005)

Heindlin ja Bifflin (2005) mainitseman vaatimusten jäljittämisen kalleuden huomioiden onkin syytä miettiä keinoja, kuinka vaatimusten jäljittämiseen käytetyille rahoille saadaan mahdollisimman hyvin vastinetta. Cleland-Huang, Ze-

mont ja Lukasik (2004) ovat koonneet erilaisia tapoja, jotka parantavat vaatimusten jäljittämiseen käytettyjen investointien tuottoja. He esittelevät mahdollisuuksiksi erilaisia jäljittämisstrategioita. Yhtenä strategiana on dynaamisten linkkien maksimoiminen, mikä tarkoittaa sitä, että minimoidaan vaatimusten väliset linkit, jotka pitää tehdä ja ylläpitää manuaalisesti. Toinen strategia on se, että tehdään ainoastaan tarpeelliset vaatimuslinkit eli arvioidaan kunkin linkin kohdalla, miksi se olisi tarpeellista tehdä. Kolmas strategia on tehokkaimman jäljitysmenetelmän valitseminen. Jokaisen vaatimuksen kohdalla mietitään erikseen, millä tekniikalla se olisi helpoin jäljittää. Neljäs strategia liittyy lyhytaikaisten ja pitkäaikaisten jäljitysketjujen erotteluun. Tämä strategia liittyy siihen, että ei ylläpidetä turhaan jäljitysketjuja, joita ei enää tarvita. Esimerkiksi vain ohjelmiston kehittämisen aikana tarvittavia jäljityslinkkejä ei ole taloudellisesti hyödyllistä pitää ajantasaisina enää ylläpidon aikana. (Cleland-Huang ym. 2004)

3.6 Vaatimustenhallinnan työkaluja

Vaatimustenhallintaan kehitettyjen työkalujen tarkoituksena on olla teknisiä apuvälineitä vaatimustenhallintaan, mutta niiden hyödyllisyydestä on kirjallisuudessa erilaisia mielipiteitä. Esimerkiksi Heindl ja Biffl (2005) kritisoivat, että teknisten apuvälineiden käyttö ei takaa hyvää tulosta. Heidän mukaansa esimerkiksi vaatimusten jäljityksen automatisointiin tarkoitettujen työvälineiden käyttäminen on riskialtista ja monimutkaista. Ruhe, Eberlein ja Pfahl (2002) ovat puolestaan sitä mieltä, että esimerkiksi vaatimustenhallintaan tarkoitettuja työkaluja käytetään yhä yleisemmin ja ne ovat auttaneet parantamaan ohjelmistojen laatua.

Heindl ja Biffl (2005) mainitsevat vaatimusten jäljittämiseen kehitettyjen työkalujen keskeiseksi ongelmaksi sen, että ne eivät erottele vaatimuksia niiden arvon perusteella. Niin kuin vaatimuksiakin on eriarvoisia, myöskään eri vaatimusten jäljittäminen ei ole samanarvoista. Jäljittämisen arvo riippuu muun mu-

assa sidosryhmän tärkeydestä, vaatimuksen riskialttiudesta ja mahdollisista jäljityskustannuksista. Tähän esitetään yhdeksi ratkaisuksi arvopohjaista vaatimusten jäljittämistä. (Heindl & Biffl 2005)

Vaatimustenhallintaan tarkoitetut työkalut ovat useimmiten kohdealueesta riippumattomia, eivätkä ne siis voi tarjota kohdealueeseen liittyvää tukea. Tämä heikentää esimerkiksi vaatimusanalyysia. On kuitenkin olemassa myös joitakin työkaluja, jotka antavat kohdealueesta riippuvaa tukea. Kohdealueriippuvaista tukea antavat työkalut eivät ole kuitenkaan yleistyneet ja niissä on vielä lukuisia ongelmia. Ongelmat johtuvat muun muassa siitä, että kohdealueita on vaikeaa mallintaa tarpeeksi tarkalla tasolla ja vaikka siinä onnistuttaisiinkin, niin malleja on vaikea pitää ajan tasalla, koska kohdealueet voivat kehittyä nopeasti. (Ruhe, Eberlein & Pfahl 2002)

Systemaattinen vaatimusten jäljittäminen on edullisempaa tehdä kehitysvaiheen aikana kuin silloin, kun ohjelmisto on jo käytössä. Jäljittäminen on edullisempaa kehitysvaiheessa kuin käyttöönoton jälkeen siksi, koska kehitysvaiheen aikana ovat mukana alkuperäiset järjestelmän kehittäjät, kun taas käyttöönoton jälkeen he eivät välttämättä enää ole mukana. (Heindl & Biffl 2005)

Dokumentteihin (erityisesti vaatimusmäärittelydokumenttiin) perustava vaatimustenhallinta on mahdollista, mutta siinä on useita ongelmia. Dokumentteja on vaikea pitää ajan tasalla ja dokumentteihin tehtävät muutokset joudutaan tekemään manuaalisesti. Vaatimusten välisten riippuvuuksien ja vaatimusten jäljittäminen on vaivalloista. Dokumenttien hallinta on haasteellista, mikäli dokumenttia täydentäviä henkilöitä on useita ja jos he ovat maantieteellisesti hajallaan. Myöskään hylätyille ja poistetuille vaatimuksille ei ole mielekästä säilytyspaikkaa. (Wiegers 2003, 367–368)

Vaatimustenhallintatyökalujen käytön etuihin puolestaan kuuluu se, että ne helpottavat vaikutusanalyysia mahdollistamalla vaatimusten linkittämisen, jolloin vaatimusmuutosten vaikutusten analysointi on helpompaa. Myös vaati-

musten tilojen seuranta helpottuu, mikäli vaatimukset tallennetaan esimerkiksi tietokantaan. (Wiegers 2003, 370–371)

3.7 Yhteenveto

Vaikka suurin osa vaatimusmäärittelyprosessista keskittyy ohjelmiston elinkaaren alkuvaiheeseen, vaatimustenhallinnalla on erityisen keskeinen rooli ohjelmiston elinkaaren myöhemmissä vaiheissa, kun ohjelmisto on jo toimitettu ja sitä ylläpidetään. Vaatimusten tunnistaminen, mallintaminen, analysointi ja varmentaminen pyritään toki tekemään ohjelmiston elinkaaren alkuvaiheessa mahdollisimman hyvin, mutta vaatimusten muuttuessa ohjelmiston elinkaaren aikana tarvitaan vaatimustenhallintaa.

Kun vaatimukset muuttuvat ajan kuluessa ja asiakas esittää muutospyynnön, muutospyynnön riskit, vaikutukset, hyödyllisyys ja kustannukset on arvioitava. Muutospyynnön tulee myös olla liiketoimintatavoitteiden ja resurssien mukainen. Kun arvioidaan, hyväksytäänkö muutospyyntö, myös ohjelmiston elinkaaren vaiheella voi olla merkitystä. Ohjelmiston elinkaaren loppuvaiheessa esitettyä riskialtista muutospyyntöä ei välttämättä enää olla valmiita toteuttamaan, mikäli on tiedossa, että ohjelmisto ei ole enää kauan käytössä. Toisaalta ohjelmiston elinkaaren loppuvaiheessa esitettyyn muutospyyntöön voidaan suostua myös asiakkuuden ylläpitämiseksi.

Versionhallinnan tarkoituksena on hallita ohjelmiston elinkaaren aikana kehitettyjä ohjelmistoversioita ja ohjelmistoon liittyvää dokumentaatiota. Kun ohjelmisto on jo elinkaarensa loppuvaiheessa, on versiohistoriaa jo kertynyt. Jos esimerkiksi ohjelmiston ylläpidon aikana tulee asiakkaalta muutospyyntö, on syytä ensin tarkistaa, onko samaa muutospyyntöä ehdotettu jo aiemminkin. Se liittyy myös toiseen vaatimustenhallinnan osaan, vaatimusten tilojen seurantaan. Vaatimusten tilojen seurannassa painotetaan sitä, että määritellään ja kirjataan ylös, missä vaiheessa kukin vaatimus on. Niin kuin vaatimusten tilojen seurantaa käsittelevän kohdan yhteydessä kerrottiin, myös hylätyt vaatimukset

on syytä kirjata ja tärkeää on myös kirjata syy hylkäämiseen. Jo kerran hylätyt vaatimukset voivat tulla elinkaaren myöhemmässä vaiheessa uudestaan esille. Silloin säästetään työtä, mikäli jo kerran hylättyjen vaatimusten toteutusmahdollisuuksia ei tarvitse alkaa tutkimaan uudestaan.

Muutostenhallinnan, versionhallinnan ja vaatimusten tilojen seurannan lisäksi myös vaatimusten jäljittäminen on osa vaatimustenhallintaa. Vaatimusten jäljittämisessä selvitetään vaatimusten riippuvuuksia toisista vaatimuksista, vaatimusten taustoja ja sitä, mihin ohjelmiston osaan kukin vaatimus liittyy. Vaatimusten jäljittäminen on edullisempaa tehdä ohjelmiston elinkaaren alkuvaiheessa, kun alkuperäiset kehittäjät ovat vielä mukana. Alkuperäisille kehittäjille on kehitystyön aikana kertynyt syvällistä ymmärrystä ohjelmiston lainalaisuuksista. Ylläpitovaiheessa alkuperäiset kehittäjät eivät enää välttämättä ole mukana, joten dokumentoidusta vaatimusten jäljittämisestä voi ylläpitovaiheessa saada arvokasta tietämystä ohjelmistosta.

Vaatimustenhallintaan on olemassa erilaisia menetelmiä ja työkaluja. Niistä voi olla oikein käytettynä apua vaatimustenhallinnassa, mutta ne eivät takaa hyvää lopputulosta. Esimerkiksi Heindl ja Biffl (2005) mainitsevat vaatimustenhallintaan tarkoitettujen työkalujen ongelmaksi sen, että ne eivät useinkaan erottele vaatimuksia niiden arvon perusteella. Vaatimuksen ja sitä kautta myös vaatimuksen jäljittämisen arvo riippuu esimerkiksi sidosryhmän tärkeydestä, riskistä ja jäljityskustannuksista. Toisaalta taas edellä mainittuihin asioihin vaikuttaa se, missä elinkaaren vaiheessa ohjelmisto on. Esimerkiksi vaatimuksen riskin suuruus voi olla sitä suurempi, mitä pidemmälle elinkaari on edennyt.

Elinkaaren hallinnalla on yhtymäkohtia vaatimustenhallintaan. Oikeastaan elinkaariajattelu antaa syvyyttä vaatimustenhallinnalle. Vaatimustenhallinta ei ole irrallaan ympäristöstään, vaan esimerkiksi elinkaaren vaiheella voi olla merkitystä muutospyynnön toteuttamiseen. Seuraavassa luvussa käsitelläänkin tarkemmin elinkaaren hallintaa ja elinkaarimalleja.

4 ELINKAAREN HALLINTA JA ELINKAARIMALLIT

Ohjelmiston elinkaari kattaa yleensä vaiheet erilaisilla painotuksilla elinkaarimallista riippuen ohjelmiston hankkimisen suunnittelusta aina ylläpitoon ja ohjelmiston käytöstä luopumiseen asti. Eri malleissa kuitenkin painotetaan elinkaaren eri vaiheita.

Tässä luvussa esitellään elinkaaren hallintaa ja muutamia elinkaarimalleja. Ensin kerrotaan lyhyesti yleisesti tuotteiden elinkaaresta ja pohditaan, missä määrin ohjelmiston elinkaaresta löytyy yhtenevyyksiä esimerkiksi fyysisten tuotteiden elinkaareen ja niihin kehitettyihin malleihin. Sen jälkeen kuvaillaan ohjelmiston elinkaariajattelun taustaa. Luvussa esitellään myös muutamia elinkaarimalleja ja pohditaan, miten elinkaaren kustannuksia voidaan arvioida. Elinkaarimalleja esiteltäessä keskitytään erityisesti ylläpidon aikaan painottuvaan Bennettin ja Rajlichin (2000a; 2000b) elinkaarimalliin. Luvun lopuksi kootaan yhteen luvun keskeisin sisältö.

4.1 Tuotteen elinkaari

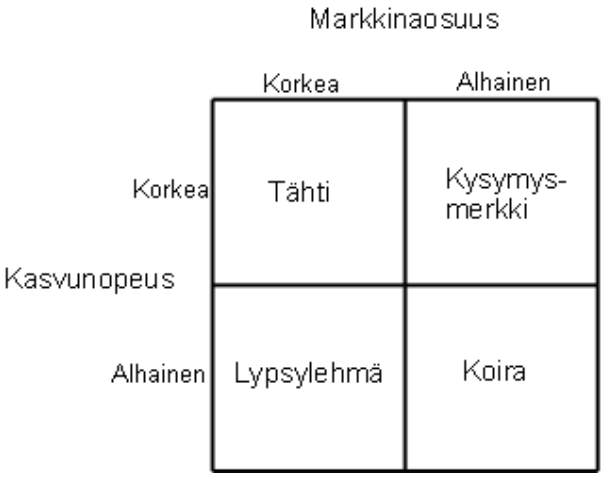
Tuotteiden elinkaarta (product life cycle, PLC) on tutkittu paljon aina 1960–1970-luvuilta alkaen (esim. Kotler 1965; Wasson 1971; Hofer 1975). Kotler (2003, 328–329) jakaa tuotteen elinkaaren neljään vaiheeseen. Vaiheet ovat esittely-, kasvu-, kypsyys- ja taantumisvaihe.

Esittelyvaiheessa tuotteen myynnin kasvu on vielä hidasta eikä voittoa juuri kerry. Kasvuvaiheessa myynnin ja tuoton kasvu on nopeaa. Kypsyysvaiheessa myynti ja tuotot ovat saavuttaneet huippunsa ja vakiintuvat tai alkavat vähentyä, kun kilpailu kovenee. Tuotteen elinkaari noudattelee S-käyrää aina esittely- ja kasvuvaiheesta kypsyysvaiheeseen. Taantumisvaiheessa sekä myynti että tuotot heikentyvät. Tuotteen strategiaa muutetaan elinkaaren vaiheesta riippuen. Esimerkiksi hinnoittelu voi muuttua sekä tuotteen ominaisuuksiin ja myyn-

ninedistämiseen käytetään resursseja eri tavalla eri elinkaaren vaiheissa. (Kotler 2003, 328–340)

Myös ohjelmiston elinkaarta kuvailtaessa käytetään vastaavanlaisia vaiheita kuin Kotler (2003, 328–329) käyttää jakaessaan tuotteen elinkaaren eri vaiheisiin. Esimerkiksi Lehner (1991) erottelee ohjelmiston elinkaaresta esittely-, kasvu- ja kypsyysvaiheen mittaamalla transaktioiden määrää ajan kuluessa. Taantumisvaihetta Lehner ei ole tutkimuksissaan kyennyt mittaamalla todentamaan.

Yksi tunnetuimmista tuotteen elinkaaren huomioivista malleista on Boston Consulting Groupin (BCG) matriisi. Boston Consulting Groupin portfoliomatriisissa tuotteen elinkaaren vaihe otetaan kasvunopeuden kautta huomioon yhtenä kahdesta muuttujasta, joihin kiinnitetään erityistä huomiota tehtäessä strategisia päätöksiä tuotteen hallinnasta. Toisena muuttujana on markkinaosuus. Kuviossa 4 havainnollistetaan BCG-matriisin nelikenttää, jonka osat on nimetty lypsylehmäksi, koiraksi, tähdeksi ja kysymysmerkiksi. (Henderson 1973)



KUVIO 4. BCG-matriisi (Henderson 1973)

Lypsylehmä kuvaa tuotteita, joilla yrityksellä on korkea markkinaosuus, mutta kasvu on hidasta (viittaa elinkaaren loppupuoleen). Lypsylehmäryhmään kuu-

luvista tuotteista yritys saa tuottoja, jotka voidaan kohdistaa elinkaaren alkuvaiheessa oleviin tuotteisiin, joilla on kasvupotentiaalia. Tällaisia tuoteryhmiä ovat kysymysmerkit ja tähdet. Tähdillä on paljon kasvupotentiaalia ja suuri markkinaosuus. Toisaalta ne kuluttavat paljon rahaa, mutta myös tuottavat hyvin. Kysymysmerkeillä on kasvupotentiaalia, mutta pieni markkinaosuus, niiden menestys ei siis ole varmaa. Markkinaosuutta kasvattamalla kysymysmerkeistä voi tulla tähtiä, mutta markkinaosuuden kasvattaminen vaatii yleensä huomattavaa taloudellista tukea. Kysymysmerkkejä kutsutaan myös jossain lähteissä villikissoiksi (esim. Hambrick, MacMillan & Day 1982). Koiraryhmään kuuluvat tuotteet ovat elinkaarensa lopulla ja niiden markkinaosuus on myös pieni. Niistä tulisikin pyrkiä eroon, koska ne eivät ole tuottavia eikä niistä todennäköisesti ole tulossakaan tuottavia. (Henderson 1973; Hambrick ym. 1982)

Ohjelmistojen elinkaaren hallinnassa voi havaita yhteneväisyyksiä perinteisempien tuotteiden elinkaaren hallinnan kanssa. Esimerkiksi S-käyrän mukaiset vaiheet ovat melko yleispäteviä monenlaisille tuotteille. Vaikka ohjelmiston elinkaaren hallinnassa onkin joitakin samoja piirteitä perinteisempien tuotteiden elinkaaren hallintaan, ohjelmiston elinkaaren hallinnassa on myös merkittäviä erityispiirteitä. Niitä käsitellään seuraavassa kohdassa tarkemmin.

4.2 Ohjelmiston elinkaariajattelun taustaa ja erityispiirteitä

Ohjelmiston elinkaari jakaantuu luontevasti ohjelmiston toimitusta edeltävään aikaan ja toimituksen jälkeiseen aikaan. Kohdassa 4.3 esitellään lyhyesti muutamia tunnettuja elinkaarimalleja, joissa painotetaan toimitusta edeltävää aikaa. Kohdassa 4.4 esitellään elinkaarimalli, joka puolestaan painottuu toimituksen jälkeiseen aikaan, kun ohjelmisto on jo otettu käyttöön.

Ohjelmiston elinkaaren keskeinen piirre on toimituksen jälkeisen ajan, ylläpidon, merkittävä asema niin ajallisesti kuin kustannustenkin näkökulmasta. Ylläpidolla tarkoitetaan toimituksen jälkeistä aikaa ja IEEE:n standardissa 1219 vuodelta 1998 ohjelmiston ylläpito määritellään seuraavasti: *”toimituksen jälkeen*

tapahtuva ohjelmistotuotteen muuntaminen, jonka tarkoituksena on korjata vikoja, parantaa suorituskykyä tai muita ominaisuuksia tai mukauttaa tuote muuttuneeseen ympäristöön."

Jo vuonna 1980 julkaistussa tutkimuksessa Lientz ja Swanson toteavat, että ohjelmistojen elinkaaresta on kirjoitettu paljon, mutta huomio on kiinnitetty pääosin ohjelmiston suunnitteluun ja toteutukseen. Heidän mukaansa ylläpito on jäänyt vähemmälle huomiolle, vaikka siihen kuluu huomattava määrä resursseja ja se on kestoltaankin hyvin merkittävä. Esimerkiksi Sommervillen (1998, 673) mukaan ohjelmiston ylläpitokustannukset ylittävät yleensä ohjelmiston kehittämiskustannukset. Ylläpidon merkitys on erityisen keskeinen ohjelmistoilla, joiden elinaika on pitkä ja jotka ovat laajoja, monimutkaisia ja asiakkaille kriittisiä (Koskinen, Lintinen, Sivula & Tilus 2004a, 13).

Niessink ja van Vliet (2000) väittävät jopa, että ohjelmiston kehitys ja ylläpito koetaan täysin eri tavalla. Heidän mielestään ohjelmiston kehitys koetaan tuotteen kehittämisenä, kun taas ylläpito koetaan palvelun tarjoamisena, jolloin myös laatu koetaan eri tavalla. Oikeastaan Niessinkin ja van Vlietin (2000) voisi tulkita tarkoittavan sitä, että ohjelmisto muuttuu toimittajan näkökulmasta elinkaarensa aikana tuotteesta palveluksi.

4.3 Tunnettuja elinkaarimalleja

Tässä tutkielmassa keskitytään ylläpitoaikaa painottavaan Bennettin ja Rajlichin (2000a; 2000b) elinkaarimalliin. Koska tunnetuimmat elinkaarimallit kuitenkin kiinnittävät eniten huomiota ohjelmiston elinkaaren alkupuolelle sijoittuvaan ohjelmistokehitykseen, kerrotaan tässä luvussa lyhyesti niistä, jotta ei synny virheellistä mielikuvaa, että ohjelmiston ylläpitoon painottuvat elinkaarimallit olisivat yleisimpiä elinkaarimalleja. Tässä kohdassa esiteltyjen kahden tunnetun elinkaarimallin esittelyjen ei ole tarkoitus olla kattavia kuvauksia tai analyyseja malleista, vaan ainoastaan muistuttaa mallien keskeisestä asemasta.

Vesiputousmalli on tunnetuin elinkaarimalli ja siitä on olemassa useita erilaisia versioita. Lähtökohta vesiputousmallille on Winston Roycen vuonna 1970 julkaistu malli. Vesiputousmallissa on tutkijasta ja tutkimuksen ajankohdasta riippuen erilaisia vaiheita, mutta yleensä vaiheet sisältävät vaatimusmäärittelyn, analyysin, suunnittelun, toteutuksen, testauksen ja ylläpidon. Esimerkiksi Royce (1987) jakaa vaatimusmäärittelyn vielä tarkemmin järjestelmän vaatimusten määrittelemiseen ja ohjelmiston vaatimusten määrittelemiseen.

Vesiputousmallin lähtökohtana on, että vaiheet tehdään peräkkäisessä järjestyksessä ja edellisen vaiheen tuotos on seuraavan vaiheen syöte (Guimarães & Vilela 2005). Lisäksi Boehm (1981, 36–37) vielä lisää, että jokaiseen vaiheeseen kuuluu vahvistaminen ja varmentaminen, joiden avulla pyritään varmistamaan, että ohjelmisto vastaa laadittuja suunnitelmia ja toisaalta varmistetaan, että ohjelmisto täyttää sille asetetun tavoitteen.

Spiraalimallia on kuvailtu vesiputousmalliksi, jossa joka vaihetta tai sykliä edeltää riskianalyysi ja sen toteuttaminen on inkrementaalista. Spiraalimallissa yhdistetään useiden muiden mallien piirteitä. Spiraalimallissa on vaikutteita prototyyppimallista, evoluutiomallista, sykleihin perustuvasta mallista ja päätoiminnot on omaksuttu vesiputousmallista. (Guimarães & Vilela 2005)

Spiraalimallissa edetään sykleittäin ja jokaisessa syklissä on neljä vaihetta. Ensimmäisessä vaiheessa päätetään tavoitteista, vaihtoehdoista ja rajoitteista. Toisessa vaiheessa arvioidaan vaihtoehtoja ja analysoidaan riskejä. Tässä vaiheessa voidaan rakentaa prototyyppi tai simulaatio ohjelmistosta. Kolmas vaihe on varsinainen kehitysvaihe, johon kuuluu esimerkiksi määrittelyä, suunnittelua, toteutusta ja varmentamista. Neljännessä vaiheessa arvioidaan meneillään olevaa sykliä ja suunnitellaan seuraavia vaiheita. (Boehm 1986; Guimarães & Vilela 2005)

4.4 Vaiheistettu malli – esimerkki toimituksen jälkeiseen aikaan painottuvasta elinkaarimallista

Bennett ja Rajlich (2000a; 2000b) jakavat vaiheisiin perustuvassa mallissaan ohjelmiston elinkaaren viiteen vaiheeseen ja painottavat ylläpidon merkitystä. He kritisoivat sitä, että usein ylläpitoa pidetään vain yhtenä vaiheena ja siihen luetaan kuuluvaksi kaikki, mitä tapahtuu toimituksen jälkeen. On kuitenkin olemassa useita tutkimuksia (esim. Lientz & Swanson 1980; Kemerer & Slaughter 1999; Sahin & Zahedi 2001), joissa erotellaan ylläpidosta erilaisia tehtäviä tai vaiheita. Lientzin ja Swansonin (1980, 68) mukaan on olemassa mukauttavaa, täydentävää ja korjaavaa ylläpitoa. IEEE:n standardissa 1219 vuodelta 1998 mainitaan neljänneksi ylläpitotyypiksi vielä ehkäisevä ylläpito. Lisäksi esimerkiksi Chapin, Hale, Khan ym. (2001) jakavat ylläpidon vielä hienojakoisemmin klustereiden avulla.

Sahin ja Zahedi (2001) puolestaan jaottelevat käyttöönoton jälkeiset toimenpiteet kolmeen erilaiseen tyyppiin: takuu-, ylläpito ja parannustoimenpiteisiin. Takuutoimenpiteillä (*warranty*) varmistetaan, että järjestelmä toimii vaaditulla tavalla. Ylläpitotoimenpiteillä (*maintenance*) parannetaan ohjelmiston olemassa olevia toimintoja. Parannus- tai päivitystoimenpiteillä (*upgrade*) lisätään ohjelmistoon uusia toimintoja tai ominaisuuksia virheiden poistamisen lisäksi. (Sahin & Zahedi 2001)

Taulukossa 2 esitellään lyhyesti Bennettin ja Rajlichin (2000a; 2000b) vaiheisiin perustuvan elinkaarimallin vaiheet. Taulukossa on lyhyet kuvaukset vaiheista, mutta vaiheita kuvaillaan tarkemmin myöhemmin tässä kohdassa. Lisäksi taulukkoon on myös tiivistetty vaatimustenhallinnan merkitys eri vaiheissa. Vaatimustenhallinnan merkitystä eri elinkaaren vaiheissa perustellaan tarkemmin luvussa 5.

TAULUKKO 2. Bennettin ja Rajlichin (2000a; 2000b) elinkaarimallin vaiheet täydennettynä vaatimustenhallinnan merkityksellä eri vaiheissa

Vaiheen nimi suomeksi	**Vaiheen nimi englanniksi**	**Kuvaus**	**Vaatimustenhallinnan merkitys**
Alkukehitysvaihe	Initial development	- Ensimmäinen versio ohjelmistosta toteutetaan. - Ohjelmistosta saattaa vielä puuttua ominaisuuksia. - Arkkitehtuuri muotoutuu. - Ohjelmistotiimin osaaminen kohdealueesta lisääntyy.	Vaatimustenhallinta tukee vaatimusmäärittelyprosessin muita toimintoja.
Evoluutiovaihe	Evolution	- Usein ohjelmisto julkaistaan evoluutiovaiheessa. - Tehdään iteratiivisia muutoksia ja kehitystä. - Mukautetaan ohjelmisto muuttuviin vaatimuksiin. - Ohjelmistoa ollaan valmiita parantamaan aktiivisesti.	Vaatimustenhallinnan merkitys on keskeinen, koska muutoksia tehdään paljon.
Huoltovaihe	Servicing	- Ohjelmistolla ei ole enää tarkoituksenmukaista arkkitehtuuria, eikä toimivaa kehitystiimiä. - Muutosten tekeminen on vaikeaa ja kallista -> muutosten määrä pyritään minimoimaan. - Muutosten kustannus-hyötysuhde on huono. - Ohjelmisto ei ole enää liiketoiminnan kannalta ydintuote.	Muutosten määrä pyritään minimoimaan, mutta jos muutoksia tehdään, vaatimustenhallinta on erityisen tärkeää.
Vaiheittaisen käytöstäpoiston vaihe	Phase-out	- Ohjelmisto poistetaan vaiheittain käytöstä. - Ohjelmistoa ei enää muokata tai kehitetä. - Ohjelmistolla voi olla vielä käyttäjiä. - Ohjelmistosta kerätään vielä tuottoja niin kauan kuin se on mahdollista.	Vaatimustenhallinnan merkitys on vähäinen, koska ohjelmistoon ei tehdä muutoksia, vaikka uusia vaatimuksia esitettäisiinkin.
Lakkautusvaihe	Close-down	- Ohjelmisto poistetaan käytöstä. - Käyttäjät ohjataan käyttämään korvaavaa ohjelmistoa, jos mahdollista.	Vaatimustenhallinnan merkitys on vähäinen, koska ohjelmistoon ei tehdä muutoksia (siirryttäessä käyttämään korvaavaa ohjelmistoa, on vaatimustenhallinnalla merkitystä).

