Extreme programming – menetelmien hyödyt tietojärjestelmien ylläpidossa

Oulun yliopisto
Tietojenkäsittelytieteiden laitos
Pro gradu
Janne Rytinki
20.5.2013

Tiivistelmä

Tietojärjestelmien ylläpitovaiheen tehtäviin kuuluvat kaikki käyttöönoton jälkeiset muutokset, korjaukset ja kehitystyöt. Tietojärjestelmien ylläpitovaihe kestää vuosista kymmeniin vuosiin ja ylläpitovaiheen menetelminä on käytetty perinteistä vesiputousmallia. Ylläpitovaihetta pidetään myös usein kalliina, muutosten toteutus kestää usein ja virheenkorjauksia joudutaan odottamaan kohtuuttomia aikoja. Kuitenkin viime vuosina Agile–menetelmät ovat enemmän sääntö kuin poikkeus. Tämän tutkimuksen tarkoituksena on tutkia, voidaanko Extreme Programming (XP) -menetelmiä hyödyntää tietojärjestelmän ylläpitovaiheen tehtäviin menestyksekkäästi ja miten XP:n menetelmät voivat tukea tietojärjestelmäkehityksen ylläpitovaiheen tehtäviä.

Tämä tutkimus on laadullinen tapaustutkimus, joka kartoittaa ongelmakohtia yritys Alfassa kyselylomakkeen ja haastattelujen avulla. Saatujen vastausten avulla pyrittiin kartoittamaan yritys Alfassa olevat ongelmakohdat ja kuinka XP:n menetelmät pystyvät parantamaan niitä. Tiedonkeräys suoritettiin yritys Alfan tuki- ja ylläpitotiimin jäsenille ja siinä mukana olleille. Tiedonkeräyksen avulla saavutetut tulokset tukevat hypoteesia, jonka mukaan XP:n menetelmät pystyvät parantamaan tietojärjestelmän ylläpitovaiheen tehtäviä. Tutkimuksen aikana havaittiin, että XP:n menetelmät soveltuvat hyvin ylläpitovaiheen tehtävien suorittamiseen ja niillä pystyttäisiin nopeuttamaan käytössä olevia prosesseja. XP:n menetelmät tukevat hyvin niitä kohtia, jotka nähtiin yritys Alfassa haasteellisiksi tai joissa nähtiin puutteita.

Löydettyjä ongelmakohtia on muun muassa riittämätön palaute, vähäinen kontakti asiakkaaseen, lähdekoodin kompleksisuus, testauksen haasteet ja käyttäjätarinoiden puutteellisuus. XP tarjoaa näihin käytäntöjä, jotka parantavat ongelmakohtia. Parannusehdotuksia ovat esimerkiksi: Refaktoroinnilla saavutetaan yksinkertaisempi lähdekoodi, käyttäjätarinoiden tietosisällön korostaminen, yksikkö- ja hyväksymistestaus parantaa kommunikointia ja palautetta asiakkaan ja ohjelmoijan välillä.

Tutkimuksen tulokset auttavat parantamaan olemassa olevia prosesseja tietojärjestelmien ylläpidossa ja tarjoaa tukea tahoille, joissa havaitaan samankaltaisia haasteita. XP:n menetelmät tarjoavat hyvän viitekehyksen tietojärjestelmien ylläpitovaiheen tehtäville.

Avainsanat
Extreme Programming, laadullinen tutkimus, ITIL

Alkusanat

Haluan kiittää tämän tutkimuksen ohjaajaa FT Tero Vartiaista hyvistä neuvoista ja ohjauksesta tämän työn aikana. Haluan myös kiittää kaikkia haastatteluihin osallistuneita ja kyselylomakkeeseen vastanneita henkilöitä panoksestanne tämän työn aineistoon. Iso kiitos kuuluu myös työnantajalleni, joka mahdollisti tämän tutkimuksen tekemisen.

Kiitos kuuluu myös opponoijalleni FT Juha Iisakalle, joka auttoi selkeyttämään tämän työn lopputulosta.

Janne Rytinki

Oulu, toukokuu 20, 2013

Sisällysluettelo

1. Johdanto

Extreme Programming (XP) (Beck, 1999) käyttö tietojärjestelmän ylläpitovaiheessa on vähemmän käytetty tai vähemmän tiedostettu, johtuen siitä, että XP mielletään usein ohjelmistokehityksen viitekehykseksi. Kuitenkin ylläpitovaihe sisältää samoja elementtejä kuin itse kehityskin, näin ollen XP:tä voidaan soveltaa menestyksekkäästi myös ohjelmiston elinkaarimallin ylläpitovaiheessa (Martin & McClure, 1983). Tietojärjestelmien kehitysvaihde kestää usein kuukausista muutamaan vuoteen mutta ylläpitovaihe voi kestää kymmeniä vuosia. Tästä esimerkkinä pankki- tai tehdasjärjestelmät, jotka ovat olleet jo toiminnassa hyvin kauan, näistä käytetäänkin usein nimitystä ”legacy information system”. Ohjelmiston saavutettua ylläpitovaiheen, sen kehitys kuitenkin harvoin päättyy kokonaan. Pääpainopiste voi siirtyä enemmänkin virheiden korjaukseen, jolloin päivittäinen järjestelmäkehitys voi väistyä taka-alalle. Järjestelmänkehitys harvoin kuitenkaan päättyy kokonaan, vaan järjestelmään tehdään muutoksia, johtuen asiakkaiden toiveista tai tilauksista, lain vaatimat muutokset ym. tapahtumat, jotka laukaisevat järjestelmäkehityksen sen ylläpitovaiheessa.

Ylläpitovaiheessa järjestelmä on täydessä käytössä ja siinä tapahtuvat virhetilanteet aiheuttavat usein palautevyöryn asiakkaalta. Palautetta saadaan myös usein siitä, että virheitä ei korjata tarpeeksi nopeasti tai vaikka korjaus olisi olemassa, ei sitä ole saatu toimitettua asiakkaalle riittävän nopeasti. Nämä ongelman usein johtuvat organisaatiossa käytössä olevista prosesseista, prosessi voi olla kankea ja vähintäänkin yhtä vanha kuin itse järjestelmä. Vertailuna voidaan käyttää esimerkiksi XP vs. vesiputousmalli. Ylläpitovaiheen ongelma on pitää tietty palvelunlaatu yllä asiakkaalle, kuitenkin huomioitta jättämättä asiakkaan tarpeita sen omille asiakkaille. Asiakkaan palvelun laatuun vaikuttaa suoraan järjestelmän toimivuus, kuten käyttökatkot ja virhetilanteet.

Tämän pro gradun tarkoituksena on tutkia tapaus yritys Alfan tietojärjestelmät yksikön käytössä olevia menetelmiä tietojärjestelmien ylläpitovaiheessa sekä kartoittaa haastattelujen avulla ongelmia ja parannuskohteita. Löydettyihin ongelmakohtiin ehdotetaan parannuksia perustuen Extreme Programmingin (XP) menetelmiin.

1.1 Tutkimuksen tausta

Tämän tutkimuksen idea syntyi omista työtehtävistä ja niistä tehtyihin havaintoihin tietojärjestelmän ylläpidossa. Työskentelen yritys Alfan tietojärjestelmäyksikössä sovellusasiantuntijana ja omaan noin kolmen vuoden työkokemuksen kyseisestä tehtävästä. Toimenkuvaani kuuluvat ylläpidolliset tehtävät, kuten ongelmien selvitys, virheiden korjaus, muutostyöt ja koulutus. Näissä tehtävissä olen tehnyt havaintoja, että nykyiset menetelmät joilla asioita hoidetaan, on jokseenkin jäykkiä ja hitaita. Näiden havaintojen johdosta käynnistin tarkemman tutkimuksen, jonka tarkoituksena on löytää ongelmia ylläpitovaiheen tehtävistä ja tarjota niihin ratkaisuja XP:n menetelmistä.

Valitsin Extreme Programming – menetelmän, koska se sisältää elementtejä, jotka näen oman kokemukseni perusteella hyödyllisiksi. Tämän tutkimuksen tulos määrittää sen, nähdäänkö kyseiset menetelmät tai sen osat hyödyllisinä ja pystyvätkö ne parantamaan tämän tutkimuksen kohteen olemassa olevia menetelmiä tai käytäntöjä. Tutkimuksen kulku etenee ensin tutustumalla kirjallisuuteen ja artikkeleihin aiheesta. Tämän jälkeen

tehtävään kartoitukseen yritys Alfan tietojärjestelmäyksikön nykyisin käytössä olevista menetelmistä tietojärjestelmien ylläpidossa ja suoritettiin haastattelut kyseisen tiimin jäsenille. Haastattelut tehtiin avoimella kyselylomakkeella ja käyttäen kyselylomaketta haastattelun runkona.

1.2 Tutkimusongelma ja -kysymykset

Tässä työssä tullaan etsimään yritys Alfassa tällä hetkellä käytössä olevat menetelmät ja niiden ongelmakohdat, sekä pyritään etsimään XP:n tarjoamista menetelmistä parannusehdotuksia löydettyihin ongelmakohtiin. Työssä tullaan vastaamaan seuraavaan ongelmaan:

Miten XP:n menetelmät voivat tukea tietojärjestelmäkehityksen ylläpitovaiheen tehtäviä tapaus Yritys Alfassa?

Tutkimuskysymyksiä, joihin tämä tutkimus pyrkii vastaamaan tutkimusongelman yhteydessä:

Mitkä ovat ylläpitovaiheen ongelmakohtia tapaus Yritys Alfassa?

Miten XP:n menetelmiä voidaan soveltaa menestyksekkäästi tietojärjestelmäkehityksen ylläpitovaiheeseen ITIL-prosesseissa?

Tutkimus on tapaustutkims. Tutkimus kohdistuu ainoastaan yhteen yritykseen ja sen ylläpitotiimin menetelmiin. Näin ollen, kaikkia tuloksia ei välttämättä pystytä yleistämään. Laadullisen tutkimuksen haastattelut ovat myös hyvin tapaus- ja organisaatiokohtaisia, joten haastatteluista saatavat tiedot voivat olla myös vaikeasti yleistettävissä.

1.3 Tutkimusmenetelmät ja tutkimuksen rakenne

Tämä tutkimus on laadullinen tapaustutkimus. Tiedonkeruutekniikoina tässä työssä toimivat yrityksessä tehtävät haastattelut, oma havainnointi ja arkistomateriaali (Järvinen & Järvinen, 2000). Tapaustutkimuksella pyritään kokoamaan tietoa monella eri menetelmällä (Metsämuuronen, 2005), tässä tutkimuksessa tulee esille myös henkilökohtainen havainnointi ja kokemus sekä näiden vertailu haastattelujen tuloksiin. Tiedonkeräys tehtiin kyselylomakkeella, joka lähetettiin koko tiimille. Tämän jälkeen valittiin 5 henkilöä tiimistä, joiden kanssa kyselylomakkeen kysymyksiä käytiin tarkemmin läpi ja suoritettiin sen pohjalta haastattelu.

Tutkimuksessa perehdytään ensiksi olemassa olevaan kirjallisuuteen aiheesta Extreme Programming ja ITIL (IT Infrastructure Library) – menetelmät (Office of Government Commerce. 2007). Nämä muodostavat tutkimukselle viitekehyksen, jonka pohja haastattelukysymyksen on laadittu. ITIL–menetelmät kuvaavat tässä tutkimuksessa sitä, mihin Yritys Alfassa käytössä olevat menetelmät perustuvat. ITIL–menetelmistä otetaan mukaan vain eniten tietojärjestelmän ylläpitoon liittyvät menetelmät ja metodologiat.

1.4 Tietojärjestelmien kehitys ja ylläpito

Tietojärjestelmän kehitys tapahtuu ennen ylläpitovaihetta ja on tärkeä osa ylläpitovaiheen tapahtumia. Jos kehitys on tehty huonosti ja järjestelmä on otettu käyttöön keskeneräisenä, lisää se luonnollisesti ylläpitovaiheen tehtäviä ja kuluja (Dekleva, 1992). Dekleva kertoo artikkelissaan että modernit kehitysmenetelmät eivät

vähentäneet suoraan järjestelmän ylläpitokuluja ja siihen käytettävää aikaa. Moderneina menetelminä käsitetään neljä eri aspektia. Näitä ovat: Ohjelmistotuotanto, jossa määrittelyä, suunnittelua ja ohjelmointia suoritetaan ohjelmiston elinkaarimallin mukaan järjestyksessä. Tässä mallissa määrittely ja suunnittelu ovat keskittyneet liiketoimintaprosessien ympärille. Toisena menetelmänä informaation käsittely, jossa järjestelmää lähestytään tietomassan kautta ja tarkoituksena on järjestää tuleva data esimerkiksi tietokantaan siihen muotoon, että data on sieltä tulevaisuuden tarpeisiin helposti saatavissa. Kolmantena menetelmänä käsitetään prototypointi, jossa järjestelmää kehitettään tietyn mallin avulla lyhyellä syklillä asiakkaan kanssa. Neljäntenä mallina käsitetään CASE-mallia, joka tarkoittaa tietokoneavusteista ohjelmistotuotantoa. Tämä itsessään ei ole menetelmä mutta luo ympäristön, jossa kaikkia kolmea edellistä hyödynnetään tietokoneen avulla (Martin & McClure, 1985).

Eniten järjestelmän ylläpidettävyyteen vaikuttaa järjestelmän laajuus ja kompleksisuus. Dekleva ehdottaakin, että kehittäjien täytyisi miettiä, mitkä ominaisuuden ovat tärkeitä ja välttämättömiä järjestelmän toimivuuden kannalta. Tällä edesautettaisiin järjestelmän ylläpitovaiheen toimivuutta.

Tietojärjestelmän ylläpitovaihe saavutetaan ohjelmistonkehityksen elinkaarimallin loppupäässä. Elinkaarimalli koostuu viidestä erillisestä vaiheesta: määrittely, suunnittelu, ohjelmointi, testaus ja ylläpito (Martin & McClure, 1983). Ylläpitovaiheeseen päästäkseen, tulee suoriutua neljästä edellisestä vaiheesta, jotta koko järjestelmä saadaan tähän pisteeseen. Ylläpitovaihe jatkuu aina järjestelmän alasajoon saakka (White & Cronan, 1997). Tietojärjestelmän ylläpito käsittää kaikki tehtävät muutokset ohjelmistoon sen käyttöönoton jälkeen. Näitä ovat muun muassa toimenpiteet, joilla pyritään pitämään järjestelmä toimintakuntoisena sekä suorituskykyisenä siten, että se vastaa käyttäjän vaatimuksia. Muutokset, joilla toteutetaan lakien tai muiden tahojen määräämät muutokset järjestelmään (Edwards, 1984).

Swanson (1976) esitteli kolme ohjelmiston muutoskategoriaa: i) Parannus, järjestelmän ongelmatilanteiden korjaus datan prosessoinnissa, suorituskyvyssä tai implementoinnissa. ii) Mukautuva, vastataan ennakoituihin muutoksiin datan käsittelyssä. iii) Havainnointi, parannetaan järjestelmän tehokkuutta, suorituskykyä tai ylläpidettävyyttä. Kaksi edellistä kategoriaa on yleisesti hyväksyttyjä teollisuudessa selittämään ohjelmiston ylläpidon aktiivisuutta. Eniten ylläpitotehtäviä aiheuttaa asiakkaan tuomat vaatimukset ja toiveet järjestelmän suorituskyvystä ja toiminnallisuuksista (White & Cronan, 1997).

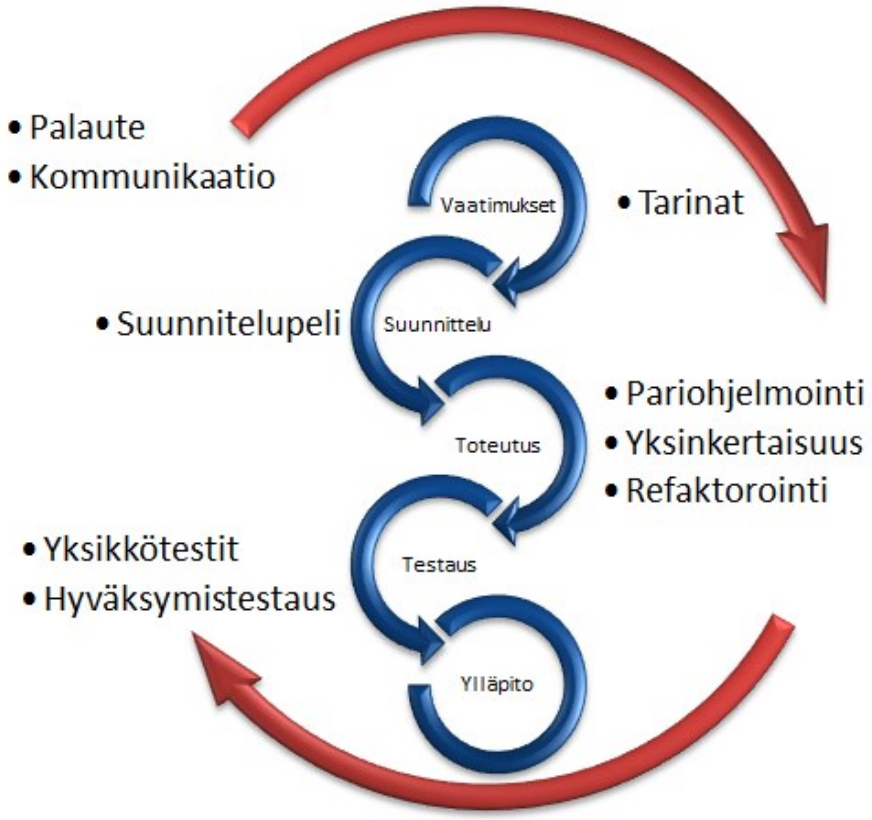
Tietojärjestelmä koostuu ohjelmistosta ja laitteistosta (Edwards, 1984). Tässä työssä tietojärjestelmää käsitellään pelkän ohjelmiston kannalta. Itse laiteympäristöön ja sen ylläpidettävyyteen ei oteta kantaa. Tämä tehdään sen takia, koska laitteiston ylläpito eroaa hyvin paljon ohjelmiston ylläpidosta.

2. Extreme programming

Tässä luvussa kerrotaan XP:n yleiskuvaus. Yleiskuvauksen alaluvussa 2.1 kerrotaan XP:n prosessin kulku ja sen vaiheet. Alaluvussa 2.2. perehdytään XP:n arvoihin ja alaluvussa 2.3 kuvataan XP:n käytännöt. Alaluvut 2.4 ja 2.5 yhdistää XP:n menetelmiä kokonaisprosessiin ja ylläpitovaiheen tehtäviin.

2.1 XP:n yleiskuvaus

Extreme Programming (XP) kehitti ja määritteli Kent Beck (1999), Beckin mukaan XP on kevyt ja ketterä menetelmä, jolla pyritään vähentämään vaatimusmäärittelyä ja laajoja dokumentaatioita. Beckin kirjan mukaan XP koostuu: pariohjelmoinnista, jatkuvasta yksikkötestauksesta ja refaktoroinnista, metaforien käyttäminen jatkuvassa arkkitehtuurisuunnittelussa, jatkuva integraatio, lyhyet suunnittelusyklit ja pyrkimyksenä yksinkertaisuus. XP:n tarkoituksena on tuottaa näkyvää tuotetta asiakkaalle nopeammin sisältäen vähemmän virheitä. Menetelmän mukaan asiakkaisiin ollaan läheisessä yhteydessä, joka edesauttaa edellä mainittuihin tavoitteisiin pääsemistä.

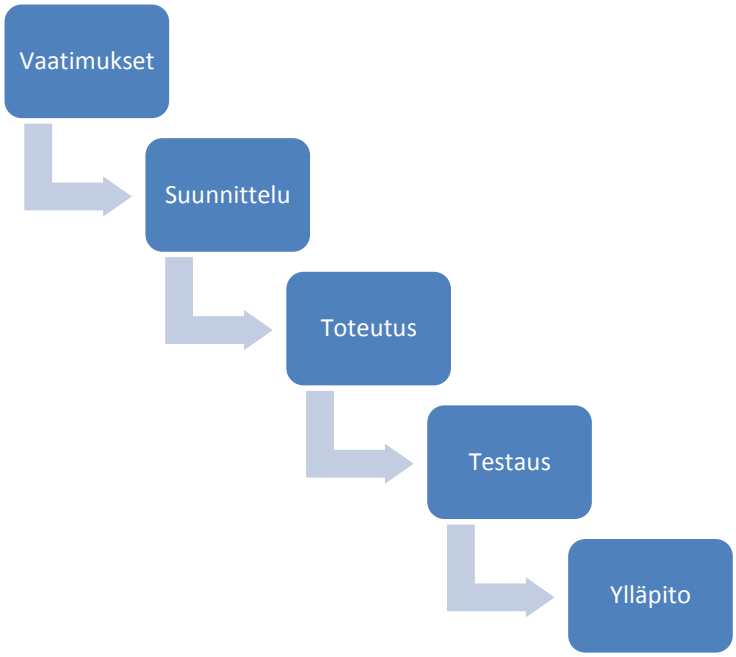


Kuva 1. Extreme Programming prosessikaavio

XP on ohjelmistokehitysmetodologia (kuva 1), joka on suunniteltu nopeaksi tavaksi kehittää ohjelmistoja ja ojelmistojen toiminnallisuuksia. XP on vähäriskinen, tehokas ja joustava ohjelmistokehitystapa, joka keskittyy nykyhetkeen sisältämällä määrittelyn,

suunnittelun, analysoinnin, toteutuksen, testauksen ja käyttöönoton samassa lyhyessä iteraatiossa (Beck, 1999). Iteraatio käsittää kuvassa 1 nähtävän kokonaisen kierroksen. Iteraatioita tehdään siihen saakka, kunnes tavoitteet saavutetaan. Jokaisen iteraation lopussa saadaan jotakin valmista toiminnallisuutta, mikä voidaan ottaa käyttöön.

