Petri Heinua

TUOTEVERSIOINTI OSANA SAAS –PALVELUN HINNOITTELUSTRATEGIAA

JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2016

TIIVISTELMÄ

Heinua, Petri
Tuoteversiointi osana SaaS –palvelun hinnoittelustrategiaa
Jyväskylä: Jyväskylän yliopisto, 2016, 64 s.
Tietojenkäsittelytiede, pro gradu -tutkielma
Ohjaaja: Ojala, Arto

Tuoteversiointi on tehokas keino yritysten työkalupakissa niiden kilpaillessa asiakkaista ja markkinaosuuksista. Versioinnilla asiakkaat voidaan jakaa joko maksukyvyn tai eriävien tarpeiden mukaan valitsemaan itselleen sopivin tuote. Perinteisiin ohjelmistotuotteisiin nähden SaaS, eli verkon yli palveluna tarjottava ohjelmisto, mahdollistaa uudenkaltaiset hinnoittelun ja myynnin mallit ja sen on luvattu tarjoavan skaalaetuja ja helpompaa käyttöönottoa. Multitenantissa SaaS –arkkitehtuurissa yksi sovellusinstanssi palvelee monia asiakkaita siten että asiakkaan käyttökokemus vastaa oman sovelluksen käyttöä ja asiakkaan datat on eriytetty muiden asiakkaiden datasta. Multitenantin arkkitehtuurin mahdollistamalla pienemmällä resurssien kulutuksella ja nopealla käyttöönotolla myös pienempien asiakkaiden palveleminen kustannustehokkaasti on mahdollista. Versioinnin toteuttaminen multitenantin SaaS-arkkitehtuurin yhteydessä on haastava ja uusi ongelma, johon ei ole helppoa ratkaisua. Tässä monitapaustutkimuksena toteutettavassa tutkielmassa tutustutaan neljän eri yrityksen tuotteisiin ja selvitetään miten kyseiset yritykset ovat ratkaisseet tämän ongelman. Tulosten perusteella näyttää siltä että versiointia käytetään yleisimmin erikokoisten asiakkaiden tavoittamiseen. Asiakas- tai toimialakohtaisten versioiden ongelmana voi olla koodin sirpaloituminen ja tästä johtuen koodi tuodaan usein osaksi päätuotetta ja toimiala- ja asiakaskohtaiset ominaisuudet ovat saman tuotteen ajonaikaisesti konfiguroitavia ominaisuuksia.

Asiasanat: versiointi, SaaS, tuotedifferointi, hinnoittelu

ABSTRACT

Heinua, Petri
Product versioning as a part of SaaS pricing strategy
Jyväskylä: University of Jyväskylä, 2016, 64 p.
Computer Science, Master's Thesis
Supervisor: Ojala, Arto

Product versioning is an efficient tool for companies when fighting for customers and market shares. With versioning, the customers can be segmented by their ability to pay or their individual needs to choose the version they see most fit for their purpose. Compared to traditional software products, SaaS (Software-as-a-Service) enables new models of pricing and sales and promises benefits of scale and easier deployment. In a multi-tenant SaaS architecture one application instance serves several customers while allowing a user experience similar to customers' own software and separating the data of different customers. Multi-tenant architecture, by enabling smaller resource usage and efficient deployment, makes serving smaller customers possible and cost-efficient. Product versioning in the context of the multi-tenant SaaS architecture is a challenging and new problem with no easy solution. This multiple case study introduces products from four different companies and finds out how these companies have solved this problem. Based on the results it seems like the most common use of versioning is to target customers of different sizes. The problem of customer- and industry specific versions can be code fragmentation and because of this the code is often merged to the base product and customer- and industry specific features are often taken into use with runtime configurations.

Keywords: versioning, SaaS, product differentiation, pricing

TAULUKOT

SISÄLLYS

1 JOHDANTO

Ohjelmistojen hinnoittelu on jatkuvassa muutoksessa alan kehityksen mukana. Perinteisen tuotemyynnin osuus ohjelmistoyritysten liikevaihdosta on laskemassa, ja painopiste on siirtymässä kohti palveluiden myyntiä (Cusumano, 2007).

Software-as-a-Service (SaaS) –palvelussa sovellus tarjotaan asiakkaalle palveluntarjoajan palvelimelta selaimen yli käytettäväksi, asennuksen ja ylläpidon jäädessä palveluntarjoajan vastuulle. Yksi SaaS -palveluihin usein liitettävä piirre on multitenantti ympäristö: useita asiakkaita palvellaan samalla sovellusinstanssilla. Multitenantilla ympäristöllä saavutetaan skaalaetua asiakaskohtaisten kustannusten pienentyessä ja siksi se onkin suositeltu tapa SaaS -palvelun toteuttamiseen. Se sopii hyvin tilanteeseen, jossa kaikille asiakkaille tarjotaan samanlaista versioita vain pienillä konfiguroitavilla muutoksilla. (Benlian, Hess & Buxmann, 2009.). SaaS -palvelut voidaan luokitella SaaS –kypsyysmallin mukaan neljään eri asteeseen. Kypsyysmallin kahdella ensimmäisellä tasolla jokaista asiakasta palvelee oma sovellus- ja tietokantainstanssi. Kahdella viimeisellä tasolla toimitaan multitenantissa ympäristössä. (Chong & Carraro, 2006.).

Korkea SaaS -maturiteetti ja multitenantti ympäristö sopii hyvin yrityksille jotka pyrkivät kilpailemaan kustannustehokkuudella. Toinen yrityksen kilpailuetu on differointi (Porter, 1985). Tuotedifferoinnilla tarkoitetaan tuotteen erilaistamista siten että se eroaa kilpailijoiden tarjonnasta. Tuotedifferointi voidaan jakaa horisontaaliseen ja vertikaaliseen differointiin. Vertikaalisessa differoinnissa tuotteesta tarjotaan laadullisesti poikkeavia versioita eri hinnoilla. Horisontaalisessa differoinnissa tuotteen versiot poikkeavat toisistaan ominaisuuksiltaan ja soveltuvat siten erilaisille asiakasryhmille. (Katzmarzik, 2011.). Differointia voidaan käyttää kilpailustrategiassa joko markkinajohtajuu- den tavoittelemiseen tai tietyn asiakassegmentin tarpeiden täyttämiseen (Mannion & Savolainen, 2010). Asiakaskohtaiset muokkaukset ovat multitenan- tissa ympäristössä ei-toivottuja ja usein haastavia toteuttaa. (Mietzner, Leymann & Unger, 2011). Korkea SaaS -maturiteetti ja multitenantin ympäristön käyttö ei aina ole itsestäänselvyys.

Lainsäädännölliset seikat, palvelunlaatuun liittyvät vaatimukset ja asiakaskohtaiset vaatimukset voivat olla peruste pysyä asiakaskohtaisissa ympäristöissä (Krebs, Momm & Kounev, 2012). Asiakaskohtaiset muutokset voidaan toteuttaa joko kustomoinnilla tai konfiguraatiolla. Kustomoinnissa muutokset tehdään ohjelmiston lähdekoodiin, kun taas konfiguraatiossa asiakaskohtainen ominaisuus otetaan käyttöön muuttamalla jotain ennaltamääriteltyä parametria (Sun, Zhang, Guo, Sun & Su, 2008).

Ohjelmistojen versioinnin, hinnoittelun ja arkkitehtuurin välillä on selkeitä riippuvuuksia, joita on kokonaisuutena tutkittu vähän. Parhaiten nämä näkökulmat on huomiotu ohjelmistotuoteperheitä koskeva kirjallisuuden parissa, jossa otetaan huomioon organisaatio kokonaisuutena niin liiketoiminnan, arkkitehtuurin kuin prosessienkin näkökulmasta (Van Der Linden, Bosch, Kamsties, Känsälä, & Obbink, 2004). Mannionin ja Savolaisen (2010) mukaan menestyvällä ohjelmistotuotelinjalla on selkeä tavoite, liiketoimintastrategia, kohdemarkkina, ja teknologinen strategia, joka on linjassa kaikkien edellisten kanssa. Arkkitehtuurin, versioiden ja hinnoittelumallin valinnan pitäisi siis lähteä yrityksen liiketoiminnallisista tarpeista ja tukea kilpailustrategiaa. Multitenantin arkkitehtuurin käyttöä SaaS –palveluissa on tutkittu esimerkiksi hinntoittelumallien yhteydessä, jossa tiedetään, että multitenantti arkkitehtuuri asettaa rajoitteita ohjelmiston muokattavuudelle mutta voi toisaalta myös yksinkertaistaa hinnoittelua (Laatikainen & Ojala, 2014). Ohjelmistotuoteperheiden parissa multitenantit SaaS –palvelut ovat vielä melko vähän käsitelty osa-alue, mutta esimerkiksi ohjelmistotuotteiden muokattavuuteen käytettäviä menetelmiä on sovitettu myös multitenanttiin ympäristöön (Jansen, Houben & Brinkkemper, 2010; Kabbedijk & Jansen, 2012) ja näitä tuloksia voitaisiin soveltaa myös käsiteltäessä multitenanttien SaaS –palveluiden versiointia.

Tuotteiden versiointi on SaaS –palvelujen piirissä yleisesti käytetty tapa asiakkaiden eriävien tarpeiden täyttämiseen. Tuoteversiointi voi yksinkertaisimmillaan tarkoittaa jonkin toiminnallisuuden kytkemistä pois päältä halvemmassa versioissa, mutta monessa tapauksessa vaaditaan asiakaskohtaista kustomointia, integraatioita ulkoisiin järjestelmiin ja vastaavia toimintoja, jotka on haastava toteuttaa, jos samalla halutaan multitenantin ympäristön tarjoamat skaalaedut. Empiiristä tutkimusta siitä, miten yritykset ovat ratkaisseet tämän ongelman, tai millä tavalla ne ovat sopeutuneet tilanteeseen tarvitaan ja tämä tutkielma pyrkii täyttämään tämän tarpeen.

Tämä tutkielma käsittelee ohjelmistoversioinnin, arkkitehtuurin ja hinnoittelustrategian välisiä yhteyksiä SaaS –palvelua tarjoavissa yrityksissä. Tavoitteena on selvittää, miten yritykset mukauttavat tuotteensa erilaisille markkinoille ja miten tuotteiden versioinnin ja erilaisten hinnoittelumallien yhdistelmällä voidaan vastata erilaisten asiakkaiden vaatimuksiin.

Tutkimusmenetelmänä on laadullinen monitapaustutkimus. Tutkimusta varten tullaan haastattelemaan asiantuntijatehtävissä toimivia henkilöitä yrityksille suunnattuja SaaS –palveluja tarjoavista yrityksistä. Tutkimuskysymyksenä on “Miten SaaS -palveluntarjoajat hyödyntävät

tuotedifferointia tuotteissaan?". Kysymyksen selvittämiseksi tarvitaan tietoa sekä palvelun kaupallisista, että teknisistä osa-alueista. Kaupallisen osa-alueen osalta voidaan ongelma jakaa alikysymyksiin:

- Mitkä tuotedifferoinnin menetelmät ovat käytössä?
- Minkälaisille asiakasryhmille tuotteen eri versioita tarjotaan?

Teknisten ominaisuuksien ja SaaS –arkkitehtuurin osalta alikysymykset ovat:

- Miten tarjottaviin moduuleihin tai sovellusversioihin ja niiden hinnoitteluun on päädytty?
- Kuinka paljon asiakaskohtaista räätälöintiä tehdään?
- Minkälaisia arkkitehturaalisia haasteita ohjelmiston versiointi on asettanut tuotteelle?

Johdannon jälkeen seuraa kirjallisuuskatsaus luvuissa 2-5. Luvussa kaksi perehdytään hinnoittelua ja tuotedifferointia koskevaan liiketalouden alan kirjallisuuteen ja käsitellään myös yleistä ohjelmistojen tuoteversiointia käsittelevää kirjallisuutta. Luvussa 3 esitellään ohjelmistotuotelinjoja koskevaa tutkimusta erityisesti variaation ja arkkitehtuurien näkökulmasta. Luvussa 4 keskitytään SaaS –arkkitehtuuriin ja luku 5 on yhteenveto kirjallisuuskatsauksesta. Luku 6 esittelee tutkimusmenetelmän. Luvussa 7 esitellään tulokset ja luvussa 8 johtopäätökset. Lopuksi luvussa 9 on yhteenveto.

2 HINNOITTELUN TEORIA

Tässä luvussa käydään läpi hinnoittelun ja yritysten kilpailustrategian teoriaa. Ensiksi käydään läpi hinnoittelustrategioita, erityisesti SaaS –palveluiden näkökulmasta. Kilpailustrategioihin liittyen käydään myös läpi tuotedifferointia ja sitä kautta ohjelmistojen versiointiin liittyvää teoriaa.

2.1 Hinnoittelustrategiat

Hinnoittelustrategiat voidaan yleisesti jakaa kolmeen ryhmään: asiakasarvo-, kustannus- ja kilpailupohjaiseen hinnoittelustrategiaan (Porter, 1980). Kustannuspohjaisessa hinnoittelussa hinnoittelun perustana käytetään tuotteen valmistuksen, myynnin ja markkinoinnin kustannuksia. Lisenssipohjaista ohjelmistomyyntiä käsitelleessä kirjallisuudessa ohjelmistojen monistamisen oletettiin olevan lähes ilmaista, jonka takia kustannuspohjaisen hinnoittelun on katsottu soveltuvan huonosti ohjelmistoalalle (Shapiro & Varian, 1998). SaaS -palveluissa ohjelmiston ajamisesta voi kuitenkin aiheutua erilaisia kustannuksia, riippumatta siitä ajetaanko ohjelmistoa palveluntarjoajan omilla palvelimilla tai kolmannen osapuolen alustoilla kuten AWS tai Azure, jonka takia kustannuksia ei voida jättää huomiotta.

Kilpailupohjaisessa hinnoittelussa hinnat asetetaan kilpailijoiden hintojen mukaan. Ohjelmistoteollisuudessa markkinaosuuden kasvattaminen on tärkeää verkostovaikutusten ja niistä seuraavien lock-in -vaikutusten takia. Sen seurauksena kilpailupohjaisella hinnoittelulla on myös merkittävä rooli arvopohjaisen hinnoittelun rinnalla (Lehmann & Buxmann, 2009). Lock-in riski on olemassa myös SaaS -palvelumarkkinoilla, sillä palvelun käyttöönoton myötä asiakas voi menettää oman datansa kontrollin palvelua tarjoavalle yritykselle (Ma & Kauffman, 2014).

Ohjelmistojen hinnoittelussa asiakasarvoon pohjautuva hinnoittelu on todettu monissa tutkimuksissa taloudellisesti yhdeksi parhaista hinnoittelustrategioista (Ingenbleek, Debruyne, Frambach & Verhallen, 2003),

mutta siitä huolimatta se on vähiten tutkittu ja myös yrityksissä harvoin hyödynnetty (Hinterhuber, 2008). Asiakasarvoon pohjautuvassa hinnoittelussa tuotteen tai palvelun hinnoittelun pohjana toimii asiakkaan tuotteelle asettama arvo. Asiakasarvon mittaaminen on vaikeampaa kuin kustannusten laskeminen tai kilpailijoiden hintaan vastaaminen. Ohjelmistoja suunnitellessa tulisi kuitenkin tietää mitä ominaisuuksia asiakkaat arvostavat ja paljonko he ovat valmiita maksamaan niistä (Harmon, Raffo & Faulk, 2005). Hinnoittelumallin tulisi olla yhteensopiva sekä yrityksen liiketoimintatavoitteiden, että asiakasarvon kanssa (Bontis & Chung, 2010).

Wu, Wortmann ja Tan (2014) ovat tutkineet SaaS -palveluntarjoajien lisensointistrategioita ja huomanneet niiden suosivan arvopohjaista hinnoittelua. Toinen löydös oli, että palveluntarjoajat suuntaavat palvelunsa laajoille markkinoille mittakaavaedun saavuttamiseksi. Useiden erilaisten hinnoittelumenetelmien käyttöönoton nähtiin mahdollistavan asiakkaiden erilaisten preferenssien huomioimisen. (Wu, Wortmann & Tan, 2014.). Palveluna tarjottavien ohjelmistojen on myös huomattu tavoittavan pienemmätkin yritykset, sillä käyttöönotto on edullista verrattuna perinteisiin ohjelmistolisensseihin, ja siten kasvattavan palveluntarjoajien asiakassegmenttiä (Ojala, 2012). Toisaalta yritykset ovat ulkoistaneet ensisijaisesti ohjelmistoja joilla on vähäinen strateginen arvo ja joiden käyttöönoton taloudellinen ja tekninen riski on vähäinen. (Benlian, Hess & Buxmann, 2009.). SaaS -palvelun käyttöönoton myötä asiakas voi menettää oman datansa hallinnan. Datan siirtäminen toiseen palveluun voi tulla kalliiksi ja nostaa vaihtokustannuksia. Asiakkaan korkeat vaihtokustannukset mahdollistavat palveluntarjoajalle joko opportunistisen hinnoittelun jossa asiakkaan tilannetta hyödynnetään korottamalla hintoja tai maltillisempaa hinnoittelustrategiaa markkinoiden segmentoimiseksi ja suoran kilpailun välttelemiseksi. (Ma & Kauffman, 2014.).

SaaS –palvelut joutuvat kilpailemaan toistensa lisäksi myös perinteisempien kaupasta ostettavien ohjelmistotuotteiden kanssa. Joissain tapauksissa tämä voidaan nähdä kilpailuetuna, koska SaaS –palvelu tarjoaa asiakkaalle poikkeavan vaihtoehdon ohjelmistotuotteeseen nähden (Ojala, 2012). Palveluntarjoajan näkökulmasta yksi strategia, jossa SaaS –palveluilla on etu perinteisiin ohjelmistoihin nähden, voi olla niiden asiakkaiden palveleminen jotka käyttävät palvelua vähän tai vaihtelevalla volatiliteetilla (Ma & Seidmann, 2008). SaaS –palvelujen tarjoamisen kustannuksista ja perinteisiin ohjelmistoihin verrattuna heikommasta suorituskyvystä johtuen SaaS –palveluntarjoajat joutuvat lisäksi investoimaan palvelujensa suorituskykyyn ja kustannustehokkuuteen pysyäkseen kilpailussa mukana (Ma & Seidmann, 2008). Yhteisinä haasteina SaaS –palveluntarjoajilla ja ohjelmistotuotteita myyvillä yrityksillä on muun muassa korkeat myynti- ja markkinointikulut sekä viive kassavirrassa tuotteen kehityksen ja myyntitulojen välissä (Tyrväinen & Selin, 2011).

2.2 Hinnoittelumallit

Lehmann ja Buxmann (2009) ovat esittäneet kuusi parametria joita voidaan käyttää ohjelmistojen hinnoittelumallien rakentamiseen:

1. Hinnoittelun pohja
2. Maksutapa
3. Arviointipohja
4. Hintadiskriminaatio
5. Pakettihinnoittelu
6. Dynaamiset hinnoittelustrategiat

Hinnoittelun pohjaan sisältyy sekä hinnoittelun peruste, joka voi olla joko kustannus-, arvo- tai kilpailupohjainen, että interaktiivisuuden määrä eli asiakkaan mahdollisuus neuvotella hinnasta. (Lehmann & Buxmann, 2009). Arvopohjainen hinnoittelu perustuu kustannustehokkuuteen ja laskentaresurssien tehokkaaseen käyttöön ja on altis hintakilpailulle teknologian kehittyessä (Harmon, Demirkan, Hefley & Auseklis, 2009). Maksutapa voi olla kertamaksu, toistuvat maksut tai näiden yhdistelmä. Arviointipohjaan kuuluu hinnoittelun komponenttien lukumäärä, käytöstä riippuvaiset tekijät kuten levytilan tai käytön kesto tai käytöstä riippumattomat tekijät kuten käyttäjien lukumäärä tai käytössä olevien prosessoreiden määrä. Hintadiskriminaatio voi olla ensimmäisen-, toisen- tai kolmannen asteen diskriminaatiota tai moniulotteista. Myös moniulotteinen hintadiskriminaatio on yleisessä käytössä, hinnoittelun perusteena voi olla yhtä aikaa esimerkiksi asiakkaan sijainti sekä myyntivolyymi. Pakettihinnoittelussa tuotteita voidaan tarjota usean tuotteen pakettina tai tuotteen kanssa tarjotaan esimerkiksi ylläpitoa tai tukipalveluita. (Lehmann & Buxmann, 2009.).

Dynaamisiin hinnoittelustrategioihin luetaan seuraavat strategiat:

1. Penetraatiohinnoittelu
2. ”Seuraa ilmaista” –strategia
3. Skimmausstrategia

Penetraatiohinnoittelun tavoitteena on markkinaosuuden kasvattaminen Matalan hinnoittelun avulla. Seuraa ilmaista –strategiassa tuotteesta tarjotaan ilmainen versio, jolla yritetään sitouttaa asiakkaat tuotteeseen. Myöhemmässä vaiheessa tuotteelle voidaan asettaa hinta tai yrittää saada asiakkaat siirtymään maksullisen version pariin. Skimmausstrategiassa hinta asetetaan aluksi korkeammalle mutta sitä lasketaan asteittain. Tavoitteena on ensiksi saavuttaa korkeampi tuotto maksukykyisempien asiakkaiden avulla ja kattaa sillä tuotekehityksen kustannuksia. Hintojen laskulla tavoitetaan laajempi asiakasryhmä ja toisaalta vastataan hintakilpailuun kilpailijoiden tullessa markkinoille (Lehmann & Buxmann, 2009.). Harmon ym. (2005) ovat luokitelleet arvopohjaiset hinnoittelustrategiat penetraatiohinnoitteluun,

skimmausstrategiaan ja hybridihinnoitteluun eli kahden ensimmäisen yhdistelmään. He katsovat penetraatiohinnoittelun soveltuvan markkinoille joilla asiakkaiden varaushinta (reservation price) on alhainen, kun taas skimmausstrategia soveltuu silloin kun asiakkaiden etsintäkustannukset ovat korkeat. (Harmon ym., 2005.).