Vaiheistetun mallin ensimmäiseen vaiheeseen kuuluu ohjelmiston alkukehitys (*initial development*). Tässä vaiheessa ohjelmistosta tehdään ensimmäinen toimiva versio. Ensimmäisestä versiosta saattaa vielä puuttua joitakin ominaisuuksia, mutta ohjelmiston arkkitehtuuri on jo tämän vaiheen lopussa sama kuin myöhempänäkin elinkaaren aikana. Keskeistä vaiheen aikana on se, että ohjelmistotiimin jäsenten ymmärrys ja osaaminen kohdealueesta lisääntyy ja järjestelmän arkkitehtuuri muotoutuu. (Bennett & Rajlich 2000a; 2000b)

Toisena tulevassa evoluutiovaiheessa (*evolution*) ohjelmistoon tehdään iteratiivisia muutoksia ja kehitystä. Vaiheen keskeisenä tavoitteena on mukauttaa ohjelmisto käyttäjien muuttuviin vaatimuksiin ja ympäristön muutoksiin. Evoluutiovaiheessa tehdään muutoksia, jotka ovat seurausta siitä, että sekä kehittäjät että käyttäjät ovat oppineet ja saaneet kokemuksia ohjelmistosta ja pystyvät sen vuoksi tarkentamaan vaatimuksia. Evoluutiovaihe on vaihe, joka kestää niin kauan, kun ohjelmistoa ollaan valmiita parantamaan ja kehittämään aktiivisesti. Muutoksia tehdään asiakkaiden pyynnöstä tai kun kilpailukyvyn vuoksi kehitetään uusia toiminnallisuuksia. Usein muutokset voivat johtua myös lainsäädännön muuttumisesta. Joissain tapauksissa ohjelmisto julkaistaan jo alkukehitysvaiheen jälkeen, mutta yleensä ohjelmisto julkaistaan vasta evoluutiovaiheessa, kun ohjelmisto on jo ehtinyt käydä läpi useita iteraatiokierroksia. (Bennett & Rajlich 2000a; 2000b)

Ohjelmiston arkkitehtuuri ja ohjelmistotiimin tietämys mahdollistavat ohjelmiston evoluution. Ne mahdollistavat huomattavienkin muutosten tekemisen rikkomatta arkkitehtuurin yhtenäisyyttä. Kun jompikumpi edellä mainituista häviää, siirtyy ohjelmisto pois evoluutiovaiheesta ja siirtyy elinkaarensa seuraavaan vaiheeseen. Evoluutiovaiheen jälkeen tulevassa kolmannessa vaiheessa, huoltovaiheessa (*servicing*), muutosten tekeminen on jo vaikeaa ja kallista, ja siksi toteutettavien muutosten määrä pyritäänkin minimoimaan. Tässä vaiheessa ohjelmistoon tehdään enää pienehköjä muutoksia ja ohjelmisto ei ole enää liiketoiminnan kannalta ydintuote. Lisäksi muutosten kustannus-hyötysuhde

on jo melko huono. Toimintakriittisten ohjelmistojen ei pitäisi ikinä edes päästä tähän vaiheeseen. (Bennett & Rajlich 2000a; 2000b)

Neljännessä vaiheessa (*phase-out*) ohjelmisto poistetaan vaiheittain käytöstä. Tässä vaiheessa ohjelmistoa ei enää muokata tai kehitetä, mutta ohjelmistosta kerätään vielä tuottoja sen aikaa, kun se on mahdollista. Ohjelmistolla voi olla vielä käyttäjiä, mutta koska ohjelmistoon ei tehdä enää muutoksia, tulee käytöstä ajan mittaan vaikeaa. Tästä vaiheesta ei pääse enää helposti palaamaan aiempiin vaiheisiin, koska se vaatisi huomattavan muutospyyntömäärän toteuttamista. (Bennett & Rajlich 2000a; 2000b)

Viidennessä ja viimeisessä vaiheessa (*close-down*) ohjelmisto poistetaan käytöstä. Käyttäjät ohjataan käyttämään korvaavaa ohjelmistoa, jos sellainen on olemassa. Monissa yrityksissä järjestelmään tallennettu tieto on yrityksen toiminnan kanalta elintärkeää. Kun ohjelmisto poistetaan käytöstä, järjestelmään tallennetun tiedon siirto uuteen järjestelmään onkin suoritettava erityisen huolellisesti. (Bennett & Rajlich 2000a; 2000b)

4.5 Elinkaaren kustannusten arviointi

Ohjelmiston elinkaaren kustannusten arvioimiseen ei ole yksinkertaista ja tarkkaa keinoa, jota voisi yksiselitteisesti hyödyntää erityyppisille ohjelmistoille. Toisaalta haastavuutta lisää vielä se, että ohjelmistoprojektin alkuvaiheessa ei ole vielä saatavissa yksityiskohtaista tietoa ohjelmistosta, vaan kustannusarvio on perustettava ohjelmistolle asetettuihin vaatimuksiin, jotka saattavat olla vielä melko yleisellä tasolla. Myös uuden teknologian käyttö ja epätietoisuus ohjelmistoprojektiin osallistuvien henkilöiden osaamisesta vaikeuttavat kustannusten arviointia projektin alkuvaiheessa. (Sommerville 1998, 595–596). Toisaalta kustannusten arvioinnin vaikeus ei rajoitu vain ohjelmiston elinkaaren alkuvaiheeseen. Haastavuutta lisää myös se, että useimmat kustannusten arviointiin tarkoitetut mallit eivät sovellu hyvin ylläpitovaiheen kustannusten arviointiin (Koskinen, Lintinen, Sivula & Tilus 2004a, 13).

Ohjelmiston elinkaaren kustannusten arviointiin on kuitenkin olemassa useita erilaisia tekniikoita. Asiantuntija-arvioinnissa useat asiantuntijat tekevät oman arvionsa kustannuksista ja sen jälkeen arvioita vertaillaan ja niistä keskustellaan. Arviointi jatkuu iteratiivisesti, kunnes saavutetaan yhtenäinen arvio. Analogian perusteella arvioitaessa puolestaan uuden ohjelmiston kustannuksia arvioidaan yhtenevyyksien avulla jo päättyneisiin projekteihin. Analogiaan perustuvan arvion tekeminen edellyttää kuitenkin sitä, että vertailukohtia täytyy löytyä samalta kohdealueelta. Parkinsonin lakiin perustuvassa arvioinnissa lähtökohtana on, että kustannuksista päätetään käytettävissä olevien resurssien perusteella. Otetaan siis huomioon, kuinka kauan aikaa on käytettävissä ja kuinka monta henkilöä osallistuu työhön ja lasketaan arvio suoraan käytettävissä olevasta ajasta ja henkilömäärästä. Yhdessä kustannusten arviointitekniikassa asetetaan kustannusarvioksi se summa, joka asiakkaalla on mahdollista käyttää projektiin. Tällöin arvio pohjautuu täysin asiakkaan budjettiin, eikä lainkaan ohjelmiston toiminnallisuuksiin. Mainittujen laadullisten tekniikoiden lisäksi on olemassa myös kvantitatiivisempia tekniikoita kustannusten arvioimiseen. Algoritmiset kustannusarvioinnit voivat perustua jo päättyneiden projektien ominaisuuksien ja kustannusten arviointiin. Yksi tunnetuimmista algoritmisista kustannusten arviointimalleista on COCOMO. (Sommerville 1998, 596–599)

Boehm (1981) kehitti alkuperäisen COCOMO-mallin, mutta mallia on sen jälkeen paranneltu ja siitä on tehty uusia versioita (esim. Boehm, Clark, Horowitz ym. 1995). COCOMO pohjautuu toteutetuista hankkeista saatuun kokemusperäiseen tietoon, ja siinä kuvataan yhtälöt projektin vaatimalle työmäärälle sekä kestolle. COCOMO:sta on olemassa kolme mallia, yksinkertainen, yksityiskohtainen ja niiden välimuoto. Yksikertainen malli sopii aikaisessa vaiheessa tehtävään karkeaan kustannusten arviointiin, kun taas yksityiskohtaisemmissa malleissa otetaan huomioon monipuolisemmin eri tekijöitä ja huomioidaan myös

elinkaaren vaihe. (Boehm 1981, 57–58; Boehm ym. 1995; Koskinen, Lintinen, Sivula & Tilus 2004b, 37)

COCOMO ei ole välttämättä kuitenkaan tehokas malli koko elinkaaren kustannusten arviointiin. Mallia on arvosteltu esimerkiksi siitä, että sen avulla ei pysty arvioimaan kovinkaan tarkasti ylläpitovaiheen kustannuksia, jotka vievät kuitenkin keskeisen osan koko ohjelmiston elinkaaren kustannuksista. Ylläpidon kustannusten arvioiminen olisi tärkeää myös siksi, että voitaisiin tehdä perusteltuja päätöksiä siitä, missä määrin ylläpidon avulla pyritään pidentämään ohjelmiston elinikää. (Koskinen ym. 2004a, 13–14)

On olemassa myös ylläpidon kustannusten arviointiin kehitettyjä kustannusten arviointimenetelmiä. Yksi sellainen on Softcalc, joka perustuu COCOMO- ja FPA-malleihin. Albrechtin ja Gaffneyn (1983) mukaan FPA:ssa (Function Point Analysis) eli toimintopisteanalyysissa painotetaan toimintopisteiden määrä erilaisilla kertoimilla (esimerkiksi monimutkaisuuskertoimella) ja muunnetaan saatu tulos työmääräksi. Softcalc-mallia käytetään yksittäisten ylläpitopyyntöjen kustannusten arviointiin. Softcalc-mallin huonona puolena on kuitenkin se, ettei sitä ole validoitu empiirisesti. (Sneed 1995; Koskinen ym. 2004b, 12, 15, 39)

Softcalc-mallissa on seitsemän vaihetta. Ohjelmiston koko, monimutkaisuus ja laatu määritellään automaattisessa auditoinnissa. Sen jälkeen rajataan vaikutusalue, jota esitetty ylläpitopyyntö koskee. Vaikutusalueen koko mitataan vähintään kahdella tavalla (esimerkiksi koodirivien määrällä). Saatuun mittaustulokseen huomioidaan alun auditoinnissa selvitetty monimutkaisuuskerroin, ulkoiset ja sisäiset laatutekijät sekä tuottavuuskerroin. Kun mittaustuloksessa on huomioitu edellä mainitut tekijät, muutetaan se ylläpitotyöksi tuottavuustaulukon avulla. Yleisesti voidaan sanoa, että ohjelmiston ylläpidon tuottavuus on noin puolet ohjelmiston kehittämisen tuottavuudesta. Tämä johtuu toisaalta siitä, että ylläpidossa menee aikaa ohjelmiston ja sen toiminnan ymmärtämiseen ja

toisaalta muutettujen komponenttien integroimiseen ja uudelleentestaamiseen. (Sneed 1995; Koskinen ym. 2004b, 36)

4.6 Yhteenveto

Tuotteen elinkaari ei ole koko tuotteen eliniän ajan samanlainen, vaan usein alun melko hitaan kasvun jälkeen seuraa kiihtyvän kasvun vaihe. Kiihtyvä kasvu hidastuu kypsyysvaiheessa ja pikkuhiljaa alkaa taantumisvaihe. Ennen pitkää tuote häviää markkinoilta tai korvautuu toisella tuotteella. Ohjelmiston elinkaarikaan ei ole tasapaksu ja kauttaaltaan samanlainen, vaan siinä on erilaisia vaiheita. Ohjelmistolle on tyypillistä, että sen elinkaari jakautuu luontevasti aikaan ennen ohjelmiston toimitusta ja aikaan ohjelmiston toimituksen jälkeen.

Monissa ohjelmiston elinkaarimalleissa painotetaan ohjelmiston kehittämisvaiheita eli aikaa ennen ohjelmiston toimittamista. Ohjelmiston toimittamisen jälkeinen ylläpitoaika on kuitenkin sekä kustannusten että keston näkökulmasta hyvin merkittävä osa koko ohjelmiston elinkaarta. Yksi toimituksen jälkeiseen aikaan painottuvista elinkaarimalleista on kohdassa 4.4 esitelty Bennettin ja Rajlichin (2000a; 2000b) vaiheistettu malli, jossa jaetaan ylläpitoaika erilaisiin osiin niiden keskeisimpien tehtävien perusteella. Tätä Bennettin ja Rajlichin (2000a; 2000b) mallia käytetään pohjana seuraavassa luvussa koottavalle viitekehykselle, jossa elinkaarimallia käytetään perustana vaatimustenhallintaan liittyvien päätösten tekemiseen.

Ohjelmiston kustannusten arviointiin on useita malleja. Matemaattisista malleista yksi tunnetuimpia on COCOMO, mutta ylläpidon kustannusten arvioinnin kannalta huonona puolena on se, että COCOMO-malli soveltuu huonosti ylläpidon kustannusten arviointiin. On olemassa myös nimenomaan ylläpidon kustannusten arviointiin suunniteltuja malleja, kuten Softcalc-malli. Softcalc-mallia ei ole kuitenkaan empiirisesti validoitu.

Vaikka malleja ohjelmiston elinkaaren kustannusten arviointiin on olemassa, kaikkien mallien toimivuutta ei ole testattu käytännössä. Vaikka malleille olisikin empiirisesti todistettua näyttöä, ei niitä välttämättä käytetä kovin yleisesti esimerkiksi todellisissa ylläpitotilanteissa muutospyyntöjen arvioinnissa. Kustannus-hyötyajattelu on varmasti taustalla esimerkiksi ylläpitopyyntöjen toteutuksen arvioinnissa, vaikka muutospyynnön kustannusten arvioinnissa ei mitään varsinaista matemaattista mallia käytettäisikään. Muutospyyntöjen hyväksymisen tai hylkäämisen taustalla voi olla kuitenkin myös tekijöitä, jotka vaikuttavat välillisesti kustannuksiin ja hyötyihin. Esimerkiksi asiakkuudella, ohjelmiston elinkaaren vaiheella ja lukuisilla muilla tekijöillä voi olla vaikutusta kustannuksiin ja hyötyihin laaja-alaisemmin ajateltuna. Tällöin pelkkä muutoksesta aiheutuvien suorien kustannusten arviointi ei välttämättä ole kokonaisuuden, esimerkiksi asiakkuuden elinkaaren, kannalta paras ratkaisu. Seuraavassa luvussa kootaan viitekehys, jossa otetaan huomioon myös muita kuin suoria kustannuksia vaatimustenhallintaan liittyvien päätösten arvioinnissa.

5 VIITEKEHYS VAATIMUSTENHALLINTAAN

Edellisissä luvuissa on käsitelty vaatimustenhallintaa sekä elinkaaren hallintaa ja elinkaarimalleja. Tässä luvussa muodostetaan viitekehys vaatimustenhallintaan edellisten lukujen pohjalta. Muutospyynnöt ovat viitekehyksessä keskeisessä asemassa ja vaatimustenhallintaa tarkastellaankin viitekehyksessä muutospyyntöjen arvioinnin kautta.

Luvussa kootaan yhteen keskeisiä asioita, jotka tulisi ottaa huomioon vaatimustenhallintaan liittyvissä ratkaisuissa, erityisesti muutospyyntöjen arvioinnissa ja hallinnassa. Muutospyynnöillä tarkoitetaan tässä tapauksessa ohjelmiston kehittämiseen ja järjestelmävirheiden korjaamiseen liittyviä ehdotuksia tai pyyntöjä. Ensimmäistä kertaa esitettäessä muutospyynnöt voivat olla vielä tarkentumattomia ehdotuksia, mutta ennen kuin niitä voidaan toteuttaa, niistä täytyy muokata täsmällisiä ja ristiriidattomia vaatimuksia. Usein muutospyynnön ehdottaja on ohjelmistoa käyttävästä yrityksestä ja muutospyyntö ehdotetaan ohjelmistoa kehittävälle yritykselle, mutta muutospyyntöjä voidaan ehdottaa myös ohjelmistoa kehittävän yrityksen sisältä.

5.1 Viitekehyksen vaatimustenhallintaan vaikuttavat tekijät

Luvussa 3 kerrottiin vaatimustenhallinnasta ja siihen liittyvistä toiminnoista. Tässä kohdassa kootaan 3. luvun pohjalta keskeisiä asioita, jotka tulee huomioida vaatimustenhallinnassa. Koska tässä luvussa koottavassa viitekehyksessä tarkastellaan vaatimustenhallintaa muutospyyntöjen ja muutostenhallinnan kautta, vaatimustenhallintaan vaikuttavat tekijät on jaoteltu jo tässä vaiheessa tekijöihin, jotka tulee huomioida muutospyyntöjen arvioinnissa ja toisaalta tekijöihin, jotka tulee huomioida, kun muutospyyntö on hyväksytty ja se päätetään toteuttaa. Varsinaisessa viitekehyksessä käsitellään vain muutospyynnön arviointiin liittyviä tekijöitä. On kuitenkin huomioitava, että hyväksyttyjen muutospyyntöjen hallinnassakin tarvitaan vaatimustenhallintaa.

Huomioonotettavia tekijöitä muutospyyntöä arvioitaessa ovat:

- muutospyyntöjen määrä (Bennett & Rajlich 2000a; 2000b)
- kustannukset (sisältäen työmäärän arvioinnin) (Boehm 1981; Boehm, Clark, Horowitz ym. 1995; Sneed 1995; Sommerville 1998, 595–599; Koskinen, Lintinen, Sivula & Tilus 2004a & 2004b)
- riskit (Bohner 1996; Wiegers 2003, 344; Cheng ja Atlee 2007)
- vaikutukset (Bohner 1996; Wiegers 2003, 344; Cheng ja Atlee 2007)
- onko sama muutospyyntö ehdotettu ennenkin (vertailu aiemmin esitettyihin muutospyyntöihin) (Wiegers 2003, 322–323)
- vaatimuksen jäljittäminen ja linkittäminen muihin vaatimuksiin (tukee muutostenhallinnan vaikutusten arviointia) (Gotel & Finkelstein 1994; Nuseibeh & Easterbrook 2000; Wiegers 2003, 353–354; Heindl & Biffl 2005)
- ohjelmiston arkkitehtuurin yhtenäisyys (Bennett & Rajlich 2000a; 2000b)
- tietämys ohjelmistosta (Bennett & Rajlich 2000a; 2000b)
- muutospyynnön luokittelu (esimerkiksi mukauttava, korjaava, ehkäisevä, täydentävä) (Lientz ja Swanson 1980, 68; Bennett & Rajlich 2000a; Pigoski 2002)
- vaatimuksen/muutospyynnön laatu (Mannion & Keepence 1995; NATURE Team 1996; Cheng & Atlee 2007)

Vaatimustenhallinta ei pääty siihen, että muutospyyntö hyväksytään. Sen jälkeen on huomioitava monia asioita ennen kuin hyväksytystä muutospyynnöstä on tullut osa ohjelmistoa. Luvun 3 perusteella havaittiin, että ainakin seuraaviin asioihin tulee kiinnittää huomiota, kun muutospyyntö on päätetty hyväksyä:

- tilatietojen ylläpito (Dart 1991; Wiegers 2003, 314)
- prioriteetti (Cheng ja Atlee 2007; Wiegers 2003, 333–338)
- toteutusaikataulu (Wiegers 2003, 333–338)
- muutoksen varmentaminen (Wiegers 2003, 333–338)
- ohjelmistoversioiden kontrollointi (Dart 1991; SWEBOK 2004, 108–112)
- julkaisujen suunnittelu ja hallinta (miten ja milloin muutos toimitetaan asiakkaalle) (Dart 1991; SWEBOK 2004, 108–112)
- dokumentteihin tulevat muutokset (Estublier 2000; Wiegers 2003, 314–319)

Kohdassa 5.4 esiteltävässä viitekehyksessä jaotellaan tässä kohdassa mainittuja tekijöitä. Viitekehykseen on valittu tämän kohdan perusteella sellaisia muutospyyntöä arvioitaessa huomioonotettavia tekijöitä, jotka vähenevät tai lisääntyvät elinkaaren etenemisen myötä.

5.2 Viitekehyksen taustalla oleva elinkaarimalli

Kohdassa 4.4 esiteltiin Bennettin ja Rajlichin (2000a; 2000b) vaiheisiin perustuva ohjelmiston elinkaarimalli, joka useimmista muista elinkaarimalleista poiketen jakaa ylläpidon useampaan vaiheeseen. Myös Lehner (1991) on elinkaarimalleihin ja elinkaaren hallintaan liittyvässä tutkimuksessaan painottanut käyttöönoton jälkeistä aikaa. Koska tämä tutkimus keskittyy erityisesti vaatimustenhallintaan nimenomaan ylläpidon aikana, sopii Bennettin ja Rajlichin (2000a; 2000b) ja Lehnerin (1991) elinkaarimallit pohjaksi viitekehyksessä käytettävälle elinkaarimallille.

Kuviossa 5 on esitetty elinkaarimalli, jota käytetään viitekehystä muodostettaessa. Kuviossa on kuvattu Bennettin ja Rajlichin (2000a; 2000b) vaiheiden suhde

Lehnerin (1991) jaottelemiin elinkaaren vaiheisiin. Kuvion 5 elinkaaren vaiheet ovat ajan suhteen oikeassa järjestyksessä, mutta eivät ajan suhteen mittakaavassa, koska eri vaiheiden suhteet voivat olla ohjelmistosta riippuen erilaiset.

KUVIO 5. Bennettin ja Rajlichin (2000a; 2000b) elinkaarimalli täydennettynä Lehnerin (1991) elinkaaren vaiheilla

Lehner (1991) on erotellut tutkimuksessaan esittely-, kasvu- ja kypsyysvaiheen. Esittelyvaihe sijoittuu käyttöönoton jälkeiseen aikaan, jolloin ohjelmiston käyttö on vielä vähäistä, mutta kasvaa jatkuvasti. Esittelyvaiheessa ohjelmiston käyttäjiä koulutetaan käyttämään ohjelmistoa. Esittelyvaihe kestää sitä kauemmin, mitä räätälöidymmästä ja yksilöllisemmästä ohjelmistosta on kyse.

Bennettin ja Rajlichin (2000a; 2000b) mukaan joissain tapauksissa ohjelmisto voidaan julkaista jo heti alkukehitysvaiheen jälkeen, mutta yleensä julkaisu tapahtuu evoluutiovaiheessa. Lehnerin (1991) määrittelemä kasvuvaihe puolestaan alkaa silloin, kun testit on suoritettu sekä testeissä havaitut puutteet ja virheet on korjattu. Bennettin ja Rajlichin (2000a; 2000b) nimeämä evoluutiovaihe ja erityisesti siitä vielä osa, jota Lehner (1991) kutsuu kasvuvaiheeksi, on vaatimustenhallinnan kannalta erittäin keskeinen. Silloin ohjelmistoa kehitetään ja siihen tehdään muutoksia aktiivisesti.

Bennettin ja Rajlichin (2000a) määrittelemä huoltovaihe on rinnastettavissa Lehnerin (1991) määrittelemään kypsyysvaiheeseen. Lehnerin (1991) mukaan ohjelmiston käyttö saavuttaa huippunsa tässä vaiheessa. Huoltovaiheelle on tyypillistä, että yhtenäinen rakenne ja arkkitehtuuri heikkenevät sekä ohjelmiston tuntevien henkilöiden määrä vähenee. Nämä asiat johtavat siihen, että aktiivinen ohjelmiston parantaminen on vaikeaa.

Kuvioon 6 on koottu mukailtu malli Bennettin ja Rajlichin (2000a; 2000b) sekä Lehnerin (1991) elinkaaren vaiheista. Elinkaarimallia käytetään viitekehyksen pohjana. Kuviossa on yhdistetty vaiheittaisen käytöstäpoiston vaihe ja lakkautusvaihe, koska niiden aikana ohjelmistoon ei enää tehdä muutoksia, jolloin vaatimustenhallinnan merkityskin on hyvin rajattu. Toki on huomattavaa, että lakkautusvaiheessa siirryttäessä mahdollisesti käyttämään korvaavaa ohjelmistoa, voi vaatimustenhallinnalla ja siihen liittyvillä dokumenteilla olla keskeinen merkitys. Sen käsittely on kuitenkin rajattu tästä tutkielmasta pois.

Ohjelmiston elinkaari →				
Alkukehitysvaihe	**Esittelyvaihe**	**Evoluutio/kasvuvaihe**	**Huoltovaihe**	**Vaiheittaisen käytöstäpoiston ja lakkauttamisen vaihe**

KUVIO 6. Viitekehyksen elinkaarimalli

Viitekehyksen pohjana olevassa elinkaarimallissa painopiste on käyttöönoton jälkeisessä ajassa. Käyttöönottoa edeltävä aika kuvataan vain lyhyesti alkukehitysvaiheena. Alkukehitysvaiheen jälkeen tulee esittelyvaihe, jolloin käyttäjät opettelevat käyttämään ohjelmistoa ja testauksen aikana havaittuja puutteita voidaan vielä korjata. Alkukehitysvaiheen ja esittelyvaiheen raja määritellään tässä tutkielmassa siihen hetkeen, kun ohjelmisto on toimitettu asiakkaalle.

Esittelyvaiheen alkaessa ohjelmisto ei välttämättä vielä ole tuotantokäytössä, vaan se voi olla vielä asiakkaalla testikäytössä. Kun ohjelmistossa on sopimuksessa sovitut ominaisuudet ja kun ne on saatu testattua, alkaa ohjelmiston evoluutio/kasvuvaihe. Esittelyvaihe on erotettu omaksi vaiheekseen, koska vaikka se oikeastaan kuuluukin julkaisun jälkeiseen aktiivisen kehittämisen vaiheeseen (evoluutio/kasvuvaihe), on siinä myös alkukehitysvaiheeseen kuuluvia piirteitä. Esittelyvaihe on eräänlainen välivaihe, joka yhdistää alkukehityksen siihen, kun ohjelmisto otetaan täysipainoisesti tuotantokäyttöön. Vastuun näkökulmasta takuuaika rajataan kuulumaan esittelyvaiheeseen. Evoluutiovaiheen määritellään viitekehyksen elinkaarimallissa alkavan sitä, kun ohjelmisto on asiakkaalla tuotantokäytössä, takuuaika on päättynyt ja ohjelmistossa on sopimuksen mukaiset toiminnallisuudet ja ne on myös testattu.

Evoluutiovaihe kestää niin kauan, kun ohjelmistolla on yksiselitteinen arkkitehtuuri ja kehitystiimi sekä ohjelmistoa ollaan valmiita kehittämään. Kun edellä mainitut ehdot eivät enää täyty, siirtyy ohjelmisto huoltovaiheeseen. Ohjelmiston käytön huippupiste ajoittuu evoluutio- ja huoltovaiheen rajalle.

Huoltovaiheessa ohjelmistoon tehdään muutoksia vähemmän kuin edellisissä vaiheissa. Kun ohjelmistoon ei tehdä enää ollenkaan muutoksia, se siirtyy vaiheittaisen käytöstäpoiston vaiheeseen ja lopulta lakkautusvaiheeseen. Huoltovaiheen ja vaiheittaisen käytöstäpoiston vaiheen välinen raja määritellään tutkielmassa siihen, kun ohjelmiston kehittäjäyritys tekee päätöksen, että ohjelmistoa ei enää ylläpidetä niin, että siihen tehtäisiin mitään muutoksia.

5.3 Muita näkökulmia vaatimustenhallintaan

Edellä arvioitiin vaatimustenhallintaa ohjelmiston elinkaaren vaiheen näkökulmasta. Elinkaari on kuitenkin vain yksi monista näkökulmista vaatimustenhallinnan tarkasteluun. Kuviossa 7 on mainittu muutamia muita huomioitavia näkökulmia.

KUVIO 7. Näkökulmia vaatimustenhallintaan

Gorschekin ja Wohlinin (2006) mukaan vaatimusten lähtökohtia tai näkökulmia voivat olla esimerkiksi laki, asiakkaat, kehittäjät, kilpailijat, kumppanit, jakelijat ja johto. Sommerville (1998, 131) puolestaan esittelee ei-toiminnallisia vaatimuksia jaotellessaan ulkoiset vaatimukset, joihin kuuluu lainsäädännölliset, yhteensopivuus ja eettiset vaatimukset, jotka ohjelmiston tulee täyttää. Eri lähteissä esitellään erilaisia vaatimusten jaotteluita ja näkökulmia, joista vaatimusmäärittelyä ja vaatimustenhallintaakin voi tarkastella. Koska tässä tutkielmassa painotetaan vaatimustenhallintaa elinkaaren näkökulmasta, käsitellään muita näkökulmia vain pintapuolisesti. Elinkaaren lisäksi viitekehykseen on valittu asiakkuuden ja ympäristötekijöiden näkökulmat. Viitekehykseen valitut näkökulmat ovat enemmänkin esimerkinomaisia kuin kattavia ja pääpaino on ohjelmiston elinkaaren näkökulmassa.

Muutospyynnön toteuttamiseen asiakkuuden näkökulmasta voi vaikuttaa esimerkiksi se, miten tärkeä ehdotettu muutos on asiakkaan liiketoiminnalle. Heindl ja Biffl (2005) määrittelevät vaatimuksen jäljittämisen tärkeyttä muun muassa sillä, miten tärkeänä asiakas, ja sidosryhmät yleensäkin, kokevat vaatimuksen. Tärkeät vaatimukset jäljitetään todennäköisemmin kuin vähemmän

kriittiset vaatimukset. Samaa voisi soveltaa myös muutospyyntöjen toteuttamiseen.

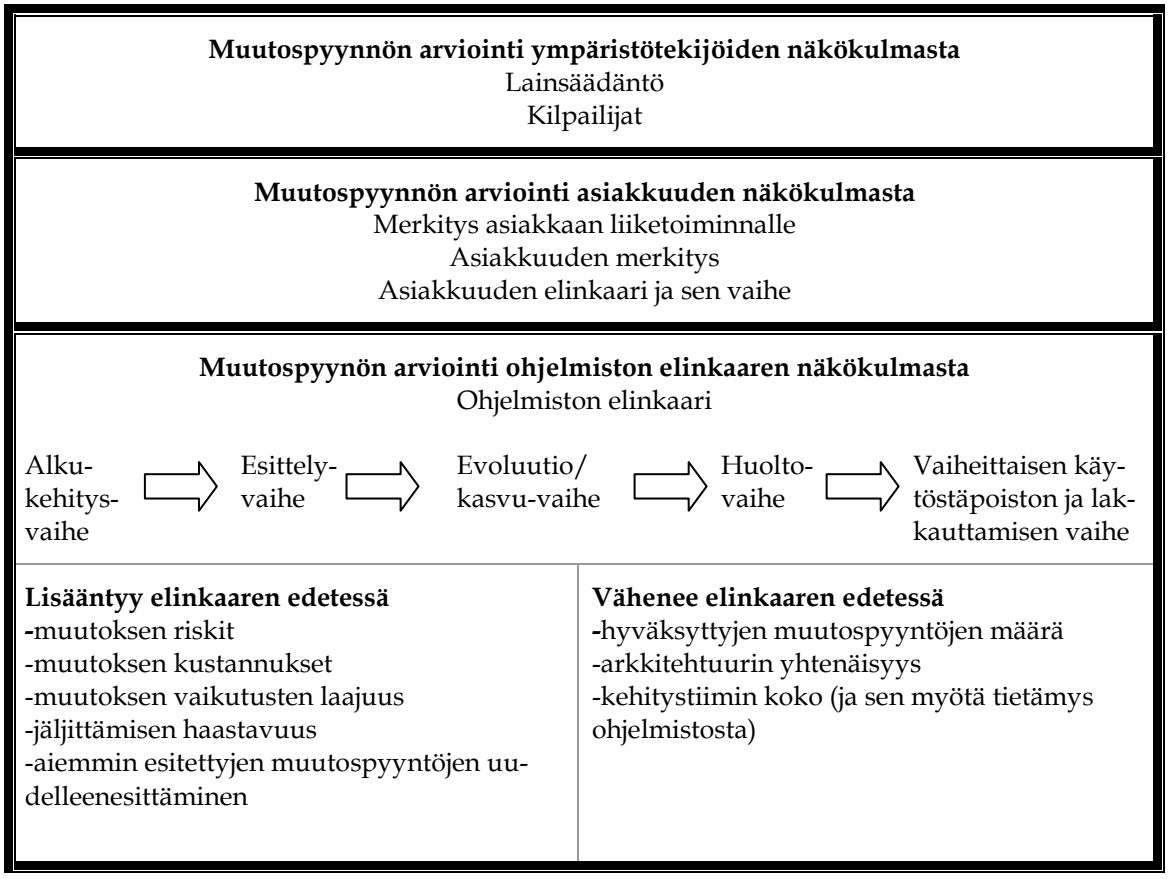
Asiakkuuden merkitys voi olla vaikuttava tekijä, kun arvioidaan toteutetaanko muutospyyntö vai ei. Myös asiakkuuden elinkaaren vaiheella voi olla merkitystä. Esimerkiksi strategisesti tärkeän asiakkaan muutospyyntöihin voidaan suhtautua muita suopeammin. Näin voi olla erityisesti silloin, jos strategisesti tärkeä asiakkuus on sellaisessa vaiheessa, jossa asiakas halutaan sitouttaa.