Jos XP:tä verrataan perinteiseen vesiputousmalliin (kuva 2), missä kehitys tapahtuu hyvin pitkillä vaiheilla, on XP huomattavan paljon ketterämpi. Vesiputousmallissa voidaan tehdä vain yksi iteraatio, joka sisältää kaikki ohjelmistotuotannon vaiheet kertaalleen ja lopputuloksena pitäisi tulla valmistuote. Vesiputousmallin suurin riski on siinä, että jos määrittely- ja suunnitteluvaiheessa tehdään virheitä, kostautuvat ne projektin loppuvaiheilla. Tehdyt virheet ovat huomattavasti kalliimpia korjata loppuvaiheessa, kuin ne olisivat olleet, jos virheet olisi huomattu jo määrittelyvaiheessa. Lähes kaikki ohjelmistosuunnittelumallit pohjautuvat osakseen vesiputousmallista, ainakin sillä tasolla, että ne sisältävät samat ohjelmistotuotannonvaiheet. Kuitenkin XP pyrkii vähentämään vesiputousmallissa esiintyvää riskiä siten, että se tekee nopeita iteraatioita ja pieniä muutoksia sekä ollaan asiakkaan kanssa läheisessä yhteistyössä (Beck, 1999). Näiden avulla iteraatioista saadaan tulokset ja mikäli määrittely on mennyt väärin, voidaan se korjata seuraavassa iteraatioissa. Tämä pienentää riskiä ja korjauksen kustannuksia, niin rahallisesti kuin ajallisestikin (Kivi, 2000).



Kuva 2. Vesiputousmalli

XP:n prosessi lähtee liikkeelle asiakkailta saaduista käyttäjäkertomuksista. Kertomukset ja kertomuksien sisältämät metaforat sisältävät kuvauksen siitä, mitä ja minkälaisia toiminnallisuuksia tai ominaisuuksia asiakas haluaisi. Ohjelmoijat valitsevat haluamansa kertomuksen, jonka ohjelmoija haluaa tehdä. Kuitenkin XP:n prosessin mukaisesti, testit kyseiselle toiminnallisuudelle tulee luoda ennen itse ohjelmointia. Testisuunnitelman ja tapausten luomisen jälkeen voidaan suorittaa itse ohjelmointityö, jota seuraa aiemmin luodut testit (Beck, 1999). Testaus on jatkuvaa ja yksikkötestit tulee mennä läpi ilman virheitä, jotta kehitystä voidaan jatkaa (English, 2002).

Yksi iteraatiokierros sisältää niin monta tehtävää kuin ohjelmoijat itse määrittelevät ja arvioivat kykenevänsä tekemään. Työmääräarviot voivat aluksi olla hyvinkin epämääräisiä mutta iteraatioiden jälkeen saadaan aikaan paljon tarkempia arvioita,

perustuen aiempiin suorituksiin (Jeffries, 2001). Iteraatioista jäävä dokumentaatio pyritään pitämään minimissä ja koko projektista kirjoitetaan minimaalinen dokumentaatio, noin 5 – 10 sivua (Beck, 1999). Dokumenttina toimii lähdekoodi ja asiakkailta saadut käyttäjätarinat ja metaforat. Näin ollen dokumentointi hoidetaan mielellään sosiaalisesti kommunikoimalla ja tarinankerronnalla kuin itse dokumentaatiolla (Juric, 2000).

2.2 XP:n arvot

XP määrittelee viisi eri arvoa, jotka ovat kommunikaatio, yksinkertaisuus, palaute, rohkeus ja kunnioitus (Beck, 1999; Beck & Andres, 2004). XP nojautuu näihin edellä mainittuihin arvoihin ja ne toimivat XP:n perustana.

Kommunikaatio: Käytettäessä mitä tahansa ohjelmistokehitysmetodologiaa on kommunikaatio tärkeässä roolissa. Yleensä kommunikaatio hoidetaan dokumenteilla mutta XP korostaa enemmän suullista viestintää. Viestinnän tulisi olla myös jatkuvaa ja säännöllistä. Tässä varmistetaan tiedonkulku ja vähennetään väärinymmärrysten sekä hämmennystä aiheuttavien asioiden määrää. XP korostaa asiakkaan läsnäoloa ja metaforia. Nämä kuuluvat vahvasti osaksi tätä XP:n arvoa. Jatkuva kommunikaatio projektiryhmän sisällä, kommunikaatio asiakkaaseen ja muihin sidosryhmiin tulee tehdä. XP määrittääkin useampia käytäntöjä miten kommunikaatioita pidetään yllä XP:n prosesseissa (Beck, 1999).

Yksinkertaisuus: Tämän arvon tarkoitus on ohjata XP:n käyttäjiä yksinkertaisimpaan mahdolliseen ratkaisuun. Tällä tarkoitetaan sitä, että esimerkiksi valitaan jokin toteutettava toiminnallisuus ja tehdään se mahdollisimman yksinkertaisesti. Tarvittavat lisäominaisuudet tai muutokset voidaan tehdä jälkeenpäin. Tällä minimoidaan ylimääräisen työn määrä. Yksinkertaisuuteen pyritään ohjelmakoodissa kuin kommunikaatioissakin. Tässä ajattelumallissa keskitytään tähän päivään ja huomisen tai ensi vuoden tarpeita ei oteta juurikaan huomioon. Tämä voi olla myös ongelmallista ja osia voidaan joutua tekemään uudestaan tämän takia. Yksinkertaisuus on suoraan yhteydessä kommunikaatioon. Mikäli kompleksisuus lisääntyy, täytyy myös kommunikaatioita lisätä asian ymmärtämiseksi. Asiakkaan on helpompi ymmärtää yksinkertaisesti toteutetut ominaisuudet. Lähdekoodin yksinkertaisuus auttaa ohjelmoijaa ymmärtämään ominaisuuden toimintalogiikan helpommin (Beck, 1999).

Palaute: XP korostaa palautteen tärkeyttä. Ilman palautetta, prosessi ei pysty toimimaan tehokkaasti. Palautetta voidaan saada asiakkaalta, tiimiltä tai itse tehtävältä järjestelmältä. Asiakaspalaute voi liittyä johonkin tehtyyn ominaisuuteen. Tiimiltä saatava palaute voi liittyä esimerkiksi arkkitehtuuriin ja järjestelmän antama palaute voi olla yksikkötestauksessa saatua palautetta ominaisuuden toimivuudesta. Palaute on vahvasti kytköksissä kommunikaation ja yksinkertaisuuden kanssa. Palaute on osa kommunikaatioita ja palautteellakin tulee pyrkiä yksinkertaisuuteen, koska palautteensaajan täytyy pystyä se ymmärtämään (Beck, 1999).

Rohkeus: Työskennellessä äärimmäisyyksien kanssa täytyy olla rohkeutta toteuttaa asioita. XP määrittää rohkeuden omaksi arvokseen ja se on edellytys useamman XP:n käytännön suorittamisessa ja päätöksen teossa. Rohkeutta on esimerkiksi tehdä päätös ohjelmakoodin refaktoroinnin tarpeesta ja refaktoroinnin tekemisestä sekä pysyä yksinkertaisimmassa ratkaisussa ja suunnittelussa. XP:n metodologiat vaativat tätä arvoa, jotta prosesseja voidaan noudattaa. XP antaa vastuuta yksilölle ja sitä johtuen yksilön tulee olla rohkea päätöksissään (Beck, 1999).

Kunnioitus: Beckin (2004) mukaan kunnioitus toisia kohtaan on tärkeää. Kunnioituksella tarkoitetaan esimerkiksi toisten tekemän työn kunnioittamista ja heidän tekemien päätösten hyväksymistä. Kunnioitus tulee koskea myös itseä, pyrkimys oman työnsä korkeaan laatuun ja omien ratkaisujen kunnioittaminen. Ilman kunnioitusta toisia tiimin jäseniä tai heidän tekemisiään kohtaan on projekti tuomittu epäonnistumaan (Beck, 1999).

2.3 XP:n käytännöt

XP sisältää 12 eri käytäntöä, suunnittelupeli, pienet julkaisut, metaforat, yksinkertainen suunnittelu, testaus, refaktorointi, pariohjelmointi, kollektiivinen omistus, jatkuvat integraatiot, 40 tunnin työviikko, läsnä oleva asiakas ja koodausstandardit. Näitä käytäntöjä käytetään XP projektin eri vaiheissa, siten että ne tukevat toinen toistaan ja kompensoivat toisen käytännön heikkouksia. Käytännöt ovat johdettu parhaista käytännöistä XP:n tarpeisiin. Mikäli käytäntöjä käytetään yksittäin, voi tällainen menettelytapa johtaa projektin epätasapainoon. XP:n menetelmien mukaan olisi suotavaa käyttää useampaa eri käytäntöä yhtä aikaa (Beck, 1999).

2.3.1 Suunnittelupeli

Suunnittelupelin tarkoitus on hahmottaa mitkä ominaisuudet tai toiminnallisuudet ovat mahdollisia ja mitkä haluttavia. Suunnittelupeliin osallistuu niin teknisiä kuin liiketoimintapuolen ihmisiä. Liiketoiminnasta vastaavien ihmisten täytyy pystyä päättämään asioista, kuten: laajuus, prioriteetti, julkaisujen sisältö ja julkaisujen ajankohdat. Ratkaisuja eivät tee pelkästään liiketoiminnan vastaavat vaan myös teknisen puolen osaajat. Teknisissä päätöksissä täytyy huomioida: aika arvio toteutuksen kestosta, teknisten ratkaisujen seuraukset, prosessin kulku ja yksityiskohtainen jaksotus (Beck, 1999).

Suunnittelupelin on tarkoitus yhdistää liiketoiminnan osa-alueet paremmin teknisten ratkaisujen kanssa ja tunnistaa tärkeät kehitysmahdollisuudet. Suunnittelupelin päämääränä on mahdollisimman korkea ohjelmiston arvo, mahdollisimman vähillä tuotannon kustannuksilla. Ohjelmiston arvoa nostaa sen sisältämät toiminnallisuudet ja kuinka kriittisiä nämä toiminnot ovat liiketoiminnalle eli asiakkaille itselleen. Suunnittelupelin parhaana pelaajana voi toimia kokenut järjestelmän käyttäjä. Kokemuksen kautta on tullut näkemys asioista ja toiminnoista, jotka ovat vitaaleja liiketoiminnan kannalta ja mitä asioita tulisi kehittää enemmän. Suunnittelupelin vaiheet ovat: tutkimus, sitoutuminen ja ohjaus. Tutkimusvaiheessa etsitään tehtäviä ominaisuuksia ja muutoksia, ominaisuuksien ja muutosten kartoittamisen jälkeen sitoudutaan tekemään valitut ominaisuudet tai muutokset ja tekemisen edetessä korjataan etenemisen suuntaa tarvittaessa (Beck, 1999).

Suunnittelupeli jakautuu kolmeen eri vaiheeseen. Ensimmäisessä vaiheessa asiakas esittää käyttötapaustarinat, jotka asiakas haluasi tehtävän seuraavassa iteraatioissa. Tiimi kyselee tarkentavia kysymyksiä tarvittaessa. Toisessa vaiheessa tiimi paloittelee saadut tarinat osiin ja määrittelevät, mitä järjestelmään implementointi vaatii. Jos ratkaisumalleja tulee useampia, tiimi valitsee yksinkertaisimman ratkaisun. Kolmannessa vaiheessa ohjelmoijat valitsevat tehtävät, jotka he aikovat suorittaa seuraavan iteraation aikana. Määritellyt tehtävät ovat kokonaisia tarinoita tai tarinan osia. Tehtävistä annetaan arvioita, jossa kerrotaan, kuinka kauan sen suorittamiseen menee aikaa (Jeffries et al. 2001).

2.3.2 Pienet julkaisut

Jokaisen julkaisun tulisi sisältää liiketoiminnan kannalta tärkeimmät toiminnot. XP suosii mahdollisimman pieniä julkaisuja, mutta kuitenkin siten, että toimitettava ominaisuudet ovat tehty valmiita ja ne käyvät järkeen (Beck, 1999). XP projektin kannalta on hyvin tärkeää että julkaisuja tehdään mahdollisimman usein. Tämä onnistuu pitämällä julkaisut pieninä. Jokaisen julkaisun jälkeen saadaan XP prosesseille tärkeää palautetta ja tietoa siitä, mitä asiakkaat oikeasti pitävät tärkeänä. Pienet julkaisut voivat tuntua mahdottomilta useissa tilanteissa mutta XP:n ajattelutapa pyrkii löytämään ratkaisuja siihen, kuinka julkaisuja voidaan tehdä usein ja ne ovat pieniä. Ajattelutapa korostaa yksinkertaisuutta, jolloin monimutkaisia rakenteita ei ole. Ohjelmiston osia voidaan esimerkiksi julkaista pieninä moduuleina tai uutta käyttöliittymää voidaan käyttää osana vanhan rinnalla kunnes se on kokonaan valmis (Jeffries et al. 2001).

2.3.3 Metaforat

Metaforalla tarkoitetaan selkeää kuvausta asiasta, joka tarjoaa mielikuvan jostakin asiasta. Metafora auttaa ymmärtämään asioita ja asiayhteyksiä sekä antaa niille konkreettisen merkityksen, tämä voi olla hyvinkin hedelmällistä ohjelmoijan ja asiakkaan välillä. Tämä sen takia, koska usein ohjelmoija on hyvin tekninen ihminen ja tekniset termit eivät usein sovi tai lisää ymmärrystä liiketoiminnan kannalta tärkeisiin asioihin. Metaforat ovat sovituspaloina eri termien välillä (Jeffries et al. 2001). Metaforilla voidaan kuvata järjestelmän arkkitehtuuria, ominaisuuksia, järjestelmän toimintoja ja ulkoasua jne. Beckin (1999) mukaan XP projektia ohjaa yksi, jopa naiivilta kuulostava metafora. Metafora käsitteenä korvaa osaksi arkkitehtuurikäsitteen XP projektissa ja sen tarkoitus on kuvata järjestelmää ja antaa siitä mielikuva, mikä on muutakin kuin viivoja ja laatikoita. Esimerkiksi arkkitehtuuria voitaisiin kuvata seitiksi. Seitti toimisi metaforana ja antaisi meille mielikuvan siitä, minkä näköinen se on ilman että sitä täytyy piirtää ja kuvata tarkasti. Kuitenkin metaforat tarvitsevat joskus selityksen, mikä tarkentaa kuvausta.

2.3.4 Yksinkertainen suunnittelu

XP pyrkii mahdollisimman yksinkertaiseen suunnitteluun. XP:n ajatuksena on se, että kaikki ylimääräinen karsitaan, mutta kuitenkin siten, että kaikki oleellinen jää jäljelle. Oleellisia ohjelman osia ovat: ohjelman täytyy suoriutua kaikista testeistä, ohjelma ei saa sisältää päällekkäistä logiikkaa, ohjelma ilmoittaa oleellisista aikomuksista ohjelmoijalle ja ohjelma sisältää mahdollisimman vähän luokkia ja metodeja (Beck, 1999). Yleisesti ottaen ohjelmistosuunnittelussa pyritään aina yksinkertaiseen ratkaisuun mutta yksinkertainen ratkaisu ei ole välttämättä helpoiten toteutettavissa. Yleensä pyritään miettimään tulevaisuuteen ja määrittelemään etukäteen, mitä asiakkaat mahdollisesti haluavat. XP:n ajattelutapa ei tue tätä ajatusta, vaan pyrkii suuntaamaan katseen ainoastaan tähän päivään. Tämä ajatusmalli perustuu siihen, että yksinkertaiseen suunnitteluun voidaan helposti lisätä uusia ominaisuuksia (Jeffries et al. 2001).

2.3.5 Testaus

XP vaatii automaattitestauksia ja metodologian mukaan yksikkötestit kirjoitetaan aina ennen kuin itse ohjelmalogiikka ohjelmoidaan. Myös asiakkaat kirjoittavat testitapauksia toiminnallisuuksille. Tämä auttaa luomaan luottamusta järjestelmää kohtaan ja valmistaa järjestelmää uusille muutoksille. XP testaa ainoastaan sellaiset osat, jotka pystyvät rikkoutumaan. Tämä tarkoittaa sitä, ettei jokaista metodia tarvitse välttämättä erikseen testata (Beck, 1999).

Hyväksymistestauksella varmistetaan tehdyn ominaisuuden vastaavuus asiakasvaatimukseen. Hyväksymistestauksen tekee aina asiakas, joka toimittaa testitapauksen ohjelmoijille jokaiseen iteraatioon. Hyväksymistestaus pyritään tekemään aina ennen toimitusta ja mielellään heti toiminnon valmistuttua. Tämä sen takia, koska ohjelmoijilla on tapana unohtaa asioita ja viivästyminen lisää unohtamista. XP arvostaa palautetta ja aikainen palaute järjestelmän toiminnasta katsotaan hyvin arvokkaaksi. Hyväksymistestaus testaa saadun käyttäjätarinan toteutusta. Tämä testi antaa palautteen asiakkaalle ja toiminnallisuuden tekijälle, heillä molemmilla on oikeus palautteeseen (Jeffries et al. 2001). Liu (2012) sanoo tutkimuksessaan, että hyväksymistestaus voi korjata huonon kommunikaation asiakkaan ja tiimin välillä.

2.3.6 Refaktorointi

Refaktorointi on prosessi, jonka tarkoituksena on käsitellä ohjelmistoa siten, ettei se muuta toimintaa mutta parantaa sisäistä rakennetta. Refaktorointi on kurinalainen tapa puhdistaa lähdekoodi ja minimoida virheiden mahdollisuus (Fowler, 1999). Lisätessään uutta ohjelmakoodia tekee ohjelmoija päätöksen tarvitseeko ohjelmakoodia refaktoroida eli muotoilla ohjelmalogiikka uudelleen. Refaktoroinnin tarve tulee, kun huomataan että toiminnallisuuden lisäämiseksi tulee tehdä päällekkäistä koodia. Refaktoroinnin tarkoituksena on pitää lähdekoodi mahdollisimman yksinkertaisena, jolloin uusien ominaisuuksien lisääminen on helpompaa (Beck, 1999).

Syy siihen, ettei koodia refaktoroida, voi olla pelko siitä, että rikotaan ohjelmalogiikkaa. Kuitenkin XP:n prosessin mukaisesti on käytettävissä yksikkötestit, joiden avulla voidaan varmistua, ettei mitään rikottu (Jeffries et al. 2001). Refaktorointi ei kuitenkaan saisi olla hallitseva osa projektia. Tietojärjestelmän arkkitehtuuriin tulisi kiinnittää huomioita projektin alussa, jolloin refaktoroinnin määrää saadaan vähennettyä tulevaisuudessa (Kivi et al. 2000). Refaktorointia voidaan tehdä myös ”legacy” tietojärjestelmien lähdekoodille. Rizvi (2011) esittää, että AOP (Aspect-Oriented Programming) metodologia voi olla siinä avuksi tai tehdä työstä helpompaa. ”Legacy” koodin refaktorointiprosessi lähtee motiivista, jonka jälkeen valitaan kaava, jolla refaktorointia lähdetään tekemään. Kaavan avulla etsitään lähdekoodista samankaltaisuuksia, joita voidaan refaktoroida kaavan mukaan.

Refaktorointitapoja on kahdenlaisia. Toinen on XP:n mukainen askel – askeleelta -tapa, tätä tapaa kutsutaan Floss Refaktoroinniksi (Floss Refactoring). Floss refaktorointia tehdään kokoajan, aina kun lisätään uutta toiminnallisuutta tai tehdään koodiin muutoksia. Tällä pyritään parantamaan koodin laatua ja rakennetta. Toinen tapa on Root Canal Refaktorointi (Root Canal Refactoring), joka eroaa floss refaktoroinnista siten, että kehitys pysäytetään ja keskitytään pelkästään refaktoroimaan koodia (Murphy-Jill, 2008; Liu, 2012).

2.3.7 Pariohjelmointi

Pariohjelmoinnilla tarkoitetaan kahden ohjelmoijan yhteistyötä samalla tietokoneella. XP metodologian mukaan kaikki tuotannon koodi kirjoitetaan pariohjelmoinnilla. Toinen käyttäjä keskittyy itse metodin kirjoittamiseen parhaalla mahdollisella tavalla ja toinen keskittyy enemmän strategiseen suunnitteluun, kuten: Onnistutaanko tällä menettely tavalla vai pitääkö sitä vaihtaa? Mitkä ovat testitapaukset, jotka eivät toimi vielä? Onko mitään tapaa yksinkertaistaa järjestelmää, jolloin koko ongelma häviäisi. Parit ovat dynaamisia, mikä tarkoittaa että paria vaihdellaan tarpeen mukaan (Beck, 1999).

Pari muodostuu kahdesta roolista, jotka ovat ajaja (driver) ja avustaja (partner). Driver nimensäkin mukaan on ajaja, jonka tehtävänä on kirjoittaa itse koodi. avustaja tarkkailee ja neuvoo ajajaa. Avustajan tehtävä ei ole vain neuvojen antaminen ja tässä tehtävässä tulee olla tahdikas. Avustaja tulee osata myös antaa ajajan tehdä työnsä eikä keskeyttää koko ajan tai syöttää ajajalle omaa strategiaansa. Pariohjelmointi vaatii myös totuttelua ja kaikista ei välttämättä ole siihen. Kuitenkin kaksi hyvin parina toimeentulevaa ohjelmoijaa voivat vastata jopa kolmea erillistä ohjelmoijaa (Jeffries et al. 2001).