Freemium –hinnoittelumallissa tuotteesta tarjotaan maksullisen version rinnalla ilmaista versiota. Freemium –hinnoittelu voi pohjautua joko määrään, jakelutapaan tai ominaisuuksiin. Määräpohjaisessa freemiumissa asiakas saa näytekappaleen, esimerkiksi rajoitetun ajan tai rajoitetun määrän toimintoja. Freemium –versiosta voi myös puuttua ominaisuuksia joita maksullisessa versiossa on tai maksullisuus voi pohjautua sovelluksen käyttöön: kaupallinen käyttö vaatii lisenssin. (Pujol, 2010.). Ilmaisen tuoteversion tarjoaminen voi palvella erilaisia tarkoituksia yrityksen kehitysvaiheesta riippuen. Varhaisessa vaiheessa ilmaisella versiolla voidaan rakentaa tuotetta varhaisten omaksujien avulla, myöhemmin sillä voidaan kasvattaa markkinaosuutta ja löytää uusia asiakasryhmiä ja lopulta saavuttaa tarpeeksi suuri käyttäjämäärä lock-in efektin saavuttamiseksi (Günzel-Jensen & Holm, 2015.).

2.3 Liiketoimintastrategiat

Taulukossa 1 on esitetty Porterin (1985) neljä geneeristä liiketoimintastrategiaa. Strategiat on jaoteltu kahdelle akselille yrityksen kilpailuedun lähteen ja markkinoiden laajuuden mukaan.

TAULUKKO 1 Porterin nelikenttä

	Kustannukset	**Differointi**
Laaja	Kustannusjohtajuus	Differointi
Kapea	Kohdistettu kustannusten painotus	Kohdistettu differointi

Mannion ja Savolainen (2010) ovat liittäneet porterin nelikenttään tekniset strategiat jotka ovat yleisesti käytössä ohjelmistotuotelinjojen yhteydessä. Kustannusjohtajuusstrategiaa käytettäessä tuotekehityksen painopiste on yleisten komponenttien uudelleenkäytössä. Kohdistetun kustannuspainotuksen strategiassa tekninen strategia keskittyy referenssiarkkitehtuurin ja eri ohjelmistoversioiden välillä uudelleenkäytettävien komponenttien luomiseen. Differointistrategiaa käytettäessä on tärkeää erilaisten ominaisuuksien yhdistelmien luominen kilpailijoista erottumiseksi, jolloin variaatio sallitaan uudelleenkäytettävyyden kustannuksella. Kohdistetun differoinnin strategiassa tuotekehitys voi keskittyä joko tuotejohtajuuteen, jolloin tuotteesta luodaan variaatioita, joilla vahvistetaan yrityksen imagoa innovaattorina ja markkinajohtajana tai asiakkaan liiketoiminnan ymmärtämiseen, jolloin asiakkaille luodaan

pitkälle kustomoituja kyseisen asiakkaan tarpeeseen räätälöityjä tuotteita. (Mannion & Savolainen, 2010.).

2.4 Verkostovaikutukset

Verkostovaikutuksella tarkoitetaan ilmiötä, jossa tuotteen arvo asiakkaalle kasvaa sitä mukaa kun tuotteen käyttäjien määrä kasvaa. Esimerkkejä positiivisesta verkostovaikutuksesta ovat muun muassa P2P -verkot, tai Facebookin kaltaiset palvelut. Mitä enemmän käyttäjiä näillä palveluilla on, sitä hyödyllisemmäksi ja houkuttelevammiksi ne käyvät muiden käyttäjien näkökulmasta. (Katz & Shapiro, 1985.). Sun, Xie ja Cao (2004) ovat tunnistaneet neljä vaihtoehtoista tuotestrategiaa markkinoille joilla on verkostovaikutuksia:

1. Yhden tuotteen monopolistrategia
2. Teknologialisensointistrategia
3. Tuotelinjan laajennusstrategia
4. Yhdistelmästrategia

Yhden tuotteen monopolistrategia sopii markkinoille joilla ei ole vahvaa verkostovaikutusta ja alhaisen laadun tuotteen valmistutkustannukset ovat korkeat. Tuotelinjan laajennusstrategia on taas kannattavampi kuin yhden tuotteen monopoli– tai teknologialisensointistrategia silloin kun uuden version luomisen kustannukset ovat matalat (Sun, Xie & Cao, 2004). Vahvojen verkostovaikutusten myötä yrityksen kannattaa pyrkiä mahdollisimman nopeaan käyttäjäkunnan kasvattamiseen marginaalivoiton tavoittelun sijaan, sillä kasvava käyttäjämäärä vaikuttaa käytäjien tyytyväisyyteen ja voi aiheuttaa lock-in vaikutuksen jonka jälkeen asiakkaat ovat valmiita maksamaan korkeampaa hintaa palvelusta. (Lee & O'Connor, 2003.).

2.5 Hintadiskriminaatio

Hintadiskriminaatiolla tarkoitetaan tuotteen myymistä eri hintaan eri asiakkaille. Klassisen jaottelun mukaan (Pigou, 1932) hintadiskriminaatio voidaan jakaa kolmeen eri kategoriaan. Ensimmäisen asteen hintadiskriminaatiossa yritys hinnoittelee tuotteensa suoraan asiakkaan maksuhalun mukaan. Yrityksillä kuitenkin on vain harvoin tarkkaa tietoa yksittäisten asiakkaiden maksuhalusta, joten tämä voi käytännössä olla lähes mahdotonta. Kolmannen asteen hintadiskriminaatiossa hinnoittelun perusteena käytetään jotain asiakkaan ominaisuutta, jolla tiedetään olevan yhteys asiakkaan preferensseihin. Tällaisia ominaisuuksia ovat esimerkiksi asiakkaan ikä tai asema, jolloin hinnoittelussa voidaan käyttää eläkeläis- tai opiskelija-alennusta. (Lehmann & Buxmann, 2009). Ohjelmistotuotteiden kannalta

mielenkiintoisin ja myös yleisimmin käytössä oleva (Laatikainen, Ojala & Mazhelis, 2013). on kuitenkin toisen asteen hintadiskriminaatio. Toisen asteen hintadiskriminaatio voidaan jakaa kolmeen tyyppiin, joissa tuote hinnoitellaan joko tuotteen ominaisuuden (versiot), myynnin ajankohdan (early-bird alennus) tai määrän (könttäalennus) mukaan. Kaikki nämä kolme tapaa ovat käytössä myös SaaS -palveluissa. Palvelusta voi olla tarjolla eri ominaisuuksilla varustettuja versioita, hinta voi vaihdella kellon- tai vuodenajan mukaan, tai esimerkiksi osakekursseja tarjoava palvelu tarjoaa halvemmalla viivästytettyjä kurssitietoja. (Lehmann & Buxmann, 2009.).

2.6 Tuotedifferointi

Tuotedifferoinnilla tarkoitetaan tuotteen erilaistamista siten että se eroaa kilpailijoiden tarjonnasta. Versiointi on yksi tuotedifferoinnin muoto (Varian, 1997). Tuotedifferointi, eli tuotteen erilaistaminen kilpailijoiden tuotteista tai oman tuotteen eri versioiden tuottaminen, voidaan jakaa horisontaaliseen ja vertikaaliseen differointiin. (Katzmarzik, 2011). Vertikaalisessa differoinnissa tuotteesta tarjotaan useita erilaatuisia versioita halvimmasta kalleimpaan. Vertikaalisessa differoinnissa asiakkailla on yhteinen käsitys versioiden paremmuusjärjestyksestä. Tämä jakaa asiakkaat markkinasegmentteihin sen mukaan mistä versiosta he ovat valmiita maksamaan omien mieltymystensä perusteella, eli kyseessä on myös toisen asteen hintadiskriminaatio (Bhargava & Choudhary, 2001). Yksi esimerkki vertikaalisesta differoinnista on MacBook, josta on saatavilla useita erikokoisia versioita, suuremmalla ja pienemmällä kiintolevyllä sekä muistin määrällä. Horisontaalisessa differoinnissa tuotteesta tarjotaan rinnakkaisia versioita jotka eroavat ominaisuuksiltaan. Erona vertikaaliseen differointiin on, että ominaisuudet vastaavat asiakkaiden erilaisiin tarpeisiin, eivätkä ole siten vertailukelpoisia (Katzmarzik, 2011).

2.7 Versiointi

Versiointi on ollut yleisessä käytössä ohjelmistoalalla jo hyllystä myytävien ohjelmistotuotteiden ajoista lähtien. Käyttöjärjestelmistä, kuvanmuokkausohjelmistoista ja monista muista ohjelmistoista on ollut useita eri käyttäjäryhmille suunnattuja versioita myynnissä. Sekaannusten välttämiseksi tämän tutkielman yhteydessä ei siis puhuta versioinnissa sovelluksen julkaisuversion (esim. 1.1, 1.2) tai sovelluskoodin versionhallinnan näkökulmasta, vaan kyseessä on aina kaksi eri hintaista ja/tai ominaisuuksiltaan poikkeavaa versiota joita tarjotaan asiakkaille vaihtoehtoina.

Tuotteen laadun heikentämiseen perustuva versiointi on ollut suosittua erityisesti monilla teollisuuden aloilla. On ollut kustannustehokkaampaa valmistaa tuote ensin korkeampaa laatua ja ominaisuuksia vaativille

markkinoille, ja heikentää sen laatua halvemman ja ominaisuuksiltaan huonomman version tuottamiseksi, kuin tuottaa kaksi täysin eri tuotetta. (Denekere & McAfee, 1996.). Versiointi on yksi toisen asteen hintadiskriminaation muoto (Varian, 1997). Versioinnissa asiakkaalle tarjotaan samasta ohjelmistosta useita vaihtelevan laatuisia versioita eri hintaluokkiin. Asiakkaat valitsevat itselleen sopivimman version tuotteen ominaisuuksille asettamansa arvon mukaan.

Ohjelmistojen tuottajille Varianin (1997) tutkimuksessa on kaksi tärkeää huomiota: ensimmäiseksi tuote täytyy suunnitella siten, että se on helposti versioitavissa. Toiseksi tuotteesta täytyy suunnitella ensin täysilaatuinen versio, joka versioidaan myöhemmin markkinoiden alemmille osille laatua heikentämällä. Yksi suosituksista liittyi ohjelmistoversioiden lukumäärään. Niinsanotun "goldilocks- pricing" periaatteen mukaan kahden version sijasta tulisi tehdä kolme eri versiota, sillä joidenkin tutkimusten mukaan keskimmäisellä vaihtoehdolla voidaan houkutella asiakkaita jotka eivät osaa päättää kahden ääripään välillä. (Varian, 1997.). Myöhempi tutkimus on toisaalta kyseenalaistanut kannattaako versiointia tehdä lainkaan, yksi riski on oman tuotteen kannibalisointi: halvemman tuotteen markkinoilletulo voi syödä tuloja, jos asiakkaat valitsevat kalliimman tuotteen sijasta halvemman version, mutta halvemmalla versiolla ei pystytä kasvattamaan markkinaosuutta merkittävästi (Jones & Mendelson, 2011).

Versioinnin on todettu kannattavan esimerkiksi silloin kun tuotteella on verkostovaikutuksia tai lisätulonlähteenä voidaan käyttää esimerkiksi mainostuloja (Bhargava & Choudhary, 2001). Tuotteen ominaisuuksia karsimalla tapahtuvan versioinnin muodon on todettu olevan tehokas markkinoiden laajentamisen keino erityisesti silloin kun asiakkaan arvotukset ominaisuuksien välillä korreloivat negatiivisesti (Hahn, 2001). Versiointi toimii joissain tapauksissa tehokkaana keinona rajoittaa kilpailua. Markkinoille ensimmäisenä päässyt toimija estää kilpailun korkea-arvoisemmissa tuotteissa parantamalla oman tuotteensa laatua. Alempilaatuinen, karsittu tuote tuodaan markkinoille estämään kilpailijoiden pääsy matala-arvoisemmille markkinoille. (Wei & Nault, 2008.).

Miten versiointi sitten kannattaisi toteuttaa? Informaatiotuotteiden tuotedifferointia ja markkinasegmentointia tutkineet Nault ja Wei (2005) ovat määritelleet informaatiotuotteet kokoelmaksi ominaisuuksia (characteristics), jotka voidaan luokitella neljään ryhmään sen mukaan, miten hyvin ominaisuuksia voidaan käyttää tuotteen differoinnissa. Näitä luokituksia voidaan käyttää apuna tehtäessä tuotteen differointia koskevia päätöksiä, luokitukset on esitelty taulukossa 2. Ominaisuuksien valintaan vaikuttaa myös yrityksen liiketoiminta- ja tekninen strategia, Mannion ja Savolainen (2010) ovat esitelleet, miten ohjelmistotuotelinjojen tuottannossa käytettäviä ominaisuusmalleja (feature models) voidaan käyttää erilaisten strategioiden yhteydessä.

TAULUKKO 2 Informaatiotuotteiden ominaisuudet

	Asiakkaiden arviointi	**Versiointi**
1	Kaikki asiakkaat kaikista markkinasegmenteistä arvostavat näitä ominaisuuksia	Nämä ominaisuudet sisällytetään kaikkiin versioihin
2	Korkeamman markkinasegmentin asiakkaat arvostavat näitä ominaisuuksia. Ei arvoa matalamman markkinasegmentin asiakkaille	Vertikaalinen differointi, sisällytetään yleensä korkeammalle markkinasegmentille
3	Yksittäiset asiakkaat eri segmenteissä arvostavat näitä ominaisuuksia ryhmän ja oman yksilöllisen makunsa mukaan.	Tuotteita jotka sisältävät vain tämän ryhmän ominaisuuksia ei tulisi versioida. Jos tuotteella on kuitenkin muita ominaisuuksia jotka mahdollistavat versioinnin, tämän ryhmän ominaisuudet sisällytetään korkeimmille markkinasegmenteille suunnattuihin versioihin.
4	Vain osa asiakasryhmistä arvostaa näitä ominaisuuksia. Ei arvoa muille.	Sisällytetään asiakasryhmäkohtaisiin versioihin

Palveluarkkitehtuurien piirissä tunnetaan käsite granulaarisuus, jolla tarkoitetaan yksinkertaisimmillaan yksittäisen palvelun kokoa. Granulaarisuudella on merkitystä esimerkiksi koodin uudelleenkäytettävyyden kannalta, pienempiä ja laajudeltaan kapeampia palveluita voidaan käyttää useampaan tarkoitukseen ja eriävien versioiden luominen sovelluksesta helpottuu (Katzmarzik, 2011). Granulaarisuus voidaan jakaa kolmeen eri tyyppiin: dataan, liikearvoon ja toiminnallisuuteen. Granulaarisuuden muokkaamisella on erilaiset vaikutukset kaikkien näiden osalta ja suositukset optimaalisen palvelukoon suhteen vaihtelevat sen mukaan (Haesen, Snoeck, Lemahieu & Poelmans, 2008). Katzmarzik (2011) on esittänyt mallin, jonka avulla palveluiden granulaarisuutta voidaan käyttää apuna SaaS -ohjelmiston versioinnissa. SaaS –palveluissa versiointi voidaan toteuttaa itsenäisillä palveluilla, joista asiakas voi valita tarvitsemansa. Liian suuri granulaarisuus (palveluiden suuri määrä) aiheuttaa kustannuksia, joten mallin tarkoituksena oli toimia apuna differointipäätöksiä tehtäessä. (Katzmarzik, 2011.).

3 Ohjelmistotuotelinjat

Tässä luvussa käydään läpi ohjelmistotuotelinjojen teoriaa. Ohjelmistotuotelinjojen piirissä on käsitelty tapoja miten tuoteperheitä voidaan luoda uudelleenkäytettävien resurssien avulla. Tuloksia on sovellettu myös versiointiin ja SaaS –ohjelmistoihin.

3.1 Ohjelmistotuotelinjojen perusteet

Suuri osa ohjelmistojen versiointia koskevasta kirjallisuudesta tulee ohjelmistotuotelinjojen piiristä. Ohjelmistotuotelinjalla (Software Product Line) tar- koitetaan kokoelmaa ohjelmistoja jotka jakavat yhteisiä ominaisuuksia ja jotka on kehitetty tietyn markkinasegmentin tai tavoitteen tarpeet huomioiden joukosta peruskomponentteja. (Clements & Northrop, 1999). Pohl, Böckle ja Van Der Linden (2005) ovat määritelleet ohjelmistolinjojen kehityksen (Software Product Line Engineering) seuraavasti:

> Ohjelmistolinjojen kehitys on ohjelmistojen (ohjelmistoriippuvaisten järjestelmien ja ohjelmistotuotteiden) kehittämiseen alustoja ja massakustomointia käyttäen (Pohl, Böckle ja Van Der Linden, 2005, s.14)

Ohjelmistotuotelinjan ja ohjelmistotuoteperheen (Software Product Family) käsitteitä käytetään monesti synonyymeina. Geyerin ja Beckerin (2002) mukaan käsitteet lähestyvät asiaa eri näkökulmasta ja niiden merkittävin ero on se, että ohjelmistotuotelinja kuvaa toisiinsa liittyviä järjestelmiä niiden yhteisten ominaisuuksien kautta, kun taas ohjelmistotuoteperhe kuvaa kokoelman järjestelmiä jotka on rakennettu yhteisiä komponentteja käyttäen (Geyer & Becker, 2002).

Koodin ja ohjelmistokomponenttien uudelleenkäyttöä edistämällä pyritään nopeampaan markkinoillepääsyyn ja kulujen vähentämiseen. Ohjelmistotuotelinjan keskeisiä aktiviteetteja ovat ydinkomponenttien kehittäminen, tuotekehitys ja hallinto. Nämä aktiviteetit kytkeytyvät toisiinsa iteratiivisina pro-

sesseina ja voivat ilmetä missä tahansa järjestyksessä. Ydinkomponenttien ja tuotteiden kehittäminen voi tapahtua joko rakentamalla tuotteita ydinkomponenteista tai johtamalla komponentit olemassaolevista tuotteista. (Clements & Northrop, 1999.). Krueger (2001) kutsuu näitä proaktiiviseksi ja ekstraktiiviseksi malliksi ja tunnistaa lisäksi vielä reaktiivisen mallin. Ekstraktiivinen malli, eli olemassaolevista tuotteista lähteminen, sopii erityisesti silloin kun asiakkaalta tulee usein odottamattomia vaatimuksia. Proaktiivinen malli soveltuu käyttöön, kun asiakkaiden vaatimukset ovat hyvin tiedossa ja toteutus voidaan suunnitella hyvin etukäteen. Reaktiivisessa mallissa tuotelinjaa päivitetään vasta uuden vaatimuksen ilmaantuessa ja se soveltuu tilanteeseen jossa asiakkaiden odottamattomat vaatimukset ovat yleisiä. (Krueger, 2001.).

Bosch (2006) tunnistaa kolme eri lähestymistapaa ohjelmistotuoteperheiden rakentamiseksi:

1. Integraatio-orientoituneen
2. Hierarkisen
3. Kompositio-orientoituneen

Integraatio-orientoituneessa lähestymistavassa tuotetta ja alustaa kehittävän organisaation välillä on tiukka jako. Komponenttitiimit kehittävät kokoelman komponentteja jotka integroidaan alustaan. Kehitysalusta julkaistaan suurena, integroituna ja testattuna ohjelmistona jonka rajapintoja tuotetiimit voivat käyttää rakentaakseen tuotteensa sen päälle. Hierarkisessa lähestymistavassa uudelleenkäytettävät komponentit ja ohjelmistotuoteperheen arkkitehtuuri järjestetään hierarkisesti. Ainoastaan ominaisuudet jotka kaikki tuotteet jakavat toteutetaan kehitysalustaan yhteisen tuotekehitystiimin toimesta. Kompositio-orientoituneessa lähestymistavassa komponentit ovat vapaasti yhdisteltävissä. Koostettavuus (compositionality) saavutetaan yhteisten arkkitehturaalisten periaatteiden ja sääntöjen kautta. (Bosch, 2006.). Komponenttien kehitys voidaan hoitaa joko niiden kehittämistä varten perustetun tiimin tai tuotekehitystiimien toimesta. Molemmissa tavoissa on omat haasteensa, kehitys voi joko keskittyä liikaa alustaan asiakkaiden tarpeiden jäädessä liian vähälle huomiolle, tai alustan komponenttien kehitys voi ohjautua liikaa seuraavan tuotteen vaatimusten mukaan. (Birk, Heller, John, Schmid, von der Maßen, & Muller, 2003). Ensimmäisten yhteisten komponenttien valinnassa voidaan käyttää kolmea tapaa. Liikkeelle voidaan lähteä helpoimmista komponenteista, eli valita vanhimmat, yleiskäyttöisimmät ja arkkitehtuurissa matalalla tasolla olevat komponentit, jotka ovat hyvin ymmärrettyjä ja laajasti käytettyjä. Toinen tapa on valita olemassa olevia komponentteja jotka ovat yhä aktiivisessa kehityksessä. Tällöin nykyinen tuotekohtainen kehitystyö voidaan mahdollisesti keskittää yhteen yleiseen komponenttiin. Kolmas tapa on aloittaa uusista komponenteista, joille on käyttöä useammassa tuotteessa. Olemassaolevia komponentteja ei korvata, joten kehitys- ja ylläpitokustannukset pysyvät matalina. (Bühne, ym. 2003.).