Muutospyynnön hyväksymiseen vaikuttaa myös ympäristötekijöitä. Esimerkiksi lainsäädännön muutokset voivat vaikuttaa siihen, hyväksytäänkö muutospyyntöä vai ei. Mikäli lainsäädäntö tai siihen tullut muutos edellyttää ohjelmistoon muutosta, muutospyynnön toteuttamisen painoarvo kasvaa. Näin siksi, että ohjelmistosta voisi tulla käyttökelvoton, mikäli muutosta ei tehtäisi, eikä ohjelmisto näin ollen täyttäisi lainsäädännön vaatimuksia. Lainsäädännön muuttumisella perusteltavia muutospyyntöjä voi olla tyypillisesti esimerkiksi verotukseen tai kirjanpitoon liittyvissä ohjelmistoissa.

Myös esimerkiksi markkina- ja kilpailutilanne voivat olla muutospyynnön toteuttamiseen vaikuttavia ympäristötekijöitä. Mikäli muutospyynnön toteuttaminen parantaa ohjelmiston kilpailukykyä merkittävästi, voi sillä olla vaikutusta muutospyynnön hyväksymiseen.

5.4 Viitekehys vaatimustenhallintaan muutospyyntöjen kautta tarkasteltuna

Viidennen luvun alussa on ensin koottu kirjallisuuskatsauksesta poimittuja vaatimustenhallintaan vaikuttavia tekijöitä. Ensin käsiteltiin vaatimustenhallintaan vaikuttavia tekijöitä, sen jälkeen koottiin elinkaarimalli ja käsiteltiin lyhyesti muita vaatimustenhallintaan vaikuttavia näkökulmia. Kuvioon 8 on edellä mainituista asioista koottu viitekehys, jossa tarkastellaan vaatimustenhallintaa muutospyyntöjen arvioinnin kautta.



KUVIO 8. Viitekehys vaatimustenhallintaan muutospyyntöjen arvioinnin kautta tarkasteltuna

Viitekehyksessä vaatimustenhallintaa tarkastellaan muutospyyntöjen arvioinnin kautta. Viitekehyksestä on jätetty pois muutospyynnön hyväksymisen jälkeiseen aikaan liittyvät tekijät. On kuitenkin huomioitava, että vaikka muutospyynnön hyväksymisen jälkeiset tekijät on rajattu viitekehyksestä pois, on vaatimustenhallinnalla keskeinen merkitys myös niissä.

Viitekehyksen muutospyyntöjen arviointiin liittyvät tekijät jaotellaan viitekehyksessä kahteen osaan: tekijöihin, jotka lisääntyvät elinkaaren edetessä ja tekijöihin, jotka vähenevät elinkaaren edetessä. Viitekehyksessä ohjelmiston elinkaaren näkökulma huomioidaan korostuneesti ja muita mahdollisia näkökulmia esitellään suppeasti, eikä niihin liittyen ole pyrittykään kattavuuteen. Elin-

kaaren ohella muita viitekehyksessä olevia näkökulmia ovat asiakkuus ja ympäristötekijät.

Viitekehyksen ei ole tarkoituskaan olla kattava, vaan sen on tarkoitus olla tukena ja ohjata ymmärtämään ja kehittämään vaatimustenhallintaa, erityisesti muutospyyntöjen arviointia systemaattisemmaksi. Seuraavassa kohdassa esitellään tarkemmin viitekehyksessä esitettyjen muutospyyntöjen arvioimiseen vaikuttavien tekijöiden soveltamista elinkaaren eri vaiheissa.

5.5 Viitekehyksen soveltaminen

Taulukossa 3 on esitetty viitekehyksessä havaittuja muutospyynnön arviointiin vaikuttavia tekijöitä ohjelmiston elinkaaren eri vaiheissa. Taulukossa 3 tehdyt arviot perustuvat kirjallisuuden perusteella tehtyihin tulkintoihin. Kutakin muutospyynnön arviointiin vaikuttavaa tekijää arvioidaan erikseen ohjelmiston eri elinkaaren vaiheessa. Viitekehyksessä mainittujen tekijöiden lisäksi taulukossa arvioidaan kirjallisuuteen pohjautuen ylläpidon eri tyyppien osuutta kussakin vaiheessa.

Taulukossa 3 kuvataan sitä, miten esimerkiksi hyväksyttyjen muutospyyntöjen määrä, muutosten kustannukset, riskit ja vaikutusten laajuus vaihtelevat ohjelmiston elinkaaren eri vaiheissa. Esimerkiksi muutoksen aiheuttamaa riskiä on taulukossa kuvattu alkukehitys- ja esittelyvaiheessa yhdellä tähdellä, evoluutiovaiheessa kahdella tähdellä ja huoltovaiheessa kolmella tähdellä. Sillä tarkoitetaan, että muutoksen aiheuttama riski on alkukehitys- ja esittelyvaiheessa pieni, evoluutiovaiheessa keskinkertainen ja huoltovaiheessa suuri. Taulukossa arvioidaan myös mukauttavan, täydentävän ja korjaavan ylläpidon osuutta elinkaaren eri vaiheissa.

TAULUKKO 3. Viitekehyksen soveltaminen ohjelmiston elinkaaren eri vaiheissa

Muutospyynnön arviointi ympäristötekijöiden näkökulmasta Lainsäädäntö Kilpailijat					
Muutospyynnön arviointi asiakkuuden näkökulmasta Merkitys asiakkaan liiketoiminnalle Asiakkuuden merkitys Asiakkuuden elinkaari ja sen vaihe					
Muutospyynön arviointi ohjelmiston elinkaaren näkökulmasta					
	Alkuke-hitys-vaihe	Esittely-vaihe	Evoluu-tio/ kasvu-vaihe	Huolto-vaihe	Vaiheittaisen käytöstäpois-ton ja lakkaut-tamisen vaihe
Hyväksyttyjen muutospyyntö-jen ja vaatimusmuutosten määrä	***	***	**	*	ei muutoksia
Muutoksen aiheuttamat kustan-nukset	*	*	**	***	ei muutoksia
Muutoksen aiheuttama riski	*	*	**	***	ei muutoksia
Muutoksen vaikutusten laajuus	*	**	**	***	ei muutoksia
Muutoksen jäljittämisen haasta-vuus	*	**	**	***	***
Aiemmin esitettyjen vaatimus-ten/muutospyyntöjen uudel-leenesittäminen	*	*	***	***	ei muutoksia
Arkkitehtuurin yhtenäisyys	***	***	**	*	*
Tietämys ohjelmistosta (esim. toimiva kehitystiimi)	***	***	**	*	*
Mukauttavan ylläpidon osuus (suhteutettuna koko ylläpidon määrään vaiheen aikana)	ei yllä-pitoa	*	**	**	ei muutoksia
Täydentävän ylläpidon osuus (suhteutettuna koko ylläpidon määrään vaiheen aikana)	ei yllä-pitoa	**	***	***	ei muutoksia
Korjaavan ylläpidon osuus (suh-teutettuna koko ylläpidon mää-rään vaiheen aikana)	ei yllä-pitoa	***	**	*	ei muutoksia

*** suuri ** keskimääräinen * pieni

Vaatimukset muuttuvat ja siitä johtuen esitetään muutospyyntöjä. Niin kuin muutostenhallintaa käsittelevässä luvussa todettiinkin, niin muutos on väistämätöntä ohjelmiston elinkaaren aikana. Myös Lehmanin ensimmäinen laki tukee jatkuvan muutoksen olemassaoloa (Lehman 1996). Vaatimukset muuttuvat ja tarkentuvat erityisesti alkukehitysvaiheessa, kun sekä asiakkaan että kehittäjien tietämys ohjelmistosta ja sen tarkoituksesta tarkentuvat. Vaatimustenhallintaa käsittelevän luvun yhtenä keskeisimmästä sisällöstä oli kuitenkin se, että vaatimukset muuttuvat myös käyttöönoton jälkeen. Esittelyvaiheessa käyttäjät oppivat käyttämään ohjelmistoa ja he saattavat havaita ohjelmistossa virheitä ja muutostarpeita oppiessaan yhä paremmin käyttämään ohjelmistoa.

Esittelyvaiheen lisäksi myös evoluutiovaiheessa tehdään vielä aktiivisesti muutoksia ohjelmistoon, mutta huoltovaiheessa hyväksyttyjen muutospyyntöjen määrä pyritään minimoimaan. Bennettin ja Rajlichin (2000a; 2000b) mukaan evoluutiovaiheen ja huoltovaiheen rajana onkin se, että huoltovaiheessa ohjelmistolla ei ole enää tarkoituksenmukaista arkkitehtuuria, eikä toimivaa kehitystiimiä. Tällöin muutosten määrä pyritään minimoimaan, koska muutosten toteuttaminen on hankalaa ja kallista (Bennett & Rajlich 2000b). Huoltovaiheessa tehdyllä muutoksella voi olla arvaamattoman laajat vaikutukset ja muutoksen toteuttaminen onkin riskialtista.

Kun siirrytään vaiheittaisen käytöstäpoiston vaiheeseen, ohjelmistoon ei tehdä enää muutoksia. Tehtyjen muutosten määrä siis vähenee, mitä pidemmälle ohjelmiston elinkaaren vaiheet etenevät. Myös vaatimusten jäljittämisen haastavuus lisääntyy, mitä pidemmällä ohjelmisto on elinkaarellaan. Tätä tukee se, että Lehmanin toisen lain mukaan ohjelmiston monimutkaisuus lisääntyy ajan myötä, kun ohjelmistoa kehitetään (Lehman 1996). Taloudellisesta näkökulmasta vaatimusten jäljittäminen on edullisempaa elinkaaren alkuvaiheessa. Heindlin ja Bifflin (2005) mukaan systemaattinen vaatimusten jäljittäminen on edullisempaa tehdä kehitysvaiheen aikana kuin silloin, kun ohjelmisto on jo käytössä. Vaatimusten jäljittäminen on kehitysvaiheessa edullisempaa siksi, että sil-

loin alkuperäiset järjestelmän kehittäjät ovat vielä mukana ohjelmiston kehittämisessä, kun taas käyttöönoton jälkeen he eivät välttämättä ole enää mukana (Heindl & Biffl 2005).

Niin kuin vaatimusten tilojen seurantaa käsittelevässä kohdassa 3.4 todettiin, hylätyt vaatimukset tulevat usein elinkaaren myöhemmissä vaiheissa uudestaan esille. Siksi hylätyt vaatimukset ja syy hylkäämiseen kannattaa kirjata (Wiegers 2003, 322–323). Tällöin vältetään muutospyynnön vaikutusten, kustannusten ja riskien arviointi uudelleen. Hylättyjen vaatimusten hylkäämisen syyn kirjaaminen helpottaa erityisesti, mikäli muutosehdotus esitetään uudelleen huoltovaiheessa. Syyn kirjaaminen auttaa huoltovaiheessa erityisesti sen vuoksi, että huoltovaiheessa ei enää välttämättä ole käytettävissä ohjelmiston syvällisesti tuntevia kehittäjiä.

Muutospyynnön toteuttamismahdollisuuksiin vaikuttaa merkittävästi se, ovatko edellytykset muutoksen tekemiselle kunnossa. Bennett ja Rajlich (2000a) mainitsevat, että yhtenäinen arkkitehtuuri ja kehitystiimin alkukehitysvaiheessa hankkima tietämys ohjelmistosta mahdollistavat muutosten tekemisen. Kun toinen tai molemmat häviävät, muutosten tekemisestä tulee vaikeaa. Siksi esimerkiksi huoltovaiheessa tehdään enää vain vähäisiä muutoksia ohjelmistoon.

Mukauttavassa ylläpidossa tehdään ohjelmistoon muutoksia, jotka johtuvat muutoksissa ohjelmiston ympäristössä. Täydentävässä ylläpidossa parannetaan toimintakykyä tai ylläpidettävyyttä. Täydentävään ylläpitoon kuuluvat käyttäjien uudet vaatimukset. Korjaava ylläpito on virheiden korjaamista. Ennaltaehkäisevässä ylläpidossa selvitetään ja korjataan piileviä vikoja ohjelmistosta. (Bennett & Rajlich 2000a; Pigoski 2002)

Lientzin ja Swansonin (1980, 68) mukaan noin 20 prosenttia ylläpitoon käytettävästä ajasta on korjaavaa ylläpitoa. Noin 25 prosenttia yläpidon ajasta on mukauttavaa ylläpitoa. Suurin osa, yli 50 prosenttia ylläpitoon käytettävästä ajasta on täydentävää ylläpitoa. Virheiden korjaaminen vie siis huomattavasti vä-

hemmän aikaa kuin ympäristön ja käyttäjien vaatimusten muutoksista aiheutuva ylläpito. On kuitenkin huomattava, että Lientzin ja Swansonin luokittelu ei sisällä ehkäisevää ylläpitoa. Ehkäisevä ylläpito on jätetty pois myös taulukosta 3 sen vuoksi, että sen arvioimiseen elinkaaren eri vaiheissa ei löytynyt tarpeeksi lähdekirjallisuutta. (Lientz & Swanson 1980, 68; Pigoski 2002)

Esittelyvaiheessa korjaavan ylläpidon määrä on merkittävä. Kasvu- ja evoluutiovaiheessa korjaavan ylläpidon rinnalle merkittäväksi ylläpidon muodoksi tulee mukauttava ylläpito. Kasvu- ja evoluutiovaiheessa käyttäjät ovat jo oppineet käyttämään ohjelmistoa ja alkavat ymmärtää ohjelmiston tuomia mahdollisuuksia. (Lehner 1991). Tällöin he esittävät myös uusia ominaisuuksia ja toiminnallisuuksia, joita haluaisivat ohjelmistossa olevan. Tätä tukee se, että Lientzin ja Swansonin (1980, 68) mukaan yli puolet ylläpitoon käytetystä ajasta on täydentävää ylläpitoa.

Seuraavaksi kuvaillaan esimerkki muutosprosessista, jossa viitekehystä voidaan soveltaa. Se esitetään muutostenhallintaa käsittelevässä kohdassa 3.2 esitetyn Wiegersin (2003, 335) muutosprosessin pohjalta muokattuna.

Esimerkki muutosprosessista:

1. Muutospyyntö esitetään.

2. Onko muutospyyntö tullut sovitulta taholta ja sovitun kanavan kautta (täyttyykö alkuehdot)? Jos ei, niin muutosprosessi keskeytyy.

3. Onko muutosehdotus esitetty ymmärrettävästi ja yksiselitteisesti (muutosehdotuksen laatu)? Jos ei, niin pyydetään asiakkaalta tarkennuksia tai tarkennetaan muutosedotusta yhdessä asiakkaan kanssa.

4. Tallennetaan muutosehdotus käytettävään järjestelmään ja määritellään sille tila.

5. Tunnistetaan ohjelmiston elinkaaren vaihe, asiakkuuden merkitys ja muut taustatekijät ja dokumentoidaan arviot.

6. Arvioidaan muutospyyntöä käyttäen apuna taulukkoa 3 ja dokumentoidaan arviot.

7. Jos muutospyyntöä ei hyväksytä, muutosprosessi keskeytyy (tällöin päivitetään muutospyynnön tilaksi hylätty). Jos muutospyyntö hyväksytään, siitä tulee uusi vaatimus. Määritellään vaatimuksen toteuttamiselle prioriteetti, toteutusaikataulu ja toteuttaja. Tallennetaan vaatimuksen tiedot ja tila.

8. Muutoksen toteutuksen ja testauksen aikana ja niiden jälkeen päivitetään tarvittavat muutokset dokumentteihin ja päivitetään vaatimuksen tila.

9. Varmennetaan tehty muutos.

10. Suunnitellaan ja päätetään, miten ja milloin muutos toimitetaan asiakkaalle.

Tässä luvussa esitelty viitekehys voi tukea muutospyyntöjen arviointia siten, että se auttaa tiedostamaan ohjelmiston elinkaaren vaiheesta riippuvia tekijöitä. Esimerkiksi elinkaaren vaiheesta riippuen muutos voi aiheuttaa erisuuruisia riskejä ja kustannuksia sekä vaikutusten laajuus voi vaihdella. Tässä tutkimuksessa koottuun viitekehykseen tukeutuminen muutospyyntöä arvioitaessa ei tee arvioinnista automaattisesti järjestelmällistä. Viitekehyksen tarkoitus on kuitenkin kannustaa systematisoimaan muutospyyntöjen arviointia ja kehittämään viitekehystä käyttökelpoisemmaksi siinä jo olevien näkökulmien osalta ja toisaalta täydentämään viitekehykseen kokonaan uusia näkökulmia, jotka arvioinnissa tulee huomioida. Siten viitekehyksestä voi tulla väline järjestelmällisempään muutostenhallintaan.

6 TAPAUSTUTKIMUKSEN TOTEUTTAMINEN

Tässä luvussa esitellään empiirisen osan tutkimusmenetelmä ja tiedonkeruutavat sekä kerrotaan tutkimuksen taustasta ja tavoitteista. Luvussa kuvataan myös lyhyesti tutkimuksen kohdeorganisaatiota ja kerrotaan siihen liittyvästä tutkimuksen rajauksesta. Tarkoituksena on kertoa, miten empiirinen osa on toteutettu sekä perustella ja arvioida siinä tehtyjä valintoja. Varsinaisia tutkimustuloksia käsitellään luvussa 7.

6.1 Tausta ja tavoitteet

Tapaustutkimuksen kohdeorganisaatio on Digian Samstock-tuotteisiin rajautuva osa. Tutkimuksen kohde on finanssialalla käytettävien Samstock Digia -ohjelmistojen vaatimustenhallinta.

Digia on tieto- ja viestintäteknologiaratkaisuja toimittava yritys, joka toimii useilla toimialoilla. Digian toimialoja ovat esimerkiksi telekommunikaatio-, kauppa-, teollisuus-, julkishallinto-, matkailu- ja finanssiala. Digia työllistää yli 1200 työntekijää ja sen liikevaihto oli vuonna 2007 105,8 miljoonaa euroa. (Digia 2008a)

Digia Samstock-tuotteet (myöhemmin Samstock-tuotteet tai -ohjelmistot) ovat muun muassa rahastojen, omaisuudenhoidon ja säilytyksen back office -järjestelmiä. Samstock-ohjelmistot ovat perustaltaan samanlaisia kaikilla asiakkailla, mutta ne räätälöidään tarvittaessa asiakaskohtaisesti. Samstock-ohjelmistojen asiakkaina on pohjoismaisia pankkeja, pankkiiriliikkeitä, rahastoyhtiöitä ja omaisuudenhoitajia.

Tutkimuksessa keskitytään Samstock-tuotteisiin ja muut Digian tuotteet rajataan pois tutkimuksesta. Samstock-tuotteista tutkitaan niiden vaatimustenhallintaa pääosin muutospyyntöjen kautta. Tapaustutkimuksessa selvitetään kohdeorganisaation vaatimustenhallinnan tilannetta ja kypsyystasoa arvioimalla

vaatimustenhallinnan vahvuuksia, heikkouksia, mahdollisuuksia ja uhkia. Samstock-tuotteiden vaatimustenhallinnan käytänteille on ollut merkitystä sillä, että Samstock on aiemmin ollut itsenäinen yritys, kunnes vuonna 2006 Digia (silloinen SysOpen Digia) osti sen. Yrityskaupan mukanaan tuomista ohjelmistokehitykseen vaikuttavista asioista yksi keskeisimmistä on mahdollisuus hyödyntää Digian OpenMethod-menetelmää sovelluskehityksessä ja OpenProject-menetelmää projektien hallinnassa.

Kirjallisuuskatsauksen perusteella on saatu viitteitä siitä, että elinkaaren vaiheella on merkitystä ainakin muutostenhallinnassa. Sen vuoksi tapaustutkimuksessa tutkitaan vaatimustenhallintaa nimenomaan muutospyyntöjen kautta ja selvitetään, onko elinkaaren vaiheella merkitystä muutospyyntöjen arvioinnissa ja muutostenhallinnassa yleisemminkin.

Samstock-tuotteiden vaatimustenhallintaan liittyen vertaillaan uudempien ja vanhempien, elinkaarensa eri vaiheissa olevien ohjelmistojen vaatimustenhallintaa keskenään. Käyttötarkoitukseltaan ne ovat kuitenkin pitkälti toisiaan vastaavia, vaikka esimerkiksi käyttöliittymät ja toimintojen suoritustavat eroavat toisistaan. Sekä uusiin että vanhoihin ohjelmistoihin kuuluu useampia eri finanssialan osa-alueeseen painottuvia ohjelmistoja. Jatkossa eri elinkaarensa vaiheessa oleviin Samstock-ohjelmistoihin viitataan termeillä uudet ja vanhat ohjelmistot.

Osasta uusista ja vanhoista ohjelmistoista on olemassa käyttötarkoitukselliset vastinparit. On siis olemassa uusi ohjelmisto, joka on käyttötarkoitukseltaan melko yhtenevä vanhan ohjelmiston kanssa. On kuitenkin huomattava, että Samstock-tuotteissa on olemassa myös ohjelmistoja, joille ei löydy vastinparia uusien ja vanhojen ohjelmistojen välillä. On sellaisia tapauksia, joissa vanhalle ohjelmistolle ei ole vastaavaa tuotetta uusista ohjelmistoista. On myös tapauksia, joissa uudelle ohjelmistolle ei ole vastaavaa tuotetta vanhoissa ohjelmistoissa.

Uusien ja vanhojen ohjelmistojen vertailun tavoitteena on selvittää, miten ohjelmiston elinkaaren vaihe vaikuttaa vaatimustenhallintaan. Tätä selvitetään pääasiassa muutospyyntöjen kautta vertailemalla sitä, miten kehityspyyntöihin suhtaudutaan uusissa ja vanhoissa ohjelmistoissa. Vertailun avulla voidaan selvittää, mitä eroja ja tyypillisiä piirteitä eri elinkaaren vaiheilla on ja miten erot voidaan huomioida muutospyyntöjä koskevissa päätöksissä. Se voi auttaa muutosten arvioinnin kehittämistä systemaattisemmaksi esimerkiksi siten, että muutospyyntöjen toteuttamisen arviointiin on mahdollista muodostaa eri elinkaaren vaiheille erilaisia päätöksentekoa tukevia listoja tai etenemismalleja. Tällöin muutospyyntöjen arvioinnista voi tulla järjestelmällisempää ja henkilöstä riippumattomampaa.

6.2 Tutkimusmenetelmän esittely

Empiirisen osan tutkimusmenetelmänä on tapaustutkimus, jossa tarkastellaan yhtä tapausta. Hirsjärven, Remeksen ja Sajavaaran (2004, 125) mukaan tapaustutkimuksesta saadaan yksityiskohtaista tietoa yksittäisestä tapauksesta tai pienestä joukosta toisiinsa suhteessa olevia tapauksia. Tapaustutkimuksen tyypillisiin piirteisiin kuuluu se, että kiinnostuksen kohteena ovat usein prosessit ja yksittäistapausta tutkitaan yhteydessä ympäristöönsä (luonnollisissa tilanteissa), joista yksittäistapaus on osa. Tapaustutkimukselle on tyypillistä kerätä aineistoa useita metodeja käyttämällä, esimerkiksi havainnoimalla, haastattelemalla ja dokumentteja tutkimalla. (Hirsjärvi ym. 2004, 125–126)

Tapaustutkimusta on kritisoitu yleistettävyyden puutteesta (esim. Järvinen & Järvinen 2000, 62; Metsämuuronen 2006, 92). Tapaus voi kuitenkin olla askel yleistämisessä (Metsämuuronen 2006, 92). Tapaustutkimuksen kritiikin huomioidenkin menetelmä sopii hyvin tähän tutkimukseen, koska yleistettävyys ja kattavuus eivät ole tutkimuksen tavoitteina. Tutkimuksen tavoitteena on pikemminkin kiinnittää huomiota vaatimustenhallinnan kehittämiseen systemaattisemmaksi ja saada viitteitä siitä, päteekö kirjallisuuskatsauksessa havai-

tut asiat käytännössä esimerkkitapauksessa. Lisäksi tapaustutkimuksen avulla pyritään myös mahdollisuuksien mukaan kartoittamaan, ilmeneekö tapaustutkimuksesta vielä muita keskeisiä piirteitä kuin kirjallisuuden perusteella havaitut. Tutkimuksessa pääasialliseksi lähestymistavaksi valittiin ohjelmiston elinkaaren näkökulma, mutta on huomioitava, että se on vain yksi monista näkökulmista, joista vaatimustenhallintaa voidaan tarkastella.

Niin kuin tapaustutkimuksessa yleensäkin, myös tässä tutkimuksessa tiedonkeruutapoja oli useita. Haastattelut olivat tutkimuksen keskeisin tiedonkeruutapa. Metsämuurosen (2006, 113) mukaan haastattelua voidaan pitää tarkkailun ohella perusmenetelmänä, joka soveltuu moneen tilanteeseen. Metsämuuronen (2006, 113) mainitsee tilanteita, joihin haastattelu sopii. Tässä tapaustutkimuksessa niistä pätee ainakin se, että halutaan tulkita kysymyksiä tai täsmentää vastauksia, kartoitetaan tutkittavaa aluetta ja halutaan kuvaavia esimerkkejä.

Haastattelujen lisäksi tietoa kerättiin tutustumalla myös esimerkiksi tuki- ja muutospyyntöjen tallennusjärjestelmään sekä yrityksen kehittämää OpenMethod-menetelmää käsittelevään koulutusmateriaaliin. Tutkimuksessa tutustuttiin myös kohdeorganisaatioon ja sen tuotteisiin. Tiedonkeruutapoja kuvaillaan tarkemmin seuraavassa kohdassa.

6.3 Tiedonkeruutavat

Tiedonkeruu suoritettiin kolmella menetelmällä:

1) Tutustuttiin kohdeorganisaation vaatimustenhallintaan liittyvään koulutusmateriaaliin sekä kohdeyritykseen ja sen tuotteisiin yleensä.

2) Tutustuttiin yrityksen järjestelmään, johon tallennetaan Samstock-tuotteisiin liittyvät tuki- ja muutospyynnöt.

3) Tehtiin haastatteluja kohdeorganisaatiossa (keskeisin tiedonkeruutapa).

Digia myy OpenMethod-menetelmän konsultointia asiakkailleen ja myös yrityksen työntekijöitä koulutetaan OpenMethod-menetelmän käyttöön. Yhtenä tiedonkeruutapana perehdyttiin OpenMethodin määrittelyä ja vaatimusmäärittelyä käsitteleviin osiin, erityisesti sellaisiin kohtiin, jotka sivuavat vaatimustenhallintaa. *Koulutusmateriaaliin tutustumisen* tavoitteena oli selvittää yrityksen vaatimustenhallinnan yleisiä linjoja ja kartoittaa haastateltavien taustatietoja vaatimustenhallinnasta. Lisäksi tavoitteena oli selvittää, missä määrin koulutusmateriaalin ja OpenMethodin vaatimustenhallintaa käsittelevät osat sisältävät yhtenevyyksiä tämän tutkimuksen kirjallisuuskatsauksen kanssa. OpenMethodin ja sen koulutusmateriaaliin tutustumisesta oli myös tavoitteena saada lisätietoa haastatteluja varten.

Toisena tiedonkeruutapana *tutustuttiin Samstock-tuotteiden tuki- ja muutospyyntöjen käsittely- ja tallennusjärjestelmään*. Järjestelmään tutustumisessa tutkittiin tuki- ja muutospyyntöjen tietojen tallentamista. Lisäksi selvitettiin myös esimerkiksi muutospyyntöjen jakautumista erityyppisiin muutospyyntöihin (kehityspyynnöt, järjestelmävirheet) sekä muutospyyntöjen etenemistä ja vaiheita.

Järjestelmään tutustumisen tavoitteena oli ymmärtää tutkimuskontekstia. Tutkimuskontekstin ymmärtäminen auttoi paitsi tekemään syvällisempiä haastatteluja, myös analysoimaan tuloksia luotettavammin.

Keskeisimpänä tiedonkeruutapana olivat *kohdeorganisaatiossa tehdyt haastattelut*. Haastatteluja tehtiin yhteensä 10. Haastattelut olivat puolistrukturoituja teemahaastatteluja lukuun ottamatta yhtä avointa haastattelua, jonka tarkoitus oli olla muiden haastatteluiden tulosten tulkinnan apuna. Haastateltavina oli sellaisia henkilöitä, joiden työtehtävissä vaatimustenhallinnalla (erityisesti ylläpidon aikana) on keskeinen merkitys.

Taulukkoon 4 on koottu tietoa haastateltavista ja haastateltavien valintaan vaikuttaneista asioista. Haastateltavia valittiin siten, että vaatimustenhallinnasta ja erityisesti muutostenhallinnasta saatiin mahdollisimman kattava näkemys.

Haastateltavien valinnassa auttoi kehityspäällikkö, joka vastaa kohdeorganisaation laatu- ja prosessiasioista. Koska tapaustutkimuksessa tarkastellaan vaatimustenhallintaa muutospyyntöjen kautta, valittiin haastateltaviksi useita henkilöitä, jotka osallistuvat muutospyyntöjä koskevaan päätöksentekoon. Muutospyyntöjä koskevaan päätöksentekoon osallistuvia henkilöitä valittiin siten, että saatiin haastateltavia, jotka arvioivat muutospyyntöä eri rooleista tai näkökulmista (ks. taulukko 4). Keskeinen kriteeri haastateltavia valitessa oli myös se, että haastateltaviksi valittiin sekä sellaisia henkilöitä, joiden tehtävät liittyvät uusiin ohjelmistoihin että sellaisia henkilöitä, joiden tehtävät liittyvät vanhoihin ohjelmistoihin. Uusien ja vanhojen ohjelmistojen vertailun näkökulmasta erityisen hyödyllisiksi koettiin haastateltavat, joiden työtehtävät liittyvät sekä vanhoihin että uusiin ohjelmistoihin.