Pariohjelmointi auttaa ymmärtämään paremmin laajoja kokonaisuuksia ja voivat tämän avulla suunnitella paremmin ohjelmakoodia eteenpäin. Samalla ominaisuuden oppii tuntemaan kaksi henkilöä kerralla, yhden sijaan. Pariohjelmoinnilla voidaan opettaa kokemattomia ohjelmoijia ja siirtää paremmin hiljaista tietoa. Hiljaista tietoa ovat esimerkiksi vanhemman ohjelmoijan kokemusperäinen tieto järjestelmän rakenteesta tai ohjelmointitekniikoista. Tämä on myös sosiaalinen prosessi, joka vähentää dokumentaation tarvetta. Tämä johtuu siitä, että useampi ohjelmoija ymmärtää tehdyn toiminnallisuuden heti teko vaiheessa, koska ovat itse olleet mukana (Kivi et al. 2000). Kivi et al. kertoo, että kyseisen projektin parinmuodostus ja pariohjelmointi ei onnistunut. Epäonnistuminen tapahtui sen takia, että projektin henkilöiden aikataulut eivät täsmänneet. Tällaiseen ongelmaan voidaan törmätä jos käytössä on liukuvat työajat.

Padberg (2003) tutki pariohjelmointia viiden eri mittarin avulla. Nämä mittarit olivat:

- Yhden ohjelmoijan tuottavuus
- Pariohjelmoinnin nopeusetu
- Virheiden esiintyminen
- Virheiden esiintyminen pariohjelmoinnilla
- Virheiden poistamiseen käytetty aika

Tutkimuksessa keskityttiin tutkimaan milloin pariohjelmointia olisi hyödyllistä käyttää ja milloin se ei ole välttämätöntä. Tutkimuksen päätelmä on se, että pariohjelmoinnilla voidaan saavuttaa etuja, kuten luottamusta liiketoiminnalta ja toiminnallisuuksien nopeammalla valmistumisella järjestelmään (Padberg, 2003).

XP:n pariohjelmointia on kritisoitu ja sitä ei yleisesti ottaen käytetä, vaikka organisaatiossa käytettäisiinkin muita XP:n menetelmiä. Copelandin artikkelissa James Gosling, varapresidentti ja tutkija Sun Microsystemsissa, sanoo että ”yritys käyttää joitakin XP:n tekniikoita kuten yksikkö- ja suorituskykytestaukseen mutta ovat jättäneen pariohjelmoinnin väliin” (Copeland, 2001). Samaa sanoo myös Robert L. Glass (2001) artikkelissaan ” Extreme Programming: The Good, the Bad, and the Bottom Line”, “En voisi kuvitellakaan keskustelevani parini kanssa kun toimin luovassa tilassa”. Pariohjelmointi vaatii tiettyä ihmistyyppiä, joka pystyy työskentelemään siten, että joku toinen neuvoo vieressä (Glass, 2001). Sillitti (2012) päättelee tutkimuksessaan, että pariohjelmointi auttaa ohjelmoijia keskittymään tuottaviin aktiviteetteihin. Pariohjelmointi auttaa myös ohjelmoijaa ja työparia keskittymään asiaan pitempiä aikoja kuin työskennellessään yksin.

2.3.8 Kollektiivinen omistus

Kollektiivisella omistajuudella tarkoitetaan sitä, että jokaisella, jolla on tarve tehdä muutoksia lähdekoodiin, on vapaa näin tekemään. Jokainen ohjelmoija omistaa koodin. Jokainen ohjelmoija on vastuussa kaikesta koodista ja he eivät tarvitse erillistä lupaa

muutoksien tekemiseen lähdekoodiin. XP:n mukaan, jokainen on vastuussa tekemisistään ja jokaisen tulisi tuntea jotakin kaikista osista koodia. Koodin tuntemuksessa auttaa aina pariohjelmointi, jolloin koodia on tekemässä aina kaksi ihmistä. Koodin tuntemus ei näin lepää vain yhden ihmisen varassa (Beck, 1999). Jokainen ohjelmoija omistaa koodin. Omistajan vastuulle kuuluu pitää huolta omistamastaan, näin ollen ohjelmoijan tulee katsoa lähdekoodiaan kuin omaansa. Ohjelmoijan tulee tehdä refaktorointia tarvittaessa, siirtää koodia tai muuttaa sitä. Ohjelmoija tekee toimenpiteitä, mitkä tekevät koodista paremman ja selkeämmän (Jeffries et al. 2001).

Monet ohjelmistoprojektit kärsivät siitä, ettei kukaan omista koodia. Tällä tarkoitetaan sitä, että ohjelmoijat tekevät muutoksia kaikkialle koodiin mutta ottamatta vastuuta teoistaan. Mikäli omistajuutta ei ole, se voi kärsiä seuraavanlaisista ongelmista:

- Huono tai puuttuva dokumentaatio
- Lähdekoodi on vaikeasti luettavaa, johtuen epäyhtenäisistä koodaus tavoista
- Epäjohdonmukaisuuksia käyttöliittymäsuunnittelussa
- Pitkiä virheenkorjausjaksoja, jossa samat virheet näyttävät esiintyvän toistuvasti
- Jatkuva aikataulusta myöhästyminen ilman näkyvää syytä
- Tiimit, joissa on suuri henkilövaihtuvuus, huono kommunikaatio ja kurinalattomat työskentelytavat

Kollektiivisen omistajuuden ero ei-omistajuuteen on se, että kollektiivinen omistajuudessa tiimi varmistaa järjestelmän yhtenäisyydestä kun siihen tehdään muutoksia. Yhtenäisyydellä tarkoitetaan ohjelman kokonaisuutta, jossa muutoksien tekeminen eri järjestelmän osiin ei riko niiden välillä olevaa rakennetta. Kollektiivinen omistajuus ei toimi ilman useampaa XP:n muuta käytäntöä. Nämä käytännöt ovat jatkuva integraatio, yksikkötestaukset ja vahvat koodausstandardit. Kollektiivinen omistajuus vaatii tiimiltä dynamiikkaa, hyvää kommunikaatiota ja yksilön vastuuta (Nordberg, 2003).

Nordbergin mukaan kollektiivisen omistajuuden hyviä puolia ovat:

- Automaattinen kommunikaation jakaminen toiminnallisuudesta
- Kannustaa parempaan tiimin kanssakäymiseen, yhteenkuuluvuuteen ja luovuuteen
- Vaatii ja vahvistaa yhtenäistä tyyliä ja filosofiaa läpi järjestelmän
- Hallitsee helposti tiimin kasvamisen tai pienenemisen
- Vastaa nopeasti ja tehokkaasti vaatimuksien lisääntymiseen tai muutoksiin

Kollektiivisen omistajuuden huonoja puolia ovat (Nordberg, 2003):

- Epätavallinen organisointi ja aikataulumalli voivat olla hankalia luoda ja hallita
- Vastuun jakaminen tehtäville ja ongelmille voi olla vaikeaa
- Tiimin skaalautuvuus on rajoitettu kymmenestä viiteentoista kehittäjään
- Potentiaalinen riski koodin satunnaisille muutoksille, johtuen erilaisista ohjelmointityyleistä
- Järjestelmältä voi puuttua kokonaisarkkitehtuuri tai – runko. Yhteinen suunta ja tarkoitus voi olla vaikeasti hahmotettavissa

2.3.9 Jatkuva integraatio

Koodia integroidaan jatkuvasti, aina muutaman tunnin ohjelmoinnin jälkeen tai niin usein kun mahdollista. Tuotettu koodi lisätään sen hetkiseen jakeluun ja aloitetaan testaus. Testit pitää saada suoritettua täydellisesti. Mikäli testejä ei saada läpi, täytyy muutokset hylätä ja palata aloituspisteeseen ennen integraatioita. Jatkuva integraatio voi kuulostaa työläältä, mutta se kannattaa. Mitä pitemmäksi integraatioiden välillä oleva aika kasvaa, sitä vaikeammaksi tulee virheen etsiminen uudesta koodista. On paljon helpompaa löytää virhe ja saada palaute hyväksymistestauksesta, kun testattava kokonaisuus ei ole kerralla suuri, vaan se tehdään lisäämällä vähän kerralla. Näin on selkeämmin nähtävillä, milloin ja missä vaiheessa virhe on muodostunut ja sitä kautta paikantaa virhe ja korjata se. (Beck, 1999; Jeffries et al. 2001).

Jatkuvaan integraatioon käytetään yleensä sitä varten tehtyä palvelinta. Palvelimelle syötetään koodi ja se käynnistyy joko automaattisesti tai käskystä. Palvelimelle voidaan syöttää useamman ihmisen tekemää koodia ja palvelimella oleva ohjelmisto kääntää koodin automaattisesti ja ajaa yksikkötestit. Jatkuva integraatio on toistoa, joten sitä varten palvelin toimii paremmin kuin ihmisen tekemä työ (Miller, 2008). Cannizzo (2008) kuvaa tutkimuksessaan, että jatkuvan integraatio ja sille osoitettu palvelin ovat suureksi hyödyksi. Automaattitesteillä pystytään ajamaan yksikkötestit, suorituskykytestit, skaalautuvuustestit ym. testit, mitkä katsotaan tarpeelliseksi.

2.3.10 40 tunnin työviikko

Beck (1999) korostaa 40 tunnin työviikon merkitystä. Jos projektissa täytyy tehdä paljon ylitöitä, on se merkki jostakin ongelmasta ja jotakin on jo lähtökohtaisesti pielessä. Jokaisella ihmisellä on omat rajansa mutta kukaan ei jaksa 60 tuntia useampaa viikkoa peräkkäin. Liiallinen ylityö kuluttaa innostuksen ja luovuuden. Tämän takia työn täytyisi olla säännöllistä. Tällä saavutetaan työntekijöiden innostus työhön. Jeffries et al (2001) viittaa omiin kokemuksiin projektissa, jossa ylityönä tehty projektin viimevaiheet osoittautuivat hankaliksi. Tämä johtui siitä, että tiimin jäsenet olivat väsyneitä jatkuvan ylityön takia ja se johti huonoihin ratkaisuihin, tekemättömiin testauksiin ja refaktoroinnin tekemättä jättämiseen.

2.3.11 Läsnäoleva asiakas

XP vaatii aina läsnäolevaa asiakasta, joka pystyy olemaan tiimin mukana. Asiakas pyrkii vastaamaan kysymyksiin, selvittämään haasteita ja asettamaan pienempiä tavoitteita. Oikealla asiakkaalla tarkoitetaan kyseisen järjestelmän käyttäjää, koska hänellä on näkemystä siitä, mitä oikeasti tarvitaan. Asiakkaalla on myös parempi liiketoiminnan tuntemus kuin pelkällä ohjelmointitiimillä. Huonona puolena läsnä olevassa asiakkaassa on se, jos projekti perutaan, menee myös läsnäolevan asiakkaan käyttämä aika täysin hukkaan. Asiakkaan antamat tarkennukset ja määritykset poistuvat lähdekoodin mukana (Beck, 1999).

Läsnäoleva asiakas auttaa parantamaan ohjelmoijan ja asiakkaan kommunikointia ja kokonaiskuvan ymmärtämistä. Kommunikointi paranee, kun ohjelmoijalla on mahdollista esittää kysymykset suoraan asiakkaalle suullisesti. Kommunikaatio voisi tapahtua myös esimerkiksi sähköpostin välityksellä mutta se olisi huomattavasti hitaampaa. XP:n tarkoitushan on olla mahdollisimman ketterä. XP:n vaatimukset perustuvat käyttäjätarinoihin ja ne eivät ole kovin kattavasti kirjoitettuja. Näin ollen läsnäoleva asiakas voi tarkentaa tarvittaessa tarinaa ja siten ominaisuudesta saadaan parempi ja kattavampi (Jeffries et al. 2001).

Asiakas ei voi aina olla läsnä. Tämä voi johtua siitä, ettei ole varaa irrottaa asiakasta hänen oikeasta työstään ja asettaa osaksi tiimiä. Syy voi olla myös se, että ohjelmointitiimi sijaitsee fyysisesti niin kaukana asiakkaasta, että läsnä oleva asiakas tulisi aivan liian kalliiksi. Jeffries et al. (2001) esittää muutamia seikkoja, jolla läsnä olevaa asiakasta voidaan korvata. Nämä ovat: Asiakas voidaan korvata siten, että joku henkilö joka tuntee aihealueen hyvin edustaa asiakasta. Yritä saada oikea asiakas vähintään suunnittelutapaamisiin. Vieraile asiakaan luona. Julkaise ohjelmisto tai järjestelmä usein. Varaudu väärinymmärryksiin. Läsnäolevalla asiakkaalla on tärkeä rooli antaa myös jatkuvaa ja välitöntä palautetta tiimille. Palautteen saaminen on XP prosessille kriittistä tietoa, joka auttaa pitämään suunnan oikeana (Kivi et al. 2000).

2.3.12 Koodausstandardit

Koodausstandardit ovat tärkeitä XP:n toimivuuden kannalta. Tiimillä tulee olla sellainen tai sellaiset, koska XP:n käytännöt vaativat standardin toimiakseen. Lähdekoodista ei pitäisi pystyä erottamaan useampaa kuin yhden tyylistä ohjelmakoodia. Tämä on tärkeää sen takia, koska jokaisella tiimin jäsenellä on oikeus muuttaa ja refaktoroida koodia. Kun koodi on yhtenäistä, se auttaa edellä mainituissa asioissa huomattavasti (Beck, 1999). Bass (2012) sanoo tutkimuksessaan Agile – menetelmien käytöstä ohjelmistoprojekteissa että koodausstandardit nähtiin välttämättöminä laajoissa ja jakautuneissa projektitiimeissä.

Koodausstandardi voi tuntua hitaalta aluksi mutta se maksaa vaivan vähän ajan päästä. Standardin omaksuminen voi kestää jonkin aikaa, mutta siihen tottuu ajan kanssa ja loppujen lopuksi se muotoutuu ohjelmoijan uudeksi omaksi tyylikseen. Muutamia hyödyllisiä avainasioita koodaus standardeissa ovat: Sisennykset, kapitalisointi, kommentointi, metodin koko ja nimeäminen. Näiden tarkoitus on selkeyttää koodia ja auttaa ohjelmoijia ymmärtämään koodia paremmin. Nimeämisellä tarkoitetaan luokkien, metodien, muuttujien ym. ohjelmointikielen nimeämistä vaativia osia. XP:ssä korostetaan metaforia ja niitä tulisi käyttää myös nimeämisessä. Jos esimerkiksi luokka nimetään selkeällä ja kuvaavalla metaforalla, auttaa se kaikkia hahmottamaan, mitä kyseinen luokka tekee (Jeffries et al. 2001).

2.4 XP osana tietojärjestelmien ylläpitovaiheen prosessia

Agile - menetelmät ovat vallanneet alaa ohjelmoinnin saralla hyvin paljon. Näitä menetelmiä käytetään hyvin paljon ja tämän hetken suosituin menetelmä on SCRUM, joka polveutuu osakseen XPstä (Sutherland, 2005). Sutherland (2005) mainitsee, että SCRUM on lähtöisin Takeuchi and Nonakan tutkimuksesta ”The New Product Development Game”. Tietojärjestelmien ylläpito sisältää erilaisia tehtäviä, kuten osia tietojärjestelmien kehityksestä. Ylläpitovaiheen tehtävät ovat yleensä pienempiä ja osaset rakennetaan valmiiseen runkoon. Näin ei tietenkään välttämättä aina ole ja suurempiakin projekteja voidaan tehdä vielä ylläpitovaiheessa. Erilliset projektit voivat kuitenkin käyttää eri menetelmiä ja ne eivät välttämättä ole kovin paljoa sidoksissa ylläpitovaiheeseen, jos projekteissa käytetään perinteisempiä ohjelmistokehityksen menetelmiä kuten esimerkiksi vesiputousmallia, jossa käyttöönotto tulee vasta projektin lopussa.

Choudhari ja Suman (2010) esittää iteratiivisen ylläpitovaiheen elinkaarimallin käyttäen hyödyksi XP:n menetelmiä. Kyseinen malli koostuu seitsemästä eri vaiheesta, jotka ovat: Analyysi, suunnittelu, muutoksen suunnittelu, muutoksen implementointi, regressio/järjestelmätestaus, hyväksymistestaus ja toimitus. Choudhari ja Suman esittää myös kaksi erillistä järjestelmää, mitkä laukaisevat muutokset. Ensimmäinen on uusien

vaatimusten järjestelmä ja toinen on virheidenseurantajärjestelmä. Uusien vaatimusten tietojärjestelmä sisältää kaikki kehitysideat ja asiakkailta tulleet uudet vaatimukset. Virheiden seurantatietojärjestelmään kerätään kaikki järjestelmästä havaitut virheet.

Analyysivaiheessa uudet kehitysideat saapuvat asiakkailta tarinoina. Nämä tarinat tai kertomukset ovat lyhyitä kuvauksia halutusta ominaisuudesta ja ne ovat korkeantason kuvauksia. XP:n mukaiset tarinat antavat kuvan kehittäjälle mihin moduuleihin tai järjestelmän osiin muutos mahdollisesti vaikuttaa, karkean arvion muutoksen kustannuksista ja sen laajuudesta. Suunnitteluvaiheessa päätetään mukaan otettavat tarinat, tehdään julkaisusuunnitelma ja tehdään selväksi että tiimin jäsenet ovat tietoisia ympärillä tapahtuvista muutoksista. Suunnitteluvaihe XP:n mukaan on hyvin lyhyt ja lopputuloksena on julkaistava versio järjestelmästä, joka voidaan periaatteessa toimittaa asiakkaalle. Choundharin ja Sumanin mukaan näitä kahta vaihetta seuraa muutoksen suunnittelu, joka käsittää kaiken olemassa olevan järjestelmän dokumentaatioista, lähdekoodista ja tietokannasta. Nämä käydään läpi ja pyritään löytämään paras mahdollinen tapa tehdä muutos.

Muutosten implementointi tapahtuu käyttäen XP:n tarjoamia ohjelmointimenetelmiä. Choundhari ja Suman viittaa tutkimuksessaan pariohjelmointiin ja sen käyttämiseen tässä vaiheessa. Pari ohjelmointi ei ole nostanut juurikaan suosiota (Copeland, 2001) mutta XP:n käytännöt määrittävät sen osaksi prosessia. Implementointivaiheessa suuri merkitys on kehittäjällä, joka implementoi kyseisen tarinan tietojärjestelmään. Kuten XP:n määrittelyssäkin kuvataan, käyttääkseen tätä XP:n menetelmiä, täytyy tiimin olla ammattitaitoisia ja kokeneita työntekijöitä. Tässä pariohjelmoinnin hyöty tulee esille, kun opetetaan uutta työntekijää. Uusi työntekijä pääsee näkemään ja tutustumaan järjestelmän rakenteeseen ja standardeihin helposti pari ohjelmoinnin avulla. Vanhempi ohjelmoija neuvoo vieressä ja tällä menetelmällä uusi työntekijä pääsee nopeasti sisälle tiimiin (Poole, Murphy, Huisman & Higgins, 2001).

Choundhari ja Suman painottavat, että kollektiivinen omistajuus tuotteesta on tärkeä osa motivoimaan kehittäjiä. Kehittäjien motivaatio on tärkeässä roolissa aina kun tehdään tietojärjestelmäkehitystä. Hyvin motivoituneet kehittäjät haluavat tehdä hyvää jälkeä ja he haluavat, että tehdystä työstä ei tule jälkipuheita. Kehittäjien moraalin tärkeydestä puhuvat myös Poole, Murphy, Huisman & Higgins, (2001) tutkimuksessaan Extreme Maintenance. Heidän tutkimus on tapaustyyppinen, missä havainnot perustuvat olemassa olevan yrityksen tiimiin. XP tarjoaa paremman tuen tiimissä mukana oleville kehittäjille, mahdollisuuden vaikuttaa asioihin, tarjoaa enemmän vastuuta ja motivoi kehittäjiä tekemään asioita hyvin.

Testausvaiheessa on regressio/järjestelmätestaus ja hyväksymistestaus. Ensimmäisessä vaiheessa testataan koko järjestelmä läpi että kokonaisuus on toimintakuntoinen ja että muutokset eivät ole vaikuttaneet kokonaisuuteen negatiivisesti. Hyväksymistestauksessa käyttäjätarinat käydään läpi ja tarkastellaan palveleeko tehty muutos tarinaa. Koko kehitys tapahtuu XP:n testaus painotteisen kehityksen avulla, jolloin testaus tapahtuu koko ajan kehityksen yhteydessä. Viimeinen vaihe on muutoksien toimitus asiakkaalle, mistä seuraa käyttöönottoasennukset ja mahdolliset käyttöönottokoulutukset.

2.5 XP tietojärjestelmien ylläpidossa

Edellisen kappaleen ylläpitovaiheen prosessin läpikäynnin jälkeen tarkastellaan asiaa olemassa olevien ongelmien kautta. Poole, Murphy, Huisman ja Higgins (2001) käsittelivät tutkimuksessaan ylläpitovaiheen ongelmia olemassa olevasta tuotteesta.

Suurimmat ongelmat näyttivät olevan ylläpitäjien ja kehittäjien motivaatio, epäyhtenäiset prosessit, ohjelmointistandardin puuttuminen, testaus ja läpinäkyvyys. Ylläpitovaiheessa työ koostuu pääasiassa pienistä kehitystöistä ja virheen korjauksesta. XP:n määrityksen mukaan se pyrkii vähentämään vaatimusmäärittelyä ja laajoja dokumentaatioita.

Dokumentaation supistaminen kuulostaa epäsopivalta yhdistettynä tietojärjestelmien ylläpitovaiheeseen. Tämä sen takia, että on pidetty tärkeänä tuottaa tarkat dokumentaatiot, joiden avulla tietojärjestelmää pystytään tulevaisuudessa hallitsemaan ja ymmärtämään. XP:tä ei tule soveltaa edellä mainittuun tapaan, vaan sillä tarkoitetaan vaatimusten ja virheiden dokumentoinnin tarkkuutta. XP:n tarinat pyrkivät juuri vähentämään turhaa dokumentaatiota ja pyrkivät siihen, että asiat ovat kuvattu lyhyesti ja ytimekkäästi, mutta kuitenkin siten, että se on ymmärrettävässä muodossa. Tämä tietenkin vaatii sen, että asiakkaat ja kehittäjät tai ylläpitäjät puhuvat samaa kieltä ja ymmärtävät toisiaan. XP korostaakin kommunikaation merkitystä, mikä auttaa ymmärtämään paremmin kaikkia osapuolia ja vähentämään väärinkäsityksiä (Beck, 1999).

