Massaräätälöinti ohjelmistotuotannossa kustannustehokkuuden näkökulmasta

Perttu Hallikainen

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Ohjaaja: Mikko Ruohonen
Toukokuu 2015

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Perttu Hallikainen: Massaräätälöinti ohjelmistotuotannossa kustannustehokkuuden näkökulmasta
Pro gradu -tutkielma, 77 sivua
Toukokuu 2015

Ohjelmistotuotanto on nopeasti kehittyvä ala, jossa täytyy pystyä vastaamaan globaalin kilpailun ja muuttuvien markkinoiden asettamiin haasteisiin hyvin nopeasti. Asiakkaat ovat tänä päivänä entistä hinta- ja laatutietoisempia ja vaativat entistä nopeampia tuotantoaikoja. Näin ollen ohjelmistokehitykseen on löydettävä entistä joustavampia ja tehokkaampia malleja. Perinteiset teollisuudessa ja palveluiden tuottamisessa käytettävät massaräätälöinnin menetelmät tarjoavat työkaluja, joita voidaan hyödyntää myös ohjelmistotuotannossa, jotta saavutettaisiin kustannussäästöjä ja lyhempiä tuotannon läpimenoaikoja. Tässä Pro gradu -tutkielmassa tarkastellaan, mitä massaräätälöinnin keinoja voidaan hyödyntää ohjelmistotuotannossa ja mitä vaatimuksia ne asettavat ohjelmistolle sekä tuotantoprosessille.

Avainsanat ja -sanonnat: Massaräätälöinti, ohjelmistotuotanto.

Sisällys

1. Johdanto

Nykyaikaiset liiketoimintamallit ja markkinakehitys vaativat yrityksiltä sopeutumista nopeasti muuttuvaan ympäristöön ja globaaliin kilpailuun. Tästä syystä ohjelmistotoimittajien asiakkaille on tärkeää saada järjestelmä ketteräsi käyttöön, jotta pystytään vastaamaan nopeasti muuttuvien markkinoiden tuomiin haasteisiin ja uusiin mahdollisuuksiin. Ohjelmistotuotantoprojektin kokonaiskustannukset ovat usein sidoksissa projektiin kuluvaan aikaan. Aikatauluvaatimukset projekteille ovat yleensä tiukat, koska tämän päivän talousmalli ja markkinatilanne vaativat sitä.

Asiakkaat ovat tänä päivänä myös hyvin kustannus- ja laatutietoisia, joten ohjelmisto täytyy pystyä toimittamaan kustannustehokkaasti siten, että ohjelmisto on laadultaan hyvä. Ohjelmistotuotannon asiakkaiden odotukset ja vaatimukset myös kasvavat koko ajan entisestään [Zakál *et al.,* 2011]. Ohjelmistoalan kehitys nojaakin tällä hetkellä vahvasti joustavien ja kustannustehokkaiden ratkaisumallien kehitykseen.

Massaräätälöinti on tehokas tapa pyrkiä vastaamaan tämän päivän ohjelmistotuotannon kustannustehokkuusvaatimuksiin. Massaräätälöinnin keinot mahdollistavat näihin vaatimuksiin vastaamisen hyödyntäen muun muassa modulaarisuutta, tuoteperhemallia ja tuotelinjoja [Zakál *et al.,* 2011]. Tehokas massaräätälöintiratkaisujen tuottaminen on kuitenkin haasteellista ja toimivan kokonaisuuden saavuttaminen vaatii monipuolista suunnittelua.

Tämän tutkielman tarkoituksena on pohtia, mitä erilaisia massaräätälöinnin keinoja voidaan hyödyntää ohjelmistotuotannossa paremman kustannustehokkuuden saavuttamiseksi ja millaisia vaatimuksia se asettaa tuotantoprosessille sekä itse tuotteelle. Kokonaisvaltaisen massaräätälöinnin toteuttaminen ei ole helppoa, joten tässä tutkielmassa pohditaan myös massaräätälöintiin liittyviä ongelmia ja mahdollisia kompastuskiviä, joita tulisi välttää.

Haasteellisuudesta huolimatta useat empiiriset ja teoreettiset tutkimukset osoittavat, että massaräätälöinnin keinoilla voidaan saavuttaa huomattavasti nopeammat tuotantoajat siten, että asiakaskohtainen räätälöinti on mahdollista [Piller, 2004]. Asiakaskohtainen räätälöinti on usein jopa välttämätöntä ja yleensä se tuo vähintään huomattavaa lisäarvoa ohjelmistotuotteelle.

Massaräätälöinnin keinot mahdollistavat myös globaalin toiminnan siten, että tuotantoyksiköt voivat toimia maantieteellisesti täysin erillään toisistaan. Yksi

tehokkaasti massaräätälöintiä jo pitkään hyödyntänyt yritys on esimerkiksi ruotsalainen telekommunikaatiojärjestelmiä valmistava Ericsson [Mathiassen and Sandberg, 2014].

Jotta pystytään tuottamaan tehokkaasti laadukkaita tuotteita massaräätälöinnin keinoin, myös tuotantoprosessin täytyy olla sellainen, että se vastaa asetettuja vaatimuksia. Massaräätälöinti ei siis kohdistu välttämättä pelkästään lopulliseen tuotteeseen tai palveluun, vaan sitä voidaan kohdistaa itse tuotantoprosessiin. Näin muun muassa juuri Ericssonin tuotantomalli on toteutettu [Mathiassen and Sandberg, 2014].

Ericssonin lisäksi myös useat muut yritykset ovat saavuttaneet kustannustehokkaamman tuotantoprosessin massaräätälöinnin keinoja hyödyntämällä. Maailman suurimmista ohjelmistotuottajista muun muassa Microsoft, IBM, SAP ja Oracle ovat saavuttaneet suurta hyötyä massaräätälöinnin keinoilla [Kumar, 2004].

Useat perinteiset massaräätälöinnin keinot ovat tuttuja ohjelmistokehityksen historian alkuajoilta asti. Vaikka samoja keinoja on periaatteessa käytetty jo vuosikymmeniä, on näistä menetelmistä vain harvoin puhuttu samoilla termeillä. Tässä tutkielmassa massaräätälöintiin perehdytään esittämällä liittymäkohdat perinteisen teollisuuden ja ohjelmistotuotannon välillä.

Tutkielma on jaettu rakenteellisesti siten, että toisessa luvussa esitellään tutkimusmenetelmät, tutkimuskysymykset sekä aikaisempaa aiheeseen liittyvää tutkimusta. Luvussa kolme perehdytään siihen, mitä massaräätälöinti yleisten määritelmien mukaan tarkoittaa ja kuinka massaräätälöintiä hyödynnetään teollisuudessa ja palveluiden tuottamisessa. Luvussa neljä käydään läpi yleisiä massaräätälöinnin menetelmiä, jotka ovat modulaarisuus, tuotealustat, tuoteperheet ja tuotelinjat.

Luvussa viisi käsitellään sitä, kuinka massaräätälöintiä voidaan hyödyntää nimenomaan ohjelmistotuotannossa ja kuinka luvussa kolme esitelty massaräätälöinnin kohdentaminen soveltuu ohjelmistotuotantoon. Lisäksi tässä luvussa perehdytään esimerkkiin massaräätälöidystä ohjelmistotuotteesta WordPress -sisällönhallintajärjestelmän avulla.

Luvussa kuusi käsitellään massaräätälöitävän ohjelmistoarkkitehtuurin suunnittelua. Arkkitehtuurin suunnittelussa pohditaan muun muassa, mitä vaatimuksia massaräätälöinti asettaa itse arkkitehtuurille ja mitä vaatimuksia se asettaa arkkitehtuurin suunnitteluprosessille. Arkkitehtuurin lisäksi käsitellään myös sitä, kuinka mallintamisen avulla voidaan tukea arkkitehtuuria ja sen suunnittelua. Luvussa pohditaan myös lyhyesti, kuinka nykyaikaiset ohjelmointikielet tukevat tällaista arkkitehtuurin toteutusta.

Luvussa seitsemän käydään läpi konstruktiivisten esimerkkien avulla, kuinka massaräätälöintiä voitaisiin kohdentaa suunniteltaessa kokonaisvaltaista HR-järjestelmää. Lisäksi luvussa esitetään myös esimerkkejä massaräätälöinnin mahdollistavan arkkitehtuurin suunnittelusta.

Luvussa kahdeksan käsitellään massaräätälöintiin kohdistuvaa kritiikkiä ja ongelmia sekä pohditaan yleisimpiä massaräätälöinnin käyttöönottoon liittyviä haasteita. Luvussa otetaan myös kantaa siihen, kuinka ongelmat voidaan mahdollisesti välttää.

Lopuksi luvussa yhdeksän esitetään yhteenveto ja johtopäätökset. Tässä luvussa kiteytetään saadut tulokset sekä pohditaan tulosten merkitystä. Lisäksi luvussa käsitellään myös sitä, kuinka tutkimusta voitaisiin laajentaa tai syventää myöhemmin.

2. Tutkimusmenetelmät ja aikaisemmat tutkimukset

2.1. Tutkimuskysymykset ja tutkimusmenetelmät

Tässä tutkielmassa tutkimuskysymys on kaksiosainen. Ensimmäisen tutkimuskysymys keskittyy massaräätälöinnin keinoihin ja toinen tutkimuskysymys massaräätälöinnin asettamiin vaatimuksiin.

1) Mitkä massaräätälöinnin keinot voivat parantaa ohjelmistotuotannon kustannustehokkuutta?

Tämän tutkimuskysymyksen avulla halutaan löytää sellaisia massaräätälöinnin keinoja, joita voidaan soveltaa ohjelmistotuotannossa siten, että ne parantavat ohjelmistotuotannon kustannustehokkuutta.

2) Millainen ohjelmiston ja tuotantoprosessin tulisi olla, jotta massaräätälöintiä voitaisiin hyödyntää?

Tämän tutkimuskysymyksen avulla halutaan selvittää, millainen itse kehitettävän ohjelmiston tulisi olla, jotta massaräätälöinnin keinojen hyödyntäminen olisi mahdollista. Tämän lisäksi halutaan myös selvittää, että millaisilla työkaluilla ja menetelmillä tuotantoprosessia voidaan tukea siten, että massaräätälöinnin tekeminen on mahdollista.

Tutkimusmenetelminä tässä tutkielmassa käytetään kirjallisuuskatsausta ja käsiteanalyysiä. Olen kerännyt ja käynyt läpi aiheeseen liittyvää kirjallisuutta ja tutkimusmateriaalia, joiden pohjalta kirjallisuuskatsauksen ja käsiteanalyysin kautta avataan yleinen massaräätälöinnin määritelmä ja massaräätälöintiin käytettävät menetelmät. Suurin osa tutkielman lähdemateriaalista on tuotettu viimeisen kymmenen vuoden sisällä.

Yleisen määrittelyn ja massaräätälöinnin menetelmien jälkeen tutkielmassa esitetään teollisuudessa käytettyjen massaräätälöinnin keinojen liittymät ohjelmistotuotantoon. Ohjelmistotuotannon osalta kirjallisuus ja tutkimusmateriaali keskittyvät massaräätälöintiin ohjelmistotuotannossa sekä ohjelmistojen arkkitehtuurien suunnitteluun. Materiaaleissa käsitellään myös ohjelmistojen mallintamista ja muita menetelmiä, joilla voidaan tukea massaräätälöintiä ohjelmistotuotannossa.

Tutkielmassa esitetään myös esimerkkejä siitä, miten massaräätälöintiä voidaan hyödyntää tosielämän järjestelmäkehityksessä. Näitä esimerkkejä tarkastellaan kuvitteellisen HR -järjestelmän avulla. Kyseessä ei kuitenkaan ole varsinainen

tapaustutkimus, vaan tarkoitus on havainnollistaa kirjallisuuskatsauksessa esitettyä teoriaa esimerkkien kautta. Esimerkkien toteutuksessa on hyödynnetty konstruktiivista tutkimusmenetelmää [Piirainen and Gonzalez, 2013].

Konstruktiivisen menetelmän tarkoituksena on pyrkiä esittämään, kuinka kuvitteellisen HR -järjestelmän toteutuksessa voitaisiin hyödyntää massaräätälöinnin keinoja tosielämässä. Konstruktiivinen osio linkitetään vahvasti käsiteanalyysin ja kirjallisuuskatsauksen kautta esitettyyn teoriaan. Lopputuloksena esitetään konstruktioita, jotka toimivat esimerkkeinä siitä, kuinka massaräätälöinnin keinoja voitaisiin hyödyntää HR -järjestelmän toteutuksessa.

2.2. Aikaisemmat tutkimukset

Massaräätälöintiä on tutkittu paljon teollisuuden osalta viime vuosikymmeninä. Massaräätälöinnin termin teki tunnetuksi varsinkin Pine *et al.* kirjallisuudellaan 1990-luvun alkupuolella. Myös Frank Piller on tuottanut paljon kirjallisuutta liittyen massaräätälöintiin ja personointiin. Muun muassa tämän kirjallisuuden pohjalta on tuotettu melko paljon tutkimusta liittyen erilaisten tuotteiden ja palveluiden massaräätälöintiin.

Tässä tutkielmassa tullaan viittaamaan massaräätälöinnin menetelmiä käsiteltäessä paljon Frank Pillerin ja Mitchell Tsengin kirjaan *Handbook of Research in Mass Customization and Personalization*. Tähän kirjaan on koottu useiden eri toimijoiden toteuttamia tutkimuksia massaräätälöintiin liittyen.

Tietojärjestelmien ja ohjelmistotuotannon kohdalla massaräätälöintiä ei ole tutkittu yhtä paljon kuin teollisuudessa. Massaräätälöinnin hyödyntämistä ohjelmistotuotannossa on kuitenkin alettu kasvavissa määrin tutkia viime vuosikymmeninä. Vaikka massaräätälöintiä terminä ei ole käytetty ohjelmistotuotannon historian alusta lähtien, on useita massaräätälöinnin piirteitä hyödynnetty ohjelmistotuotannossa jo melko pitkään.

Massaräätälöintiä on tutkittu myös Suomessa ja tässä tutkielmassa massaräätälöinnin yleinen määrittely perustuu pitkälti Ahoniemen *et al.* [2007] kirjaan *Massaräätälöinnillä kilpailukykyä*. Massaräätälöintiä on tutkittu myös Tampereen yliopistossa ja massaräätälöintiin liittyen on viime vuosina toteutettu tietojenkäsittelyopista kaksi Pro gradu -tutkielmaa. Antti Sand [2012] on julkaisut joulukuussa 2012 Pro gradu -tutkielman aiheesta *Ohjelmistotuotannon massaräätälöinnistä tietojärjestelmätuotannossa*. Pasi Paunu [2014] on julkaissut kesäkuussa 2014 suunnittelukonfiguraattoreihin liittyvän Pro gradu -tutkielman aiheesta

Design Configurator – Managing the Order Engineering Challenge in ETO Companies. Kyseisissä tutkielmissa hyödynnettyjä lähdemateriaaleja on käytetty osittain myös tässä tutkielmassa.

3. Massaräätälöinti yleisesti

3.1. Massaräätälöinnin määrittely

Massaräätälöinti ei itsessään ole kovinkaan uusi keksintö, sillä massaräätälöinnin perusajatusta on hyödynnetty jo satoja vuosia. Tämä perusajatus on erilaisten tuotteiden ja palveluiden tuottaminen yksilöllisiin tarpeisiin sarjatuotantona ja kustannustehokkaasti [Ahoniemi *et al.,* 2007]. Jo satojen vuosien takaa löytyy tuotantomalleja, joissa on pyritty yhdistämään massatuotannon tehokkuus ja asiakaskohtaisen räätälöinnin tuoma arvo.

Massatuotanto on jo pitkään ollut tunnetuin teollisuuden tuotantotapa. Massatuotannon perusajatuksena on pyrkiä tuottamaan standardoitua tuotetta mahdollisimman tehokkaasti mahdollisimman pienillä tuotantokustannuksilla. Tuotetta pyritään yleensä tuottamaan varastoon ennustetun menekin mukaan. Erilaisten tuotevariaatioiden määrä pyritään pitämään mahdollisimman alhaisena ja samalla pyritään pitämään tuotantoprosessi mahdollisimman vakiona tuotantovolyymin ja alhaisten kustannusten varmistamiseksi. [Ahoniemi *et al.,* 2007]

Massatuotanto sopii erittäin hyvin tuotantoon, jossa voidaan helposti ennustaa tuotteen tai palvelun menekki. Tuotteiden kohdalla varastoinnin tulee olla helppoa ja tarve erilaisille tuotevariaatioille tai räätälöinnille tulee olla hyvin pieni. Tällöin massatuotannosta pystytään saamaan maksimaalinen hyöty.

Yksi ehkä tunnetuimpia massatuotannon esimerkkejä on Henry Fordin kehittämä liukuhihnamenetelmä Fordin T-mallin autotuotantoon. Vaikka tuotannossa ei ollut juurikaan mukana automaatiota, on nimenomaan tämä malli luonut pohjaa myöhemmin teollisuudessa käytetyille pitkälle automatisoiduille massatuotantomalleille.

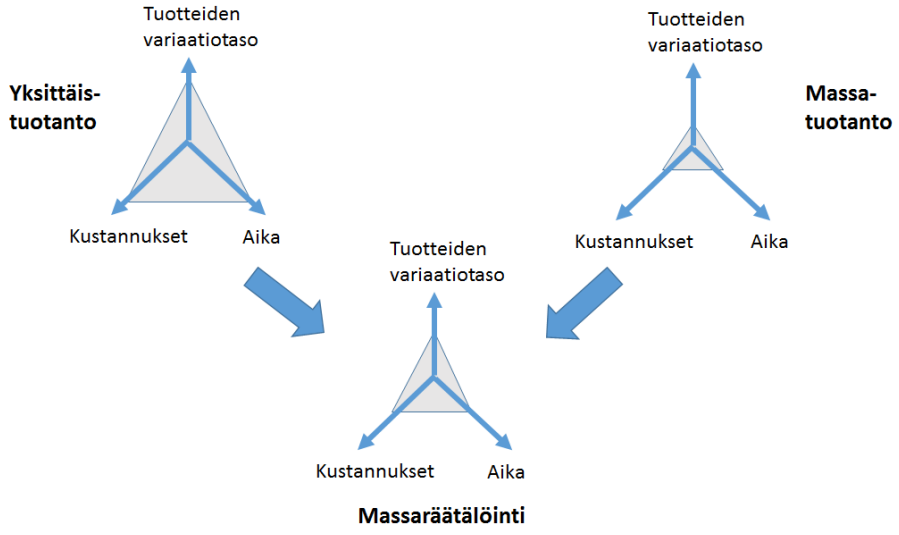
Massatuotannon vastakohtana voidaan nähdä uniikki asiakaskohtainen räätälöity tuotanto. Tällä tuotantotavalla pyritään tuottamaan yksilökohtaiset tarpeet täyttäviä tuotteita ja palveluita ilman, että tuotantoprosessi on vakioitu. [Ahoniemi *et al.,* 2007]. Asiakaskohtaisella uniikilla tuotannolla pyritään ainutlaatuisuuteen, joka tuo tarvittavan kilpailukyvyn. Tämä malli kuitenkin edellyttää usein korkeita tuotantohintoja ja pitkää tuotantoaikaa.

Asiakaskohtaisille uniikeille tuotteille on aina kysyntää, mutta tuotantovolyymi harvoin pystyy kattamaan tällaiselle tuotannolle aiheutuvat kustannuspaineet. Esimerkiksi tuotantoprosessin asiakaskohtainen muunteleminen ei voi olla kovinkaan tehokasta, sillä tuotantoprosessin muunteleminen aiheuttaa aina läpimenoajan kasvua ja kustannusten nousua [Ahoniemi *et al.,* 2007].

Massaräätälöinti on massatuotannon ja asiakaskohtaisen räätälöinnin yhdistelmä, jonka avulla pyritään tarjoamaan asiakkaille yksilöllisiä räätälöityjä ratkaisuja siten, että tuotantoprosessi sisältää massatuotannon tehokkuuden. Asiakkaille voidaan siis tuottaa räätälöityjä ratkaisuja hyödyntäen massatuotannon tuomia suurtuotannon etuja [Merle *et al.*, 2010].

Suurin suurtuotannosta saatava etu on yleensä se, että tuotannon yksikkökustannukset laskevat tuotantomäärän kasvaessa. Tämä yleensä vaatii vakioidun, mutta kuitenkin joustavan tuotantoprosessin ja organisaatiorakenteen [Ahoniemi *et al.*, 2007]. Laajemmin voidaan mieltää, että massaräätälöinti on hyödykkeiden tai palveluiden kehittämistä, markkinointia, tuottamista ja toimittamista monenlaisilla variaatiomahdollisuuksilla lähes massatuotannon kustannustehokkuudella [Piller and Tseng, 2010]. Massaräätälöinti ei siis koske välttämättä pelkästään lopullista tuotetta tai palvelua, vaan massaräätälöinti voi vaikuttaa koko tuotantoprosessiin.

Palveluita tai tuotteita toimittavalle yritykselle on yleensä toimialasta riippumatta tärkeää saavuttaa kilpailuetu neljällä eri osa-alueella, jotka ovat joustavuus, laatu, hinta ja tuotantoaika [Alfnes and Skjelstad, 2010]. Juuri näihin kilpailuetua tuottaviin osa-alueisiin voidaan vaikuttaa massaräätälöinnin avulla. Massatuotannon ja asiakaskohtaisen tuotannon yhdistämisestä saatava hyöty on esitetty kuvassa 1.



Kuva 1. Massatuotannon tehokkuuden ja yksittäistuotannon tuotevariaatiotason yhdistyminen massaräätälöinnissä [Ahoniemi et al., 2007].

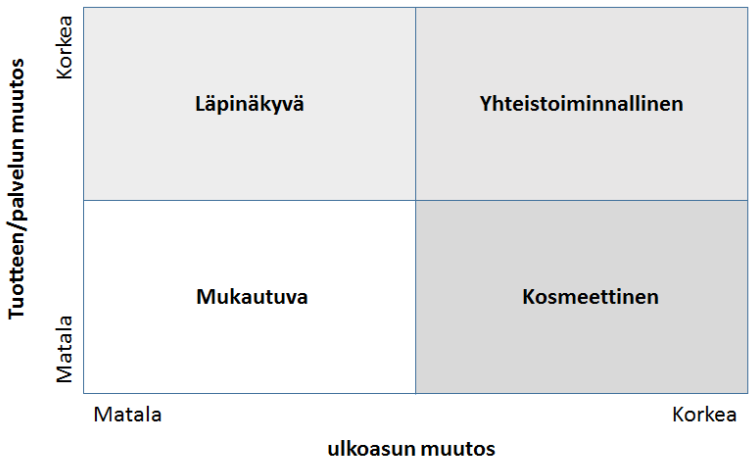
Länsimaissa, kuten Suomessa, missä palkka- ja tuotantokustannukset ovat korkeita, massaräätälöinnin keinoilla voidaan aikaansaada huomattavaa kustannustehokkuuden parannusta verrattuna perinteisiin massatuotannon prosesseihin tai asiakaskohtaisiin tuotantomalleihin. Tärkeimmät kustannussäästöjä tuovat ja kilpailukykyä kasvattavat tekijät ovat useimmiten jo pelkästään toimitusaikojen lyhentyminen ja tuotantoprosessin tehostuminen. Vaikka toimitusajat lyhenevät ja prosessi tehostuu, voidaan asiakaskohtaisen räätälöinnin tuoma arvo ja kilpailukyky säilyttää [Ahoniemi *et al.*, 2007]. Kun tuotteita tai palveluita voidaan kohdentaa entistä tarkemmin asiakaskohtaisesti ilman tuotantokustannusten merkittävää nousua, luo tämä liiketoiminnan näkökulmasta hyvin merkittävän kilpailukykyaseman [Piller and Tseng, 2010].

Läpimenoaikojen lyhentymisen ja tuotantokustannusten alentumisen lisäksi massaräätälöinnin avulla pyritään myös parempaan tuotannon laatuun sekä asiakkaiden sitouttamiseen [Alfnes and Skjelstad, 2010]. Asiakkaat ovat tänä päivänä kaikilla toimialaloilla hyvin laatutietoisia, joten tuotteen tai palvelun laatu täytyy lähtökohtaisesti olla hyvä. Massaräätälöinnillä voidaan pyrkiä parempaan laatuun esimerkiksi siten, että käytetään jo aikaisemmin muissa yhteyksissä toimiviksi todettuja resursseja ja komponentteja uudelleen.

Asiakkaan sitouttaminen pitkään asiakassuhteeseen on myös tärkeää. Asiakkaan mukaan ottaminen räätälöintiprosessiin yleensä sitouttaa asiakasta myös jatkossa pysymään asiakkaana, jolloin saavutetaan parempi asiakasuskollisuus. Pitkät asiakassuhteet ovat tärkeitä, koska suurin osa voitosta tehdään yleensä vasta myöhemmissä vaiheissa asiakassuhteita, ei heti asiakassuhteen alussa.

3.2. Massaräätälöinnin kohdentaminen

Massaräätälöinnin kohdentaminen voi vaihdella saman tuotteen ja tuotantoprosessin sisällä asiakaskohtaisesti. Massaräätälöinti voidaan jakaa neljään eri alueeseen sen mukaan, miten räätälöinti vaikuttaa tuotteen ominaisuuksiin tai ulkoasuun. Nämä neljä osa-aluetta ovat mukautuva, läpinäkyvä, yhteistoiminnallinen ja kosmeettinen massaräätälöinti. [Ahoniemi *et al.*, 2007]. Massaräätälöinnin neljä osa-aluetta on esitetty kuvassa 2. Näitä neljää luokittelun osaa voidaan soveltaa myös tuotantoprosessin eri osissa tai esimerkiksi asiakassuhteiden tai asiakasvuorovaikutuksen hallinnassa.



Kuva 2. Massaräätälöinnin kohdentamisen vaihtoehdot. [Ahoniemi et al., 2007].

Mukautuva massaräätälöinti tarkoittaa mallia, jossa asiakas voi itse suunnitella ja päättää, minkälaisista moduuleista tai osakokonaisuuksista lopputuote koostuu [Ahoniemi *et al,* 2007]. Asiakkaalle siis esitetään mahdollisuus kaikista tuotevariaatioista, joita eri komponentteja yhdistelemällä voidaan saada aikaan. Esimerkiksi tuotteen jakaminen selkeisiin toisistaan riippumattomiin moduuleihin mahdollistaa sen, että asiakas voi saada käyttöönsä vain ne ominaisuudet, jotka hän tarvitsee. Toisaalta asiakas voi saada helposti käyttöönsä moduuleita, jotka eivät välttämättä ole kriittisiä, mutta tuovat kuitenkin asiakkaalle hyötyä. Jotta lopullisen kokoonpanon toteuttaminen olisi tehokasta, täytyy eri moduulien olla mahdollisimman riippumattomia toisistaan. Toisaalta eri moduulien väliset liittymät tulee olla standardoidut ja helpot toteuttaa. Tällaisella mallilla lopullinen kokoonpano on tehokasta toteuttaa, jolloin pystytään lyhentämään tuotantoaikaa sekä säästämään tuotantokustannuksissa. Tuote halutaan tehdä siis mahdollisimman helposti muokattavaksi. Modulaarisuutta käsitellään tarkemmin luvussa 4.1.