Ohjelmistotuotelinjan käyttöön siirryttäessä täytyy ottaa huomioon konteksti, jossa siirtymä tapahtuu. Kontekstiin sisältyvät markkinat, organisaatio ja sen sisällä liiketoimintayksikkö ja yksilöt. Liian pienet markkinnat eivät esimerkiksi jätä tilaa tuoteyhdistelmien hyödyntämiselle. Markkinoilla olevien tuotteiden ja kohderyhmien tuntemus auttaa ennustamaan tuotteiden kehitystä ja siten parantaa tuotelinjoihin sijoittamisen todennäköistä tuottoa. Organisaation tasolla vaikuttavia tekijöitä ovat esimerkiksi organisaation asema markkinoilla ja asiakkaiden liiketoiminnan tunteminen. Lisäksi myös liiketoimintayksikön kokemus toteutettavista tuotteista sekä olemassaolevien assettien määrä helpottavat siirtymää. Yksilötasolla yksilöiden kokemus, koulutus ja motivaatio ovat helpottavia tekijöitä. (Bühne, Chastek, Käkölä, Knauber, Northrop, & Thiel, 2003.). Selkeällä organisaatiorakenteella ja selkeillä rooleilla, vahvalla kulttuurilla, organisaation sitoutumisella, muutostenhallinnalla ja oppimisen tukemisella on havaittu olevan positiivinen vaikutus ohjelmistotuotelinjoihin siirtymisen onnistumiseen, ohjelmistotuotelinjojen tehokas hyödyntäminen vaatii siis koko organisaation tuen taakseen. (Ahmed, Capretz, & Sheikh, 2007.).

3.2 Ohjelmistotuotelinjojen maturiteettimallit

Bosch (2002) on kehittänyt ohjelmistotuotelinjojen maturiteettimallin jossa ohjelmistotuotelinja voi sijoittua kuudelle eri tasolle:

1. Standardisoitu infrastruktuuri
2. Kehitysalusta
3. Ohjelmistotuotelinja
4. Konfiguroitava ohjelmistotuotelinja
5. Ohjelmistotuotelinjaohjelma
6. Ohjelmistopopulaatio

Standardisoitussa infrastruktuurissa yrityksessä on yhteisesti käytössä oleva infrastruktuuri, johon kuuluu käyttöjärjestelmä ja mahdolliset kaupalliset komponentit, joita tuotekehityksessä hyödynnetään. Seuraavalla tasolla tuotteet rakennetaan käyttämällä kehitysalustaa, joka sisältää kaikki tuotteille yhteiset ominaisuudet. Ohjelmistotuotelinjoihin siirryttäessä myös toiminnallisuudet jotka ovat yhteisiä vain osalle tuotteita tulevat osaksi kehitysalustaa. Vakaassa toimintaympäristössä kehitys voi siirtyä kohti konfiguroitavia ohjelmistotuotelinjoja, jolloin usean erillisen tuotteen kehityksestä siirrytään kehittämään yhtä konfiguroitavaa tuotepohjaa. (Bosch, 2002.). Konfiguroitavan ohjelmistotuotelinjan on katsottu sopivan erityisesti asiakkaan tarpeisiin vastamiseen, ja sen etuna on nopea käyttöönotto. Haitaksi voidaan katsoa korkeat kehitys- ja ylläpitokustannukset (Niemelä, 2005). Ohjelmistotuotelinjaohjelmaa voidaan käyttää erityisesti laajoihin järjestelmiin. Arkkitehtuuri määrittelee järjestelmän käyttämät komponentit joista joko osa

tai kaikki ovat itsessään ohjelmisto- tuotelinjoja. Ohjelmistopopulaatiossa taas järjestelmä voidaan koostaa vapaasti komponenteista jotka ovat ohjelmistotuotelinjoja. (Bosch, 2002.).

Raatikainen, Soininen, Männistö ja Mattila (2003) ovat tehneet empiiristä tutkimusta konfiguroitavien ohjelmistotuoteperheiden käytöstä. Tutkimukseen osallistuneet yritykset katsoivat konfiguroitavuuden helpottavan asiakaskohtaisia käyttöönottoja. Organisaatioissa kehitys- ja käyttöönotto-organisaatiot olivat monesti eriytetyt. Käyttöönotosta, eli konfiguroinnista vastasi työntekijä jolla oli toimialan tuntemusta. Konfiguroitavuuden haittapuolia ovat kallis kehitys ja ohjelmiston kehittämisen vaikeutuminen. (Raatikainen, ym. 2003.). Jaettujen komponenttien käytön kokemuksia kartoittaneessa tutkimuksessa osallistuneet yritykset kokivat komponenttien käytön vähentävän kehitykseen kuluvaa aikaa ja nostavan tehokkuutta. Toisaalta komponenttien käytön nähtiin pahimmillaan aiheuttavan viiveitä ja ongelmia koko ohjelmistotuotelinjan halki, jos yhteisistä komponenteista löytyi virheitä. (Sharma, Aurum & Paech (2008.).

Van Der Linden, Bosch, Kamsties, Känsälä, ja Obbink (2004) ovat esitelleet neliulotteisen arviointimallin jota yritys voi käyttää organisaationsa ohjelmistotuotelinjojen kehityksen arviointiin. BAPO (Business, Architecture, Process, Organization) –mallin neljä ulottuvuutta ovat liiketoiminta, arkkitehtuuri, prosessi ja organisaatio. Liiketoimintatasoon sisältyy organisaation kyky hallita, ennustaa ja ohjata kehityksen kuluja ja tuottoa. Arkkitehtuuritason kehitystasot vastaavat kahta viimeistä kohtaa lukuun ottamatta Boschin (2002) maturiteettimallia. Prosessitasolla arvioidaan ohjelmistokehityksen prosessia. Organisaatiotasolla arvioidaan organisaation kykyä hallita erilaisia rooleja ja organisaation sisäisiä yhteyksiä. Taulukossa 3 on esitetty kehitystasot kaikille neljälle ulottuvuudelle Van Der Linden, ym. (2004) mukaan.

TAULUKKO 3 BAPO –maturiteettitasot

Taso	**Liiketoiminta**	**Arkkitehtuuri**	**Prosessi**	**Organisaatio**
1	Reaktiivinen	Itsenäiset tuotteet	Alustava	Yksikkökeskeinen
2	Tietoinen	Standardisoitu infrastruktuuri	Hallittu	Liiketoimintayksikkö
3	Ekstrapoloiva	Kehitysalusta	Määritelty	Liiketoimintaryhmä
4	Proaktiivinen	Tuotevariantit	Mittaamalla hallittu	Organisaatioiden välinen yhteistyö
5	Strateginen	Itsekonfiguroitavat tuotteet	Optimoiva	Avoin

3.3 Muunneltavuus

Yksi ohjelmistotuotelinjojen keskeisimmistä käsitteistä on muunneltavuus (variability). Bachmannin ja Clementsin (2005) mukaan muunneltavuuden tarkoitus on:

> Maksimoida sijoitetun pääoman tuotto (ROI) tuotteiden rakentamisessa ja ylläpidossa ennaltamäärätylle ajalle tai tuotteiden lukumäärälle (Bachmann & Clements, 2005, s. 10).

Svahnberg, Van Gurp ja Bosch (2005) määrittelevät muunneltavuuden ohjelmiston tai artefaktin kykynä olla tehokkaasti laajennettavissa, vaihdettavissa, kustomoitavissa tai konfiguroitavissa käytettäväksi tietyssä kontekstissa. Variantti on yksittäinen ominaisuus joka voi muuttua sovelluksen eri versioiden välillä. Variantin kiinnittämistä ohjelmistoversioon kutsutaan sidonnaksi. (Svahnberg, ym. 2005.). Muunneltavuus voidaan jakaa sisäiseen ja ulkoiseen muunneltavuuteen. Sisäisen muunneltavuuden vaikutukset näkyvät ainoastaan kehittäjille, ulkoisen muunneltavuuden vaikutukset ovat myös sovelluksen käyttäjän nähtävissä. Sisäisen muunneltavuuden lähteitä ovat erilaiset tekniset ratkaisut esimerkiksi skaalautuvuuteen tai siirreettävyyteen liittyen. Ulkoisen muunneltavuuden lähteitä ovat muun muassa lait ja asetukset sekä asiakkaiden vaatimukset. (Pohl, ym., 2005.). Ohjelmiston kehittäjät tarvitsevat kolmenlaista tietoa voidakseen tehdä päätöksiä oikeiden muunneltavuustapojen valitsemiseksi (Bachmann & Clements, 2005):

1. Käytettävissä olevat muunneltavuusmekanismit
2. Tuotetietous
3. Tuotantostrategia

Muunneltavuusmekanismeista tarvitsee tietää esimerkiksi minkälaisille komponenteille ne soveltuvat ja minkälaisia teknisiä taitoja ja välineitä niiden käyttö vaatii. Tuotetietouteen sisältyy muun muassa tieto tuotteen eri versioissa vaadittavista ominaisuuksista ja niihin sisältyvistä laatuvaatimuksista. Tuotantostrategiaan liittyy muun muassa päätös siitä, tehdäänkö tuotteita komponentit edellä, vai lähdetäänkö liikkeelle sarjasta tuotteita, joista yleistettävät komponentit johdetaan. (Bachmann & Clements, 2005.).

Ominaisuusmallit (feature models) ovat yleinen tapa mallintaa ohjelmistotuotelinjan sisällä esiintyviä yhteneväisyyksiä (commonality) ja muunnelta- vuutta. Ominaisuusmalli sisältää kokoelman pakollisia, valinnaisia ja vaihtoehtoisia ominaisuuksia joista tuote voidaan rakentaa. Ominaisuuksien valintaan vaikuttaa tyypillisesti laatuominaisuudet ja asiakkaan vaatimukset sekä tekniset rajoitteet. (Kang, Cohen, Hess, Novak & Peterson, 1990.). Lee & Kang (2010) lisäsivät myös käyttökontekstin tärkeäksi tekijäksi ominaisuuksien valintaprosessissa. Lee ja Kang (2010) mukaan ominaisuuksien valinnassa tulisi ottaa huomioon myös tuotteen fyysinen, sosiaalinen ja liiketoiminnallinen

konteksti. Seikat kuten sijainti (esimerkiksi fyysisen ympäristön rajoitteet), kulttuuri (tavat ja lait) sekä asiakkaat (maksukyky) tulisi ottaa huomioon jo tuotetta suunniteltaessa. (Lee & Kang, 2010.).

4 Software as a Service

Tässä luvussa käydään läpi SaaS –sovellusten erityispiirteitä arkkitehtuurin näkökulmasta. Aluksi esitellään pilvipalveluihin liittyviä käsitteitä ja niiden syntyhistoriaa, sen jälkeen käydään läpi pilvipalvelualustat ja pilvipalveluiden ekosysteemiä ja lopussa esitellään SaaS –sovellusten arkkitehtuuria, niiden arviointiin soveltuvia maturiteettimalleja ja erilaisia sovellusten konfigurointi- ja kustomointimenetelmiä.

4.1 Ohjelmisto palveluna

Ohjelmistojen tarjoaminen palveluna nousi pinnalle jo 1990 -luvun lopulla palveluntarjoajamallin (Application Service Provisioning) myötä. ASP -mallissa asiakkaat vuokrasivat sovelluksen palveluntarjoajalta, jonka konesalissa asiakaskohtaisia instansseja ajettiin. Asiakkaat käyttivät sovellusta usein selaimen kautta. (Xin & Levina, 2008.). Sovellusvuokraus ei kuitenkaan menestynyt odotetusti ja Software-as-a- Service (SaaS) -ohjelmifstot ilmestyivät sen korvaajaksi uudella arkkitehtuurilla. Ideana oli edelleen tarjota sovellus asiakkaalle palveluntarjoajan palvelimelta selaimen yli käytettäväksi, asennuksen ja ylläpidon jäädessä palveluntarjoajan vastuulle. Uusina piirteinä mukaan tuli multitenantti ympäristö: useita asiakkaita palveltiin samalta sovellusinstanssilta, jolloin asiakaskohtainen kustomointi jäi vähäisemmäksi mutta samalla kustannuksia saatiin pienennettyä palvelinten tehokkaamman hyödyntämisen myötä. (Xin & Levina, 2008.). Ohjelmistopalvelujen ensimmäisessä aallossa yritykset ulkoistivat vähemmän liiketoimintakriittisiä sovelluksia, kuten asiakkuudenhallinta-, palkanlaskenta- tai yhteistyötyökaluja (Benlian, Hess & Buxmann, 2009). Salesforce.com oli ensimmäinen suuri menestystarina ja monet yrittäjät ovat seuranneet sen jalanjäljissä. Asiakkaiden kokemuksen kertyessä ja muun muassa tietoturvaan ja luotettavuuteen liittyvien huolien hälvetessä ohjelmistopalvelujen nähdään todennäköisesti siirtyvän yhä lähemmäksi asiakkaiden liiketoiminnan ydintä. (Dubey & Wagle,

2007.). Cusumano (2009) arvioi siirtymän tapahtuvan vaiheittain: vanhojen sovellusten uudelleenkirjoittaminen ja datan siirtäminen suljetuista ympäristöistä ja tietokannoista vie aikaa.

SaaS -termille ei ole yleisesti hyväksyttyä määritelmää ja sitä käytetäänkin hyvin monenlaisten palveluiden yhteydessä. Mäkilä, Järvi, Rönkkö ja Nissilä (2010) ovat kartoittaneet kirjallisuutta ja tunnistaneet viisi piirrettä jotka tyypillisesti yhdistetään SaaS: iin:

- Tuotetta käytetään selaimen kautta
- Tuotetta ei ole räätälöity erikseen asiakaskohtaisesti
- Tuote sisällä ohjelmistoa joka täytyy asentaa asiakkaan luona
- Tuote ei vaadi erityistä integrointi- tai asennustyötä
- Tuotteen hinnoittelu on käyttöpohjaista

Monet määritelmät lukevat myös multitenantin ympäristön yhdeksi SaaS -palvelun ominaisuudeksi. Mäkilä ym. (2010) toteuttivat empiirisen tutkimuksen, jossa tutkittiin ylläolevien kriteerien toteutumista suomalaisissa ohjelmistoyrityksissä. Tuloksista tunnistettiin "pure SaaS" -ryhmä, jossa kaikki ehdot toteutuivat ja "high SaaS" -ryhmä jossa toteutui neljä ehdoista. "pure SaaS" oli erittän harvinaista toteutuen yrityksistä vain neljän prosentin kohdalla. Erottava tekijä "pure SaaS" ja "high SaaS" ryhmien välillä oli, että "high SaaS" ryhmän tuotteet vaativat asiakaskohtaisia julkaisuja. (Mäkilä, ym., 2010.).

SaaS:in etuja asiakkaan näkökulmasta ovat muun muassa käyttöönoton helppous ja halpuus, nopeammassa syklissä tapahtuvat automaattiset päivitykset, sekä ylläpidon ulkoistus palveluntoimittajalle (Ju, Wang, Fu, Wu & Lin, 2010). SaaS -palvelut ovat mahdollistaneet uudenlaisia hinnoittelumalleja, esimerkiksi käyttöperustainen hinnoittelu on hyvin yleistä ja myös erilaisten hinnoittelumallien yhdistelmät ovat mahdollisia (Laatikainen, Ojala & Mazhelis, 2013). Tuotetta voidaan tarjota täysin ilmaiseksi, mainosten kanssa tai freemium -mallilla jossa yritys tarjoaa ilmaisen version ohjelmistostaan ja houkuttelee asiakkaat sen avulla paremman, maksullisen version piiriin. (Cusumano, 2008).

Suomalaisia ASP ja SaaS –palveluntarjoajia tutkineet Luoma, Rönkkö ja Tyrväinen (2012) pisteyttivät yritysten tuotteita tuotteen erilaisten ominaisuuksien perusteella ja löysivät pisteytyksen pohjalta kaksi erilaista liiketoimintamallia ASP- ja SaaS –palveluille, jotka on esitetty taulukossa 4.

TAULUKKO 4 Ohjelmistopalveluiden –liiketoimintamallit

	ASP	**SaaS**
Pure-play	Pure-play ASP	Pure-play SaaS
Enterprise	Enterprise ASP	Enterprise SaaS

ASP-sarakkeen yrityksiin kuuluivat yritykset, jotka tarjosivat ohjelmistoa selaimen kautta käytettävänä, erona SaaS –sovelluksiin ne kuitenkin olivat monesti asiakaskohtaisia ohjelmistoja ja vaativat integraatiota käyttöönoton yhteydessä. Pure-play ja Enterprise mallien eroina olivat muun muassa asiakkaan koko ja tuotteen kompleksisuus. Pure-play –mallin tuotteet olivat

yksinkertaisia, niitä myytiin usein netissä ja käyttöönotto oli helppoa. Enterprise –mallin tuotteet taas olivat monimutkaisempia ja niiden käyttöönotto vaati vähintäänkin koulutusta. Kolmantena SaaS –kategoriana tunnistettiin myös Self-service SaaS, joka on niin pitkälle yksinkertaistettu ja standardisoitu sovellus, että käyttäjät voivat ostaa ja ottaa sen käyttöön täysin omatoimisesti. Taulukossa 5 on esitelty näiden kolmen mallin ominaisuudet. (Luoma, Rönkkö ja Tyrväinen, 2012.).

TAULUKKO 5 SaaS –liiketoimintamallien ominaisuudet

	Pure-play SaaS	**Enterprise SaaS**	**Self-service SaaS**
Sovellus	Standardoitu web-sovellus	Massakustomoitu, mutta kompleksi sovellus joka vaatii tukipalvelun	Erittäin yksinkertainen ja helposti käyttöönotettava sovellus
Tulonlähde	Käyttöönottomaksu ja toistuva maksu	Käyttöönottomaksu, toistuva maksu ja palvelumaksut	Freemium, mainokset tai pieni toistuva maksu
Kohderyhmä	Pienet ja keskisuuret yritykset, kohteena keskijohto ja loppukäyttäjät	Suuret yritykset, kohteena IT-johtajat ja ylempi johto	Loppukäyttäjät ja yksittäiset kuluttajat
Myynti	Yksittäisten asiakkaiden tarpeisiin keskittyvää	Luottamukseen perustuvat kumppanuudet ja räätälöidyt sopimukset	Automatisoitu itsepalvelukäyttöönotto
	Vaatii osaamista sekä asiakkaan liiketoiminnasta että sovelluskehityksestä	Henkilökohtainen ja konsultoiva myynti, kumppanuuskanavat	Mainonta ja viraalimarkkinointi
Resurssit	Kehityskustannukset voivat olla korkeat mutta tavoite pienissä käyttökuluissa	Liiketoimintaosaaminen ja kumppanuusyritykset resursseina	Käyttökustannukset erittäin pienet

4.2 Pilvialustat

Läheisesti SaaS –termiin liittyvät myös Platform as a Service (PaaS) sekä Infrastructure as a Service (IaaS). Infrastruktuuri palveluna tarkoittaa yksinkertaisesti levytilan ja laskentatehon, eli laitteiston tarjoamista palveluna virtuaalisesti verkon yli. Alusta palveluna (PaaS) tarjoaa lisäksi sovelluskehyksiä ja rajapintoja sovellusten rakentamista varten, esimerkkejä PaaS –palveluntarjoajista voisivat olla vaikkapa Microsoft Azure tai Salesforcen force.com. Taulukossa 6 on esitelty pilvipalveluiden kenttää esittelevä viitekehys (Kushida, Murray, & Zysman, 2012).

TAULUKKO 6 Pilvipalvelujen viitekehys

	Laitteentarjoaja	**Verkon tarjoaja**	**Pilvipalvelun tarjoaja**
Sovellus	Sovellukset	-	SaaS
Alusta	Laitteen palvelut	Verkostopalvelut	PaaS
Infrastruktuuri	Käyttöjärjestelmä ja laitteisto	Verkostoinfra	IaaS

Viitekehyksen vaaka-akselilla on palveluntarjoajan tyyppi, eli joko laitteiston, verkon tai pilvipalvelun tarjoaja. Pystyakselilla on arkkitehtuurikerros, jossa pohjalla on fyysinen infrastruktuuri, keskellä sovellusalusta ja korkeimpana itse sovellus. Tämän tutkielman mielenkiinnon kohteena on pääasiassa viimeinen sarake, eli pilvipalvelujen tarjoajat. Pilvipalveluissa yksi kehityssuunta on siirtyminen pilvipalvelualustoihin. Siirtymää ovat tehneet sekä aiemmin SaaS kuin IaaS –palveluja tarjonneet yritykset. Esimerkkinä edellisestä Salesforcen Force3.com ja jälkimmäisestä Amazonin AWS. (Kushida, Murray, & Zysman, 2012.). Salesforce.com perustettiin vuonna 1999 tarjoamaan asiakkuudenhallintajärjestelmää pilvipalveluna (SaaS). Se oli aikanaan ketterä ja halpa vaihtoehto kalliille ja monimutkaisille järjestelmille. Myöhemmin salesforce siirtyi PaaS –tarjoajaksi, Force.com palvelun avulla kehittäjät voivat rakentaa sovelluksia jotka integroituvat Salesforce.com rajapintoihin ja joita ajetaan Salesforce.com infrastruktuuria käyttäen. (Kushida, Murray, & Zysman, 2012.). Rajapintojen avaamisella ja PaaS –tarjoajaksi siirtymällä on mahdollista päästä hyötymään verkostovaikutuksista, kehittäjien määrän ja ekosysteemin kasvaessa alustan, ja sitä kautta oman sovelluksen käyttö ja suosio kasvaa (Jacobson, Woods & Brail, 2011). PaaS –alustan käyttäjät tekevät myös liiketoimintaa alustan avulla, joten vaatimukset alustaa kohtaan voivat olla kovat, Giessmannin ja Stanoevskan (2012) toteuttaman kyselyn mukaan alustasta täytyisi löytyä tuki vähintään viidelle ominaisuudelle:

- 99% saavutettavuus
- Täysin automatisoitu skaalautuvuus
- Standardisoitu ohjelmointirajapinta
- Korkeat tietoturvastandardit ja pääsynhallinta
- Varmuuskopiointi ja palautustoiminnot

Ohjelmistoekosysteemeihin siirtymisen puolesta on useita argumentteja (Bosch, 2009):

- Oman sovelluksen asiakasarvon lisäys
- Uusien asiakkaiden houkuttelu
- Asiakkaiden sitouttaminen tuotteeseen
- Innovaation lisääminen ekosysteemissä
- Kehitystyön jakaminen partnereiden kanssa vähentää kustannuksia

Kiinnostus mashupeja, eli avoimia rajapintoja hyödyntäviä web –sovelluksia, kohtaan on noussut viime vuosina. Rajapintojen avulla tarjotaan aina joko

informaatiota tai palvelua, jolla on oltava jokin liiketoiminnallinen hyöty. Rajapintojen avaamista voidaan käyttää keinona yrityksen brändin laajentamiseksi, niche –markkinoiden tavoittamiseksi, laajemman sovellusalusta- ja laitetarjonnan saavuttamiseksi tai omaan tuotteeseen liittyvien innovaatioiden edistämiseksi. (Jacobson, ym. 2011.).