Poole, Murphy, Huisman ja Higgins (2001) kertovat tutkimuksessaan pariohjelmoinnin hyödyistä, viitaten sen mahdollistamaan keskusteluun tiimin jäsenten kesken ja parantamalla opetusta nuoremmille kehittäjille. Pariohjelmointi on jakanut mielipiteitä eri tutkimusten kesken. Osa pitää asiaa hyvänä ja osa ei pystyisi kuvittelemaan toimivansa tällä tavoin. Lienee hyvin yksilöllistä, kuinka hyvin kyseinen XP:n käytäntö toimii. Selvää on kuitenkin, että se parantaa keskustelua ja vuorovaikutusta jäsenten kesken mutta epäselväksi jää vähentääkö se vanhemman kehittäjän tuottavuutta ja parantaa enemmän nuoremman kehittäjän taitoja. Pariohjelmointi ja tiimityöskentelyn parantaminen vaatii ponnisteluja myös tarjolla olevien tilojen suhteen. Pariohjelmointia voi tietenkin harrastaa pienimmässäkin huoneessa, mutta kuten Poole et al. mainitsivat, on suuremman yhteisön mahdollistama tuki tärkeämpää.

Poole, Murphy, Huisman ja Higgins korostavat nopean palautteen ja refaktoroinnin merkitystä tietojärjestelmän ylläpidossa. Nopea palaute on tärkeää ketteryyden kannalta. Nopeutta hidastaa huomattavasti jos joudutaan odottelemaan vastausta johonkin yksinkertaiseen kysymykseen. Tämä on usein pullonkaulana vanhoissa, ei niin ketterissä menetelmissä. Jokin asia vaatii johtajan suostumuksen vaikka asia ei periaatteessa ole mitenkään kiinni siitä, mitä johtaja tasolta vastataan, mikäli kyseessä on kehittäjille kuuluva asia. Tällä tarkoitetaan sitä, että organisaation ei saa olla liian byrokraattinen tälle menetelmälle, koska silloin menetelmä menettää merkityksen. Ketteryys perustuu paljon XP:ssä yksilön vastuuseen tuotteesta ja siihen tekemästään muutoksesta tai korjauksesta. Kuten XP:n menetelmä määrittelee, on jatkuva parantaminen järjestelmäkehityksessä tärkeää.

Refaktoroinnilla saavutetaan hyviä tuloksia järjestelmän ylläpidon kannalta. Järjestelmän lähdekoodia saadaan supistettua huomattavasti, mikä vähentää ylläpidettävän ohjelmakoodin määrää. Tämä luonnollisesti supistaa virheiden mahdollisuutta koodissa. Refaktorointi ei välttämättä sovellu lyhyisiin virheenkorjaus- tai kehitystehtäviin. Refaktorointi on kuitenkin tehokas tapa tehdä olemassa olevasta koodista tehokkaampaa ja helpommin ylläpidettävää. Kuitenkin, kuten Poole, Murphy, Huisman ja Higginsin tapauksessa, suurin osa kokeneista kehittäjistä oli siirtynyt muihin tehtäviin ja ylläpitotehtävät oli jäänyt kokemattomille kehittäjille. Tämä luonnollisesti luo oman haasteensa refaktoroinnille ja yleensäkin XP:n käyttämiselle ylläpidossa. Jos tilanne on se, ettei kokeneita ihmisiä löydy ylläpitotiimistä, liikutaan yleensä riskirajoilla.

Swanson (1976) esitteli kolme ohjelmiston muutoskategoriaa: i) Parannus: järjestelmän ongelmatilanteiden korjaus datan prosessoinnissa, suorituskyvyssä tai implementoinnissa. Tähän vaatimukseen XP pystyy vastaamaan varsin hyvin edellä mainittuihin tilanteisiin. Tietojärjestelmää harvoin lähdetään parantamaan ilman asiakkaalta tulevaa pyyntöä, tämä on tietenkin riippuvainen järjestelmän elinkaaren vaiheesta. Mikäli järjestelmä ei ole enää ns. kaupallinen, on sen omaehtoinen parantaminen melko vähäistä. Tämä tietenkään ei ole ongelma, jos koodia ollaan elinkaaren alkuvaiheesta asti refaktoroitu ja kehitetty paremmaksi. Tämä tila tietenkin saavutetaan vain siinä tapauksessa, että edellä mainittuja menetelmiä on käytetty tai ainakin osaa niistä. Dekleva (1992) mainitsee kuitenkin sen, että modernit menetelmät eivät lisää järjestelmän ylläpidettävyyttä, vaikka käytettäisiin moderneja menetelmiä.

ii) Mukautuva: vastataan ennakoituihin muutoksiin datan käsittelyssä. Muutos voi tulla asiakkaalta tai laki voi määrätä jonkin asetuksen, joka vaikuttaa suoraan järjestelmän toimintaan. Tällaisia asetuksia voi esimerkiksi olla laskutuksessa olevat ja esimerkiksi siihen liittyvät perintämääräykset ja -asetukset. Kun muutos on saatu vastaan joltakin taholta, voidaan se nostaa mukaan XP:n prosessiin. Asetukset ja määräykset eivät välttämättä tule suoraan XP:n mukaisina tarinoina ja metaforina, vaan ne joudutaan ensin tekemään ennen kuin ne voidaan siirtää kehittäjille. Asetuksien ja määräysten ymmärtäminen voikin olla ohjelmoijalle helpompaa, jos niistä saadaan luotua kuvaavia metaforia.

iii) Havainnointi: parannetaan järjestelmän tehokkuutta, suorituskykyä tai ylläpidettävyyttä. Asiakkaalta tai järjestelmän kehittäjältä tulee havainto mahdollisesta järjestelmän ongelmasta. Tähänkin kohtaan voidaan soveltaa hyvin XP:n tarjoamaa refaktorointia. Refaktorointi ei välttämättä auta havaitsemaan tällaisia kohteita, mutta sen vaikutus koodiin on huomattava, mikä tekee tästä osa-alueesta helpommin hallittavan. XP:n korostama refaktorointi nouseekin usein esille ylläpitovaiheen tehtävissä. Silti se on aika vähän käytetty menetelmä ylläpitovaiheen päivittäisissä rutiineissa. Kivi et al. (2000) kritisoikin refaktorointia siitä, että se voi muodostua vallitsevaksi osaksi työtä. Tämä ei ole se, mihin refaktoroinnin käytöllä halutaan pyrkiä. Kivi et al. mainitseekin, että alussa olisi hyvä käyttää aikaa kokonaissuunnitteluun, jolloin refaktoroinnin määrää ja tarvetta saadaan vähennettyä

3. ITIL – menetelmät

Tässä luvussa kuvataan ITIL – menetelmät yleisesti ja perehdytään tarkemmin ylläpitovaiheessa oleviin tehtäviin. Ylläpitovaiheen tärkeimmät tehtävät ovat muutosten- ja virheiden hallinta sekä järjestelmän pitäminen toimintakunnossa.

3.1 Yleiskuvaus

ITIL (Information Technology Infrastructure Library) – menetelmät ovat tarkoitettu IT–palvelujen hallintaan. IT–palveluilla tarkoitetaan kaikkia sellaisia palveluita, joissa IT on osana toimintaa. Information technology (IT) on terminä paljon käytetty ja sen merkitys muuttuu sisällön mukaan. IT – termiä voidaan tarkastella neljästä eri perspektiivistä, jotka ovat:

- IT - järjestelmä, ohjelmat ja infrastruktuuri ovat suuremman tuotteen komponentteja. Ne mahdollistavat tai sisältyvät prosesseihin ja palveluihin
- IT on organisaatio, jolla on omat kyvykkyydet ja resurssit. IT-organisaatioita voi olla usean erityyppisiä, kuten liiketoiminta funktiot, jaetut palveluyksiköt ja yritystason ydin yksiköt
- IT on liiketoiminnan käyttämä palveluiden kategoria. Tyypillisesti se sisältää ohjelmistoja tai infrastruktuuria, jotka on pakattu ja tarjotaan palveluna, joko sisäisenä IT organisaatioina tai ulkoisena palvelun tarjoajana. IT-kulut katsotaan liiketoiminnan kuluiksi
- IT on liiketoiminnan käyttämä palveluiden kategoria, joka tarjoaa hyötyä sen omistajille. IT-kulut katsotaan sijoitukseksi

ITIL – menetelmät ovat viitekehys palvelujen hallinnalle. Viitekehys on koottu parhaista käytännöistä. ITIL koostuu viidestä eri ytimen osasta. Nämä ydinosat ovat: palvelustrategia, palvelusuunnittelu, palvelumuutos, palveluoperaatio ja jatkuva palvelun kehittäminen. Jokainen osa sisältää sen tarvitsemia prosesseja (OGC, Service strategy, 2007; Bon, 2009). Nämä prosessit on kuvattu taulukossa 1.

Palvelustrategia: Strategialla tarkoitetaan suunnitelmaa jonkin päämäärän saavuttamiseksi. Palvelustrategian tarkoituksena on tukea liiketoiminnan strategiaa ja luoda pohja tuleville ja olemassa oleville palveluille. Päätarkoituksena on operoida ja kasvattaa palveluita menestyksekkäästi. Palvelustrategia määrittelee, miksi jotakin kannattaa tehdä. Strategia pyrkii etsimään uusia kohteita tai asiakkaita palveluille ja mitä palveluja asiakkaat voisivat tarvita. Palvelustrategia pyrkii myös vastaamaan kysymyksiin, kuinka luodaan asiakkaille lisäarvoa tai miten määritetään palvelunlaatu. Strategialla pyritään saavuttamaan aina jotakin, ja sen määrittäminen muodostuu perimmäisestä päämäärästä. Palvelustrategian päämäärä on tuottaa lisäarvoa asiakkaalle. Lisäarvoa tuottaa palveluiden hyöty ja takuu siitä, että palvelu vastaa vaatimuksia. Jotta lisäarvoa voidaan tuottaa, tarvitsee yrityksellä olla kyvykkyyksiä ja resursseja sen tuottamiseen. Arvon määrittää aina asiakas. Arvo voi olla pelkästään taloudellista mutta se tulee kuitenkin määritellä kolmeen eri osa-alueeseen, mitkä ovat saavutetut liiketoimintatulokset, asiakkaan mieltymykset ja asiakkaan mielikuvat.

Taulukko 1. ITIL – elinkaaren vaiheet ja prosessit (OGC, Service strategy, 2007; Bon, 2009).

Palvelustrategia	**Palvelu-suunnittelu**	**Palvelutransitio**	**Palvelutuotanto**	**Jatk. palvelun parantaminen**
IT – palvelujen strategianhallinta	Suunnittelun koordinointi	Transition suunnittelu ja tuki	Herätteiden-hallinta	7 Askeleen kehittämisprosessi
Palvelusalkun hallinta	Palveluluettelon hallinta	Muutoksenhallinta	Häiriönhallinta	
Taloudenhallinta	Palveluntason-hallinta	Palveluomaisuuden ja konfiguraationhallinta	Palvelupyyntö-prosessi	
Kysynnän hallinta	Saatavuuden-hallinta	Jakelun- ja käyttöönotonhallinta	Ongelman-hallinta	
Liiketoiminta-suhteiden hallinta	Kapasiteetin-hallinta	Palvelun validointi ja testaus	Pääsynhallinta	
	IT-palvelun jatkuvuuden-hallinta	Muutoksen evaluointi		
	Tietoturvan hallinta	Tietämyksenhallinta		
	Toimittaja-hallinta			

ITIL:in palvelustrategia sisältää IT–palvelujen strategianhallinnan, palvelusalkun hallinnan, taloudenhallinnan, kysynnän hallinnan ja liiketoimintasuhteiden hallinnan. Palvelusalkunhallinnalla tarkoitetaan yrityksen kaikkien palveluiden kokonaishallintaa, mikä pyrkii riskien ja kustannusten hallintaan ja lisäarvon maksimointiin. Tämä saavutetaan, kun omistetaan palvelusalkku, millä voidaan vastata asiakkaiden tarpeisiin. Taloudenhallinnalla pyritään tuottamaan tietoa palveluiden kustannuksista ja tuotoista, mitkä tukevat strategisia päätöksiä. Samalla se varmistaa rahoituksen, jolla pysytään suunnittelemaan, kehittämään ja hallitsemaan palveluita. Kysynnän hallinnalla pyritään hallitsemaan kysyntä ja tarjonta ja ennustamaan kysyntä ja tasapainottamaan tarvittavat resurssin sen mukaan (OGC, Service strategy, 2007; Bon, 2009).

Palvelusuunnittelu: Tarkoituksena on suunnitella IT–palveluita, joita tarvitaan strategian täyttämiseen. Palvelusuunnittelu sisältää hallintamenettelyt, -prosessit ja -politiikat, millä pyritään mahdollistamaan strategiaan pääseminen. Palvelusuunnittelun tavoitteena on suunnitella IT–palveluita, mitkä pystyvät tarjoamaan asiakkailleen korkeantason palveluita. Palvelunsuunnittelulla pyritään tekemään vaikuttavia palveluita, joita ei tarvitse paljon muuttaa palvelunelinkaaren aikana. Palvelunsuunnittelu tuottaa strategian määrittämille päämäärille ratkaisuja. Suunnitteluvaihteen tulee kattaa viisi eri näkökulmaa.

- Palveluratkaisujen suunnittelu
- Palvelusalkun suunnittelu
- Arkkitehtuuri- ja teknologiasuunnittelu
- Prosessien suunnittelu

- Mittausjärjestelmien ja mittareiden suunnittelu

Palvelusuunnittelu sisältää kahdeksan eri prosessia, mitkä ovat: suunnittelun koordinointi, palvelutasonhallinta, palveluluettelon hallinta, saatavuudenhallinta, tietoturvan hallinta, toimittajahallinta, kapasiteetinhallinta ja IT–palvelun jatkuvuuden hallinta. Koordinoinnilla varmistetaan se, että palvelusuunnittelun päämäärät ja tavoitteet saavutetaan. Palvelutasonhallinnalla varmistetaan, että palvelut tuotetaan palvelutasotavoitteiden vaatimalla tasolla. Palveluluettelo sisältää tiedon kaikista tuotannossa ja kehitteillä olevista palveluista. Luettelo asettaa informaation saataville kaikille sitä tarvitseville osapuolille. Saatavuudenhallinta pyrkii varmistamaan sen, että palvelu on saatavilla ja vastaa sovittua tasoa. Toimittajanhallinnalla tarkoitetaan organisaation sisäisiä tai ulkoisia toimijoita ja sen tarkoitus on varmistaa että sopimukset toimittajien kanssa tukevat liiketoiminnan tarpeita (OGC, Service design, 2007; Bon, 2009).

Palvelutransitio: Tarkoituksena on varmistaa, että uudet, muutetut tai poistuvat palvelut vastaavat strategiaa sekä palvelusuunnittelussa kartoitettuja liiketoiminnan vaatimuksia. Palvelutransitiossa tärkeimpiä tehtäviä ovat palvelun rakentaminen, testaaminen, pilotointi, käyttöönoton suunnittelu ja valmistelu, käyttöönotto sekä käytöstä poistaminen. Tämän ITIL–elinkaaren vaiheen tarkoitus on tehdä itse suorittava työ, jonka tuloksena on uusi palvelu, sen muuttaminen tai sen käytöstä poistaminen.

Palvelutransitio sisältää seitsemän eri prosessia, mitkä ovat: transition suunnittelu ja tuki, muutoksenhallinta, palveluomaisuuden ja konfiguraationhallinta, jakelun- ja käyttöönoton hallinta, palvelun validointi ja testaus, muutoksen evaluointi ja tietämyksenhallinta. Transition suunnittelu ja tuki vastaa transition kokonaiskuvasta ja mahdollisesti useiden yhtäaikaisten transitioiden koordinoimisesta. Palvelutransitio luo perustan palvelun tekemiselle, ei kuitenkaan itsessään tee yksittäisiä muutoksia. Perustana käsitetään arkkitehtuurit, prosessit, hallintajärjestelmät ja työvälineet. Muutoksenhallinta vastaa kaikkien muutosten hallinnasta palveluihin. Tavoitteena on ohjata kaikki hyödylliset muutokset palveluihin, joiden avulla parannetaan palvelua tai muutetaan vastaamaan paremmin liiketoiminnan vaatimuksia. Muutoksenhallinnassa on muutoskomitea CAB (Change Advisory Board) ja ECAB (Emergency Advisory Board), näiden tehtävänä on muutosten arviointi, priorisointi, aikatauluttaminen ja hyväksyminen. ECAB on tarkoitettu hätätilanteita varten, jolloin apua on saatava nopeasti. Palveluomaisuuden ja konfiguraationhallinta vastaavat palveluomaisuuden yksityiskohtaisista tiedoista. Palveluomaisuutta ovat kaikki resurssit tai kyvykkyydet, jotka voivat edistää palvelun tuottamista. Tietämyksenhallinta vastaa olemassa olevien näkökulmien ja kokemusten saatavuudesta kun niille on tarvetta. Se pyrkii vähentämään tietämyksen uudelleenluonnin tarvetta (OGC, Service transition, 2007; Bon, 2009).

Palvelutuotanto: Tämän elinkaaren vaiheen tehtävänä on toteuttaa ja koordinoida prosessit, joita tarvitaan palvelun päivittäisessä toiminnassa. Tavoitteena on ylläpitää sovittua palveluntasoa ja luoda luottamusta ja tyytyväisyyttä asiakkaalle palvelusta. Palvelutuotanto toimii suorana rajapintana asiakkaisiin ja tuottaa tärkeää tietoa muille ITIL–prosesseille. Palveluntuotannossa yhdistyy ihmiset, teknologia, tuotettavat palvelut ja niiden prosessit. Palveluntuotanto voi olla joko proaktiivista tai reaktiivista, yleensä sopiva sekoitus molempia. Tämä määrittää, onko palveluntuotanto aktiivinen kaikissa toiminnoissaan vai vastaako se vasta ulkoa tuleviin ärsykkeisiin.

Palveluntuotanto sisältää viisi eri prosessia, mitkä ovat herätteidenhallinta, häiriönhallinta, palvelupyyntöprosessi, ongelmanhallinta ja pääsynhallinta.

Herätteidenhallinta vastaa IT–infrastruktuurin valvomisesta ja sen normaalista toiminnasta. Herätteet ovat tilan muutoksia, jotka voivat aiheuttaa eri tason herätteitä. Tilat ovat informatiivinen, varoitus ja poikkeama. Herätteidenhallinta vaatii automatisointia ja heräte voi laueta esimerkiksi jonkin raja-arvon ylittymisestä. Tällainen voi olla esimerkiksi levytilan täyttyminen yli määritetyn raja-arvon. Tästä syntyy automaattinen heräte, johon tulee reagoida. Häiriönhallinnan tavoitteena on palauttaa palvelun normaalitoiminta mahdollisimman nopeasti. Herätteestä voi muodostua häiriö, kun heräte käsitellään ja eskaloidaan tarvittaessa. Häiriöllä tarkoitetaan IT–palvelun suunnittelematonta keskeytystä. Häiriöt yleensä priorisoidaan jollakin tasolla ja niihin reagoidaan priorisoinnin vaatimalla herkkyydellä. Palvelunpyyntöprosessi vastaa kaikista käyttäjien pyytämistä palvelupyynnöistä. Palvelupyyntö voi olla mitä tahansa. Näin ollen organisaatioiden pitää määritellä, mitä otetaan vastaan kuhunkin rajapintaan. Palvelupyyntöjen vastaanotto ja käsittely voi olla myös osana toista prosessia. Ongelmienhallinta pyrkii estämään ongelmien muodostumista ja niiden aiheuttamia häiriöitä. Ennakoiva ongelmanhallinta pyrkii estämään tunnistamaan ongelman ennen kuin ne aiheuttavat häiriöitä palvelussa. Pääsynhallinta varmistaa pääsyn palveluihin ja toteuttaa olemassa olevaa tietoturvapolitiikkaa.

Palvelunhallinta sisältää myös neljä eri funktiota. Funktioita ovat palvelupiste, tekninen hallinta, sovellushallinta ja IT–käyttöpalvelun hallinta. Palvelupiste on rajapintana asiakkaaseen ja sen tavoitteena on vastata häiriöistä ja palvelupyynnöistä. Tekninen hallinta vastaa IT–infrastruktuurin teknisestä tietämyksestä ja sovellushallinta sovellukseen liittyvistä erikoisosaamisesta. IT-käyttöpalvelun hallinta varmistaa, että palvelu toimii sille määritetyn suorituskyvyn mukaan ja ylläpitää sen nykytilaa (OGC, Service operation, 2007; Bon, 2009).