Kosmeettinen massaräätälöinti on osa-alue, jolla voidaan vaikuttaa hyvin paljon saman tuotteen tai palvelun yksilöllisyyteen eri asiakkaiden kohdalla [Ahoniemi *et al,* 2007]. Kosmeettisella massaräätälöinnillä voidaan esimerkiksi vaikuttaa tuotteen ulkonäköön ja muihin ulkoisiin ominaisuuksiin ilman, että tarvitsee muokata itse tuotteen ominaisuuksia tai rakennetta. Tämä mahdollistaa sen, että tuotteelle voidaan hakea yksilöllisyyden tuomaa kilpailukykyä, mikä voi myös nostaa tuotteen arvoa asiakkaan silmissä. Tämä pyritään tekemään ilman, että tuotantokustannukset tai toimitusaika nousevat merkittävästi.

Läpinäkyvä massaräätälöinti ei ole suoraan asiakkaalle näkyvää räätälöintiä, vaan se tapahtuu keräämällä erilaisista kanavista asiakastietoa, jota käytetään palvelun tai

tuotteen räätälöinnissä [Ahoniemi *et al,* 2007]. Tätä tietoa analysoimalla pystytään ennustamaan ja päättelemään asiakastarpeiden muutoksia paremmin ja pystytään jopa löytämään uusia tarpeita. Tietoja analysoimalla voidaan jopa löytää kokonaan uusia tuotteen tai palvelun hyödyntämismahdollisuuksia. Läpinäkyvää massaräätälöintiä hyödyntämällä voidaan parantaa kustannustehokkuutta muun muassa kohdentamalla tuote tai palvelu asiakkaille paremmin. Erilaiset verkkotyökalut mahdollistavat tänä päivänä melko helpon tavan tiedon keräämiseen erilaisilta kohderyhmiltä. Läpinäkyvän tiedonkeruun lisäksi massaräätälöintiä voidaan toteuttaa myös suoraan asiakkailta saadun tiedon perusteella [Stotko and Snow, 2010]. Suoraan asiakkailta tuleva palaute ja tieto ovat yleensä ensiarvoisen tärkeitä oman liiketoiminnan kehittämisen kannalta.

Yhteistoiminnallinen massaräätälöinti on kustannuksiltaan korkein ja haastavin massaräätälöinnin muoto [Ahoniemi *et al,* 2007]. Yhteistoiminnallinen massaräätälöinti tarkoittaa käytännössä asiakkaan kanssa tehtävää yhteistyötä lopputuotteen tuottamiseksi ja usein tämä sisältää ainakin asiakkaan kanssa tehtävän ominaisuuksien määrittelyn. Yhteistoiminnallista massaräätälöintiä tarvitaan usein varsinkin projekteissa, joissa asiakas ei pysty kovin hyvin määrittelemään, millainen lopputuloksen täytyy olla. Markkina-alueesta riippuen voi olla jopa yleistä, että asiakas ei tiedä täysin, millaisen lopputuloksen haluaa [Piller and Tseng, 2010]. Monimutkaisten tuotemoduulien yhdistelemisessä ja suunnittelussa tarvitaankin usein asiakkaan ja toimittajan yhteistyötä.

Yhteistoiminnallisuudella on tärkeä osa kestävän asiakassuhteen luomisessa, koska se tekee asiakkaasta osan suunnittelu- ja toteutusprosessia, mikä taas sitouttaa asiakasta [Piller, 2004]. Kynnys vaihtaa toimittajaa on asiakkaalle huomattavasti korkeampi, kun asiakas on itse ollut osa ratkaisun toteuttamista [Piller, 2004]. Yhteistoiminnallisuus onkin massaräätälöinnissä yksi tärkeimpiä osa-alueita, mikä erottaa sen massatuotannosta.

Yhteistoiminnallisen massaräätälöinnin yhteys suoranaiseen kustannustehokkuuden paranemiseen ei välttämättä ole kovin yksiselitteinen. Yhteistoiminnallinen massaräätälöinti on kuitenkin monissa tapauksissa välttämätöntä, jotta päästään laadullisesti riittävään lopputulokseen. Yhteistoiminnallisella massaräätälöinnillä voidaan varmistaa asiakkaan ja toimittajan yhtenevä mielikuva siitä, millainen lopputuloksen tulee olla. Useat massaräätälöivät tuotteet voivat olla jopa niin monimutkaisia, että asiakas ei välttämättä pysty ilman apua suunnittelemaan lopullista tuotetta [Piller and Tseng, 2010]. Tällöin tuotteen lopullinen kokoonpano on suunniteltava yhdessä toimittajan ja asiakkaan kesken. Asiakkaan osallistuminen

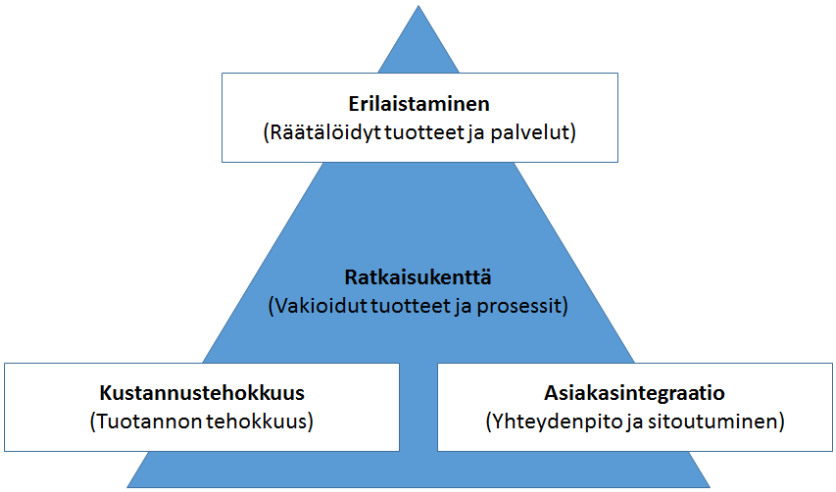
projektiin voi myös tuoda usein asiakkaalle lisäarvoa, joka voi käytännössä kasvattaa asiakkaan maksuhalukkuutta [Merle *et al*,. 2010].

Nämä neljä erilaista massaräätälöinnin osa-aluetta eivät ole toisiaan poissulkevia, vaan niitä voidaan käyttää rinnakkain tai prosessin eri vaiheissa. Näitä neljää massaräätälöinnin tapaa voidaan myös soveltaa saman tuotteen tai palvelun kohdalla eri tavalla eri asiakkaille. [Ahoniemi *et al,* 2007]

Kun massaräätälöinnin keinot on hyvin luokiteltu, voidaan räätälöintitarpeet tehokkaasti kohdentaa sille osa-alueelle, joka kulloinkin kyseessä olevalle asiakkaalle on tärkeää. Massaräätälöinnin kohdentamisella pystytään myös tehostamaan tuotantoprosessia ja tuotannon läpimenoaikaa sekä lisäämään yksilöllisen räätälöinnin tuomaa arvoa.

3.3. Massaräätälöinnin kohdentaminen

Toimivan massaräätälöintimallin luominen edellyttää sellaisen kokonaisvaltaisen ratkaisukentän saavuttamista, jonka avulla yritys pystyy vastaamaan hyvinkin erilaisten asiakkaiden tarpeisiin [Ahoniemi *et al.*, 2007]. Tällainen Frank Pillerin [2005] esittämä ratkaisukenttä on esitetty kuvassa 3.



Kuva 3. Massaräätälöinnin edellytykset [Ahoniemi et al., 2007].

Toimivan massaräätälöintimallin saavuttaminen edellyttää kustannustehokkuuden hallintaa, tehokasta tuotteiden ja palveluiden erilaistamista sekä tehokasta asiakasvuorovaikutusta ja sitouttamista [Ahoniemi *et al.*, 2007]. Ratkaisukentän ja sen osa-alueiden kehittäminen on aina prosessi- ja yrityskohtaista. Kuitenkin löytämällä sopiva tasapaino ratkaisukentän osa-alueiden välille on mahdollista toteuttaa malli, jossa räätälöinnistä aiheutuvat kustannukset säilyvät riittävän alhaisina ja tuotannon läpimenoaika riittävän nopeana.

Ratkaisukentän muodostumiseen vaikuttaa se, kuinka paljon räätälöinnin tarve vaikuttaa tuotantokustannuksiin ja tuotantoaikaan. Jokaisen yrityksen täytyy analysoida omakohtaisesti, millaisten variaatiomahdollisuuksien tuottaminen on kannattavaa, jotta tuotanto säilyy riittävän tehokkaana ja tuotantokustannukset riittävän alhaisina. On siis kyettävä arvioimaan, kuinka suuret räätälöinnistä aiheutuvat kustannukset asiakkaat ovat valmiita maksamaan ja kuinka pitkät tuotantoajat asiakkaat ovat valmiita hyväksymään. [Ahoniemi *et al*., 2007]

Vakaalla ratkaisukentällä halutaan toteuttaa joustava ympäristö, jossa voidaan tuottaa tehokkaasti erilaisia räätälöityjä ratkaisuja [Piller, 2004]. Ratkaisukentän vakaus tarkoittaa sitä, että ratkaisuja voidaan tuottaa tehokkaasti erilaisiin tarpeisiin, mutta vaihtoehtoja ei kuitenkaan ole loputtomasti. Erilaisia ratkaisuja voi olla useita, mutta ratkaisujen määrä täytyy kuitenkin olla rajallinen [Piller, 2004].

Liiketoiminnan näkökulmasta massaräätälöinnin prosessi voidaan jakaa kolmeen resurssiin, jotka ovat ratkaisukentän kehittäminen (Solution Space Development), kestävä arvoketjun suunnittelu (Robust Value Chain Design) ja valinnan yksinkertaistus (Choice Simplification) [Piller and Tseng, 2010]. Yrityksen täytyy tunnistaa siis oma markkina-alueensa ja asiakkaidensa tarpeet. Yrityksen on kartoitettava, millaisilla tuote- tai palvelukokonaisuuksilla pystytään täyttämään asiakkaan tarpeet suhteessa siihen, kuinka paljon asiakas on valmis maksamaan [Piller and Tseng, 2010]. Tämän pohjalta voidaan lähteä suunnittelemaan ratkaisukenttää siten, että tuotannosta ei tule liian monimutkaista ja tuotantokustannukset eivät nouse liian korkeaksi.

Kestävän arvoketjun suunnittelu tarkoittaa sitä, että tuotantomallit ja prosessit pidetään riittävän tehokkaina. On pidettävä huolta siitä, että esimerkiksi valtava asiakasvaatimusten määrä ei pääse tekemään tuotantoprosessista tai tuotantovariaatioista liian monimutkaisia, jotta tuotannon kustannustehokkuus ei laske [Piller and Tseng, 2010]. Tärkeä osa tuotantoprosessin tehokkuutta on esimerkiksi jo toteutettujen komponenttien ja resurssien uudelleenhyödyntäminen ja yhdisteleminen tehokkaasti ilman, että uuden vaatimuksen takia täytyy tehdä uutta kehitystyötä prosessiin tai tuotteeseen. Kestävän arvoketjun avulla yritys voi tuottaa räätälöityjä palveluita ja tuotteita lähes massatuotannon tehokkuudella [Piller and Tseng, 2010].

Valinnan yksinkertaistus tarkoittaa sitä, että tuetaan asiakasta heidän omien ongelmiensa ja ratkaisumahdollisuuksiensa määrittelemisessä ja sitä kautta ratkaisun löytämisessä [Piller and Tseng, 2010]. Lähtökohtana tälle toimii se, että pyritään pitämään monimutkaisuus mahdollisimman vähäisenä ja vähentämään valinnanvaikeutta

sekä hämmennystä, jota asiakas voi kohdata. Liian suuri valintamahdollisuuksien määrä voi aiheuttaa asiakkaalle enemmän arvon laskua kuin arvon nousua [Piller, 2004]. Mikäli kokonaisuudet ovat hyvin monimutkaisia ja asiakas saa vain vähän apua ratkaisun suunnittelussa, voi asiakkaalle tulla mielikuva, että toimittajan kanssa on vaikeaa asioida. Tämän välttämiseksi on pidettävä monimutkaiset valintamahdollisuudet vähäisinä, suunniteltava hyvät työkalut valintojen tekemiseen sekä tuettava asiakasta valintojen teossa [Piller and Tseng, 2010].

3.4. Asiakassuhteen hallinta

Monilla aloilla pitkät asiakassuhteet ovat jopa elintärkeä menestystekijä yrityksen liiketoiminnan kannalta [Siems and Walcher, 2010]. Massaräätälöinnin keinoilla pystytään tehokkaasti tukemaan pitkäkestoisten asiakassuhteiden asettamia tarpeita. Pitkät asiakassuhteet mahdollistavat yhteisten toimintatapojen muodostumisen ja yhteisen verkostoitumisen. Pitkillä asiakassuhteilla voidaan myös saavuttaa käyttöönottoprosessien yksinkertaistumista ja olemassa olevien integraatioiden hyödyntämistä. Tämä tuo lisäarvoa asiakkaalle ja mahdollistaa tehokkaamman tuotannon [Siems and Walcher, 2010]. Toimivalla asiakashallintaratkaisulla voidaan pitää huolta siitä, että asiakassuhde kestää myös tulevaisuudessa.

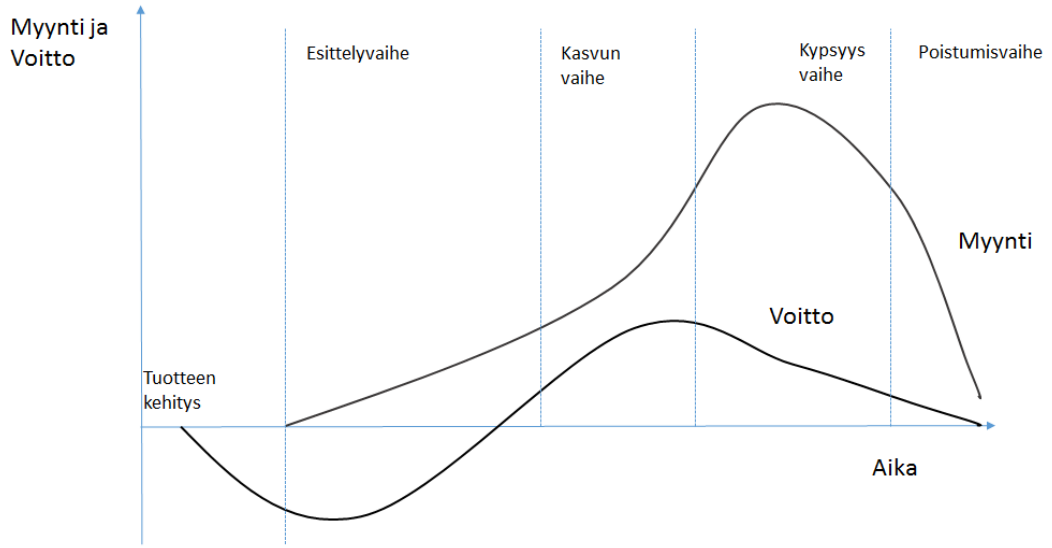
Kokonaisvaltainen asiakassuhteen hallinta koko tuotteen tai palvelun elinkaaren ajan on erittäin tärkeää monilla toimialoilla [Siems and Walcher, 2010]. Uusasiakashankinnan lisäksi monille yrityksille on tärkeää pitää kiinni olemassa olevista asiakassuhteistaan. Monet yritykset hyödyntävätkin erilaisia asiakastietojärjestelmäratkaisuja olemassa olevien asiakassuhteidensa hallinnassa [Siems and Walcher, 2010].

Yksi tehokas tapa pitää kiinni asiakassuhteista on verkostoitua muiden toimialaan liittyvien tekijöiden kanssa ja tehdä tuote helposti integroituvaksi muiden toimialan tuotteiden kanssa [Siems and Walcher, 2010]. Tällainen verkostoituminen mahdollistaa sen, että asiakkaalle voidaan suositella ennalta määriteltyjä tuotteita tai palveluita sellaisiin tarpeisiin, joihin oma tuote tai palvelu ei täysin pysty vastaamaan. Mietitään esimerkiksi tilannetta, jossa yritys A ei voi täyttää kaikkia asiakkaan tarpeita. Yritys A voi myydä oman tuotteensa asiakkaalle ja suositella tähän integroitavaksi yrityksen B tuotetta. Vastavuoroisesti toisen asiakkaan kohdalla yritys B voi suositella yritystä A vastaavassa tilanteessa. Tällaisella toimintamallilla voidaan mahdollistaa asiakkaan

pysyminen oman verkoston piirissä eikä asiakasta menetetä kilpailijalle [Siems and Walcher, 2010].

Pitkissä asiakassuhteissa on hyvin tavallista, että asiakkaiden tarpeet ja vaatimukset muuttuvat asiakassuhteen elinkaaren aikana. Ei välttämättä ole järkevää pyrkiä vastaamaan asiakkaan aivan kaikkiin muuttuviin tarpeisiin tai vaatimuksiin, vaan ratkaisu voidaan pyrkiä löytämään toiselta toimittajalta oman verkoston piiristä. Jotta tämä malli olisi toimiva, on oman tuotteen tai palvelun oltava joustava siten, että se voi toimia yhdessä muiden toimijoiden tuotteiden tai palveluiden kanssa.

On yleistä, että pitkien asiakassuhteiden alkuvaiheessa tuotantokustannukset ovat suuret ja vastaavasti voitot ovat pienet johtuen esimerkiksi alkuvaiheen tarjouskilpailusta [Siems and Walcher, 2010]. Myöhemmissä asiakassuhteen vaiheissa yleensä tuotantokustannukset, ylläpitokustannukset ja käyttöönottokustannukset ovat pienemmät ja asiakas on usein maksuvalmiimpi. Tätä asiakassuhteen kestoa ja tuotteen elinkaarta on kuvattu kuvassa 4. Kuvasta käy ilmi, mikä pitkäkestoisen ja toimivan asiakassuhteen hyöty on kustannustehokkuuden kannalta. Suurin kustannustehokkuuden hyöty saavutetaan usein vasta asiakassuhteen myöhemmissä vaiheissa.



Kuva 4. Perinteinen tuotteen elinkaari [Siems and Walcher, 2010].

Pitkissä asiakassuhteissa tärkeimpiä osa-alueita asiakassuhteen säilyttämiseksi ovat erilaiset ylläpitopalvelut asiakkaan käytössä oleville tuotteille [Siems and Walcher, 2010]. Esimerkiksi autoteollisuudessa tuotantoprosessi voi sisältää useita erilaisia massaräätälöinnin piirteitä ja tämän lisäksi asiakassuhteen ylläpitoon voi kuulua erilaiset autojen huolto- ja korjaussopimukset. Ylläpitopalveluiden lisäksi tuotteiden tai palveluiden käyttöön voi kuulua tai olla ostettavissa erilaisia konsultointipalveluita.

Mikäli asiakas kokee tuotteen ja siihen liittyvät palvelut toimiviksi, on tämä omiaan sitouttamaan asiakasta pitkään asiakassuhteeseen.

Massaräätälöityjen tuotteiden ja palveluiden kohdalla on myös tärkeää pyrkiä rohkaisemaan asiakkaita itse suunnittelemaan ja kokoonpanemaan räätälöityjä ratkaisuja [Piller, 2004]. Tällöin asiakas voi koko asiakassuhteen elinkaaren ajan itse miettiä, kuinka räätälöidä tai yhdistellä tuotekokonaisuuksia siten, että se tuottaa hyötyä asiakkaan muuttuvilla markkinoilla. Kun asiakas itse osallistuu suunnitteluun ja toteutukseen, tuottaa se yleensä arvoa asiakkaalle ja näin asiakas voidaan sitouttaa pitkiin asiakassuhteisiin [Piller, 2004].

Monien tuotteiden ja palveluiden kohdalla ei ole ymmärretty riittävän kommunikaation ja palveluiden merkitystä pitkille asiakassuhteille [Siems and Walcher, 2010]. On hyvin yleistä, että asiakkaan tarpeet muuttuvat asiakassuhteen aikana, joten tätä on hyvä pyrkiä ennakoimaan. On tärkeää pyrkiä analysoimaan sitä, kuinka asiakkaan tarpeet mahdollisesti tulevat muuttumaan asiakassuhteen myöhemmissä vaiheissa. Tämä tulee myös ottaa huomioon tuotteen tai palvelun suunnittelussa ja tuotannossa. Tuotteen koostuminen yksittäisistä moduuleista on tehokas keino taata tuotteen joustavuus ja ennakoida tulevia muutoksia. Tuotteiden modulaarisuutta käsitellään tarkemmin seuraavassa luvussa.

4. Massaräätälöinnin menetelmät

4.1. Modulaarisuus

Tuotteiden tai palveluiden modulaarisuus on tärkeä lähtökohta massaräätälöinnille. Modulaarisuutta voidaan pitää massaräätälöinnin kulmakivenä, sillä se tarjoaa aidon mahdollisuuden toteuttaa räätälöintiä tehokkaasti [Kumar, 2004]. Räätälöinnin toteutuminen on tärkeää, sillä asiakas pitää lähestulkoon aina räätälöityä tuotetta tai palvelua parempana kuin parasta tarjolla olevaa standardoitua ratkaisua [Piller, 2004].

Modulaarisuudella tarkoitetaan sitä, että tuote tai palvelu voidaan jakaa itsenäisiin erillisiin komponentteihin tai moduuleihin, joista jokaisella on oma uniikki tehtävänsä. Tavoitteena on myös tehdä jako niin, että komponentit eivät ole riippuvaisia toisistaan. Komponentit pyritään toteuttamaan yhtenäisesti ja standardoidusti siten, että komponentteja voidaan yhdistellä toisiinsa. Modulaarista jakoa voidaan tehdä eri tasoilla, kuten komponentteihin, moduuleihin tai alijärjestelmiin. Tässä tutkielmassa komponentilla viitataan yleisesti ottaen tuotteen osaan, moduulilla viitataan tuotteen osakokonaisuuteen ja alijärjestelmällä osaan järjestelmäkokonaisuutta.

Modulaarisuuden saavuttamisessa voidaan nähdä olevan kolme päätavoitetta. Ensimmäinen tavoite on tuotteen jakaminen itsenäisiin ja yhteensopiviin komponentteihin, jotta voidaan toteuttaa pienellä komponenttimäärällä suurempi lopputuotteiden variaatioiden määrä. Toinen tavoite on toteuttaa standardoidut rajapinnat komponenttien välille, jotta komponenttien yhdisteleminen olisi helppoa ja tehokasta. Kolmas tavoite on pystyä päivittämään ja muokkaamaan lopputuotteita vakaasti vaihtamalla tai muokkaamalla vain pientä määrä komponentteja. [Blecker and Friedrich, 2007]

Tuotekomponenttien jakaminen erillisiin moduuleihin ja räätälöinnin tekeminen mahdollisimman myöhäisessä vaiheessa tuotantoa ovat tehokkaita massaräätälöinnin tapoja [Jørgensen, 2010]. Erillisiin komponentteihin jakaminen mahdollistaa sen, että tuotantoprosesseissa voidaan keskittyä erillisten tuotekomponenttien yhtäaikaiseen kehitykseen.

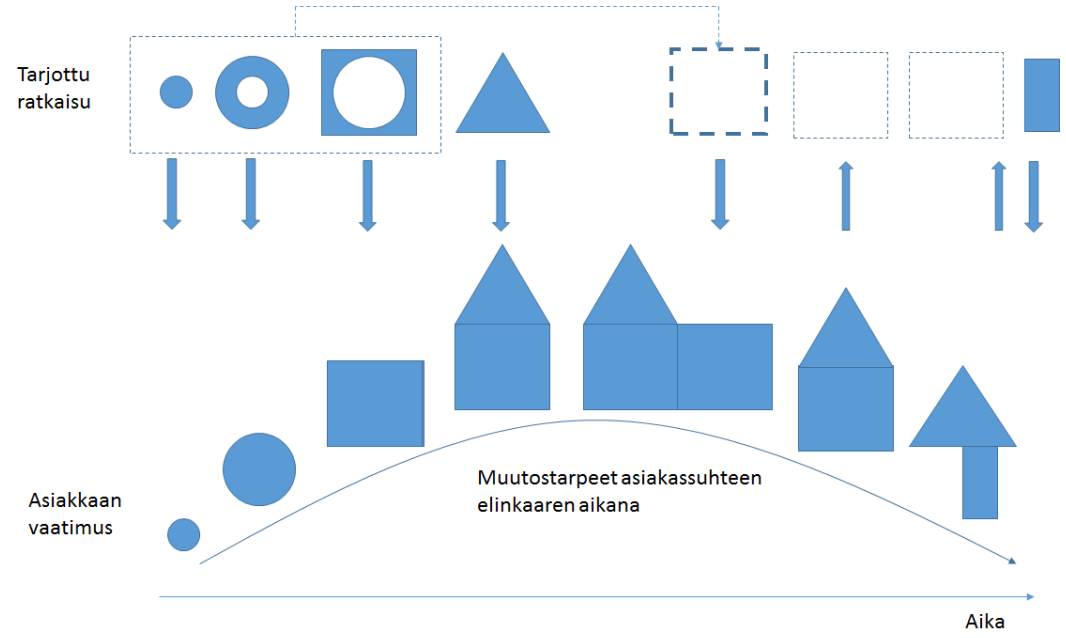
Erillisiä toisistaan riippumattomia tuotekomponentteja voidaan käyttää myöhemmin uudelleen, jolloin komponenttia ei tarvitse valmistaa uudelleen uudessa projektissa [Johannesson and Gedell, 2010]. Komponenttien uudelleenkäytöllä voidaan saavuttaa huomattavia kustannussäästöjä ja tuotantoajan lyhentymistä [Moon *et al.* 2010b]. On jopa mahdollista, että lopullinen tuote kootaan täysin valmiista erillisistä

komponenteista, jolloin varsinaista komponenttien kehitystä ei tarvita lainkaan. Tämä on ideaali tilanne modulaarisuuden hyödyntämisestä. Yleisesti ottaen pyrkimys on siihen, että varsinainen räätälöinti tehtäisiin mahdollisimman myöhäisessä vaiheessa tuotantoa. Tällä on yleensä suuri vaikutus tuotantoaikojen lyhenemiseen ja kustannusten pienenemiseen [Alfnes and Skjelstad, 2010].