4.3 SaaS -arkkitehtuuri

Multi -tenancy on SaaS –arkkitehtuurimalli, jossa yhdellä sovellusinstanssilla palvellaan useita asiakkaita. Kabbedijk, Bezemer, Jansen ja Zaidman (2015) määrittelevät multi- tenancyn:

> sellaisen järjestelmän ominaisuudeksi jossa useat asiakkaat (tenants) läpinäkyvästi jakavat järjestelmän resursseja, kuten palveluita, sovelluksia, tietokantoja tai laitteistoresursseja tavoitteena alentaa kustannuksia pystyen silti konfiguroimaan järjestelmää vastaamaan asiakkaan tarpeita (Kabbedijk, Bezemer, Jansen & Zaidman, 2015, s. 17).

Bezemerin ja Zaidmanin (2010) mukaan multi-tenancyn kolme pääpiirrettä ovat:

1. Sovelluksen kyky jakaa laitteistoresursseja
2. Palvelun korkean konfiguroitavuusasteen tarjoaminen
3. Arkkitehturaalinen lähestymistapa jossa käyttäjät käyttävät yhtä sovellus- ja tietokantainstanssia

Multitenantin arkkitehtuurin mahdollisia hyötyjä palveluntarjoajalle ovat laitteistoresurssien tehokkaampi käyttö, helpompi ja halvempi sovelluksen ylläpito, pienemmät käyttökulut ja mahdollisuudet tehokkaampaan tiedon keräämiseen ja analytiikkaan (Bezemer & Zaidman, 2010). Multi-tenancy asettaa kuitenkin haasteita sovelluksen toteuttamiselle: koska asiakkaat jakavat saman sovellusinstanssin, ja joissain tapauksissa myös tietokannan, ongelmat yhden asiakkaan kohdalla heijastuvat myös muihin. Jaettu data asettaa myös lisävaatimuksia skaalautuvuuden, tietoturvan ja käyttökatkojen osalta. (Bezemer, Zaidman, Platzbeecker, Hurkmans & Hart, 2010.). Myös tietyn suorituskykytason takaaminen yksittäisille asiakkaille esimerkiksi palvelun laatuun (QoS) liittyvien takuiden osalta voi olla haastavaa, eli kuinka estetään yksittäisten asiakkaiden sellainen käyttö joka heikentää palvelun laatua muiden asiakkaiden osalta (Krebs, Momm & Kounev, 2012).

4.4 SaaS -kypsyysmallit

Chong & Carraro (2006) ovat esitelleet luokituksen SaaS -sovelluksen kypsyysasteen määrittelemiseksi. SaaS -sovellukset voidaan sen avulla jakaa

neljään eri tasoon arkkitehtuurin kypsyyden perusteella. Kang ym. (2010) ovat laajentaneet mallia lisäämällä siihen toiseksi ulottuvuudeksi palvelukomponentin. Mallin palvelukomponentteja ovat data, järjestelmä, palvelu ja liiketoiminta. Palvelukomponentit voidaan esittää myös tasoina joissa ne rakentuvat toistensa päälle, datan ollessa alimpana ja liiketoiminnan ylimpänä tasona. Kypsyysaste eri komponenttien välillä voi vaihdella vapaasti. (Kang, ym., 2010.). Taulukossa 7 on vertailtu Chongin ja Carraron (2006) sekä Kangin ym. (2010) kypsyysastemallien ominaisuuksia.

TAULUKKO 7 SaaS –kypsyysmallit

Taso	Chong, ym.	Kang, ym.	Konfiguroitava	Multitenantti	Skaalautuva
1	Ad Hoc	Ad Hoc			
2	Konfiguroitava	Standardointi	X		
3	Multitenantti	Integrointi	X	X	
4	Skaalautuva	Virtualisointi	X	X	X

Taulukossa 8 on esitelty Kangin, ym. (2010) kypsyysastemalli jossa on horisontaalisena ulottuvuutena palvelukomponentti.

TAULUKKO 8 SaaS –kypsyysmalli (Kang, ym., 2010)

Taso	Data	Järjestelmä	Palvelu	Liiketoiminta
Ad Hoc	Dedikoitu tietokanta ja skeema	Ad Hoc useita instansseja	Erillinen järjestelmäintegraatio	Ad Hoc / Erillinen sopimus
Standardointi	Jaettu tietokanta / Dedikoitu skeema	Konfiguroitava useita instansseja	Konfiguroitava pakettiohjelmisto	Standardisoitu SLA
Integrointi	Jaettu tietokanta ja skeema	Multitenantti	Web –pohjainen järjestelmä palveluliittymällä	Mitattava SLA
Virtualisointi	Tietokanta pilvessä	Virtuaalinen järjestelmä kuormantasauksella	Palvelu täydellä SOA:lla	Optimoitu SLA

Ensimmäisellä tasolla jokaisella asiakkaalla on oma sovellusinstanssinsa. Sovellukset on usein myös käännetty eri koodipohjasta, asiakkailla on versionhallinnassa oma haaransa, johon asiakaskohtaisia koodimuutoksia tehdään. Tämä taso vastaa perinteistä ASP -mallia. Sovellukset käyttävät omaa tietokantaansa ja tietomallia. (Chong & Carraro, 2006.).

Toisella tasolla jokaisella asiakkaalla on edelleen oma sovellusinstanssinsa, mutta parannuksena ensimmäiseen tasoon sovellukset on käännetty samasta koodipohjasta. Asiakaskohtaiset ominaisuudet ja teemoitukset hoidetaan ajonaikaisella konfiguraatiolla. (Chong & Carraro, 2006.). Kang ym. (2010)

kypsyysastemallissa sovellukset jakavat toisella tasolla yhteisen tietokantapalvelimen mutta tietomallit ovat edelleen erilliset.

Kolmannella tasolla palveluntarjoaja ajaa vain yhtä sovellusinstanssia, joka palvelee kaikkia asiakkaita. Asiakaskohtaiset teemoitukset ja ominaisuudet ovat konfiguroitavissa metadatan avulla. Verrattuna aiempiin tasoihin, taso kolme mahdollistaa palvelinlaitteiston tehokkaamman käytön ja vaikuttaa siten suoraan kustannuksiin niitä alentavasti. (Chong & Carraro, 2006.). Usean asiakkaan palveleminen yhdeltä palvelimelta voi kuitenkin aiheuttaa ongelmia, esimerkiksi kappaleessa 4.3 mainitut huomiot palvelun laadusta pätevät tällä tasolla (Krebs, Momm & Kounev, 2012).

Neljännellä tasolla arkkitehtuuri saavuttaa skaalautuvuuden. Kuormantasaajan takana toimii useita instansseja sovelluksesta, joita voidaan lisätä tai poistaa dynaamisesti, jotta järjestelmä mukautuu senhetkiseen kuormitukseen. Instanssit ovat täysin identtisiä ja niiden lisääminen tai poistaminen on asiakkaalle täysin näkymätön operaatio. (Kang ym. 2010.). Skaalautuvuus aiheuttaa paljon haasteita järjestelmän arkkitehtuurille ja tekniselle toteutukselle. Asiakkaiden datan erottaminen toisistaan voi joskus olla suositeltavaa, esimerkiksi tehokkuussyistä tai lainsäädännöllisten seikkojen vuoksi (Krebs, Momm & Kounev, 2012). Kypsyysasteen määritelmä ei suoraan ota kantaa siihen, miten tenanttien data on eroteltu multitenantissa ympäristössä. Yleisesti käytössä on kolme eri tapaa (Ju, Wang, Fu, Wu & Lin, 2010):

1. Oma tietokanta jokaiselle tenantille
2. Yhteinen tietokanta, erillinen tietokantaskeema
3. Yhteinen tietokanta ja tietokantaskeema

Tenanttikohtaisen tietokannan etuja ovat hyvä tietoturva, sillä tenanttien datat ovat täysin erillään toisistaan. Myös tietokannan varmuuskopiointi ja palauttaminen yhden tenantin osalta on tässä mallissa helppoa. Jaetun tietokannan ja erillisen tietokantaskeeman käyttämisen etuna on pienempi resurssien tarve. Varmuuskopiointi kuitenkin vaikeutuu, sillä jokaisella tenantilla on omat taulunsa. Jaettu skeema on resurssien käytön kannalta edullisin. Tietokannan tauluihin tuodaan uusi kenttä, jolla tunnistetaan mille tenantille data kuuluu. Tämä on myös varmuuskopioinnin ja palautuksen kannalta haastavin tapa, lisäksi se asettaa paljon suuremmat vaatimukset tietoturvan osalta, sovelluksen täytyy aina palauttaa oikeat rivit tietokannasta tenantille. Sovelluksen muokattavuuden näkökulmasta jaettu tietokanta ja skeema on haastavin, sillä asiakaskohtaiset lisäkentät täytyy tuoda kaikkien muidenkin asiakkaiden dataan mukaan. (Chong, Carraro & Wolter, 2006.). Yhteisen tietokannan etuina on sovelluspäivitysten helppous ja tiedonkeräämisen helpottuminen, esimerkiksi käyttäjien toiminnan seuraaminen helpottuu tiedon sijaitessa yhdessä paikassa (Bezemer & Zaidman, 2010).

Single tenantissa ympäristössäkin voidaan tunnistaa useita eri tapoja asiakaskohtaisten ympäristöjen toteuttamiseen. Toisessa ääripäässä on oman

palvelinraudan tarjoaminen jokaiselle asiakkaalle. Nykyään todennäköisempää on kuitenkin virtualisointityökalujen käyttö. Samalla palvelinraudalla voi toimia useita virtuaalikoneita, jotka palvelevat eri asiakkaita. Myös sovelluspalvelintasolla yksi sovelluspalvelin voi palvella useita asiakkaita esimerkiksi sovelluspoolien kautta. (Chong, 2006.). Arkkitehtuurin valinnassa ei siis ole pelkästään kysymys siitä valitaanko single- vai multi –tenancy, vaan väliin mahtuu monia vaihtoehtoisia toteutustapoja niin sovelluksen kuin tietokannan tasolla. Virtuaalikoneiden tehokkuus on parantanut ohjelmistojen ja rautatuen kehittyessä ja konttiteknologialla tehokkuusero natiivisovelluksiin verrattuna on jo lähtökohtaisesti pieni (Felter, Ferreira, Rajamony & Rubio (2015). Pilvipalvelujen ja virtuaalisten ympäristöjen hallintaan on kehittynyt myös valikoima työkaluja, joilla ympäristöjen asennuksia ja sovellusten hallintaa voidaan yksinkertaistaa tai jopa automatisoida (Benson, Prevost & Rad, 2016).

4.5 SaaS -sovelluksen konfigurointi ja kustomointi

Single -tenant ympäristössä jokaisella asiakkaalla on oma dedikoitu instanssinssa, joka on konfiguroitu asiakkaan tarpeita varten. Tässä ympäristössä yhdelle asiakkaalle tehtävät muutokset voidaan toteuttaa asiakaskohtaiseen haaraan versionhallinnassa ja asentaa vain yhdelle instanssille. Multitenanttiympäristössä tämä ei ole mahdollista vaan asiakaskohtaiset eroavaisuudet pitää pystyä toteuttamaan konfiguraation avulla, sillä kaikkia asiakkaita palvellaan samasta yhdellä sovellusinstanssilla (Bezemer & Zaidman, 2010).

Asiakaskohtaisia muutoksia voidaan toteuttaa joko konfiguroinnilla tai kustomoinnilla. Konfiguroinnilla tarkoitetaan sovelluksen toiminnan tai ulkoasun muokkaamista ilman koodimuutoksia muokkaamalla ennaltamääriteltyjä parametreja. Kustomoinnilla tarkoitetaan asiakaskohtaisten koodimuutosten tekemistä (Sun, Zhang, Guo, Sun & Su, 2008). Asiakkaan näkökulmasta sovelluksen täytyy toimia samalla tavalla kuin ajettaessa asiakkaalle dedikoitua instanssia. Tämä tarkoittaa esimerkiksi yrityskohtaisten logojen ja teemoituksen käyttöä ja jotta siihen päästäisiin, Bezemerin ym. (2010) mukaan sovelluksen täytyy olla konfiguroitavissa vähintään neljällä eri tavalla:

1. Ulkoasu, kuten asiakaskohtainen teemoitus
2. Yleinen konfiguraatio, kuten salausavaimet ja profiilit
3. Tiedosto I/O, esimerkiksi asiakaskohtaiset tiedostopolut raporttien generointiin.
4. Työnkulku

Yksi tapa palvella useita asiakkaita samalla sovellusinstanssilla säilyttäen yksittäisten palvelujen maksimaalinen kustomoitavuus on palveluorientoituneen arkkitehtuurin piiristä. Mietzner, Leymann ja Unger

(2011) ovat esitelleet suunnittelumalleja joiden avulla palveluja ja komponentteja voidaan yhdistellä tavalla, joka sallii sekä asiakkaille yhteisten osien uudelleenkäytön, että osien asiakaskohtaisen kustomoinnin.

Jansen, Houben ja Brinkkemper (2010) ovat tutkineet kustomointitapoja multitenanteissa web -sovelluksissa ja tunnistaneet viisi eri kustomointimenetelmää (Customization Realization Technique, CRT) jotka voidaan luokitella kahteen eri tyyppiin: MVC- ja järjestelmäkustomointiin. MVC -kustomoinnin osat viittaavat suosittuun Model-View-Controller (Malli-Näkymä-Käsittelijä) arkkitehtuuriin. Kustomointi kohdistuu siinä joko tietomalliin (model), näkymään (view) tai käsittelijään (controller). Järjestelmäkustomointi jakautuu järjestelmäliitännän vaihtoon (system connector change) ja järjestelmäkomponentin vaihtoon (system component change). Järjestelmäliitännän vaihdolla tarkoitetaan kustomointia, jossa jokin kytkentä ulkoiseen sovellukseen voidaan vaihtaa. Järjestelmäkomponentin vaihdolla tarkoitetaan kustomointia, jossa kokonainen komponentti pystytään asiakaskohtaisesti vaihtamaan toiseen. (Jansen, Houben & Brinkkemper, 2010.).

Ohjelmistotuotelinjoja koskevan tutkimuksen puolella esiteltiin luvussa kolme käsite muunneltavuus. Svahnberg, Van Gurp ja Bosch (2005) ovat muodostaneet taksonomian muunneltavuuden toteuttamiseen käytetyistä tekniikoista ja listanneet myös sidonta-ajat eri tekniikoille:

1. Arkkitehtuurin muodostaminen
2. Sovelluksen kääntäminen
3. Linkkaus
4. Ajonaikainen

Multitenantissa ympäristössä asiakaskohtaiseen muunneltavuuteen näistä voidaan käyttää ainoastaan ajonaikaista sidontaa (Kabbedijk & Jansen, 2012). Single tenant -ympäristössä useita erillisiä sovellusversioita ajettaessa myös muut vaihtoehdot voivat tulla kyseeseen, kun toimitaan SaaS – maturiteettimallin alimmalla tasolla. Muunneltavuuden soveltuvuutta SaaS, ja erityisesti multitenanttiin ympäristöön tutkineet Kabbedijk ja Jansen (2012) ja määritelleet käsitteen tenantti- pohjainen muunneltavuus (tenant-based variability). Vastaavan käsitteen asia- kaslähtöinen muunneltavuus (customer-driven variability) ovat esittäneet Mietzner, Metzger, Leymann ja Pohl (2009), jotka tunnistivat kaksi lähdettä muunneltavuudelle: asiakaslähtöisen (customer-driven variability) ja toteutuslähtöisen (realization-driven variability) muunneltavuuden. Asiakaslähtöinen muunneltavuus tarkoittaa eri asiakkaiden tarpeista lähtöisin olevaa muunneltavuutta, esimerkiksi asiakaskohtaisesti konfiguroitavaa teemoitusta. Toteutuslähtöinen muunneltavuus taas lähtee ohjelmiston toteutuksessa tehtävistä valinnoista ja näkyy lähinnä kehittäjille (Mietzner ym., 2009).

5 SAAS –PALVELUN VERSIOINTI

Tässä luvussa käsitellään versiointia ja hinnoittelua SaaS –palvelujen konteksissa. Luvussa kerätään yhteen edellisten lukujen teoriat ja käsitellään hinnoittelun ja versioinnin ongelmia ja niiden aiheuttamia vaatimuksia SaaS –arkkitehtuurin suhteen.

5.1 SaaS –sovelluksen versiointi

Ohjelmistojen myynti poikkeaa monilla tavoin muiden tuotteiden myynnistä ja on selvää, että ohjelmistopalvelujen myynti poikkeaa perinteisestä ohjelmistomyynnistä (Cusumano, 2003). Moderni SaaS -arkkitehtuuri mahdollistaa laitteiston tehokkaan hyödyntämisen ja palvelun tarjoamisen suurelle asiakasjoukolle pienillä kustannuksilla. Pitkälle standardisoidun palvelun tarjoaminen suurelle asiakasjoukolle onkin ollut suositeltu tapa menestyä SaaS –markkinoilla. Monesti tämä on tarkoittanut sitä, että palvelu on suunnattu yrityksen tukitoimintoihin. Tietoturva-, tehokkuus- ja jatkuvuushuolet ovat edelleen yksi syy minkä takia monet yritykset eivät uskalla ulkoistaa kriittisempiä toimintojaan pilveen. Liikuttaessa lähemmäksi yritysten liiketoiminnan ydintä saattavat palvelun laatuun, tietoturvaan ja asiakaskohtaisiin räätälöinteihin liittyvät vaatimukset kasvavat. (Benlian, Hess & Buxmann, 2009.).

Joustavuus ohjelmistojen hinnoittelussa vaatii joustavuutta myös ohjelmistoarkkitehtuurilta. Modulaarinen arkkitehtuuri mahdollistaa myös erilaisten hinnoittelumallien käytön (Laatikainen & Ojala, 2014). Ohjelmiston on skaalauduttava erikokoisten ja erilaisia tarpeita omaavien asiakkaiden käyttöön ja joissain tapauksissa ohjelmiston helppo muokattavuus voi olla merkittävä menestystekijä (Sun, Zhang, Guo, Sun & Su 2008).

Ohjelmiston versiointia voidaan käyttää monin eri tavoin tuotteen myynnin ja markkinoinnin työkaluna. Versiointia voitaisiin soveltaa esimerkiksi dynaamisten hinnoittelustrategioiden kanssa. Laadullisesti toisistaan poikkeavia versioita voidaan käyttää sellaisten asiakasryhmien

houkuttelemiseksi jotka eivät ole valmiita maksamaan täyttää hintaa tuotteesta. Ominaisuuksiltaan toisista poikkeavilla versioilla taas voidaan täyttää erilaisten asiakasryhmien erilaiset tarpeet. Asiakaskohtaisten räätälöintien voidaan katsoa kuuluvan tämän jälkimmäisen versioinnin piiriin.

SaaS -arkkitehtuurin maturiteettimallin korkeimmat tasot tähtäävät helposti skaalautuvaan sovellukseen joka palvelee useita asiakkaita tehokkaasti. (Chong & Carraro, 2006). Nämä tavoitteet ovat ristiriidassa asiakaskohtaisen kustomoitavuuden kanssa ja ovat arkkitehtuurin kannalta haastavia ratkaistavia ongelmia. (Mietzner, Leymann & Unger, 2011). Erilaisia ratkaisuja ohjelmistoversioiden helppoon luomiseen ja koodin tehokkaaseen uudelleenkäyttöön on esitetty muun muassa ohjelmistotuotelinjoja koskevan tutkimuksen parissa. Esimerkiksi konfiguroitavia ohjelmistotuotelinjoja voisi mahdollisesti käyttää myös single-tenant ympäristössä asiakaskohtaisten versioiden tehokkaaseen luomiseen (Niemelä, 2005). SaaS –palvelua toteutettaessa mukaan tulee omat eritysongelmansa muun muassa multitenantin ympäristön ja palvelinten te- hokkaan käytön vaatimusten mukana.

Perinteisistä ohjelmistotuotteista poiketen palvelun pyörittämisestä ja käytöstä koituu kustannuksia palveluntarjoajalle. Palveluntarjoajan etu on palvelinlaitteiston mahdollisimman tehokas käyttö. Tämä tarkoittaa ideaalitilanteessa vain yhden ohjelmistoversion ja tietokannan käyttämistä kaikkien asiakkaiden palvelemiseen. Asiakaskohtaiset kustomoinnit sekä ohjelmistoversiot ja asiakkaalle luvatun palvelutason ylläpitäminen voivat olla hyviä syitä poiketa korkean SaaS -maturiteetin vaatimuksista.