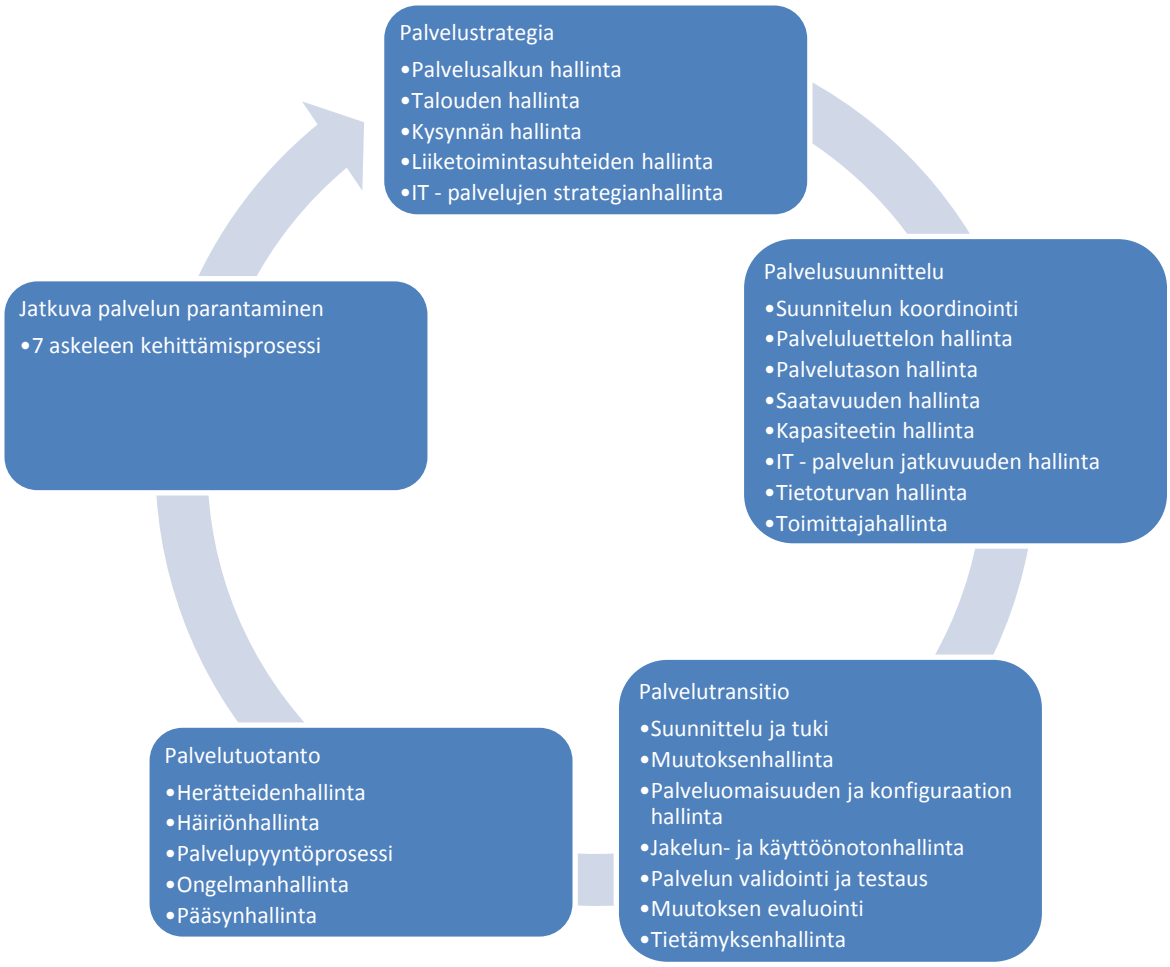
Jatkuva palvelun kehittäminen: Tarkoituksena on varmistaa IT–palveluiden vastaavuus liiketoiminnan tarpeisiin ja tunnistaa muuttuvat tarpeet sekä tekemällä parannuksia olemassa oleviin palveluihin. Palvelun kehittämisessä on tärkeää määrittää perustaso, jossa palvelu on tällä hetkellä. ITIL–menetelmien mukaan palvelun kehittäminen perustuu mittareihin, joilla pystytään mittamaan nykytilaa ja kehitysidean implementoinnin jälkeistä tilaa. Mittareita on kolme, mitkä ovat: teknologiamittari, joka mittaa sovellusten suorituskykyä ja saatavuutta. Prosessimittari, joka mittaa palvelunhallin prosessien suorituskykyä. Kolmas mittari on palvelumittari, joka mittaa palvelua päästä päähän eli koko palvelua. Tällä pystytään varmistumaan siitä, että kehitys on tuottanut tulosta joko negatiiviseen tai positiiviseen suuntaan. Jatkuva palvelun kehitys koskee jokaista ITIL–elinkaarenvaihetta ja pyrkii tuottamaan kehitysideoita jokaiselle vaiheelle.

ITIL kuvaa jatkuvan palvelun kehittämiseen seitsemän askeleen kehitysprosessin. Nämä kehitysaskeleet ovat:

1. Määrittele mitä pitäisi mitata
2. Määrittele mitä voidaan mitata
3. Tiedon keräys
4. Tiedon käsittely
5. Tiedon analysointi
6. Informaation esittäminen ja käyttäminen
7. Implementoidaan kehitystoimenpiteet

Prosessissa määritellään kriittiset menestystekijät, joiden mukaan prosesseille voidaan määrittää keskeiset suorituskykymittarit. Mittareiden tulisi tuottaa sellaista dataa, mitä

pystytään käsittelemään ja siitä on hyötyä. Mittarit ovat hyödyttömiä jos niiden tuottamaa dataa ei voida käsitellä (OGC, Continual service improvement, 2007; Bon, 2009).



Kuva 3. ITIL:in ydin (OGC, Continual service improvement, 2007)

3.2 Tietojärjestelmän ylläpitovaiheen menetelmät

Palveluoperaatiot tuottavat havaintoja ja herätteitä, mitkä koskevat tietojärjestelmää tai tietojärjestelmiä. ITIL–menetelmät kuvaavat palveluprosesseja, jossa IT on vahvasti mukana. Tämä aiheuttaa muutoksia sekä tietojärjestelmään ja sitä käytettävään palvelun tuottamiseen. Swanson (1976) esitteli kolme ohjelmiston muutoskategoriaa: i) Parannus: järjestelmän ongelmatilanteiden korjaus datan prosessoinnissa, suorituskyvyssä tai implementoinnissa. ii) Mukautuva: vastataan ennakoituihin muutoksiin datan käsittelyssä. iii) Havainnointi: parannetaan järjestelmän tehokkuutta, suorituskykyä tai ylläpidettävyyttä. Kaksi edellistä kategoriaa ovat yleisesti hyväksyttyjä teollisuudessa selittämään ohjelmiston ylläpidon aktiivisuutta. Edwards (1984) esitti tietojärjestelmän ylläpitovaiheen muutoksiksi sekä käyttäjän vaatimukset että lakien tai muiden tahojen määräämät muutokset. Ylläpitovaiheen keskeinen tehtävä on myös järjestelmävirheiden korjaus ja korjausten toimitus asiakkaalle.

Tietojärjestelmien ylläpitovaihe sisältää kaikki siihen tehtävät muutokset käyttöönoton jälkeen. ITIL – menetelmistä tietojärjestelmän ylläpitovaiheeseen voidaan luokitella palvelutransitiosta muutosten hallinta ja jakelun- ja käyttöönotonhallinta. Palvelutuotannosta tietojärjestelmän ylläpitoon vaikuttaa eniten ongelmienhallinta, herätteidenhallinta, häiriönhallinta ja palvelupyynnöt. Palvelutuotannon prosessit tuottavat herätteitä, jotka eskaloituessaan voivat vaikuttaa tietojärjestelmään. Kaikki ongelmat eivät palveluntuotannossa ole välttämättä itse tietojärjestelmän aiheuttamia.

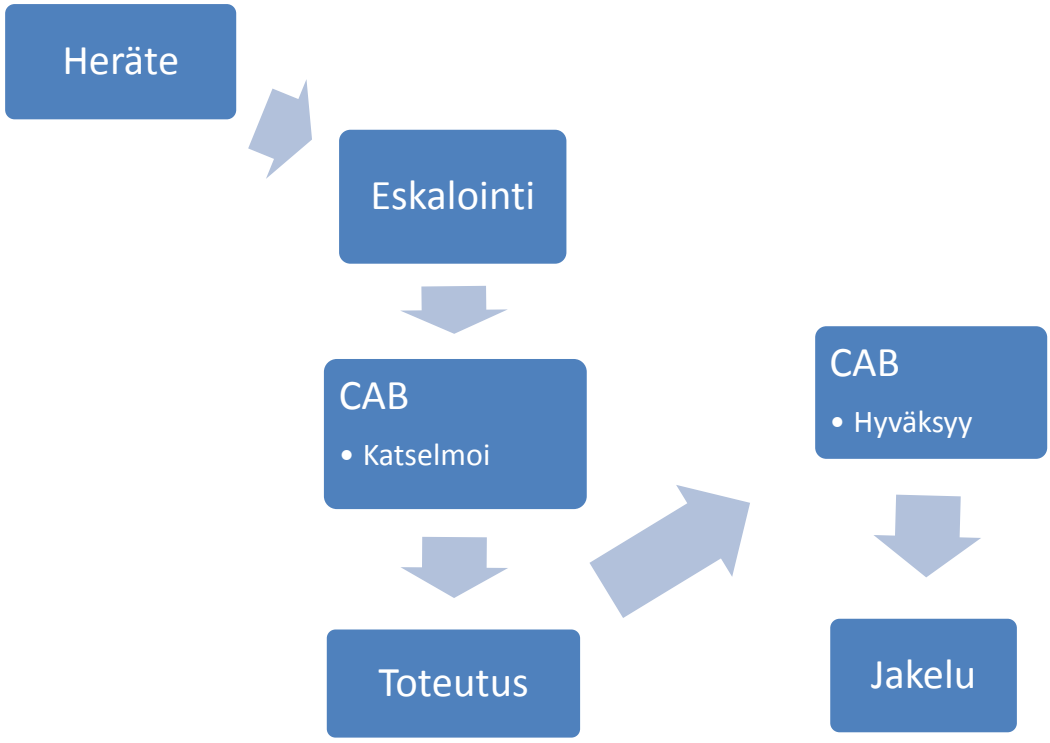
ITIL – menetelmät tarjoavat viitekehyksen palvelunhallinnan menetelmille, kun taas XP on ohjelmistokehityksen viitekehys. Palveluntuotanto tuottaa herätteitä ja muutoksia ja ohjelmistotuotanto toteuttaa ne. Menetelmät kuitenkin integroituvat helposti ja raja niiden kohtaamispisteessä on häilyvä. On kuitenkin oleellista käsittää, mistä ja mitkä aiheuttavat muutoksia tietojärjestelmään. Mitkä ovat ne kanavat, joista muutokset ja herätteet saapuvat.

4. Tutkimusaineiston keruu

Tässä luvussa kuvataan yritys Alfa lyhyesti ja perehdytään tietojärjestelmäyksikön ylläpitotiimillä käytössä oleviin menetelmiin sekä kuvataan tiedonkeruumenetelmät ja kohderyhmä. Alakappaleessa 4.1 kuvataan yritys ja tiimin toiminta sekä käytössä olleet menetelmät. Alakappaleessa 4.2 käydään läpi teemahaastattelu ja sen kulku sekä kyselylomake ja sen rakenne. Alakappaleessa 4.3 kuvataan tapaustutkimus ja 4.4 kyselylomakkeen pilotointi. Alakappaleessa 4.5 kuvataan analysointivaiheen lähestymistapa.

4.1 Tapaus yritys Alfan yleiskuvaus

Yritys Alfa toimii energiasektorilla, jossa yritys tuottaa palveluita ja tietojärjestelmiä energia-alan yrityksille. Yritys Alfan tietojärjestelmän ylläpitotiimillä on hoidettavanaan useita erillisiä tietojärjestelmiä. Tietojärjestelmän ylläpidosta vastaa ylläpitotiimi, joka on kooltaan noin kymmenen henkeä. Ylläpitotiimin tehtäviin kuuluvat tietojärjestelmän päivittäinen tuki-, muutos- ja ongelmanhallinta, käyttöönottoprojektit ja tietojärjestelmän kehitys laki- ja asiakasmuutosten osalta. Omaehtoista tietojärjestelmänkehitystä ei enää tehdä vanhoihin tietojärjestelmiin tai se on hyvin pienimuotoista. Yritys Alfan tietojärjestelmien ylläpitotiimissä on käytössä ITIL–menetelmiin perustuvat prosessit. Prosessit on jaettu muutoksien- ja ongelmienhallintaan. Suuremmat muutokset tehdään erillisissä projekteissa, joissa on käytetty Scrum-menetelmiä ohjelmistokehityksessä. Kuvassa 4 nähdään perusprosessi, missä prosessi lähtee etenemään herätteestä.



Kuva 4. Perusprosessi

Tässä heräte voi olla löydetty ongelma tai muutos, mikä tulee prosessiin asiakkaalta. Eskaloinnilla tarkoitetaan tässä sitä, että eteneekö heräte CABin katselmoitavaksi vai eteneekö se muuhun prosessiin. Tämän työn luonteen kannalta muihin prosesseihin etenevät herätteet ovat rajattu pois.

4.2 Haastattelut ja kyselylomake

Haastattelussa käytettiin sekä kyselylomaketta että teemahaastattelua. Kyselylomakkeen kysymykset perustuvat Poole et al. (2000) ja Choudhari et al. (2010) tutkimuksiin, jossa XP:n menetelmiä implementoidaan tietojärjestelmän ylläpitovaiheen toimintoihin sekä OGC (2007) ITIL–menetelmien ylläpitoprosesseihin. Kysymys numero kymmenen pidettiin avoimena, minkä tarkoituksena oli löytää jotain ennalta arvaamatonta. Hirsjärvi et al. (2000) mainitsee lomakehaastattelun olevan nopea tapa kerätä tietoa ja mahdollistaa hypoteesien nopean testaamisen. Kysymykset pidetään avoimina silloin, kun aihepiiri ei ole vielä jäsentynyt (Järvinen & Järvinen, 2000).

Kyselylomakkeen muodostaminen tapahtui pääosin lukujen 2.4 ja 2.5 kirjoittamisen yhteydessä. Poole et al. (2000) ja Choudhari et al. (2010) tuovat tutkimuksissaan esille ongelmakohtia ja niihin löydettyjä parannuksia. Nämä ongelmat ja parannukset valittiin tässä tutkimuksessa käytetyn kyselylomakkeen teemoiksi. Teemojen valintaan vaikutti myös oma näkemys siitä, missä ongelma-alueita olisi mahdollisesti löydettävissä. Teemojen avulla pyrittiin siten myös vahvistamaan omia hypoteeseja ongelmista.

Teemahaastattelu kohdistuu tiettyyn aihepiiriin ja se voidaan käsittää puolistrukturoituna haastatteluna (Hirsjärvi, 1995). Teemoina tämän tutkimuksen haastatteluissa olivat XP:n ja ITIL:in menetelmät. Vaikka teemahaastattelussa käytettiin kyselylomaketta, samoja kysymyksiä ei kuitenkaan esitetty suoraan uudelleen vaan pyrittiin esittämään jatkokysymyksiä annettuihin vastauksiin. Jokainen kysymys edustaa eri teemaa. Hirsjärvi (1995) kuvaa, että teemahaastattelu ei sisällä yksityiskohtaista kysymysluetteloa vaan teema-alueluettelon. Tässä tutkimuksessa teema-alueet ovat kyselylomakkeen runkona. Kysymysluettelon teemat ovat:

1. Metaforat ja tarinat
2. Refaktoroinnin tarve
3. Läsnäoleva asiakas
4. Testaus
5. Jatkuva integraatio ja pienet julkaisut
6. Pariohjelmointi
7. Kommunikaatio ja palaute

Kyselylomake (liite A) lähetettiin tiimin jäsenille sähköpostilla. Vastaukset kerättiin sähköisinä dokumentteina, johon vastaukset olivat kirjattu. Vastausten keruun ja niiden pintapuolisen tutkimisen jälkeen valittiin viisi tiimin jäsentä tarkempaan haastatteluun. Haastattelussa käytiin samoja kyselylomakkeen kysymyksiä läpi ja esitettiin lisäkysymyksiä kysymyksiin että haastateltavan ennalta antamiin vastauksiin.

4.3 Tapaustutkimus

Tapaustutkimuksessa tietoa voidaan kerätä usealla eri menetelmällä, kuten haastattelut, kyselyt, havainnointi ja arkistomateriaali. Tapaustutkimus voi koskea yhtä tai useampaa tapausta. Yksittäisen tapauksen tutkimusta käytetään, kun on pääsy harvinaiseen tai vaikeasti lähestyttävään tapaukseen. (Järvinen & Järvinen, 2000). Tapaustutkimuksen

tulokset ovat vaikeasti yleistettävissä, mutta niistä voi löytyä myös yhdistäviä tekijöitä (Metsämuuronen, 2003).

Tässä tutkimuksessa keskityttiin yhteen tapaukseen. Hypoteesina oli, että nykyiset käytössä olevat menetelmät ovat jossakin määrin raskaita ja byrokraattisia. Tähän hypoteesiin on päästy oman kokemuksen kautta ja tässä tutkimuksessa oli tarkoitus selvittää, onko tämä yleinen käsite kyseisessä tapauksessa. Tutkimuksen tuloksena saadaan tukea omalle hypoteesille ja niiden pohjalta pystytään antamaan parannusehdotuksia perustuen XP:n eri menetelmiin.

4.4 Pilotointi

Kyselylomakkeen pilotointi suoritettiin yhdelle tiimin jäsenelle. Kyselylomakkeen kysymykset toimivat myös haastattelujen runkoja, joten tämä pilotointi varmisti kysymysten laadun myös haastatteluja varten. Kyselylomakkeen täyttäminen tapahtui täysin itsenäisesti, jolloin haastattelija ei ollut mitenkään sidoksissa haastateltaviin. Järvinen et al. (2000) pitää esihaastatteluja kriittisinä teemahaastattelujen onnistumiselle.

Kyselylomakkeen kysymysten rakenne ja niihin saatavat vastaukset haluttiin ensin testata ja pilotointi toimi siinä hyvin. Tähän vaikutti varmasti osaksi se, että kysely ja haastattelut tehtiin saman tiimin jäsenille, joiden kanssa olen työskennellyt useamman vuoden. Näin ollen kysymysten termit ovat yleisesti käytettyjä ja kaikkien tiimin jäsenten tiedostamia. Pilotoinnista saatuihin vastauksiin ollaan tyytyväisiä ja todettiin, että ne tulevat toimimaan hyvin laajemmassa jakelussa sekä haastattelun runkona. Kyselylomakkeeseen lisättiin vielä kaksi demografista kysymystä, ikä ja sukupuoli sekä hiottiin hieman kysymysten muotoilua. Pilotoinnilla saadut vastaukset liitettiin osaksi empiriaa.

4.5 Analysoinnin lähestymistapa

Hirsjärvi (2001) kuvaa kuusi erilaista tapaa aineiston analyysille. Ensimmäisessä tapauksessa haastateltavat kuvaavat kokemuksiaan ja näkemyksiään spontaanisti ja kuvausta ei juurikaan tulkita haastattelun aikana. Toisessa tavassa haastateltavat havaitsevat itse yhteyksiä asioiden välillä. Kolmannessa tavassa haastattelija tiivistää ja tulkitsee haastateltavan ulosantia haastattelun edetessä. Neljäs tapa tapahtuu pelkästään haastattelijan oman tulkinnan avulla tai yhdessä muiden tutkijoiden kanssa. Viides tapa on uudelleenhaastattelu, jolloin haastattelu pohjautuu ainakin osaksi olemassa olevaan dataan. Kuudes tapa voi olla toiminnallinen, jonka myötä haastateltavat alkavat käyttäytyä tai toimia haastattelusta havaittujen seikkojen perusteella.

Tässä tutkimuksessa analysointitapa oli lähempänä kolmatta ja viidettä lähestymistapaa. Kyselylomakkeen läpikäyntiä voitiin käsittää uudelleen haastatteluksi, vaikkakin kyselylomakkeen data ei välttämättä ollutkaan yhtä laajaa ja rikasta. Kolmas tapa esiintyi siten, että haastateltavalta varmistettiin hänen antamiaan vastauksia ja sitä, vastasivatko ne haastattelijan tekemiä johtopäätöksiä niistä.

5. Analyysi ja tulokset

Tässä luvussa käsitellään tämän tutkimuksen aineistoa ja niiden tuloksia. Alakappaleessa 5.1 kuvataan kerätty aineisto. Alakappaleessa 5.2 kuvataan kyselylomakkeista ja haastatteluista kerätyn datan analysointi. Alakappaleessa 5.3 ehdotetaan parannusehdotukset löydettyihin ongelmiin perustuen XP:n menetelmiin.

5.1 Kyselylomakkeen ja haastattelujen aineisto

Kyselylomake lähetettiin 18 henkilölle. Vastausprosentti oli 66 %. Ensimmäiset neljä kysymystä oli demografisia, joissa kysyttiin vastaajien ikää, sukupuolta, työkokemusta IT–alalta ja nykyistä asemaa. Vastanneista naisia oli 50 % ja keski-ikä 38,5 vuotta. Työkokemusta IT–alalta oli keskimäärin 13 vuotta. Kyselyyn vastanneista 74 % oli asiantuntija-asemassa ja 16 % päällikkötason asemassa. Kysymykset 5 – 11 olivat avoimia kysymyksiä, jotka menevät 4.1 alaluvussa mainitussa teemajärjestyksessä. Kysymys 12 oli avoin kysymys, mikä ei liity ennalta asetettuihin teemoihin. Haastattelut pidettiin viidelle eri henkilölle. Haastateltavat henkilöt valittiin eniten jatkokysymyksiä herättäneiden kyselylomakkeiden perusteella. Tällä parannettiin kysymysten tulkinnan oikeellisuutta ja varmistettiin, että vastaajien vastaukset olivat ymmärretty oikein.

Kysymyksellä 5 pyrittiin selvittämään, millaisia käytäntöjä tai menetelmiä muutosten ja ongelmien kuvauksessa on. Taulukossa 2 nähdään kysymyksen 5 koonti vastauksista. Kysymyksen teemana ovat metaforat ja tarinat, vastauksista ei juuri käy ilmi metaforien olemassaolo. Kuitenkin vastauksista voidaan päätellä se, että virheiden ja ongelmien kuvaus nähdään haasteellisena. Haasteita näyttää aiheuttavan oletus siitä, että muutoksen tai ongelman tekijän oletetaan tietävän asiasta enemmän kuin hän oikeasti tietää. Kuvaukset jäävät usein yleiselle tasolle, mikä aiheuttaa sen, että muutoksen tai ongelman tekijän täytyy perehtyä asiaan syvällisemmin, ennen kuin hän pystyy toteuttamaan kyseisen tapauksen. Tapauksien kirjaamiselle ei ole ennalta sovittua spesifistä kaavaa.