Vaikka lopputuotteet olisivat keskenään hyvin erilaisia, on täysin mahdollista, että ne jakavat keskenään samoja komponentteja ja samoja alijärjestelmiä [Piller ja Tseng, 2007]. On jopa mahdollista, että koko tuotanto ja kaikki tuotteen osat jaetaan erillisiin komponentteihin. Tällöin tarvitaan tehokkaita menetelmiä komponenttien hallintaan, kuten erilaisia komponenttikirjastoja.

Hyvin toteutettu komponenttien hallinta on tärkeä osa modulaarista tuotesuunnittelua. Huono komponenttien hallinta hankaloittaa komponenttien uudelleenkäyttöä ja se voi aiheuttaa turhaa monimutkaisuutta tai komponenttien päällekkäisyyttä. Hyvä komponenttien hallinta taas mahdollistaa sen, että komponentit ovat tarkoituksensa puolesta uniikkeja, standardoituja ja hyvin luokiteltuja. Tämä taas mahdollistaa sen, että komponentit ovat helposti uudelleenkäytettävissä ja yhdisteltävissä. Yhdistelemistä ja uudelleenkäyttöä voidaan tarvita kokonaan uusissa tuotteissa tai asiakassuhteissa, mutta myös vanhoissa asiakassuhteissa ja vanhojen tuotteiden päivittämisessä.

On hyvin yleistä, että asiakkaan tarpeet ja vaatimukset voivat muuttua asiakassuhteen aikana [Siems and Walcher, 2010]. Myös tämä on syytä ottaa huomioon tuotekokonaisuutta suunnitellessa. Modulaarisella tuotekokonaisuudella tätä voidaan ennakoida. Kun tuote koostuu toisistaan riippumattomista yhteensopivista moduuleista, voidaan moduuleita lisätä tai poistaa asiakkaan tarpeiden mukaan. Hyvä modulaarinen tuote mahdollistaa jopa sen, että asiakas voi itse lisätä tai poistaa moduuleita oman tarpeensa mukaan [Siems and Walcher, 2010]. Kuvassa 5 on esitetty, miten modulaarisella tuotekokonaisuudella voidaan vastata asiakkaan muuttuviin tarpeisiin asiakassuhteen aikana.



Kuva 5. Muutostarpeet asiakassuhteen elinkaaren aikana [Siems and Walcher, 2010].

Modulaarisuus itsessään voidaan jakaa kolmeen osa-alueeseen, jotka ovat funktionaalinen modulaarisuus, tekninen modulaarisuus ja fyysinen modulaarisuus [Jiao and Tseng, 1999]. Tarkasteltaessa näitä kolmea osa-aluetta funktionaalisuus keskittyy siihen, miten modulaarisuus ilmenee toiminnallisuudessa. Toisin sanoen siis, kuinka modulaarisuus ilmenee toiminnoissa, joita modulaarisen tuotteen on tarkoitus toteuttaa. Tekninen modulaarisuus keskittyy siihen, kuinka modulaarisuus teknisesti toteutetaan. Fyysinen modulaarisuus keskittyy taas modulaarisuuden fyysisiin ilmiöihin, kuten fyysiseen rakenteeseen.

Standardoitu modulaarisuus mahdollistaa sen, että yhden tuotantoyksikön tai yrityksen ei tarvitse tuottaa kaikkia lopputuotteessa käytettäviä komponentteja [Blecker and Friedrich, 2007]. Standardoitua modulaarisuutta hyödyntämällä moduulit voivat tulla usealta eri toimittajalta ja ne kasataan yhteen samassa paikassa ennen kuin ne toimitetaan asiakkaalle. Tällaisessa toimintamallissa voidaan tehokkaasti hyödyntää eri toimijoiden välisiä verkostoja.

4.2. Tuotealusta

Tuotealusta on määritelty joukko moduuleita ja parametreja, joihin eri tuotteet pohjautuvat [Gao *et al.*, 2009]. Toisin sanoen tämä joukko muodostaa rungon, jonka ympärille tuotteet rakennetaan. Tuotealustan tarkoituksena on määritellä ne yhteiset komponentit, moduulit ja alijärjestelmät, joiden pohjalta erilaisia tuotekokonaisuuksia voidaan nopeasti ja tehokkaasti tuottaa. Joissain tapauksissa tuotealusta voidaan nähdä

myös itsenäisenä moduulina, jonka kaikki tuotealustaan perustuvat tuotteet jakavat [Blecker and Friedrich, 2007]. Tuotealustat ovat tehokas tapa toteuttaa massaräätälöintiä ja ne ovat usein välttämättömiä, mikäli halutaan toteuttaa toimiva tuotelinja- ja tuoteperhemalli [Gao *et al.,* 2009]. Tuoteperhe- ja tuotelinjamallia käsitellään tarkemmin luvuissa 4.3 ja 4.4.

Hyvin määritellyn tuotealustan tarkoituksena on pyrkiä hyödyntämään mahdollisimman paljon sellaisia komponentteja, joita voidaan käyttää useisiin eri tuotteisiin ja joita voidaan myös käyttää uudelleen [Johanssen ja Gedell, 2010]. Hyvin määritelty tuotealusta vähentää myös komponenttien ja moduulien määrää ja näin ollen mahdollistaa paremmat suurtuotannon tuomat edut [Moon *et al.,* 2010b]. Koska erillisten komponenttien ja moduulien määrä on pienempi, eivät erilaisten tuotteiden kokoonpanot ole niin monimutkaisia. Pienemmät moduuli- ja komponenttimäärät mahdollistavat myös nopeamman tuotannon läpimenoajan alhaisemmat tuotantokustannukset.

Hyvän tuotealustan määrittäminen vaatii sen, että itse tuotteet on hyvin määritelty. Täytyy siis tietää, mitä tuotteita halutaan tuottaa ja mitkä näiden tuotteiden vaatimukset ovat. Tuotealusta voidaan määritellä ja suunnitella vasta, kun tiedetään, millaisten tuotteiden pohjana se tulee toimimaan. Tuotealusta tulee myös hyvin todennäköisesti muuttumaan ja kehittymään tulevaisuudessa vastatakseen muuttuviin markkinoihin ja vaatimuksiin. Näin ollen myös tulevaisuuden muutokset täytyy ottaa huomioon alustan suunnittelussa.

Jotta massaräätälöintiä voidaan toteuttaa tehokkaasti, täytyy tuotealusta olla huolellisesti suunniteltu. Tuotealustan suunnittelun voidaan katsoa koostuvan kolmesta eri vaiheesta [Gao *et al.,* 2009]. Ensimmäisessä vaiheessa pyritään valitsemaan ne moduulit, joita tuotealustassa käytetään. Toisessa vaiheessa pyritään valitsemaan ne yksittäiset moduulien parametrit, joita tuotealustassa voidaan käyttää. Nämä parametrit voivat sisältää esimerkiksi määritykset suunnittelumenetelmille, moduulienhallinnalle ja palveluille koko tuotannon elinkaaren ajaksi [Gao *et al.,* 2009]. Kolmannessa vaiheessa pyritään suunnittelemaan koko tuotealustaa koskevat parametrit, jotka vaikuttavat kaikkiin tuotealustan moduuleihin. Näiden vaiheiden tavoitteena on toteuttaa kokonaisvaltainen ja joustava alusta, joka mahdollistaa tehokkaan tuotannon.

Uuden tuotealustan suunnittelua voidaan lähestyä myös niin, että tuotealusta muodostetaan vaatimusten ja käyttötarkoitusten pohjalta. Luokittelemalla asiakkaiden tarpeet voidaan suunnitella tuotealusta luokiteltujen asiakastarpeiden pohjalta [Jiao and

Tseng, 1999]. Löytämällä yhteiset ja samankaltaiset vaatimukset voidaan tehdä luokittelu, jonka pohjalta yhteiset ja samankaltaiset komponentit muodostetaan.

Tuotealustaa voidaan tarkastella kahdesta erilaisesta näkökulmasta, jotka ovat moduuliperustainen ja luokkaperustainen näkökulma [Gao *et al.,* 2009]. Moduuliperustaisessa näkökulmassa tarkastellaan nimenomaan komponentteja ja sitä, kuinka niitä yhdistelemällä voidaan toteuttaa erilaisia kombinaatioita. Luokkaperustaisessa lähestymistavassa keskitytään enemmän tuotealustan parametrisointiin ja siihen, kuinka räätälöintiä voidaan viedä vielä syvemmälle kuin pelkästään erilaisiin komponenttiyhdistelmiin [Gao *et al.,* 2009]. Tuotealustan avulla tulee voida nähdä kokonaiskuva siitä, millaisia tuotteita voidaan tällä hetkellä toteuttaa ja millaisia tuotteita tulevaisuudessa voidaan toteuttaa [Jiao and Tseng, 1999].

4.3. Tuoteperheet

Eräs tehokas malli kustannustehokkaan modulaarisen massaräätälöinnin toteuttamiseen on tuoteperhe, jossa hyvinkin erilaisia lopputuotteita voidaan toteuttaa siten, että ne perustuvat yhteiseen tuotealustaan. Tuoteperheet perustuvat hyvin pitkälti modulaarisuuteen ja komponenttien samankaltaisuuteen [Blecker and Friedrich, 2007]. Lopulliset tuotteet voivat poiketa hyvinkin paljon toisistaan ja tähdätä jopa täysin erillisille markkina-alueille, mutta ne voivat silti perustua yhteiseen alustaan [Moon *et al.,* 2010a]. Vaikka komponentit keskenään voivat olla hyvin samankaltaisia, voi niistä kootut lopputuotteet olla hyvinkin erilaisia [Blecker and Friedrich, 2007].

Esimerkki tuoteperheestä voi olla akkukäyttöisten työkalujen sarja, johon kuuluu porakone, naulain, sirkkeli ja hiomakone [Moon *et al.,* 2010a]. Näistä kaikki ovat omia tuotteitaan, mutta ne voivat sisältää samanlaisia elektronisia komponentteja, samanlaisen akkuliittimen, samanlaisen akun ja samanlaisen akkulaturin. Näin ollen lopputuotteiden käyttötarkoitus voi olla hyvinkin erilainen, mutta tuotteet pohjautuvat samaan tuotealustaan.

Tuoteperheessä pyritään yleensä toteuttamaan yhteiset komponentit siten, että ne sopivat mahdollisimman hyvin yhteen muiden tuoteperheen komponenttien kanssa [Jørgensen, 2010]. Tämä mahdollistaa sen, että tuoteperheen tuotteet voidaan parhaimmillaan toteuttaa pelkästään olemassa olevia komponentteja yhdistelemällä.

Tuoteperhemalliin pyritään tuotteiden samankaltaisuudella, jaetuilla tuotteiden osilla ja tehokkaalla tuotteiden uudelleenkäytöllä [Jiao and Tseng, 1999]. Uudelleenkäyttöä pyritään hyödyntämään kaikkien resurssien osalta, eikä pelkästään

komponenttien kohdalla. Uudelleenkäyttö on tärkeää myös tietämyksessä, osaamisessa ja erilaisissa prosesseissa [Jiao and Tseng, 1999]. Kaikkien resurssien tehokkaalla uudelleenkäytöllä voidaan saavuttaa huomattavaa kustannustehokkuushyötyä tuova tuoteperhemalli [Jiao and Tseng, 1999].

Alustapohjaiseen tuoteperhemalliin on yleensä kaksi lähestymistapaa: niin sanottu ylhäältä-alas- (Top-down) ja alhaalta-ylös (Bottom-up) -malli [Simpson *et al.,* 2006]. Ylhäältä-alas -malli tarkoittaa sitä, että yritys kehittää tuotteista koostuvan perheen, jotka pohjautuvat tuotealustaan ja sen osiin. Alhaalta-ylös -malli taas tarkoittaa tilannetta, missä yrityksellä on joukko erillisiä tuotteita, mutta ne suunnitellaan uudelleen ja standardoidaan siten, että niistä voidaan muodostaa tuoteperhe tuotannon tehostamiseksi [Simpson *et al.,* 2006]. Ylhäältä-alas -mallissa tuotteet siis perustuvat tuotealustaan. Alhaalta-ylös -mallissa tuotealusta taas pohjautuu tuotteisiin.

Lähestymismalli riippuu hyvin paljon siitä, kuinka pitkään yritys on toiminut ja millä markkina-alueella yritys toimii. Vaihtoehtoinen lähestymistapa on niin sanottu skaalautuvapohjainen tuoteperhe, jossa skaalautuvat muuttujat muokkaavat tuotealustaa tarpeen mukaan [Simpson *et al.,* 2006]. Tuotealustaan määritetään siis asetukset ja parametrit, joita muokkaamalla tuotealustaa voidaan muokata.

Toimivan tuoteperheen ja tuotealustan rakentaminen voidaan nähdä jatkuvana kehittyvänä toimintana [Johannesson and Gedell, 2010]. Tuoteperheen ja alustan ei yleensä nähdä olevan valmis ikinä, vaan sen erilaisten variaatioiden ja komponenttien kehitys jatkuu niin kauan kuin liiketoiminta jatkuu. Tämän avulla voidaan huolehtia siitä, että tuotekokoonpanot vastaavat muuttuvia markkinoita ja asiakkaiden muuttuvia vaatimuksia.

Toimiva tuoteperhemalli vaatii hyvää arkkitehtuurin suunnittelua, jotta asiakkaiden tarpeet voidaan täyttää yhdistelemällä erilaisia moduuleja ja muokkaamalla valmiita asetuksia [Jiao and Tseng, 1999]. Toimivaa tuoteperhearkkitehtuuria varten täytyy suunnitella tuotteiden arkkitehtuuri, tuotealustan arkkitehtuuri ja tuoteperheen arkkitehtuuri. Hyvä tuoteperhearkkitehtuuri mahdollistaa tehokkaan moduulien uudelleenkäyttämisen sekä tulevaisuudessa tapahtuvat muutokset [Jiao and Tseng, 1999].

Uusiokäyttämällä ja yhdistelemällä tuoteperheen eri osia, informaatiota ja resursseja yritykset voivat saavuttaa taloudellista hyötyä ja parantaa tuotannon joustavuutta sekä reagointikykyä [Moon *et al.,* 2010b]. Tämän avulla asiakkaiden vaatimusten muutoksiin ja kokonaan uusiin vaatimuksiin pystytään reagoimaan nopeammin ja näin pystytään parantamaan tuotannon kustannustehokkuutta.

Tuoteperheen tuotteiden täytyy olla hyvin joustavia, jotta ne vastaavat tehokkaasti asiakkaiden tarpeita. Tuoteperheiden kohdalla tuotteiden konfigurointi täytyy olla helposti tehtävissä. Konfiguroinnilla tarkoitetaan yleensä ennalta määrättäviä asetuksia ja parametreja, joita voidaan muokata asiakaskohtaisesti. Parhaimmillaan asiakkaat voivat tehdä konfiguroinnin jopa itse.

Helposti uudelleenkäytettävät ja yhdisteltävät tuoteperheen moduulit mahdollistavat myös joustavampien ja asiakasorientoituneempien tarjousten ja asiakasratkaisujen tekemisen kustannustehokkaasti [Johannesson and Gedell, 2010]. Tämä voi parhaimmillaan tarjota huomattavan kilpailuedun muihin toimijoihin nähden. Asiakas voi nähdä tarjouskilpailuvaiheessa joustavuuden niin suurena hyötynä, että on valmis maksamaan siitä enemmän.

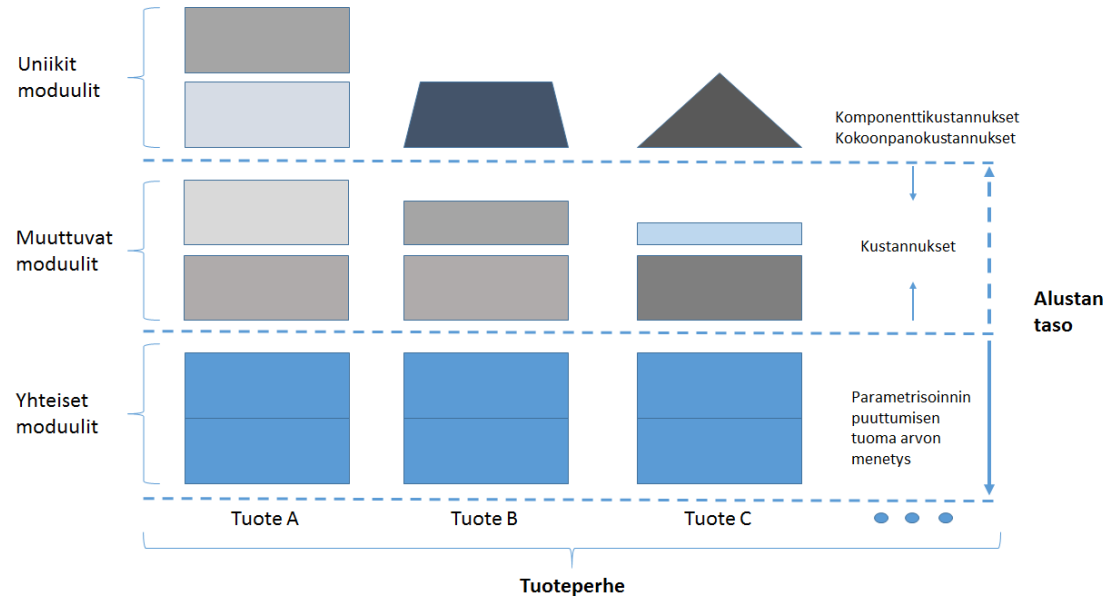
Toisaalta tuoteperheen tuotteiden variaatio ei saa olla liian suuri tai tuoteperheen tuotekokonaisuudet eivät saa olla liian monimutkaisia. Suuri variaatioiden määrä ja monimutkaisuus aiheuttaa helposti esimerkiksi toimitusaikojen pitenemistä, mikä taas tuo kilpailuhaittaa ja kustannuksia [Blecker and Friedrich, 2007].

Tuoteperhemallin soveltamiselle löytyy monia perusteluita sen tuomien hyötyjen pohjalta. On esimerkiksi yleistä, että asiakas ei tarvitse vain yhtä tuotetta tuoteperheestä, vaan useampia toisiinsa liittyviä tuotteita. Jos mietitään esimerkiksi luvun alussa mainittua sähkötyökaluesimerkkiä, on hyvin todennäköistä, että asiakas tarvitsee useampia työkaluja työkaluperheestä. Käyttämällä saman perheen työkaluja asiakas voi käyttää eri työkaluissa samaa akkua ja samaa laturia. Tämä tuo asiakkaalle suuren hyödyn mietittäessä, ostaako asiakas jonkun muun valmistajan työkalun vai kyseisen tuoteperheen työkalun.

Monet tuoteperheet ovat myös sellaisia, että tuotteita voidaan liittää toisiinsa ja ne voivat toimia yhdessä. Tällöin on tärkeää, että tuotteet tehdään hyvin ja helposti yhteensopiviksi keskenään [Wijnstra, 2002]. Tuotteiden välisten liittymien täytyy olla siis standardoituja ja yhteensopivuus täytyy ottaa huomioon jo tuotealustan ja komponenttien suunnittelussa.

Tuoteperhemallissa moduulit voidaan jakaa kolmeen toiminnallisuuteen perustuvaan kategoriaan: uniikit, yhteiset ja muuttuvat moduulit [Moon et al., 2010b]. Uniikit moduulit sisältävät yksilöllisiä funktioita toteuttavia komponentteja, joita ei voida korvata toisten moduulien komponenteilla [Moon et al., 2010b]. Uniikit moduulit ovat siis yleensä hyvin tuotespesifisiä ja niitä pyritään harvoin käyttämään uudelleen. Yhteiset moduulit perustuvat yhteisiin toiminnallisuuksiin eri tuotteiden välillä [Moon et al.,

2010b]. Näitä moduuleita voidaan siis käyttää tehokkaasti uudelleen ja ne voidaan jakaa eri tuotteiden kesken. Muuttuvat moduulit perustuvat yhteisiin toiminnallisuuksiin, joita voidaan kuitenkin parametrisoita tai muokata hieman eri tavalla eri tuotteiden kohdalla [Moon et al., 2010b] Muuttuvia moduuleita voidaan siis yhdistellä eri tavalla tai käyttää hieman eri asetuksilla riippuen tuotteesta. Moduulien jakautuminen tuotealustatasoilla on esitetty kuvassa 6.



Kuva 6. Moduulien jakautuminen tuotealustalla [Moon et al., 2010b].

Kuvassa 6 on esitetty myös kustannusten muodostuminen tuotteen uniikkiuden tuoman arvon lisääntymisen ja parametrisoinnin tuottaman arvonmenetyksen suhteen. Tämä tarkoittaa siis sitä, että uniikkien komponenttien toteuttaminen lisää aina tuotantokustannuksia ja pidentää läpimenoaikaa [Moon et al., 2010b]. Lisäksi uniikit komponentit aiheuttavat myös lisää kokoonpanokustannuksia. Toisaalta uniikit komponentit tuovat asiakkaalle todennäköisesti lisäarvoa ja asiakas on hyvin mahdollisesti valmis myös maksamaan yksilöllisyydestä [Moon et al., 2010b]. Näin ollen yksilöllisyyden ja yhteisten komponenttien käytön välille on löydettävä sopiva suhde maksimaalisen hyödyn saavuttamiseksi. Tämä edellyttää yleensä vahvaa markkina-alueen ja asiakkaiden tarpeiden tuntemusta.

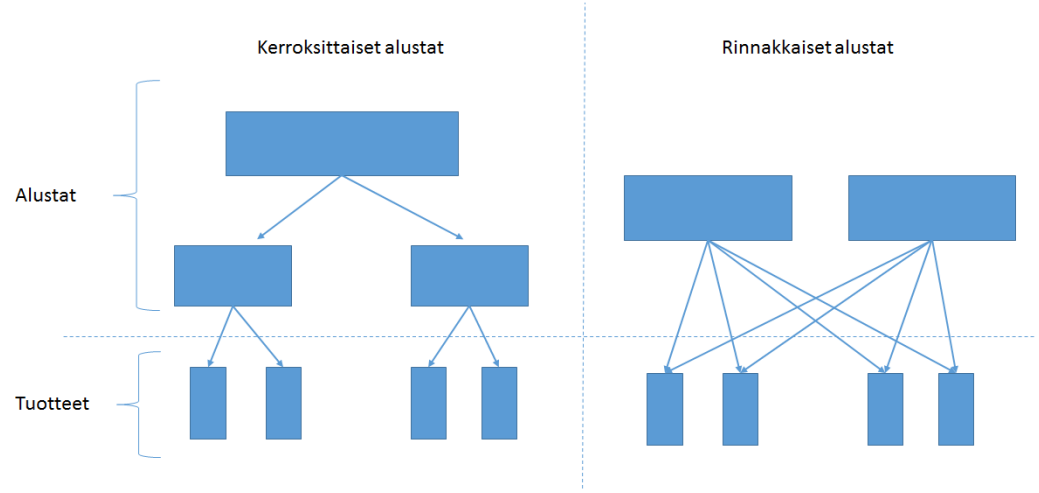
Monet tuotekokoonpanot sisältävät nykyään erilaisia monimutkaisia tekniikoita sekä tietokoneohjelmistoja. Näiden monimutkaisten ohjelmistojen toteutukseen tarvitaan usein tuhansia rivejä ohjelmakoodia ja satoja henkilötyöpäiviä [Wijnstra, 2002]. Mikäli tällaisissa järjestelmissä voidaan hyödyntää jo olemassa olevaa koodia, ei järjestelmää

kannata toteuttaa alusta alkaen kokonaan uudelleen. Tällaisissa tuotekokonaisuuksissa uudelleenkäyttö kannattaa pyrkiä maksimoimaan.

Kun komponentteja hyödynnetään tehokkaasti uudelleen, voidaan tämän avulla myös varmistaa uusien tuotteiden hyvä laatu [Wijnstra, 2002]. Kun komponentteja on käytetty jo aikaisemmin, tiedetään ne toimiviksi ja esimerkiksi testaamista ei tarvita niin paljon kuin mitä alusta alkaen tehdyn tuotteen kohdalla tarvittaisiin. Pienemmät testaustarpeet tuovat kustannussäästöjä tuotantovaiheessa. Lisäksi komponenttien mahdolliset ongelmat tai heikkoudet on jo todennäköisesti löydetty aikaisemmin eri tuotteiden kohdalla ja ne on voitu jo korjata. Tämä niin sanotusti lisää luottamusta tuotteen laatua kohtaan [Wijnstra, 2002].

Koska tuoteperheet ja tuotealustat ovat yleensä hyvin isoja kokonaisuuksia, menee niiden kehittämiseen yleensä paljon aikaa ja ne vaativat isoja investointeja. Näin ollen tuoteperhe ja -alusta on syytä suunnitella alusta alkaen siten, että niissä otetaan huomioon mahdollisesti muuttuvat teknologiat, vaatimukset ja markkinatilanteet [Wijnstra, 2002]. Tuoteperheen tai alustan elinkaari voi hyvinkin olla esimerkiksi yli kymmenen vuotta kestävä, jolloin teknologiat ja menetelmät ehtivät kehittymään paljon. Muutosten ennakoiminen mahdollistaa sen, että suuria investointeja vaativaa tuoteperheen kokonaisvaltaista uusimista ei tarvitse tehdä niin usein. Näin voidaan pidentää tuoteperheen elinkaarta ilman, että hävitään teknologiassa tai muissa muuttuvien markkinatilanteiden vaatimissa muutoksissa [Wijnstra, 2002].

Suurissa monimutkaisissa tuoteperhekokonaisuuksissa on mahdollista, että tuotteet eivät perustu vain yhteen alustaan, vaan ne voivat perustua myös useampaan alustaan [Wijnstra, 2002]. Useampi alusta voi olla hyvä ratkaisu nimenomaan silloin, kun tuoteperhe on monimutkainen ja käsittää esimerkiksi erilaisia teknologioita. Tällöin pelkästään yksi tuotealusta voi tuoda liikaa rajoitteita tuotteisiin. Toisaalta tuotealustoja ei saa olla liian monta, koska silloin menetetään helposti tuoteperhemallista saatavat hyödyt. Useat tuotealustat voivat olla joko kerroksittaisia, jolloin tuotealusta perustuu toiseen tuotealustaan, tai rinnakkaisia, jolloin tuotteet voivat perustua useampaan tuotealustaan. Esimerkiksi jos tuote koostuu elektroniikasta ja hydrauliikasta, voivat elektroniset komponentit pohjautua elektronisten komponenttien alustaan ja hydrauliset komponentit hydraulisten komponenttien alustaan. Kerroksittaiset ja rinnakkaiset mallit on esitetty kuvassa 7.



Kuva 7. Kerroksittaiset ja rinnakkaiset alustat [Wijnstra, 2002].