5.2 SaaS –arkkitehtuurin huomiointi tuotteen suunnittelussa

Arkkitehtuurin valinta kytkeytyy kiinteästi yrityksen kilpailustrategiaan. Mannionin ja Savolaisen (2010) kustannusjohtajuusstrategiassa tuotekehityksen painopiste on yleisten komponenttien uudelleenkäytössä. Tuotteen tarjoaminen suurelle asiakasjoukolle kustannustehokkaasti vaatii alusta asti multitenanttia ympäristöä. Multitenantti ympäristö ja pitkälle standardisoitu sovellus voi tarjoa monenlaisia etuja kuten tuotteen käyttöönoton itsepalveluna, jolloin asiakas voi helposti kokeilla tuotetta ennen ostopäätöstä. Freemium –mallin käyttöönotto ja siihen pohjautuvien hinnoittelustrategioiden tehokas hyödyntäminen myös helpottuu, jos ohjelmistoa voidaan tarjota suurelle asiakasjoukolle pienillä kustannuksilla. Toisaalta multitenantin ympäristön tukeminen nostaa ohjelmiston ominaisuuksien kehitys- ja ylläpitokustannuksia, joten asiakkaiden vaatiman kustomointiasteen ja saavutettavien skaalaetujen välillä voi joutua tekemään kompromisseja. (Katzan & Dowling, 2010.). Erilaiset differointistrategiat taas voivat keskittyä joko luomaan useita variaatioita tuotteista tai keskittymään pienemmän asiakasryhmän vaatimusten täyttämiseen (Mannion & Savolainen, 2010).

Tietyt kustomointipisteet tulisi ottaa huomioon, vaikka tehtäisiinkin pitkälle standardisoitua palvelua. Minimissään asiakaskohtaisten logojen ja teemoituksen on oltava kustomoitavissa. Palvelun kytkeytyessä ulkoisiin järjestelmiin, esimerkiksi kirjanpito- tai laskutusaineiston toimittamiseksi, myös ulkoisten integraatioiden on oltava helposti kustomoitavissa tai konfiguroitavissa. (Bezemer ym. 2010.).

Modulaarisuudella voidaan saavuttaa helposti kustomoitava tuote, jossa ominaisuudet voidaan valita asiakaskohtaisesti ja palvelu räätälöidä asiakkaan tarpeita vastaavaksi. Ohjelmistotuotelinjoja koskevassa tutkimuksessa on perehdytty koodin uudelleenkäyttöön ja tuoteperheiden rakentamiseen samasta koodipohjasta. Multitenantissa ympäristössä on kuitenkin yhtä aikaa vain yksi sovellus käynnissä, eli varioitavien ominaisuuksien sidonnan on tapahduttava ajonaikaisesti (Kabbedijk & Jansen, 2012). Tämä rajoittaa jossain määrin ohjelmistotuoteperheiden parissa syntyneiden käytäntöjen sovellettavuutta SaaS -arkkitehtuurissa. Ominaisuusmallit ja käyttökontekstin analysointi ominaisuuksien valinnassa voivat olla hyödyllisiä työkaluja myös SaaS –sovellusten maailmassa. Ohjelmistotuotelinjojen muunneltavuutta koskevia menetelmiä on jo ehditty sovelta myös SaaS –ympäristöön (Kabbedijk & Jansen, 2012). Menetelmiä soveltuvan arkkitehtuurin luomiseksi on myös kehitetty, skenaariopohjaisissa malleissa otetaan huomioon sekä asiakkaan vaatimukset, että tekniset vaatimukset ja käytetään niistä muodostettuja skenaarioita apuna arkkitehtuurien arvioinnissa ja lopulta valinnassa (Rommes & America, 2006).

5.3 SaaS –palvelu osana hinnoittelustrategiaa

Miten liiketoimintamalli, arkkitehtuuri ja versiointi sitten vaikuttavat toisiinsa? Liiketoimintamalleissa (Luoma, ym. 2012), samoin kuin arkkitehtuurissa toisessa ääripäässä on kustannustehokas standardoitu sovellus, jonka asiakkailla ei ole toisistaan poikkeavia tarpeita. Toisessa ääripäässä on täysin eriytetty, yksittäistä asiakasta ja hänen tarpeitaan palveleva sovellus. Taulukossa 9 on esitetty liiketoimintamallit Luoma ym. (2012) jaottelun mukaisesti ja lisätty niihin parhaiten soveltuva arkkitehtuurimalli ja suositeltu tuotedifferoinnin malli.

TAULUKKO 9 SaaS –palvelun arkkitehtuuri ja liiketoimintamalli

Liiketoimintamalli	**Arkkitehtuuri**	**Kilpailuetu**	**Differointi**
Self-service SaaS	Multitenantti	Kustannusjohtajuus	Vertikaalinen
Pure-play SaaS	Multi/Single	Kustannusjohtajuus / asiakkaan tuntemus	Horisontaalinen / Vertikaalinen
Enterprise SaaS	Multi/Single	Asiakkaan tuntemus	Horisontaalinen

Self-service SaaS on ensiksi mainittu ääripää, ja multitenantti arkkitehtuuri on kustannustehokkuuteen ja monen asiakkaan palvelemiseen suunnatun luonteensa takia itsestään selvä vaihtoehto. Kilpailuetuna on tällöin kustannusjohtajuus ja vertikaalinen differointi, eli erihintaiset versiot soveltuvat parhaiten, jos versiointia yleensä halutaan tehdä. Toisessa ääripäässä Enterprise SaaS keskittyy suurten asiakkaiden palvelemiseen, asiakkaiden liiketoiminnan tuntemus ja asiakaskohtaiset räätälöidyt sopimukset ovat tärkeitä, jolloin myös yhden asiakkaan ympäristö voi tulla kyseeseen arkkitehtuurin näkökulmasta. Kilpailuetuna on asiakkaan tuntemus ja horisontaalinen differointi, palvelun ominaisuuksien räätälöinti asiakaskohtaisesti voi olla tarpeen. Ääripäiden välissä Pure-play SaaS liiketoimintamalli voi arkkitehtuurin ja muidenkin kohtien osalta kallistua kumpaan suuntaan tahansa. (Luoma, ym. 2012.).

6 TUTKIMUSMETODOLOGIA

Tässä luvussa esitellään tutkimuksen aihepiiriä koskeva aiempi tutkimus ja rajataan tutkimusongelma. Lisäksi esitellään käytetty tutkimusmenetelmä ja perustelut sen valinnalle. Lopuksi käydään läpi tapaustutkimukseen valitut yritykset ja valintaperusteet.

6.1 Tutkimusmenetelmä

Tutkimusmenetelmänä on laadullinen monitapaustutkimus. Empiirisen osion tiedonkeruumenetelmänä käytetään puolistrukturoituja haastatteluja. Tutkimustapausten rajaus B2B -yrityksiin on tehty B2C- ja B2B -markkinoiden erilaisen rakenteen vuoksi. Versiointi ja asiakaskohtainen kustomointi näyttäytyvät erilaisessa roolissa yritysten välisessä kaupassa. Versioinnin käyttö SaaS -palveluiden tuottamisessa ja hinnoittelussa on vähän tutkittu ja myös käytännön kannalta uusi ilmiö. Tapauksen määritelmä tämän tutkimuksen piirissä on yksittäinen sovellus, josta voi olla useita eri versioita. Monessa kohti kuitenkin puhutaan tapausyrityksistä, sillä kaikki sovellukset ovat eri yrityksiltä ja tapauksia on yksi per yritys.

Tutkittavat yritykset on rajattu suomalaisiin yritykseltä yritykselle (B2B) SaaS -palveluja tarjoaviin yrityksiin. Tarkkaa rajausta siihen mikä lasketaan SaaS –palveluksi ei valinnassa tehdä. Käytännössä kaikki yritykset, jotka tarjoavat itse tekemäänsä sovellusta useille asiakkaille käytettäväksi verkon yli itse hallinnoimiltaan palvelimilta, luetaan mukaan. Toinen ehto on että kaikki tutkittavat yritykset tarjoavat palveluaan joko erihintaisina versioina, valinnaisina moduuleina tai täysin asiakaskohtaisesti räätälöitävänä kokonaisuutena. Toimialoja tai sovelluksen tyyppiä ei ole rajattu. Pikemminkin olisi suotavaa löytää yrityksiä mahdollisimman monelta eri toimialalta ja tuotteita jotka kytkeytyvät asiakkaan eri prosesseihin, jotta voidaan nähdä vaikuttaako läheisempi suhde asiakkaan liiketoimintaan esimerkiksi kustomoinnin asteeseen ja sitä kautta hinnoittelumalleihin.

Aiemman tutkimuksen perusteella voidaan olettaa, että esimerkiksi suurelle asiakasjoukolle, pitkälle standardisoituja ohjelmistoja tarjoavat yritykset pyrkisivät korkeammalla SaaS -arkkitehtuurin maturiteettitasolle kuin paljon asiakaskohtaista kustomointia tekevät palveluntarjoajat.

Versioinnin, arkkitehtuurin ja hinnoittelun välillä on paljon riippuvuuksia, joita on tutkittu vähän. Yrityksissä tämä kokonaisuus kattaa henkilöstöä sekä myynnin että ohjelmistoarkkitehtuurin saralta. Kompromissit hinnoittelun, tuotteen kohderyhmien ja sovelluksen arkkitehtuurin osalta ovat arkipäivää.

Tapaustutkimus ja erityisesti ohjelmiston kehityksessä eri rooleissa toimivien henkilöiden haastattelut, sekä sitä kautta saadut erilaiset näkemykset ohjelmiston kehityksen aikana tehdyistä ratkaisuista, voivat valottaa tutkimuksen ongelmaa siitä, miten tuoteversiointi realisoituu moderneissa SaaS -palveluissa. Tapaustutkimuksen on nähty soveltuvan erityisen hyvin tilanteisiin joissa tutkittavan alueen käsitteet eivät vielä ole selkeitä, sekä silloin kun toimijoiden kokemukset ja toiminnan konteksti ovat merkityksellisiä (Darke, Shanks & Broadbent, 1998), sekä tutkimuksiin joissa halutaan tietää "miten", tai "miksi" jokin ilmiö esiintyy (Yin, 2013). Tapaustutkimus voi kohdistua yhteen tai useaan tapaukseen. Yhden tapauksen tapaustutkimus voi soveltua joko täysin ainutlaatuisen ilmiön tai olemassa olevan teorian kriittiseen testaukseen (Jin, 2013). Monitapaustutkimuksessa tutkitaan useita eri kohteita, mutta kohteiden valinta ei ole satunnainen vaan sillä voidaan esimerkiksi replikoida aiempi tutkimus tai valita toisistaan poikkeavia tapauksia jotka ovat rakennettavan tai testattavan teorian kannalta mielenkiintoisia (Eisenhardt, 1989).

Haastattelu on aina keinotekoinen tilanne ja monet seikat kuten ajan tai luottamuksen puute tai haastattelijan omat kokemukset tai tulkinnat voivat vaikuttaa haastattelun tulokseen (Myers, & Newman, 2007). Tämän gradun aihe liittyy hyvin läheisesti nykyiseen työhöni, ja haasteisiin joita olen aihepiirin ympärillä kohdannut, joten erityistä huomiota täytyy asettaa siihen, etteivät omat mielipiteeni aiheesta ohjaa tulkintoja haluttuun suuntaan. Erilaisten tiedokeruumenetelmien yhdistäminen tai esimerkiksi useiden tutkijoiden tai haastattelijoiden käyttäminen parantaa tulosten luotettavuutta (Eisenhardt, 1989). Jälkimmäinen tapa oli tämän gradun puitteissa poissuljettu, mutta muita tietolähteitä hyödynnettiin rajatusti. Tietoja kerättiin muun muassa yritysten uutiskirjeistä ja muusta markkinointimateriaalista. Näiden aineistojen hyödyntäminen jäi kuitenkin pääasiassa haastattelujen valmisteluvaiheeseen.

Aineiston keruun ja analyysin olisi hyvä tapahtua lomittain. Analyysi voidaan aloittaa jo ensimmäisten haastattelujen valmistuttua ja löydösten perusteella voidaan vielä tehdä muutoksia tiedonkeruuseen. Tiedonkeruumenetelmien muuttaminen kesken tutkimuksen on hyväksyttävää, kun tarkoituksena on jokaisen tutkittavan ilmiön mahdollisimman tarkka ymmärtäminen, eikä täysin vertailukelpoinen lista kysymyksiä ja vastauksia tapausten välillä. (Eisenhardt, 1989.).

6.2 Tapausten valinta

Soveltuvien tapausten valinta osoittautui haastavaksi. Sopivia tuotteita etsin googlesta hakusanoja "SaaS", "cloud", "pilvipalvelu" ja niiden yhdistelmiä käyttäen. Myöhemmin lisäsin mukaan myös toimialakohtaisia termejä kuten "toiminnanohjaus" tai "tuntiseuranta". Hain tietoa yritysten nettisivuilta ja uutiskirjeistä ja kirjastin tuotteista ylös muun muassa tarjottavien versioiden lukumäärän, maininnat asiakaskohtaisista räätälöinneistä ja pilvipalvelun luonteesta sekä valinnaisista moduleista. Versioinnin granulaarisuus ja asiakasokohtaisen räätälöinnin määrä olivat tärkeimmät tekijät, joilla tutkittaviin tuotteisiin haluttiin tuoda eroja. Alkukartoituksen jälkeen löytyi muutamia kymmeniä yrityksiä, joiden tuote vaikutti sopivalta. Vähimmäisvaatimuksena oli, että tuotteesta on tarjolla pilvipalvelu, josta on erilaisia versioita ja/tai valinnaisia moduleja. Näistä noin kymmenenkunta yritystä vaikutti sopivan tutkimuspiirin rajaukseen, erityisesti versioinnin osalta. Lähestyin näitä kymmentä lupaavinta yritystä ensin mailitse, ja soitin perään parin päivän kuluttua. Lopulta tapaustutkimukseen valittiin neljä eri yritystä. Yhteisenä tekijänä kaikilla yrityksistä oli SaaS -palveluna tarjottu tuote, josta oli tarjolla erihintaisia versioita tai räätälöintiä. Kaikki yritykset toimivat etelä-suomessa mutta niiden koot vaihtelivat pienestä suureen.

Tapauksien välille yritettiin saada kontrastia valitsemalla yrityksiä jotka tarjoavat palvelua hieman erilaisille toimialoille, lisäksi yritysten tarjontaa tutkimalla yritettiin myös löytää yrityksiä jotka tekevät paljon räätälöintiä ja yrityksiä jotka myyvät mahdollisimman pitkälle standardisoitua pakettia. Lisäksi useille eri toimialoille tuotettaan tarjoavien yrityksien vastapainoksi otantaan valittiin yksi yritys, joka on keskittynyt yhden toimialan asiakkaisiin. Tapaustutkimuksen yritykset on esitelty taulussa 10.

Kaksi yrityksistä tarjoaa toiminnanohjausjärjestelmiä (ERP, eli Enterprise Resource Planning) eri toimialoille ja kaksi tarjoaa asiantuntijayrityksen toiminnanohjausta (PSA, eli Professional Services Automation) asiantuntijayrityksille. ERP –ohjelmisto sisältää monesti varastonhallintaa, tuotannon järjestelmiä, laskutusta ja kirjanpitoa. PSA –ohjelmistoon taas voi kuulua esimerkiksi projektinhallintaa, tuntikirjanpitoa, laskutusta ja raportointia. Kaikista sovelluksista oli tarjolla vähintään kaksi erihintaista ja/tai tasoista versiota.

TAULUKKO 10 Tapaustutkimuksen yritykset

Yritys	**Tarjonta**	**Toimiala**	**Yrityksen koko**
A	ERP	Teollisuus, maahantuonti, ym…	Keskisuuri
B	PSA	Asiantuntijayritykset	Suuri
C	PSA	Asiantuntijayritykset	Pieni
D	ERP	Korjaamoala	Pieni

Tapauksiksi valikoitui toiminnanohjausjärjestelmiä lähinnä siitä syystä, että monet muut yrityksille suunnatut ohjelmistot, kuten työajanseurannat, pikaviestimet, tai vaikka yksinkertaisemmat tiketointiohjelmistot sopivat yleensä sellaisenaan kaikille käyttäjille ja siten tilaa versioinnille ja differoinnille ei juurikaan ollut. Toiminnanohjausjärjestelmissä asiakas- ja toimialakohtaista versiointia käytetään paljon, sillä ohjelmistot pitää saada sopeutumaan asiakkaan omiin prosesseihin.

6.3 Tiedonkeruu ja analysointi

Jokaisesta yrityksestä haastateltiin yhtä henkilöä, joka toimi johto- tai asiantuntijatason tehtävissä ja jolla oli näkemystä tuotekehityksestä. Kaikki haastatellut olivat miehiä, heillä oli ylempi korkeakoulututkinto ja he olivat työskennelleet vähintään kaksi vuotta nykyisessä työpaikassaan. Haastattelujen kesto oli 25 – 60 minuuttia, keskimäärin haastattelu kesti 43 minuuttia. Haastattelua varten käytettiin kysymyspohjaa (Liite 1), jota täydennettiin yrityskohtaisesti yrityksen nettisivuilta ja markkinointimateriaalista nousseiden lisäkysymysten avulla. Kaikki haastattelut nauhoitettiin. Ensimmäinen haastattelu (Tapaus A) oli puhelinhaastattelu, joka nauhoitettiin käyttämällä älypuhelimeen asennettua sovellusta. Muut haastattelut tehtiin yrityksissä paikan päällä. Nämä haastattelut nauhoitettiin käyttäen Olympuksen sanelukonetta. Kaikki äänitteet purettiin tekstimuotoon pian haastattelun jälkeen.

Haastattelut tapahtuivat noin kuukauden ajanjakson aikana. Tapausten analysointi aloitettiin lomittain tiedonkeruun aikana. Tapaukset analysointiin ensin yksittäin, jokaisen tapauksen kysymys-vastausparit luokiteltiin teemojen mukaan, esimerkiksi arkkitehtuuri, hinnoittelu, versiointi tai räätälöinti. Tapauksista kerättiin joitain avainmuuttujia, kuten arkkitehtuurin tyyppi ja versioiden lukumäärä. Osa näistä muuttujista on taulukoitu luvun 7 vertailuosiossa.

Tapausten ristiinanalysoinnissa tapauksia verrattiin joko samankaltaisuuksien tai eroavaisuuksien kannalta. Vertailu tapahtui monesti teemojen sisällä, esimerkiksi minkälaisia hinnoitteluun liittyviä yhteneväisyyksiä tai eroavaisuuksia tapauksista löytyi. Yritysten välillä vertailupareja voitiin tehdä joko yrityksen koon, räätälöinnin määrän tai toimialojen perusteella.

7 TULOKSET

Tässä luvussa käydään läpi tutkimuksen tulokset. Jokainen haastattelu avataan omassa kappaleessaan ja pääteemoihin liittyvät kysymykset käydään läpi. Lopuksi tapauksia vertaillaan toisiinsa.

7.1 Tapaus A – Kustomoitava usean toimialan ERP

Yritys A tarjoaa ERP –ratkaisua muun muassa tuotannonohjaukseen, varastonhallintaan ja taloushallintoon. Asiakkaita on monilta toimialoilta ja pienistä suuriin yrityksiin.

Sovelluksesta on ajossa useita eri instansseja, joista osa toimii asiakkaiden omalla laitteistolla. Lisenssipohjainen hinnoittelu on yleistä mutta myös käyttäjäkohtaista hinnoittelua ja näiden yhdistelmää voidaan käyttää:

> meillähän ei ole itsellään mitään suosikkia, että onko se näin vai noin vaan se on asiakkaan omasta mielihalusta kiinni, että onko lisenssiä vaikka kuukausihintaista.

Moni asiakas kuitenkin on arvioinut lisenssipohjaisen hinnoittelun halvemmaksi vaihtoehdoksi. Tässä asiakkaan koko ja toimiala saattavat olla selittäviä tekijöitä, kun toiminnanohjaus menee tarpeeksi lähelle asiakkaan liiketoiminnan ydintä niin käytössä tuskin tapahtuu sellaista vaihtelua että sovellus jäisi vaille käyttöä ja olisi siten turha menoerä:

> Moni asiakas on esimerkiksi hankkinut sitten meidän pilvipalvelimelle oman lisenssin, kun on laskenut, että tulee edullisemmaksi näin.

Hybridimalleilla saadaan joustavuutta hinnoitteluun, esimerkiksi käyttäjien lukumäärän vaihtuessa tai ollessa epäselvä:

> myydään hyridimalleja, että voidaan myydä, esimerkiksi vaikka perus-ERP lisenssinä mutta sitten jos ei tiedetä tarkalleen tai asiakas ei tiedä vaikka paljonko on tietynlaisia käyttäjiä vaikka ostolaskun hyväksyjiä tai onko alihankkijat siinä samassa

> järjestelmässä kiinni tai asentajat tuolla niin niille voidaan taas määritellä käyttäjäkohtainen kuukausihinta.

Yrityksen tarjoama toiminnanohjausjärjestelmä on mukautettavissa asiakkaan tarpeisiin valinnaisten moduulien avulla. Lisäksi tarjolla on eri toimialojen tarpeisiin suunnattuja toimialakohtaisia versioita. Toimialakokonaisuudet ovat syntyneet yksittäisten asiakkaiden tarpeista, joista on sitten johdettu muillekin soveltuvat tuote:

> kyllä siellä aina joku asiakastarve on mistä on sitten lähdetty liikkeelle ja kun jollekin on saatu toimimaan, niin kyllä siellä on sana kiirinyt ja tiettyä tämmöisiä voi sanoa, että toimialakokonaisuuksia on syntynyt.