Kuudennella kysymyksellä selvitettiin, nähdäänkö muutoksien tai virheidenkorjausten tekeminen haasteellisena ja mitkä ovat sen pääongelmakohtia. Kysymyksen teema peilaa refaktoroinnin tarpeeseen. Useissa vastauksissa esille nousee järjestelmän laajuus ja kompleksisuus, mikä vaikuttaa siihen, ettei pystytä hahmottamaan muutoksen kokonaisvaikutusta. Ongelmaksi koettiin myös virheidenkorjausten ja muutosten hidas kierto. Koodi nähdään myös vanhentuneena ja kaikkia osia ei ole tehty hyvin.

Seitsemännellä kysymyksellä perehdyttiin asiakkaan läsnäoloon muutoksien toteutuksessa. Kysymyksen teemana on läsnäoleva asiakas. Asiakkaan läsnäolo on usein aika vähäistä. Usein määritys saadaan asiakkaalta ja määritys voi olla hyvin lyhyt, siitä huolimatta kyseisen käyttäjätarinan informaatioin mukaan mennään määrityksestä jakeluun asti. Joissakin tapauksissa asiakas on mukana myös testauksessa. Asiakkaalta saatu tieto nähdään tärkeänä, vastauksissa kuitenkin viitataan siihen, että tiedon saanti asiakkaalta voi olla haasteellista.

Taulukko 2. Poimintoja kysymyksien 5 - 7 vastauksista.

Kysymys	**Vastaukset**
5. Kuvaile, miten muutospyynnöt tai tietojärjestelmää koskevat ongelmat ovat kuvattu ja ovatko ne helposti ymmärrettävissä?	• Riippuu ihmisestä, kuka muutospyynnön tai ongelman on kirjannut. • Ongelmat on monesti hyvin yleisellä tasolla kuvattu. • Muutospyynnössä on kuvattu haluttu lopputulos ja usein asioita pidetään itsestäänselvyytenä. • Mikäli ongelman on kuvannut asiakas, voi joskus olla vaikeaa ymmärtää kuvausta. • Kirjaamiseen ei ole kunnollista sovittua kaavaa. • Muutospyyntöjen ja ongelmien kuvaamisessa ei ole käytössä formaalia kuvaustapaa. • Asiakkaalta saatu ongelma- tai muutospyynnön kuvaus on usein epäselvä ja tulkinnanvarainen, puutteellinen. Vaatii siis asian selvittämistä.
6. Ovatko virheenkorjaukset tai muutoksien implementoinnit helposti tehtävissä? Mikä niissä on haastavinta?	• Haastavinta ovat puutteelliset määrittelyt niin virheenkorjauksissa että muutoksissa. • Aluksi helpolta näyttävä muutos voikin osoittautua hankalaksi ja heijastella muuhun toiminnallisuuteen, jos kaikkia asioita ei ole mietitty etukäteen hyvin. • Haastavaa on siis se, ymmärtääkö, mitä on tarkoitus tehdä, ja lisäksi se, että tietää, mihin kaikkeen muutos vaikuttaa. • Suurimman haasteen eteen joutuu, kun asiaa pitää alkaa testaamaan. • Virheenkorjaukset saattavat lojua virhetietokannassa pitkiäkin aikoja ennen kun CAP niitä ehtii allokoimaan. • Koodia ei välttämättä ole tehty aina viimeisen päälle. Muutoksen voi joutua tekemään useampaan paikkaankin, koska toiminnot voivat olla hajautettu. Ja osa palveluista on vanhentuneita. • Muutoksessa pitää miettiä myös eri tavat käyttää järjestelmää (asiakkaiden eri prosessit, käyttötavat jne..). • Monesti virhetilanteet ovat hankalasti toistettavia, mikä vaikeuttaa korjausten toteutusta ja testausta. • Tuotekehityksenkannalta virheenkorjaus-prosessin tulisi toimia suunnitellusti, jotta korjaus huomioitaisiin tuotannossa olevassa versiossa sekä myös kehitysympäristössä.

	• Mikäli määrittelyä tehdään samalla kuin toteutetaan, on aivan varma, että saadaan aikaan vain uusia ongelmia.
7. Miten kuvailet asiakkaan **osallistumista muutoksien toteutuksessa?**	• Määrittelyt tehdään yhdessä asiakkaan kanssa, ovat yleensä heiltä lähtöisin. Myös lainsäädäntöön liittyvät muutokset katselmoidaan usein asiakkaan toimesta. • Asiakas ei yleensä osallistu muutosten toteutukseen ainakaan pienempien muutosten kohdalla, mutta suurempien muutosten/projektien kohdalla asiakas osallistuu toteutukseen sprinttien ja katselmointien kautta. • He osaavat myös kertoa, mikä toimintatapa palvelisi heitä parhaiten. • Testaajina asiakkaat olisivat hyviä, mutta ilmeisesti asiakkaat eivät kovin innokkaasti testausta tee. • Arvokasta käyttäjätietoakin pitää lypsää ja se auttaa muutoksen vaatimusmäärittelyssä. • Asiakas kertoo, jos muutos ei toimi. • Asiakkaat osallistuvat muutoksiin pääsääntöisesti määrittely – ja testausvaiheessa. • Pienissä muutoksissa ei välttämättä asiakas osallistu kun muutoksen määrittelyyn alussa. • Asiakkaat antavat mielellään lisäkommentteja ja tarkentavat määrittelyä muutoksien toteutuksen aikana. • Tuotteen oikeellisuuden ja eheyden kannalta kaikki asiakasmuutokset selvitetään tuotevastaavan ja palveluvastaavien kesken.

Kysymyksessä 8 pyrittiin selvittämään asiakkaiden muutoksien ja virheidenkorjausten vastaanottoa. Teemana on testaus, joka käsittää myös hyväksymistestauksen. Asiakkailta saatava palaute on lähinnä negatiivistä ja muutoksiin suhtaudutaan hyvin kriittisesti. Hiljaisuus koetaan hyväksymisen merkiksi, erillistä hyväksyntää ei saada. Asiakkaat myös kokevat virheenkorjausten ja muutoksien toimituksen hitaaksi.

Kysymyksellä 9 selvitettiin, miten muutosten ja virheenkorjausten jakelu koetaan. Teemana tässä kysymyksessä on jatkuva integraatio ja pienet julkaisut. Tarkoituksena on ymmärtää, ovatko kyseiset käytännöt jo käytössä, ja jos ne eivät ole, niin nähdäänkö niille tarvetta. Jakelu koetaan ongelmalliseksi, mutta vastaukset ovat myös osaksi ristiriitaisia. Osa pitää toimitusprosessia hitaana ja muutokset sekä korjaukset pitäisi saada asiakkaalle nopeammin. Osa on kuitenkin sitä mieltä, että nykyinen CAB-käytäntö on hyvä asia ja se on parantanut jakelussa tapahtuneita ongelmia. Erillistoimitukset on koettu sekaviksi ja koetaan, että ne aiheuttavat ongelmia jakelussa.

Kysymyksellä 10 selvitettiin miten työkaverin läsnäolo koetaan ja nähdäänkö se riittäväksi tai yleensä tarpeelliseksi. Teemana tässä kysymyksessä on pariohjelmointi.

Muutokset toteutetaan yleensä yksin ja työkaverin apua käytetään lähinnä tarvittaessa. Työkaverin apu koetaan tarpeelliseksi määritystyössä ja vaikeampien tai monimutkaisempien asioiden toteutuksessa. Jo asiasta keskusteleminen voi selkeyttää ajatuksia ja siten auttaa ratkaisun tekemisessä. Vastauksista käy myös ilmi, että kaikkien työkavereiden kanssa kemiat eivät aina kohtaa ja koetaan, ettei työkaveri ole välttämättä aina kiinnostunut neuvomaan tai auttamaan asiassa.

Taulukko 3. Poimintoja kysymyksien 8 – 10 vastauksista.

Kysymys	**Vastaukset**
8. Kuvaile asiakkaiden reaktioita / kommentteja tehtyihin muutoksiin tietojärjestelmässä?	• Yleensä asiakkaat eivät ilmoita mitään jos muutos toimii hyvin. • Mutta joskus harmittavasti testaus jää puutteelliseksi ja tällöin voi muutoksesta aiheutua lisää virheitä. • Virheiden korjausta on toivottu nopeammaksi, sillä monesti joutuu odottamaan korjattua osuutta seuraavaan versioon. • Räätälöity muutos, tulee se yleensä tyrmättyä toisen asiakkaan taholta. • Muutosvastarintaa esiintyy, mikäli toiminnallisuus ei ole suora virheenkorjaus. • Asiakkaat ovat tyytyväisiä kun saavat toimivan muutoksen. Jos muutoksessa on virhe, silloin ei niin tyytyväisiä ja yleensä halutaan korjaus mahdollisimman pian. • Parhaiten asiat sujuvat silloin, kun asiakasta on informoitu työn edistymisestä toteutustyön aikana. • Tässä yhteydessä voi nousta esiin myös uusia muutostarpeita, jatkokehitystä jo tehtyyn. • Tieto ei kulje riittävästi tai sitä on niin paljon, että yksittäinen muutos hukkuu massaan.
9. Miten muutosten ja virheidenkorjausten jakelu on mielestäsi toiminut tähän asti?	• Versioita toimitetaan noin 2 kertaa vuodessa. HOTFIX:t ovat tietenkin erikseen ja ne toimitetaan heti asiakkaalle kun vika havaitaan. • Voitaisiin ehkä toimittaa ripeämmällä aikataululla. • Ollut ainakin ilmeisen sekava, koska muutoksia tms. toimitellaan asiakkaille muutenkin kuin versiotoimituksissa. • Kukaan ei saa enää viedä korjauksia ilman CAP ryhmän hyväksyntää, mikä on mielestäni todella hyvä juttu. • Prosessia on selkeytetty ja mm. toimitusten kirjaamista on parannettu. • Jos yksittäiset muutokset toteutetaan hallitusti, eli koodi löytyy tarvittavista paikoista palvelimiltakin,

	korjauksia ei todennäköisesti menetetä seuraavan päivityksen yhteydessä. • Toimittaessa sovittujen prosessien mukaan on jakelu myös toiminut hyvin. • Itse versiokoonti on jo massiivinen operaatio nykymenetelmin.
10. Kuvaile työkaverin läsnäoloa muutosten ja virheenkorjausten toteutuksen yhteydessä?	• Yleensä muutokset toteutetaan yksin. • Yleensä on hyvä, että vaikeampien muutosten/korjausten kohdalla asiaa katsoo useampi silmäpari. • Riippuu työkaverista, joskus vastaus on se, ettei tiedä asiasta mitään eikä oikeastaan viitsi asiaan paneutuakaan. • Porukka on hyvin avuliasta. • Jo asian esitteleminen helpottaa ajattelemaan asiaa eri puolilta. Lisäksi toinen todennäköisesti pystyy antamaan hyödyllisiä neuvoja asiasta. • Toteutustyön laatu paranee ja hiljainen tieto kasvaa kun prosessiin osallistuu useampi henkilö. • Työyhteisömme on pieni ja tiivis, joten toivoakseni kysymyksiin saa helposti vastauksia ja työtä ohjataan riittävästi.

Kysymyksellä 11 pyrittiin selvittämään nähdäänkö kommunikaatio ja palaute riittävänä. Teemana tässä kysymyksessä ovat juurikin kommunikaatio ja palaute. Kommunikaatioita katsotaan olevan tiimin jäsenten kesken mutta enemmänkin sitä toivottaisiin. Palaute on yleensä negatiivista, jos sellaista joskus saadaan. Kommunikaatioita prosessien välillä on pyritty parantamaan CAB-käytäntöjen avulla mutta siinäkin nähdään vielä puutteita.

Taulukko 4. Poimintoja kysymyksien 11 – 12 vastauksista.

Kysymys	**Vastaukset**
11. Onko muutosten- ja ongelmanhallinnan prosessien osa-alueiden välinen kommunikaatio riittävää? Minkälaista palautetta saat?	• On yleensä riittävää, en yleensä saa palautetta. • Liian usein toimitetaan muutos/korjaus josta ei ole tarpeeksi tietoa toisella osapuolella. • Palautetta ei juuri saa, muuten kuin jos jotakin on jäänyt tekemättä tai pitää täydentää jotakin tietoja. • Palautetta ei kauheana anneta työkavereiden kesken, mutta asiakkaalta toki saadaan palautetta silloin, kun jokin ei mene käsikirjoituksen mukaan. • Enempi voisi tiedottaa, kun jotakin isoja korjauksia/muutoksia on menossa johonkin.

	• Palautetta saa vaatimusmäärittelyn ajalta kun CAP käsittelee muutoksen, mutta monesti testaaminen ja muutoksen vienti jää täysin omalle kontille. • Kommunikaatiota on pyritty parantamaan mm. CAB käsittelyllä. • Yksittäinen muutoksen tai ongelman toteuttaja ei välttämättä hahmota kokonaisuutta mihin muutos vaikuttaa. • Asiakkaat ilmoittavat ongelma eivätkä sen jälkeen saa juuri tietoa, milloin tai miten ongelma ratkaistaan.
12. Mikä on mielestäsi ongelmallisinta tietojärjestelmän ylläpidossa?	• Tiedotusten puute / kommunikaation puute kaikkien työntekijöiden välillä, jotka ovat tietojärjestelmän kanssa tekemisessä. Tietojärjestelmän laajuuden ymmärtäminen. • Järjestelmä on suuri ja liikkuvia osia paljon. • Ehkä se, että ymmärtää kokonaisuuden ja sen, mihin kaikkeen jokin muutos vaikuttaa. • Että kellään ei tunnu olevan kokonaiskuvaa tietojärjestelmästä ja sen osista. • Jaettu vastuu ei ole vastuuta laisinkaan ja mikäli ei ole henkilöä kenen vastuulla tietyt asiat ovat, ne jäävät monesti hoitamatta muiden kiireiden takia. • Koodia on äärettömän paljon. Käytännössä ei ole mahdollista tietää kaikkea kaikesta. • Sovittujen toimintaprosessien noudattaminen luo raamin jolla ylläpitotoimintaa voidaan ohjatusti hallita. • Toiminnan avoimuus asiakkaiden suuntaan.

Kysymys 12 oli avoin kysymys siitä, mikä on vastaajan mielestä ongelmallisinta tietojärjestelmän ylläpidossa. Tämä kysymys ei perustu ennalta valittuun teemaan. Vastaajien mielestä suurin ongelma on kokonaisuuden hahmottaminen. Liian vähäinen kommunikaatio tai tiedottaminen nähdään ongelmaksi. Vastuunkannon ja kokonaiskuvan puuttuminen nousi myös esille vastauksista.

5.2 Datan analysointi

Jokaisella kysymyslomakkeen kysymyksellä on teema, lukuun ottamatta demografisia kysymyksiä alussa sekä kysymystä 12. Lomakkeen kysymykset ovat valittu XP:n menetelmistä ja käytännöistä. Kaikista menetelmistä ei kysytty suoraa kysymystä, koska sitä ei katsottu tarpeelliseksi. Esimerkiksi 40 tunnin työviikko on yritys Alfassa standardi, jota harvemmin ylitetään pakollisten ylitöiden merkeissä. Aiheen luonteen takia kaikkea ei ole myös tarpeellista kysyä, koska XP:n menetelmät limittyvät keskenään, kuten myös vastauksista voidaan päätellä. Haastatteluilla tarkennettiin

kyselylomakkeella saatuja tuloksia. Lomakkeen kysymyksiä pidettiin haastattelun runkona ja haastattelu eteni kysymysten mukaan. Kysymyksiin aiemmin saatuihin vastauksiin kysyttiin tarkentavia kysymyksiä ja teema pyrittiin pitämään samana.

5.2.1 Tarinat ja metaforat

Metaforat eivät nousseet minkään kysymyksen johdosta esille. Juric (2000) esittääkin metaforan problematiikan, hän esittää metaforan käytännön enemmän koherenttina tarinana, joka antaa jokaiselle mielikuvan tehtävän päämäärästä. Kysymyksen yksi myötä esille nousi ajatuksia siitä, että ”*tarinat ja käyttäjäkertomukset vaihtelevat hyvin paljon riippuen sen kirjaajasta”*. Aineistossa on useampia kommentteja, kuten nämä: *”Riippuu ihmisestä, kuka muutospyynnön tai ongelman on kirjannut”, ”Mikäli ongelman on kuvannut asiakas, voi joskus olla vaikeaa ymmärtää kuvausta.”* Kirjaajan kokemus, toimiala – ja järjestelmätuntemus vaikuttavat paljon siihen, minkälainen tarina muutoksesta tai ongelmasta kirjataan. Käyttäjätarinoiden analysointi jää usein yhden ihmisen harteille ja koska ei ole sovittua kaavaa siitä, miten tarinat kirjataan, jää oleellisia asioita puuttumaan. Tarinoiden analysoinnin on katsottu parantuneen CAB-käytännön myötä, mutta esille tulee sen toimivuus. CAB hyväksyy muutoksen tai virheenkorjauksen ja siirtää sen eteenpäin, joko tarkempaan määritykseen tai toteutukseen. Tästä huolimatta, tiimin jäsenet katsovat, että usein tehtävän tarina on puutteellinen ja ei sisällä tarvittavaa tietoa tehtävän suorittamiseksi.

5.2.2 Refaktoroinnin tarve ja kollektiivinen omistajuus

Tarinoiden puutteet heijastuvat suoraan muutosten ja ongelmien implementointiin. Jos tarinoiden puutteellinen tai virheellinen määritys pääsee toteutusvaiheeseen, tulee se aiheuttamaan virheellisiä toimintoja järjestelmään. Tiimin jäsenet näkevät muutoksien tekemisen järjestelmään haastavana sen laajuuden ja kompleksisuuden takia. Järjestelmä on laajentunut aikojen saatossa ja järjestelmää on ollut tekemässä useita henkilöitä. Kaikilla tiimin jäsenillä on oikeus muuttaa koodia. Kollektiivinen omistajuuden riskinä on muutoksien tai virheidenkorjauksien tekeminen paikkoihin, joissa ei ymmärretä sen kaikkia vaikutuksia (Choudhari & Suman, 2010). Edellä mainittu riski näyttää korreloituvan ja aiheuttavan ongelmia tietojärjestelmä ylläpidossa tässä tapauksessa. Kollektiivinen omistajuus on osaksi rajoittunut CAB-käytännön myötä ja selkeiden tapausten virheidenkorjaus jää odottamaan prosessin läpivientiä.

Esille tulee myös erilaiset koodaustavat ja useiden samanlaisten toimintojen olemassaolo. Muutoksia joudutaan usein tekemään useampaan paikkaan ja se lisää haastavuutta siinä, että kaikki tällaiset kohdat varmasti löydetään toteutuksen yhteydessä. Datan keräyksessä esille edelliseen kohtaan tuli seuraavan kaltaisia kommentteja: *”Koodia ei välttämättä ole tehty aina viimeisen päälle. Muutoksen voi joutua tekemään useampaan paikkaankin, koska toiminnot voivat olla hajautettu. Ja osa palveluista on vanhentuneita.”* Kompleksisuutta lisää useat eri toimintamallit, mitä järjestelmä sallii. Järjestelmää on myös hyvin pitkälle parametrisoitu, jolloin eri toiminnallisuuden osia voidaan kytkeä päälle tai muuttaa toiminnon käyttäytymistä. Koodia ei juurikaan refaktoroida, vaan muutokset ja virheidenkorjaukset pyritään tekemään koskematta koodin rakenteeseen. Tämä lisää koodin kompleksisuutta.

5.2.3 Asiakkaan osallistuminen

Asiakkaan läsnäolo on vähäistä. Asiakkaiden mukanaoloa pidetään tärkeänä ja heiltä saatua palautetta arvokkaana. Suurin osa muutoksista ja ongelmista tulevat asiakasrajapinnasta, milloin asiakkaan ja tiimin jäsenen kommunikaatio voi aiheuttaa

sekaannuksia. Sekaannusta voi aiheuttaa yhteisen kielen puute tai osapuolten toimialatuntemuksen puute. Asiakkaat eivät ole fyysisesti tekemisissä suurimman osan tiimin jäsenten kanssa, vaan kommunikaatio tapahtuu puhelimen tai sähköpostin välityksellä. Osa tiimin jäsenistä kokee, että kontakti asiakkaaseen on riittävää ja pienemmistä muutoksista tai virheenkorjauksista ei välttämättä tarvita palautetta tai kommentteja. Katsotaan, että asiakkaan läsnäolon lisääminen voi kasvattaa myös muutoksien laajuutta, koska asiakas voi keksiä uusia ominaisuuksia. Asiakkaiden katsottaisiin olevan hyviä esimerkiksi testauksessa: *”Testaajina asiakkaat olisivat hyviä, mutta ilmeisesti asiakkaat eivät kovin innokkaasti testausta tee.”,* milloin korjaukset ja muutokset tulisivat samalla hyväksytettyä asiakkaalla. Koetaan, että asiakkaat antavat mielellään lisätietoja muutoksista ja ongelmista. Kuitenkin asiakkailta harvemmin pyydetään lisäkommentteja muutokseen tai ongelmaan. Tällä hetkellä käytettävä menetelmä on Jeffries et al. (2001) mainitsema läsnäolevan asiakkaan korvaustapa, jossa tuotteen omistaja tai vastaava henkilö toimii osakseen asiakkaan roolissa. Korvaavan menetelmän haasteen luo järjestelmän laajuus ja pelkästään tuotteen omistaja ei pysty välttämättä hallitsemaan koko järjestelmää asiakkaalle edullisella tavalla. Projektoiduissa muutoksissa käytetään Scrum:ia. Scrum-käytännöt, kuten asiakkaan kanssa pidettävät demot katsotaan hyödyllisiksi. Näissä tapahtumissa päästään suoraan asiakkaan kanssa tekemisiin ja saadaan suoraa palautetta tehdystä työstä ja ennalta määritellyistä tehtävistä.