Kaikkien haastattelujen tavoitteena oli kartoittaa kohdeorganisaation vaatimustenhallinnan kypsyystasoa. Lisäksi jokaiselta haastateltavalta pyrittiin saamaan tietoa haastateltavan työtehtäviin liittyvistä asioista, joita kuvataan taulukossa 4 kohdassa haastattelun painopiste. Sellaisia haastateltavia, joiden työtehtäviin liittyvät sekä vanhat että uudet ohjelmistot, pyydettiin vertailemaan uusien ja vanhojen ohjelmistojen vaatimustenhallintaa. Myös muita pyydettiin kertomaan työtehtävistä riippuen joko uusien tai vanhojen ohjelmistojen vaatimustenhallinnasta ja osaamisensa mukaan vertailemaan uusien ja vanhojen ohjelmistojen vaatimustenhallintaa.

Haastateltavina oli kaksi eri tuotteiden tuotepäällikköä ja neljä eri alueiden kehityspäällikköä sekä asiakasvastaava, asiakaspalvelun esimies, projektipäällikkö ja ohjelmistosuunnittelija. Kaikki haastattelut nauhoitettiin ja litteroitiin. Haastattelut kestivät noin 20 minuutista noin 55 minuuttiin. Litteroidut haastattelut annettiin haastatelluille luettaviksi ja kommentoitaviksi, jotta voitiin pienentää väärinymmärrysten riskiä.

TAULUKKO 4. Haastatteluiden painopisteet

Haastateltava	**Tehtävät liittyvät ohjelmistoihin:**		**Haastattelun painopiste**	**Rooli muutospyyntöön liittyvässä päätöksenteossa**
	vanhat	**uudet**		
Asiakaspalvelun esimies	X	X	– Uusien ja vanhojen ohjelmistojen muutostenhallinta (erityisesti järjestelmävirheet)	
Asiakasvastaava *	X	X	– Uusien ja vanhojen ohjelmistojen kehityspyyntöjen arviointi	Kommunikoi muutospyynnöstä asiakkaan kanssa ja tekee tarjouksen
Kehityspäällikkö 1*	X		– Vanhojen ohjelmistojen muutostenhallinta (sekä järjestelmävirheet että kehityspyynnöt)	Arvioi muutoksen teknistä toteuttamista ja resursseja
Kehityspäällikkö 2*		X	– Uuden rahastonhallintaohjelmiston muutostenhallinta (sekä järjestelmävirheet että kehityspyynnöt)	Arvioi muutoksen teknistä toteuttamista ja resursseja
Kehityspäällikkö 3*		X	– Uuden salkunhallintaohjelmiston muutostenhallinta (sekä järjestelmävirheet että kehityspyynnöt)	Arvioi muutoksen teknistä toteuttamista ja resursseja
Kehityspäällikkö 4	X	X	– Muutostenhallintaprosessin kulku ja kokonaiskuva organisaation vaatimustenhallinnasta – Muiden haastattelujen analysoinnin apu	
Ohjelmistosuunnittelija		X	– Uusien ohjelmistojen muutosten toteuttaminen käytännössä	
Projektipäällikkö*		X	– Muutostenhallinta projektin aikana	Arvioi muutoksen toteuttamista suhteessa projektin resursseihin ja sovittuihin vaatimuksiin, kommunikoi asiakkaan kanssa
Tuotepäällikkö 1*		X	– Kehityspyyntöjen arviointi tuotteen näkökulmasta uudessa rahastonhallintaohjelmistossa	Arvioi muutoksen sopivuutta ohjelmistotuotteen kannalta
Tuotepäällikkö 2*	X	X	– Kehityspyyntöjen arviointi tuotteen näkökulmasta sekä uudessa että vanhassa salkunhallintaohjelmistossa	Arvioi muutoksen sopivuutta ohjelmistotuotteen kannalta

*-merkki haastateltavan perässä kertoo, että haastateltava on muutospyyntöihin liittyen päätöksentekijä

Kahdelle tuotepäällikölle, kolmelle kehityspäällikölle ja asiakasvastaavalle tehtiin laajemmat haastattelut, joissa tutkittiin tarkemmin muutospyyntöjen arviointia erikseen uusien ja vanhojen ohjelmistojen osalta. Näitä laajempia haastatteluja oli siis yhteensä kuusi. Suppeampi haastattelu sisälsi muuten samat kysymykset kuin laajempi haastattelu, mutta siinä ei käsitelty muutospyyntöjen arviointia yhtä kattavasti kuin laajemmassa haastattelussa. Tosin myös suppeammissa haastatteluissa käsiteltiin kunkin haastateltavan osaa muutospyyntöprosessissa. Suppeampia haastatteluja tehtiin yhteensä kolme. Lisäksi yksi avoin haastattelu tehtiin sen jälkeen, kun muut haastattelut oli tehty ja jo osittain analysoitu. Avoimessa haastattelussa oli haastateltavana kehityspäällikkö, jonka työtehtäviin kuuluu laatu- ja prosessiasioiden kehittäminen. Avoimessa haastattelussa käytiin läpi aiempien haastattelujen tuloksia sekä pohdittiin ja analysoitiin niitä tarkemmin. Avoimen haastattelun toisena keskeisenä tarkoituksena oli myös havaita ja poistaa mahdolliset haastatteluiden alustavassa analysoinnissa tapahtuneet virhetulkinnat.

Haastattelurunko suunniteltiin kirjallisuuskatsauksen ja erityisesti siitä muodostetun viitekehyksen pohjalta. Haastattelussa keskeisimmässä osassa oli vaatimustenhallinnan, erityisesti muutospyyntöjen arvioinnin, merkitys ohjelmiston elinkaaren näkökulmasta. Haastatteluissa pyrittiin vertailemaan viitekehyksessä esitettyjä kohtia, jotka kirjallisuuden mukaan muuttuvat ohjelmiston elinkaaren edetessä. Tällaisia ovat esimerkiksi muutoksista aiheutuvat riskit ja laajuusvaikutukset.

Haastattelujen aluksi selvitettiin vaatimustenhallinnan käsitettä ja kuvailtiin lyhyesti siihen liittyviä keskeisiä toimintoja. Vaatimustenhallintaa määriteltäessä kuitenkin selvennettiin, että mainitut toiminnot eivät kata yksiselitteisesti koko vaatimustenhallinnan kokonaisuutta ja haastateltavia kannustettiin laajentamaan vaatimustenhallinnan käsitettä, mikäli he haastattelun aikana havaitsisivat jotain muita kuin alussa esitellyt vaatimustenhallinnan keskeiset toiminnot. Haastattelujen tarkoituksena ei ollut saada uusia määritelmiä vaatimustenhal-

linnan käsitteelle tai sisällölle. Toisaalta haastatteluissa ei kuitenkaan haluttu rajautua ennalta määriteltyihin vaatimustenhallinnan toimintoihin, mikäli haastateltava käsitti vaatimustenhallinnan laajemmin. Käsitteiden selvittämisellä pyrittiin vain takaamaan se, että haastateltava saa konkreettisen käsityksen siitä, mitä vaatimustenhallinta pitää ainakin sisällään.

Elinkaaren lisäksi vaatimustenhallintaa arvioitiin myös muista viitekehyksessä esitetyistä näkökulmista ja mahdollisuuksien mukaan etsittiin myös mahdollisia muita näkökulmia vaatimustenhallintaan. Viitekehyksessä painotettua ohjelmiston elinkaaren näkökulmaa pyrittiin sisällyttämään haastatteluihin erityisesti vertailemalla uusien ja vanhojen Samstock-ohjelmistojen vaatimustenhallintaa keskenään. Esimerkiksi kehityspäälliköistä haastateltiin kahta eri uusien ohjelmistojen kehityspäällikköä, jotka vastasivat kysymyksiin uudempien ohjelmistojen osalta vastuuohjelmistoihinsa painottuen. Lisäksi haastateltiin myös yhtä vanhoista ohjelmistoista vastaavaa kehityspäällikköä, jonka vastaukset pohjautuivat erityisesti vanhoihin ohjelmistoihin ja niiden vaatimustenhallintaan.

Vaikka kehityspäälliköt vastasivatkin pääosin omiin vastuualueisiinsa painottuen, vertailivat he myös kokemuksiensa mukaan uusien ja vanhojen ohjelmistojen yhtäläisyyksiä ja eroja vaatimustenhallinnassa. Lisänäkemystä saatiin myös tuotepäälliköiden haastatteluista. Tuotepäälliköiden osalta jako ei mennyt uuden ja vanhan ohjelmiston perusteella, vaan ohjelmistojen sisällön perusteella. Yhdellä tuotepäälliköllä on siis vastuualueenaan tiettyyn tarkoitukseen ja tietyntyyppisille asiakkaille tarkoitetut ohjelmistot, jolloin vastuualueeseen voi kuulua sekä uusia että vanhoja ohjelmistoja. Tämä auttoi osaltaan uusien ja vanhojen ohjelmistojen vertailussa. Sekä tuote- että kehityspäälliköillä on kokemusta sekä ylläpidosta että asiakasprojekteista.

Asiakaspalvelun toiminnasta ja laadusta vastaavalla henkilöllä on kokemusta sekä uusista että vanhoista ohjelmistoista, koska asiakaspalveluun tulevat yh-

teydenotot voivat liittyä ylläpidossa oleviin, sekä uusiin että vanhoihin ohjelmistoihin. Viitekehyksen elinkaarimallin alkuosaan painottuvasta vaatimustenhallinnasta saatiin tietoa projektipäällikköä ja ohjelmistosuunnittelijaa haastattelemalla. Lisäksi sekä haastatellulla projektipäälliköllä että ohjelmistosuunnittelijalla on myös kokemusta ylläpidon aikaisesta vaatimustenhallinnasta, koska projektipäällikkö toimii ylläpidon aikaisena asiakasvastaavana muutamalle asiakkaalle, joiden projektissa hän on ollut projektipäällikkönä. Myös ohjelmistosuunnittelija tekee alkukehitysvaiheeseen kuuluvan projektityöskentelyn lisäksi jossain määrin ylläpitoon liittyviä muutoksia ohjelmistoon. Tämän vuoksi hänen oli mahdollista vertailla elinkaaren alkukehitysvaiheessa ja ylläpidon aikana tehtyjen muutosten eroavuutta vaatimustenhallinnan kannalta.

Haastateltavat valittiin niin, että eri ohjelmistojen elinkaaren eri vaiheista saatiin mahdollisimman tarkka ja kattava käsitys. Haastattelun teemat painottuivat ylläpidon aikaan, mutta haastatteluissa käsiteltiin myös ylläpitoa edeltäviä asiakasprojekteja, joiden aikana ohjelmisto räätälöidään asiakkaan käyttöön ja tarpeisiin.

Ennen varsinaisen haastattelun alkua selvennettiin vaatimustenhallinnan sisällön ohella myös ohjelmiston elinkaaren käsitettä. Tutkimuksessa käytetty elinkaarimalli kuvailtiin lyhyesti ja kerrottiin kunkin vaiheen tyypilliset piirteet. Keskeisten käsitteiden havainnollistamiseen käytetyt kuviot ja koko haastattelusuunnitelma on liitteessä 1.

7 TAPAUSTUTKIMUKSEN TULOKSET

Tässä luvussa esitellään ja analysoidaan tapaustutkimuksen tuloksia. Ensin kuvaillaan Samstock-tuotteiden hallintaan liittyvää tuki- ja muutospyyntöjen tallennusjärjestelmää ja sen avulla saatuja tietoja vaatimustenhallinnasta ja erityisesti muutos- ja tukipyyntöjen dokumentoinnista kahden kohdeorganisaation asiakasyrityksen kohdalla. Sen jälkeen tutustutaan Digialla käytössä olevaan OpenMethod-menetelmään vaatimustenhallinnan näkökulmasta. Tuki- ja muutospyyntöjen tallennusjärjestelmä on näkyvimpiä välineitä kohdeorganisaation muutospyyntöjen käsittelyssä ja OpenMethod-menetelmällä on jo tällä hetkellä ja tulevaisuudessa vielä voimakkaammin merkittävä asema kohdeorganisaation sovelluskehitykseen liittyvien käytäntöjen yhtenäistäjänä. Sen vuoksi kumpaakin käsitellään omassa kohdassaan kohdeorganisaation vaatimustenhallinnan taustan selvittämisessä ennen vaatimustenhallinnan tilanteen arviointia.

Kohdeorganisaation vaatimustenhallinnan tilannetta havainnollistetaan SWOT-analyysin avulla. Sen jälkeen esitellään tapaustutkimuksessa havaittuja uusia näkökulmia ohjelmiston elinkaariajatteluun ja keskitytään myös tarkemmin siihen, millainen merkitys tutkimuksessa havaittiin elinkaarella ja sen vaiheella olevan vaatimustenhallintaan. Lopuksi kootaan vielä tapaustutkimuksessa saatujen tietojen perusteella muutostenhallinnan päätöspuita ja muokataan kirjallisuuskatsauksen pohjalta rakennettua viitekehystä tutkimuksessa havaittujen asioiden perusteella.

Koska tässä luvussa esitellään sekä haastattelujen tuloksia että niistä tehtyjä johtopäätöksiä, on suoraan haastatteluista saadut tiedot ja toisaalta niistä tehdyt johtopäätökset eroteltava toisistaan. Suoraan haastatteluista saatujen tietojen kohdalla mainitaan, että ne ovat tulleet ilmi haastatteluissa. Joissakin kohdissa mainitaan lisäksi, monessako haastattelussa kyseessä oleva aihe tuli esille. Kohdat 7.1 ja 7.2 perustuvat muihin tiedonkeruutapoihin kuin haastatteluihin. Koh-

ta 7.1 perustuu tuki- ja muutospyyntöjärjestelmästä saatuihin tietoihin ja kohta 7.2 OpenMethod-koulutusmateriaaliin ja -dokumentaatioon.

Suoraan haastatteluista saadut tiedot erotetaan johtopäätöksistä niin, että kohdissa esitellään ensin haastatteluista ilmenneet asiat ja niiden jälkeen mainitaan alkavan johtopäätökset, jotka jatkuvat kunkin kohdan loppuun. Viimeiset kohdat sisältävät kuitenkin miltei kokonaan johtopäätöksiä, joten niissä mainitaan johtopäätösten alkavan miltei luvun alusta. Lisäksi lopuksi kohdassa 7.10 kootaan edellisten kohtien johtopäätösten keskeisimmät asiat yhteen ja pohditaan niiden merkitystä koko tutkimuksen kannalta.

7.1 Kohdeorganisaation tuki- ja muutospyyntöjärjestelmä

Samstock-tuotteisiin liittyvät tuki- ja muutospyynnöt tallennetaan yhteen järjestelmään. Asiakkaan yhteydenotot jaotellaan tukipyyntöihin, järjestelmävirheisiin ja kehityspyyntöihin. Asiakkaiden yhteydenotot, jotka liittyvät joko kehityspyyntöihin tai järjestelmävirheisiin, ovat muutospyyntöjä. Joissain tapauksissa asiakkaan yhteydenotto voidaan jaotella myös projektipyynnöksi. Projektipyyntöjä ei kuitenkaan tarkastella tässä tutkimuksessa.

Suurin osa asiakkaiden yhteydenotoista on tukipyyntöjä, joissa pyydetään neuvoja Samstock-tuotteiden käyttöön tai selvitystä siitä, miten Samstock-tuotteissa on toteutettu jokin toiminto. Tyypillinen esimerkki asiakkaalta tulleesta tukipyynnöstä voi olla esimerkiksi sellainen, että asiakas on saanut omissa laskelmissaan erilaisen tuloksen vaikkapa koron tai tuoton laskennasta jossain erityistilanteessa ja sen vuoksi tiedustelee, miten ohjelmisto koron tai tuoton laskee. Toinen tyypillinen esimerkki on sellainen, että asiakas on tehnyt arvopaperille yhtiötapahtuman (esimerkiksi osingonmaksun, rahastoannin tai splitin) ja huomaa jälkikäteen sen menneen väärin. Tällöin asiakas voi tiedustella, miten yhtiötapahtuman voi peruuttaa ja tehdä uudelleen niin, ettei siitä aiheudu virheellisiä tilanteita omistuksissa tai kirjanpidossa. Tukipyynnöt ovat sellaisia asiak-

kaan yhteydenottoja, jotka vaativat selvittämistä tai opastamista, mutta eivät vaadi ohjelmakoodin muuttamista.

Oleellista tutkimuksen kannalta on se, että tukipyynnöt eivät aiheuta muutoksia ohjelmistoon. Toki on kuitenkin huomattava, että useimmat asiakkailta tulevat yhteydenotot ovat aluksi tukipyyntöjä, mutta niiden luokittelu voi myöhemmin muuttua järjestelmävirheeksi tai kehityspyynnöksi, mikäli paljastuu, että ohjelmistoon tarvitaan muutoksia. Lisäksi tukipyyntöjen selvittämisen kannalta vaatimustenhallinnalla on hyvin keskeinen merkitys. Mikäli ohjelmistosta on olemassa ajan tasalla olevia vaatimusmäärittely- ja testausdokumentteja, toiminnallisuuksien syitä toimia tietyllä tavalla on helpompi jäljittää ja perustella asiakkaalle ohjelmiston toimintaa.

Järjestelmävirheiksi luokitellaan ohjelmistossa olevat virheet, jotka korjataan ylläpidon puitteissa ilman erillistä maksua. Kehityspyynnöiksi luokitellaan muutospyynnöt, jotka laajentavat tai muuttavat ohjelmiston toimintaa. Olennaista kuitenkin on, että kehityspyynnöiksi luokiteltavat muutospyynnöt voivat olla asiakkaalle maksullisia, mikäli kehityspyyntöä ei luokitella niin tärkeäksi ja yleishyödylliseksi, että se halutaan toteuttaa uuden ohjelmistoversion kehitystyönä. Kun tässä tutkimuksessa mainitaan yleisesti muutospyynnöt, tarkoitetaan niillä sekä järjestelmävirheisiin liittyviä yhteydenottoja että kehityspyyntöjä.

Järjestelmävirheiden ja kehityspyyntöjen välinen ero ei ole aina selkeä. Tulkinnanvaraisuutta on esimerkiksi silloin, kun kyse ei ole varsinaisesta ohjelmointivirheestä, vaan siitä, että epätarkkojen vaatimusmäärittelyiden vuoksi jokin toiminnallisuus on toteutettu eri tavalla kuin asiakas olisi halunnut. Toiminnallisuus voi olla sinänsä toteutettu ihan oikein, mutta perustuen epätarkkoihin tai virheellisiin vaatimuksiin.

Tuki- ja muutospyyntöjen tallennusjärjestelmään kirjataan pyyntöjen perustiedot, kuten pyynnön esittäjä ja ajankohta sekä pyynnön tila ja kuvaus pyynnös-

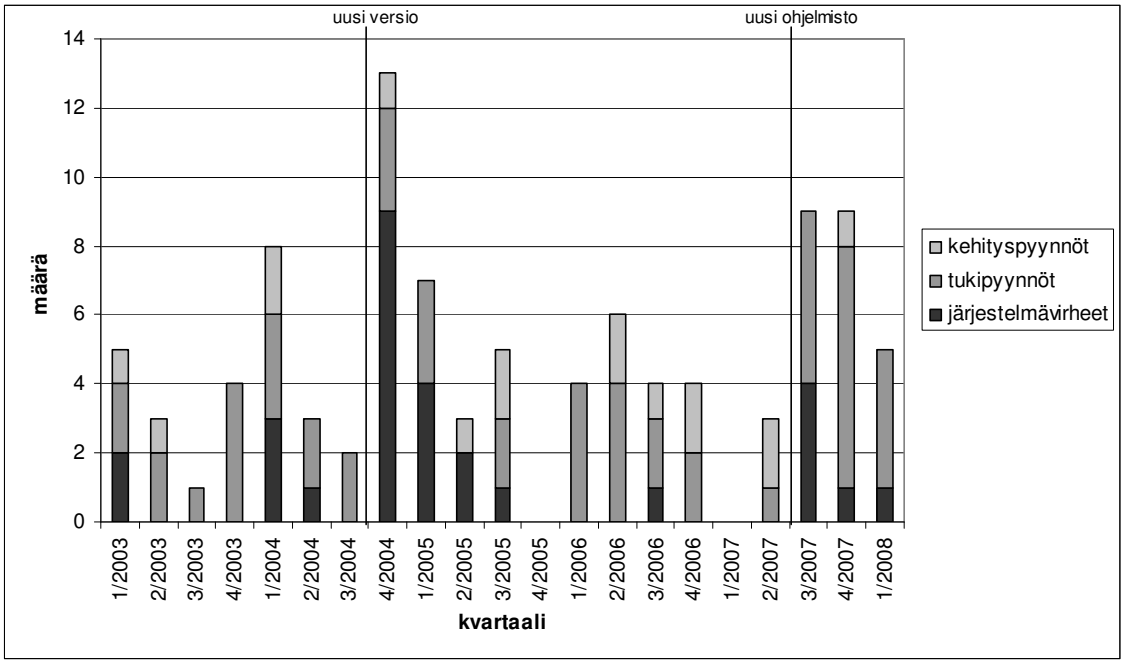
tä. Kunkin pyynnön yhteyteen kirjataan asiasta käyty keskustelu (esimerkiksi sähköpostit) ja tilanteesta riippuen esimerkiksi aiheeseen liittyviä dokumentteja. Vanhoista jo selvitetyistä pyynnöistä voi saada apua uusien vastaavien tapausten selvittämiseen.

Kuvioon 9 on koottu asiakasyritys A:sta ja kuvioon 10 asiakasyritys B:stä saadut tiedot rahastojenhallintaohjelmistoon liittyvistä tuki- ja muutospyynöistä vuodesta 2003 vuoden 2008 ensimmäiseen neljännekseen. On kuitenkin huomattava, että kuvion 9 tuki- ja muutospyynnöt liittyvät vain yhteen osa-alueeseen asiakkaalla olevista useista Samstock-tuotteista. Kyseessä on rahastonhallintaohjelmisto, josta asiakkaalla on ollut aluksi käytössä vanha ohjelmisto ja myöhemmin se on korvattu uudella, samaan tarkoitukseen suunnitellulla ohjelmistolla. Rahastonhallintaohjelmisto on valittu sen vuoksi, koska asiakkaalla on pitkä historia ohjelmiston käytöstä ja sen vuoksi asiakkaan yhteydenottoihin liittyviä tietoja on saatavilla pitkältä aikaväliltä.

Kun kuvioihin 9 ja 10 liittyen mainitaan elinkaaren vaiheita, tarkoitetaan ohjelmiston tai ohjelmistoversion elinkaaren vaihetta nimenomaan asiakkaan näkökulmasta. Ohjelmisto saattaa olla itsenäisenä tuotteena jo esimerkiksi evoluutiovaiheessa, kun se räätälöidään asiakkaan käyttöön. Esimerkiksi kun asiakas on ottamassa käyttöön tarpeisiinsa räätälöityä ohjelmistoa, sen käsitetään silloin olevan asiakkaan näkökulmasta esittelyvaiheessa, vaikka varsinainen ydintuote olisikin ohjelmiston näkökulmasta jo evoluutiovaiheessa.

On myös huomioitava, että kuvioiden 9 ja 10 asiakkaiden yhteydenottojen jaottelu tukipyyntöihin, kehityspyyntöihin ja järjestelmävirheisiin perustuu yhteydenottojen läpikäymisen perusteella tehtyyn tulkintaan, koska kaikissa tapauksissa jaottelu ei ole yksiselitteinen. Tulkintaa on käytetty myös kuvion 9 osalta siinä, kun on arvioitu, kuuluuko asiakkaan yhteydenotto valittuun ohjelmistoon vai johonkin muuhun ohjelmistoon. Asiakasyritys A:lla on ollut käytössä yhtä aikaa useita Samstock-ohjelmistoja, joissa on yhteneviä toiminnallisuuksia

esimerkiksi käyttäjienhallintaan liittyen. Epäselvissä tapauksissa on pyritty arvioimaan asiayhteyden perusteella, mihin ohjelmistoon asiakkaan yhteydenotto todellisuudessa liittyy. On myös huomioitava, että kuviot 9 ja 10 eivät kuvaa asiakkuutta alusta asti, vaan aloitusajankohdaksi on valittu vuoden 2003 ensimmäinen kvartaali, koska kattavia tietoja asiakkaiden yhteydenotoista ei ole tallennusjärjestelmästä saatavilla asiakkuuden alusta saakka.



KUVIO 9. Asiakasyritys A:n muutos- ja tukipyyntöjen määrät kvartaaleittain

Asiakasyritys A on käyttänyt vuoden 2007 kolmanteen kvartaaliin asti vanhaa ohjelmistoa ja siirtynyt sen jälkeen käyttämään uutta ohjelmistoa. Kuviossa 9 on kaksi selkeää piikkiä. Vuoden 2004 neljännen kvartaalin aikana tukipyyntöjä ja virheraportteja on tullut tavallista enemmän. Samanlainen tilanne on myös vuoden 2007 kolmannessa ja neljännessä kvartaalissa.

Vuoden 2007 kolmannen ja neljännen kvartaalin kohdalla oleva piikki selittyy sillä, että asiakasyritys A on siirtynyt silloin vanhan rahastojenhallintaohjelmiston käytöstä uuden rahastojenhallintaohjelmiston käyttöön. Ohjelmisto on ollut

vuoden 2007 kolmannen ja neljännen kvartaalin aikana asiakkaan näkökulmasta esittelyvaiheessa, siirtymässä evoluutiovaiheeseen, jolloin asiakas on toisaalta tarvinnut tavallista enemmän tukea ohjelmiston käyttöön ja toisaalta havainnut uudesta ohjelmistosta asioita, jotka eivät toimi halutulla tavalla tai vaatimusten mukaisesti.

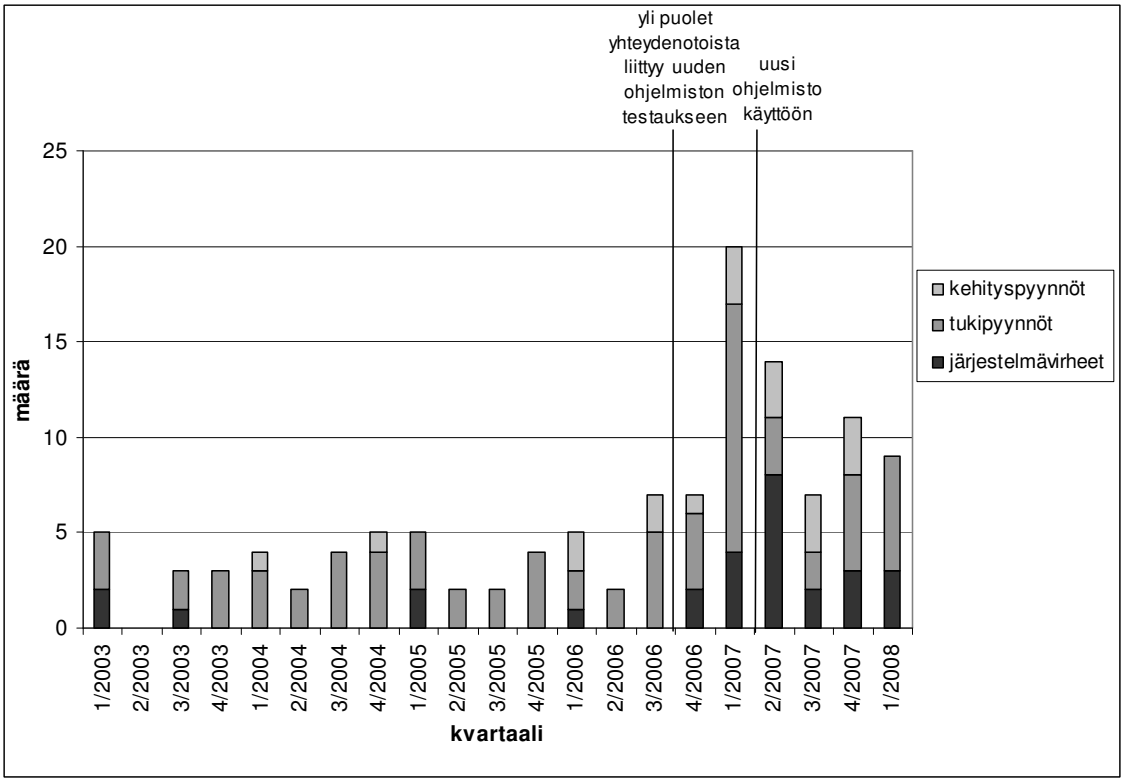
Vuosina 2004 ja 2005 on ollut ensimmäisessä kvartaalissa hieman keskimääräistä enemmän yhteydenottoja. Yhteydenotot liittyvät enimmäkseen viranomaisraportointiin, joka suoritetaan alkuvuodesta. Myös alkuvuodesta 2003 ja 2008 on yhteydenottoja tullut lievästi keskimääräistä enemmän. Myös niihin heijastuu alkuvuoteen painottuva viranomaisraportointi.

Vuoden 2004 neljänteen kvartaaliin sijoittuva piikki selittyy puolestaan sillä, että asiakasyritys A on ottanut käyttöön silloin vanhasta ohjelmistosta uuden version. Tällöin on myös havaittu tavallista enemmän virhetilanteita ja toisaalta tarvittu ohjelmiston käyttöön tavallista enemmän opastusta. Tällöin ohjelmistoversio on ollut asiakkaan näkökulmasta esittelyvaiheessa ja siirtymässä evoluutiovaiheeseen. Tapaustutkimuksessa havaittuja erilaisia näkökulmia ja laajennuksia (esimerkiksi ohjelmistoversion elinkaari) käsitellään tarkemmin kohdassa 7.6.

Kuviossa 10 on kuvattu asiakasyritys B:ltä tulleita yhteydenottoja. Asiakasyritys B:llä on ollut vuoden 2007 toiseen kvartaaliin asti käytössä vanha rahastonhallintaohjelmisto ja toisen kvartaalin aikana yritys B on ottanut tuotantokäyttöön uuden rahastonhallintaohjelmiston.

Keskeisenä erona asiakasyritys A:han on se, että B:llä on ollut uusi rahastonhallintaohjelmisto huomattavan pitkään testikäytössä ennen varsinaista tuotantokäyttöön ottamista. Suurin osa asiakasyritys B:n yhteydenotoista on liittynyt uuteen ohjelmistoon vuoden 2006 neljännestä kvartaalista alkaen. Myös vuoden 2006 kolmannen kvartaalin aikana ja sitä ennenkin on tullut muutamia uuteen ohjelmistoon liittyviä yhteydenottoja. Aktiivisen testaamisen aloittamisesta oh-

jelmiston varsinaiseen tuotantokäyttöönottoon on kulunut yli puoli vuotta. Tuona aikana asiakkaan yhteydenottojen määrä on kasvanut huomattavasti verrattuna aikaan ennen kuin uutta ohjelmistoa on alettu testaamaan. Testaamisen aikana erityisesti järjestelmävirheistä on raportoitu huomattavan paljon tavallista enemmän, mutta myös tukipyyntöjä ja kehityspyyntöjä on esitetty keskimääräistä enemmän.