Toiminnallisuuksia toteutetaan myös yksittäisten asiakkaiden tarpeita ajatellen. Yksittäisille asiakkaille tehdyt muutokset pyritään kuitenkin aina tuomaan osaksi yleistä versiota:

> meillähän on pääsääntöisesti se periaate, että jos on hyvä idea, niin tota se tuodaan sitten ihan siihen meillä yleiseen versioon hyödyntämään sitten muitakin että se on nähty että mitä laajemmin järjestelmää käytetään niitä piirteitä niin sitä enemmän se sitten luo hyötyä koko sille kyseiselle toimialalle

Räätälöinti ja toimialakohtaiset versiot näyttävät olevan yritys A:lla suuremmassa roolissa kuin muilla tapausyrityksillä. Yksi syy tähän voi olla muita laajempi toimialojen kirjo ja ehkä myös läheisempi suhde asiakkaiden ydinliiketoimintaan. Myös erillisten palvelinten käyttö ja asiakaskohtaiset asennukset saattavat johtua toimialojen erityistarpeista.

Muista tapausyrityksistä poiketen yritys A tarjoaa myös kattavia taloushallinnon ja IT -ylläpidon palveluita ohjelmistonsa rinnalla:

> kaikki aina on asiakkaan todellisen tarpeen mukaan jollain asiakkaalla vaan avustetaan vaikkapa tilinpäätöksen teossa muuten asiakas on aika omatoiminen mutta joillain se voi tarkoittaa sitä, että koko taloudenhallinta aika lailla hallintopuoli on ulkoistettu taikka meillä asiakas vaan itse tekee järjestelmään toimituskirjaukset ja sitten me hoidetaan kaikki siitä eteenpäin.

Yritys A erottuikin muista tapauksista erityisesti tämän läheisemmän suhteen kautta. Tarjonnassa oli eniten räätälöintiä ja tietylle toimialalle kohdennettuja ratkaisuja, sekä muista tapauksista poiketen kattava palvelutarjonta myös sovelluksen perustuen ulkopuolelta.

7.2 Tapaus B – Multitenantti yleiskäyttöinen PSA

Yritys B tarjoaa asiantuntijayrityksille suunnattua toiminnanohjausta, eli niinsanottua PSA –tuotetta. Tuote tarjotaan puhtaasti pilvipalveluna ja siitä on saatavilla kolme eritasoista versiota. Versioiden ominaisuudet kumuloituvat, eli

kalliimpaan versioon sisältyvät aina kaikki halvempien versioiden ominaisuudet. Tarjolla on lisäksi joitain valinnaisia palveluita ja moduuleita joita voi ottaa käyttöön kaikkien versioiden päälle. Varsinaisia eri toimialoille suunnattuja moduuleja tai versioita ei ole tarjolla vaan toiminnallisuuksista on pyritty tekemään mahdollisimman yleiskäyttöiset tai asiakkaan tarpeisiin mukautuvat:

> Mutta toki se tarpeen tota, mistä se on saanut alkunsa ja miltä asiakkaalta se on tullut se toive ja mitä ominaisuuksia se sisältää, niin esimerkiksi, vaikka tarjoustyökalu mikä meillä on, niin kyllä se toimii kaikille toimialoille ja se on suunniteltu niin että se toimii, mutta lähtökohta on ollut alun perin, vaikka mainostoimisto. Ja sit se niinku tarve on lähtenyt sitä kautta syntymään ja sitten me on kartoitettu, että mites nää muut toimialat ja miten tää pitäis mennä ja tehään sitten semmoinen yleinen, mikä on vaan mahdollisimman hyvin määriteltävissä ja konffattavissa ite et sä voit tehdä siitä erilaisia.

Hinnoittelu painottuu halvemmissa versioissa käyttäjäkohtaiseen maksuun, kalliimmissa versioissa yrityskohtaisen hinnoittelun osuus kasvaa ja käyttäjäkohtaisten maksujen osuus laskee. Tarkoituksena on ollut luoda hinnoittelumalli, jolla pystytään houkuttelemaan pieniä asiakkaita ja samalla helpottamaan kustannusten ennustettavuutta suurille asiakkaille:

> sen hinnoittelumallin idea on, että se kasvais sen asiakkaan mukana, ja se hinnoittelumalli eläis siinä.

Asiakaskohtaisia versioita ei tehdä lainkaan. Sovellus toimii multi-tenantissa ympäristössä. Yksi huomio oli, että laajempi asiakaskohtainen kehitys oli hyödyllistä tuotekehityksen varhaisessa vaiheessa:

> Tällä hetkellä ei, että siitä on pitkälti luovuttu, että kyl silloin tietysti alku aikoina kun ei asiakkaita ollut niin paljon niin silloin on aika paljon tietysti tehty sitä mut se on myös ollut sitä et rakennetaan sille alalle sopivaa kun keskustellaan alan parhaimpien toimijoiden kanssa ja sitä kautta me saadaan tehtyjä jotain

Nykyisten asiakkaiden erikoistarpeita palvellaan mahdollisuuksien mukaan, mutta toteutettavien ominaisuuksien on istuttava kokonaisuuteen:

> Niin, sen pitää niinku auttaa kaikkia. Et meillä on tietynlainen tapa millä me vaan priorisoidaan se työmäärän haasteellisuuden ja hyödyn suhteen.

Yksi tapa mukautua asiakkaiden tarpeeseen multitenantissa ympäristössä, vaarantamatta sovelluksen ytimen yhtenäisyyttä on aukaista sovelluksen rajapintoja asiakkaan vapaasti käytettäväksi. Rajapintojen avaamisella mahdollistetaan myös helpompi mobiilisovellusten toteuttaminen ja toimittajariippumattomien laajennusten teettäminen kolmansilla osapuolilla:

> Ja aika paljon justiinsa API:n päälle kuitenkin rakennetaan, että jos meillä on, vaikka meidän oma mobiilisofta mikä nyt tuli niin se on vaan tehty ihan puhtaasti sen API:n

> päälle että kuka vaan vois periaatteessa tehdä sellaisen itelleen ja se on ihan avoin dokumentaatio mikä sieltä löytyy. Ja paljon just tämmöisiä raportontihaasteita muita mitä tulee isoilla konserneilla niin ne vedetään sen API:n kautta.

Avointen rajapintojen lisäksi integraatiot muihin järjestelmiin ovat toiminnanohjausjärjestelmissä tärkeitä. Erityisesti Suomessa tunnettuihin taloushallinnon järjestelmiin täytyy löytyä integraatiot.

7.3 Tapaus C – Multitenantti yleiskäyttöinen PSA

Tapaus C tarjoaa myös asiantuntijayrityksille suunnattua toiminnanohjausjärjestelmää. Versioita on tarjolla kolme, versiot on suunnattu lähinnä erikokoisten yritysten tarpeita ajatellen. Hinnoittelu on asiakaskohtainen. Asiakaskohtaiset hinnat vaihtelevat käytetyn paketin mukaan, aloitustason sovelluksessa ne ovat pienimmät ja kasvavat ominaisuuksien määrän mukaan. Myöskään yritys C:n tarjonnassa ei ole suoraan tietylle toimialalle suunnattuja versioita, vaan toimialakohtaiset erikoisuudet on käsitelty mallipohjilla, joilla sovellus saadaan konfiguroitua tietylle toimialalle sopivaksi:

> Mutta sitku otetaan se softa käyttöön, niin meillä on ihan niinku teknisessä mielessä eroteltu tämmöset niinku mallipohjat, mitä me käytetään. Eli se niinku sisältää totta kai dokumentaation mutta myös sen, että me otetaan pakasta toimialalle sopiva template ja sit sillä on sopivia ominaisuuksia ja jos aatellaan nyt vaikkapa mainostoimistoja niin se tarkoittaa sitä että siellä ei välttämättä puhuta niin paljon semmosista tietyntyyppisistä raporteista mitä insinööritoimistoissa seurataan, eli ne on pois.

Sovellukset toimivaat yksinomaan toimittajan pilvipalvelussa. Kanta asiakkaan omilla palvelimilla tapahtuvaan hostaukseen on selkeästi kielteinen:

> No, meillä kerran oli, ei tuu toista kertaa. Eli siis kyllä me, se on sataprosenttisesti niin että me hostataan ja tarjotaan.

Niistä luovuttiin muun muassa ylläpidon hankaluuden takia ja siitä syystä, että asiakkaan ympäristössä sijaitsevan sovelluksen ylläpidon aiheuttamat kustannukset olivat suuret, eikä niitä voitu lopulta periä täysimittaisena asiakkaalta. Yhden asiakkaan ja multitenantin ympäristön ero voidaan nähdä osana samaa jatkumoa. Ylläpidon ja hallinnan hinta laskee ympäristöjen vähetessä. Yrityksen palvelu on sovellustasolla multitenantti. Jokaisella asiakkaalla on kuitenkin oma tietokantansa. Asiakkaan tietojen käsittelyyn liittyen on myös erilaisia tarpeita:

> jotku tilaakin säännölliset tämmöset niinku offline backupit ittelleen, tai sit myös on mahdollisuus tähdätä niinku escrow palvelua itselleen et tavallaan me taataan et me

> siirretään niinku datat ja muut tiettyyn paikkaan talteen, riippumattomalle taholle, jotta ne saa tietyissä epäselvyystilanteissa.

Backupit, escrow –palvelu ja vastaavat varmistukseen ja tiedon keräämiseen liittyvät toimenpiteet on helppo toteuttaa standardityökaluilla, kun asiakkaalla on oma tietokantainstanssinsa. Korkeamman maturiteettitason mallissa, esimerkiksi taulujen asiakaskohtaisilla nimeämisillä tai tenanttitunnisteella jokaisessa taulussa, työkalut tämänkaltaisia toimenpiteitä varten täytyisi myös koodata itse.

Myös yritys C pyrkii tekemään asiakaskohtaisia muutoksia ainostaan silloin kun ne voidaan tuoda yleiseen versioon ja niistä on yleisempää hyötyä asiakkaille. Joitain asiakasspesifejä tarpeita pystytään toteuttamaan myös avattujen rajapintojen kautta:

> varsinkin tällaisia juttuja että halutaan joku liittymä, että miten tuodaan ulkoa, vaikka justiin alihankintoja tai myyntejä, niin meillä on tottakai standardirajapinnat millä tuot jotain ostoja ja myyntejä sisään jotka pohjautuu johonkin tiettyyn softaan joka toimii tietyllä tavalla, mutta jos onki joku ihan tämmönen custom tarve niin ollaan tehty esimerkiksi näin että ottaa excelin, ja exceliin on tehty simppelit makrot, jotka tunkee REST:llä tavaraa sisään

Rajapintojen nähdään soveltuvan erityisen hyvin esimerkiksi asiakkaalle spesifien raporttien tuottamiseen:

> koska se voi olla ihan niinku täysin spesifi, miten heidän firmassa jotkut tiimit on jaettu ja on sovittu, että toi pena tossa myy tota ja toi voi tota. Niin ei me voida semmoista niinku mallintaa sinne softaan et se ois kestävällä pohjalla, ja silloin me voidaan ottaa REST:llä se data ulos ja tavallaan tehdä pivotti joka sitten niinku laskee tietyllä kaavalla tai miten se asiakas on sen sillä hetkellä mieltänyt

Asiakkaan itsepalveluna tapahtuva käyttöönotto on periaatteessa mahdollista mutta se nähdään käytännössä ongelmallisena, ensinnäkin siksi että monesti itsepalvelukäyttöönottoa yrittävät haluavat säästää vääristä paikoista:

> siinä on pari juttua, että yks on se tietysti, että löytyykö sitä itsekuria tehdä site hommaa kunnolla totta kai on se ensimmäinen homma aika monesti se on ne jotka haluu sen itte ottaa ja optimoida sen ettei menis rahaa niin ne on myös niin kiireisiä ettei senkään puolesta tuu mitään niille vois olla ihan tervettä että joku tavallaan ottaa sen aikalisän ja istuis niinku muutaman tunnin edes niitten ihmisten kanssa läpi ja kertois että miks tämmöisiä järjestelmiä ylipäänsä kannattaa käyttää…

Toisekseen siksi että datan sovittaminen yhteen muiden järjestelmien kanssa vaatii monesti asiantuntija-apua:

> Sit se toinen juttu on siinä se että koska tässä tulee niitä liittymiä sitte helposti joihinkin muihin softin, taloushallintoon ja kirjanpitoon ja muualle niin jos näitä niinku aattelee että jotenkin itte vaan tuolta exportoi jotain ja tilitoimisto osaa sen homman hoitaa, niin se loppujen lopuks kuitenkin saattaa kaatua niin että me sitten tukipalvelun puolella hoidetaan sitä tavallaan ilmaiseks sitä asiaa.

Itsepalvelu ja helppo käyttöönotto nähdään siis hyvää keinona, jos tavoitteena on maksimoida uusien asiakkaiden määrä. Haastateltava ei kuitenkaan näe sen johtavan pitkäaikaisiin asiakkuuksiin ja kannattavaan liiketoimintaan.

7.4 Tapaus D – Multitenantti toimialaspesifi ERP

Yritys D on kasvuvaiheessa oleva yritys, jonka tuote on edelleen kehityksen alla. Tuote on suunnattu autoalalle, asiakkaina on muun muassa autohuolto- ja renkaanvaihtepalveluita tarjoavia yrityksiä. Tuotteen ensimmäinen versio tehtiin suoraan ensimmäisen asiakkaan tarpeita varten. Palvelu on jaettu itse sovellukseen, joka tarjoaa huolto- ja varastotoiminnallisuudet ja datapalveluun joka tarjoaa huolto- ja ajoneuvodataa. Lisäksi on olemassa mobiilisovelluksia, esimerkiksi rengashallityöntekijän sovellus jonka avulla mobiililaitteella voi skannata viivakoodit rengas- ja hyllytarroista ja niiden avulla hallita varastopaikkoja. Palvelun hinnoittelu on käyttäjäkohtainen, lisäksi datapalveluilla on transaktiopohjainen hinnoittelu. Muista tapauksista poiketen tuotetta ei ole siis tavoitteena laajentaa koskemaan muita toimialoja, vaan tarkoituksena on palvella autohuoltoalan yritysten erityistarpeita.

Toimialan erityispiirteitä on esimerkiksi tavarantoimittajien katalogidata, joka pitää syöttää järjestelmään ja saada sovitettua huoltoihin:

> pitää saada toimipistetiedot, pitää saada työntekijätiedot, pitää saada asiakasrekisterit, mitä ajoneuvoa siellä on, mitä tilauksia siellä on, pitää tämmöiset taustatiedot pystyä ajamaan…tuotekatalogit pitää tuoda ja silloin puhutaan kuitenkin siitä, että pitää kysellä, tai keskustella, neuvotella heidän kanssaan mitä toimittajia heillä on Suomessa. Meillä on tiettyjen toimittajien tuotekatalogit, joihin pystytään meidän huollot mätsäämään ja sitä kautta he pystyy suoraan tilaamaan näitä huoltoja.

Integraatioita olemassaoleviin järjestelmiin on myös tehty, eli palvelu voi toimia osana kokonaisuutta esimerkiksi kuluttajien ajanvarausväylänä olemassa olevaan toiminnanohjausjärjestelmään. Autokorjaamojen toiminnanohjausjärjestelmät ovat monesti alalle spesifejä sovelluksia.

Palvelu toimi alun perin suomalaisen palveluntarjoajan pilvipalvelussa dedikoidulla palvelimella, mutta myöhemmin se siirrettiin Amazonin pilvipalveluun:

> me sitten tehtiin päätös siirtyä amazoniin, ja itse asiassa myös kustannussyistä ja siitä että miten helppo se verkko on konfiguroida siellä ja myös se että yleisesti ottaen enemmän osaajia amazoniin.

Taustasovelluksen ja käyttöliittymän välinen kommunikaatio on toteutettu REST –rajapintaa hyödyntäen ja lähtökohta on että rajapintojen käyttö sallitaan myös muille osapuolille. Avoimien rajapintojen hyödyntäminen on muita tapauksia selkeämmin tietoinen strateginen valinta, omat käyttöliittymät

nähdään enemmänkin referenssitoteutuksina, joilla pääsee käsiksi dataan, joka on varsinainen myytävä tuote:

> muutenkin kun ajatus on se, että se meidän toteutus, oli se sit kuinka hyvä tahansa, niin se on kuitenkin referenssitoteutus ja taustajärjestelmä on se mikä me halutaan myydä niin hehän voi tehdä siihen päälle sen mobiilisovelluksen tai nettisovelluksen niin, me tarjotaan vaan et mitkä on ne kaikki mahollisuudet miten sitä API:a käytetään niin he saavat sit vapauden myös tehdä siitä paremman.

Sovellusta rakennetaan multitenantiksi. Tietokannoista on jokaisella asiakkaalla oma versio, sillä ratkaisu katsottiin selkeäksi, helpoksi toteuttaa ja myös suurten tuotekatalogimassojen käsittelyn kannalta turvallisimmaksi:

> päädyttiin tähän erillisten kantojen rakenteeseen, ehkä selkeyden ja helppouden vuoksi.

Asiakkaan itsepalvelukäyttöönotto on suunnitteilla, käyttöönotosta halutaan tehdä mahdollisimman helppo:

> jos nyt tulis joku pilvimyynti tai pilviostokanava et sää voit sit suoraan netistä klikata niin siinähän pitäis olla sit semmoinen, et jos sä haluut laittaa sun toimipisteet työntekijät tänne sisään tai tuotekatalogit, niin ne pitää olla tämmöisessä muodossa… Lataat tämmöisessä muodossa tuotteet niin kaikki menee ok. Ja luodaan niinku semmoinen prosessi siihen oston tai pystytyksen yhteyteen, jotta se onnistuu se niinku asiakas tai yritys luomaan sen palvelun sinne, ilman että meidän tarttis olla siinä prosessissa mukana

Tapaus D poikkeaa muista tapauksista ensinnäkin keskittymällä ainoastaan yhteen toimialaan. Sovellus on alusta asti suunniteltu siten että rajapintoja voidaan hyödyntää. Versioinnin näkökulmasta sovellus on jaettu osittain kerroksiin ja palveluihin, joita voidaan yhdistellä tarpeen mukaan.

7.5 Tapausten vertailu

7.5.1 Versiointi

Yritykset A, B ja C toimivat kaikki usean eri toimialan asiakkaiden parissa. B ja C olivat kuitenkin keskittyneet asiantuntijaorganisaatioihin, joiden tarpeet olivat jossakin määrin yhteneväisiä. Näiden asiakkaiden koko oli monesti pienestä keskisuureen, kun taas yritysten A ja D toimialoilla oli myös suurempia asiakkaita. Yrityksellä A oli kaikista laajin toimialojen kirjo, kun taas yritys D keskittyi palvelemaan ainoastaan yhtä toimialaa. Jokaisella yrityksistä oli sovelluksessaan myös valinnaisia moduleja, joilla pystyttiin kattamaan joko eri toimialojen tai toimialasta riippumatta eri asiakkaiden tarpeita. Taulukossa 11 on esitelty yhteenveto yritysten tarjoamista versioista.

TAULUKKO 11 Tuotteiden versiot

Yritys	Versioiden määrä	Toimialakohtaiset versiot	Valinnaisia ominaisuuksia
A	4	Kyllä	Kyllä
B	3	Ei	Kyllä
C	3	Ei	Kyllä
D	2	Ei	Ei

Versioiden lukumäärä, jos ei huomioida yritys A:n toimialakohtaisia versioita, oli 1-3. Valinnaiset lisäominaisuudet täydensivät kolmessa ensimmäisessä tapauksessa perusversiota, eikä yksikään yritys rajoittanut lisäominaisuuksia perusversion perusteella.

Yhdessäkään haastattelussa projektityönä tehtyjä sovellusversioita ei pidetty erityisesti tavoittelemisen arvoisena. Yritysten pääbisnes oli tuotteen lisenssi- tai käyttömaksuissa. Asiakaskohtaiset räätälöinnit ja projektimyynti voivat kuitenkin olla tarpeellisia etenkin yrityksen alkuvaiheessa tai hyvin toimiala- tai asiakasspesifejä ratkaisuja rakennettaessa.

Toimialakohtaisia versioita oli ainoastaan yrityksellä A, johtuen todennäköisesti siitä, että yrityksen asiakkaiden toimialojen kirjo oli laajempi kuin muilla tapausyrityksillä. Toimialakohtaiset versiot syntyivät ensin yksittäiselle asiakkaalle räätälöidyn sovelluksen pohjalta, muokkaamalla siitä myöhemmin toimialalle yleisemmin sopiva versio.

7.5.2 Hinnoittelu

Hinnoittelu oli kaikissa tapauksissa joustavaa. Varsinaisia listahintoja käytettiin vähän, monesti listahintoja ei edes julkaistu yrityksen verkkosivuilla. Yritys A tarjosi edelleen lisenssipohjaista hinnoittelua ja sen rinnalla myös käyttäjäkohtaista hintaa asiakkaan preferenssin mukaan. Yritykset B-D painottivat käyttäjäpohjaista hinnoittelua, vaihtelevan yrityskohtaisen maksun lisäksi. Hinnoittelun painopisteen nähtiin siirtyvän jatkuvasti kohti käyttäjäkohtaisia maksuja. Käytössä oli myös erilaisia tasoja valitun sovellusversion ja lisäominaisuuksien perusteella. Alennukset käyttäjämäärän kasvaessa olivat myös yleisiä.