5.2.4 Testauksen tila

Asiakkaat eivät juuri kommentoi muutoksia tai virheenkorjauksia toimituksen jälkeen, mikäli ne toimivat halutulla tavalla. Hyväksymistestaus tapahtuu enemmänkin tuotannossa ja samanlaista palautetta on myös tullut tiimin jäseniltä varsinaisista yksikkötestauksista. Testausta pidetään lähtökohtaisesti vaikeana, johtuen kunnollisten yksikkötestausten puutteesta. Tämä aiheuttaa ongelmia jo varsinaisessa implementointivaiheessa, koska muutokset pyritään tekemään koskematta muuhun logiikkaan. Refaktorointi jää tällä tavoin hyvin vähälle ja koodi pääsee kompleksoitumaan. Jeffries et al. (2001) kuvaa kyseisen ongelmatilanteen kirjassaan. Testauksen puutteet nousivat usean tiimin jäsenen haastatteluissa esille*: ”Mutta joskus harmittavasti testaus jää puutteelliseksi ja tällöin voi muutoksesta aiheutua lisää virheitä.”* Testausta pidetään tärkeänä, mutta tällä hetkellä erittäin aikaa vieväksi prosessiksi. Testaus on käytännössä manuaalista työtä ja automaattitestausta ei voida suorittaa tällä hetkellä.

Hyväksymistestaus jää käytännössä myös hyvin vähälle. Suurempien projektien Scrum-käytännöt herättävät positiivisia mielikuvia tiimin jäsenissä ja niitä pidetään hyvinä myös muutoksen hyväksymistestauksen kannalta. Asiakkaan informointi katsotaan tärkeäksi ja asiakastyytyväisyyttä edistäväksi asiaksi: *”Parhaiten asiat sujuvat silloin, kun asiakasta on informoitu työn edistymisestä toteutustyön aikana.”* Informointi katsotaan paremmin toimivaksi projektien osalta ja pienemmät muutokset ja virheenkorjaukset jäävät vähemmälle huomiolle. Päivittäisessä ylläpidossa informaation katsotaan kulkevan asiakkaalle paremmin jo tällä hetkellä. Täytyy kuitenkin ottaa huomioon se, että pelkkä informaation välitys asiakkaalle ei ole hyväksymistestausta. Informaatioin ajankohtaisuus on myös tärkeää sen merkityksen kannalta. Jos palaute saadaan liian myöhään, voi muutokset olla jo siirretty asiakkaan järjestelmään.

5.2.5 Jakelun käytännöt

Muutosten ja virheidenkorjausten jakelussa on ollut ongelmia mutta parempaan suuntaan on menty CAB-käytännön myötä. Ongelmana on ollut yksittäisten korjausten

tai muutosten toimitusten aiheuttamat ongelmat CAB - käytännön katsotaan parantaneet toimituksia, koska näin se tapahtuu aina kootusti: *”Kukaan ei saa enää viedä korjauksia ilman CAP- ryhmän hyväksyntää, mikä on mielestäni todella hyvä juttu.”, ”Prosessia on selkeytetty ja mm. toimitusten kirjaamista on parannettu.”* Negatiivisena puolena tämän käytännön myötä on tullut korjausten ja muutosten toimitusten hitaus. Muutoksia ei viedä ilman lupaa ja mikäli CAB-käytännössä tapahtuu poikkeuksia, voi muutoksen tai virheenkorjauksen toimitus kestää hyvinkin kauan. Tiimin jäsenten mielestä käytäntö on sen selkeyden vuoksi hyvä ja vastuu saadaan siirrettyä CAB-ryhmälle. Osa tiimin jäsenistä on sitä mieltä, että kyseinen käytäntö hidastaa muutosten ja virheidenkorjausten toimitusta huomattavasti. Kriittiset virheenkorjaukset toimitetaan erillisellä HOTFIX – käytännöllä, jossa tapaus käsitellään välittömästi sen kriittisyyden perusteella. Julkaisuja on kuitenkin hyvin vähän per vuosi, noin kaksi kappaletta. Mikäli virheenkorjaukset jäävät odottamaan seuraavaa versiota, tulee toimituksen väliksi jopa puoli vuotta. Puoli vuotta on hyvin pitkä aika asiakkaalle. Jatkuva integraatio ei ole kovin kriittisessä osassa, koska käytetty tekniikka ei tarvitse erillistä kääntämistä (Build). Kääntäminen tapahtuu käytännössä suoraan ohjelmoinnin yhteydessä ja valmis koodi siirretään testipalvelimelle.

5.2.6 Työskentelytavat ja -tottumukset

Työskentely tapahtuu hyvin itsenäisesti tehtävienjaon jälkeen. Tehtävän jaon suorittaa CAB, jossa tehtävät priorisoidaan ja tehtävät jaetaan ohjelmoijille. Tiimin jäsenet katsovat työkavereiden olevan hyödyksi ja muutosten tai virheenkorjausten yhteydessä työkaverin apua käytetään lähinnä määrittelyvaiheessa: *”Yleensä on hyvä, että vaikeampien muutosten/korjausten kohdalla asiaa katsoo useampi silmäpari.”, ”Jo asian esitteleminen helpottaa ajattelemaan asiaa eri puolilta. Lisäksi toinen todennäköisesti pystyy antamaan hyödyllisiä neuvoja asiasta.”*. Tiimin jäsenillä on ristiriitaisia kommentteja siitä, katsotaanko työkaverin läsnäolo aina tarpeelliseksi: *”Riippuu työkaverista, joskus vastaus on se, ettei tiedä asiasta mitään eikä oikeastaan viitsi asiaan paneutuakaan.”* Suurin osa pitää itsenäisestä työskentelystä ja toivovat työrauhaa tehtävien ajaksi. Osa tiimin jäsenistä katsoo työkaverin läsnäolon hyvin tarpeelliseksi ja mieltää pariohjelmoinnin yhdeksi vaihtoehdoksi työskentelytapana. Muutokset ja virheenkorjaukset katsotaan usein sen verran pieniksi tehtäviksi, ettei paria tarvita sen toteutukseen tai määritykseen.

Suurin osa tiimin jäsenistä työskentelee huoneissa, missä on läsnä työkavereita. Beck (1999) mainitsee, että työkaverin läsnäolo tilojen puitteissa tulisi olla mahdollista. Tämä toteutuu osaksi ja työtilat tarjoavat mahdollisuuden kysyä ongelmista työkavereilta ja näin mahdollistaa suoremman kommunikaation. Työskentelytavaksi on muodostunut tapa kutsua työkaveri paikalle tarvittaessa. Tiimin jäsenet kokevat, että apua saa tarvittaessa ja työkaverit auttavat yleensä mielellään: *”Porukka on hyvin avuliasta.”*. Poikkeuksina löytyy myös tapaus, jossa katsotaan, että työkaveri ei ole kiinnostunut ongelmasta, vaikka apua kysytään. Tällaisissa tapauksissa ilmenee samanlaista ongelmaa, kuin parinmuodostuksen ongelma pariohjelmoinnissa. Glass (2001) mukaan pariohjelmointi vaatii tietynlaista ihmiskemiaa parin välillä, joka mahdollistaa kyseisen toimintatavan menestyksekkäästi.

5.2.7 Kommunikaatio ja palaute

Työskentely työkavereiden kanssa katsotaan parantavan kommunikaatiota ja sitä pidetään pääsääntöisesti riittävänä työkavereiden välillä. Kommunikaation vähäisyyttä katsotaan olevan enemmän asiakkaan ja tiimin välillä. Tiimin jäsenet kokevat, että tiedotuksia ja ilmoituksia saisi olla enemmänkin, ja osa tärkeistä asioista jää hyväksi

katsotusta kommunikaatioista huolimatta saapumatta sitä tarvitseville osapuolille: *”Enempi voisi tiedottaa, kun jotakin isoja korjauksia/muutoksia on menossa johonkin*.”. Osa tiimin jäsenistä näkee, että CAB-käytäntö pyrkii parantamaan kommunikaatiota. Ilmi tulee kuitenkin myös se, että kommunikaatioketju ei ole aina täydellinen CAB-käytännössäkään, vaan kommunikaatio tapahtuu tehtävän alussa, jonka jälkeen kommunikaatio jää vähemmälle tai katkeaa kokonaan. Tiimin jäsenet kokevat, että palautetta ei tehtävistä juuri tule ja mikäli sitä tulee, on se luonteeltaan negatiivista: *”Palautetta ei juuri saa, muuten kuin jos jotakin on jäänyt tekemättä tai pitää täydentää jotakin tietoja.”, ”Palautetta ei kauheana anneta työkavereiden kesken, mutta asiakkaalta toki saadaan palautetta silloin, kun jokin ei mene käsikirjoituksen mukaan.”*. Palauteketjun katsotaan toimivan lähinnä silloin, kun asiakkaalle toimitetaan virheellisiä toiminnallisuuksia tai virheenkorjaukset eivät toimi halutulla tavalla. Tiimin jäsenet eivät saa palautetta siitä, kuinka hyvin tai huonosti jokin muutos on toteutettu. Tähän yhdistyy myös testauksen manuaalisuus, joka vähentää palautteen saamista, koska testit eivät ole niin kattavia. Palautteen saamista hidastavat myös pitkät jakeluvälit. Hyväksymistestaus jää käytännössä aina tuotantovaiheeseen, jolloin palaute tulee vasta sitten, kun asia huomataan tuotannossa. Palautetta saadaan jonkin verran työkavereilta mutta sitä pidetään liian vähäisenä. Tiimin jäsenet kokevat, että positiivinen palaute auttaisi heitä parantamaan työskentelyä ja motivoisi heitä työtehtävissä. Kuitenkin suurin osa palautteesta katsotaan olevan negatiivista.

Yleisellä tasolla tiimin jäsenet näkevät tietojärjestelmien ylläpidon haasteellisena sen laajuuden takia. Hallittavat tietojärjestelmät kattavat laajoja osa-alueita, jotka sisältävät paljon asiakasmäärittelyihin ja lakeihin pohjautuvia toiminnallisuuksia ja sääntöjä. Suurentuvat datamassat aiheuttavat teknisiä haasteita ja luovat uusia ratkaistavia ongelmia. Osa tiimin jäsenistä tuntee, että vastuualueita ei ole jaettu tarpeeksi ja se tekee kokonaisuuksien hallinnasta vaikeaa. Katsotaan, ettei kenelläkään ole kokonaiskuvaa ylläpidettävistä tietojärjestelmistä. Kokonaiskuvan puuttuminen vaikeuttaa vastuunkantoa, koska ei ole selkeää kuvaan siitä, kuinka asian pitäisi tehdä tai toimia.

5.3 Parannusehdotukset

Datan analysoinnilla havaittiin ongelmakohtia, joihin XP:n menetelmillä voidaan tarjota parannusehdotuksia. Poole, Murphy, Huisman & Higgins, (2001) tutkimus XP:n käyttöönottamisesta tietojärjestelmän ylläpidossa tuotti tuloksia heidän tapaustutkimuksessaan. Choudhari ja Suman (2010) esittävät XP:seen perustuvan prosessimallin, joka parantaa XP:n menetelmien avulla epärakenteellista koodia, tiimimoraalia, projektin läpinäkyvyyttä, kommunikaation puutetta ja ylläpitoprosessia. Tämän tutkimuksen myötä ongelmia löydettiin jokaisesta kysytystä teema-alueesta. On kuitenkin hyvin epätodennäköistä, että päästäisiin tilanteeseen, mistä ei tarvitse tai voi enää parantaa. Osa ongelmista on suurempia kuin toiset ja osa voi kuulostaa isommalta ongelmalta kuin onkaan. Tilanteessa, kuten tietojärjestelmän ylläpito, on monta liikkuvaa osaa ja kaikki osat vaikuttavat jollakin tapaa toisiinsa. XP tarjoaakin menetelmiä ja käytäntöjä, jotka pyrkivät yhdessä parantamaan ohjelmistonkehitystä. Kyselyjen ja haastattelujen pohjalta saatiin positiivisia kokemuksia projekteissa käytetystä Scrum – menetelmistä. Tämä kuvastaa, että Agile – menetelmien hyödyt nousevat nopeasti esille ja sen, että niiden soveltaminen kannattaa. XP menetelmänä on tarkoitettu pienemmille tiimeille, kuten tässä tapaustutkimuksessa oleva tiimi. Seuraavaksi käsitellään mitkä XP:n menetelmät ja arvot voisivat hyödyntää yritys Alfan tietojärjestelmän ylläpitotiimiä. Taulukossa 5 nähdään löydettyjä ongelmakohtia ja niihin ehdotettuja parannustoimenpiteitä.

Taulukko 5. Löydetyt ongelmakohdat ja parannusehdotukset.

Ongelma	**Parannusehdotus**
1. Asiakkailta saadut ongelmakuvaukset ja muutospyynnöt nähtiin usein puutteellisina.	• Käyttäjäkertomuksien täytyy sisältää tarpeellinen tieto muutoksien ja virheenkorjausten toteutuksen mahdollistamiseksi. • Suunnittelupeli mahdollistaisi laajemman toimiala – ja järjestelmätuntemuksen tehtävien jako ja määrittelyvaiheessa. • Pyrkimys yksinkertaisuuteen kaikessa tekemisessä
2. Muutosten – ja ongelmien toteutus nähtiin haasteellisena. Koodimuutoksia joudutaan usein tekemään useaan eri paikkaan. Koodi sisältää vanhentuneita palveluita, joita ei enää käytetä.	• Refaktoroinnilla voidaan vähentää lähdekoodin kompleksisuutta, joka helpottaa uusien muutosten toteutusta. • Suunnittelupeli pystyy tarjoamaan myös tähän ongelmaan parannusta. Pelin avulla saadaan useamman asiantuntijan näkemys tarinasta, jolloin saadaan luotua yhteinen näkemys asiasta. • Tarvittaessa voidaan käyttää pariohjelmointia. Hyvä käytäntö esimerkiksi kisälli – mestari periaate, jossa mestari opettaa kisälliä.
3. Asiakkaan läsnäolo katsotaan riittäväksi mutta asiakastestauksen lisääminen voisi parantaa laatua.	• Hyväksymistestauksella varmistetaan muutosten oikeellisuus. Siirtää osan testausvastuuta myös asiakkaalle. • Asiakkaan läsnäoloa olisi hyvä lisätä esimerkiksi yhteisten muutosten ja virheidenkorjausten katselmointien parissa.
4. Testaus koetaan raskaaksi prosessiksi, koska automaatiotestausta ei ole ollenkaan. Hyväksymistestausta ei tehdä kuin erillisissä projekteissa.	• Valmiit yksikkötestit, testaus painotteinen kehitys ja automaatiotestit parantavat ohjelmakoodin laatua ja luotettavuutta. • Hyväksymistestauksesta saatava palaute helpottaa myös yksittäisen testaajan kuormaa.
5. Virheenkorjauksia ei toimiteta riittävällä nopeudella asiakkaalle.	• Lyhyet toimitussyklit nopeuttavat virheenkorjausten ja muutosten toimitusta. • Virheenkorjaukset voitaisiin toimittaa erillisenä prosessina muutoksien toimituksesta.
6. Palautetta ei juuri saada ja kommunikaatiota voisi parantaa	• Nopean palautteen merkitystä tulee nostaa esille. Suoran palautteen antaa hyväksymistestaus ja yksikkötestit. • Rakentavan palautteen antaminen työkavereiden kesken. • Kommunikaatiota tulee yksinkertaistaa sisäisesti ja parantaa asiakkaan kanssa käytävää kommunikaatioita. Näitä parantavat XP:n käytäntöjen käyttöönotot, esimerkiksi hyväksymistestaus.

5.3.1 XP:n menetelmien soveltaminen

Metaforien ja tarinoiden tärkeys korostuu asiakkailta saaduissa määrittelyissä ja käyttäjäkertomuksissa. Kyselylomakkeella ja haastatteluilla kerätty data kertoo sen, että tarinoiden ymmärtäminen voi olla vaikeaa, riippuen joko asiakkaasta tai tehtävän kirjaajasta. XP:n menetelmät tarjoavat tähän avuksi useampaa menetelmää, kuten metaforia kuvaamaan objekteja paremmin ja suunnittelupeliä tarinoiden tueksi ohjelmoijille. Metaforien ongelmallisuutta ja moniulotteisuutta kritisoitiin jo aiemmin ja sen ottamista käyttöön vanhaan järjestelmään, missä termistöt ja käsitteet ovat vakiintuneet, ei katsota kovin käytännölliseksi. Kuitenkin tarinoiden merkitys on suuri ja kerättyyn dataan viitaten voidaan todeta, että tarkemmat spesifikaatiot tarinoista ovat hyödyllisiä. Tarinoilla tulisi olla runko, jonka varaan ne rakentuvat. Näin ollen tarinat sisältävät aina minimitiedot. XP:n tarjoama suunnittelupeli toimisi hyvin määritysten varmistajana ja kehittäjien kokoontuminen antaisi hyviä toteutusvinkkejä muille osallistujille. Tällä hetkellä CAB päättää ja jakaa tehtävät, jolloin se jää usein CAB:in tai yksin ohjelmoijan vastuulle tulkita käyttäjätarinaa. Suunnittelupeli kokoaisi ohjelmoijat, joilla on kooditason tuntemusta. Tämä auttaisi myös ilmenneeseen laajuuden ja kompleksisuuden ongelmaan. CAB:in käyttäminen nykyisessä mallissa on myös osaksi ITIL:in määritysten vastaista, koska ITIL-menetelmät ovat palvelunhallintaa, ei virheenkorjausta tai kooditason hallintaa. Suunnittelupelin tarkoitus yhdistää liiketoiminta ja tekniset ratkaisut (Beck 1999; Jeffries et al. 2001).

Tietojärjestelmien laajuus ja kompleksisuus nousi vahvasti esille tässä tutkimuksessa. Kaikkein haasteellisena tietojärjestelmän ylläpidossa pidetään sen laajuuden ja kompleksisuuden hallitsemista ja sitä, että pystyy ymmärtämään kaikki, mihin jokin muutos vaikuttaa. XP:n käytännöistä eniten kompleksisuutta vähentää refaktorointi, mikä olisi hyödyllistä ottaa osaksi ohjelmointikäytäntöä. Käyttämättömän ja huonosti suunniteltujen koodien refaktorointi helpottaa vanhan koodin ylläpitämistä (Poole et al. 2001). Choudhari ja Suman (2010) kuvaavat tutkimuksessaan, että refaktorointi vähensi jopa 40 % koodin määrää ja näin ollen teki siitä vähemmän kompleksista. Kuten Kivi et al. (2000) mainitsi, ei refaktoroinnista saa tulla ohjelmoinnin päätarkoitus. Laajassa järjestelmässä koodia on paljon ja refaktorointiprosessi kestäisi kauan, mikäli haluttaisiin käydä koko järjestelmä läpi. Siksi järkevämpää on käydä läpi alueita, joihin kohdistuu paljon muutoksia, ja joissa ilmenee paljon virheitä. Fowler (1999) on tutkinut Kent Beckin kanssa, kuinka koodista voidaan löytää refaktoroitavia alueita. Näiden menetelmien pohjalta voidaan valita tarpeelliseksi katsottavat menetelmät koodin refaktorointiin. Tärkeää on kuitenkin puuttua rakenteellisesti epävakaaseen koodiin aina kun sitä tulee vastaan ohjelmoijalle ja tämä tulisi olla muutosten ja virheenkorjausten tekijöiden mielessä.

Testauksen haasteet ja ongelmallisuus nousi esille useassa eri kysymyksessä ja haastattelussa. Kuten Jeffries et al. (2001) mainitsee, XP:n menetelmien mukaan pyritään testaamaan kaikki, mikä voi vaan mennä rikki. Tätä sanontaa ei juurikaan noudateta yritys Alfassa. Testaus jää usein vajaaksi, johtuen usein ajanpuutteesta ja testauksen hankaluudesta. Käytännössä testaus tapahtuu ohjelmoijan omilla yksikkötesteillä ja version systeemitesteillä. Tämä jättää yksikkötestit täysin ohjelmoijan varaan ja niiden tuloksia ei yleensä ole saatavissa tai välttämättä edes toistettavissa. XP:n menetelmät tarjoavat käytännöksi yksikkötestausta ja hyväksymistestausta ns. testaus painotteista kehitystä (test driven development). Tavoitteena on automatisoida kaikki testit, mitkä vain voidaan ja ottaa työpari jokaisen virheenkorjauksen tekemisen ajaksi. Näillä käytännöillä pyritään lisäämään koodin laatua ja virheettömyyttä (Poole et al. 2001).

Hyväksymistestauksella lopullinen versio hyväksytetään asiakkaalla (Liu, 2012). Hyväksymistestauksessa asiakas pääsee testaamaan tehdyt muutokset ja sanomaan viimeiset sanansa muutoksesta tai virheenkorjauksesta (Jeffries et al. 2001). Yritys Alfassa hyväksymistestaus on parantunut jonkin verran versiokuvausten parannusten jälkeen, mikä on edesauttanut asiakkaiden parempaan tietoisuuteen tehdyistä muutoksista. Hyväksymistestauksella voidaan oleellisesti parantaa asiakkaan ja järjestelmätoimittajan kommunikaatiota (Liu, 2012). Tätä testausmuotoa tulisi lisätä myös yritys Alfan osalta. Esimerkkinä implementaatiosta voisi olla jatkuva integraatio asiakkaiden omiin testausjärjestelmiin ja pyrkiä siten sitouttamaan asiakasta muutosten testaukseen ennen kuin muutokset ovat tuotannossa. Tämä pitäisi saada asiakkaalle mielekkääksi ja hyödylliseksi toiminnoksi, jolloin testauksen kokonaisuutta saataisiin laajennettua paljon suhteellisen pienillä kustannuksilla.