KUVIO 10. Asiakasyritys B:n muutos- ja tukipyyntöjen määrät kvartaaleittain

Asiakasyritys A:n ja B:n välillä on ero siinä, missä uutta ohjelmistoa otettaessa käyttöön on tullut yhteydenottojen huippupiste. A:n kohdalla yhteydenottojen määrät ovat nousseet uuden ohjelmiston käyttöönoton aikaan ja sen jälkeisenä kvartaalina. B:n kohdalla puolestaan yhteydenottojen huippupiste on vuoden 2007 ensimmäisen kvartaalin kohdalla, joka on käyttöönottokvartaalia edeltävä kvartaali. Kuitenkin molemmissa tapauksissa yhteydenottojen määrän huippukohta näyttäisi olevan esittelyvaiheessa tai esittely- ja evoluutiovaiheen vaih-

teessa. Koska sekä asiakasyritys A:n että asiakasyritys B:n tapauksessa uuden ohjelmiston käyttöönotto on tapahtunut kohtuullisen vähän aikaa sitten, on vielä vaikea päätellä, tasaantuuko asiakkaiden yhteydenottojen määrä jossain vaiheessa samalle tasolle kuin ennen uuteen ohjelmistoon siirtymistä.

Johtopäätöksenä kohdeorganisaation tuki- ja muutospyyntöjen tallennusjärjestelmään tutustumisesta voidaan todeta, että kohdeorganisaatiossa luokitellaan asiakkailta tulevat tuki- ja muutospyynnöt, mutta luokittelu ei ole yksiselitteistä. Luokittelua voi vaikeuttaa esimerkiksi se, jos vaatimusmäärittelyä ei ole tehty ja dokumentoitu yksiselitteisesti. Tällöin ei voida tukeutua vaatimusmäärittelydokumenttiin tai testausdokumentteihin epäselvissä tapauksissa.

Esimerkkiasiakkaiden tukipyyntöjen tarkasteleminen (kuviot 9 ja 10) antaa viitteitä siitä, että ainakin kahdessa esimerkkitapauksessa elinkaaren vaihe asiakkaan näkökulmasta tarkasteltuna vaikuttaa asiakkaalta tulleiden yhteydenottojen määrään. Kun asiakas on ottanut käyttöön uuden version tai kokonaan uuden ohjelmiston, on asiakkaan yhteydenottojen määrä kasvanut.

Esimerkkitapauksissa tehdyt havainnot tukevat myös esimerkiksi vaatimustenhallinnan viitekehykseen liittyvässä taulukossa 3 esitettyä mainintaa siitä, että korjaavan ylläpidon määrä on suuri esittelyvaiheessa. Tässä tapauksessa korjaavaan ylläpitoon lukeutuvat asiakkaiden järjestelmävirheisiin liittyvät yhteydenotot, ja niiden osuus on selkeästi koholla verrattuna kehityspyyntöihin. Järjestelmävirheisiin liittyvät yhteydenotot kuuluvat korjaavaan ylläpitoon ja kehityspyynnöt liittyvät pääasiassa täydentävään ylläpitoon, mutta on kuitenkin huomioitava, että kohdeorganisaation muutospyyntöjen jaottelu ei ole yhtenevä taulukossa 3 käytetyn Lientzin ja Swansonin (1980, 68) jaottelun kanssa.

Esimerkkitapaus tukee sitä, että elinkaaren vaiheella on merkitystä ainakin tuki- ja muutospyyntöjen määriin ja näin ollen elinkaaren näkökulma saattaisi olla yksi mahdollinen näkökulma vaatimustenhallinnassa ainakin muutospyyntöjen arvioinnin ja käsittelyn kannalta. On kuitenkin huomioitava, että yhteen

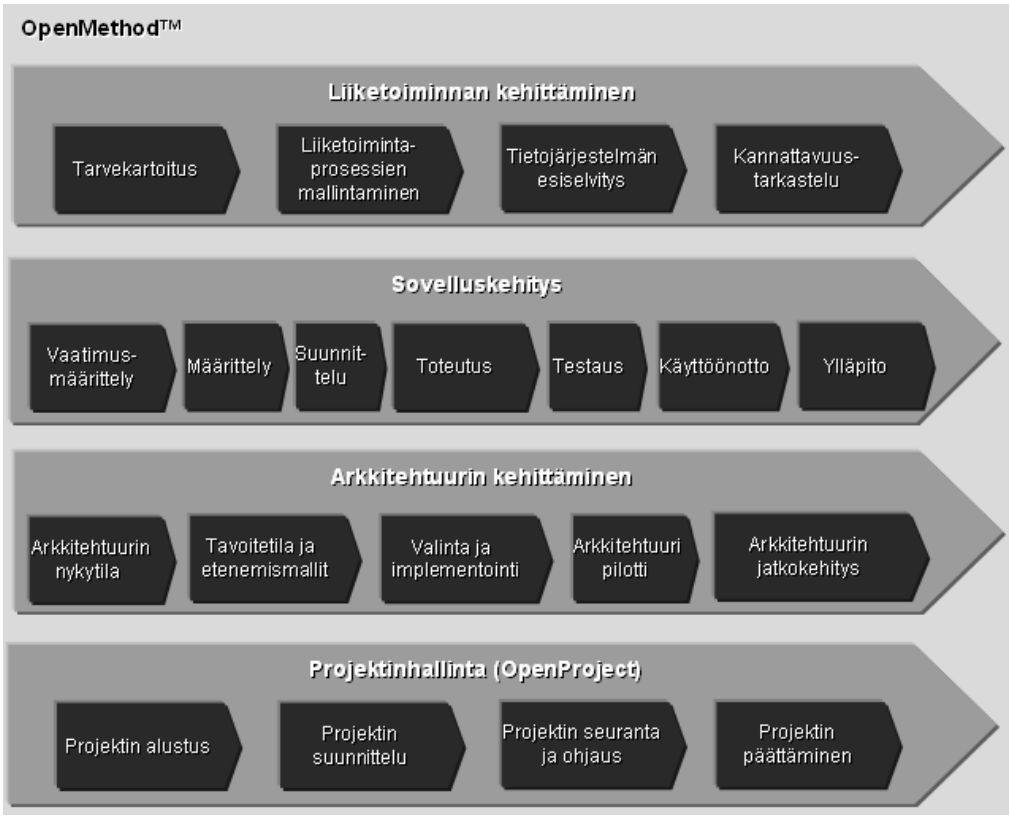
kohdeorganisaatioon liittyvä kahden asiakasyritysten tuki- ja muutospyyntöjen määrien tutkiminen ei ole yleistettävissä laajemmin. Lisäksi jotta voitaisiin arvioida todellista muutospyyntöjen arvioinnin ja päätöksenteon vaihtelua eri vaiheissa, tulisi tutkia sitä, miten asiakkaan raportoimiin järjestelmävirheisiin ja kehityspyyntöihin liittyviin yhteydenottoihin reagoidaan eri vaiheissa. Tämän tutkielman puitteissa muutospyyntöihin reagointiin ei ole kuitenkaan ollut mahdollisuutta syventyä.

7.2 OpenMethod-menetelmä vaatimustenhallinnan näkökulmasta

OpenMethod on Digian käyttämä ja myös asiakkaille myytävä kehittämismenetelmä, joka on koottu niin sanotuista parhaista käytännöistä (best practices). Sen voidaan ajatella olevan systeemityömalli, joka sisältää prosessikuvauksia, käytännönläheisiä ohjeita, dokumenttipohjia, tarkistuslistoja ja esimerkkejä. OpenMethod-menetelmä on tarkoitettu sekä tietojärjestelmähankkeita toteuttaville että tietojärjestelmiä ja tietojärjestelmähankkeita ostaville organisaatioille. (Digia 2008b)

OpenMethodin osa-alueet ovat liiketoiminnan kehittäminen, sovelluskehitys, arkkitehtuurin kehittäminen ja lisäksi osittain itsenäisenä osana OpenProject projektinhallintaan. Kuviossa 11 on kuvattu Digian kuvion perusteella OpenMethodin osa-alueet ja niiden vaiheet. Tässä tutkimuksessa tarkastellaan tarkemmin vain OpenMethodin sovelluskehityksen osa-aluetta, joka liittyy vaatimusmäärittelyyn ja erityisesti vaatimustenhallintaan.

OpenMethodin vaatimusmäärittelyä käsittelevässä koulutusmateriaalissa kuvataan vaatimustenhallinta systemaattiseksi menettelyksi, jolla liiketoiminnan tai järjestelmän muuttuvia vaatimuksia löydetään, dokumentoidaan, organisoidaan, jäljitetään, hyväksytään ja hallitaan. Vaatimustenhallinnan perustana on se, että kuvataan liiketoiminnan keskeiset käsitteet niin, että kaikki sidosryhmät ymmärtävät asian samalla tavalla.



KUVIO 11. OpenMethod-menetelmän osa-alueet (Digian kuvio)

Koulutusmateriaalissa käsitellään vaatimustenhallinnan tarkoitusta ja määritellään tehokkaaseen vaatimustenhallintaan kuuluvaksi seuraavia asioita:

1. Ylläpidetään ajan tasalla olevaa luettelo hyväksytyistä vaatimuksista.

2. Jokainen uusi vaatimus käsitellään ja priorisoidaan.

3. Eri sidosryhmät ymmärtävät vaatimukset samalla tavalla.

4. Pystytään tarkistamaan, että kaikki vaatimukset on huomioitu.

5. Tuote sisältää vain vaatimusten mukaiset piirteet ja vain ne.

6. Vaatimuksen muuttamisen vaikutukset selvitetään ja ennakoidaan.

OpenMethod-koulutusmateriaalissa määriteltyyn tehokkaaseen vaatimustenhallintaan kuuluville asioille on mahdollista havaita perustelu kirjallisuuskatsaukseen perustuvista luvuista 2 ja 3. Listan kohta 1 liittyy vaatimusmäärittelydokumenttien päivittämiseen, jonka merkitystä Wiegerskin (2003, 317–319) korostaa. Kohta 2 liittyy muutostenhallintaan ja esimerkiksi kuviossa 2 käsiteltyyn muutosprosessiin. Kohtaan 3 liittyviä asioita käsiteltiin kohdassa 2.2, kun kuvailtiin vaatimusmäärittelyn haasteita. Cheng ja Atlee (2007) toteavat muun muassa, että vaatimusmäärittelyn haastavuutta lisää se, että kaikilla sidosryhmien edustajilla ei ole teknistä taustaa ja ajattelutapaa, tästä huolimatta vaatimuksista tulee saada aikaan yksityiskohtaisia ja yksiselitteisiä, jotta ohjelmisto voidaan toteuttaa niiden pohjalta.

Listan kohdat 4 ja 5 liittyvät vaatimusten jäljittämiseen, jota käsiteltiin kohdassa 3.5. Kohdan 4 vaatimus siitä, että pystytään tarkistamaan kaikkien vaatimusten huomioiminen, pystytään varmentamaan eteenpäin suuntautuvan jälkijäljitettävyyden avulla (Jarke 1998; Wiegers 2003, 354–355). Kohdan 5 mukaan tulee varmistaa, että tuote sisältää vaatimusten mukaiset piirteet ja vain ne. Tämä liittyy taaksepäin suuntautuvaan jälkijäljitettävyyteen (Jarke 1998; Wiegers 2003, 354–355).

Listan kohta 6 puolestaan liittyy muutostenhallinnan vaikutusanalyysiin. Vaikutusanalyysissa pyritään tunnistamaan komponentit, joita pitäisi tehdä, muokata tai poistaa, mikäli vaatimusmuutos hyväksytään (Wiegers 2003, 345).

OpenMethod-vaatimusmäärittelyn koulutusmateriaalissa esitellään vaatimustenhallintaa, mutta se painottuu pääosin asiakasprojektin aikaan. Ylläpidon aikaiseen vaatimustenhallintaan on kiinnitetty vähemmän huomiota, vaikka sovelluskehitysprosessissa on kuvattu oma ylläpito-aliprosessi. Vaatimustenhallintaan liittyviä tehtäviä, jotka ovat ajankohtaisia myös ylläpitovaiheessa vaatimusten muuttuessa, ovat ainakin vaatimusten analysointi, luokittelu ja jäljitysketjujen tekeminen, ylläpidon aikana painottuen kuitenkin vaatimusmuutok-

siin. Myös vaatimusluettelon ylläpitäminen on keskeistä myös projektin jälkeen ylläpitovaiheessa. OpenMethod luokittelee myös vaatimusten katselmoinnit yhdeksi vaatimustenhallinnan keskeiseksi tehtäväksi.

Useimmille OpenMethodin vaatimusmäärittelyn koulutusmateriaalissa esitetyille vaatimustenhallintaan liittyville tehtäville ja tavoitteille on havaittavissa perustelut kirjallisuuskatsaukseen perustuvista luvuista 2 ja 3. Koulutusmateriaalissa on keskitytty vaatimustenhallintaan projektin aikana, mutta siitä huolimatta monet keskeiset tehtävät, kuten vaatimusten jäljittäminen, on huomioitu.

OpenMethodin mukaiset käytännöt eivät toteudu vielä täysimääräisesti Samstock-tuotteiden vaatimusmäärittelyssä ja vaatimustenhallinnassa. OpenMethodin käyttöön siirrytään pikku hiljaa. Esimerkiksi uudet projektit pyritään jo tekemään mahdollisimman tarkasti OpenMethodin mukaisesti. Kuitenkaan uusissakaan projekteissa esimerkiksi vaatimusten jäljittämistä ei ole otettu kovin aktiivisesti käyttöön. Tämä voi johtua osittain siitä, että Samstock-tuotteiden vaatimustenhallinnassa ei ole käytettävissä OpenMethod-menetelmäsä suositeltua vaatimustenhallintaan soveltuvaa ohjelmistoa, joka tukee myös jäljitysketjujen tekemistä.

Koulutusmateriaalin lisäksi OpenMethod-menetelmässä ja siihen keskeisesti liittyvässä OpenProject-menetelmässä on myös konkreettisia keinoja vaatimustenhallinnan helpottamiseksi. Esimerkiksi muutostenhallintaan ja konfiguraationhallintaan on olemassa omat toimintaohjeensa. Muutostenhallinnan toimintaohjeissa kuvataan yksityiskohtaisella tasolla esimerkiksi muutostenhallinnan vastuujako ja muutostenhallinnan kohteet. Muutostenhallinnan kohteita määriteltäessä on toimintaohjeessa esitetty lista erilaisista dokumenteista ja muista kokonaisuuksista, jotka voidaan määritellä muutostenhallinnan kohteiksi. Myös muutostenhallintaprosessin vaiheet on määritelty.

Muutostenhallinnan mahdollistamiseksi on tehty dokumenttipohjat muutospyynnöille ja muutosrekisterille. Muutospyyntöpohjassa kuvataan asiat, jotka

muutospyynnöstä on ainakin kirjattava. Muutosrekisteri on puolestaan Excel-dokumenttipohja, johon on määritelty muun muassa muutospyynnön tila, aikataulut, vastuuhenkilöt sekä työmääriin ja kustannuksiin liittyvää tietoa.

Johtopäätöksenä voidaan todeta, että OpenMethodissa on otettu kantaa ja perusteltu vaatimustenhallinnan tärkeyttä, mutta siihen tarvitaan vielä selkeämpiä ohjeita ja käytänteitä ylläpidon aikaiseen vaatimustenhallintaan. Siitä huolimatta OpenMethodin vaiheittain etenevä käyttöönotto varmasti kuitenkin auttaa merkittävästi Samstock-tuotteiden vaatimustenhallinnan systematisoinnissa.

7.3 Vaatimustenhallinta kohdeorganisaatiossa

Tutkimuksessa selvisi, että Digia Samstock -tuotteiden vaatimustenhallinnassa on vielä kehitettävää, mutta toisaalta vaatimustenhallinnan tärkeys on tiedostettu. Kuvioon 12 on koottu SWOT-nelikenttään haastattelujen perusteella havaittuja vahvuuksia, heikkouksia, mahdollisuuksia ja uhkia Digia Samstock -tuotteiden vaatimustenhallinnasta.

Vaatimustenhallinnan yhtenä keskeisimmistä vahvuuksista on hiljainen tietämys. Hiljaisen tietämyksen huomattava merkitys vaatimustenhallinnassa tuli esille kahdeksassa kaikkiaan kymmenestä haastattelusta. Pitkään Samstock-tuotteiden parissa työskennelleillä henkilöillä koettiin olevan vahvan IT-alan osaamisen lisäksi myös huomattavan paljon finanssialan toimialaosaamista ja -ymmärrystä.

Haastatteluissa koettiin vahvuudeksi myös asiakkailta tulevat kehitysideat, jotka edesauttavat ohjelmiston kehittämistä asiakkaiden todellisia tarpeita vastaaviksi. Samstock-tuotteiden käyttäjillä on kattava tietämys siitä, kuinka Samstock-tuotteita käytetään heidän liiketoimintansa näkökulmasta, joten käyttäjiltä saadaan usein kehitysehdotuksia, joita ei ole välttämättä edes mahdollista huomata tuntematta asiakasyritysten liiketoimintaa sisältäpäin. Asiakkaiden ehdottamia muutospyyntöjä pyritään haastateltujen mukaan toteuttamaan mahdolli-

simman aktiivisesti resurssien rajoissa. Yksi haastateltava kommentoi asiakkaiden ehdottamia muutoksia seuraavasti: ”*Me arvostetaan sitä meidän asiakaskuntaa kyllä tosi vahvasti siinä mielessä, et me kuunnellaan niitä herkällä korvalla. Sieltä se paras tietämys monesti löytyy ja se tarve ja hyvät kehitysideat tulee sieltä monesti.*”

Vahvuudet	**Heikkoudet**
Paljon hiljaista tietämystä ja vahvaa toimialaosaamista Uusissa projekteissa vaatimusmäärittely tehdään huolellisesti ja OpenMethodin mukaisesti Esimerkiksi kehityspyyntöjen käsittelyn prosessi on kuvattu Asiakkaat ohjelmistojen kehittämisen ideoijina	Pääsy dokumentteihin ja tietoon Vaatimustenhallintaan liittyvien dokumenttien pitäminen ajan tasalla ja siihen liittyvän vastuun epäselvyys Vaikka olisi halua parantaa vaatimustenhallinnan kypsyystasoa, päätöksiä konkreettisista keinoista ja tavoista puuttuu Alkuperäisen dokumentaation puuttuminen
Mahdollisuudet	**Uhat**
OpenMethod mahdollistaa systemaattisemman vaatimusmäärittelyn ja sen jälkeisen vaatimustenhallinnan Hiljaisen tietämyksen dokumentoiminen toisi käyttöön laajan tietovaraston Vaatimustenhallinnan merkitys on alettu tiedostaa Vaativat asiakkaat kannustavat systemaattisempaan vaatimusmäärittelyyn ja vaatimustenhallintaan Projektissa hyvin dokumentoitu vaatimustenhallinta mahdollistaa työn jakamisen (esim. testauksen osalta) ja sujuvoittaa myöhemmin ylläpitoon siirtymistä Työnkierto tuotekehityksessä	Ohjelmistojen nopea kehitys uhkaa muutostenhallinnan dokumentaation ajantasaisuutta ja luotettavuutta Ohjelmistojen nopea kehitys asettaa huomattavia haasteita konfiguraationhallinnalle Muutosten laajuuden arviointi on hyvin riippuvainen kokeneiden henkilöiden hiljaisesta tietämyksestä Pieni muutos yhdessä ohjelmistossa voi aiheuttaa laajat seuraukset useaan ohjelmistoon Puutteellisesta dokumentaatiosta aiheutuvat ongelmat muutospyyntöjen luokittelussa laskutettaviin ja ei-laskutettaviin Vanhojen huoltovaiheessa olevien ohjelmistojen hallitun käytöstäpoistamisen haastavuus

KUVIO 12. SWOT-analyysi kohdeorganisaation vaatimustenhallinnasta

Työnkierto havaittiin haastattelujen perusteella olevan mahdollisuus, josta voi kehittyä ja on varmasti osittain kehittynytkin jo vahvuus. Erään haastateltavan

sanoin tarkoituksena on, että ”*yritetään kierrättää porukkaa silleen, että kaikki pääsisi tutustumaan suurimpaan osaan ominaisuuksia*”. Työnkierrolla ei tässä tapauksessa kuitenkaan tarkoiteta työtehtävien täydellistä vaihtamista, vaan esimerkiksi sitä, että tiettyä ohjelmistoa kehittävät ohjelmoijat eivät kehitä jatkuvasti samaa osaa ohjelmistosta, vaan heille pyritään saamaan mahdollisen oman erikoisosaamisalueen lisäksi ainakin jossain määrin kokonaiskuva ohjelmistosta ja sen riippuvuuksista.

Kohdassa 7.2 kerrottiin OpenMethod-menetelmästä, jonka tarkoitus on yhtenäistää vaihtelevia käytäntöjä. OpenMethodin käyttöönoton koettiin olevan yksi vaatimustenhallinnan tärkeimmistä mahdollisuuksista, koska se luo puitteet laadukkaille ja yhtenäisille käytännöille. Haastatteluissa selvisi myös, että prosessien kuvaamista on tehty viime aikoina aktiivisesti ja esimerkiksi muutospyyntöjen arvioimiseen on olemassa kuvattu menettelytapa.

Haastatteluissa kävi ilmi, että OpenMethodin mukaisiin käytänteisiin ollaan siirtymässä asteittain. Esimerkiksi uusissa projekteissa käytetään jo mahdollisuuksien mukaan OpenMethodin mukaisia toimintatapoja, jolloin myös syntyvä dokumentaatio on yhtenäisyytensä vuoksi helpommin hyödynnettävää jatkossa.

Haastatteluissa pidettiin ongelmallisena kuitenkin sitä, että vaikka OpenMethodin käytänteet koettiinkin pääosin hyviksi, esimerkiksi vaatimustenhallinnan muuttamista OpenMethodin mukaiseksi sekä muutoksen aikataulua ja prioriteettia ei ole tehty selväksi. Vaatimustenhallinnan ongelmana on se, että aiempaa dokumentaatiota, esimerkiksi ajantasaisia vaatimusmäärittelydokumentteja ei ole välttämättä saatavilla. Haastattelujen perusteella havaittiinkin, että kiinnostusta OpenMethodin käyttöönottoon on, mutta toisaalta kaivattaisiin selkeää linjausta siihen, miten tulisi esimerkiksi dokumentoinnin osalta menetellä tilanteissa, jossa tehdään muutos sellaiseen toiminnallisuuteen, josta ei löydy alkuperäistä dokumentaatiota.

Ohjelmistoista ei siis ole olemassa kattavia ja kaikilta osin dokumentoituja vaatimusmäärittelyitä. Lisäksi haastateltavien mukaan ongelmia aiheutti ensinnäkin se, että vaikka ohjelmiston toiminnallisuuksiin ja vaatimuksiin liittyvää dokumentaatiota olisi olemassa, sitä ei välttämättä aina löydetä. Lisäksi vaikka dokumentit löydettäisiinkin, ei uskalleta luottaa siihen, että ne olisivat ajantasaisia. Haasteet dokumentteihin pääsyssä ja toisaalta dokumenttien ajan tasalla pitämisessä nousivat esiin kuudessa haastattelussa.

Haastateltavat totesivat, että vaatimuksiin ja vaatimusten muuttumiseen liittyvän dokumentaation tulisi olla helposti saatavilla ja sen ajantasaisuuteen pitäisi voida luottaa. Haastatteluissa ilmennyt tarve on kohdeorganisaatiossa jo ainakin osittain tiedostettu. Tietovarastoinnin merkitys tuli muutamissa haastatteluissa esille ja esimerkiksi tietämyskannan kehittäminen koettiin tärkeäksi.

OpenMethodin lisäksi systemaattisen vaatimusmäärittelyn ja vaatimustenhallinnan kannustajina ovat myös vaativat asiakkaat. Haastatteluissa asiakkaan vaativuus koettiin lähes yksiselitteisesti positiiviseksi asiaksi. Kun asiakas vaatii täsmällistä ja dokumentoitua vaatimusten- ja erityisesti muutostenhallintaa, sen tärkeyttä ei kyseenalaisteta, eikä sitä heikennetä, vaikka se muuten saattaisi jäädä usein kiireisen aikataulun vuoksi vähemmälle huomiolle. Haastateltavien mukaan kaikki asiakkaat eivät kuitenkaan yhtä aktiivisesti kiinnitä huomiota kurinalaiseen vaatimustenhallintaan. Tällaisissa tilanteissa haastateltavat kokivat asiakkaan vastuuttamisen toimivaksi mahdollisuudeksi.

Vaikka vaativat asiakkaat koettiin positiivisena asiana, ei oma-aloitteinen vaatimustenhallinnan systematisoiminen saisi jäädä sen varjoon. Erityisesti yhdessä haastattelussa esitettiin näkemys, että asiakkaiden vaatimukset voivat kannustaa kehittämään vaatimustenhallintaa, mutta mikäli vaatimustenhallintaan kiinnitetään huomiota vasta asiakkaan aloitteesta, voi siitä aiheutua ongelmia. Näkemystä perusteltiin sillä, että järjestelmällistä vaatimustenhallintaa vaativa asiakas hoitaa todennäköisesti oman vaatimustenhallintansa systemaattisesti ja

ei välttämättä halua yhteistyöhön yrityksen kanssa, jossa ei oma-aloitteisesti ilman asiakkaan painostusta hoideta vaatimustenhallintaa systemaattisesti. Tiivistetysti kyseisen haastateltavan ajatuksena oli, että asiakkaan vaativuus vaatimustenhallinnassa on hyvä keino kehittää vaatimustenhallinnan kypsyystasoa, mutta lisäksi tarvitaan myös yrityksen oma-aloitteista kiinnostusta ja keinoja vaatimustenhallinnan kypsyystason parantamiseksi.

Haastatteluissa kävi ilmi, että puutteellinen vaatimusmäärittely ja vaatimustenhallinta ovat aiheuttaneet ongelmia. Ongelmia on aiheutunut asiakasprojektin aikana muun muassa silloin, kun asiakkaalta tulee muutospyyntö. Muutospyyntöä on ongelmallista luokitella kehityspyynnöksi ja sen myötä maksulliseksi, mikäli asiakas on sitä mieltä, että kyseinen muutos kuuluu alkuperäisiin vaatimuksiin. Mikäli vaatimusmäärittely on tehty alusta alkaen huolellisesti ja sitä on päivitetty muutosten myötä, se toimii hyvänä arviointikriteerinä sille, onko asiakkaan muutospyyntö uutta kehitystä vai ei. Haastatteluissa havaittiin myös, että esimerkiksi testaustehtäviä on helpompi jakaa, jos vaatimusmäärittely on ajan tasalla ja sen pohjalta on mahdollista tehdä yksiselitteiset testitapaukset. Tällöin testaaminen on mahdollista tehdä todellisiin ja hyväksyttyihin vaatimuksiin peilaten, eikä testaajilla tarvitse olla projektin sisäistä tietämystä, jotta testaaminen olisi mahdollista.

Haastattelujen mukaan haasteita vaatimustenhallinnan kehittämiselle on aiheuttanut viime aikoina ohjelmistojen nopea kehitystahti. Erityisesti muutostenhallinta ja konfiguraationhallinta koettiin haastatteluissa erittäin keskeisiksi kehityskohteiksi. Esimerkiksi konfiguraationhallintaa kuvailtiin tärkeäksi kehityskohteeksi yhdeksässä haastattelussa. Yhdessä haastattelussa mainittiin konfiguraationhallinnan tavoitteeksi se, että kunkin asiakkaan osalta olisi järjestelmällisesti dokumentoituna konfiguraatio ja tiedot siitä, milloin ja mitä ohjelmistopäivityksiä ja korjauspaketteja asiakkaalle on lähetetty ja tiedot siitä, mitä asiakkaan konfiguraatioon kuuluu.

Yhdeksi uhaksi tai ehkä enemmänkin haasteeksi koettiin haastatteluissa se, että pienikin muutos yhteen ohjelmistoon voi aiheuttaa laajat vaikutukset useaan ohjelmistoon. Vaikutusten laajuuden arvioinnin vaikeus mainittiin viidessä haastattelussa. Muutosten vaikutusten laajuuden arviointi kuuluu pääosin tuotekehityksen vastuulle, ja yleisimmin vaikutusten arvioinnin vaikeus nousikin esille juuri tuotekehityksen henkilöiden haastatteluissa. Toisaalta haastatteluissa vahvuutena pidetty kokemuksen myötä kehittynyt asiantuntijuus koettiin keskeiseksi keinoksi suoriutua muutosten laajuusvaikutusten arvioinnista.

Vaatimustenhallinnan kypsyystason nostamista näyttäisi haastattelujen mukaan hidastavan se, että ei ole olemassa tukevaa perustaa, jolle vaatimustenhallinnan järjestelmällistämistä voisi rakentaa. Eli ohjelmistoilta puuttuu osittain alkuperäiset vaatimusmäärittelyt, joten pelkkä muutostenhallinta ei riitä, vaan ensin pitäisi saada dokumentoitua alkuperäiset vaatimukset. Tämä lienee kuitenkin käytännössä mahdotonta ohjelmistojen laajuuden vuoksi. Erityisesti yhdessä haastattelussa painotettiin, että jotta vaatimustenhallinnan kypsyystasoa voitaisiin parantaa, täytyisi tehdä päätöksiä siitä, millä tasolla muutokset ja niiden laajuudet dokumentoidaan ja miten toimitaan tilanteessa, jossa alkuperäisiä vaatimuksia ei ole saatavilla.

Yhdeksi keskeiseksi uhaksi, tai enemmänkin haasteeksi, koettiin uusiutumisen vaikeus. Tässä tilanteessa sillä tarkoitettiin esimerkiksi vanhojen ohjelmistojen hallittua käytöstäpoistamista ja korvaamista uusilla ohjelmistoilla niin, että asiakassuhde säilyy. Tästä kerrotaan tarkemmin kohdan 7.4 loppuosassa.

Johtopäätöksenä kohdeorganisaation vaatimustenhallinnan tilasta arvioidaan haastattelujen tulkinnan perusteella, että Samstock-tuotteiden vaatimustenhallinta ei ole vielä saavuttanut kovin korkeaa kypsyystasoa, mutta suuntana on selkeästi vaatimustenhallinnan systemaattisuuden lisääminen.