Ominaisuuksien perusteella hinnoittelua tapahtui kaikilla yrityksistä, pois lukien yritys D. Eri hintaiset versiot poikkesivat aina ominaisuuksiltaan. Monesti ominaisuudet oli valittu siten, että ne sopivat eri kokoisille yrityksille, halvimman version sopiessa paremmin pienille yrityksille. Lisäksi yritykset A – C tarjosivat valinnaisia ominaisuuksia, joita pystyi ottamaan käyttöön versiosta riippumatta. Yritysten hinnoittelu- ja liiketoimintamallit on esitelty taulukossa 12.

TAULUKKO 12 Hinnoittelu- ja liiketoimintamallien vertailu

Tapaus	Hinnoittelumalli	Liiketoimintamalli	
Yritys A	Lisenssi tai perus +	Enterprise SaaS	

	käyttäjäkohtainen		
Yritys B	Perusmaksu + käyttäjäkohtainen	Pure-play SaaS	
Yritys C	Perusmaksu + käyttäjäkohtainen	Pure-play SaaS	
Yritys D	Perusmaksu + käyttäjäpohjainen + transaktiopohjaiset palvelut	Pure-play SaaS	

7.5.3 Arkkitehtuuri

Kolme tapausyrityksistä toimi jollain tasolla multitenantissa ympäristössä. Tästä huolimatta yhteinen tietokanta oli ainoastaan yhdellä tapauksista. Tietokantatason multitenancy ei siis vaikuta tavoitettavan arvoiselta ainakaan yrityksille palveluja myytäessä. Tietokantojen varmuuskopiointiin liittyvät haasteet ja tekniset ongelmat saattavat olla osasyynä tähän. Myös yrityksen C mainitsema escrow –palvelu, eli datan tai backupin siirtäminen riippumattomalle kolmannelle osapuolelle sopimussyistä, voi olla asia joka puhuu eriytetyn tietokannan puolesta. Lisäksi ongelmiksi mainittiin esimerkiksi asiakaskohtaiset tietokantojen huoltotarpeet ja tehokkuushuolet. Eriytetyistä tietokannoista huolimatta asiakaskohtaiset tietokantaskeemat, esimerkiksi omat taulut tai kentät, eivät olleet käytössä multitenanteissa sovelluksissa. Osittain tämä selittyy sillä, että vaikka asiakaskohtaisia tietokantalaajennuksia olisi helppo tehdä tällä kantaratkaisulla, muutokset välittyvät sovelluskoodiin, joka haluttiin kaikissa tapauksissa pitää mahdollisimman yhtenäisenä tenancy- ja tietokantaratkaisuista riippumatta.

Kaikkien tapausyritysten tuotteet pystytään kustomoimaan ajonaikaisella konfiguraatiolla. Toimialakohtaisiin käyttöönottoihin käytettiin joko valmiita templateja tai erillisiä sovelluksia tai sitten toiminnallisuus pyrittiin tekemään mahdollisimman yleiskäyttöiseksi kaikilta osin. Yksi yleistyvä tapa tarjota asiakaskohtaisia lisäominaisuuksia on rajapintojen avaaminen. Esimerkiksi mobiilisovellukset tarjotaan monesti hyödyntämällä avoimia rajapintoja. Samaten erilaiset dataliittymät, raportoinnin tarpeet ja muut vastaavat oli helpompi hoitaa niiden avulla. Pisimmälle tämä ajattelumalli oli nähty tapauksessa D, jossa käyttöliittymä keskusteli backendin kanssa REST –rajapinnan avulla ja omat sovellukset nähtiin enemmänkin referenssisovelluksina ja ulkoiset toimijat olivat vapaita toteuttamaan omia sovelluksiaan taustajärjestelmän päälle.

Multitenanttiin ympäristöön siirtyminen vaikutti monessa tapauksessa tapahtuneen sovelluksen evoluution myötä. Kaksi haastateltavista kertoi, että yrityksen alkuvaiheissa kustomointia ja asiakaskohtaisia asennuksia tehtiin, mutta ympäristöjen hallinnasta johtuvat ongelmat aiheuttivat niistä luopumisen hiljalleen.

Taulukossa 13 on esitetty yhteenveto tapausten arkkitehtuureista. Hiukan yllättäen yritysten välillä ei ollut kovin suuria tai helposti selitettäviä eroja

arkkitehtuurissa. Yksi syy tähän voi olla se, että suuri asiakasmäärä pakottaa lopulta yritykset käyttämään samankaltaisia ratkaisuja ylläpidettävyys- ja hallittavuushaasteiden ratkaisemiseksi. Alkuvaiheessa erilliset ympäristöt ja asiakaskohtaiset versiot voivat olla helposti hallittavissa mutta asiakasmäärän kasvaessa ongelmatkin kasvavat. Myös yritys A:n kohdalla, vaikka sovellukset sijaitsivat useilla eri palvelimilla, ohjelmiston koodipohja pyrittiin pitämään yhtenäisenä.

TAULUKKO 13 Arkkitehtuurien vertailu

Tapaus	**Multi-tenantti**	**Asiakaskohtainen tietokanta**	**Asiakaskohtaisten ominaisuuksien toteutus**
Yritys A	Ei	Kyllä	Konfigurointi / Versiointi
Yritys B	Kyllä	Ei	Konfigurointi
Yritys C	Kyllä	Kyllä	Konfigurointi
Yritys D	Kyllä	Kyllä	Konfigurointi / Versiointi

7.5.4 Muut huomiot

Varsinaista palvelumyyntiä teki vain yksi yritys. Muilla palvelun laajuus rajoittui käyttöönoton aikaiseen opastukseen ja koulutukseen, tekniseen asiakaspalveluun ja vastaaviin tukitoimintoihin. PSA –tarjoajilla liittymät tarjontaa tukeviin järjestelmiin ja esimerkiksi kumppanuudet esimerkiksi tilitoimistojen kanssa nähtiin tärkeämpänä, jolloin ei haluttu lähteä kilpailemaan tilitoimistojen kanssa koska he olivat myös tärkeitä asiakkaita ja toimivat monesti myös suosittelijoina, kun tilitoimiston asiakas harkitsi toiminnanohjausjärjestelmän ostoa. Erilaiset kumppanuusverkostot myös integraatioiden osalta olivat tärkeitä, jos omaa sovellusta ei haluttu lähteä kattamaan kaikkia mahdollisia asiakkaiden tarpeita.

Nykypäivänä pilvipalveluja ostavat asiakkaat eivät enää epäile siirtää tietojaan pilvipalveluun. Huolet tietoturvasta ja käytettävyydestä ovat hälventyneet muun muassa erilaisten kuluttajille suunnattujen pilvipalveluiden yleistyttyä. Maailmalla menestyneiden palvelujen katsottiin vaikuttavan myös hinnoitteluun. Kuluttajien kokemukset voivat siis siirtyä kuluttajapalveluista myös työelämään pienellä viiveellä.

Kaikki haastatteluun osallistuneet olivat samaa mieltä siitä, että nykyään palvelun ulkoistamisessa pilvipalveluun ei ole enää asiakkaan suunnalta epäilyksiä. Datakeskuksen sijainnillakaan ei monissa tapauksissa ollut merkitystä, jotkut asiakkaat tosin katsoivat eduksi sen, että data sijaitsee Suomessa ja on edelleen olemassa sopimuksia, viranomaisvaatimuksia ja muita seikkoja jotka voivat rajoittaa datan käsittelyä ja maantieteellistä sijaintia. Palveluita tarjottiin sekä yksityisten että julkisten pilvipalvelujen kautta.

Sovelluksen itsepalvelukäyttöönotto oli teknisesti mahdollista useimmissa yrityksistä. Sen nähtiin olevan hyvä keino asiakkaiden nopeaan hankkimiseen mutta monelta kohti ongelmallinen. Kaikilla yrityksillä lähtökohtana oli, että käyttöönotto vaatii vähintäänkin lyhyen koulutuksen asiakkaan kanssa. Itsepalvelukäyttöönoton vaarana oli säästäminen väärästä paikasta, jolloin kustannukset siirtyvät palveluntoimittajalle, kun asiakas ei pääse tehokkaasti liikkeelle sovelluksen käytössä.

8 POHDINTA

Tässä luvussa verrataan saavutettuja tuloksia ensiksi alkuperäiseen tutkimusongelmaan pääongelman ja jokaisen aliongelman osalta. Tuloksia verrataan myös kirjallisuuskatsaukseen ja lopuksi arvioidaan niiden merkitystä ja luotettavuutta.

8.1 Tutkimusongelmat

Tutkimusen pääongelma oli "Miten SaaS -palveluntarjoajat hyödyntävät tuotedifferointia tuotteissaan?". Pääongelmaan liittyvät läheisesti myös aliongelmat "Mitkä tuotedifferoinnin menetelmät ovat käytössä?" ja "Minkälaisille asiakasryhmille tuotteen eri versioita tarjotaan?", jotka edustivat tutkimusongelman liiketoiminnallista näkökulmaa, johon vastaan ensiksi.

Käytännössä esiintyi ainoastaan horisontaalista differointia, jota hyödynnettiin kahdella tavalla. Ensinnäkin yleistä oli, että tuote oli jaettu eri hintaisiin versioihin. Halvemmilla versioilla tavoiteltiin monesti pienempiä yrityksiä, kalliimpien versioiden ominaisuudet liittyivät monesti esimerkiksi raportointiin tai muihin järjestelmiin tarjolla oleviin liittymiin, jotka olivat tarpeellisia toimintoja suuremmille asiakkaille. Toisena differointimenetelmänä yhdellä yrityksistä oli tarjolla toimialakohtaisia versioita, jotka oli suunniteltu erityisesti yhden toimialan tarpeita varten.

Tuotteita tarjottiin kaikenkokoisille asiakkaille, eikä keskittymistä esimerkiksi pienten asiakkaiden palvelemiseen ollut, vaan asiakasryhmät rajattiin käytännössä enemmänkin asiakkaiden toimialan mukaan. Erityisesti yritysmaksun ja käyttäjäkohtaisen kuukausimaksun yhdistelmää suosittiin koska sillä hinta saatiin skaalautumaan asiakkaan koon ja käytön mukaan. Tyrväisen ja Selinin (2011) tutkimuksessa olleet suomalaiset SaaS – palveluntarjoajat tarjosivat myös tuotteitaan erikokoisille asiakkaille, pienempien yritysten keskittyessä monesti myymään self-servicenä

käyttöönotettavaa ohjelmistoa pienille asiakkaille. Tapausyritykset eivät kuitenkaan suhtautuneet self-service ohjelmistoihin täysin varauksetta. Tiiviimmin yrityksen prosesseihin ja ydintoimintoihin kytkeytyvät ohjelmistot harvoin saa käyttöön ilman kustomointia ja ohjelmistoalan osaamista, jolloin käyttöönoton kustannukset voivat olla pienelle asiakkaalle liian suuret. Toimialakohtaisia erojakin on, asiantuntijayrityksille ohjelmistoa tarjonneen yrityksen mukaan esimerkiksi insinööritoimistot pystyisivät helposti konfiguroimaan sovellukset itsekin.

Seuraavat aliongelmat liittyivät enemmän tuotteiden teknisiin ominaisuuksiin ja SaaS –sovelluksen versioinnin teknisiin haasteisiin. Ensimmäinen käsiteltävä aliongelma on "Miten tarjottaviin moduuleihin tai sovellusversioihin ja niiden hinnoitteluun on päädytty?" Yleisin tapa jakaa sovellus versioihin perustui asiakasyrityksen kokoon. Eri versioiden ominaisuudet liittyivät kaikissa tapauksissa eri kokoisten yritysten tarpeisiin esimerkiksi raportoinnin osalta. Toimialakohtaisia versioita oli ainoastaan yhdellä yrityksistä ja ne olivat syntyneet tuotteistamalla yhden yrityksen tarpeeseen rakennettu sovellus. Valinnaiset moduulit olivat usein edistyneempään käyttöön tarkoitettuja ominaisuuksia tai esimerkiksi integraatiota vaativia ominaisuuksia, kuten verkkolaskupalvelu tai datan synkronointi ulkoiseen järjestelmään, joita ei peruspaketissa voitaisi ottaa käyttöön ilman lisätyötä ja mahdollisia maksullisia ulkoisia palveluita.

Toinen aliongelma liittyi asiakaskohtaiseen räätälöintiin: "Kuinka paljon asiakaskohtaista räätälöintiä tehdään?". Asiakaskohtaiset versiot ovat varmasti tehokkain tapa yksittäisen asiakkaan tarpeisiin vastaamiseen, mutta erityisesti SaaS –palveluissa niiden toteuttaminen on haastavaa. Asiakaskohtaiset versiot nähtiin ongelmallisina pääasiassa ylläpidettävyyden näkökulmasta. Kaksi tapausyrityksistä kuitenkin teki räätälöityjä versioita, ensimmäinen mahdollisesti toimialan erityispiirteiden takia ja toinen varhaisen tuotekehityksen tukena. Uusien tuotteiden tai toimialakohtaisten versioiden kehittäminen nähtiinkin hyvänä perusteena asiakaskohtaisille räätälöinneille. Eniten räätälöintiä tekevän yrityksen tapauksessa ohjelmiston lisensointi oli myös yleisesti käytössä isommilla asiakkailla, jotka vastasivat Ojalan (2013) esittämän mallin kriteereitä. Konfiguraation kautta tehtävää asiakaskohtaista kustomointia esiintyi kaikissa sovelluksissa. Vähimmillään esimerkiksi käyttöliittymän teemoitus ja asiakaskohtaiset logot sivustolle tai vaikka laskuihin täytyi olla konfiguroitavissa.

Viimeinen aliongelma oli "Minkälaisia arkkitehturaalisia haasteita ohjelmiston versiointi on asettanut tuotteelle?". Multitenantin arkkitehtuurin etuina on katsottu olevan tehokas resurssien käyttö, joka mahdollistaa palvelun kustannustehokkaan tarjoamisen myös pienemmille asiakkaille (Chong & Carraro, 2006). Tämä näkyi myös tuloksissa siten, että yritykset jotka tarjosivat tuotettaan myös pienemmille asiakkaille, ajoivat kaikki palveluaan multitenantissa ympäristössä. SaaS –maturiteettimallien ylimmät tasot eivät kuitenkaan kaikkien osa-alueiden osalta vaikuttaneet toivotuilta. Erityisesti tietokantojen osalta erillinen tietokanta yhteisellä tietokantaskeemalla kaikille

tenanteille näytti olevan yleisin ja myös käytännöllisin. Asiakaskohtaisia tauluja tai tietokantakolumneja ei haluttu kuitenkaan tehdä, vaan kaikki asiaskohtaisesti eriytetty toiminnallisuus toteutettiin parametrien avulla. Krebs, ym. (2012) mainitsemista haasteista suorituskykyisolaatio tuli haastatteluissa usein esille, tietokantojen eriyttämisen nähtiin omalta osaltaan parantavan asiakaskohtaista suorituskykyä.

Riippumatta siitä oliko käytössä single- vai multitenantti ympäristö, kaikkien sovellusten asiakaskohtaiset toteutukset pyrittiin tuomaan aina osaksi yhteistä koodipohjaa. Tämä johti käytännössä siihen, että sovellukset sisälsivät satoja tai tuhansia erilaisia ajonaikaisia parametreja, joilla sovellus muokattiin asiakkaan toiveiden mukaiseksi. Asiakasmäärän kasvaessa suuri parametrimäärä vaikeutti ylläpitoa ja teki ohjelmistoista monimutkaisia.

Yksi tapa hallita suurta määrää parametreja olivat erilaiset mallipohjat, joilla kerättiin tietylle toimialalle ominaisia parametrivalintoja yhteen. Ohjelmistotuotelinjojen parissa tehty tutkimus varioinnin ja arkkitehtuurien osalta voisi olla hyödyksi myös SaaS –sovellusten versioinnissa. Tapausyrityksissä oli törmätty samankaltaisiin haasteisiin kuin esimerkiksi Raatikaisen, ym. (2003) tutkimuksessa konfiguroitavia ohjelmistotuoteperheitä käyttäneet yritykset. Sovellusten pilkkominen komponenteiksi tai palveluiksi, joista voitaisiin yhdistellä asiakas- tai toimialakohtaisia versioita, voisi selkeyttää ja helpottaa tuoteversioiden rakentamista. Erilaisia tapoja palveluiden hyödyntämiseen multitenantissa ympäristössä on jo esitetty (Mietzner, ym. 2011).

Toiminnanohjausjärjestelmien käyttöönotossa tietojen importointi vanhoista järjestelmistä ja erilaiset integraatiot muihin järjestelmiin ovat yleisiä. Monesti niitä tehdään puoliautomaattisesti erilaisia skriptejä ja makroja hyödyntämällä tai pienempien aineistojen osalta myös käsin syöttämällä. Avointen rajapintojen merkitys voi kuitenkin kasvaa sovellusten siirtyessä osaksi PaaS –ekosysteemejä. Avoimilla rajapinnoilla asiakas voi itse tai kolmannen osapuolen toteuttamana laajentaa sovelluksen ominaisuuksia esimerkiksi raportoinnin tai mobiilikäyttöliittymien avulla. Boschin (2009) esittämistä argumenteista ohjelmistoekosysteemiin siirtymisen puolesta ainakin kehitystyön kustannusten jakaminen partnereiden kanssa toteutui jo yhdessä tapausyrityksessä ja sitä kautta myös asiakasarvon lisäys. Tekniseltä näkökannalta voitaisiin myös esittää, että alustan tarjoava yritys voi keskittyä omaan ydintoiminnallisuuteensa ja tarjota asiakkaille ainoastaan alustan erikoisempien vaatimusten toteuttamiselle.

8.2 Tutkimuksen rajoitteet

Yksi ongelma tulosten yleistettävyyden kannalta on tapausten ja haastateltavien lukumäärä. Haastateltuja henkilöitä oli ainoastaan yksi per yritys ja haastattelujen painopiste saattoi vaihdella hiukan henkilön vastuualueen mukaan. Kaikilla haastatelluilla oli kuitenkin jonkinlainen rooli tuotteen

kehityksessä, mutta osa työskenteli enemmän myynnin ja osa arkkitehtuurin parissa. Ideaalisesti jokaisesta yrityksestä olisi ollut molempien näkökantojen edustaja mutta tarvittavien henkilöiden hankalan tavoitettavuuden myötä tämä ei ollut mahdollista. Haastattelujen sisällön suhteen tämä aiheutti kuitenkin vähän ongelmia, sillä esimerkiksi arkkitehtuuria koskevat kysymykset eivät olleet erityisen teknisiä ja niihin saatiinkin vastaus riittävällä tasolla.

Tutkimuksen aihealueen rajaus oli laaja ja monitieteellinen. Tämä asetti omat ongelmansa haastattelujen ja niiden tulkinnan suhteen. Haastattelujen painopiste vaihteli kaupallisen ja teknisen näkökulman välissä joka vaikeutti tapausten vertailua keskenään. Vastaukset kysymyksiin jäivät joissain kohti melko pinnallisiksi ja toista näkökulmaa tai tarkentavia kysymyksiä olisi tarvittu.

Kaikki tapaukset edustavat samantyyppistä ohjelmistoa, toiminnanohjausjärjestelmää, vaikkakin ne poikkeavat toisistaan osittain arkkitehtuurin ja myös asiakkaiden toimialojen osalta. Silti tulosten yleistettävyys vaatisi laajemman otannan ja vertailun sovellustyypeittäin.

8.3 Tulosten merkitys tieteelle ja käytäntöön

Tulokset osoittivat osittain, että ohjelmistotuoteperheiden, erityisesti konfiguroitavien ohjelmistotuoteperheiden tuloksia voidaan ja tulisi soveltaa myös SaaS –maailmassa. Versioiden rakentaminen konfiguraation avulla on tiettyjen ohjelmistotyyppien parissa hyvin yleistä ja samat ongelmat jotka on tunnistettu ohjelmistotuoteperheitä koskevassa kirjallisuudessa (Raatikainen, ym. 2003) tulevat eteen myös käytännön elämässä. Palveluarkkitehtuurin ja multitenanttien komponenttien yhdistämistä on jo kokeiltu ja tuloksista voikin olla hyötyä, kun halutaan yhdistää multitenantin sovellusten hyödyt ja samalla helpottaa sovellusten versiointia ja kustomointia (Mietzner, ym. 2011).

Tuloksista voi olla apua käytännön SaaS –palveluiden suunnittelussa. SaaS –sovelluksessa arkkitehtuuriin tulisi kiinnittää erityistä huomiota. Konfiguroitavuus on tärkeä tekijä tarjottaessa asiakkaiden tarpeiden ja käyttökokemuksen huomioimisessa. Konfiguroitavuus voi kuitenkin johtaa vaikeasti hallittavaan kokonaisuuteen. Avuksi tähän voi olla ohjelmiston jakaminen hallittaviin komponentteihin, myös rajapintojen avaamisella voidaan mahdollistaa ohjelmiston erilaiset käyttöskenaariot ilman että kaikkia asiakkaan vaatimuksia tarvitsee toteuttaa itse ohjelmistoon. Single- ja multitenantin välinen valintakaan ei ole joko/tai, vaan välissä on monia variaatioita, joilla on sekä etunsa että haittansa. Tämän tulisi olla kuitenkin liiketoimintaa ja yrityksen strategiaa tukeva päätös, tulosten perusteella arkkitehtuuri kuitenkin muodostuu monesti evoluution kautta eteentulleiden ongelmien muovaamana.

9 YHTEENVETO

Tutkimuksen tavoitteena oli selvittää miten yrityksille suunnattuja SaaS –palveluja tarjoavat yritykset hyödyntävät tuoteversiointia. Tähän sisältyi sekä selvitys siitä, minkälaisille asiakkaille sovellus ja sen eri versiot on suunnattu, että se minkälainen sovelluksen arkkitehtuuri on. Tavoitteena oli tutkia sovelluksen versiointia sekä teknisestä että kaupallisesta näkökulmasta ja selvittää minkälaisia yhteyksiä niiden välillä mahdollisesti on.