Yritys Alfassa työparin kanssa työskentely koettiin ristiriitaisena mutta määrittelyjen ja isompien korjausten tai muutostöiden yhteydessä se katsottiin positiivisemmaksi menetelmäksi. Kuitenkin, kuten Poole et al. (2001) tutkimuksessa pariohjelmointia alettiin sopeuttaa enemmän olemassa olevaan työkulttuuriin. Hyvän työparin löytäminen pienestä tiimistä ei välttämättä ole aina helppoa ja riippuu paljolti työkulttuurista ja tehtävistä onko se menestyksekkäästi käytettävissä oleva menetelmä jokaisessa tapauksessa. On hyvin paljon kiinni ihmistyypistä onnistuuko pariohjelmointi henkilöiden välillä (Glass, 2001). Työparin läsnäolo on kuitenkin tärkeää ja sen lisäämistä tietyissä tehtävissä lienee järkevää tämänkin tutkimuksen tulosten kannalta. Työparin läsnäolo voi tukea tai jopa korvata suunnittelupelin osia joissakin pienemmissä tapauksissa. Yritys Alfassa työparin läsnäolo on vähemmän suunniteltua ja perustuu enemmänkin tarpeeseen. Suunnittelupelin myötä virheenkorjaus ja muutokset tulevat todennäköisesti keskittymään paremmin, jolloin parityöskentely tulee myös helpommaksi, koska työparit ovat usein perillä siitä, mitä kukin tekee.

5.3.2 XP:n arvojen soveltaminen

Kommunikaatio katsottiin olevan riittävää mutta siinäkin nähtiin jonkin verran parannettavaa. Tiivis tiimi edesauttaa kommunikaatioita ja yritys Alfassa ei nähdä olevan kynnystä synnyttää kommunikaatioita asiasta tai kysyä tarkennuksia asiaan. XP:n menetelmät tukevat toinen toisiaan (Beck, 1999) ja näin ollen ehdotettujen parannusehdotusten odotetaan lisäävän myös kommunikaatioita sekä tiimin että yksilöiden välillä. Hyväksymistestauksen lisäämisen myötä kommunikaatio asiakkaan ja tiimin välillä tulee myös parantumaan entisestään. Kommunikaation tulee olla avointa ja rehellistä. Negatiivisetkin asiat pitää pystyä tuomaan esille ilman että siitä rangaistaan (Beck, 1999). Näitä Beckin sanoja noudattaen päästään varmasti hyvään tulokseen kommunikaation kanssa. Jokaisen tulee kantaa vastuu kommunikaatioista ja siitä, miten sitä antaa muille ja miten sitä itse ottaa vastaan. Tiimin työtilat eivät saa häiritä tai estää kommunikaatiota ja tiimin jäsenten välinen keskustelu tulee olla mahdollista helposti. Jo eri kerroksessa työskentely katsotaan rajoittavan kommunikaatiota (Beck 1999; Jeffries et al. 2001). Yritys Alfassa tilat ovat kohtuullisella tasolla ja suurin osa tiimin jäsenistä työskentelee huoneissa, joissa on useampi henkilö.

Nopea palaute asiakkaalta auttaa korjaamaan järjestelmän toiminnallisuutta ja kohdentamaan refaktorointia järjestelmän ongelmakohtiin. Nopea palaute kohdistuu kaikkiin prosessin vaiheisiin, kuten esimerkiksi käyttäjätarinoihin, implementointiin, testaukseen ja toimitukseen (Poole et al. 2001). Palautetta ei koettu juurikaan tulevan yritys Alfassa ja mikäli sitä tuli, oli se luonteeltaan negatiivista. Palautekanavissa on parannettavaa ja XP:n menetelmät tulevat niitä myös parantamaan. Hyväksymistestaus antaa lisää palautetta asiakkaalta, yksikkötestaukset antavat palautetta suoraan

järjestelmästä. Suunnittelupeli parantaa palautteen saamista työkavereilta tehdystä ja tehtävistä töistä. Suomalaisilla on tosin taipumus negatiivisuuteen mutta kannattanee kokeilla välillä myös positiivissävytteistä palautetta. Palaute toimii osana kommunikaatioita (Beck, 1999) ja sen nopeus on tärkeää XP:n kannalta.

XP korostaa myös rohkeutta. Rohkeuden avulla saavutetaan päämääriä ja pystytään tekemään XP:n menetelmien kannalta tärkeitä päätöksiä (Beck 1999; Jeffries et al. 2001). Rohkeutta vaaditaan joka tasolla. Ohjelmoijalle vaatii rohkeutta tehdä päätös refaktoroida jokin alue tai tuotteen omistajalle ottaa jokin uusi ominaisuus kehitykseen. Rohkeutta on kantaa vastuu tekemisistään ja rohkeutta on myöntää epäonnistuneensa. XP pyrkii yksinkertaisuuteen jokaisella osa-alueella. Osa-alueita voi olla käyttäjätarinat ja niistä kirjoitettavat spesifikaatiot. Pyri pitämään ne yksinkertaisena, koska kukaan ei hyödy monimutkaisista määrityksistä ja asiat selviävät nopeammin keskustelemalla asiasta (Jeffries et al. 2001). Kun asiat pyritään pitämään mahdollisimman yksinkertaisena, vähentää se kommunikaation kompleksisuutta (Beck, 1999). Yksinkertaisuus suunnittelussa, toteutuksessa, kommunikaatiossa tai missä tahansa tehtävässä on tärkeässä roolissa XP:n menetelmissä. Lähes jokainen tiimin jäsen piti vaikeimpana asiana tietojärjestelmien ylläpidossa järjestelmien laajuutta ja kompleksisuutta. Tämän takia yksinkertaisuus arvona pyrkii vähentämään kompleksisuutta ja helpottaa ymmärtämistä. Laajat tietojärjestelmät sisältävät lähes aina monimutkaisia prosessiketjuja ja ominaisuuksia on paljon. Mikäli kokonaisuuden ymmärtämistä pystytään edes vähän parantamaan yksinkertaisuuteen pyrkimällä, kannattaa se käyttää hyödyksi. Beck (2004) määrittää vielä viidenniksi arvoksi kunnioituksen. Kunnioitus toisten työtä kohtaan on tärkeää ja sen avulla saadaan nostettua työmoraalia tiimissä. Jos kunnioitusta tai arvostusta toisten työtä kohtaan ei ole, aiheuttaa se paljon ristiriitaisia tilanteita, kärjistyneitä tunteita, työn laatu heikkenee ja kokonaisuudesta tulee päivä päivältä huonompi. Kunnioitusta tulee olla myös itseään ja tekemäänsä työtä kohden. Itsekunnioitus parantaa oman työn laatua, koska silloin ei haluta tehdä huonoa jälkeä ja asioita ei jätetä puolitiehen.

Kommunikaation, palautteen ja yksinkertaisuuden korostaminen yritys Alfassa koettaisiin tarpeelliseksi. Nämä XP:n arvot katsottiin olevan vajavaisia ja ne tarvitsivat parantamista. XP:n menetelmien käyttöönotot ja soveltamiset parantavat näitä arvoja. Kunnioituksen ja rohkeuden mittaaminen kerätystä tutkimusdatasta on haastavaa ja niiden tilaa on vaikea kartoittaa. Kuitenkaan ei voida kiistää näiden kahden arvon merkitystä jo yleisellä tasolla ajateltuna.

6. Yhteenveto

Tällä tutkimuksella pyrittiin selvittämään, miten XP:n menetelmät voivat tukea tietojärjestelmäkehityksen ylläpitovaiheen tehtäviä tapaus yritys Alfassa. Tutkimusongelmaa lähdettiin purkamaan ensin ongelmatapausten etsimisellä kyselylomakkeella ja haastatteluilla. Tämän tutkimuksen tulokset ovat laadullisia, jotka perustuvat laadullisella tapaustutkimuksella kerättyihin tietoihin. Kerätyn datan pohjalta muodostettiin käsitys siitä, mitkä XP:n menetelmät olisivat hyödyksi yritys Alfassa. Tutkimuksen tietojen ja tuloksien analysointiin vaikuttaa myös oma henkilökohtainen kokemus tämän tutkimuksen tapaukseen yritys Alfaan.

Tämän tutkimuksen löydösten perusteella XP:n menetelmät tarjoavat useampia käytäntöjä tietojärjestelmien ylläpitovaiheeseen. Parannettavaa löydettiin jokaisesta kysytystä teemasta. Teemat olivat: metaforat ja tarinat, refaktoroinnin tarve, läsnäoleva asiakas, testaus, jatkuva integraatio ja pienet julkaisut, pariohjelmointi sekä kommunikaatio ja palaute. Osa XP:n menetelmistä on jo käytössä yritys Alfassa, vaikka niitä ei XP:n menetelmiin ole ennalta yhdistetty. Näitä olemassa olevia menetelmiä voidaan vahvistaa tämän tutkimuksen tulosten pohjalta ja uusia menetelmiä ottaa käyttöön vahvistamaan näitä. Olemassa olevista käytännöistä käytössä on mm. kollektiivinen omistajuus ja koodaus standardit, mitkä edesauttavat järjestelmän ohjelmoinninhallinnassa. Muitakin käytäntöjä on käytössä mutta ei samalla laajuudella.

Metaforien ja tarinoiden osalta suurimman huomion saa tarinoiden rakenne ja tietosisältö. Huomiota tulee kiinnittää vastaanotettaviin tarinoihin ja sen tarjoaman spesifikaation laatuun. Tarinoiden tulee tarjota vähintään minimimäärä informaatiota muutoksen tai virheenkorjauksen suorittamista varten. Suunnittelupeli tarinoiden arvioinnissa ja yksinkertaisimman ratkaisun löytämiseksi katsotaan tarpeelliseksi. Refaktoroinnin tarve nousi esille tutkimusdatasta, jossa järjestelmää pidettiin kompleksisena ja ohjelmakoodi täytti joissakin osissa refaktoroinnin tarpeen kriteerit. Refaktorointi luo pohjaa uusien muutoksien implementoinnille ja vähentää virheiden määrää sekä niiden paikantamista. Järjestelmän lähdekoodi tulee selkeämmäksi ja helpommaksi lukea.

Läsnäolevan asiakkaan puute tai sen korvaaminen tuotteen omistajalla tai muulla vastaavalla taholla katsottiin joko riittäväksi tai puutteelliseksi. Asiakkaalta saatua informaatioita pidetään tärkeänä. Asiakkaiden kanssa tehtävä yhteistyö lisää määritysten tarkkuutta ja asiakkaalta saatava palaute hyväksymistestauksista on määritysvirheiden löytämisen kannalta merkittävää. XP:n suosimat automatisoidut yksikkötestaukset ja testaus orientoitunut ohjelmointi vähentää selkeästi ohjelmakoodiin pääsevien virheiden määrää. Pariohjelmointia ei juurikaan katsottu positiivisesti, mutta työkaverin läsnäoloa tarvittaessa arvostettiin. Jatkuvan integraation ja pienten julkaisujen avulla saadaan nopeampi toimitussykli asiakkaille. Asiakkaiden tarve virheidenkorjausten nopeille toimituksille parantaa asiakastyytyväisyyttä ja vähentää datan korjausten määrää. Jatkuvaa integraatioita asiakkaiden testipalvelimelle tulee harkita ja kartoittaa sieltä mahdollisesti saatavat hyväksymistestaukset.

XP:n arvot tukevat toinen toistaan sekä ne tukevat myös XP:n käytäntöjä. XP:n menetelmät tukevat toisiaan ja mikään niistä ei toimi hyvin yksinään. XP:n käytännöt luovat pohjan, joka synnyttää kommunikaatioita sekä palautetta. Näitä syntyy joko

prosessin tuotoksena tai tiimin välisinä keskusteluina asiakkaiden kanssa. Jokaisessa menetelmässä tulee pyrkiä yksinkertaisuuteen. Yksinkertaisuus vaatii myös rohkeutta ja kunnioitusta. Tämän tutkimuksen osalta XP:n arvoista eniten esille nousi kommunikaatio ja palaute. Uusien käytäntöjen implementointi tai vanhojen olemassa olevin parantaminen tulee väistämättä kasvattamaan näitä molempia. Tämä kuitenkin tulee tehdä siten, että se tehdään mahdollisimman yksinkertaisesti.

Tämän tutkimuksen rajoituksena on se, että kyseessä on ainoastaan yksi tapaus. Tapaustutkimusten yleistettävyyttä on myös kritisoitu sen erityslaatuisuutensa vuoksi. Jokainen tapaus on hieman erilainen, johtuen esimerkiksi organisaatioissa olevista ihmisistä. Tämä tutkimus kuitenkin antaa edellytykset kartoittaa XP:n menetelmien soveltuvuutta tai tarvetta yleisellä tasolla. Tutkimustulosten luotettavuuden kannalta tehty otanta voi olla liian suppea haastattelujen osalta. Haastatteluista olisi hyvä saada kattavampia.

Tämän tutkimuksen aikana havaittiin, kuinka mielenkiintoista olisi tutkia XP:n implementointia organisaatioon, mikä käyttää jotakin muuta menetelmää. Kuten tässä tutkimuksessa viitatut lähteet osoittavat (Merisalo-Rantanen et al. 2005), (Choudhari & Suman, 2010), (Poole et al. 2001), on organisaatioissa yleensä käytössäkin XP:n tarjoamia menetelmiä vaikkakaan sitä organisaatio ei välttämättä itse tiedosta tai edes ajattele. Mielenkiintoisia mittareita tässä olisi esimerkiksi ylläpitotiimin kokemus ja sen vaikutus XP:n menetelmien omaksumiseen ylläpitovaiheessa.

Toisena mielenkiintoisena aiheena voisi olla ylläpitovaiheen muutoksien ja virheenkorjauksen toteutus ja laatu eri menetelmiä hyväksikäyttäen. Voitaisiin asettaa jotkin mittarit, joilla tarkastellaan järjestelmämuutosten tai virheidenkorjauksen nopeutta ja laatua. Yksistään nopeus ei koskaan riitä, koska se ei kerro onko korjaus tai muutos onnistunut. Mittarit voisivat olla joko laadullisia tai määrällisiä.

Kolmantena aiheena XP:n menetelmien ja prosessien käyttöönoton vaikutus organisaatiolle. Vaikuttaako se pelkästään positiivisesti vai löytyykö esimerkiksi negatiivisia vaikutuksia tuotekehitykselle tai hallinnolle. Hyväksyykö organisaatio kyseiset menetelmän helposti, vai aiheuttaako se ongelmia tai vastarintaa. XP:n menetelmiä voitaisiin implementoida ITIL:in prosesseihin ja tutkia menetelmien vaikutusta siihen. Tutkimus voisi olla toimintatutkimus, jossa prosessin etenemistä tarkastellaan pitemmällä jaksolla ja rakennettuun prosessiin tehdään parannuksia kehityssyklien edetessä.

Lähteet

Bass, J. M. (2012). Influences on agile practice tailoring in enterprise software development. AGILE India (AGILE INDIA), 2012, 1.

Beck, K. (1999). Extreme programming explained: Embrace change. Reading, MA: Addison-Wesley.

Beck, K. and Andres, C. (2004) Extreme Programming Explained: Embrace Change, Second Edition, Addison–Wesley.

Bon, J. (2009), ITIL V3 Taskukirja. Zaltbommel: Van Haren.

Burton Swanson, E. (1976). The dimensions of maintenance. ICSE '76: Proceedings of the 2nd International Conference on Software Engineering,

Cannizzo, F. (2008). Pushing the boundaries of testing and continuous integration. Agile, 2008.AGILE '08.Conference, 501.

Copeland, L. (2001). Extreme programming. Computerworld,

Dekleva, S. M. (1992). The influence of the information systems development approach on maintenance.

Edwards, C. (1984). Information systems maintenance: An integrated perspective. MIS Quarterly, 8(4), 237-256.

English, A. (2002). Extreme programming, it's worth a look. IT Professional, 48.

Fowler M. (1999), Refactoring: Improving the Design of Existing Code, Addison-Wesley, Don Mills, Ontario.

Glass, R. L. (2001). Extreme programming:The good, the bad, and the bottom line.

Hagen, S. (2011). Towards solid IT change management: Automated detection of conflicting IT change plans. Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on, 265.

Hirsjärvi, S. & Hurme H. (1995) Teemahaastattelu. Helsinki: Gaudeamus

Hirsjärvi, S. & Hurme H. (2001) Tutkimushaastattelu – Teemahaastattelun teoria ja käytäntö. Helsinki: Yliopistopaino.

Järvinen, P., & Järvinen A. (2000). Tutkimustyön metodeista. Tampere: Opinpajan kirja.

Jeffries, R., Anderson, A. & Hendrickson, C., (2001), Extreme programming installed. Boston : Addison-Wesley

Jitender Choudhari, D. U. S. (2010). Iterative maintenance life cycle using eXtremeProgramming.

Juric, R. (2000). Extreme programming and its development practices. Information Technology Interfaces, 2000.ITI 2000.Proceedings of the 22nd International Conference on, 97.

Kivi, J. (2000). Extreme programming: A university team design experience. Electrical and Computer Engineering, 2000 Canadian Conference on, 816.

Lahtela, A. (2011). Challenges and problems in release management process: A case study. Software Engineering and Service Science (ICSESS), 2011 IEEE 2nd International Conference on, 10.

Liu, H. (2012). An initial study on refactoring tactics. Computer Software and Applications Conference (COMPSAC), 2012 IEEE 36th Annual, 213.

Liu, L. (2012). Application of agile method in the enterprise website backstage management system: Practices for extreme programming. Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on, 2412.

Martin, J. and C. McClure. (1983). Software Maintenance: The Problem and Its Solution, Englewood Cliffs, NJ: Prentice-Hall.

Martin, J. and McClure, C. (1985). Structured Techniques for Computing, Prentice-Hall, Englewood Cliffs, NJ.

Merisalo-Rantanen Hilkka,Tuunanen Tuure, Rossi Matti. (2005). Is extreme programming just old wine in new bottles:A comparison of two cases. Journal of Database Management, 16(3), 41-61.

Metsämuuronen, J. (2005). Tutkimuksen tekemisen perusteet ihmistieteissä. 3. Laitos, Helsinki : International Methelp.

Miller, A. (2008). A hundred days of continuous integration. Agile, 2008.AGILE '08.Conference, 289.

Murphy-Hill, E. (2008). Refactoring tools: Fitness for purpose. Software, IEEE, 38.

Nordberg,Martin E., I.,II. (2003). Managing code ownership. Software, IEEE, 26.

Office of Government Commerce. (2007) Continual service improvement / Office of Government Commerce. London.

Office of Government Commerce. (2007) Service design / Office of Government Commerce. London.

Office of Government Commerce. (2007) Service operation / Office of Government Commerce. London.

Office of Government Commerce. (2007) Service strategy / Office of Government Commerce. London.

Office of Government Commerce. (2007) Service transition / Office of Government Commerce. London.

Padberg, F. (2003). Analyzing the cost and benefit of pair programming. Software Metrics Symposium, 2003.Proceedings.Ninth International, 166.

Poole, C. J., Murphy, T., Huisman, J. W., & Higgins Allen. (2001). Extreme maintenance.

Rizvi, S. A. M. (2011). A methodology for refactoring legacy code. Electronics Computer Technology (ICECT), 2011 3rd International Conference on, 198.

Sillitti, A. (2012). Understanding the impact of pair programming on developers attention: A case study on a large industrial experimentation. Software Engineering (ICSE), 2012 34th International Conference on, 1094.

Sutherland, J. (2005). Future of scrum: Parallel pipelining of sprints in complex projects. Agile Conference, 2005.Proceedings, 90.

Takeuchi, H. and I. Nonaka, The New New Product Development Came. Harvard Business Review.

White, Steven D. Cronan, Timothy Paul. (1997). Software maintenance: An empirical study of the influence of maintenance characteristics on user information satisfaction. The Journal of Computer Information Systems, 38. 1(The Journal of Computer Information Systems), 1-14.

Liite A. Kyselylomake

Tämän kyselyn tarkoituksena on kartoittaa käytössä olevia tietojärjestelmien ylläpitovaiheen menetelmiä ja löytää mahdollisia ongelmakohtia. Tutkimusongelmana on: Miten Extreme Programming (XP) - menetelmät voivat tukea tietojärjestelmäkehityksen ylläpitovaiheen tehtäviä? Ongelmaa lähestytään tämän kyselyn ja sitä seuraavien haastattelujen avulla. Kysymysten vastaukset analysoidaan ja niiden pohjalta tehdään parannusehdotukset löydettyihin ongelmakohtiin.

Tutkimuksen kannalta olisikin erittäin tärkeää, että vastaat mahdollisimman laajasti ja kattavasti tämän kyselyn kysymyksiin.

1. Sukupuoli

   a. Mies b. Nainen

2. Ikä

3. Työkokemus IT – alalta

4. Asema

5. Kuvaile, miten muutospyynnöt tai tietojärjestelmää koskevat ongelmat ovat kuvattu ja ovatko ne helposti ymmärrettävissä?

6. Ovatko virheenkorjaukset tai muutoksien implementoinnit helposti tehtävissä? Mikä niissä on haastavinta?

7. Miten kuvailet asiakkaan osallistumista muutoksien toteutuksessa?

8. Kuvaile asiakkaiden reaktioita / kommentteja tehtyihin muutoksiin tietojärjestelmässä?

9. Miten muutosten ja virheidenkorjausten jakelu on mielestäsi toiminut tähän asti?

10. Kuvaile työkaverin läsnäoloa muutosten ja virheenkorjausten toteutuksen yhteydessä?

11. Onko muutosten- ja ongelmanhallinnan prosessien osa-alueiden välinen kommunikaatio riittävää? Minkälaista palautetta saat?

12. Mikä on mielestäsi ongelmallisinta tietojärjestelmän ylläpidossa?