Vaatimustenhallinnan vahvuutena olevassa suuressa asiantuntijaosaamisen ja hiljaisen tietämyksen määrässä piilee myös uhka. Mikäli hiljaisen tietämyksen

olemassaoloon ja säilyvyyteen luotetaan liikaa, voi järjestelmällisen vaatimustenhallinnan tärkeys hämärtyä, koska esimerkiksi muutostenhallinta saattaa sujua ilman ajantasaista dokumentaatiotakin.

Järjestelmällisen vaatimustenhallinnan hyötyjä on esitelty tämän tutkimuksen kirjallisuuskatsausluvuissa ja toisaalta myös haastatteluiden tuloksia esiteltäessä. Haastatteluiden tuloksissa mainittiin esimerkiksi, että dokumentoitu vaatimusmäärittely auttaa arvioimaan, onko asiakkaan myöhemmin esittämä muutospyyntö lisäkehitystä vai alkuperäisten vaatimusten toteuttamista. Järjestelmällisellä ja dokumentoidulla vaatimustenhallinnalla näyttäisi olevan selkeitä hyötyjä. Vaikka henkilöstön kattava osaaminen ja hiljainen tietämys ovatkin vahvuuksia, ei niihin vedoten kuitenkaan saisi perustella järjestelmällisen vaatimustenhallinnan laiminlyömistä.

Vaatimustenhallinnan tämänhetkiseen tilanteeseen lienee vaikutusta kohdeorganisaation historialla. Kohdeorganisaatio on aloittanut pienenä ja itsenäisenä yrityksenä, jolloin myös vaatimustenhallinta on ollut yksinkertaisempaa, koska ohjelmiston kehittämisessä ja ylläpitämisessä on työskennellyt vain vähän henkilöitä. Tällöin dokumentoidun ja järjestelmällisen vaatimustenhallinnan tärkeys ei välttämättä ole korostunut. Ohjelmistojen kehittyessä ja monimutkaistuessa sekä henkilöstön lisääntyessä hallitusta ja järjestelmällisestä vaatimustenhallinnasta tulee yhä tärkeämpää, koska kaikki henkilöt eivät voi hallita kaikkia osa-alueita. Tällöin esimerkiksi ajantasaisen ja helposti saatavilla olevan dokumentaation merkitys painottuu.

Muutosten laajuuden arvioinnissa turvaudutaan haastateltujen mukaan usein hiljaiseen tietämykseen perustuvaan asiantuntija-arviointiin. Liiallinen riippuvuus siitä voi olla uhka, josta voi aiheutua ongelmia esimerkiksi lomien aikaan. Vaatimusten jäljittäminen ja jäljittämisestä muodostetut dokumentoidut jäljitysketjut voisivat auttaa hiljaisen tietämyksen siirtämisessä eksplisiittiseen muotoon. Tätä tukee myös Heindlin ja Bifflin (2005) näkemys siitä, että vaati-

musten jäljittäminen voi parantaa ohjelmiston laatua, koska sen avulla saadaan ihmisistä riippumatonta tietämystä ohjelmistosta.

Tällä hetkellä jäljitysketjuja ei ole kohdeorganisaatiossa kuitenkaan juuri käytetty. Kuten kohdassa 3.5 vaatimusten jäljittämistä kuvatessa mainittiin, vaatimusketjujen tunnistaminen ja ylläpitäminen voi olla kallista vaatimusten määrän kasvaessa ja jäljityksen tarkkuuden lisääntyessä (Heindl & Biffl 2005). Koska Samstock-tuotteiden kokonaisuus on kehittynyt laajaksi ja monimutkaiseksi tuoteperheeksi, joiden ohjelmistoilla on toisistaan riippuvia osia, vaatisi hienojakoinen vaatimusten jäljittäminen huomattavan paljon resursseja. Myös jäljitysketjujen ylläpitäminen olisi haastavaa. Vaatimusten jäljittäminen voisi kuitenkin hieman karkeammalla tarkkuustasolla olla toimiva työkalu muutosten aiheuttamien vaikutusten arviointiin. Kohdassa 3.5 mainitut Cleland-Huangin, Zemontin ja Lukasikin (2004) jäljitysstrategiat voisivat auttaa jäljityksen kustannustehokkuuden parantamisessa.

Haastateltavat korostivat muutosten vaikutusten arvioinnin haastavuutta erityisesti ylläpidon aikana. Vaikutusten arviointi ja erityisesti niin sanotut ylläpitomuutosten heijastusvaikutukset on havaittu myös kirjallisuudessa yhdeksi ylläpidon keskeiseksi haasteeksi (esim. Yau, Collofello & MacGregor 1978; Queille, Voidrot, Wilde & Munro 1994; Bohner 1996; Wiegers 2003, 344–345). Kohdeorganisaatiossa muutosten vaikutusten laajuuden arvioinnin tekee vielä erityisen haastavaksi se, että yhteen ohjelmistoon tehtävä muutos voi vaikuttaa myös muihin ohjelmistoihin, koska Samstock-tuoteperheen eri ohjelmistoissa on kaikille ohjelmistoille yhteisiä osia. Siksi dokumentoidut ja edes kohtuullisella tarkkuustasolla muodostetut jäljitysketjut voisivat aiheuttamistaan kustannuksista huolimatta olla käyttökelpoinen ratkaisu kehittämään vaatimustenhallintaa henkilöriippumattomammaksi ja järjestelmällisemmäksi.

7.4 Vanhojen ja uusien ohjelmistojen elinkaaren vaiheet ja niiden erityispiirteet

Haastatteluissa koettiin uusien ohjelmistojen olevan enimmäkseen evoluutio/kasvuvaiheessa. Esimerkiksi rahastojen ja salkkujen hallintaan suunnatut uudet ohjelmistot koettiin melko yksimieleisesti olevan evoluutiovaiheessa. Tosin esille tuli myös näkemyksiä, joiden mukaan uudet ohjelmistotkin alkaisivat siirtyä huoltovaiheen suuntaan, sillä vielä uudempaan teknologiaan siirtyminenkin koettiin jo ajankohtaiseksi. Toisaalta joidenkin uusien ohjelmistojen koettiin olevan vasta esittely- ja evoluutiovaiheen välissä. Pääosin kuitenkin koettiin, että uudet ohjelmistot ovat elinkaarellaan evoluutiovaiheessa, koska niitä ylläpidetään ja kehitetään aktiivisesti. Myös uusiin ohjelmistoihin tuleviin kehityspyyntöihin suhtaudutaan lähtökohtaisesti avoimesti, mutta luonnollisestikin kehityspyyntöjen toteuttamisen hyödyllisyys arvioidaan tapauskohtaisesti.

Vanhojen ohjelmistojen elinkaaren vaiheita ei koettu haastatteluissa yhtä selkeiksi kuin uusien ohjelmistojen. Rahastojen hallintaan liittyvä vanha ohjelmisto kategorisoitiin kuitenkin melko yksiselitteisesti vaiheittaisen käytöstäpoiston ja lakkauttamisen vaiheeseen. Ohjelmistolle on olemassa vastaavaan tarkoitukseen suunniteltu korvaava uusi ohjelmisto. Rahastojen hallintaan käytettävään vanhaan ohjelmistoon ei tehdä enää muutoksia. Myös sen käyttäjämäärä on hyvin pieni.

Salkunhallintaan tarkoitettu vanha ohjelmisto jaoteltiin haastatteluissa pääosin huoltovaiheeseen kuuluvaksi, mutta joissakin yksittäisissä haastatteluissa koettiin, että se olisi vasta evoluutiovaiheen loppupuolella tai jo vaiheittaisen käytöstäpoiston vaiheessa. Uuden salkunhallintaan tarkoitetun ohjelmiston ja vanhan salkunhallintaan tarkoitetun ohjelmiston elinkaaret koettiin olevan kuitenkin eri vaiheissa. Asiakkailla on vielä yleisesti käytössä molempia ohjelmistoja. Haastatteluissa ilmeni myös, että kaikki vanhaa salkunhallintaohjelmistoa käyt-

tävät asiakkaat eivät ole todennäköisesti vielä lähitulevaisuudessa siirtymässä uudempaan ohjelmistoon.

Sekä uuteen että vanhaan salkunhallintaohjelmistoon tehdään markkina- ja lakimuutosten vaatimia muutoksia ja ohjelmistoissa ilmenneitä virheitä korjataan. Ohjelmiston kehittämisessä ja parantamisessa on kuitenkin keskeinen ero vanhan ja uuden ohjelmiston välillä. Eräs haastateltava kuvasi vanhan salkunhallintaohjelmiston kehittämisen tilannetta sanoen: ”*me ei käytännössä sinne hirveesti lähetä mitään omaehtoista tuotekehitystä tekemään*”. Kehitystä kuitenkin tehdään, mikäli sen koetaan parantavan tuotteen kilpailukykyä merkittävästi tai jos asiakas on valmis maksamaan kehityksestä. Sen sijaan uutta salkunhallintaohjelmistoa kehitetään haastateltavien mukaan aktiivisesti myös omaehtoisesti.

Haastatteluissa havaittiin, että uusien ja vanhojen ohjelmistojen elinkaaret eivät ole toisistaan riippumattomia, joten sekin voi vaikuttaa ohjelmistojen elinkaarien vaiheiden määrittelyyn. Sekä vanhoissa että uusissa ohjelmistoissa on yhteistä lähdekoodia ja vanhoissa ohjelmistoissa on sekä uusien että vanhojen ohjelmistojen kannalta keskeistä liiketoimintalogiikkaa. Vaikka vanhat ohjelmistot itsenäisinä tuotteina olisivatkin jo elinkaarellaan melko loppuvaiheessa, niiden asema koko Samstock-ohjelmistoperheen osana on kuitenkin hyvin keskeinen. Eräs haastateltava kuvasi vanhojen ohjelmistojen merkitystä mainiten, että ”*se on jokaisen meidän muun tuotteen moottorina eli aina kun kehitetään muita tuotteita, niin sieltä voi tulla näitä linkkejä tänne –– ja ikään kuin sitä kautta se [vanhat ohjelmistot]–– kasvaa.*” Toisin sanoen vaikka vanhat ohjelmistot ohjelmistosta riippuen luokiteltaisiinkin joko huolto- tai vaiheittaisen käytöstäpoiston vaiheeseen, ei se kuitenkaan anna kokonaiskuvaa todellisesta tilanteesta. Vaikka vanhat ohjelmistot asiakkaiden pääasiallisina ohjelmistoina jossain vaiheessa häviäisivätkin, ne tulevat todennäköisesti olemaan uusien ohjelmistojen taustalla vielä pitkään.

Yhdessä haastattelussa tuli esiin näkökulma, jonka mukaan kokonaisvaltaiseen ohjelmistoperheen hallintaan tulisi kiinnittää enemmän huomiota ja vanhoista

ohjelmistoista pitäisi osata luopua ja ne tulisi lakkauttaa hallitusti. Kyseisen haastateltavan mukaan vanhojen tuotteiden hallittu lakkauttaminen liittyy yrityksen uusiutumis- ja reagointikykyyn. Haastateltava kuvasi uusiutumiskykyä ja oma-aloitteista kehitystä sanoen, että ”*jos meillä tehdään pelkästään sitä, mitä asiakas pyytää, niin me ollaan aina jäljessä*”.

Johtopäätöksenä uusien ja vanhojen ohjelmistojen elinkaaren vaiheista ja niiden erityispiirteistä havaitaan, että kirjallisuuteen peilatessa vanhan salkunhallintaohjelmiston elinkaaren vaihe ei ole itsestään selvä. Haastatteluissa ilmeni, että kaikki vanhaa salkunhallintaohjelmistoa käyttävät asiakkaat eivät ole todennäköisesti vielä lähitulevaisuudessa siirtymässä uudempaan ohjelmistoon. Se puoltaisi sitä, että vanhakaan ohjelmisto ei olisi vielä huoltovaiheessa. Näin siksi, että Bennettin ja Rajlichin (2000a; 2000b) mukaan toimintakriittisten ohjelmistojen ei pitäisi ikinä edetä huoltovaiheeseen saakka, ja salkunhallintaohjelmistot ovat asiakasyritysten liiketoiminnalle kriittisiä ohjelmistoja. Vanha ja uusi salkunhallintaohjelmisto ovat kuitenkin elinkaarillaan melko erityyppisissä vaiheissa, ja sen vuoksi molempien sijoittaminen evoluutiovaiheeseen ei välttämättä olisi elinkaaren vaiheisiin jaottelemisen perusteella kuvaavaa.

Omaehtoinen kehittäminen voisikin olla yksi keskeinen piirre, joka erottaa evoluutiovaiheen huoltovaiheesta. Tällöin vanha salkunhallintaohjelmisto voisi olla huoltovaiheessa, vaikka siihen tehdäänkin vielä virheiden korjaamisen lisäksi myös jossain määrin kehitystyötä. Vanhan ohjelmiston kehitystyö on kuitenkin yleensä asiakkaan aloitteeseen perustuvaa, kun taas uuteen ohjelmistoon tehdään asiakkaiden aloitteeseen perustuvan kehityksen lisäksi myös oma-aloitteista tai omaehtoista kehitystä.

Toinen keskeinen johtopäätös liittyy ohjelmistokokonaisuuksien elinkaarten hallintaan. Haastatteluiden perusteella ilmeni, että huoltovaiheessa olevan ohjelmiston tilalle pitäisi jo hyvissä ajoin tuoda uusi ohjelmisto ja vanhalle ohjelmistolle tehdä käytöstäpoistamisaikataulu. Tällöin vanhan tuotteen ylläpitämis-

tä ei jatkettaisi kovin kauan tilanteessa, jossa sitä ei enää kehitetä aktiivisesti. Ei ole välttämättä asiakkaankaan edun mukaista, että huoltovaiheessa olevaa ohjelmistoa ylläpidetään pitkään kehittämättä sitä aktiivisesti. Lisäksi ylläpidon kannalta on helpompaa, kun ylläpitoon ei kuulu suurta joukkoa vanhoja ohjelmistoja ja ohjelmistoversioita.

On kuitenkin huomattava, että kaikki asiakkaat eivät halua siirtyä käyttämään uutta ohjelmistoa, mikäli vanha ohjelmisto täyttää heidän vaatimuksensa. Siitä huolimatta tuotekokonaisuuksien suunnitelmalliseen hallitsemiseen on kiinnitettävä huomiota niin, että vanhoista ohjelmistoista siirrytään sovitulla aikatauluilla uudempiin. Tällöin vanhat ohjelmistot tai ohjelmistoversiot eivät jää päättymättömään ylläpitoon, jonka aikana niitä huolletaan vain välttämättömien tarpeiden osalta. Jos vanhan ohjelmiston ylläpitoa jatketaan kauan kehittämättä ohjelmistoa, on vaarana, että ohjelmisto ei enää täytäkään asiakkaan ehkä jo huomattavastikin muuttuneita tarpeita. Tällöin on riskinä asiakkuuden menettäminen. Jatkuvuuden kannalta voi olla hyödyllistä lakkauttaa hallitusti vanhoja ohjelmistoja, ja hyvissä ajoin ennen vanhan tuotteen lakkauttamista esitellä asiakkaalle korvaavia vaihtoehtoja.

Jos ohjelmistoa kehitetään vain asiakkaan aloitteesta, eikä juuri lainkaan omaaloitteisesti, voi asiakas kokea sen huonona asiakassuhteen hoitamisena. Tämä on ongelmana lähinnä huoltovaiheessa olevilla ohjelmistoilla. Tätä tukee myös Bennettin ja Rajlichin (2000a; 2000b) näkemys siitä, että toimintakriittisten ohjelmistojen ei pitäisi ikinä edetä huoltovaiheeseen saakka. Huoltovaiheessa olevien ohjelmistojen hallittu vaiheittainen käytöstäpoistaminen ja uusilla ohjelmistoilla korvaaminen onkin kohdeorganisaatiossa keskeinen haaste. Tähän liittyy myös se, että mikäli ohjelmistoa kehitetään vain asiakkaiden muutospyyntöjen mukaisesti ja asiakaskohtaisesti, on vaarana, että ohjelmistosta tulee hajanainen vaikeuttaen muun muassa konfiguraationhallintaa.

7.5 Vaatimustenhallinnan keskeisten toimintojen erot ja yhtäläisyydet uusissa ja vanhoissa ohjelmistoissa

Muutospyyntöjen kirjaamisessa ja dokumentoimisessa ei haastattelujen perusteella koettu olevan merkittäviä eroavaisuuksia vanhojen ja uusien ohjelmistojen välillä. Muutosten arviointiprosessi etenee sekä uusissa että vanhoissa ohjelmistoissa samalla tavalla toki huomioiden sen, minkä osa-alueen tuotepäällikköön tai kehityspäällikköön otetaan missäkin tapauksessa yhteyttä.

Vaatimusten jäljittämistä ei ole tehty kattavasti uusissa eikä vanhoissa ohjelmistoissa. Tosin haastatteluissa ilmeni, että uusien projektien myötä vaatimustenhallinnan dokumentaatioon kiinnitetään enemmän huomiota, mutta varsinaista vaatimusten linkittämistä ei kuitenkaan tehdä systemaattisesti uusissakaan projekteissa. Vaatimusten jäljittämiseen ei ole ainakaan vielä käytössä mitään siihen suunniteltua työkalua tai ohjelmistoa, vaikka vaatimusten jäljittämiseen soveltuviin ohjelmistoihin on alustavasti tutustuttukin OpenMethodin myötä. Sekä uusien että vanhojen ohjelmistojen ongelmaksi koettiin haastatteluissa se, että esimerkiksi alkuperäisiä kattavia vaatimuksia ei ole dokumentoitu ainakaan kokonaisten ohjelmistojen osalta. Tämä vaikeuttaa kattavan muutosten dokumentoimisen aloittamista, koska aina ei ole alkuperäistä dokumentaatiota, johon muuttuneet vaatimukset voisi muokata dokumentaatiossa. Koska dokumentaatio on uusissa projekteissa huomioitu huolellisemmin, voi vaatimusten jäljittäminen olla helpommin toteutettavissa tulevaisuudessa. Esimerkiksi käyttötapaus-, vaatimusmäärittely- ja testausdokumenttien huolellinen dokumentointi projektivaiheessa luo perustaa vaatimustenhallinnankin kehittymiselle systemaattisemmaksi.

Koska dokumentoituja jäljitysketjuja ei ole olemassa, turvaudutaan sekä vanhojen että uusien ohjelmistojen muutospyyntöjen arvioinnissa asiantuntijaarviointiin. Vanhoissa ohjelmistoissa henkilöriippuvuus näyttäisi haastattelujen mukaan kuitenkin joiltain osin olevan suurempaa kuin uusissa ohjelmistoissa.

Vanhoissa ohjelmistoissa on uusia ohjelmistoja selkeämmin sellaisia toiminnallisuuksia, joiden kehityksen ja ylläpidon ovat miltei alkukehitysvaiheesta asti hoitaneet tietyt henkilöt ja heillä on myös näihin toiminnallisuuksiin liittyen avainosaaminen.

Konfiguraationhallinta koetaan haastattelujen perusteella keskeiseksi haasteeksi sekä uusissa että vanhoissa ohjelmistoissa. Uusien ja vanhojen ohjelmistojen konfiguraationhallinnan yhdeksi eroavaisuudeksi mainittiin haastatteluissa se, että uusien ohjelmistojen kehittäminen on edennyt asiakasprojektipainotteisemmin kuin vanhempien ohjelmistojen. Tästä johtuen uusien ohjelmistojen konfiguraationhallinta on enemmän asiakaslähtöistä tai asiakaskohtaista kuin vanhojen ohjelmistojen konfiguraationhallinta. Se voi aiheuttaa riskejä. Esimerkkinä mainittiin, että asiakkaan ohjelmiston konfiguraatio muodostuu asiakasprojektin aikana, mutta projektin loppuessa asiakkaan konfiguraationhallintaan tarvittava tieto voi jäädä projektiryhmän sisälle, eikä välity ylläpidon aikana konfiguraationhallinnasta vastaaville henkilöille.

Muutostenhallinnan kannalta vaikuttaa haastattelujen perusteella siltä, että uusiin ohjelmistoihin tuleviin kehityspyyntöihin suhtaudutaan lähtökohtaisesti myönteisemmin kuin vanhoihin ohjelmistoihin tuleviin kehityspyyntöihin. Kehityspyyntöjen toteuttamispäätöksiin näyttäisi kuitenkin vaikuttavan monissa tapauksissa ohjelmiston elinkaaren vaihetta enemmän asiakkuus.

Elinkaaren vaiheella ei haastattelujen perusteella näyttäisi olevan merkitystä ohjelmiston virheiden korjauksiin eikä markkina- tai lakimuutosten aiheuttamiin muutoksiin ohjelmistossa. Virheet korjataan sekä laki- ja markkinamuutosten vaatimat muutokset tehdään sekä uusiin että vanhoihin ohjelmistoihin. Toki on kuitenkin huomioitava, että käytöstäpoiston vaiheessa olevaan vanhaan rahastojen hallinnan ohjelmistoon ei tehdä enää muutoksia. Muutoin lainsäädännön, markkinoiden tai virheiden aiheuttamat muutokset ja korjaukset tehdään sekä uusissa että vanhoissa ohjelmistoissa. Virheistä aiheutuvat muutokset ei-

vät ole asiakkaille laskutettavia, mutta laki- ja markkinamuutoksista aiheutuvien ohjelmistojen muutosten maksullisuus arvioidaan sekä uusissa että vanhoissa ohjelmistoissa tilannekohtaisesti.

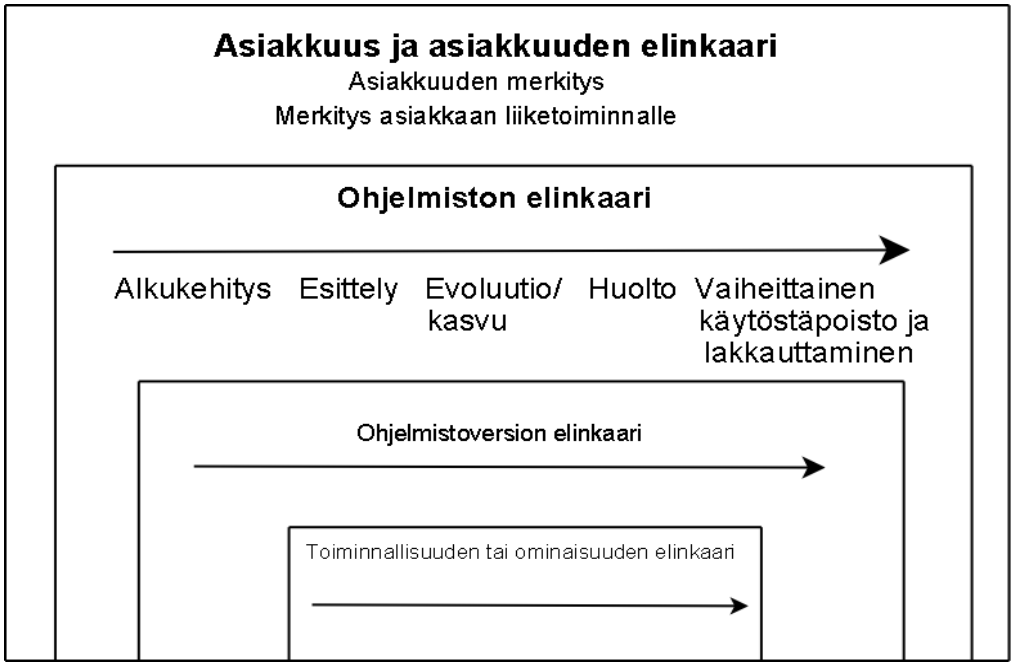
Hylättyjen kehityspyyntöjen määristä ei ole tarkkoja numerotietoja, mutta haastateltavien arvioiden mukaan muutospyyntöjä hylätään lopullisesti melko vähän. Kehityspyyntöjä ei kuitenkaan aina toteuteta heti, vaan vähemmän kiireelliset kehityspyynnöt saatetaan toteuttaa esimerkiksi ohjelmiston seuraavaan versioon. Vanhoihin ohjelmistoihin liittyvissä kehityspyynnöissä on kuitenkin tilanteita, että jos asiakkaan pyytämä kehityspyyntö on jo toteutettu vastaavaan tarkoitukseen käytettävässä uudessa ohjelmistossa, voidaan tilanteen mukaan jättää kehitys tekemättä vanhaan ohjelmistoon. Tällöin asiakasta kannustetaan siirtymään uuden ohjelmiston käyttäjäksi.

Johtopäätöksenä kohdeorganisaation vanhojen ja uusien ohjelmistojen eroista on havaittu, että keskeisin ero uusien ja vanhojen ohjelmistojen välillä on kehityspyyntöjen hyväksymisessä ja ohjelmistojen aktiivisessa kehittämisessä. Sen sijaan esimerkiksi vaatimusten dokumentoimisessa ei havaittu olevan eroa uusien ja vanhojen ohjelmistojen välillä. Myöskään vaatimusten jäljittämisessä ei huomattu olevan merkittäviä eroavuuksia, mutta se voi johtua siitä, ettei jäljitysketjuja tehdä säännöllisesti vanhoissa eikä uusissa ohjelmistoissa. Sen sijaan konfiguraationhallinta koettiin vaikeaksi, mutta jotta voitaisiin tarkemmin määritellä sen eroja uusissa ja vanhoissa ohjelmistoissa, pitäisi sitä tutkia tarkemmin. Konfiguraationhallinnan erojen määrittely vaatisi tarkempaa selvitystä, koska haastatteluissa keskityttiin muutospyyntöjen arviointiin ja käsiteltiin konfiguraationhallintaa vain yleisellä tasolla.

7.6 Elinkaariajattelun laajentaminen

Viitekehyksen elinkaarimallissa on kuvattu ohjelmiston elinkaaren vaiheet. Kohdeorganisaation useimpien ohjelmistojen elinkaaren vaihe on melko selkeä ja useimmat haastateltavat määrittelivät ohjelmistojen elinkaaren monilta osin

yhtenevästi, vaikka eroavaisuuksiakin oli. Haastattelujen perusteella tehtiin *johtopäätöksiä* elinkaariajattelun laajentamisesta. Ohjelmiston elinkaaren vaihetta määriteltäessä havaittiin, että pelkkä kokonaisen ohjelmiston elinkaaren määrittely ja sen vaiheet eivät riitä kuvaamaan todellista tilannetta. Yhtenä tutkimuksen keskeisenä tuloksena onkin se, että kohdeorganisaation tapauksessa koko ohjelmiston elinkaaren rinnalla myös esimerkiksi versio sekä ominaisuus tai toiminnallisuus havaittiin merkityksellisiksi näkökulmiksi. Myös asiakkuuden elinkaari voisi olla yksi näkökulma elinkaariajatteluun. Kuviossa 13 on esitetty tutkimuksessa havaittuja erilaisia näkökulmia elinkaariajattelun hyödyntämiseen.



KUVIO 13. Elinkaariajattelun laajentaminen tulosten perusteella

Tutkimuksessa havaittiin, että koko ohjelmiston elinkaari voi olla eri vaiheessa kuin esimerkiksi ohjelmistoon liittyvät eri versiot. Ohjelmistossa voi olla myös ominaisuuksia, toiminnallisuuksia tai niiden muodostamia kokonaisuuksia, joilla on ohjelmiston elinkaaren sisällä oma melko itsenäinen elinkaarensa.

Konkreettisena esimerkkinä haastatteluissa ilmeni, että vaikka rahastojen hallintaan tarkoitettu uusi ohjelmisto on kokonaisena tuotteena aktiivisen kehittämisen evoluutiovaiheessa, sen vanhimmat versiot ovat kuitenkin jo huoltovaiheessa tai jopa vaiheittaisen käytöstäpoiston vaiheessa. Asiakkaita, jotka käyttävät uuden ohjelmiston vanhaa versiota, kannustetaan vaihtamaan saman tuotteen uudempaan versioon. Siihen, milloin ohjelmistosta tulisi tehdä ja julkaista uusi versio tai versiopäivitys, on asiakastyytyväisyyden näkökulmasta ottanut kantaa esimerkiksi Sahin ja Zahedi (2001).

Yksittäisellä toiminnallisuudella tai ominaisuudella ohjelmistossa voi myös olla oma elinkaarensa. Yksi haastateltava mainitsi esimerkkinä suoraveloitukseen liittyvän toiminnallisuuden, jota on kehitetty melko itsenäisenä osana ohjelmistosta, johon se kuuluu. Se elää melko itsenäistä elinkaartaan ja saattaa tulla elinkaarensa loppuun jo ennen kuin koko ohjelmisto lakkautetaan.

Tutkimuksessa havaittiin myös, että elinkaaren eri vaiheiden sisälläkin voi olla vaihtelua kehityksen nopeudessa. Esimerkiksi uusi rahastojen hallintaan tarkoitettu ohjelmisto on kokonaisena ohjelmistona evoluutiovaiheessa ja siinä on tällä hetkellä menossa nopean kehityksen vaihe.

Vaikka kokonaisten ohjelmistojen elinkaaren eri vaiheet tiedostettiinkin, ja esimerkiksi uusien ja vanhojen ohjelmistojen koettiin olevan eri vaiheissa elinkaarellaan, niin elinkaaren vaihetta ei kuitenkaan koettu ainakaan ainoaksi ratkaisevaksi tekijäksi esimerkiksi muutospyynnön hyväksymistä arvioitaessa. Asiakkuus koettiin useissa tapauksissa tärkeämmäksi vaikuttujaksi kuin ohjelmiston elinkaari. Asiakkuuden merkitys nousi esille kahdeksassa haastattelussa.