Mikäli käytössä on useampia tuotealustoja, täytyy tuotealustojen väliset yhteensopivuudet suunnitella tarkasti. On esimerkiksi varauduttava muutoksiin muissa tuotealustoissa, mikäli yhteen tuotealustaan tulee muutoksia. Onkin tärkeää hyväksyä, että tuotealusta ei ole vakaa muuttumaton pohja, vaan on syytä olettaa, että alusta kehittyy ja siihen tulee muutoksia [Wijnstra, 2002]. Tuotealustan muutokset voivat johtua esimerkiksi muuttuvista vaatimuksista, uusista teknologioista, uusista tuotteista tai muuttuvasta markkina-alueesta. Tuotealustan täytyy olla joustava muutoksille, jotta muuttuviin markkinoihin ja vaatimuksiin voidaan vastata tehokkaasti [Wijnstra, 2002].

Tuoteperhemalli ei käsitä vain tuotteiden modulaarisuutta ja tuoteperheen arkkitehtuuria, vaan tuoteperhemalli täytyy omaksua myös muissa liiketoiminnan osissa, kuten myynnissä, markkinoinnissa ja projektijohtamisessa [Wijnstra, 2002]. Jotta tuoteperhemalli toisi todellista kustannustehokkuushyötyä, on tämä toimintamalli sisäistettävä kaikilla liiketoiminnan osa-alueilla.

Tehokkaan tuoteperhemallin saavuttaminen vaatii markkina-alueen tuntemusta sekä jatkuvaa markkina-alueen kehityksen seurantaa. [Moon *et al.*, 2010b]. Yritysten kannattaa pyrkiä ennakoimaan muutoksia ja parantaa reagointikykyään mahdollisiin muutoksiin. Toimiva tuoteperhemalli vaatii myös tehokasta tiedonhallintaa, toimintastrategiaa ja kaikkien mahdollisten resurssien uudelleen käyttöä [Johannesson and Gedell, 2010]. Toimivan tuoteperhemallin saavuttamiseksi kannattaa siis käyttää resursseja markkina-alueiden tutkimiseen, asiakastutkimuksiin ja yleisen markkinakehityksen seuraamiseen. Erilaisten mittareiden ja tutkimusten avulla voidaan seurata tämän päivän tilannetta, mutta myös tulevaisuuden muutoksia on tärkeää pyrkiä ennustamaan. Tulevien trendien muutoksia, asiakkaiden maksuvalmiuden muutoksia ja tuotantokustannuksien muutoksia ei kuitenkaan ole välttämättä helppoa ennustaa.

4.4. Tuotelinjat

Kokonaisvaltainen tuotelinja ja sen hallinta on tärkeä osa toimivaa massaräätälöintiprosessia [Alfnes and Skjelstad, 2010]. Tuotelinja sisältää kaiken sen dokumentoinnin, tietämyksen, prosessit ja työkalut, joita massaräätälöityjen tuotteiden toteuttamiseen tarvitaan. Tuotelinja tulisi suunnitella siten, että kaikissa linjan osissa otetaan huomioon massaräätälöinnin tuomat edut. Linjan osien välillä tarvitaan hyvä kommunikaatio, jotta vaatimusten mukaisten laadukkaiden tuotteiden ja palveluiden tuottaminen kustannustehokkaasti ja nopealla toimitusajalla on mahdollista.

Tuotelinjan yhteys tuoteperheisiin löytyy siitä, että sen tarkoituksena on tuottaa tuoteperheitä. Tuotelinjan on tarkoitus luoda siis kokonaisvaltainen infrastruktuuri sille, että tuoteperhe voidaan toteuttaa tehokkaasti ja vaatimukset täyttäen. Hyvin määritellyn tuoteperheen muodostamiseen pyritään hyödyntämään kaikkia yhteisiä resursseja, joita on käytettävissä. Näitä resursseja ovat muun muassa prosessit, tietämys, ihmiset, suhteet ja osaaminen [Simpson *et al.*, 2006].

Tuoteperheen tuotteiden räätälöinti tuotelinjalla perustuu siihen, että komponentit itsessään tehdään helposti yhdisteltäviksi ja muokattavaksi selkeän ja standardoidun arkkitehtuurin kautta. Toinen vaihtoehto on malli, jossa määritellään erikseen tuoteperheeseen kuuluvat kiinteät yhteiset komponentit ja tuotekohtaiset uniikit komponentit [Tuck et *al.*, 2010]. Joka tapauksessa tuotelinjalla on syytä tehdä selkeä ryhmittely eri komponenttien ja moduulien välille, jotta niitä voidaan kehittää itsenäisesti omina yksikköinään tuoteperheiden tarpeisiin.

Tuotelinjat ovat yleisesti käytetty malli ja niiden haasteena nähdään usein suuri variaatioiden määrä. Tuotelinjan suunnittelussa ja käyttöönotossa on tärkeää toteuttaa variaatioidenhallinta [Jiao and Tseng, 1999]. Variaatioidenhallinta voidaan toteuttaa useilla eri tavoilla, mutta esimerkiksi mallintaminen on tehokas keino [Ma and Tan, 2006]. Mallintamisen avulla voidaan kuvata, minkälaisia eri suhteita moduulien välillä on ja minkälaisia eri kokonaisuuksia moduuleista voidaan tuottaa. Variaatiomahdollisuuksia ei voi olla täysin loputtomasti, koska se ei ole enää kustannustehokasta ja se myös tuo huomattavaa monimutkaisuutta. Suuri variaatioiden määrä aiheuttaa suoraan lisää kustannuksia [Blecker and Friedrich, 2007]. Monimutkaisuus ja suuri variaatioiden määrä johtaa usein myös siihen, että kokoonpano- ja toimitusajat kasvavat, mistä on selvää haittaa asiakkaille [Blecker and Friedrich, 2007].

Variaationhallinnalla tuotelinjassa pyritään myös siihen, että valmiita moduuleja käytetään järkevästi ja tehokkaasti uudelleen [Jiao and Tseng, 1999].

Variaationhallinnalla voidaan pyrkiä esimerkiksi siihen, että samaa toiminnallisuutta ja samoja vaatimuksia ei toteuteta usealla eri moduulilla vaan yksi moduuli vastaa näihin tarpeisiin kaikissa tuotteissa. Näin voidaan vähentää variaatioiden määrää ja monimutkaisuutta.

5. Massaräätälöinti ohjelmistotuotannossa

Ohjelmistotuotantoa on perinteisesti tehty massatuotantoon verrattavana tuotantona tai täysin räätälöitynä tuotantona. Massatuotannossa on toteutettu yksilöllinen ohjelmisto, joka toimitetaan käytännössä samanlaisena kiinteänä pakettina kaikille asiakkaille. Täysin räätälöidyssä ohjelmistotuotannossa taas on perinteisesti toteutettu täysin uniikki ohjelmisto tyhjästä asiakkaan tarpeiden mukaan.

Massatuotannon heikkous ohjelmistotuotannossa tulee siitä, että yhdellä yksiselitteisellä asetusmäärittelyllä ohjelmisto ei palvele kaikkia mahdollisia potentiaalisia asiakkaita [Meyer and Webb, 2005]. Jos ohjelma kuitenkin tehdään modulaariseksi ja helposti parametrisoitavakasi, voidaan järjestelmälle tavoittaa huomattavasti suurempi asiakaskunta.

Täysin räätälöityjen ohjelmistoratkaisujen heikkous tulee siitä, että ne ovat paljon aikaa vieviä projekteja ja niissä ei voida hyödyntää komponenttien uudelleenkäyttöä. Täysin räätälöidyissä projekteissa muun muassa vaatimusmäärittely, arkkitehtuurin suunnittelu, toteutus, testaaminen ja käyttöönotto joudutaan tekemään kaikissa projekteissa kokonaan uniikilla tavalla. Tällainen malli vaatii pitkän tuotantoajan ja aiheuttaa huomattavia kustannuksia. Massaräätälöinnin keinoilla voidaan kuitenkin yhdistää asiakaskohtainen räätälöinti ja massatuotannon tehokkuus.

Massaräätälöinti ohjelmistotuotannossa asettuu massatuotannon ja täysin räätälöidyn tuotannon väliin käyttäen molemmista tuotantomalleista saatavia hyötyjä [Verdouw *et al.*, 2014]. Suurimmat hyödyt ovat massaohjelmistotuotannon kustannustehokkuus sekä asiakaskohtaisen räätälöinnin tuoma arvo. Massaräätälöinti ohjelmistotuotannossa ei sinänsä ole kovinkaan uusi keksintö, koska modulaarisuutta ja muita menetelmiä on pyritty hyödyntämään ohjelmistoissa jo lähes niin kauan, kun ohjelmistoja on tuotettu [Hoek and Lopez, 2011]. Menetelmät massaräätälöinnin toteuttamiseen ovat kuitenkin kehittyneet vuosikymmenten aikana.

Tietojärjestelmät ja ohjelmistot ovat tärkeä osa massaräätälöintiprosessia erilaisten tuotteiden ja palveluiden tuottamisessa. Myös ohjelmisto tai tietojärjestelmä itsessään voi olla massaräätälöityvä ja ohjelmistotuotantoa voidaan tehdä massaräätälöinnin keinoin [Watson, 2010]. Monissa tapauksissa massaräätälöinnin periaatteet sopivatkin erittäin hyvin ohjelmistotuotantoon. Monien ohjelmistotuotantoyritysten kohdalla niiden menestys voidaan liittää nimenomaan yrityksen kykyyn tukea ohjelmistotuotteidensa massaräätälöintiä [Meyer and Webb, 2005].

Kaikkeen ohjelmistotuotantoon massaräätälöinnin periaatteita ei kannata pyrkiä soveltamaan, mutta monessa tapauksessa näillä keinoilla voidaan saada ohjelmistotuotteista joustavampia, tuotantoajoista lyhempiä ja tuotantokustannuksista pienempiä.

Ohjelmistotuotannossa massaräätälöinnin periaatteita voidaan soveltaa hyvin pitkälle. Joissain tapauksissa ohjelmistotuotantoa voidaan pitää jopa ääriesimerkkinä kaupallisten tuotteiden massaräätälöinnistä [Meyer and Webb, 2005].

Vaikka massaräätälöinnin keinoilla voidaan saavuttaa monia hyötyjä, ei se silti ole yksiselitteinen tai helppo malli toteutettavaksi. Suurin haaste on saada tehokas standardointi ja joustava räätälöinti kohtaamaan siten, että se palvelee liiketoimintaa mahdollisimman hyvin [Verdouw *et al.*, 2014].

Ohjelmistokehitysprosessi ja itse ohjelma vaatii suunnittelua ja tehokasta organisointia, jotta massaräätälöinnin keinojen hyödyntäminen on mahdollista. Ilman suunnittelua ja organisointia prosessista sekä tuotteesta tulee helposti hallitsematon kaos [Meyer and Webb, 2005]. Massaräätälöinti on helppo toteuttaa ohjelmistotuotannossa huonosti. Esimerkiksi liiallinen variaatioiden määrä, liian monimutkaiset järjestelmät ja liika räätälöinti voivat johtaa tehottomaan ja kalliiseen lopputulokseen. Hyvin suunnitellulla kurinalaisella prosessilla ja arkkitehtuurilla nämä haasteet voidaan kuitenkin torjua [Meyer and Webb, 2005].

Tuotantoprosessin tulisi siis olla kaikin puolin hyvin ja yksilöllisesti suunniteltu. Kokonaisvaltainen massaräätälöinnin ratkaisukenttä määritellään ohjelmistotuotannossa samaan tapaan kuin teollisessakin tuotannossa. Markkina-alueen tuntemus täytyy olla vahvaa ja tuotantomallien hyödyntäminen kannattavaa. Tuotantoprosessissa tulee huomioida monimutkaisuuden hallinta ja tulevien muutosten ennakoiminen. Tuotantoprosessissa täytyy hyväksyä, että esimerkiksi teknologiat ja asiakkaiden vaatimukset tulevat muuttumaan ajan saatossa. Myös pitkäkestoisten asiakassuhteiden hallinnointi on yhtä tärkeää ohjelmistotuotannossa kuin teollisessakin tuotannossa. Arkkitehtuurin suunnittelu ja suunnittelua tukevat menetelmät, ovat elintärkeä osa tuotantoprosessia, jotta voidaan tuottaa vaatimukset täyttäviä massaräätälöityviä ohjelmistotuotteita. Arkkitehtuurin suunnitteluun ja sitä tukeviin menetelmiin perehdytään tarkemmin luvussa 6.

Tuotantoprosessin lopputuloksena syntyvät järjestelmät ovat usein suuria monimutkaisia kokonaisuuksia, joiden toteutus on hyvin kallista. Tästä syystä mahdollinen valmiiden ohjelmistokomponenttien tai koodin uudelleenkäyttö voi tuoda

huomattavia kustannussäästöjä ja lyhentää tuotantoaikaa. Tästä johtuen tuotantoprosessissa uudelleenkäytön hyödyntäminen halutaan viedä mahdollisimman pitkälle.

Asiakkaat tarvitsevat järjestelmiä usein hyvin samankaltaisiin tarkoituksiin, mutta hieman erilaisilla toiminnoilla ja asetuksilla. Mietitään esimerkiksi ohjelmistoa, joka on suunniteltu työajanhallintaan ja työvuorosuunnitteluun. Käyttötarkoitukset asiakkailla ovat varmasti hyvin samankaltaiset. Tiivistetysti työntekijöille halutaan suunnitella työvuorot, joista käy ilmi, koska työntekijä on töissä ja koska vapaalla. Lisäksi halutaan mahdollisesti seurata toteutuneita tunteja, lomapäiviä ja esimerkiksi sairaspoissaolopäiviä.

Vaikka käyttötarkoitus eri asiakkaiden kohdalla olisi pohjimmiltaan sama, ovat tarpeet järjestelmän asetuksille hieman erilaiset. Jos järjestelmä on esimerkiksi käytössä useammassa maassa, vaihtelee työaikalainsäädäntö maakohtaisesti ja tällä on vaikutusta järjestelmässä mahdollisesti käytettäviin mittareihin. Työvuorosuunnittelussa ja työajan hallinnassa voi olla myös työehtosopimussidonnaisuutta toimialakohtaisesti, mikä vaikuttaa työvuorosuunnittelun toteutumiseen. Myös suunnittelun tarve voi poiketa paljon erilaisilla asiakkailla. Joillakin asiakkailla työvuorosuunnittelu on esimerkiksi hyvin kapasiteettipohjaista. Muun muassa asiakaspalvelutehtävissä tietty kapasiteetti on täytyttävä määrättyinä ajankohtina ja tätä kapasiteetin täyttymistä pitää pystyä seuraamaan järjestelmästä. Muilla toimialoilla tällaisella kapasiteetin täyttymisellä taas ei välttämättä ole merkitystä, jolloin työvuorosuunnittelun tarpeet muodostuvat eri tavalla.

Tästä tiivistetystä esimerkistä voidaan huomata massaräätälöinnistä mahdollisesti saatavat hyödyt. Ei ole kannattavaa tehdä jokaisella asiakkaalle omaa uniikkia järjestelmää, vaan suunnitella oikeat asetukset parametrisoitavaksi ja muokattavaksi.

Luvussa 4 esiteltiin massaräätälöintiin vahvasti liittyvät käsitteet: modulaarisuus, tuoteperheet, tuotealustat ja tuotelinjat. Nämä kaikki ovat käsitteitä, jotka voidaan toteuttaa massaräätälöidyssä ohjelmistotuotannossa ja ne toimivat pohjana massaräätälöityjen ohjelmistojen toteuttamiselle. Modulaarisuutta voidaan pitää ohjelmistotuotannossa massaräätälöinnin kulmakivenä ja komponenttipohjaiselle lähestymistavalle löytyy useita perusteluja. Tällöin ohjelmistossa voidaan muun muassa ottaa käyttöön vain ne komponentit, jotka kyseisessä ympäristössä tarvitaan.

Myös tuoteperhemalli on ohjelmistotuotannossa hyvin yleistä. Erilliset ohjelmat voidaan määritellä omiksi tuotteiksi, mutta niiden pohjana voi toimia samoja

komponentteja ja ohjelmat voivat olla keskenään yhteensopivia. Tuotealustat ovat tärkeä osa tuoteperhemallia myös ohjelmistotuotannossa ja ohjelmistoperheen eri sovellukset voidaan toteuttaa niin, että ne pohjautuvat yhteiseen tuotealustaan [Meyer and Webb, 2005]. Tuotealustojen hyödyistä ohjelmistotuotannossa löytyy vahvaa näyttöä, sillä useat tutkimukset osoittavat, että tuotealustan avulla voidaan tuottaa laadullisesti parempia ohjelmistoja pienemmillä kustannuksilla ja lyhemmässä ajassa [Alsawalqah *et al.,* 2013]. Myös tuotelinjoja voidaan hyödyntää massaräätälöityvässä ohjelmistotuotannossa. Modulaarisuuden, tuoteperhemallien ja tuotealustojen arkkitehtuuria käydään läpi tarkemmin luvussa 6.

Luvussa 3 mainittiin massaräätälöinnin neljä osa-aluetta: mukautuva, läpinäkyvä, kosmeettinen ja yhteistoiminnallinen massaräätälöinti. Nämä kaikki neljä osa-aluetta ovat toteutettavissa myös ohjelmistotuotannossa ja toimivat massaräätälöinnin kulmakivinä. Ruotsalainen telekommunikaatiojärjestelmiä valmistava yritys Ericsson on esimerkki globaalista yrityksestä, joka on kohdistanut massaräätälöinnin näille neljälle osa-alueelle [Mathiassen and Sandberg, 2014]. Ericssonin kohdalla on myös huomattu, että massaräätälöintiä näillä neljällä osa-alueella ei välttämättä tarvitse kohdistaa pelkästään lopputuotteeseen, vaan ne voidaan kohdistaa myös omien prosessien ja tuotantomallien toteutukseen. Näiden neljän osa-alueen hyödyntämistä pohditaan tarkemmin seuraavissa neljässä luvussa.

5.1. Mukautuvuus

Ohjelmistotuotannossa massaräätälöityvyys perustuu hyvin pitkälti ohjelmiston modulaarisuuteen. Modulaarisuus ohjelmistotuotannossa tarkoittaa sitä, että ohjelmisto hajautetaan itsenäisiin osiin, jotka voidaan myöhemmin kasata yhteen lopulliseen järjestelmään [Hoek and Lopez, 2011]. Jokainen ohjelmiston osa voidaan mieltää siis itsenäiseksi komponentiksi tai moduuliksi. Modulaarisuus ja standardoitu moduulien linkitys ovat avaintekijöitä toimivissa massaräätälöintiratkaisuissa [Verdouw et al., 2014]. Suuria monimutkaisia järjestelmiä on jopa hyvin hankalaa toteuttaa ilman modulaarista lähestymistapaa [Hoek and Lopez, 2011].

Mukautuvuus itsessään tarkoittaa sitä, että asiakkaalle voidaan koota yhteensopivista komponenteista ja järjestelmistä asiakkaan tarpeita vastaava ohjelmistokokonaisuus. Tämän toteuttaminen tehokkaasti on mahdollista vain modulaarisen toteutuksen pohjalta. Parhaassa tapauksessa asiakas voi jopa itse tehdä komponenttien valinnan ja kokoonpanon. Tämä on kuitenkin usein sen verran

monimutkainen prosessi, että toimittajan apu on välttämätöntä. Voidaan kuitenkin sanoa, että ohjelmistotuotteiden mukautuvuus syntyy ohjelmiston modulaarisuuden pohjalta.

Modulaarisuus mahdollistaa tuoteyksikköjen keskittymisen yksittäisten moduulien tuottamiseen, mikä taas mahdollistaa paremman tuotantovolyymin. Tuotantovolyymin paraneminen on perusteltavissa sillä, että tuotteet koostuvat pääasiallisesti valmiiden ohjelmistokomponenttien erilaisista yhdistelmistä. Näin ollen ohjelmistokomponentteja ei tarvitse toteuttaa jokaisen tuotteen kohdalla uudelleen, vaan kokoonpano toteutetaan valmiita komponenttikirjastoja hyödyntäen. Modulaarisuus mahdollistaa jopa sen, että tuotantoyksiköt voidaan tehokkaasti hajauttaa esimerkiksi maailmanlaajuisesti [Hoek and Lopez, 2011]. Tällöin maantieteelliset esteet on helpompi ylittää ja kehitystä voidaan tehdä rinnakkain globaalisti jopa eri puolilla maapalloa sijaitsevissa tuotantoyksiköissä.

Systemaattinen modulaarisuuden hyödyntäminen mahdollistaa myös sen, että tuotantoyksiköiden ei tarvitse olla koko toimiympäristön kattavia osaajia. Näin ollen osaaminen voidaan keskittää määrätylle spesifille alueelle, jolloin osaaminen tällä alueella voidaan hioa äärimmilleen [Zakál *et al.,* 2011]. Tällöin tuotantoyksiköt voivat olla jonkin alueen huippuosaajia sen sijaan, että ne osaisivat hieman kaikkea. Tämä mahdollistaa sen, että ohjelmistokehittäjät voivat keskittyä varsinaisiin kehitystehtäviin, eikä heidän tarvitse olla tuotealueen huippuosaajia [Zakál *et al.,* 2011]. Vastaavasti tuotealueesta vastaavien henkilöiden ei tarvitse keskittyä kehityksen ja ylläpidon keskeisiin tekijöihin.

Modulaarista ohjelmistokehitystä voidaan toteuttaa muun muassa komponenttipohjaisen tuotekehityksen avulla. Komponenttipohjaiset tuotekehitystekniikat ovat ohjelmistokehityksen tapoja, joissa ohjelmisto toteutetaan tvalmiiden komponenttien pohjalta [Li *et al.,* 2006]. Useat tutkimukset osoittavat, että komponenttipohjaisten ohjelmistotuotantotekniikoiden avulla voidaan parantaa monimutkaisten järjestelmien kehityksen tuottavuutta [Li et al., 2006].

Komponentteihin perustuva ohjelmointi on ollut tärkeä osa ohjelmistotuotantoa ohjelmoinnin historian alusta lähtien. Vuonna 2003 jopa 70% kaikesta uudesta ohjelmistotuotannosta tehtiin komponenttipohjaiseen ohjelmointiin perustuen [Brown, 2000]. Ohjelmistojen tehokkaampi komponentteihin jakaminen voidaankin nähdä tärkeimpänä tekijänä laadullisesti parempien ohjelmistojen ja tehokkaamman ohjelmistotuotannon saavuttamisessa [Aoyama, 1998].

Komponenttipohjaisuudella pyritään mahdollisimman tehokkaaseen komponenttien uudelleenkäyttöön ja ohjelmistojen joustavuuteen. Toki uudelleenkäyttöä

voidaan toteuttaa muillakin tavoilla, mutta komponenttipohjaisuus mahdollistaa useiden hyödyllisten menetelmien, kuten suunnittelumallien, viitekehysten ja valmiiden uudelleenkäyttöä tukevien arkkitehtuurimallien käytön [Aoyama, 1998].

Modulaarisia ohjelmistokokonaisuuksia voidaan toteuttaa myös niin, että eri toimittajilta tulevia ohjelmistokomponentteja voidaan yhdistellä keskenään. Tämä mahdollistaa niin sanotun best-of-breed -ratkaisumallin, jossa moduuli tulee siltä toimittajalta, joka pystyy parhaiten palvelemaan moduulille asetettua liiketoimintatarkoitusta [Verdouw *et al.*, 2014]. Tällainen malli mahdollistaa myös palvelukeskeisen toimintamallin, jossa eri toimittajien toteuttamat sovellukset nähdään määritettyjen toiminnallisuuksien ympärille toteutettuina palveluina. Nämä palvelut voivat kommunikoida keskenään standardoituja rajapintoja hyödyntäen.

Modulaarisessa ohjelmistotuotannossa tulee ottaa huomioon, että tavoitteena ei ole pelkästään tuottaa ohjelmaa, joka käyttöönottovaiheessa on mukautuva. Modulaarinen ohjelmisto ja ohjelmistoperhe halutaan tuottaa sellaiseksi, että modulaarisuutta voidaan hyödyntää tehokkaasti koko ohjelmiston tai ohjelmistoperheen elinkaaren ajan [Hoek and Lopez, 2011]. Joissain tapauksissa käytössä olevien ohjelmistojen elinkaari voi olla hyvin pitkä ja hyvin toteutettu modulaarisuus mahdollistaa sen, että ohjelmiston osia voidaan uudistaa pala kerrallaan siten, että lopulta koko ohjelmisto on täysin uusittu. Tämä tukee pitkiä asiakassuhteista ja mahdollistaa pitkän elinkaaren ohjelmistolle. Tästä on myös hyötyä asiakkaalle, sillä kokonaisvaltaiset tietojärjestelmien ja ohjelmistojen uusimiset ovat yleensä hyvin kalliita ja hankalia prosesseja.

Sen lisäksi, että modulaarisuudella saavutetaan kustannustehokkaampi tuotantomalli, tekee modulaarisuus myös kehitystyöstä helpompaa. Kehittäjätasolla modulaarisuus vähentää ohjelmistojen monimutkaisuutta ja näin ollen mahdollistaa paremman ymmärryksen ohjelmistokokonaisuuksista [Hoek and Lopez, 2011]. Monissa projekteissa kehitystyö on oletusarvoisesti sellaista, että se perustuu osittain jo olemassa olevaan koodiin ja siihen lisätään uusia komponentteja. Kun alkuperäinen koodi on tehty hyvin modulaariseksi, on myöhemmät kehitystoimenpiteet huomattavasti helpompi toteuttaa.

Tuotteen elinkaaren tasolla modulaarisuus mahdollistaa jatkuvan evoluution [Hoek and Lopez, 2011]. Tuotteen kehitystä voidaan siis jatkaa koko tuotteen elinkaaren ajan ja vanhentuvia komponentteja voidaan korvata uusilla komponenteilla. Komponentteja voidaan myös lisätä, poistaa tai muokata tuotteen elinkaaren aikana.

Tuoteperhemallia ja tuotelinja-ajattelua hyödyntämällä voidaan saavuttaa huomattavia kustannussäästöjä, kun korjauksia tai kehityksen mukaista ylläpitoa ei tarvitse tehdä päällekkäin [Alsawalqah *et al.,* 2013]. Korjaukset ja ylläpito voidaan tehdä yhteen paikkaan siten, että ne vaikuttavat suoraan kaikkiin tuotteisiin, jotka jakavat yhteisiä komponentteja. Todelliset kustannussäästöt muodostuvat kehityksessä aikaansaaduista säästöistä ja säästöistä, jotka saadaan aikaan yksinkertaisemmilla ylläpito- ja päivitystoimenpiteillä [Alsawalqah *et al.,* 2013].

Ohjelmointitasolla modulaarisuudesta voidaan käyttää useita eri termejä, kuten rakenteellinen ohjelmointi, modulaarinen ohjelmointi, komponenttipohjainen ohjelmointi tai oliopohjainen ohjelmointi. Näillä kaikilla on kuitenkin sama tavoite eli hajottaa ohjelma pienempiin hallittaviin yksiköihin [Brown, 2000]. Näiden yksiköiden avulla voidaan saavuttaa yksinkertaisempi rakenne, parempi komponenttien uudelleenkäyttö ja helpommin muokattavat ohjelmat. Tämä jako on tärkein lähestymistapa useissa ohjelmistotuotannon malleissa [Brown, 2000].