Tutkimus toteutettiin monitapaustutkimuksena käyttäen puolistrukturoituja haastatteluja tiedonkeruumenetelmänä. Haastatteluihin valittiin yksi henkilö neljästä eri yrityksestä. Kysymykset liittyivät yrityksen hinnoittelustrategiaan, versioinnin hyödyntämiseen sen osana ja ohjelmiston arkkitehtuuriin.

Hinnoittelustrategian osalta tulokset olivat hyvin samankaltaiset kuin esimerkiksi Ojalan (2012) tutkimuksessa. SaaS –palveluiden hinnoittelumallilla pyrittiin tarjoamaan joustava hinnoittelu sekä pienille että suurille asiakkaille. Ainoastaan yksi yrityksistä tarjosi palveluaan julkisen pilvipalvelun kautta, eivätkä he olleet huomanneet minkäänlaista epäröintiä asiakkaiden suunnalta. Haastateltavien mukaan yleiset asenteet pilvipalveluja kohtaan ovat kääntyneet positiivisemmiksi vuosien kuluessa ja pelot saattavat lieventyä kokemusten kasvaessa. Toisaalta osa palveluntarjoajista piti etuna sitä, että konesalit sijaitsivat Suomessa ja monen asiakkaan kohdalla oli myös erilaisia teknisiä ja laillisia rajoitteita datan sijainnille ja käsittelylle, joten vielä ei voida täysin sanoa, että kaikki asiakkaat olisivat valmiita julkisiin pilvipalveluihin.

Tulokset arkkitehtuurin osalta vastasivat Laatikaisen ja Ojalan (2014) löydöksiä, sovelluksen jako moduuleihin ja erilaisten pakettien tarjoaminen niiden pojalta oli hyvin yleistä. Multitenantti ympäristö myös vähentää mahdollisuuksia yksittäisten asiakkaiden tarvitsemien kustomointien toteuttamiseen, kaiken toiminnan pitää istua kokonaisuuteen ja hyödyttää suurempaa asiakasjoukkoa. Versioinnin tavoitteena oli useimmissa tapauksissa tarjota sovelluksesta eri kokoisille asiakkaille soveltuvia paketteja. Toinen käyttötarkoitus oli toimialakohtaisten versioiden tarjoaminen, mutta ainoastaan

yksi yrityksistä tarjosi sovellustaan tarpeeksi toisistaan poikkeaville toimialoille, jotta tämä oli tarpeen.

Luoman ym. (2012) SaaS –liiketoimintamallien jaottelusta sekä Pure play- että Enterprise –SaaS olivat käytössä tapausyrityksissä. Self service SaaS nähtiin suuntana, johon haluttiin lähteä erityisesti pienten asiakkaiden houkuttelemiseksi. Siihen liittyi kuitenkin ongelmia, jotka liittyivät ohjelmiston kompleksisuuteen, itsepalvelukäyttöönotto ei aina tuottanut optimaalista tulosta asiakkaalle eikä siten ollut välttämättä hyvä ratkaisu pitkän asiakkuuden synnyttämiseksi.

Laatikaisen ja Ojalan (2014) mukaan hinnoitteluun liittyvät päätökset tulisi kommunikoida varhaisessa vaiheessa ohjelmiston tuotekehitystä, sillä ne voivat asettaa vaatimuksia arkkitehtuurin suunnittelulle. Tämän lisäksi myös versiointitarve, kohdeasiakkaat ja heidän käyttötarpeensa tulisi selvittää varhaisessa vaiheessa. SaaS –arkkitehtuurin maturiteettimalliakaan ei voi katsoa pelkästään yhteen suuntaan, jossa yritys aina tavoittelee korkeinta maturiteettia vaan joissain tapauksissa jokin välitaso voi soveltua parhaiten SaaS –palvelun strategiaan. Keinoja yhdistää sekä multitenantin ympäristön etu, että kustomoitavuus voi löytyä esimerkiksi palveluorientoituneen arkkitehtuurin parista (Mietzner, ym. 2011).

9.1 Tuleva tutkimus

Ohjelmistotuotelinjat syntyivät elektroniikkateollisuuden tarpeita ajatellen ja suuri osa tutkimuksestakin on liittynyt fyysisten tuoteperheiden hallintaan ja kehittämiseen. Kuitenkin esimerkiksi konfiguroitavuuteen liittyvät aspektit ovat helposti sovellettavissa myös pilvipalveluiden maailmaan ja yksi hyödyllinen tutkimusalue voisi olla esimerkiksi erilaisten variaatiotekniikoiden hyödyntäminen multitenantissa ympäristössä.

Single- tai multi –tenancyn valintakaan ei ole aina itsestään selvä asia. Single –tenancynkin voi toteuttaa monella tavalla omasta laitteistosta virtualisoituihin ympäristöihin (Chong, 2016) ja virtualisointiteknologian ja virtuaalisten ympäristöjen hallintatyökalujen kehittyessä multitenantin sovelluksen suhteelliset hyödyt voivat pienentyä (Felter, ym. 2015; Benson, ym. 2016). SaaS –arkkitehtuurin ja erityisesti virtualisoinnin tutkimus sen osana voisi olla yksi tutkimuksen suunta.

Yksi mielenkiintoinen kehityssuunta on API –ekosysteemien kehittyminen. Monet suuret yritykset ovat siirtyneet ohjelmistojen ja infrastruktuurin sijasta tarjoamaan alustoja joilla muut yritykset voivat luoda palvelujaan. Avoimille rajapinnoille pohjautuvat ekosysteemit mahdollistavat entistä suurempien asiakasryhmien saavuttamisen ja verkostovaikutukset aivan uudenlaisessa mittakaavassa. Rajapintojen mahdollistamien hinnoittelu- ja ansaintamallien osalta tutkimus on vielä vähäistä.

LÄHTEET

Ahmed, F., Capretz, L. F., & Sheikh, S. A. (2007). Institutionalization of software product line: An empirical investigation of key organizational factors. *Journal of Systems and Software*, *80*(6), 836-849.

Bachmann, F., & Clements, P. C. (2005). Variability in software product lines. Versioning in the Information Economy: Theory and Applications. *CESifo Economic Studies*, Vol. 51, 2-3/2005. 331.

Benlian, A., Hess, T., & Buxmann, P. (2009). Drivers of SaaS-adoption–an empirical study of different application types. *Business & Information Systems Engineering*, 1(5), 357-369.

Benson, J. O., Prevost, J. J., & Rad, P. (2016). Survey of automated software deployment for computational and engineering research. Teoksessa *2016 Annual IEEE Systems Conference (SysCon)* (pp. 1-6). IEEE.

Bezemer, C. P., Zaidman, A., Platzbeecker, B., Hurkmans, T., & Hart, A. (2010). Enabling multi-tenancy: An industrial experience report. Teoksessa *Software Maintenance* (ICSM), 2010 IEEE International Conference on (s. 1-8). IEEE.

Bezemer, C. P., & Zaidman, A. (2010). Multi-tenant SaaS applications: maintenance dream or nightmare?. Teoksessa *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)* (s. 88-92). ACM.

Bhargava, H. K., & Choudhary, V. (2001). Information goods and vertical differentiation. *Journal of Management Information Systems*, 18(2), 89-106.

Birk, A., Heller, G., John, I., Schmid, K., von der Maßen, T., & Muller, K. (2003). Product line engineering, the state of the practice. *IEEE software*, *20*(6), 52-60.

Bontis, N., & Chung, H. (2000). The evolution of software pricing: from box licenses to application service provider models. *Internet Research*, 10(3), 246-255.

Bosch, J. (2009). From software product lines to software ecosystems. Teoksessa *Proceedings of the 13th international software product line conference* (s. 111-119). Carnegie Mellon University.

Bosch, J. (2006). The challenges of broadening the scope of software product families. *Communications of the ACM*, 49(12), 41-44.

Bosch, J. (2002). Maturity and evolution in software product lines: Approaches, artefacts and organization. Teoksessa *Software Product Lines* (s. 257-271). Springer Berlin Heidelberg.

Bühne, S., Chastek, G., Käkölä, T., Knauber, P., Northrop, L., & Thiel, S. (2003). Exploring the context of product line adoption. Teoksessa *Software Product-Family Engineering* (s. 19-31). Berlin Heidelberg: Springer.

Chong, F., Carraro, G., & Wolter, R. (2006). Multi-tenant data architecture. *MSDN Library*, Microsoft Corporation, 14-30. Haettu 14.5.2016 osoitteesta https://msdn.microsoft.com/en-us/library/aa479086.aspx

Chong, F., & Carraro, G. (2006). Architecture strategies for catching the long tail. *MSDN Library*, Microsoft Corporation, 9-10. Haettu 14.5.2016 osoitteesta https://msdn.microsoft.com/en-us/library/aa479069.aspx

Chong, F. (2006). Multi-tenancy and Virtualization. Haettu 22.5.2016 osoitteesta http://blogs.msdn.com/b/fred_chong/archive/2006/10/23/multi-tenancy-and-virtulization.aspx

Clements, P., & Northrop, L. (1999). A framework for software product line practice. *SEI interactive*, 2(3).

Cusumano, M. (2010). Cloud computing and SaaS as new computing platforms. *Communications of the ACM*, *53*(4), 27-29.

Cusumano, M. A. (2008). The changing software business: Moving from products to services. *Computer*, 41(1), 20-27.

Cusumano, M. A. (2007). The changing labyrinth of software pricing. *Communications of the ACM*, 50(7), 19-22.

Cusumano, M. (2003). Finding your balance in the products and services debate. *Communications of the ACM*, 46(3), 15-17.

Darke, P., Shanks, G., & Broadbent, M. (1998). Successfully completing case study research: combining rigour, relevance and pragmatism. *Information systems journal*, *8*(4), 273-289.

Denekere, R.J. and McAfee, R.P. (1996). Damaged goods. *Journal of Economics and Management Strategy* 5, 149–174.

Dubey, A., & Wagle, D. (2007). Delivering software as a service. *The McKinsey Quarterly*, 6(2007), 2007.

Eisenhardt, K. M. (1989). Building theories from case study research. *Academy of management review*, *14*(4), 532-550.

Felter, W., Ferreira, A., Rajamony, R., & Rubio, J. (2015). An updated performance comparison of virtual machines and linux containers. Teoksessa *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On* (pp. 171-172). IEEE.

Giessmann, A., & Stanoevska, K. (2012). Platform as a Service–A conjoint study on consumers' preferences. Teoksessa *Proceedings of the 33rd International Conference on Information Systems*. AIS.

Günzel-Jensen, F., & Holm, A. B. (2015). Freemium business models as the foundation for growing an e-business venture: a multiple case study of industry leaders. *Journal of Entrepreneurship, Management and Innovation* 10. 77-102.

Geyer, L., & Becker, M. (2002). On the influence of variabilities on the application-engineering process of a product family. Teoksessa *Software Product Lines* (s. 1-14). Berlin Heidelberg: Springer.

Haesen, R., Snoeck, M., Lemahieu, W., & Poelmans, S. (2008). On the definition of service granularity and its architectural impact. Teoksessa *International*

Conference on Advanced Information Systems Engineering (s. 375-389). Berlin Heidelberg: Springer.

Hahn, J. H. 2001. Functional Quality Degradation of Software with Network Externalities. Discussion Paper, Keele University.

Harmon, R., Demirkan, H., Hefley, B., & Auseklis, N. (2009). Pricing strategies for information technology services: A value-based approach. Teoksessa *Proceedings of 42nd Hawaii International Conference on System Sciences* (s. 1-10). IEEE.

Harmon, R. Raffo, D. Faulk, S. (2005). Value-Based Pricing For New Software Products: Strategy Insights for Developers. *Portland Int. Conf. on Management of Eng. & Tech.* 2005.

Hinterhuber, A. (2008). Customer value-based pricing strategies: why companies resist. *Journal of business strategy*, 29(4), 41-50.

Ingenbleek, P., Debruyne, M., Frambach, R. T., & Verhallen, T. M. (2003). Successful new product pricing practices: a contingency approach. *Marketing letters*, 14(4), 289-305.

Jacobson, D., Woods, D., & Brail, G. (2011). APIs: A strategy guide. O'Reilly Media, Inc.

Jansen, S., Houben, G. J., & Brinkkemper, S. (2010). Customization realization in multi-tenant web applications: Case studies from the library sector. Teoksessa *International Conference on Web Engineering (ICWE),* 2010,

Jones, R., & Mendelson, H. (2011). Information goods vs. industrial goods: Cost structure and competition. *Management Science*, 57(1), 164-176.

Ju, J., Wang, Y., Fu, J., Wu, J., & Lin, Z. (2010). Research on key technology in SaaS. Teoksessa *2010 International Conference on Intelligent Computing and Cognitive Informatics* (s. 384-387). IEEE.

Kabbedijk, J., Bezemer, C. P., Jansen, S., & Zaidman, A. (2015). Defining multi tenancy: A systematic mapping study on the academic and the industrial perspective. *Journal of Systems and Software*, 100, 139-148.

Kabbedijk, J., & Jansen, S. (2012). The role of variability patterns in multi-tenant business software. Teoksessa *Proceedings of the WICSA/ECSA 2012 Companion Volume* (s. 143-146). ACM.

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., & Peterson, A. S. (1990). *Feature-oriented domain analysis (FODA) feasibility study* (No. CMU/SEI-90-TR-21). Software engineering institute, Carnegie-Mellon university.

Kang, S., Myung, J., Yeon, J., Ha, S. W., Cho, T., Chung, J. M., & Lee, S. G. (2010). A general maturity model and reference architecture for saas service. Teoksessa *Database Systems for Advanced Applications* (s. 337-346). Berlin Heidelberg: Springer.

Katz, M. L., & Shapiro, C. (1985). Network externalities, competition, and compatibility. *The American economic review*, 75(3), 424-440.

Katzan Jr, H., & Dowling, W. A. (2010). Software-as-a-Service economics. *The Review of Business Information Systems*, *14*(1), 27.

Katzmarzik, S. (2011). Product Differentiation for Software-as-a-Service Providers. *Business & Information Systems Engineering*, 3, 19-31.

Krebs, R., Momm, C., & Kounev, S. (2012). Architectural Concerns in Multi tenant SaaS Applications. *CLOSER*, 12, 426-431.

Krueger, C. (2001). Easing the transition to software mass customization. Teoksessa *Software Product-Family Engineering* (s. 282-293). Berlin Heidelberg: Springer.

Kushida, K. E., Murray, J., & Zysman, J. (2012). The gathering storm: Analyzing the cloud computing ecosystem and implications for public policy.*Communications & Strategies*, (85), 63-85.

Laatikainen, G., & Ojala, A. (2014). SaaS architecture and pricing models. Teoksessa E. Ferrari, R. Kaliappa, & P. Hung (toim.), *Proceedings of the 2014 IEEE international conference on services computing (SCC 2014)* (s. 597-604). IEEE.

Laatikainen, G., Ojala, A., & Mazhelis, O. (2013). Cloud services pricing models. Teoksessa *International Conference of Software Business* (s. 117-129). Berlin Heidelberg: Springer.

Lee, Y., & O'Connor, G. C. (2003). New product launch strategy for network effects products. *Journal of the Academy of Marketing Science*, 31(3), 241-255.

Lee, K., & Kang, K. C. (2010). Usage context as key driver for feature selection. Teoksessa *Software Product Lines: Going Beyond* (s. 32-46). Berlin Heidelberg: Springer.

Lehmann, D. W. I. S., & Buxmann, P. (2009). Pricing strategies of software vendors. *Business & Information Systems Engineering*, 1(6), 452-462.

Luoma, E., Rönkkö, M., & Tyrväinen, P. (2012). Current Software-as-a-Service Business Models: Evidence from Finland. Teoksessa *Software Business* (s. 181-194). Berlin Heidelberg: Springer.

Mannion, M., & Savolainen, J. (2010). Aligning business and technical strategies for software product lines. Teoksessa *Software Product Lines: Going Beyond* (s. 406- 419). Berlin Heidelberg: Springer.

Ma, D., & Kauffman, R. J. (2014). Competition between software-as-a-service vendors. *Engineering Management, IEEE Transactions on*, 61(4), 717-729.

Ma, D., & Seidmann, A. (2008). The pricing strategy analysis for the "Software-as-a-service" business model. Teoksessa *Grid Economics and Business Models* (s. 103-112). Berlin Heidelberg: Springer.

Mietzner, R., Leymann, F. & Unger, T. (2011). Horizontal and vertical combination of multi-tenancy patterns in service-oriented applications. *Enterprise Information Systems* 5 (1), 59–77.

Mietzner, R., Metzger, A., Leymann, F., & Pohl, K. (2009). Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications. Teoksessa *Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems* (s. 18-25). IEEE Computer Society.

Myers, M. D., & Newman, M. (2007). The qualitative interview in IS research: Examining the craft. *Information and organization*, *17*(1), 2-26.

Mäkilä, T., Järvi, A., Rönkkö, M., & Nissilä, J. (2010). How to Define Software-as-a-Service–An Empirical Study of Finnish SaaS Providers. Teoksessa *Software business* (s. 115-124). Berlin Heidelberg: Springer.

Nault, B. R., & Wei, X. D. (2005). Product differentiation and market segmentation of information goods. SSRN.

Niemelä, E. (2005). Strategies of product family architecture development. Teoksessa *Software Product Lines* (s. 186-197). Berlin Heidelberg: Springer.

Ojala, A. (2013). Software-as-a-Service Revenue Models. *IT Professional*, 15(3), 54-59.

Ojala, A. (2012). Software renting in the era of cloud computing. Teoksessa *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on* (s. 662-669). IEEE.

Pigou, A. C. (1932). The economics of welfare, 1920. London: McMillan&Co.

Pohl, K., Böckle, G., & van Der Linden, F. J. (2005). *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media.

Porter, M. E. (1980). Competitive strategy: Techniques for analyzing industries and competition.

Porter, M. E. (1985). Competitive advantage: creating and sustaining superior performance, 1985.

Pujol, N. (2010). Freemium: attributes of an emerging business model. Available at SSRN 1718663.

Raatikainen, M., Soininen, T., Männistö, T., & Mattila, A. (2003). A case study of two configurable software product families. Teoksessa *Software Product-Family Engineering* (s. 403-421). Berlin Heidelberg: Springer.

Rommes, E., & America, P. (2006). A scenario-based method for software product line architecting. Teoksessa *Software Product Lines* (s. 3-52). Berlin Heidelberg: Springer.

Shapiro, C. and Varian, H. (1998). Versioning: the smart way to sell information. *Harvard Business Review* (Nov-Dec), 106–118

Sharma, D., Aurum, A., & Paech, B. (2008, September). Business value through product line engineering-a case study. Teoksessa *Software Engineering and Advanced Applications, 2008. SEAA'08. 34th Euromicro Conference* (s. 167-174). IEEE.

Sun, B., Xie, J., & Cao, H. H. (2004). Product strategy for innovators in markets with network effects. *Marketing Science*, *23*(2), 243-254.

Sun, W., Zhang, X., Guo, C. J., Sun, P., & Su, H. (2008). Software as a service: Configuration and customization perspectives. Teoksessa *Congress on Services Part II*, 2008. SERVICES-2. (s. 18-25). IEEE.

Svahnberg, M., Van Gurp, J., & Bosch, J. (2005). A taxonomy of variability realization techniques. *Software: Practice and Experience*, 35(8), 705-754.

Tyrväinen, P., & Selin, J. (2011). How to sell SaaS: a model for main factors of marketing and selling software-as-a-service. Teoksessa *International Conference of Software Business* (s. 2-16). Berlin Heidelberg: Springer.

Varian, H. (1997). Versioning information goods.

Van Der Linden, F., Bosch, J., Kamsties, E., Känsälä, K., & Obbink, H. (2004). Software product family evaluation. Teoksessa *Software Product Lines* (s. 110-129). Berlin Heidelberg: Springer.

Wei, X., & Nault, B. R. (2008). Vertically differentiated information goods: Monopoly power through versioning. Julkaisematon, Haskayne School of Business, University of Calgary. Haettu 14.5.2016 osoitteesta http://ssrn.com/abstract=1677561.

Wu, S., Wortmann, H., & Tan, C. W. (2014). A pricing framework for software-as-a-service. Teoksessa *Innovative Computing Technology* (INTECH), 2014 Fourth International Conference on (s. 152-157). IEEE.

Xin, M., Levina, N. (2008). Software-as-a-service model: Elaborating client-side adoption factors. Teoksessa R. Boland, M. Limayem & B. Pentland (toim.), *Proceedings of the 29th International Conference on Information Systems*. Pariisi, Ranska.

Yin, R. K. (2013). *Case study research: Design and methods*. Sage publications.

LIITE 1 HAASTATTELURUNKO

Asiakkuudet

- Kuinka monta asiakasta tuotteella on?
- Minkä kokosia yrityksiä asiakkaissa on?
- Mille toimialoille tuote on suunnattu?

Arkkitehtuuri

- Single- vai multi -tenancy?
- Onko tietokanta asiakaskohtainen vai yhteinen?
- Onko arkkitehtuuri skaalautuva?
- Miten asiakaskohtaiset kustomoinnit on eriytetty koodin ja/tai versionhallinnan tasolla?

Versiointi ja hinnoittelu

- Mikä on tuotteen hinnoittelumalli?
- Onko tuotteesta erihintaisia versioita?
- Miten versiointi on toteutettu (downgrade, valittavat ominaisuudet)?
- Kuinka laajasti asiakaskohtaista kustomisointia tehdään?
- Millä perusteella versioiden ominaisuudet valittiin?