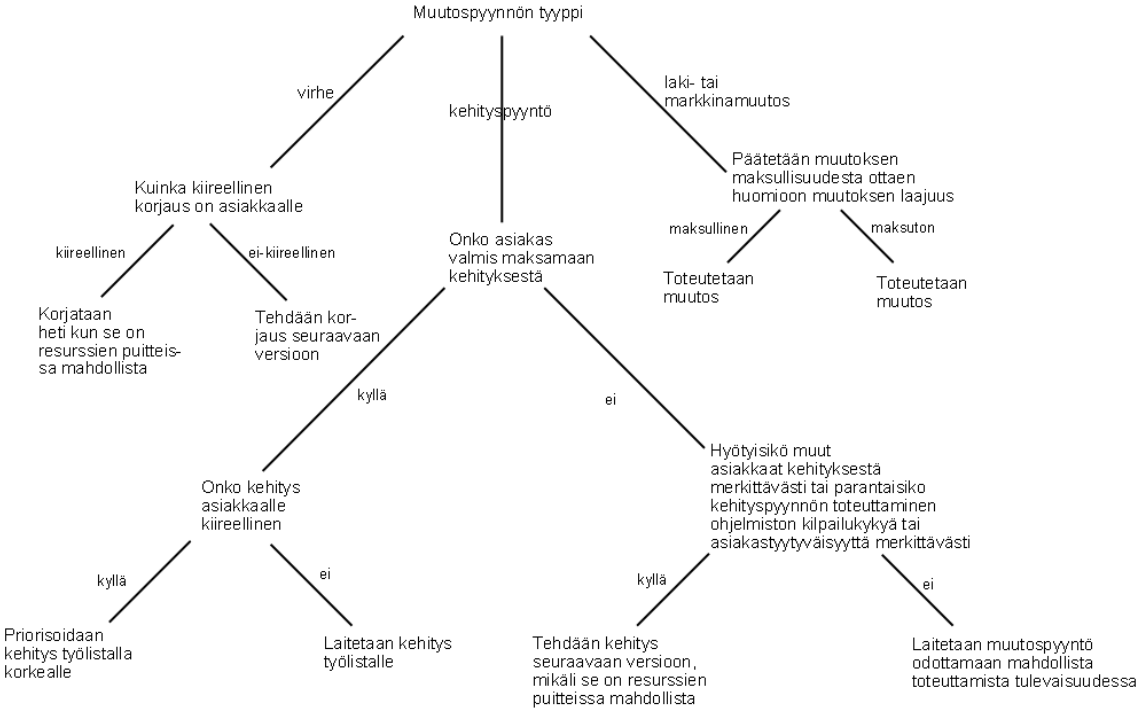
Asiakkuuden merkitys korostuu myös siinä, miten esimerkiksi tuotepäälliköiden toimenkuva on kohdeorganisaatiossa määritelty. Tuotepäällikön toimenkuva ei rajoitu vain yhteen ohjelmistoon, vaan se on määritelty tietyntyyppisten asiakkaiden tarpeiden perusteella. Tämä tukee asiakkuuden elinkaaren merkitystä, sillä tuotepäällikkö tuntee asiakkaan tarpeet laajemmin kuin vain yhden

ohjelmiston osalta. Lisäksi esimerkiksi strategisesti tärkeimpien asiakkuuksien painoarvo ilmenee konkreettisesti niin, että strategisesti tärkeiden asiakkaiden muutospyynnöt pyritään toteuttamaan huolimatta siitä, missä vaiheessa elinkaarta ohjelmisto on.

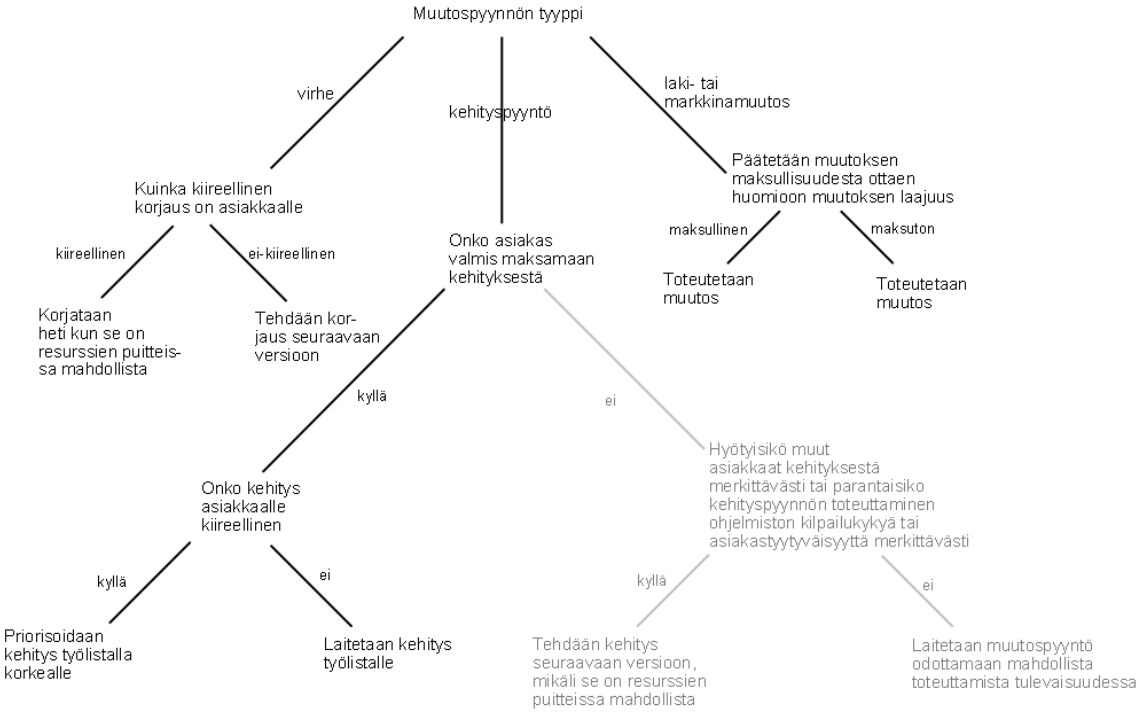
Asiakkuuden keskeisen merkityksen huomioiden on mahdollista, että ohjelmisto, jota strategisesti tärkeä asiakas käyttää, ei edes siirry elinkaarellaan vaiheittaisen käytöstäpoiston vaiheeseen ennen kuin asiakas on siirtynyt käyttämään uutta ohjelmistoa. Asiakkuudella ja oikeastaan eräänlaisella asiakkuuden elinkaaren vaiheella on merkitystä muun muassa muutospyyntöjen toteuttamisen arvioinnissa. Termillä asiakkuuden elinkaari tarkoitetaan tässä tapauksessa sitä, että yksittäisen ohjelmiston ja sen elinkaaren sijaan painotetaan sitä, missä vaiheessa asiakas on elinkaarellaan suhteessa ohjelmistoja kehittävän yrityksen ohjelmistotarjontaan ja kokonaisasiakkuuteen. Asiakkuuden elinkaaren hallinnassa pyritään jatkuvuuteen, jotta olemassa oleva asiakkuus saataisiin säilytettyä sopeutuen asiakkaan ja ympäristön liiketoiminnan muutoksiin. Asiakkuuden elinkaari ei rajoitu välttämättä yhteen ohjelmistoon, vaan siihen voi kuulua useita eri ohjelmistoja tai niiden osia siten, että ne täyttävät asiakkaan tarpeet mahdollisimman hyvin. Asiakkuuden elinkaareen voi siis kuulua useita ohjelmistoja ja ne voivat vaihtua tai kehittyä asiakkuuden kehittymisen myötä.

7.7 Tapaustutkimuksen perusteella muodostetut päätöspuut muutosten arviointiin

Haastatteluiden tuloksista tehtyjen *johtopäätösten* perusteella kuviossa 14 on karkealla tasolla selvitetty, millaisia päätöksiä muutosten toteuttamisen arvioinnissa tehdään evoluutiovaiheessa ja kuviossa 15 on karkealla tasolla selvitetty päätöksentekoa huoltovaiheessa. Asiakkuuden vaikutus kuviossa selvitetään sanallisesti.



KUVIO 14. Päätöspuu muutosten toteuttamisesta evoluutiovaiheessa



KUVIO 15. Päätöspuu muutosten toteuttamisesta huoltovaiheessa

Muodostettaessa kuvioissa 14 ja 15 olevia päätöspuita verrattiin vanhan ja uuden salkunhallintaohjelmiston muutospyyntöjen toteuttamisen arviointia. Oletuksena oli, että vanha salkunhallintaohjelmisto on huoltovaiheessa ja uusi salkunhallintaohjelmisto on evoluutiovaiheessa. Nämä oletukset perustuvat siihen, että suurin osa haastateltavista koki uuden salkunhallintaohjelmiston olevan elinkaarellaan evoluutiovaiheessa ja vanhan salkunhallintaohjelmiston huoltovaiheessa. Vaikka päätöspuut on muodostettu koko salkunhallintaohjelmiston elinkaaren vaihetta arvioiden, voisivat samantyyppiset päätöspuut olla taustalla myös esimerkiksi version tai yksittäisen toiminnallisuuden elinkaaren eri vaiheissa tehtäviä päätöksiä arvioidessa.

Kuvioissa 14 ja 15 ei ole huomioitu asiakkuuden merkitystä päätöksiin. Asiakkuuden merkityksen havaittiin kuitenkin haastatteluissa olevan hyvin keskeinen. Asiakkuuden vaikutusta on kuitenkin vaikea kuvata melko suoraviivaisia päätöksiä kuvaavaan päätöspuuhun. Asiakkuuden vaikutukset eivät välttämättä ole aina tietyllä kaavalla eteneviä suoraviivaisia päätöksiä, vaan vaikutus voi olla tilannekohtaisempaa. Siksi asiakkuuden vaikutuksista kerrotaan sanallisesti samalla, kun kuvaillaan päätöspuiden rakennetta.

Tutkimuksessa havaittiin, että sekä evoluutio- että huoltovaiheessa olevassa ohjelmistossa korjataan havaitut järjestelmävirheet, eikä vaiheiden välillä havaittu sen suhteen eroja. Virheiden korjausnopeus riippuu virheiden kriittisyydestä. Mikäli ohjelmistossa oleva virhe on kierrettävissä helposti, eikä se ole ohjelmiston käyttöä estävä, on mahdollista, että korjaus toteutetaan seuraavaan ohjelmistoversioon. Mikäli virhe on kriittinen asiakkaan liiketoiminnan kannalta, se korjataan mahdollisimman nopeasti. Asiakkuuden merkitys virheiden korjaamisessa on lähinnä sellainen, että strategisesti tärkeiden asiakkaiden raportoimiin virheisiin pyritään reagoimaan erityisen nopeasti.

Tutkimuksessa havaittiin, että myös laki- ja markkinamuutosten aiheuttamat muutokset ohjelmistoon toteutetaan sekä evoluutio- että huoltovaiheessa. Lain-

säädäntö ja siihen tulevat muutokset sekä markkinamuutokset ovat myös tekijöitä, joiden vaikutus on keskeinen toimialalla, jossa Samstock-ohjelmistoja käytetään. Kunkin laki- tai markkinamuutoksen osalta arvioidaan, onko muutos asiakkaalle laskutettavaa. Vaiheiden välillä ei havaittu selkeää eroa siinä, onko laki- ja markkinamuutokset joko evoluutio- tai huoltovaiheessa useammin asiakkaille maksullisia. Asiakkuuden merkitys laki- ja markkinamuutosten aiheuttamien ohjelmistomuutosten tekemisessä ei ole niin suuri kuin esimerkiksi kehityspyyntöjen osalta. Laki- ja markkinamuutokset vaikuttavat yleensä melko samalla tavalla useisiin asiakkaisiin, joten niihin liittyvät muutokset toteutetaan usein keskitetysti yhdellä kertaa.

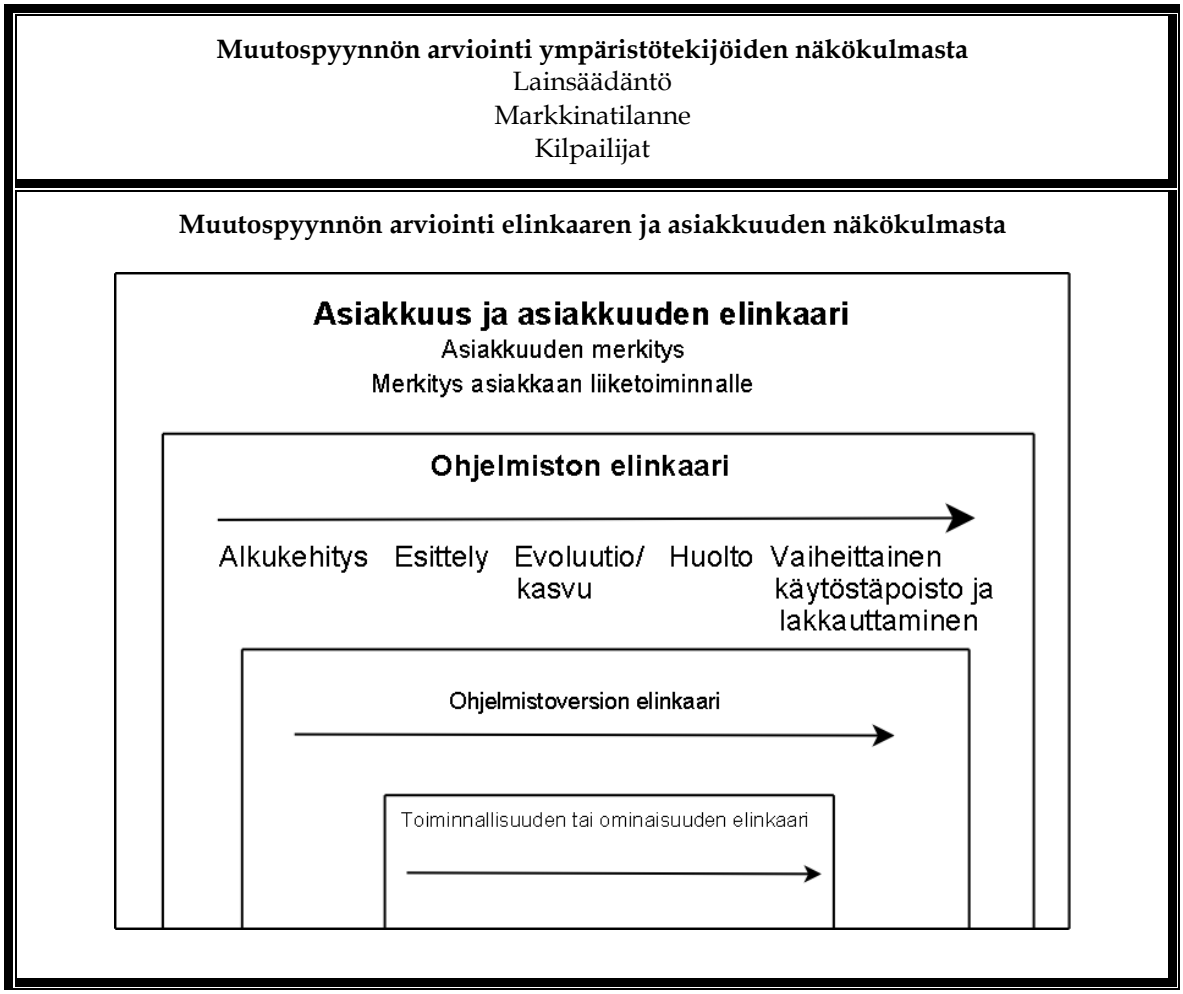
Kehityspyyntöjen toteuttamisessa havaittiin eroja evoluutio- ja huoltovaiheessa olevien ohjelmistojen välillä. Niin kuin kohdassa 7.4 todettiin, huoltovaiheessa olevaan ohjelmistoon tehtävän omaehtoisen tai oma-aloitteisen kehityksen määrä on vähäinen. Huoltovaiheessa olevaan ohjelmistoon tehdään vielä kehitystä asiakkaan aloitteesta, mutta usein kehityspyynnön toteuttaminen on asiakkaalle maksullista. Evoluutiovaiheessa ohjelmistoon tehdään myös usein sellaista kehitystä, jonka asiakkaat saavat käyttöön uuden ohjelmistoversion myötä. Näin toimitaan tapauksissa, joissa muutoksen koetaan olevan erityisen hyödyllinen useille asiakkaille tai muutoksen koetaan parantavan ohjelmiston kilpailukykyä merkittävästi. Tällöin kehitys ei ole asiakkaalle välttämättä ainakaan suoraan maksullista. Myös huoltovaiheessa voi tapahtua tällaista kehitystä, mutta se on hyvin paljon rajatumpaa kuin evoluutiovaiheessa. Vaikka kehityspyyntö olisi sitä ehdottaneelle asiakkaalle maksullista kehitystä, voi se tulla seuraavassa ohjelmistoversiossa myös muiden asiakkaiden käyttöön.

Asiakkuudella on merkitystä myös kehityspyyntöjen toteuttamisen arvioinnissa. Strategisesti tärkeiden asiakkaiden kehityspyyntöihin pyritään reagoimaan erityisen nopeasti. Toisaalta eräs haastateltava totesi, että usein suuret (voidaan tässä tilanteessa tulkita tarkoittamaan likimain samaa kuin strategisesti tärkeät) asiakkaat vaativat, että heidän kehityspyyntönsä toteutetaan nopeasti, mutta

toisaalta he ovat myös muita useammin valmiita maksamaan muutoksen toteutuksesta.

7.8 Tapaustutkimuksen tulosten perusteella muokattu viitekehys

Luvussa 5 esiteltiin viitekehys vaatimustenhallintaan ja erityisesti muutostenhallintaan. Luvun 5 viitekehys perustui kirjallisuuskatsaukseen, eikä siinä ollut vielä huomioitu empiirisen osan tuomia muutoksia. Kuviossa 16 on viitekehys, jota on muokattu tutkimuksessa tehtyjen *johtopäätösten* perusteella.



KUVIO 16. Tapaustutkimuksen perusteella muokattu viitekehys

Tutkimuksessa havaittiin, että koko ohjelmiston elinkaari ei ole riittävän tarkka vaatimustenhallinnan elinkaarinäkökulman lähtökohdaksi. Sen vuoksi ohjel-

miston elinkaaren rinnalle lisättiin myös käsitteet ohjelmistoversion elinkaari sekä toiminnallisuuden tai ominaisuuden elinkaari.

Tapaustutkimuksen perusteella havaittiin myös, että asiakkuus vaikuttaa vaatimustenhallinnan päätöksiin niin voimakkaasti, että elinkaariajattelussakin asiakkuuden merkitys on tiedostettava koko ajan. Sen vuoksi viitekehyksestä poistettiin asiakkuuden näkökulma itsenäisenä osanaan. Asiakkuus ja asiakkuuden elinkaaren näkökulma lisättiin elinkaariajattelun lähtökohdaksi.

Ympäristötekijöiden näkökulma jätettiin itsenäiseksi kokonaisuudekseen, koska tutkimuksen perusteella havaittiin, että esimerkiksi laki- ja markkinamuutosten ohjelmistoon aiheuttamiin muutoksiin elinkaaren ja asiakkuuden merkitys on kohtuullisen rajallinen. Tutkimuksessa saatiin viitteitä siitä, että laki- ja markkinamuutosten aiheuttamat muutokset tehdään ohjelmistoon riippumatta siitä, missä vaiheessa elinkaartaan ohjelmisto on (olettaen kuitenkin, että ohjelmisto ei ole vaiheittaisen käytöstäpoiston ja lakkauttamisen vaiheessa). Asiakkuudella ei tutkimuksessa havaittu olevan keskeistä merkitystä ympäristötekijöiden muutosten aiheuttamiin ohjelmistomuutoksiin, koska esimeriksi laki- ja markkinamuutokset koskettavat yleensä samalla useita asiakkaita ja muutoksetkin hoidetaan silloin usein keskitetysti. Ympäristötekijöitä käsiteltiin tutkimuksessa kuitenkin hyvin suppeasti.

On tärkeää huomioida, että viitekehystä muokattiin tässä tutkimuksessa käsitellyn yhden tutkitun tapauksen perusteella, eikä sen oletetakaan olevan yleistettävissä. Yhden tapauksen perusteella ei ole mahdollista saada yleistettäviä tuloksia elinkaariajattelun merkityksestä vaatimustenhallinnassa. Tutkittu tapaus voi korkeintaan antaa viitteitä siitä, että elinkaaren näkökulma voi olla yksi monista tavoista lähestyä ja tutkia vaatimustenhallintaa.

7.9 Yhteenveto johtopäätöksistä

Tapaustutkimuksessa arvioitiin ensin kohdeorganisaation vaatimustenhallinnan tilannetta SWOT-analyysin avulla. SWOT-analyysissa ei vielä eroteltu tai arvioitu vaatimustenhallintaa eri elinkaarensa vaiheessa olevien ohjelmistojen osalta erikseen, vaan analyysiin koottiin kokonaiskuvaa kohdeorganisaation vaatimustenhallinnasta. Analyysin perusteella havaittiin, että vaatimustenhallinnan kypsyystason nostamista vaikeuttaa esimerkiksi osittain puuttuva dokumentaatio, mutta toisaalta helpottaa OpenMethod-menetelmän käyttöönotto. SWOT-analyysiin koottiin asioita, joita kohdeorganisaatio voi käyttää apuna kehittäessään vaatimustenhallintaansa.

Tapaustutkimuksessa tarkasteltiin myös kahdelta asiakasyritykseltä tulleita yhteydenottoja. Esimerkkitapauksissa näyttäisi siltä, että uuden ohjelmiston tai ohjelmistoversion käyttöönotto lisää asiakkaalta tulevien yhteydenottojen määrää, kun ohjelmisto on asiakkaan näkökulmasta esittelyvaiheessa. Erityisesti järjestelmävirheistä raportoiminen näytti lisääntyneen esimerkkitapauksissa.

Tapaustutkimuksessa on tutkittu Samstock-tuotteiden vaatimustenhallintaa erityisesti muutospyyntöjen kautta. Tutkimuksessa selvisi, että kehityspyyntöjen arvioinnissa on eroja riippuen siitä, onko ohjelmisto evoluutio- vai huoltovaiheessa. Huoltovaiheessa olevan ohjelmiston kehittäminen on vähäisempää ja kehityspyyntöihin suhtaudutaan kriittisemmin. Tutkimuksessa havaittiin kuitenkin, että monissa tapauksissa asiakkuudella on ohjelmiston elinkaaren vaihetta keskeisempi merkitys esimerkiksi kehityspyyntöjä arvioitaessa. Strategisesti tärkeiden asiakkaiden kehityspyynnöt pyritään toteuttamaan huolimatta siitä, vaikka ohjelmisto olisikin jo huoltovaiheessa. Huoltovaiheessa olevaan ohjelmistoon toteutettavat kehityspyynnöt ovat kuitenkin hyvin usein asiakkaille maksullisia, ja omaehtoista kehitystä tehdään huoltovaiheessa olevaan ohjelmistoon hyvin rajoitetusti.

Tutkimuksen tuloksena tehtiin myös päätöspuu sekä evoluutio- että huoltovaiheessa olevan ohjelmiston päätöksenteon tueksi. Päätöspuussa jaoteltiin muutospyynnöt kolmeen eri ryhmään riippuen siitä, mihin asioihin niiden arvioinnissa kiinnitetään huomiota. Virheiden ja kehityspyyntöjen rinnalle kolmanneksi ryhmäksi otettiin laki- ja markkinamuutokset, koska niiden arviointi ei ole samanlaista kuin virheiden tai kehityspyyntöjen.

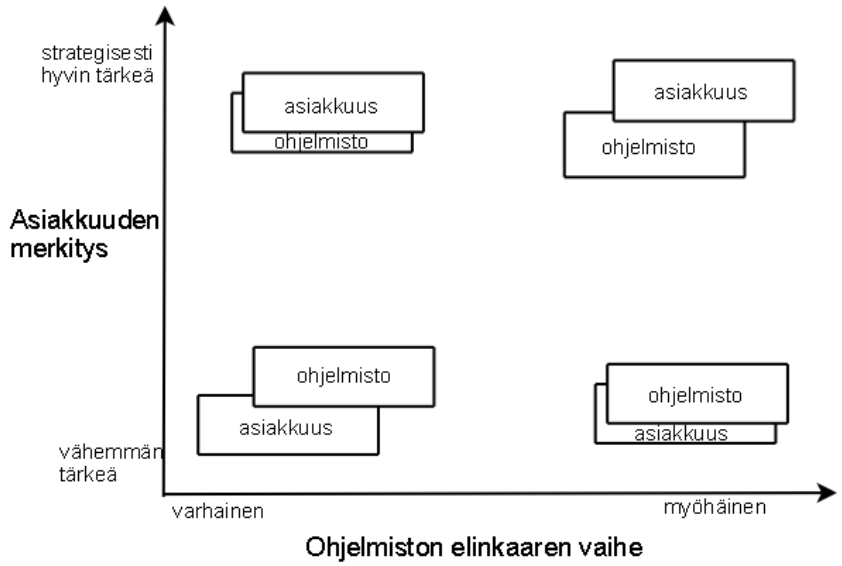
Tutkimuksessa havaittiin viitteitä myös siitä, että elinkaariajattelua voisi olla mahdollista hyödyntää ohjelmistoperhekokonaisuuksien hallinnan kehittämisessä. Ohjelmistoperheen ohjelmistojen elinkaaren vaiheiden tiedostaminen voi auttaa ohjelmistokokonaisuuksien hallinnan systematisoinnissa ja suunnittelussa. Kun ohjelmistoperheen ohjelmistojen elinkaaren vaiheet tiedostetaan, voidaan suunnitella esimerkiksi hallittua vanhoista ohjelmistoista luopumista ja sen aikataulua huomioiden sen, että poistuvien ohjelmistojen tilalle kehitetään uusia ohjelmistoja.

Tapaustutkimuksen tulokset tukevat sitä, että ohjelmiston elinkaaren näkökulma voisi olla yksi mahdollinen näkökulma vaatimustenhallintaan ainakin muutospyyntöjen kautta tarkasteltuna. Lisäksi tutkimuksessa kuitenkin havaittiin, että pelkkä kokonaisen ohjelmiston elinkaaren tarkastelu ei anna kovin kokonaisvaltaista kuvaa. Tutkimuksessa havaittiin, että ohjelmiston elinkaaren sisällä voi olla useita ohjelmistoversioiden elinkaaria ja jopa yksittäisten ominaisuuksien ja toiminnallisuuksien elinkaaria. Vaikka ohjelmisto kokonaisena tuotteena olisikin evoluutiovaiheessa, saattaa olla siihen liittyviä ohjelmistoversioita, joita ei enää kehitetä ja jotka ovat vaiheittaisen käytöstäpoiston vaiheessa.

Edellä mainittujen asioiden lisäksi tutkimuksessa elinkaariajatteluun huomattiin tarpeelliseksi liittää myös asiakkuuden elinkaaren näkökulma. Kohdeorganisaatiossa asiakkuuden merkitys korostui voimakkaasti ja asiakkuus onkin taustalla useimmissa päätöksissä, joita tehdään muutospyyntöihin liittyen. Lisäksi asiakkaille pyritään kokoamaan mahdollisimman kattava kokonaisratkai-

su, joka saattaa sisältää useita eri ohjelmistoja. Ohjelmistot voivat myös muuttua ajan myötä. Näiden havaintojen perusteella tutkimuksessa laajennettiin elinkaariajattelua myös asiakkuuden elinkaareen, joka saattaa sisältää useiden eri ohjelmistojen elinkaaria.

Kuvioon 17 on koottu ohjelmiston elinkaaren vaiheen ja asiakkuuden merkitys kehityspyyntöjen toteuttamista arvioitaessa. Kuvio on muodostettu sen perusteella, millaisia viitteitä tutkimuksessa on saatu. Kuvio on kuitenkin enemmänkin lähtökohta jatkotutkimukselle kuin vankkoihin todisteisiin nojaava kuvaus todellisesta tilanteesta. Kuviota tulkitaan niin, että se laatikko, joka on edempänä, on painoarvoltaan tärkeämpi. Lisäksi mitä vähemmän taustalla olevaa laatikkoa näkyy, sitä vähemmän sillä on painoarvoa.



KUVIO 17. Asiakkuuden ja elinkaaren vaiheen merkitys kehityspyynnöissä

Kuten päätöspuita kootessa mainittiin, haastattelujen perusteella vaikuttaa siltä, että asiakkuuden vaikutus kehityspyyntöjä koskevaan päätöksentekoon ei ole aina suoraviivaista, vaan se voi riippua tilanteesta. Tutkimuksessa saatiin kuitenkin viitteitä siitä, että asiakkuuden merkitys kehityspyynnön toteuttamisen

arvioinnissa pienenee, mitä pidemmällä ohjelmisto on elinkaarellaan. Jos on kyse strategisesti tärkeästä asiakkaasta, asiakkuuden painoarvo säilyy kuitenkin ohjelmiston elinkaaren vaiheen painoarvoa suurempana, vaikka ohjelmisto olisikin jo elinkaarellaan myöhäisessä vaiheessa. Kuitenkin silloin, kun ohjelmisto on jo myöhäisessä elinkaaren vaiheessa, kehityspyynnön toteuttamista pohditaan kriittisesti, vaikka asiakkuus olisi strategisesti tärkeä. Tällöin siis ohjelmiston ja sen elinkaaren painoarvo lisääntyy. Painoarvon lisääntymistä on kuvattu kuviossa 17 siten, että kuvion vasemmassa ylälaidassa ohjelmiston laatikko näkyy vain vähän asiakkuuden laatikon takaa. Kuvion oikeassa ylälaidassa puolestaan ohjelmiston laatikko näkyy jo enemmän asiakkuuden laatikon takaa.

Jos asiakkuus on strategisesti tärkeä ja ohjelmisto elinkaarensa alussa, otetaan kehityspyynnön toteuttamisessa hyvin vahvasti huomioon asiakkaan tarpeet. Tällöin ohjelmistoa voidaan muokata vielä hyvin olennaisiltakin osin asiakkaan toiveiden mukaiseksi. Toisaalta pyritään myös pidentämään asiakkuuden elinkaarta sitouttamalla asiakas käyttämään ohjelmistoa vielä pitkään.

Tutkimuksessa saatiin myös viitteitä siitä, että jos asiakkuus on vähemmän tärkeä, ohjelmistolla ja sen elinkaarella on asiakkuutta suurempi painoarvo kehityspyynnön toteuttamista arvioitaessa. Tosin ohjelmiston elinkaaren alussa vähemmän tärkeidenkin asiakkaiden kehityspyyntöjen painoarvo voi olla suuri, koska ohjelmistoa kehitetään ja parannetaan vielä aktiivisesti ja toisaalta pyritään asiakkuuden elinkaaren jatkuvuuteen.

Ohjelmiston elinkaaren myöhäisessä vaiheessa ohjelmiston ja sen elinkaaren vaikutus korostuu, kun ohjelmistoa ei enää kehitetä aktiivisesti. Tällöin vähemmän tärkeän asiakkuuden painoarvo ei enää välttämättä riitä siihen, että kehityspyyntö hyväksyttäisiin (ks. kuvion 17 oikea alalaita). On kuitenkin huomioitava, että ohjelmiston ollessa elinkaarensa myöhäisessä vaiheessa, asiakkaalle tulisi jo pystyä tarjoamaan mahdollisuus siirtyä uuteen korvaavaan ohjelmistoon, jota kehitetään aktiivisesti.