Telekommunikaatiojärjestelmiä tuottava Ericsson on lähestynyt mukautuvuutta omassa tuotantomallissaan siten, että he ovat määritelleet oman ketterän viitekehyksen yhteisesti kaikille tuotantoyksiköilleen [Mathiassen and Sandberg, 2014]. Kaikkien tuotantoyksikköjen toiminta pohjautuu tähän samaan prosessiviitekehykseen, joka tukee massaräätälöintiä ja ohjelmistojen modulaarisuutta. Tarpeen vaatiessa yksiköt voivat kuitenkin muokata prosessia omien tarpeidensa mukaan. Tämän avulla tuotantoyksiköt voivat toimia rinnakkain maantieteellisesti useissa eri toimipisteissä.

5.2. Läpinäkyvyys

Läpinäkyvyyteen massaräätälöitävässä ohjelmistotuotannossa liittyy paljon samoja tekijöitä kuin muussakin teollisuudessa. Erilaiset markkinatutkimukset ja niihin liittyvä tiedon analysointi on avainasemassa. Kaikki mahdollinen asiakkailta saatava palaute on tärkeää ja sitä täytyy pyrkiä keräämään. Monissa Web -pohjaisissa sovelluksissa asiakaskyselyt on helppo sijoittaa tarjottavaan ohjelmistoon ja näiden avulla voidaan jouhevasti kerätä tietoa. Myös muista asennuspohjaisista sovelluksista voidaan kerätä tietoa esimerkiksi erilaisten verkkolomakkeiden avulla.

Tutkimusten ja asiakaskyselyiden lisäksi ohjelmistokehityspuolella on tärkeää pyrkiä oppimaan, miten asiakkaat käyttävät järjestelmää [Meyer and Webb, 2005]. Kun asiakkaille tarjotaan monimutkaista ohjelmistokokonaisuutta, johon kuuluu erilaisia komponentteja, asetuksia, liittymiä ja parametreja, jokainen asiakas voi löytää hieman

erilaisen tavan soveltaa ohjelman käyttöä. Jokainen asiakas pyrkii muokkaamaan ohjelmistoa juuri omiin käyttötarkoituksiinsa. Tämä on arvokasta tietoa myös ohjelmistokehittäjälle, jotta voidaan oppia, miten asiakkaat järjestelmää käyttävät ja mitkä ovat asiakkaiden mielestä parhaaksi todetut toimintamallit [Meyer and Webb, 2005]. Joissain tapauksissa asiakas voi myös jatkokehittää ohjelmistoa tai tehdä siihen erilaisia liitännäisiä tai omia komponentteja. Myös nämä ovat arvokasta tietoa ohjelmistokehityksen kannalta, koska tämän tyyppisten toiminnallisuuksien lisääminen ohjelmiston omiin komponentteihin voi olla hyödyllistä. Yleisesti ottaen voidaan todeta, että asiakkailta oppiminen erilaisissa ohjelmiston käyttötarkoituksissa voi parantaa huomattavasti ohjelmistoyrityksen kilpailukykyä [Meyer and Webb, 2005].

Moniin ohjelmistoihin ja varsinkin palveluna tuotettaviin järjestelmiin tarjotaan asiakastukea. Asiakastuki on voitu toteuttaa useita eri kommunikaatiokanavia pitkin, kuten sähköpostilla, puhelintuella tai tikettijärjestelmällä. Asiakastuen tarpeita analysoimalla voidaan saavuttaa arvokasta tietoa järjestelmän kehityksen kannalta. Asiakastuen tietoja analysoimalla voidaan selvittää, minkä osioiden kanssa asiakkailla on eniten ongelmia ja mistä ongelmat johtuvat. Nämä ongelmat voidaan korjata saavutetun tietämyksen pohjalta.

Ericssonilla läpinäkyvyyttä tuotantoprosessissa on hyödynnetty siten, että prosessimallia suunniteltaessa on pyydetty eri toimiyksiköiltä kommentteja organisaation sisällä suunniteltuihin prosessimalleihin [Mathiassen and Sandberg, 2014]. Toimiyksiköitä on pyydetty myös itse suunnittelemaan toimintamalleja ilman, että yksiköt ovat varsinaisesti tienneet niiden tulevan käyttöön. Näiden materiaalien pohjalta yksiköille on suunniteltu prosessit ja toimintamallit, jotka heillä on käytössä.

5.3. Kosmeettisuus

Kosmeettisuutta pyritään hyödyntämään ohjelmistotuotannossa usein siten, että ohjelmiston ulkoasu tehdään helposti muokattavaksi. Näin ohjelmisto voidaan tehdä muuhun organisaation ohjelmistoulkoasuun sopivaksi. Yleistä on esimerkiksi asiakasorganisaation logojen sisällyttäminen järjestelmään. Tämän lisäksi pyritään usein muokkaamaan järjestelmän väriteemoja ja ulkoasuja asiakasorganisaation teemojen mukaiseksi. Monet sovellukset sisältävät joukon valmiita erilaisia teemoja, joista asiakas voi valita sopivimman.

Varsinkin avoimen lähdekoodin ohjelmistoissa ulkoasun muokkausta voidaan viedä vieläkin pidemmälle. Näissä ulkoasu on yleensä sisällytetty erillisiin

tyylitiedostoihin, jolloin ulkoasun muokkaaminen voidaan viedä ohjelmistokooditasolla vielä syvemmälle. Tyylitiedostoja muokkaamalla ohjelmiston ulkoasua voidaan parhaimmillaan muuttaa hyvinkin radikaalisti.

Kosmeettinen massaräätälöinti on yksi helpoimmista räätälöinnin keinoista, koska siinä ei yleensä tarvitse puuttua ohjelmiston rakenteeseen, toiminnallisuuteen tai arkkitehtuuriin. Ulkoasumuutosten teko on yleensä melko helppoa ja edullista, mutta joissain tapauksissa se voi tuoda asiakkaalle suurta lisäarvoa. Tämä pätee varsinkin erilaisiin julkaisualustoihin ja sisällönhallintajärjestelmiin.

Ericssonilla kosmeettisuus omassa ohjelmistotuotantoprosessissa on toteutettu siten, että eri puolella maailmaa olevilla toimijoilla prosessi ja mallit on räätälöity silmälläpitäen maantieteellisiä eroavaisuuksia [Mathiassen and Sandberg, 2014]. Saman prosessin ja viitekehyksen pohjalta on luotu omat räätälöidyt toimintamallit jokaiselle tuotantoyksikölle huomioon ottaen maantieteelliset ja kulttuurilliset eroavaisuudet

5.4. Yhteistoiminnallisuus

Yhteistoiminnallisuus on monissa tapauksissa tärkeä osa massaräätälöitävää ohjelmistotuotantoprosessia. Asiakkaan osallistuminen on usein välttämätöntä ainakin projektin alkuvaiheen vaatimusmäärittelyssä. Kokonaisvaltaisen vaatimusmäärittelyn tekeminen projektin alkuvaiheessa voi kuitenkin olla hyvin kallista ja haastavaa [Sommerville, 2007]. Kokonaisvaltaisen vaatimusmäärittelyn tekeminen on haastavaa varsinkin silloin, kun asiakas ei ole varsinainen asiantuntija tai kovinkaan tekninen henkilö. Tällöin voi olla vaikeaa määritellä yksiselitteisesti, mitä asiakas ohjelmistolta haluaa. Varsinkin tällaisissa tilanteissa yhteistoiminnallisuus on tärkeää, jotta vaatimusmäärittelyä voidaan tarkentaa ja muokata projektin myöhemmissä vaiheissa.

Vaatimusmäärittelyvaiheen lisäksi käyttöönotto on sellainen vaihe, jossa tarvitaan yleensä vahvaa teknistä tukea ja toimittajan asiantuntemusta [Verdouw *et al.*, 2014]. Monimutkaisten järjestelmäkokonaisuuksien kohdalla asiakas usein tarvitsee ohjelmistotoimittajan asiantuntemusta valitakseen ne komponentit ja asetukset, joilla päästään asiakkaan haluamaan lopputulokseen. Yksinkertaisemmissa ohjelmistokokonaisuuksissa osa yhteistoiminnallisesta asiakastuesta voidaan myös korvata erilaisilla työkaluilla, jotka johdattavat asiakkaan kokoonpano- ja käyttöönottovaiheen läpi [Verdouw *et al.*, 2014]. Näitä työkaluja kutsutaan yleensä konfiguraattoreiksi ja niiden tarkoitus on auttaa käyttäjä määrittely- ja kokoonpanoprosessin läpi. Vahvan asiantuntemuksen vieminen automatisoituihin

työkaluihin voi kuitenkin olla melko haastavaa. Näissä tilanteissa yhteistoiminnallisuus on välttämätöntä.

Massaräätälöinti ja komponenttipohjaisuus mahdollistavat myös ohjelmistotuotannossa verkostoitumisen toimialan muiden toimijoiden kanssa. Modulaarisuus mahdollistaa sen, että lopullinen tuote voidaan toimittaa asiakkaalle osissa siten, että eri osat tulevat eri toimittajilta [Aoyama, 1998].

Useat ohjelmistot vaativat myös erilaisia ylläpitosopimuksia ja toimittaja voi tuottaa ohjelmiston palveluna. Tällöin yhteistyö asiakkaan kanssa on tiiviimpää. Myös erilaisten konsultointipalveluiden liittäminen ohjelmiston käyttöön on mahdollista.

Ericssonilla yhteistoiminnallisuus on toteutettu siten, että eri tuotantoyksiköiden kesken on järjestetty kahdesti kuukaudessa tapaamisia [Mathiassen and Sandberg, 2014]. Näissä tapaamisissa prosessin suunnittelijat pääsevät jakamaan tietouttaan muiden kehittäjien kesken sekä muut kehittäjät pääsevät jakamaan omaa tietouttaan toisilleen [Mathiassen and Sandberg, 2014]. Ericssonilla ohjelmistokehitystä tehdään paljon rinnakkain eri toimipisteissä ja muutosten sekä oppimisen tahti on nopea. Näin ollen tapaamisissa jaettava tieto on koettu suureksi hyödyksi maantieteellisesti kaukana toisistaan olevien yksiköiden välillä.

5.5. Ketterät menetelmät ja massaräätälöinti

Hyvin suurissa ja monimutkaisissa ohjelmistokokonaisuuksissa yhteistoiminnallisuuden tueksi voidaan yhdistellä massaräätälöintiä ja ketterää ohjelmistokehitystä. Tämä voidaan kokea hyödylliseksi varsinkin projekteissa, joissa joudutaan toteuttamaan valmiiden komponenttien lisäksi paljon uniikkia toiminnallisuutta.

Ketterä ohjelmistokehitys on prosessimalli, jossa ohjelmisto pyritään saamaan mahdollisimman aikaisessa vaiheessa asiakkaan käyttöön [Pressman, 2007]. Tämä on iteratiivinen prosessi, jossa projektin eri vaiheet toteutetaan osittain samaan aikaan. Koko järjestelmää ei siis toteuteta kerralla ja toimiteta lopuksi asiakkaalle, vaan ohjelmisto pyritään ottamaan asiakkaalle käyttöön vaiheittain sitä mukaa, kun ohjelmisto valmistuu [Sommerville, 2007].

Sen lisäksi, että asiakkaan osallistuminen on yleensä jollain tasolla välttämätöntä, voidaan sillä saavuttaa myös lisäarvoa asiakkaalle. Asiakas saattaa hyvinkin kokea ohjelmistotuotteen arvokkaammaksi, mikäli hän on itse osallistunut sen toteuttamiseen. Asiakkaan osallistumisella voidaan myös sitouttaa asiakasta. Mikäli asiakkaalla on hyviä kokemuksia yhteistoiminnallisesta toteutuksesta, on todennäköistä, että hän haluaa jatkaa

asiakassuhdetta tulevaisuudessakin. Tämä on hyödyllistä varsinkin ohjelmistoperheiden tuottajille, joille pitkät asiakassuhteet ovat tärkeitä.

Ketterässä ohjelmistokehityksessä järjestelmä toteutetaan osissa. Ensin pyritään toteuttamaan järjestelmän päätoiminnallisuudet eli ne komponentit, jotka ovat ohjelmiston toiminnan kannalta asiakkaalle tärkeimpiä. Tämän jälkeen toiminnallisuuksia lisätään ohjelmistoon vaiheittain sitä mukaa, kun muut komponentit valmistuvat.

Ketterille menetelmille ominaista ovat tiivis tiimityöskentely ja asiakkaan vahva osallistuminen projektiin [Sommerville, 2007]. Tiimityöskentelyssä on tapana jakaa projekti useampiin sprintteihin, jotka kestävät yleensä viikosta muutamaan viikkoon. Sprinttien aikana voidaan kehittää eri komponentteja osittain päällekkäin. Tapana on myös päivätasolla tarkastella, miten mikäkin osa-alue etenee ja kuka tekee mitäkin. Asiakkaan osallistuminen voi olla jopa niin aktiivista, että asiakas osallistuu näihin päivittäisiin palavereihin. Tällainen toimintamalli voidaan nähdä tietynlaisena ääriesimerkkinä yhteistoiminnallisuudesta. Ehkä tunnetuin esimerkki tällaisesta projektinhallinnan viitekehyksestä on yleisesti käytetty Scrum -malli.

Ketterä ohjelmistokehitys voidaan yhdistää tehokkaasti massaräätälöintiin tuotteiden modulaarisuuden kautta. Koska ketterässä ohjelmistokehityksessä keskitytään tuotteen jakamiseen eri vaiheissa käyttöönotettaviin komponentteihin, tukee ohjelmiston modulaarisuus ja tuoteperheajattelu tätä mallia. Ohjelmistoprojekti voidaan toteuttaa siten, että yhteistyössä asiakkaan kanssa pyritään mahdollisimman aikaisessa vaiheessa ottamaan käyttöön jo valmiit olemassa olevat ydinkomponentit ja mahdollisesti tarpeeseen tulevat räätälöidyt osiot otetaan käyttöön myöhemmissä vaiheissa.

Jos asiakas ei tiedä kovin hyvin, millaisen lopullisen ratkaisun tarvitsee, voidaan ensimmäisenä ottaa käyttöön ne komponentit ja toiminnallisuudet, joiden vaatimukset tunnetaan hyvin. Tämän jälkeen muiden osioiden vaatimuksia voidaan tarkentaa ja sen myötä toimittaa toimiva ohjelmistoratkaisu. Ketterissä menetelmissä vaatimusten tarkentuminen ja muuttuminen projektin myöhemmissä vaiheissa on hyväksyttävää [Sommerville, 2007]

5.6. Esimerkkinä WordPress

Andrew Watson [2010] on käyttänyt WordPress -julkaisualustaa eräänä esimerkkinä massaräätälöidystä ohjelmistosta. WordPress on erityisen hyvä esimerkki, koska sitä voidaan käyttää hyvin erilaisiin tarkoituksiin ja koska sen arkkitehtuuri tukee

massaräätälöintiä [Watson, 2010]. WordPress on kehitetty alun perin blogi -alustaksi, mutta kehityksen myötä siitä on muodostunut kokonaisvaltainen avoimeen lähdekoodiin perustuva sisällönhallintajärjestelmä erilaisten verkkosivujen ja palveluiden julkaisemiseen [WordPress, 2015].

Pelkästään jo avoin lähdekoodi tuo massaräätälöinnin mahdollisuuden. Avoin lähdekoodi mahdollistaa sen, että muut toimijat tai käyttäjät voivat muokata lähdekoodia ominen tarpeidensa mukaiseksi. Tämän lisäksi varsinainen ydinlähdekoodi on rakennettu sellaiseksi, että sen avulla pyritään jopa rohkaisemaan muita toimijoita muokkaamaan lähdekoodia [Watson, 2010]. Rohkaisu on pyritty toteuttamaan vahvalla tuella erilaisille liitännäisille, rajapinnoille ja muokattavuudelle.

Avoin lähdekoodi on tuonut mukanaan verkostoitumisen, mikä mahdollistaa WordPress -sovelluksen kehityksen useiden eri organisaatioiden toimesta. Watson [2010] kutsuu tätä verkostoitumista ekosysteemiksi. Tällä ekosysteemillä tarkoitetaan verkostoa, joka on kehittynyt eri toimijoiden toimesta saman ohjelmistotuotteen ympärille. Ekosysteemissä esimerkiksi kehitys, tuki, ylläpito ja webhotellipalvelut voivat olla eri toimijoilla, vaikka ne ovat samaan ohjelmistoon liittyviä palveluita.

Eri toimijat voivat tarjota WordPress -sovellusta myös niin sanottuna SaaS (Software as a Service) -mallina. Tämä tarkoittaa sitä, että ohjelmisto hankitaan palveluna sen sijaan, että se hankittaisiin perinteisellä lisenssimallilla [Watson, 2010]. Tämä mahdollistaa sen, että käyttäjien ei tarvitse itse tehdä ohjelmiston asennusta tai hankkia palvelintilaa, johon ohjelmisto asennetaan.

Massatuotannon osuus WordPress -ohjelmistossa voidaan nähdä koostuvan sen kolmesta koodipohjasta, jotka ovat WordPress Classic, WordPress Multi-User ja WordPress.com [Watson, 2010]. WordPress voidaan katsoa aina perustuvan näihin koodipohjiin. Räätälöintiä WordPress -ohjelmistossa voidaan tehdä usealla eri tavalla.

Yksi räätälöinnin tapa on avoin lähdekoodi, jonka avulla kuka tahansa voi muokata ja räätälöidä WordPress -ohjelmiston koodia. Tämä on mahdollista, koska WordPress on GPL -lisenssin (GNU General Public License) alla [WordPress, 2015].

Tämän lisäksi mukautuvuus toteutuu siten, että kuka tahansa voi lisätä toiminnallisuutta ohjelmistoon tuottamalla siihen yhteensopivia liitännäisiä. Liitännäiset voivat toteuttaa räätälöityjä toiminnallisuuksia tai tuoda lisäominaisuuksia. WordPress -sovellukseen löytyy myös valmiita liitännäisiä, joita käyttäjät voivat hyödyntää omiin tarkoituksiinsa.

WordPress tukee kosmeettisuutta siten, että julkaisualustan ulkoasua voidaan muokata erilaisilla teemoilla ja muotoilemalla ohjelmiston tyylitiedostoja. Teemojen avulla voidaan toteuttaa sivuston toiminnallisuuksien sommittelu ja ulkoasun pohja. Tyylitiedostojen avulla taas voidaan tarkemmin muokata sivuston ulkoasua.

Teemojen lisäksi julkaisualustan rakennetta voidaan muokata sisällyttämällä sivustolle erilaisia pienohjelmia (widgets). Nämä pienohjelmat ovat itsenäisiä upotettuja applikaatioita, joiden toiminnallisuus on yleensä hyvin rajoitettua.

WordPress -ohjelmisto on käytössä ympäri maailmaa, joten lokalisaatioon liittyvät asetukset, kuten merkistöt ja kieliasetukset ovat muokattavissa käyttäjäkohtaisesti [Watson, 2010].

Liitännäiset ovat WordPress -sovelluksen kohdalla tehokas ja suosittu räätälöinnin tapa. Palveluun löytyy satoja tai jopa tuhansia erilaisia liitännäisiä. Kuka tahansa pystyy laajentamaan palvelunsa toiminnallisuutta liitännäisten avulla ja tuomaan liitännäiset käytettäväksi muulle WordPress -yhteisölle lisäämällä ne WordPress -palvelun liitännäiskirjastoon [Watson, 2010]. Kirjastosta on saatavilla paljon toiminnallisuutta ja lisäominaisuuksia erilaisiin tarkoituksiin, joten jokainen käyttäjä voi löytää sieltä juuri omia tarpeitaan parhaiten vastaavat liitännäiset.

Toisaalta suuresta liitännäisten valikoimasta voi olla myös haittaa joidenkin käyttäjien kohdalla. Koska samaa toiminnallisuutta toteuttavia liitännäisiä voi olla useita, voi tämä tehdä liitännäisten valinnasta hankalaa ja aiheuttaa käyttäjissä hämmennystä [Watson, 2010]. Etsittäessä tietynlaista lisäominaisuutta, voi vaihtoehtoja löytyä hyvin paljon ja sama lisäominaisuus on voitu toteuttaa usealla eri tavalla. Lisäksi voi olla vaikeaa seurata liitännäisten versioyhteensopivuutta. Voi olla mahdollista, että liitännäiset vaikeuttavat versiopäivitystä uudempaan versioon tai versiopäivitys voi rikkoa käytetyt liitännäiset [Watson, 2010]. Liitännäisten yhteensopivuudesta päivitettyyn versioon ei välttämättä ole mitään takeita, sillä liitännäisen ajan tasalla pysymisestä vastaa usein liitännäisen toteuttanut henkilö tai yhteisö.

WordPress tarjoaa kuitenkin mahdollisuuden helppoon sisällönjakamiseen ja julkaisuun vähemmän kokeneille käyttäjille. Vastaavasti todella kokeneet ja asiantuntevat käyttäjät pystyvät toteuttamaan WordPress -palvelun avulla hyvinkin monimutkaisia sivustoja ja palvelukokonaisuuksia. Tämä on suuri syy WordPress -palvelun suureen suosioon.

6. Arkkitehtuurin suunnittelu

Massaräätälöidyn ohjelmiston tärkeimpiä osa-alueita ovat sen joustava mukautuvuus ja yhteensopivuus. Jotta modulaarisuus ja mukautuvuus toteutuisivat, täytyy ohjelmiston arkkitehtuuri suunnitella ja toteuttaa huolellisesti siten, että ohjelmistokokonaisuutta on helppo muokata. Arkkitehtuurin valinta on tärkeässä asemassa, kun modulaarista ohjelmistoarkkitehtuuria lähdetään suunnittelemaan. Arkkitehtuurityylejä on monia, mutta tavoite arkkitehtuureille on hyvin samanlainen. Tarkoituksena on saavuttaa rakenneosien kuvaus, joka mahdollistaa joustavan muokattavuuden ja standardoidut riippumattomat rajapintaratkaisut. Ohjelmistoarkkitehtuurin on tarkoitus toimia toteutettavien järjestelmien karttana siten, että sen avulla voidaan toteuttaa ohjelmistotuotteet koko järjestelmäkokonaisuuden vaatimukset täyttäen [Murwantara, 2011]. Tunnettuja arkkitehtuurityylejä ovat muun muassa kerrosarkkitehtuuri ja tietovuoarkkitehtuuri.

Ohjelmistoarkkitehtuurin valinta ja suunnittelu on vaativa ja aikaa vievä prosessi, joka vaatii vahvaa asiantuntemusta. Arkkitehtuurin suunnittelu kannattaa kuitenkin tehdä huolellisesti, sillä arkkitehtuurin muokkaaminen myöhemmissä vaiheissa aiheuttaa lisäkustannuksia. Arkkitehtuurin valinnan ja suunnittelun tueksi löytyy useita erilaisia malleja [Alsawalqah *et al.,* 2013].

Arkkitehtuurin suunnittelua voidaan tehdä monella eri tasolla ja usein tämä on myös kannattavaa. Korkealla tasolla arkkitehtuuri kuvaa karkeasti, kuinka ohjelmisto on hajautettu osiin ja kuinka nämä osat vuorovaikuttavat toistensa kanssa. [Hoek and Lopez, 2011]. Alemmilla tasoilla arkkitehtuurimallit tuottavat suuntaviivat sille, kuinka modulaarinen ohjelmisto tulisi toteuttaa, jotta sen jakaminen komponentteihin ja komponenttien välinen vuorovaikutus toteutuvat oikein.

Arkkitehtuurin suunnittelussa modulaarisuus voidaan nähdä arkkitehtuurin perustana. Modulaariseen arkkitehtuuriin halutaan pyrkiä useista eri syistä. Näitä syitä ovat muun muassa parempi monimutkaisuuden hallinta, lyhemmät tuotantoajat, parempi johdonmukaisuus, tuotannon tehostaminen, parempi laatu, parempi projektien seuranta, tehokas rinnakkainen kehitystyö ja helpompi ylläpito [Brown, 2000]. Modulaarisen arkkitehtuurin pohjana voidaan nähdä komponentti ja ohjelmiston koostuminen komponenteista. Modulaarisen arkkitehtuurin suunnittelua käsitellään tarkemmin seuraavissa luvuissa.

6.1. Komponentti modulaarisen arkkitehtuurin pohjana

Ohjelmisto voidaan mieltää rakennetuksi objektiksi, kuten auto tai talo. Ohjelma koostuu samalla tavalla erillisistä osista, komponenteista, sekä erilaisista rakennusvaiheista. Kuten autolla tai rakennuksella, voidaan sanoa, että myös ohjelmalla on arkkitehtuuri. Arkkitehtuurissa on siis määritelty, mistä osista ohjelma koostuu, miten osat kommunikoivat ja mikä yhteys osilla on toisiinsa. [Meyer and Webb, 2005].

Autoissa eri komponentit voidaan ryhmitellä niiden käyttötarkoituksen mukaan. Esimerkiksi moottorin osat kuuluvat voimantuottoon ja vaihteiston osat voimansiirtoon. Mikäli nämä komponentit on tehty yhteensopiviksi, voidaan erilaisia moottoreita ja erilaisia vaihteistoja koota eri komponenteista. Samoin eri moottorit ja vaihteistot voivat olla yhteensopivia keskenään. Samaan tapaan myös ohjelmistot voidaan ryhmitellä komponentteihin ja yhdistellä erilaisia komponentteja. Nämä komponentit yhdessä muodostavat ohjelmistotuotteen määritellyn arkkitehtuurin pohjalta.

Modulaarisessa arkkitehtuurissa kaiken pohjana on siis komponentti, mutta komponentille ei ole olemassa yhtä yksiselitteistä määritelmää [Brown, 2000]. Ohjelmistokomponentti voidaan kuitenkin nähdä itsenäisenä yksikkönä, joka toteuttaa yhtä tai useampaa sille määriteltyä tehtävää. Tärkeintä on, että jako komponentteihin on selkeä siten, että komponentit ovat riippumattomia toisistaan. On myös tärkeää, että komponentit toteuttavat niille määritellyn toiminnallisuuden yksiselitteisesti. Alan Brown [2000] on määritellyt komponentin seuraavalla tavalla:

”Itsenäisesti toimitettava funktionaalisuuden osa, joka mahdollistaa pääsyn sen palveluihin rajapintojen avulla.”

Tämän määrittelyn pohjalta huomataan, että komponentti on itsenäisesti toimitettava yksikkö. Lisäksi määritelmän mukaan komponentti toteuttaa määriteltyä toiminnallisuutta ja yhdessä muiden komponenttien kanssa se voi toteuttaa joukon toiminnallisuuksia muodostaen ohjelmistokokonaisuuden. Komponentti myös mahdollistaa sen palveluiden käytön rajapintojen avulla. Näin rajapinnat ovat oleellinen osa komponenttien välistä kommunikointia. Rajapintojen käyttö mahdollistaa sen, että komponenteilla ei ole pääsyä toistensa sisäisen toteutuksen yksityiskohtiin [Brown, 2000]. Näin kommunikointi voidaan pitää standardoituna ja se mahdollistaa komponenttien riippumattomuuden toisistaan.

6.2. Modulaarinen arkkitehtuuri

Modulaarisen ohjelmistotuotannon suurimpia haasteita ovat kuhunkin tarkoitukseen parhaimpien komponenttien ja toteutustapojen löytäminen. Arkkitehtuurin suunnitteluun ja toteutukseen on useita eri tapoja, ja erilaisilla ratkaisumalleilla voidaan saavuttaa erilaiset hyödyt ja lopputulokset [Hoek and Lopez, 2011]. Näin ollen arkkitehtuurin suunnittelu täytyy miettiä ja toteuttaa aina tapauskohtaisesti.

Toteutustapa ja arkkitehtuurin valinta ovat yleensä määrättyjen ohjelmistoarkkitehtien päätettävissä. Arkkitehtuuritapaa valittaessa on tärkeää huomioida, että koko kehityksen tulee sitoutua samaan arkkitehtuuriin [Hoek and Lopez, 2011]. Kun arkkitehtuurimalli on valittu, ei arkkitehtuurista yleensä pidä poiketa millään osa-alueella.

Modulaarisen arkkitehtuurin pohjana on komponentti- ja rajapintapohjaisen toteutuksen tukeminen [Brown, 2000]. Tätä voidaan tukea muun muassa määritellyillä komponenttikirjastoilla, ennakkoon standardoiduilla rajapinnoilla ja erilaisilla komponenttimalleilla [Brown, 2000].

Komponenttipohjaisen arkkitehtuurin perustana nähdään ohjelman suunnitteleminen ja toteuttaminen komponentteihin pohjautuen [Brown, 2000]. Arkkitehtuurin tavoitteena on siis saavuttaa malli, jossa komponentteja yhdistelemällä voidaan toteuttaa ohjelmistoja, jotka palvelevat niille asetettuja kaupallisia tarkoituksia. Lisäksi tavoitteena on saavuttaa kustannustehokas kehitysprosessi, jolla voidaan taata ohjelmistojen hyvä laatu.

Kaikkien tarvittavien arkkitehtuurin osa-alueiden määritteleminen ei alkuvaiheessa tietenkään ole mahdollista, vaan ne tulevat tarkentumaan myöhemmin. Suunnittelun pohjana kuitenkin on, että ohjelmistotuotteiden täytyy pystyä mahdollisimman tehokkaasti käyttämään yhteisiä komponentteja ja toimia mahdollisesti myös yhdessä muiden ohjelmistotuotteiden kanssa. Alkuvaiheen arkkitehtuuri koostuukin yleensä tunnetuista ohjelmisto-osista, joita voidaan käyttää uudelleen [Brown, 2000]. Lisäksi alkuvaiheen arkkitehtuuri sisältää yleensä tutut toimintamallit, mahdolliset valitut standardit ja komponenttien väliset vuorovaikutustavat [Brown, 2000].

Arkkitehtuuri pyritään alkuvaiheessa määrittelemään ainakin kahdella tasolla, jotka ovat loogisten komponenttien arkkitehtuuri ja fyysisten komponenttien arkkitehtuuri [Brown, 2000]. Loogisten komponenttien arkkitehtuurin avulla pyritään jo alkuvaiheessa määrittelemään, mitä toiminnallisuutta komponenttien on tarkoitus toteuttaa. Tämän lisäksi loogisten komponenttien arkkitehtuurin avulla pyritään määrittelemään, mitä

palveluita komponentit tarjoavat rajapintojen avulla ja miten toiminta vastaa käyttötarkoitusta [Brown, 2000]. Loogisten komponenttien arkkitehtuuria voidaan pitää järjestelmän pohjapiirroksena ja sen testaaminen jo alkuvaiheessa on tärkeää, koska se määrittelee suunnan koko kehitystyölle.

Fyysisten komponenttien arkkitehtuuri esittää järjestelmän suunnittelun fyysisellä tasolla. Toisin sanoen se kuvailee teknisen infrastruktuurin sisältäen mahdollisesti käytetyt laitteistot, topologiat sekä käytetyt verkko- ja kommunikaatioprotokollat [Brown, 2000]. Arkkitehtuurin avulla pyritään saamaan kuva siitä, kuinka nämä kaikki osakokonaisuudet voidaan sitoa yhteen. Lisäksi fyysisten komponenttien arkkitehtuurin avulla pyritään ymmärtämään laatuvaatimuksia, kuten suorituskykyä ja toimintavarmuutta [Brown, 2000].

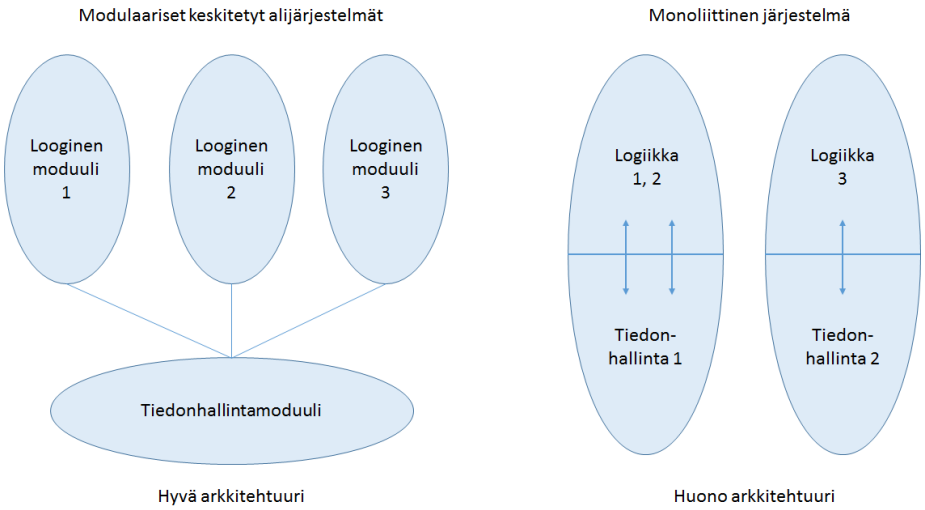
Arkkitehtuurin dokumentointi on tärkeää, jotta eri yksiköiden ja toimijoiden on helppo käsitellä ja muokata eri komponentteja. Komponentit kannattaa esimerkiksi säilyttää komponenttikirjastossa, josta niitä on helppo hyödyntää uudelleenkäytettäviksi [Li *et al.*, 2006]. Myös liittymät ja arkkitehtuurin eri osien integraatioiden dokumentointi on syytä toteuttaa huolellisesti.

Useat ohjelmistot voivat muodostaa keskenään massiivisia tietojärjestelmiä. Suuret monimutkaiset tietojärjestelmät kannattaa jakaa erillisiin alijärjestelmiin, joissa jokaisella itsenäisellä alijärjestelmällä on yksi keskitetty tarkoitus. Monimutkaisten järjestelmien jakaminen keskitettyihin alijärjestelmiin helpottaa monimutkaisuuden hallintaa ja muutosten tekemistä järjestelmään pitkällä aikavälillä. [Meyer and Webb, 2005]

Keskittäminen tarkoittaa sitä, että jokaiselle alijärjestelmälle on yksi tehtävä, jonka se voi toteuttaa ilman, että se tarvitsee siihen muita alijärjestelmiä. Alijärjestelmän toiminnan ei pidä myöskään aiheuttaa muutoksia tai toimenpiteitä muissa alijärjestelmissä. Keskittäminen mahdollistaa sen, että järjestelmän monimutkaisuutta on helpompaa hallita [Meyer and Webb, 2005]. Myös muutokset alijärjestelmiin ovat helpompia, koska muutoksilla ei ole suoranaista vaikutusta muihin alijärjestelmiin. Näin myös luonteva alijärjestelmien lisääminen tai poistaminen on mahdollista ilman ongelmia järjestelmän toimivuudessa. Alijärjestelmät voidaan mieltää rinnastettavaksi moduuleihin tai komponentteihin, mutta ne ovat suurempia kokonaisuuksia.

Moduulien ja komponenttien ei tulisi itsessään sisältää sellaista toiminnallisuutta, jota muut komponentit tai moduulit voivat tarvita. Niiden ei myöskään tulisi sisältää sellaista tietoa tai tiedonhallintaa, jota muut moduulit voivat tarvita. Niin sanottu perustieto (Master data) tulisi sijaita yhdessä paikassa ja moduuleiden pääsyä tietoon

tulisi hallinnoida keskitetysti. Myös moduuleiden keskinäinen kommunikointi kannattaa toteuttaa keskitetysti. Kuvassa 8 on esitetty esimerkit hyvästä ja huonosta tiedonhallinta-arkkitehtuurista.



Kuva 8. Keskitetty alijärjestelmäarkkitehtuuri [Meyer and Webb, 2005].

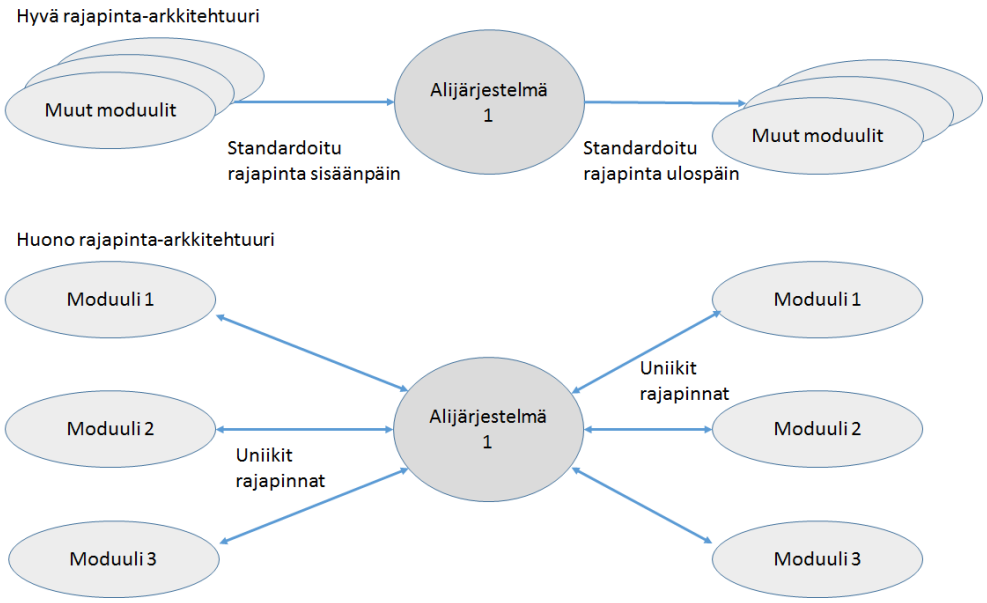
Rajapintojen suunnittelu on osa hyvän arkkitehtuurin omaavan ohjelmiston suunnittelua. Rajapintojen avulla ohjelman eri moduulit voivat kommunikoida standardoidusti keskenään ja muiden ohjelmien kanssa. Brown [2000] on määritellyt rajapinnan seuraavalla tavalla:

”Rajapinnat ovat mekanismeja, joiden avulla komponentit kuvailevat niiden toiminnallisuuden ja mahdollistavat pääsyn niiden tarjoamiin palveluihin.”

Standardoitujen rajapintojen avulla komponenttien ei tarvitse tuntea toistensa sisältöä tai toteutusta. Komponentin toteutus voi olla täysin piilotettu muilta komponenteilta ja ainoastaan rajapinta kertoo kaiken tarvittavan komponentin toiminnallisuudesta [Brown, 2000]. Tämä mahdollistaa sen, että komponenttien välille ei muodostu riippuvuuksia ja komponenttien vaihtaminen, lisääminen ja poistaminen on helpompaa.

Jokaisella moduulilla tai alijärjestelmällä tulisi aina olla oma standardoitu rajapinta tiedon saapumiseen ja tiedon lähtemiseen [Meyer and Webb, 2005]. Arkkitehtuurissa tulisi välttää sellaista mallia, jossa jokaisella moduulilla tai alijärjestelmällä on omat uniikit rajapintansa. Tämä malli aiheuttaa helposti tilanteen, jossa muutosten yhteydessä joudutaan tarkastamaan kaikkien rajapintojen toimivuus, mikäli esimerkiksi yksi moduuli vaihdetaan [Meyer and Webb, 2005]. Mikäli kaikki rajapinnat toteutetaan keskitetysti ja

standardoidusti, voidaan luottaa siihen, että ne toimivat myös muutosten jälkeen. Esimerkit hyvästä ja huonosta rajapinta-arkkitehtuurista on esitetty kuvassa 9.



Kuva 9. Rajapinta-arkkitehtuuri [Meyer and Webb, 2005].

6.3. Tuotealustan ja tuoteperheen arkkitehtuuri

Suunniteltaessa tuotealustan ja tuoteperheen arkkitehtuuria arkkitehtuurimallin valinta on tärkeässä osassa. Arkkitehtuurimallille oleellista on mahdollistaa saman tuotealustan ja jaettujen komponenttien käyttö eri ohjelmistotuotteissa. Valitun arkkitehtuurimallin ja työkalujen pohjalta voidaan muodostaa tuoteperheen arkkitehtuuri ja tuotealustan arkkitehtuuri.

Tuoteperheen arkkitehtuuri ja tuotealustan arkkitehtuuri kannattaa mieltää toisistaan erilleen. Näin alustan arkkitehtuurin suunnittelussa voidaan keskittyä siihen, miten alustassa otetaan mahdollisimman hyvin huomioon komponenttien uudelleenkäyttö ja yhteensopivuus [Wijnstra, 2002]. Alustan suunnittelussa täytyy ottaa myös huomioon tulevaisuudessa mahdollisesti tulevat muutokset ja esimerkiksi uudet teknologiat. Alustan on tarkoitus toimia ohjelmistoperheen kaikkien tuotteiden perustana [Alsawalqah *et al.*, 2013].

Tuoteperheen arkkitehtuurin suunnittelussa voidaan puolestaan keskittyä jaettuun arkkitehtuuriin tuoteperheen jäsenten kesken [Wijnstra, 2002]. Tämä siis tarkoittaa käytännössä niiden vaatimusten, toiminnallisuuksien ja sääntöjen määrittelyä, jotka tuoteperheen jäsenten täytyy toteuttaa, jotta ne olisivat tehokas osa tuoteperhettä. Näiden määrittelemiseen on syytä käyttää resursseja, sillä niiden muuttaminen myöhemmissä

vaiheissa voi olla haastavaa ja aiheuttaa kustannuksia [Wijnstra, 2002]. Jotta toimiva tuoteperhearkkitehtuuri voidaan toteuttaa, vaatii se pohjalle toimivan alustan arkkitehtuurin [Alsawalqah *et al.,* 2013].

Tuoteperheen ja tuotealustan arkkitehtuurin suunnittelussa on jo alkuvaiheessa tärkeää pyrkiä kartoittamaan eri tuotteita yhdistävät osa-alueet ja komponentit [Wijnstra, 2002]. On siis oleellista löytää nimenomaan kaikille tuotteille yhteiset osat. Suunnittelussa on myös tärkeää löytää ne yhteiset säännöt ja vaatimukset, jotka kaikkien yhteisten komponenttien täytyy toteuttaa. Mikäli näitä sääntöjä ja yhteisiä osa-alueita ei määritellä jo alkuvaiheessa, on mahdollista, että tuotteista tulee hyvin uniikkeja ja ne hyödyntävät huonosti tuotealustan tarjoamaa komponenttien uudelleenkäyttöä. Tällöin tuoteperhemallin tuomat hyödyt jäävät vähäisiksi.

Yksittäisen tuotteen ohjelmistoarkkitehtuuri toimii karttana ja ohjeena sille, kuinka ohjelmisto toteutetaan. Samaan tapaan myös ohjelmistoperheen arkkitehtuurin on tarkoitus olla karkea kuvaus siitä, kuinka ohjelmistoperheen tuotteet rakenteellisesti toteutetaan [Murwantara, 2011]. Se helpottaa uusien tuotteiden kehitystä sekä olemassa olevien tuotteiden jatkokehitystä ja ylläpitoa.

Alustan arkkitehtuurin suunnittelussa täytyy päättää, mitkä osat tuotteista kuuluvat alustan sisältöön. Tuoteperheeseen voi kuulua aivan kaikki komponentit, sellaisetkin, jotka ovat käytössä vain yhden tuotteen kohdalla [Wijnstra, 2002]. Toisaalta alustan sisältöön voi kuulua esimerkiksi vain pieni määrä sellaisia komponentteja, jotka on jaettu kaikkien tuotteiden kesken.

Arkkitehtuuri täytyy suunnitella tiukasti määriteltyjen sääntöjen ja standardien mukaan. Tämä on ainut tapa, miten monimutkaiset ja suuret ohjelmistokokonaisuuksien osat voidaan tehdä yhteensopiviksi. Myös kaikki tuoteperheeseen sisältyvät moduulit ja komponentit täytyy tehdä näiden sääntöjen ja standardien mukaisesti [Meyer and Webb, 2005]. Mikäli komponentit ja liittymät toteutetaan sääntöjä ja standardeja noudattamatta, syntyy huomattava riski, että muutokset ohjelmistoon rikkovat ohjelmiston [Meyer and Webb, 2005]. Jotta tämä riski voidaan minimoida, täytyy kaikki ohjelmistoperheen osiot tehdä samojen tiukasti määriteltyjen sääntöjen ja standardien mukaisesti ja nämä täytyy myös dokumentoida hyvin.

Tuoteperhe tulisi suunnitella siten, että tuotealustan arkkitehtuuri määrittää myös tuotteiden arkkitehtuurin [Wijnstra, 2002]. Tuotteet voidaan dokumentoida esimerkiksi tuottamalla lista tuotteessa käytetyistä komponenteista, jolloin komponenttien yhteys

toiminnallisuuteen on helpompi hahmottaa [Wijnstra, 2002]. Tämä helpottaa tehokasta komponenttien uudelleenkäyttöä.

Hyvin toteutettu tuoteperhearkkitehtuuri mahdollistaa modulaarisuuden kaikkien tuoteperheen komponenttien välillä [Meyer and Webb, 2005]. Tämä tarkoittaa sitä, että erilaisia valmiita ohjelmistoja voidaan toteuttaa yhdistelemällä valmiita komponentteja eri tavalla ilman, että ohjelmistoon tarvitsee lisätä erillistä koodia [Meyer and Webb, 2005]. Hyvin suunniteltu arkkitehtuuri edellyttää myös, että ohjelmistokomponentit täytyy suunnitella itsenäisiksi kokonaisuuksiksi, joissa komponentin toiminta tai toimimattomuus ei suoraan vaikuta muihin komponentteihin. Näin komponentteja voidaan korvata, poistaa, lisätä tai vaihtaa ilman, että ohjelmiston toiminta heikkenee.

Tällainen on ideaali arkkitehtuuri ohjelmistotuotteille ja tuoteperheille, mutta näin joustavan arkkitehtuurin toteuttaminen käytännössä on hyvin haastavaa. Useissa tapauksissa tarvitaan niin uniikkia asiakaskohtaista räätälöintiä, että asiakaskohtaisen koodin liittäminen ohjelmistoon on pakollista. Kuitenkin toteuttamalla hyvät liittymät ja muokattava arkkitehtuuri on mahdollista, että tarvittava räätälöinti voidaan tehdä myös asiakkaan päässä erilaisilla aliohjelmilla tai liitännäisillä [Meyer and Webb, 2005].

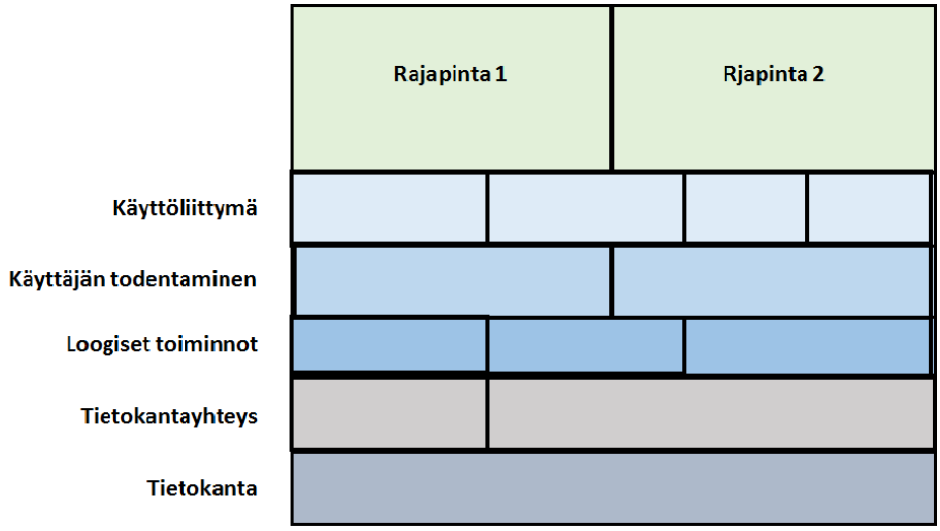
Koska ohjelmistotuotannossa toteutetaan usein samankaltaisia tuoteperhemalliin liittyviä tuotteita, on tuotelinjojen käyttö monissa tapauksissa järkevää [Meyer and Webb, 2005]. Tällöin ohjelmistoarkkitehtuurissa tulee ottaa huomioon tuotelinjojen tuomat vaatimukset. Arkkitehtuuri täytyy myös suunnitella niin, että tuotelinjat, komponentit, prosessit ja tuotteet voivat kehittyä [Meyer and Webb, 2005]. Tuotelinjan arkkitehtuuria käsitellään tarkemmin luvussa 6.5.

6.4. Kerrosarkkitehtuuri

Modulaarinen arkkitehtuuri on järkevää jakaa erillisiin kerroksiin. Ohjelmiston arkkitehtuuri voidaan jakaa kerroksiin esimerkiksi sen funktionaalisten toiminnallisuuksien näkökulmasta. Voidaan esimerkiksi määritellä kerrosarkkitehtuurin koostuvan tietokantakerroksesta, reaaliaikaisen tiedon kerroksesta, loogisesta kerroksesta ja graafisen käyttöliittymän kerroksesta.

Kerrosarkkitehtuurin ajatuksena on jakaa eri tason toiminnallisuudet eri kerroksiin siten, että kerrosten välillä toimii riippumattomat rajapinnat. Tällöin eri kerrokset eivät välttämättä ole riippuvaisia toisistaan ja osioita kerroksissa voidaan vaihtaa ilman, että se vaikuttaa suoraan muihin kerroksiin. Useimmiten kerroksissa on kuitenkin jonkin asteisia riippuvuussuhteita toiseen suuntaan, mutta ei mieluusti molempiin suuntiin. Kerros voi

siis olla esimerkiksi riippuvainen alemmasta kerroksesta, mutta silloin sillä ei tulisi olla riippuvuuksia ylempään kerrokseen. Kerroksiin jakaminen mahdollistaa myös mallin, jossa eri yksiköiden osaaminen voi keskittyä eri kerroksiin ja eri kerrokset voivat sisältää eri teknologiaa [Meyer and Webb, 2005]. Esimerkki kerroksiin jaetusta arkkitehtuurista on esitetty kuvassa 8.



Kuva 10. Kerrosarkkitehtuuri.

Kuvassa 8 on esitetty esimerkki kerrosarkkitehtuurista, jossa omat kerroksensa muodostavat tietokanta, tietokantayhteydet, loogiset toiminnot, käyttäjän todentaminen, käyttöliittymä ja rajapinnat. Kuvassa kerrokset on jaettu vielä erillisiin laatikkoihin, mikä kuvastaa sitä, että kerrokset voivat koostua useammista komponenteista. Jokaisen kerroksen tulisi sisältää joukko komponentteja, joilla on jokin oma tietty tehtävänsä ja jotka pystyvät kommunikoimaan muiden komponenttien kanssa [Meyer and Webb, 2005].

On mahdollista, että jokin komponentti voi ylittää kerrosrajat eli komponentti voi kuulua kahteen tai useampaan eri kerrokseen. Kerros voi sisältää yhden komponentin tai sillä voi olla useita komponentteja. Komponenttien väliset liittymät tulee toteuttaa standardoidusti siten, että komponentit pystyvät kommunikoimaan saman kerroksen sisällä sekä eri kerrosten välillä [Meyer and Webb, 2005].

Kerrosarkkitehtuuri voidaan myös suunnitella niin, että kerrokset muodostetaan muunneltavuuden mukaan siten, että muunneltavat tai korvattavat komponentit ovat ylemmissä kerroksissa ja vakaat yhteiset komponentit sijaitsevat alemmissa kerroksissa [Michiels et al., 2003]. Tämä helpottaa muunneltavuutta eri tuotteissa ja parantaa tuotteiden vakautta. Tämä myös mahdollistaa sen, että muutokset ylemmillä tasoilla eivät aiheuta muutoksia alemmilla tasoilla.

6.5. Tuotelinjan arkkitehtuuri

Tuotelinjoille löytyy useita eri määritelmiä ohjelmistotuotannossa, mutta pohjimmiltaan tuotelinjan on tarkoitus tarjota infrastruktuuri modulaaristen ohjelmistojen ja ohjelmistoperheiden tuottamiseen. Tuotelinja ohjelmistotuotannossa tarjoaa abstraktin näkymän kaikkien tuoteperheen tuotteiden rakenteelle [Murwantara, 2011]. Tuotelinjan voidaan katsoa sisältävän koko ohjelmistotuotantoon tarvittavan infrastruktuurin mukaan lukien muun muassa mallit, komponentit, arkkitehtuurit ja muut resurssit [Mohabbati *et al.*, 2011]. Hyvin suunniteltua tuotelinjaa hyödyntämällä ohjelmistotuotannossa voidaan saavuttaa tehokkaampi komponenttien uudelleenkäyttö, lyhemmät tuotantoajat, pienemmät tuotantokustannukset ja parempi ohjelmiston laatu [Mohabbati *et al.*, 2011]. Avaintekijä onnistuneen tuotelinjan toteutukseen on ohjelmistotuotteiden samankaltainen tuotanto, jossa uudelleenkäyttöä hyödynnetään tehokkaasti [Murwantara, 2011].

Kuten tuotelinjoissa yleensäkin, myös ohjelmistotuotannossa tuotelinjojen lähestymistapa voi olla ylhäältä-alas tai alhaalta-ylös -mallia. Tuotelinjaa voidaan lähteä kehittämään erilaisten ohjelmistojen yhteisten osien pohjalta tai sitten tuotteet voidaan kehittää suunniteltuun tuotelinjaan pohjautuen.