8 YHTEENVETO

Käyttäjillä ja muilla sidosryhmillä on vaatimuksia, joihin hankittavan ohjelmiston on kyettävä vastaamaan. Vaatimusmäärittelyssä selvitetään tarkoitus, johon ohjelmistoa tullaan käyttämään. Vaatimusmäärittelyssä onnistuminen vaatii kaikkien sidosryhmien tarpeiden ymmärtämistä sekä sen taustan ja ympäristön hahmottamista, johon ohjelmisto hankitaan. Vaatimusmäärittelyn haastavana tehtävänä onkin tunnistaa, neuvotella, päättää, dokumentoida ja varmentaa ohjelmiston vaatimukset. Tähän pyritään vaatimusmäärittelyn eri osien avulla. Osat ovat vaatimusten tunnistaminen, mallintaminen, vaatimusanalyysi, vaatimusten vahvistaminen ja varmentaminen sekä vaatimustenhallinta.

Vaikka useimmat vaatimusmäärittelyn osat painottuvatkin ohjelmiston elinkaaren alkuvaiheeseen, kattaa vaatimustenhallinta koko ohjelmiston elinkaaren. Vaatimustenhallinta on koko ohjelmiston elinkaaren aikana muuttuvien tarpeiden ja vaatimusten sekä niistä aiheutuvien muutosten tunnistamista, arvioimista, dokumentoimista ja jäljittämistä. Vaatimustenhallinnan keskeisiä toimintoja ovat muun muassa muutostenhallinta, version- ja konfiguraationhallinta, vaatimusten jäljittäminen sekä vaatimusten tilojen seuranta ja dokumentointi.

Tutkimuksessa käytettiin ohjelmiston elinkaariajattelun perustana elinkaarimallia, joka koostuu alkukehitys-, esittely-, evoluutio-, huoltovaiheesta sekä vaiheittaisen käytöstäpoiston ja lakkauttamisen vaiheesta. Empiirisessä osassa kuitenkin havaittiin, että kokonaisen ohjelmiston elinkaaren arviointi ei ollut kohdetapauksessa riittävä kuvaamaan todellista tilannetta. Sen vuoksi viitekehykseen lisättiin ohjelmistoversion elinkaari sekä ominaisuuden tai toiminnallisuuden elinkaari. Lisäksi viitekehykseen otettiin myös mukaan asiakkuuden elinkaari yläkäsitteeksi. Asiakkuuden elinkaari voi sisältää useita eri ohjelmistoja, ohjelmistoversioita ja ominaisuuksia eri elinkaarensa vaiheissa.

Tutkimuksessa pohdittiin, miten ohjelmiston elinkaaren vaihe vaikuttaa vaatimustenhallintaan. Kohdeorganisaation tapauksessa verratessa evoluutio- ja huoltovaiheessa elinkaarellaan olevia ohjelmistoja ja päätöspuita rakentamalla havaittiin, että esimerkiksi kehityspyyntöjen toteuttamiseen liittyviin päätöksiin elinkaaren vaiheella on merkitystä. Huoltovaiheessa olevan ohjelmiston kehityspyyntöihin suhtaudutaan kriittisemmin kuin evoluutiovaiheessa olevan ohjelmiston kehityspyyntöihin. Toisaalta tutkimuksessa saatiin viitteitä siitä, että strategisesti tärkeiden asiakkaiden tapauksessa asiakkuuden painoarvo on ohjelmiston elinkaaren vaihetta merkittävämpi.

Keskeisenä erona huoltovaiheessa ja evoluutiovaiheessa olevien ohjelmistojen välillä havaittiin oma-aloitteisen ja omaehtoisen kehittämisen määrä. Evoluutiovaiheessa olevaan ohjelmistoon tehdään vielä huomattavan paljon omaehtoista kehitystä, kun taas huoltovaiheessa olevaa vastaavaa ohjelmistoa ei enää kehitetä aktiivisesti omaehtoisesti, vaikka siihen vielä tehdäänkin asiakkaan aloitteesta tulevaa kehitystä. Sen sijaan virhekorjauksissa ja laki- ja markkinamuutosten aiheuttamissa muutoksissa ei havaittu eroa huolto- ja evoluutiovaiheessa olevien ohjelmistojen välillä. Molemmissa vaiheissa oleviin ohjelmistoihin tehdään virheistä aiheutuvat korjaukset sekä laki- ja markkinamuutoksista aiheutuvat muutokset.

Yhtenä tutkimuskysymyksenä oli, miten vaatimustenhallintaan (ja sen kypsyystasoon) ohjelmiston myöhemmissä vaiheissa vaikuttaa se, miten vaatimustenhallinta on hoidettu ohjelmiston elinkaaren alussa. Havaittiin, että kohdetapauksessa vaatimustenhallinnan kypsyystaso ei ollut vielä kovin korkea, mutta suuntana on selkeästi vaatimustenhallinnan systematisoiminen. Vaatimustenhallintaan ei ole aina kohdeorganisaatiossa kiinnitetty systemaattisesti huomiota. Sen koettiin vaikeuttavan järjestelmällisen vaatimustenhallinnan aloittamista. Konkreettisesti se ilmeni esimerkiksi siten, että järjestelmällisen muutosten dokumentoinnin aloittaminen koettiin haastavaksi, koska jossain tapauksissa alkuperäinen dokumentaatio saattoi puuttua miltei kokonaan.

Tutkimuksessa selvitettiin myös, miten vaatimustenhallintaa olisi mahdollista kehittää ohjelmiston elinkaaren näkökulmasta. Tähän liittyen tutkimuksessa kävi ilmi, että vaatimustenhallinnassa olisi hyvä olla suunnitelma koko ohjelmistoperheen ohjelmistojen elinkaarten hallinnasta. Tällöin esimerkiksi muutostenhallinnan linjauksia voi tehdä huomioiden koko ohjelmistoperheen ohjelmistojen elinkaaret. Konkreettisesti tämä tuli esille siinä, että tutkimuksessa kävi ilmi, että vanhoja ohjelmistoja pitäisi osata lakkauttaa hallitusti, eikä niitä tulisi pitää liian kauan huoltovaiheessa, jolloin niitä ei enää kehitetä aktiivisesti. Tällöin ohjelmistoperheeseen tulisi jo kehittää uusia ohjelmistoja, jotka korvaavat vanhat ohjelmistot. Tämä voi koskea kokonaisten ohjelmistojen lisäksi myös esimerkiksi ohjelmistoversioita tai ominaisuuksia.

Tutkimuksessa esitetyt tulokset eivät ole yleistettäviä, eikä yleistettävyys ollut edes tutkimuksen tavoitteena. Tutkimuksen tavoitteena oli kiinnittää huomiota vaatimustenhallinnan kehittämiseen systemaattisemmaksi. Sen lisäksi tavoitteena oli esitellä kohdetapauksen avulla keinoja ja näkökulmia vaatimustenhallinnan kehittämiseen.

Tutkimuksen lähtökohtana ollut ohjelmiston elinkaaren näkökulma on vain yksi monista mahdollisista näkökulmista tarkastella vaatimustenhallintaa. Muiden, tässä tutkielmassa kokonaan sivuutettujen, näkökulmien lisäksi esimerkiksi asiakkuuden näkökulma havaittiin olevan hyvin keskeinen lähtökohta vaatimustenhallinnalle ja erityisesti muutostenhallinnalle. Lisäksi tutkimuksessa havaittiin, että alun perin lähtökohtana ollut ohjelmiston elinkaaren näkökulma on liian suppea kuvaamaan todellista tilannetta. Sen vuoksi ohjelmiston elinkaaren näkökulmaa laajennettiin ohjelmistoversion ja ominaisuuden elinkaarilla.

Tutkimuksen perusteella näyttäisi siltä, että tutkimuksessa laajennettu elinkaaren näkökulma on käyttökelpoinen näkökulma vaatimustenhallintaan. Tutkimuksessa havaittiin asiakkuuden merkitys niin suureksi, että elinkaaren näkö-

kulma ilman asiakkuuden huomioimista ei ole relevanttia. Sen sijaan asiakkuuden ja elinkaaren näkökulman hyödyntäminen yhdessä antaa todenmukaisemman kuvan esimerkiksi kehityspyyntöjen arvioinnista.

Tutkimuksessa jäi myös paljon selvittämättä. Jatkotutkimusmahdollisuuksia on vielä paljon kohdeorganisaationkin vaatimustenhallinnassa. Kuviossa 17 esiteltiin hyvin alustavalla tasolla tulkinta asiakkuuden ja ohjelmiston elinkaaren vaiheen painoarvoista kehityspyyntöjen toteuttamista arvioitaessa. Asiakkuuden ja elinkaaren vaiheen merkitystä voisi tutkia jatkotutkimuksena tarkemmin. Yksi jatkotutkimusmahdollisuus kohdeorganisaatiossa olisi myös asiakasyrityksiltä tulleita dokumentoituja tuki- ja muutospyyntöjä tarkastelemalla selvittää, millaisia konkreettisia eroja niihin reagoimisessa havaitaan ohjelmiston eri elinkaaren vaiheissa. Lisäksi jatkotutkimusaiheena voisi selvittää, saataisiinko jonkun aivan eri alan kuin finanssialan ohjelmistoja tutkimalla erilaisia tuloksia ohjelmiston elinkaaren merkityksestä vaatimustenhallinnassa.

LÄHDELUETTELO

Albrecht A. J. & Gaffney J. E., 1983. Software function, source lines of code, and development effort prediction: a software science validation. IEEE Transactions on Software Engineering SE-9 (6), 639-648.

Bennett K. & Rajlich V., 2000a. Software Maintenance and Evolution: A Roadmap. Proceedings of the Conference on The Future of Software Engineering Limerick, Ireland, June 4–11. New York: ACM, 73–87.

Bennett K. & Rajlich V., 2000b. A staged model for the software life cycle. Computer 33(7), 66–71.

Boehm B., 1981. Software Engineering Economics. Englewood Cliffs, N.J.: Prentice Hall.

Boehm B., 1986. A spiral model of software development and enhancement. ACM SIGSOFT Software Engineering Notes 11(4), 14–24.

Boehm B., Clark B., Horowitz E., Westland C., Madachy R. & Selby R., 1995. Cost models for future software life cycle processes: COCOMO 2.0. Annals of Software Engineering 1(1), 57–94.

Bohner S. A., 1996. Impact Analysis in the Software Change Process: A Year 2000 Perspective. Proceedings of the International Conference on Software Maintenance (ICSM'96) Monterey, CA, November 4–8. IEEE Computer Society, 42–51.

Chapin N., Hale J. E., Khan K. Md., Ramil J. F. & Tan W-G., 2001. Types of software evolution and software maintenance. Journal of Software Maintenance and Evolution: Research & Practice 13 (1), 3–30.

Cheng B. H. C. & Atlee J.M., 2007. Research Directions in Requirements Engineering. International Conference on Software Engineering, Future of

Software Engineering (FOSE'07) Minneapolis, Minnesota, May 20–26. Washington DC: IEEE Computer Society, 285–303.

Cleland-Huang J., Zemont G. & Lukasik W., 2004. A Heterogeneous Solution for Improving the Return on Investment of Requirements Traceability. Proceedings of the 12th IEEE International Requirements Engineering Conference (RE'04) Kyoto, Japan, September 6–10. IEEE Computer Society, 230–239.

Dart S., 1991. Concepts in configuration management systems. Proceedings of the 3rd International Workshop on Software Configuration Management Trondheim, Norway, June 12–14. New York: ACM, 1–18.

Davis A. M., 1990. The analysis and specification of systems and software requirements. In Systems and Software Requirements Engineering. IEEE Computer Society Press, 119–144.

De Lucia A., Pannella A., Pompella E. & Stefanucci S., 2002. Empirical analysis of massive maintenance processes. Proceedings of the Sixth European Conference on Software Maintenance and Reengineering (CSMR'02) Budapest, Hungary, March 11–13. IEEE Computer Society, 5–14.

Digia 2008a. Toimialat [online]. Digia Oyj [viitattu 19.3.2008]. Saatavilla www-osoitteessa <http://www.digia.com/C2256FEF0043E9C1/0/405001356?opendocument&lang=fi>.

Digia 2008b. OpenMethod [online]. Digia [viitattu 19.4.2008]. Saatavilla pdf-muodossa osoitteessa <http://www.digia.com/C2256FEF0043E9C1/491D0D709B48A45AC2257355001E8AE7/$file/SYSOPENDIGIA_OpenMethod_SRC.pdf>.

Estublier J., 2000. Software configuration management: a roadmap. Proceedings of the Conference on The Future of Software Engineering Limerick, Ireland, June 4–11. New York: ACM, 279–289.

Gorschek T. & Wohlin C., 2006. Requirements abstraction model. Requirements Engineering 11(1), 79–101.

Gotel O. C. Z. & Finkelstein A. C. W., 1994. An Analysis of the Requirements Traceability Problem. Proceedings of the First International Conference on Requirements Engineering Colorado Springs, USA, 18-22 April, pages 94-101.

Guimarães L. R. & Vilela P. R. S., 2005. Comparing software development models using CDM. Proceedings of the 6th Conference on Information Technology Education Newark, New Jersey, 20–22 October. New York: ACM, 339–347.

Hambrick D. C., MacMillan I. C. & Day D. L., 1982. Strategic Attributes and Performance in the BCG Matrix--A PIMS-Based Analysis of Industrial Product Businesses. The Academy of Management Journal 25(3), 510–531.

Heindl M.& Biffl S., 2005. A Case Study on Value-based Requirements Tracing. Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering Lisbon, Portugal, September 5–9. New York: ACM, 60–69.

Henderson B. D., 1973 (alkuperäinen vuosi). The Growth Share Matrix or The Product Portfolio (uudistettu versio) [online]. The Boston Consulting Group [viitattu 1.1.2008]. Saatavilla www-osoitteessa <http://www.bcg.com/publications/files/Experience_Curve_IV_Growth_Share_Matrix_1973.pdf>.

Hirsjärvi S., Remes P. & Sajavaara P., 2004. Tutki ja kirjoita (10., osin uudistettu laitos). Jyväskylä: Gummerus Kirjapaino Oy.

Hofer C. W., 1975. Toward a Contingency Theory of Business Strategy. The Academy of Management Journal 18(4), 784–810.

Hoffmann M., Kuhn N., Weber M. & Bittner M., 2004. Requirements for requirements management tools. Proceedings of the 12th IEEE International Requirements Engineering Conference (RE'04) Kyoto, Japan, September 6–11. IEEE Computer Society, 301–308.

Hofmann H.F. & Lehner F., 2001. Requirements engineering as a success factor in software projects. IEEE Software 18(4), 58–66.

Hooks I., 2000. Requirements Engineering: Is it 'Mission Impossible'?. Requirements Engineering 5(3), 194–197.

IEEE 1987. Guide to Software Configuration Management. IEEE/ANSI Standard 1042–1987.

IEEE Computer Society 1998. Standard for Software Maintenance. IEEE Std 1219.

IEEE Computer Society 1990. IEEE Standard Glossary of Software Engineering Terminology. lEEE Std 610.12–1990.

Jarke M., 1998. Requirements tracing. Communications of the ACM 41(12), 32–36.

Järvinen P. & Järvinen A., 2000. Tutkimustyön metodeista. Tampere: Tampereen Yliopistopaino Oy.

Karlsson J., 1996. Software requirements prioritizing. Proceedings of the Second International Conference on Requirements Engineering (ICRE'96) Colorado Springs, USA, April 15–18. IEEE: 110–116.

Kemerer C. F. & Slaughter S., 1999. An Empirical Approach to Studying Software Evolution. IEEE Transactions on Software Engineering 25(4), 493–509.

Koskinen J., Lintinen H., Sivula H. & Tilus T., 2004a. Evaluation of Software Evolution Options. Tietotekniikan tutkimusinstituutin julkaisuja 14, Jyväskylän yliopisto, Jyväskylä.

Koskinen J., Lintinen H., Sivula H. & Tilus T., 2004b. Evaluation of Software Modernization Estimation Methods Using NIMSAD Meta Framework. Tietotekniikan tutkimusinstituutin julkaisuja 15, Jyväskylän yliopisto, Jyväskylä.

Kotler P., 1965. Competitive Strategies for New Product Marketing over the Life Cycle. Management Science 12 (4) Series B, B104–B119.

Kotler P., 2003. Marketing Management (11th ed.). Upper Saddle River, NJ : Prentice Hall.

Lehman M. M., 1996. Laws of Software Evolution Revisited. Proceedings of the 5th European Workshop (EWSPT '96) Nancy, France, October 9–11. Berlin: Springer, LNCS 1149, 108-124.

Lehner F., 1991. Software life cycle management based on a phase distinction method. Microprocessing and Microprogramming 32(1–5), 603–608.

Lientz B. P. & Swanson E. B., 1980. Software Maintenance Management. Addison-Wesley Publishing Company.

Mannion M. & Keepence B., 1995. SMART requirements. ACM SIGSOFT Software Engineering Notes 20(2), 42–47.

Metsämuuronen J. (toim.)., 2006. Laadullisen tutkimuksen käsikirja. Jyväskylä: Gummerus Kirjapaino Oy.

Nakajo T. & Kume H., 1991. A Case History Analysis of Software Error Cause-Effect Relationships. IEEE Transactions on Software Engineering 17(8), 830–838.

NATURE Team., 1996. Defining visions in context: Models, processes and tools for requirements engineering. Informations Systems 21(6), 515–547.

Niessink F. & van Vliet H., 2000. Software maintenance from a service perspective. Journal of Software Maintenance: Research and Practice 12(2), 103–120.

Nuseibeh B. & Easterbrook S., 2000. Requirements Engineering: A Roadmap. Proceedings of the Conference on the Future of Software Engineering Limerick, Ireland, June 4–11. New York: ACM, 35–46.

Paetsch F., Eberlein A. & Maurer F., 2003. Requirements engineering and agile software development. Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03) Linz, Austria, June 9–11. IEEE Computer Society, 308–313.

Pigoski T. M., 2002. Software maintenance. Teoksessa Marciniak J. J. (toim.) Encyclopedia of Software Engineering. New York: John Wiley & Sons Inc., 1525–1545.

Queille J-P., Voidrot J-F., Wilde N. & Munro M., 1994. The impact analysis task in software maintenance: a model and a case study. Proceedings of the International Conference on Software Maintenance Victoria, Canada, September 19–23. IEEE, 234–242.

Royce W., 1987. Managing the development of large software systems: concepts and techniques. Proceedings of the 9th International Conference on

Software Engineering Monterey, California, March 30–April 2. Los Alamitos: IEEE Computer Society Press, 328–338.

Ruhe G., Eberlein A. & Pfahl D., 2002. Quantitative WinWin – A New Method for Decision Support in Requirements Negotiation. Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering Ischia, Italy, July 15–19. New York: ACM, 159–166.

Sahin I. & Zahedi M., 2001. Policy analysis for warranty, maintenance, and upgrade of software systems. Journal of Software Maintenance: Research and Practice 13, 469–493.

Saiedian H. & Dale R., 2000. Requirements engineering: making the connection between the software developer and customer. Information and Software Technology 42(6), 419–428.

Sneed H. M., 1995. Estimating the costs of software maintenance tasks. Proceedings of the International Conference on Software Maintenance Opio, France, October 17–20. IEEE, 168–181.

Sommerville I., 1998. Software Engineering (5th ed. repr.). Harlow : Addison-Wesley.

SWEBOK., 2004. Guide to the Software Engineering Body of Knowledge 2004 version. Los Alamitos: IEEE Computer Society.

Van Lamsweerde A., 2000. Requirements Engineering in the Year 00: A Research Perspective. Proceedings of the 22nd International Conference on Software Engineering (ICSE '00) Limerick, Ireland, June 4–11. New York: ACM, 5–19.

Wasson C. R., 1971. Product management : product life cycles and competitive marketing strategy. St. Charles: Challenge Books.

Wiegers K.E., 2003. Software Requirements. Redmond: Microsoft Press.

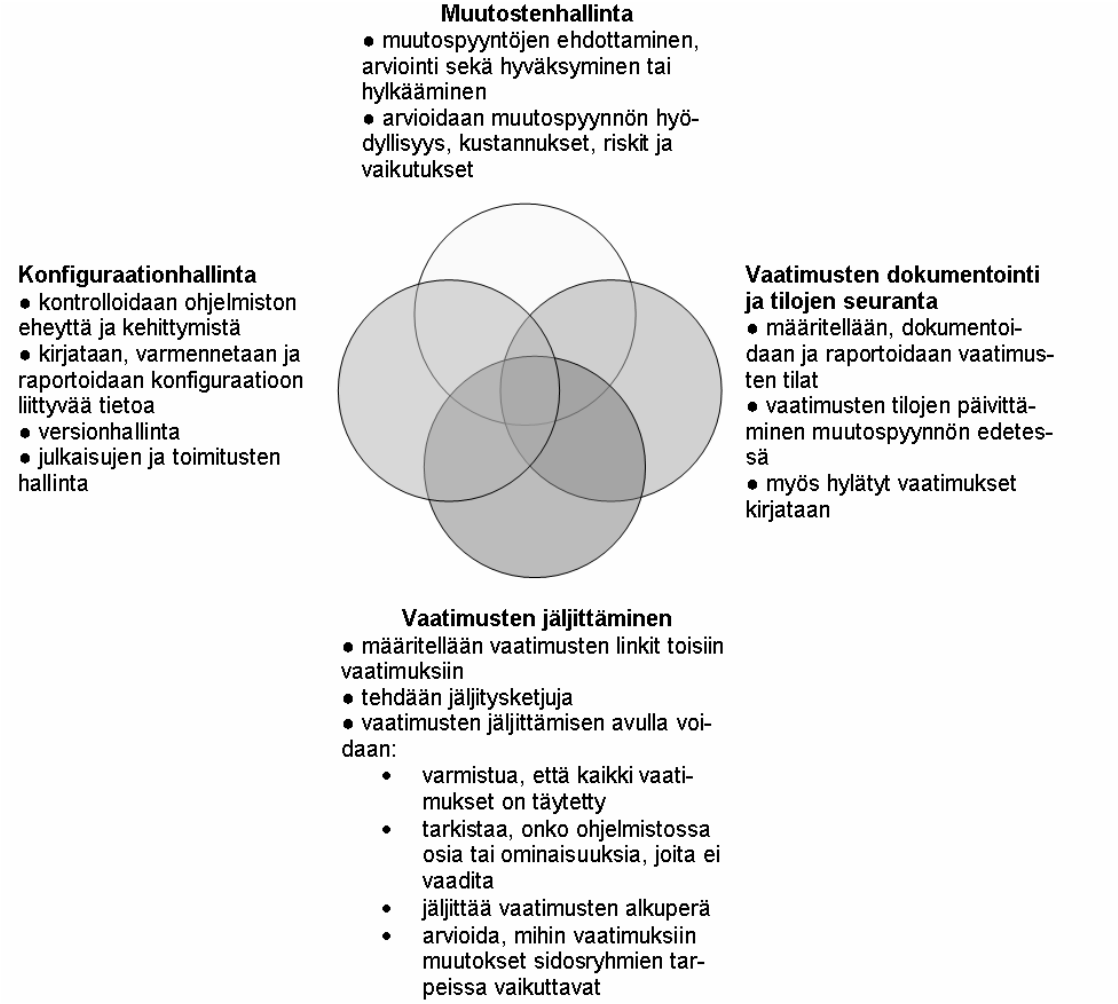
Yau S. S., Collofello J. S. & MacGregor T., 1978. Ripple effect analysis of software maintenance. Proceedings of the Computer Software and Applications Conference (COMPSAC '78) Chicago, USA. Los Alamitos: IEEE Computer Society Press, 60-65.

Zave P., 1997. Classification of Research Efforts in Requirements Engineering. ACM Computing Surveys (CSUR) 29(4), 315–321.

LIITE 1: HAASTATTELUSUUNNITELMA

Haastattelun rakenne (kaikkia haastateltavia koskeva osa)

Haastattelun aluksi kuvaillaan lyhyesti tutkimuksen aihe ja kerrotaan, mitä vaatimustenhallinnalla tarkoitetaan ja mitä keskeisiä toimintoja siihen kuuluu alla olevaa kuviota apuna käyttäen.



Haastattelun aluksi esitellään myös tutkielmassa käytetty elinkaarimalli:

Ohjelmiston elinkaari →				
Alkukehitysvaihe	**Esittelyvaihe**	**Evoluutio/ kasvu-vaihe**	**Huoltovaihe**	**Vaiheittaisen käytöstäpoiston ja lakkauttamisen vaihe**
- Ensimmäinen varsinainen toimiva versio ohjelmistosta - Ohjelmistosta saattaa vielä puuttua ominaisuuksia - Arkkitehtuuri muotoutuu - Ohjelmistotiimin osaaminen kohdealueesta lisääntyy	- Yhdistää alkukehityksen siihen, kun ohjelmisto otetaan täysipainoisesti tuotantokäyttöön - Sisältää takuuajan - Käyttäjiä koulutetaan käyttämään ohjelmistoa - Kestää sitä kauemmin, mitä räätälöidymmästä ja yksilöllisemmästä ohjelmistosta on kyse	- Tehdään iteratiivisia muutoksia ja kehitystä - Mukautetaan ohjelmisto muuttuviin vaatimuksiin - Ohjelmistoa ollaan valmiita parantamaan aktiivisesti	- Ohjelmistolla ei ole enää tarkoituksenmukaista arkkitehtuuria, eikä toimivaa kehitystiimiä - Muutosten tekeminen vaikeaa ja kallista -> muutosten määrä pyritään minimoimaan - Muutosten kustannus-hyötysuhde huono - Ohjelmisto ei ole enää liiketoiminnan kannalta ydintuote	- Ohjelmisto poistetaan vaiheittain käytöstä - Ohjelmistoa ei enää muokata tai kehitetä - Ohjelmistolla voi olla vielä käyttäjiä - Ohjelmistosta kerätään vielä tuottoja niin kauan kuin se on mahdollista - Kun ohjelmisto poistetaan käytöstä, käyttäjät ohjataan käyttämään korvaavaa ohjelmistoa, jos mahdollista

Taustatiedot

- Kuvaile tehtäviäsi ja toimenkuvaasi
- Oletko ollut vaatimusmäärittelyyn tai vaatimustenhallintaan liittyvässä koulutuksessa tai opiskellut alaa?

Vaatimustenhallinta

- Miten vaatimustenhallinta liittyy työhösi?
- Mitä vaatimustenhallintaan liittyviä dokumentteja hyödynnät/teet työssäsi? Mihin ohjelmistoon ne liittyvät (uusi vai vanha)?
- Millaiseksi koet vaatimustenhallinnan merkityksen?
- Miten omassa työssäsi voit vaikuttaa vaatimustenhallintaan?
- Mikä merkitys oman työsi kannalta on sillä, että vaatimustenhallinta on hoidettu hyvin/huonosti?
- Onko yrityksessänne olemassa vaatimustenhallintaan systemaattista menettelytapaa?
- Miten tehtäviisi ja toimenkuvaasi liittyvä osa muutospyynnön käsittelystä, arvioinnista, toteutuksesta jne. etenee ja mihin kuluu eniten aikaa?

- Käytätkö jotain työkalua tai menetelmää apuna vaatimustenhallinnassa?
- Millaisia vaatimustenhallinnan ongelmatilanteita (ja mihin ohjelmistoon liittyen) olet havainnut työssäsi?
- Mitä vaatimustenhallintaan liittyviä asioita on hoidettu erityisen hyvin (ja mihin ohjelmistoon liittyen)?
- Miten kuvailisit uuden ohjelmiston elinkaaren vaihetta, entä vanhan ohjelmiston? Miten esimerkiksi muutospyyntöihin suhtaudutaan eri ohjelmistoissa?
- Kuinka arvioisit sitä, miten usein muutospyynnöt päätetään toteuttaa ja millä perusteella?
- Miten vertailisit uuden ja vanhan ohjelmiston vaatimustenhallintaa seuraavista näkökulmista:
    - muutostenhallinta
    - konfiguraationhallinta
    - vaatimusten jäljittäminen
    - vaatimusten dokumentointi ja tilojen seuranta

Tuotepäälliköille, kehityspäälliköille ja asiakasvastaavalle esitettävät lisäkysymykset (kysymykset käydään erikseen läpi sekä uuteen että vanhaan ohjelmistoon liittyen)

- Miten arvioit muutospyyntöä?
- Miten kuvailisit muutospyynnön arviointia (nopeaa/hidasta, monimutkaista/suoraviivaista)?
- Miten varmoiksi koet muutospyyntöjen arvioinnissa tekemäsi päätökset?
- Mitä dokumentteja hyödynnät arvioidessasi muutospyyntöä?
- Mihin asioihin kiinnität huomiota arvioidessasi muutospyynnön hyväksymistä (esim. asiakkuus, kilpailijat, lainsäädäntö muutospyynnön perusteluna)?
- Vaikuttaako ohjelmiston elinkaaren vaihe muutospyynnön hyväksymiseen? Jos vaikuttaa, miten?

- Miten paljon muutospyyntöjä keskimäärin esitetään (esim. viikossa, kuukaudessa, vuodessa)?
- Ketkä esittävät muutospyyntöjä?
- Miten arvioisit hylättyjen muutospyyntöjen osuutta?
- Millaiset on bugien ja kehityspyyntöjen suhteelliset osuudet?
- Miten arvioit muutosten aiheuttamia kustannuksia (työmäärä jne.)?
- Millaisia työmääriä muutokset tyypillisesti vaativat?
- Mihin kaikkeen muutoksen toteuttamisessa kuluu aikaa? Kuinka suuri osuus menee muuhun kuin varsinaiseen ohjelmointityöhön?
- Kuinka paljon ylläpitoon osallistuvilla henkilöillä on tietämystä ohjelmistosta ja onko ylläpitotehtävissä samoja henkilöitä, jotka ovat olleet ohjelmiston alkukehityksessä mukana?
- Miten arvioit muutoksen aiheuttamaa riskiä?
- Millaiset riskit ovat keskeisimpiä muutospyyntöjen toteuttamisessa?
- Miten arvioit muutoksen vaikutusten laajuutta?
- Jäljitetäänkö muutospyyntöjen vaikutuksia?
- Esitetäänkö kerran hylättyjä muutospyyntöjä usein uudelleen?
- Kirjataanko hylätyt muutospyynnöt?
- Miten kuvailisit ohjelmiston arkkitehtuuria? Onko se yhtenäinen?