Olipa lähestymistapa kumpi tahansa, kattavan vaatimusmäärittelyn tekeminen tuoteperheelle ja tuotelinjalle on tärkeää. Vaatimusmäärittelyssä kannattaa hyödyntää erilaisia mallintamismenetelmiä, joista tunnetuimpia ovat muun muassa ominaisuusmallit ja komponenttipohjainen mallintaminen [Murwantara, 2011]. Tärkeää on myös tehdä kattava analyysi sovellus- ja markkina-alueella, jotta voidaan selvittää, onko tuotelinjan toteuttaminen yleisesti ottaen järkevää. Mikäli tuotteet ovat todella uniikkeja tai markkinat hyvin pienet, ei tuotantolinjan toteuttaminen ole välttämättä kannattavaa. Sovellus- ja kannattavuusanalyysien toteuttamiseen löytyy useita erilaisia valmiita malleja ja prosesseja.

Mikäli tuotelinja päätetään toteuttaa, on kattavan vaatimusmäärittelyn tekeminen tuoteperheelle ja tuotealustalle erittäin tärkeää. Vaatimusmäärittelyssä pyritään määrittelemään kaikki oleelliset laadulliset ja toiminnalliset vaatimukset, jotka tuotteiden tulee toteuttaa. Kuten sovellusanalyysien ja kannattavuusanalyysien tekoon, myös vaatimusmäärittelyn tekoon löytyy useita erilaisia malleja ja mallin valinta riippuu hyvin paljon organisaatiosta ja sovellusalueesta [Sommerville, 2007].

Kun kattava vaatimusmäärittely on tehty, voidaan toteuttaa komponenttien luokittelu. Tässä vaiheessa pyritään määrittelemään ne komponentit, jotka kuuluvat tuotealustaan sellaisinaan. Yleisesti ottaen tuotealustan arkkitehtuuri voi sisältää

kahdenlaisia komponentteja, jotka ovat yhteiset komponentit ja muunneltavat komponentit [Murwantara, 2011]. Tässä vaiheessa päätetään niin sanotusti tuotteiden samankaltaisuudesta [Alsawalqah *et al.,* 2013]. Yhteiset tuotealustan ohjelmistokomponentit muodostavat pohjan kaikille tuoteperheen ohjelmistotuotteille. Muunneltavat komponentit taas muodostavat pohjan muunneltavuudelle eri ohjelmistotuotteiden välillä [Murwantara, 2011]. Tämä muodostaa niin sanotun ohjelmistoalustan. Luokittelussa voidaan myös päättää, mitkä komponentit kuuluvat varsinaiseen tuotealustaan ja mitkä eivät [Alsawalqah *et al.,* 2013].

Komponenttien luokittelun lisäksi tuotelinjan suunnittelussa määritetään komponenttien väliset suhteet sekä komponenttien suhteet muihin järjestelmiin. Tuotteiden liittymät pohjautuvat siis tuotealustan määrityksiin. Tuotelinjan arkkitehtuurin tulee ottaa huomioon komponenttien muunneltavuus, komponenttien väliset suhteet, sekä komponenttien toiminnallisuus [Murwantara, 2011]. Tuotelinjan sisältämien tuotteiden muunneltavuuden sisällyttäminen tuotelinjan arkkitehtuuriin onkin yksi tuotelinjan suurimmista haasteista.

Vaikka tuotelinjan voidaan käsittää sisältävän koko kehitykseen vaadittavan infrastruktuurin, keskittyy tuotelinjan arkkitehtuuri ohjelmistotuotannossa yleensä tuotelinjan tekniseen arkkitehtuuriin. Tietojärjestelmätuotelinjan arkkitehtuuri on yhdistelmä alijärjestelmiä ja rajapintoja, jotka toimivat perustana tuotteiden sarjalle [Meyer and Webb, 2005]. Tuotelinja tulee suunnitella sellaiseksi, että se tukee joustavuutta ja tehokkuutta tuotannossa. Tämä on mahdollista toteuttamalla hyvät standardoidut rajapinnat alustaan omien alijärjestelmien välille sekä muiden järjestelmien suuntaan.

Tuotelinjan arkkitehtuurin täytyy sallia muutokset. Uusien moduulien lisääminen ja liittäminen muihin moduuleihin rajapintojen avulla tulee olla joustavaa ja helppoa ilman, että laatuvaatimuksista joudutaan tinkimään. Modulaarinen tuotelinjan arkkitehtuuri sallii järjestelmän kehittyvän organisoidusti ja tehokkaasti sitä mukaa, kun sen eri osiin tehdään muutoksia [Meyer and Webb, 2005]. Näiden muutosten ei tulisi vaikuttaa suoraan muihin ohjelmiston osiin. Myös rajapintojen muutoksia ja kehitystä täytyy pystyä tekemään siten, että se ei vaikuta muiden rajapintoja käyttävien moduulien toimintaan.

Microsoftin Windows -käyttöjärjestelmistä löytyy hyvä esimerkki toimivasta plug and play -tyyppisestä PCI -väylän rajapintaratkaisusta [Meyer and Webb, 2005]. Tässä ratkaisussa käyttäjä voi dynaamisesti lisätä laitteita tietokoneeseen ilman, että se

vaikuttaa muiden laitteiden toimintaan. Tietokoneessa voi olla esimerkiksi kiinni useita eri syöttö- ja näyttölaitteita sekä kokoonpanon komponentit. Tietokoneeseen voidaan liittää koneen käynnissä ollessa ulkoinen kovalevy USB -porttiin ja kovalevy tulee käytettäväksi ilman, että se vaikuttaa muiden komponenttien toimintaan.

Plug and play -tyyppiset ratkaisut eivät koske pelkästään fyysisiä laitekomponentteja, vaan myös ohjelmistokomponentit voidaan toteuttaa toimimaan plug and play -periaatteella. Tällöin tarkoituksena on, että ohjelmistokomponentteja voidaan liittää ohjelmistokokonaisuuden toimintaan ajon aikaisesti ilman, että ohjelman toimintaa tarvitsee pysäyttää [Aoyama, 1998].

Hyvin toteutetun tuotelinjan avulla voidaan vähentää testaamisen tarvetta ja parantaa ohjelmiston laatua. Tuotelinjan ja sen arkkitehtuurin testaaminen on kuitenkin tärkeää ja tuotelinjan arkkitehtuuria tulisi testata aina, kun siihen tehdään muutoksia. Näin voidaan varmentaa tuotelinjan hyvän laadun säilyminen ja laatuvaatimusten täyttyminen myös muutosten jälkeen.

Testaamisen lisäksi tuotelinjan arkkitehtuurin ja sen muutosten dokumentointi on tärkeää. Dokumentoinnin avulla pystytään tehostamaan linjan käyttöä sekä saavuttamaan parempi ymmärrys tuotelinjan osien toiminnallisuuksista ja osien välisistä yhteyksistä. Myös muutosten dokumentointi on tärkeää, jotta muutosten jäljitettävyys säilyy.

Vaikka tuotelinjaa testataan muutosten yhteydessä, voi olla mahdollista, että tuotelinjaan tulee ei-toivottuja ominaisuuksia tai laadun heikkenemistä. Tehtäessä muutoksia arkkitehtuuriin tulisi aina testata, että laatuvaatimusten täyttyminen arkkitehtuurissa säilyy entisellä tasollaan. Muutokset arkkitehtuurissa ja uudet riippuvuudet voivat helposti aiheuttaa muutoksia tietoturvassa, suorituskyvyssä ja suoritusajassa [Murwantara, 2011]. Näissä tapauksissa on pystyttävä jäljittämään muutokset komponenttitasolle, jotta muutosten aiheuttamat ongelmat olisi helppo korjata.

Tuotelinjan arkkitehtuuri voidaan suunnitella täysin komponenttipohjaisen kehityksen näkökulmasta. Komponenttipohjaisten tekniikoiden tavoitteena on, että tuote syntyy valmiita komponentteja yhdistelemällä ja uudelleenkäyttämällä [Li *et al.*, 2006] Komponenttipohjaiset ohjelmistokehitystekniikat tuovat kuitenkin oman haasteensa tuotelinjan arkkitehtuurin suunnitteluun. Komponentit toteutetaan vastaamaan funktionaalisia vaatimuksia, mutta niiden kaikkia uudelleenkäyttömahdollisuuksia ei kehitysvaiheessa vielä tunneta.

Jokainen komponentti vaati aina oman määränsä resursseja toimiakseen. Jotkut komponenttiyhdistelmät voivat viedä niin paljon resursseja, että ohjelmiston

laatuvaatimukset eivät täyty [Li *et al.*, 2006]. Tuotelinjan arkkitehtuuri tulee olla siinä määrin testattava, että laatuvaatimusten täyttymistä pystytään seuraamaan.

Kun uusia tuotteita kehitetään, täytyisi kehitys aloittaa kokoonpanemalla tuotelinjan yhteiset ja muokattavat komponentit, jotka tuotteeseen tarvitaan. Tuotelinjan arkkitehtuurin pitäisi jo tässä vaiheessa mahdollistaa laatuvaatimusten testaaminen valittujen komponenttien osalta [Li *et al.*, 2006]. Laatuvaatimukset voivat olla esimerkiksi suoritusaika ja muistinkäyttö. Kun nämä on testattu, voidaan lähteä suunnittelemaan kyseisen tuotteen uniikit osat, jonka jälkeen tuotekokonaisuus voidaan jälleen testata.

6.6. Palvelukeskeinen arkkitehtuuri

Palvelukeskeinen arkkitehtuuri on malli, joka on edistänyt huomattavasti ohjelmistojen modulaarisuutta ja massaräätälöinnin mahdollisuuksia. Palvelukeskeinen arkkitehtuuri on arkkitehtuurimalli, jossa sovellukset toteutetaan jonkin toiminnallisuuden ympärille itsenäisiksi palveluiksi [Verdouw *et al.*, 2014]. Nämä palvelut suunnitellaan joustaviksi siten, että ne sisältävät standardoidut rajapinnat. Nämä rajapinnat mahdollistavat sen, että muut palvelut voivat käyttää niitä verkon, esimerkiksi internetin, yli. Näin palveluiden välille voidaan rakentaa verkosto, josta useammat palvelut muodostavat palvelukokonaisuuden, jossa jokainen palvelu on oma moduulinsa tai alijärjestelmänsä.

Palvelukeskeistä arkkitehtuuria hyödynnetään muun muassa SaaS -mallin (Software as a Service) ohjelmistoratkaisuissa, joissa ohjelmistoa ei osteta asennettavana instanssina tai lisenssinä. Sen sijaan ohjelmisto tarjotaan yleensä palveluna verkossa esimerkiksi niin sanottuna pilvipalveluna. Tällöin eri käyttäjät voivat käyttää pilvipalveluna toteutettua palvelua internetyhteyden yli esimerkiksi verkkoselaimella.

Palvelukeskeisessä arkkitehtuurissa pyrkimyksenä on rakentaa palvelut toisistaan riippumattomiksi ja itsenäisiksi. Näin ollen palvelut eivät välttämättä tarvitse toisia palveluita toimiakseen. Toisaalta palvelut pyritään rakentamaan teknologiariippumattomiksi siten, että ne eivät ota kantaa siihen, millä teknologialla toinen palvelu on toteutettu. Tämä on mahdollista toteuttaa standardoituja rajapintoja, kuten Web Services -rajapintoja käyttämällä.

Useammat palvelut voivat kommunikoida keskenään lähettämällä ja vastaanottamalla viestejä toisiltaan ja sen avulla ne voivat käyttää muiden palvelukirjastoissa julkaistuja palveluita [Verdouw *et al.*, 2014]. Palvelukeskeisessä

arkkitehtuurissa itsenäisten sovellusten käyttäjiä voivat siis olla joko ihmiset tai muut sovellukset.

Web Service -esimerkkejä ovat muun muassa erilaiset sääpalvelut, verkkopankkipalvelut ja lippuvarausjärjestelmät [Sam *et al.,* 2006]. Web Service -toteutukset perustuvat kolmeen standardoituun teknologiaan, jotka ovat WSDL, UDDI ja SOAP. Nämä teknologiat mahdollistavat standardoitujen rajapintojen hyödyntämisen erilaisten palveluiden välillä riippumatta sovelluksen teknologiasta.

WSDL (Web Service Description Languag) on XML -pohjainen kieli, jonka avulla voidaan kuvata Web service -palveluiden toiminnallisuutta [W3C, 2015a]. UDDI (Universal Description Discovery and Integration) puolestaan on teknologia- ja alustariippumaton palvelurekisteristandardi, jolla palvelukeskeinen arkkitehtuuri voidaan toteuttaa [W3C, 2015b]. Rekisteri sisältää hakemiston, jossa on tarvittavat tiedot erilaisista rekisteröidyistä verkkopalveluista. Sovellukset voivat käyttää näitä tietoja hyödyntääkseen rekisterissä olevia palveluja. SOAP (Simple Object Access Protocol) on XML -pohjainen tietoliikenneprotokolla, jonka tarkoituksena on toteuttaa proseduurien etäkutsut [W3C, 2015c]. SOAP -protokollaa voidaan käyttää useiden protokollien yli, mutta yleensä sitä käytetään http -protokollan yli. Yhdessä nämä kolme teknologiaa mahdollistavat Web service -ratkaisujen toteuttamisen.

6.7. Mallintaminen

Massaräätälöitävyys ei tarkoita rajatonta määrää vaihtoehtoja, vaan se rajoittuu määrättyyn joukkoon valittavissa olevia vaihtoehtoja [Verdouw *et al.,* 2014]. Jotta nopea ja tehokas käyttöönotto on mahdollista toteuttaa, ovat nämä vaihtoehdot käytävä selvästi ilmi. Tämä on tärkeää varsinkin silloin, kun asiakkaan on tarkoitus tehdä kokoonpanovaihe itsenäisesti. Erilaisten vaihtoehtojen, riippuvuuksien ja luokittelujen esittämisessä voidaan hyödyntää erilaisia mallinnusmenetelmiä. Mallintamisen avulla kokoonpanon suunnittelevat henkilöt pystyvät helpommin hahmottamaan tarvittavat riippuvuudet ja sitä kautta löytämään erilaiset ratkaisuvaihtoehdot.

Arkkitehtuurin suunnittelu voi olla hyvin hankalaa kaikkien tarvittavien riippuvuuksien ja esimerkiksi muunneltavuuden esittämisen kannalta [Murwantara, 2011]. Tästä johtuen mallintaminen on yksi tärkeä osa arkkitehtuurin suunnittelua. Arkkitehtuuria mallintamalla pyritään muodostamaan kokonaisvaltainen kuvaus komponenttien välisistä riippuvuuksista, vaihtoehdoista ja rajapinnoista. Mallintamalla pyritään myös kuvaamaan, kuinka tieto liikkuu eri komponenttien välillä, kuinka

komponentit voidaan keskenään konfiguroida ja miten rajapinnat komponenttien välillä toimivat. [Verdouw et al., 2014]

Objektipohjaisen mallinnuksen avulla voidaan helposti mallintaa modulaarisuutta ja tuoteperheitä [Jørgensen, 2010]. Objektipohjaisessa mallinnuksessa hyödynnetään usein UML -mallinnuskieltä ja sen avulla pyritään kuvaamaan eri komponenttien ja alijärjestelmien välisiä yhteyksiä ja suhteita [Jørgensen, 2010]. Mallintamista voidaan hyödyntää tuotteiden suunnittelussa, kehityksessä sekä käyttöönottovaiheen asiakastuessa [Verdouw *et al.,* 2014].

Tuotelinjan arkkitehtuuria voidaan mallintaa esimerkiksi ominaisuusmallilla, joka mahdollistaa ilmaisuvoimaisen hierarkkisen kuvauksen siitä, mitä tuotevariaatioita ohjelmistoperhe sisältää [Zakál *et al.,* 2011]. Ominaisuusmalli siis kuvaa, mitkä ovat tuotelinjan yhteisiä jaettuja komponentteja ja mitkä ovat tuotelinjan muokattavia komponentteja. Malli sisältää myös kuvauksen erilaisten variaatioiden sisältämistä komponenttien välisistä suhteista. Ominaisuusmallilla voidaan myös kuvata funktionaalisten vaatimusten samankaltaisuutta tai erovaisuutta eri komponenteissa [Murwantara, 2011]. Ominaisuusmalli sisältää yleensä ominaisuuskaavion. Ominaisuuskaaviolla voidaan kuvata visuaalisesti helposti ymmärrettäviä malleja [Zakál *et al.,* 2011]. Ominaisuuskaavioilla pyritään myös ryhmittelemään ja organisoimaan komponentit loogisesti.

Ominaisuusmallien avulla voidaan varmistaa, että lopputuotteeseen kuuluvat kaikki tarvittavat komponentit eikä kriittisiä komponentteja jää pois [Zakál *et al*., 2011]. Toisaalta ominaisuusmallien avulla voidaan havaita mahdollisesti turhat komponentit ja jättää ne pois lopullisesta toteutuksesta. Ominaisuusmallit ovatkin tärkeä osa vaatimusmäärittelyn tekoa kaikille ohjelmistoperheen tuotteille ja koko ohjelmistoperheelle [Murwantara, 2011].

Tuotelinjan mallintaminen on yleisesti ottaen tärkeää myös tuotevariaatioiden hallinnan kannalta [Ma and Tan, 2006]. Mallintamisen avulla voidaan pitää variaatioiden määrä riittävän pienenä ja voidaan tehostaa ohjelmistokomponenttien uudelleenkäyttöä.

Muunneltavuus on tärkeä osa tuotelinjaa, mutta muunneltavuutta täytyy pystyä myös mallintamaan, jotta se on hallittavissa ja seurattavissa. Muunneltavuutta voidaan mallintaa objektipohjaisella mallinnuksella tai ominaisuuspohjaisella mallinnuksella [Ma and Tan, 2006]. Muunneltavuus voi keskittyä yhtä aikaa koko toimintaympäristöön sisältäen tuotealustan sekä yksittäiset komponentit [Ma and Tan, 2006].

Tuotelinjan muunneltavuutta voidaan mallintaa vaatimusten tasolla sekä suunnittelun tasolla [Ma and Tan, 2006]. Eri mallinnustasoja voidaan hyödyntää eri vaiheissa tuotelinjan elinkaarta ja niitä voidaan hyödyntää rinnakkain. Vaatimusmallin tarkoituksena on ilmaista olemassa olevien tuotteiden ja mahdollisesti tulevaisuudessa toteutettavien tuotteiden muunneltavuutta vaatimustasolla [Ma and Tan, 2006]. Muunneltavuuden suunnittelumallin tarkoituksena on osoittaa kehittäjille, kuinka muunneltavuus tulee toteuttaa ohjelmistokomponenteissa ja tuotteissa. Muunneltavuuden suunnittelumalli perustuu yleensä vaatimusmalliin [Ma and Tan, 2006].

Näiden mallien lisäksi tuotelinjan mallinnuksessa voidaan hyödyntää myös uudelleenkäyttömallia. Tämän mallin tarkoituksena on helpottaa uudelleenkäyttöä osoittamalla, miten komponentteja voidaan muunnella ja konfiguroida eri käyttötarkoituksiin [Ma and Tan, 2006].

6.8. Ohjelmointikielet modulaarisen arkkitehtuurin tukena

Monet ohjelmointikielet tukevat itsessään modulaarista ohjelmistotuotantoa ja ne on jopa rakennettu modulaarisen ajattelumallin ympärille [Hoek and Lopez, 2011]. Viime vuosikymmeninä on kehitetty paljon tällaisia ohjelmointikieliä, jotka sallivat eri moduulien käsittelyn ja niiden välisten suhteiden hallinnoinnin. Käytännössä lähes kaikki nykyaikaiset ohjelmistokielet mahdollistavat ohjelmiston jakamisen useisiin hallittaviin osiin [Hoek and Lopez, 2011].

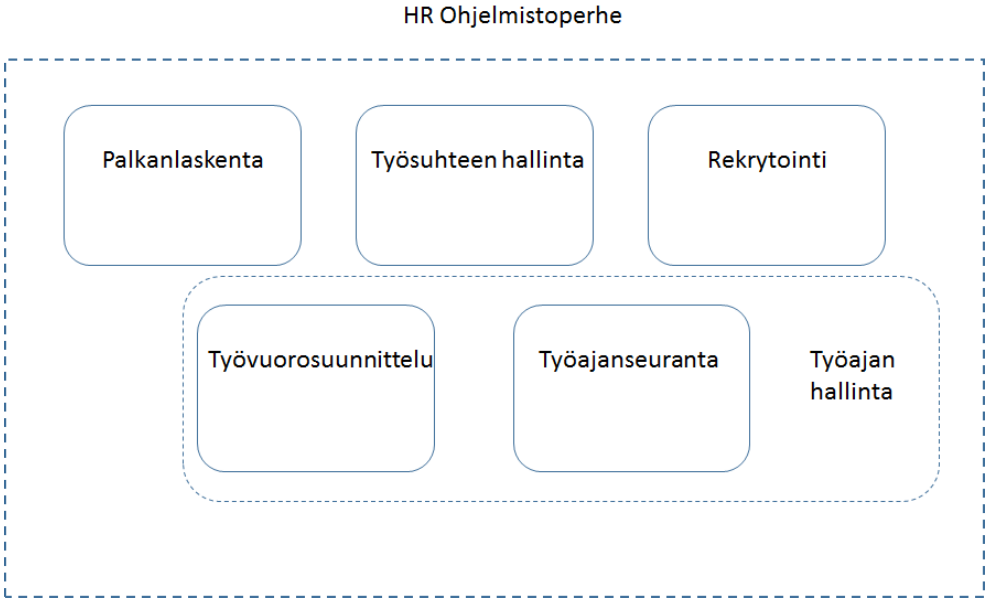
Yksi tehokas tapa muokattavuuden helpottamiseen on oliopohjaisten ohjelmointikielien mahdollistama tiedon piilottaminen [Hoek and Lopez, 2011]. Tiedon piilottaminen mahdollistaa sen, että muiden kuin alkuperäisten kehittäjien ei tarvitse nähdä sellaista tietoa, joka ei ole oleellista. Näin voidaan tehdä esimerkiksi moduulien luokista helppokäyttöisempiä ja samalla voidaan varmistaa, että luokkaa käytetään siten kuin se on tarkoitettu käytettäväksi. Tiedonpiilotus onkin yksi oliopohjaisen ohjelmoinnin perusperiaatteista.

Kehittäjien kannattaa pyrkiä ennakoimaan, kuinka moduuleita tullaan mahdollisesti muokkaamaan tulevaisuudessa ja piilottaa sellaiset osiot, joita ei pidä tai kannata muuttaa. Tiedonpiilottamisen avulla kehittäjät voivat myös suunnitella valmiiksi, miten muutokset tullaan tekemään [Hoek and Lopez, 2011]. Tämän avulla tulevien muutosten tekeminen on helpompaa ja voidaan varmistaa, että muutokset tehdään oikein. Kun piilotetaan sellaiset osiot, joita muiden moduulien ei tarvitse nähdä, voidaan varmistua siitä, että muutokset eivät vaikuta muihin moduuleihin [Hoek and Lopez, 2011].

7. Esimerkkinä HR -järjestelmä

Massaräätälöintiä voidaan hyödyntää erilaisissa ohjelmistotuotteissa ja tuotantomalleissa. Lopulliset tuotteet ja tuotantomallit voivat poiketa hyvinkin paljon toisistaan, joten kaiken kattavaa esimerkkiä massaräätälöinnistä on hankala esittää. Tässä luvussa pyritään kuitenkin havainnollistamaan esimerkkien kautta, kuinka massaräätälöintiä voitaisiin hyödyntää kuvitteellisen HR -järjestelmän toteutuksessa. Tarkoituksena ei siis ole esittää kokonaisvaltaista suunnitelmaa massaräätälöitävästä HR -järjestelmästä, vaan tarkoitus on esittää yksittäisiä suunnitteluesimerkkejä siitä, kuinka eri massaräätälöinnin keinoja voitaisiin hyödyntää.

HR -järjestelmällä tarkoitetaan tässä yhteydessä työsuhteen elinkaaren hallintaan tarkoitettua tuoteperhettä, johon kuuluvat tuotteet ovat palkanlaskenta, työsuhteen hallinta, rekrytointi, työvuoroseuranta ja työvuorosuunnittelu. Näistä jokainen on oma toisistaan riippumat tuotteensa ja asiakkaalle voidaan ottaa käyttöön vain ne tuotteet, jotka asiakas omiin tarpeisiinsa tarvitsee. Asiakkaan ei ole siis pakko ottaa käyttöön kaikkia tuotteita. Tuoteperhe on esitetty kuvassa 11.



Kuva 11 Hr -järjestelmäesimerkin tuoteperhe.

7.1. Mukautuvuus

HR-ohjelmistoperhe tulee toteuttaa niin, että kaikki tuotteet ovat riippumattomia toisistaan ja että jokainen tuote voidaan ottaa itsenäisesti käyttöön. Lisäksi tuotteiden tulee tukea standardoitua viestintää siten, että perheen tuotteet voivat kommunikoida keskenään ja mahdollisesti muiden ohjelmistojen kanssa. Tuotteiden tulee siis sisältää standardoidut rajapinnat.

Tämän lisäksi itse tuotteilta vaaditaan tietynlaista muunneltavuutta. Kun kyseessä on työsuhteen hallinta ja palkanmaksu, sisältyy toimintamalleihin paljon esimerkiksi sidonnaisuutta työehtosopimuksiin. Näin ollen tuotteet täytyy pystyä parametrisoimaan määrättyjen TES -sääntöjen mukaisesti. Erilaiset lomakkeet, kuten työsopimukset, ovat aina myös toimiala- ja yrityskohtaisia, joten ne tulee olla helposti muokattavissa asiakaskohtaisesta. Myös raportointitarpeet ovat usein toimiala- ja organisaatiokohtaisia, joten raportit tulee olla helposti muokattavissa ja parametrisoitavissa. Muunneltavuuden lisäksi ohjelmisto täytyy suunnitella muokattavaksi siten, että sen arkkitehtuuri tukee modulaarisuutta, tuoteperhemallia, tuotelinjaa ja yhteistä tuotealustaa.

HR -järjestelmä pitää pystyä myös ottamaan käyttöön organisaation mittakaavaan sovitetussa muodossa. Esimerkiksi pienissä noin 10 - 20 hengen yrityksissä työvuorosuunnittelua voidaan tehdä keskitetysti siten, että yksi henkilö toteuttaa tämän. Suurissa, jopa tuhansien henkilöiden organisaatioissa, työvuorosuunnittelu voi perustua puolestaan työvuorotoiveisiin. Tällöin työntekijöiden tulee voida syöttää työvuorojärjestelmään työvuorotoiveensa, joiden pohjalta työvuorot suunnitellaan. Tällöin periaatteessa jokainen työntekijä on järjestelmän käyttäjä.

Tällaisissa suurissa organisaatioissa tärkeä osa muunneltavuutta on käyttäjäoikeustasot ja oikeustasojen mukainen rajaaminen. Järjestelmässä näkyviä ja muokattavia osioita tulee siis voida rajata käyttäjäoikeustasojen mukaan. Suuret organisaatiot vaativat yleensä myös hyväksymisketjuja. Esimerkiksi työvuorotoiveet voidaan hyväksyä lähimmän esimiehen toimesta, mutta vasta joku muu taho organisaatiosta voi vahvistaa työvuorot. Tällaisten hyväksymisketjujen muodostaminen täytyy olla mahdollista ja ne pitää pystyä määrittelemään asiakaskohtaisesti.

7.2. Läpinäkyvyys

HR -ohjelmistojen käyttöön liittyy hyvin paljon toimialakohtaisuutta. Näin ollen on tärkeää oppia ja kartoittaa tarpeelliset variaatiot, jotta niitä voidaan tarjota valmiiksi soveltuvina eri toimialoille. Tällaisten järjestelmäkokonaisuuksien käyttöönotot ovat yleensä melko massiivisia prosesseja, joissa tarvitaan toimittajan asiantuntemusta sekä suunnittelussa että käyttöönotossa. Nämä ovat tilanteita, joissa voidaan kartuttaa ymmärrystä, miten asiakkaat käyttävät järjestelmää ja miten toimintamallit poikkeavat eri toimialoilla. Tällaisten järjestelmien elinkaari on yleensä myös melko pitkä, joten vuorovaikutusta kannattaa pitää yllä koko tuotteen elinkaaren ajan. Ymmärrystä voidaan kartuttaa koko asiakassuhteen ajalta.

Myös erilaisia asiakaskyselyitä ja asiakastyytyväisyyskyselyitä voidaan hyödyntää. Asiakkaita voidaan pyytää määrätyin väliajoin vastaamaan tämän tyyppisiin kyselyihin tai joissain tapauksissa kyselyt voidaan sisällyttää itse järjestelmään esimerkiksi verkkolomakkeilla.

Tärkeää läpinäkyvyyden hyödyntämisessä on tulosten oikeanlainen analysoiminen ja saatujen tulosten vieminen tuotantoon järjestelmän kehityksessä. Tulosten perusteella on tärkeää oppia, millaista muunneltavuutta ja parametrisointia järjestelmältä tarvitaan, jotta asiakkaille voidaan tarjota oikeanlaisia valmiita variaatioita nimenomaan HR -toimintojen osalta.

7.3. Kosmeettisuus

Kosmeettisuus HR -järjestelmässä voidaan toteuttaa muokkaamalla ulkoasua ja värejä organisaation teemaan sopivaksi. Järjestelmään voidaan myös upottaa esimerkiksi asiakasorganisaation logot.

Jotkut toiminnallisuudet järjestelmästä voivat näkyä kenelle vaan esimerkiksi yrityksen internetsivuilla. Yrityksellä voi olla esimerkiksi internetsivuillaan rekrytointi -osio, josta löytyy työhakemuslomake. Tämä voidaan käytännössä toteuttaa niin, että sivustolla on linkki, joka ohjaa rekrytointilomakkeelle, joka on osa HR -järjestelmän rekrytointi -tuotetta. Lomake siis on osa rekrytoinnin tuotetta, mutta linkki lomakkeelle löytyy organisaation internetsivuilta. Lomake voidaan muokata sellaiseksi, että se muistuttaa organisaation internetsivuja ja sisältää organisaation logon. Näin ollen käyttäjä ei välttämättä edes huomaa, että on siirtynyt järjestelmästä toiseen.

Maantieteelliset asetukset, kuten kieliasetukset, täytyy tehdä määriteltäviksi käyttöönottovaiheessa ja jopa käyttäjäkohtaisiksi. Näin esimerkiksi osa käyttäjistä voi käyttää samaa järjestelmää englannin kielellä ja osa suomen kielellä.

7.4. Yhteistoiminnallisuus

Yhteistoiminnallisuus on tärkeä osa tällaista HR -järjestelmäkokonaisuutta varsinkin käyttöönottovaiheessa. Käyttöönottoprojektit ovat yleensä melko laajoja ja vaativat suunnittelua ja parametrisointia, joten projektiin tarvitaan usein toimittajan asiantuntemusta. Asiakkaalle valitaan hänen tarpeidensa mukaiset tuotteet ja ne täytyy konfiguroida käyttötarpeisiin sopiviksi. Esimerkiksi raportit ja lomakkeet ovat usein sellaisia, joihin tarvitaan asiakaskohtaisia muokkauksia. Nämä täytyy siis määritellä yhdessä asiakkaan kanssa. Myös toimialakohtaisuus ja työehtosopimussidonnaisuus täytyy ottaa huomioon käyttöönotossa.

Usein järjestelmä täytyy myös integroida jo olemassa oleviin järjestelmiin, kuten myyntijärjestelmiin tai asiakkuudenhallintajärjestelmiin. Näiden liittymien toteuttamisessa tarvitaan usein toimittajan asiantuntemusta. Asiakkaan kanssa pitää yhdessä suunnitella tietojen määrittäminen ja liikkuminen järjestelmästä toiseen.

Yleensä tällaisten HR -järjestelmien elinkaari organisaatiossa on melko pitkä, joten tehokas asiakassuhteen hallinta on tärkeä osa järjestelmän toimitusta. On myös hyvin mahdollista, että järjestelmän tarpeet tulevat muuttumaan asiakassuhteen aikana, joten muuttuviin tarpeisiin on kyettävä vastaamaan. Hyvin sitoutettu asiakas valitsee todennäköisesti tuotteen jo olemassa olevalta toimittajalta.

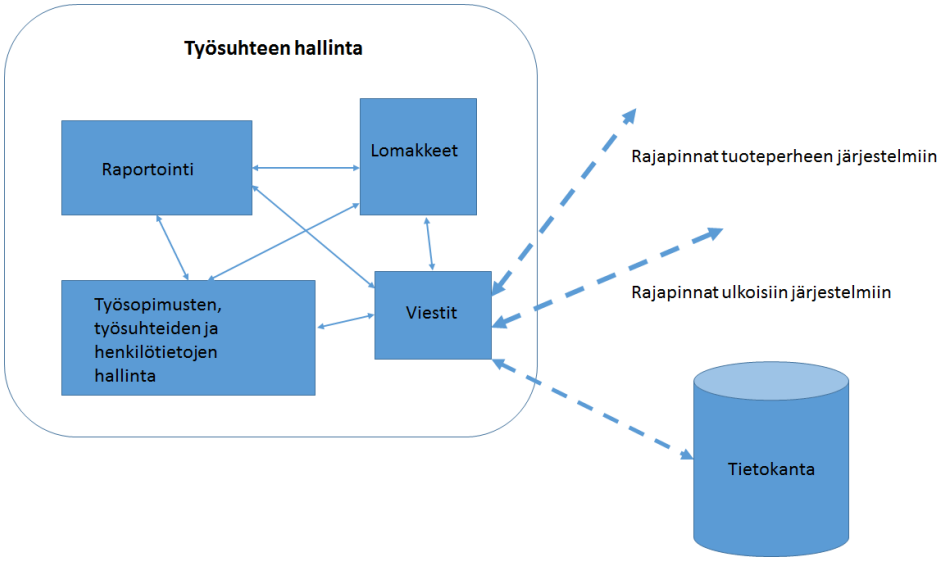
HR -järjestelmän käyttöön voidaan tarjota myös konsultointipalveluita esimerkiksi palkanlaskentaan. Tämän lisäksi erilaiset ylläpitopalvelut ovat hyvin todennäköisesti asiakkaalle tarpeellisia, jotta voidaan varmistaa järjestelmän toimivuus.

Verkostoituminen on myös tärkeä osa yhteistoiminnallisuutta. Verkostoitumisella tällaisen HR -järjestelmän kohdalla tarkoitetaan verkostoitumista muiden alaan liittyvien toimijoiden kanssa. On esimerkiksi yleistä, että yksi toimija ei tarjoa kaikkia tässä esimerkkijärjestelmässä esitettyjä tuotteita. Usein toimijat keskittyvät pelkkään työsuhteenhallintaan, palkanlaskentaan tai työvuorosuunnitteluun. Jos yksi toimija esimerkiksi tuottaa työsuhteenhallintaan kehitettyjä järjestelmiä, on hänen kannattavaa verkostoitua sellaisten toimijoiden kanssa, jotka tuottavat palkanlaskennan ja työvuorosuunnittelun järjestelmiä. Näin verkostoitumisen kautta pystytään tarjoamaan yhdessä suurempia järjestelmäkokonaisuuksia asiakkaille yhteistyö- ja suosittelukäytäntöjen kautta.

7.5. Modulaarisuus

HR -järjestelmää suunniteltaessa modulaarisuus tulee ottaa huomioon siten, että tuotteet rakennetaan itsenäisistä komponenteista, joita pyritään mahdollisimman tehokkaasti uudelleenkäyttämään. Esimerkiksi työsuhteen hallinta voidaan ottaa asiakkaalla käyttöön itsenäisenä tuotteena. Tällöin se tarvitsee käyttöönsä esimerkiksi raportoinnin omana komponenttinaan. Samaa raportointikomponenttia voidaan hyödyntää myös muissa tuotteissa, mikäli ne toimitaan itsenäisinä tuotteina.

Viestintä kannattaa hoitaa keskitetysti oman komponenttinsa kautta. Tällöin tuotteen sisäisten komponenttien viestintä voidaan hoitaa keskitetysti, kuten myös tuotteen kommunikointi muihin järjestelmiin. Tuotteen jakaminen erillisiin komponentteihin on esitetty kuvassa 12.



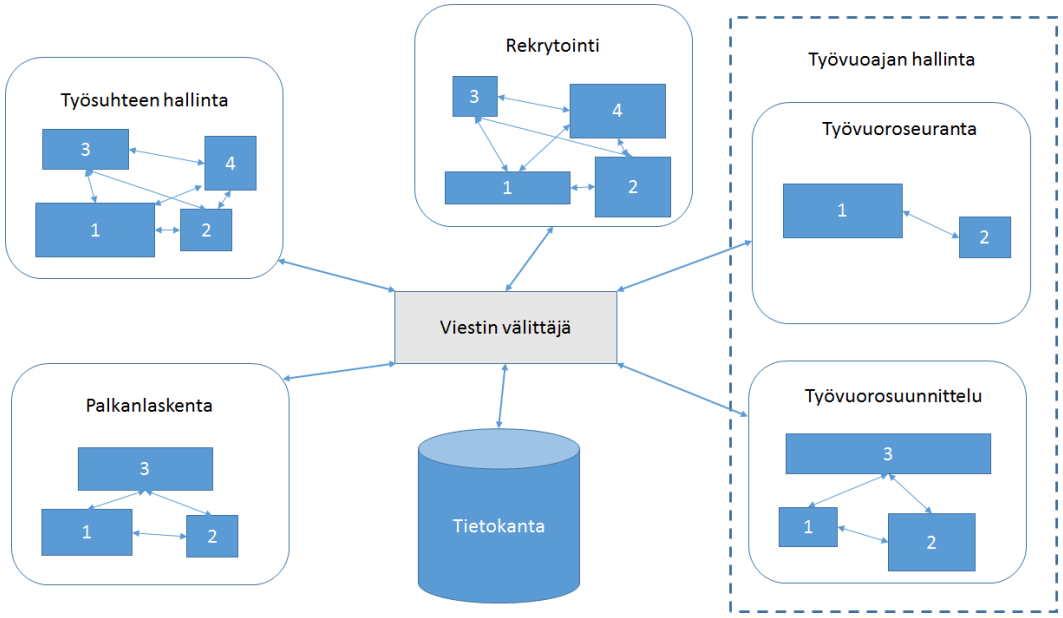
Kuva 12. Työsuhteen hallinta.

Kuvan 12 esimerkissä raportointi on sisällytetty komponentiksi tuotteen sisälle. Näin voidaan toimia, mikäli toimitetaan vain yksi tuote asiakkaalle. Kuitenkin jos asiakkaalle toimitetaan useampia ohjelmistoperheen tuotteita, voi raportointi olla keskitetty komponentti siten, että se palvelee useamman tuotteen tarpeita. Tietokanta on myös sijoitettu erilleen varsinaisesta tuotteesta, jotta se ei ole osa tuotteen toteutuslogiikkaa. Näin ollen muut ohjelmistoperheen tuotteet voivat tarpeen vaatiessa käyttää samaa tietokantaa ilman, että sillä on vaikutusta jo käytössä olevaan tuotteeseen.

7.6. Ohjelmistoperhe

Asiakkaalle voidaan ottaa käyttöön kaikki ohjelmistoperheen tuotteet tai vain se tuote, jonka asiakas tarvitsee. Tällöin perustieto kannattaa sijaita erillään tuotteiden logiikasta. Silloin se ei aiheuta riippuvuutta tuotteiden välille, vaan tuotteita voidaan lisätä tai ottaa pois käytöstä tarpeen mukaan. Tämä onnistuu siten, että tietokanta pidetään kokonaan erillään muiden järjestelmien osien logiikasta.

Tuotteiden välinen kommunikointi kannattaa hoitaa keskitetysti siten, että eri järjestelmät kommunikoivat niin sanotun viestinvälittäjän kautta. Myös mahdollinen integrointi muihin järjestelmiin kannattaa toteuttaa viestinvälittäjän avulla. Viestinvälittäjä tulee toteuttaa standardoiduilla rajapinnoilla, jotta voidaan mahdollistaa eri teknologioilla toteutettujen järjestelmien kommunikointi. Näin voidaan myös mahdollistaa se, että ohjelmistoperheen tuotteet eivät kommunikoi suoraan toistensa kanssa ja aiheuta näin ollen keskinäisiä riippuvuuksia. Tuoteperheen arkkitehtuuri on esitetty kuvassa 13.



Kuva 13. HR -järjestelmä viestinvälittäjällä.

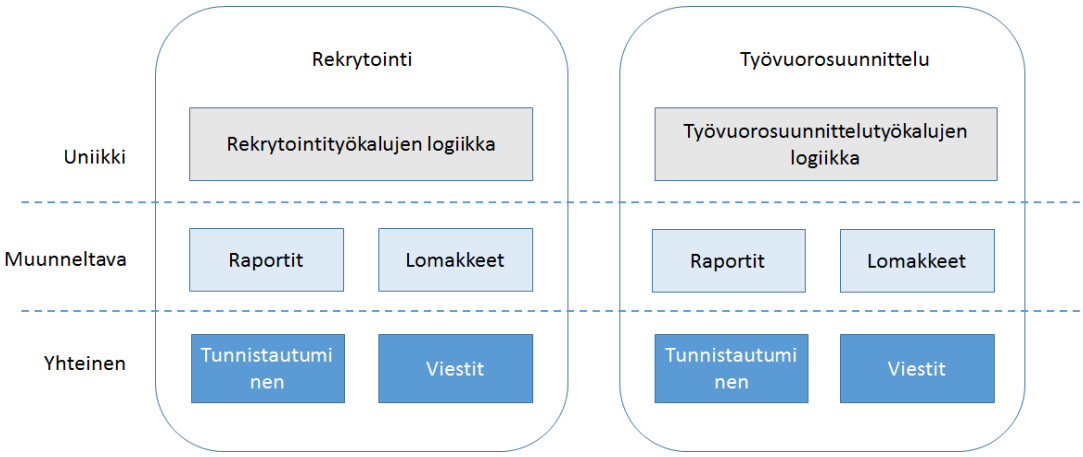
Vaikka tuoteperhe koostuu useasta tuotteesta, ei se välttämättä ilmene suoraan käyttäjälle lopullisesta käyttöliittymästä. Järjestelmää ei kannata toteuttaa siten, että jokainen tuote on erikseen avattava sovelluksensa, vaan tuotteet voivat olla omina moduuleinaan samassa käyttöliittymässä. Kun käyttöliittymän toteutus on erillään varsinaisesta tuotteiden logiikasta, ei käyttäjä välttämättä edes havaitse käyttävänsä eri tuotteita. Määrätyt komponentit voidaan myös jakaa eri tuotteiden kesken. Esimerkiksi raportointi voidaan toteuttaa keskitetysti siten, että samasta raportointiosiosta voidaan ajaa sekä rekrytointiin että työvuorosuunnitteluun liittyviä raportteja.

7.7. Tuotelinja ja tuotealusta

HR -järjestelmän tuoteperheen komponentit kannattaa rakentaa yhteisen tuotealustan pohjalle. Ne voivat jakaa useita samankaltaisia muunneltavia komponentteja, kuten tunnistautumisen, raportoinnin ja viestinvälityksen. Tuotealusta kannattaa rakentaa näiden jaettavien komponenttien pohjalta ja esimerkiksi raportointia voidaan muunnella tarpeen mukaan eri tuotteissa. Tietokanta kannattaa sijaita yhdessä paikassa kokonaan erillään yksittäisen tuotteen toteutuslogiikasta. Näin ollen viestintä tietokannan kanssa voi olla kaikilla ohjelmistoperheen tuotteilla samanlaista. Uudelleenkäyttö kannattaakin viedä mahdollisimman pitkälle.

Esimerkiksi viestinvälitys ja tunnistautuminen voivat olla kaikille ohjelmistotuotteille samanlaisia. Raportointi ja lomakkeet taas voivat olla muunneltavia komponentteja. Eri tuotteiden käytössä voi olla siis sama raportointikomponentti, mutta komponenttia voidaan muokata ja parametrisoida tuotekohtaisesti.

Tämän lisäksi tuotteisiin tarvitaan toisistaan poikkeavia loogisia komponentteja liittyen esimerkiksi työvuorosuunnitteluun tai rekrytointi-ilmoitusten tekemiseen. Rajapinnat täytyy tehdä kaikkiin tuotteisiin samalla tavalla standardoiduiksi, jotta tuotteet voidaan vaivatta liittää toisiinsa ja mahdollisesti muihin järjestelmiin. Esimerkki yhteisistä komponenteista, muunneltavista komponenteista ja uniikeista komponenteista on esitetty kuvassa 14.



Kuva 14 Yhteiset, muunneltavat ja uniikit komponentit.

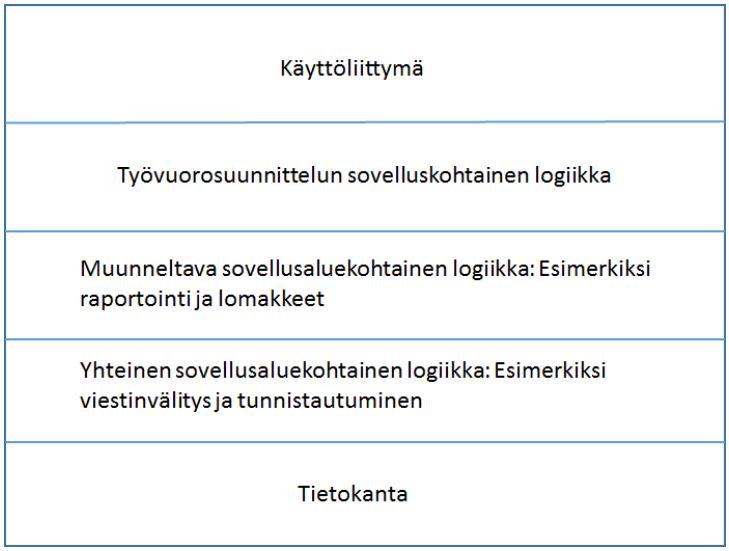
Tuotelinjan arkkitehtuurin ylläpidettävyyden ja seurattavuuden kannalta ohjelmistoperhe ja yksittäiset tuotteet täytyy mallintaa ja dokumentoida hyvin, jotta eri komponenttien väliset riippuvuudet ja suhteet käyvät helposti ilmi. Tämä helpottaa ylläpidettävyyttä ja tulevaisuudessa mahdollisesti tehtävää kehitystyötä.

Tuotelinjan arkkitehtuuri on mahdollista suunnitella myös siten, että eri tuotteet voivat jakaa keskenään saman käyttöliittymän. Käyttäjän näkökulmasta eri tuotteet voivat siis sisältyä samaan käyttöliittymään ikään kuin omina moduuleinaan. Tämä helpottaa käytettävyyttä, kun käyttäjän ei tarvitse käynnistää erikseen useampia sovelluksia. Määritellyt komponenttien toiminnallisuudet voivat olla myös yhteisiä kaikille tuotteille. Esimerkiksi tunnistautumista ei tarvitse välttämättä tehdä erikseen kaikkiin tuotteisiin, vaan tunnistautuminen voidaan tehdä keskitetysti.

7.8. Kerrosarkkitehtuuri

HR -tuoteperheen tuotteiden arkkitehtuurissa voidaan hyödyntää kerrosarkkitehtuuria. Arkkitehtuuri voidaan suunnitella siten, että alhaalla ovat yhteiset muuttumattomat osiot ja ylhäällä ovat sovelluskohtaiset osiot. Tässä esimerkissä on käytetty työvuorosuunnittelun tuotetta. Alimpana kerroksissa on tietokanta, joka on kaikille tuotteille yhteinen. Seuraavassa kerroksessa on sovellusaluekohtaiset yhteiset komponentit, kuten viestinvälitys ja tunnistautuminen. Seuraavassa kerroksessa on

toimialuekohtaiset muunneltavat komponentit, kuten raportointi ja lomakkeet. Tämän jälkeen seuraavassa kerroksessa ovat puhtaasti sovelluskohtaiset komponentit, jotka tässä tapauksessa liittyvät työvuorosuunnittelun toiminnallisuuteen. Viimeisimpänä ylhäällä on käyttöliittymä. Työvuorosuunnittelun kerrosarkkitehtuuri on esitetty kuvassa 15.



Kuva 15. Esimerkki kerrosarkkitehtuurista.

Koska tässä tapauksessa tarkastellaan yhtä tuotetta, on käyttöliittymä järkevää sijoittaa kerroksissa kaikista ylimmäiseksi, sillä esimerkissä on haluttu toteuttaa kerrosarkkitehtuuri, jossa alemmat kerrokset eivät ole riippuvaisia ylemmistä kerroksista. Kuitenkin, jos ohjelmistoperheen tuotteet halutaan toteuttaa siten, että ne jakavat saman käyttöliittymän, voidaan käyttöliittymä sijoittaa kerroksissa myös alemmaksi.

Kerrosarkkitehtuuri tulee suunnitella siten, että uniikit komponentit ovat ylimpänä, muunneltavat komponentit keskellä ja yhteiset komponentit alhaalla. Arkkitehtuuri tulee myös toteuttaa niin, että alemmat kerrokset eivät ole riippuvaisia ylemmistä kerroksista. Näin alemmat kerrokset muodostavat siis tuotealustan. Tällainen arkkitehtuuri mahdollistaa helpommin ylläpidettävän ja muunneltavan ohjelmistokokonaisuuden.

7.9. Palvelukeskeinen arkkitehtuuri

HR -järjestelmän sovellukset voidaan toteuttaa myös palvelukeskeisen arkkitehtuurin mukaisesti esimerkiksi SaaS -mallilla. Tällöin sovellukset voidaan toteuttaa itsenäisiksi pilvipalveluiksi siten, että ne ovat asiakkaan käytössä internetyhteyden yli verkkoselaimella. Tuotteiden kehityslogiikka keskittyy siis tuotteiden käyttöfunktioiden ympärille, jotka tässä tapauksessa ovat työvuorosuunnittelu, työaikaseuranta, palkanlaskenta, rekrytointi ja työsuhteen hallinta.

Hyvä puoli tämän tyyppisessä mallissa on, että asiakkaan ei tarvitse hankkia omaa palvelintilaa tai asentaa varsinaista ohjelmistoa. Tämä tekee käyttöönotosta huomattavasti helpompaa esimerkiksi sellaisille asiakkaille, joilla ei ole resursseja tai osaamista tarvittavan ympäristön käyttöönottoon. Asiakas ei käytännössä tarvitse muita resursseja kuin internetyhteyden ja verkkoselaimen käyttääkseen palvelua.

Palvelukeskeinen arkkitehtuuri ja SaaS -malli tuo suuren edun varsinkin pienemmille asiakkaille. Hyvin toteutettu palvelukeskeinen arkkitehtuuri mahdollistaa myös tehokkaan integroinnin muihin mahdollisiin järjestelmiin teknologiariippumattomasti.

8. Massaräätälöinnin haasteet ja ongelmat

Jotta massaräätälöinti olisi kannattavaa, täytyy tuotteita tai palveluita pystyä tuottamaan lähes massatuotannon tehokkuudella siten, että räätälöinti tuo kuitenkin lisäarvoa asiakkaalle. Pyrkimyksenä on, että tuotantoprosessi ja ratkaisumallit voidaan hyvin pitkälti vakioida. Kaikkien tuotteiden tai palveluiden tuottamiseen tämä ei kuitenkaan sovellu.

Massaräätälöinnin mallin toteuttaminen ei ole mahdollista tai järkevää kaikkien tuotteiden tai palveluiden tuottamisessa. Esimerkiksi, jos räätälöinti ei tuo juurikaan lisäarvoa tuotteeseen tai palveluun, tällöin massatuotanto on parempi ratkaisu. Toisaalta, jos myyntivolyymi tuotteelle tai palvelulle on vähäinen ja jokainen myytävä tuote tai palvelu on hyvin uniikki, ei massaräätälöinti myöskään tällöin ole järkevää. Tällöin puhdas yksilöllinen räätälöinti on parempi ratkaisu. Jotta massaräätälöinti olisi kannattavaa, tulee ottaa huomioon riittävä kysyntä räätälöidyille tuotteille, asiakkaiden valmius maksaa yksilöllisistä tuotteista ja asiakkaiden hyväksymä toimitusaika [Ahoniemi *et al.*, 2007]. Nämä samat säännöt pätevät yhtälailla ohjelmistotuotantoon kuin muillekin liiketoiminta-alueille.

Tuotealustapohjaisten ratkaisuiden kehitys massaräätälöintiä varten voi alkuvaiheessa aiheuttaa yritykselle hyvin suuria kustannuksia. Tuotealustan kehitys teollisuuteen voi maksaa jopa kymmenen kertaa enemmän kuin yksittäisen tuotteen kehitys [Simpson *et al.*, 2006]. Sama pätee myös ohjelmistotuotannossa. Ohjelmistoalustan kehitys voi olla huomattavasti kalliimpaa kuin yksittäisen ohjelmistotuotteen kehitys. On siis tärkeää tunnistaa yrityksen markkina-alueelta ja omasta tuoteperheestä kyseisen yrityksen tarve tuotealustalle. Mikäli markkina-alueella tarvitaan lähinnä todella uniikkeja ohjelmistoratkaisuja, ei tuotealustan hyödyntäminen ole kannattavaa [Simpson *et al.*, 2006].

Vaihtoehtojen tuominen asiakkaalle ei välttämättä tarkoita suoraan, että se toisi hyötyä asiakkaalle [Piller and Tseng, 2010]. Varsinkin jos vaihtoehtoja on hyvin paljon ja suurin osa vaihtoehdoista ei koske asiakkaan tarpeita, voi sopivan tuotekokonaisuuden löytäminen olla haastavaa ja variaatiomahdollisuudet voivat aiheuttaa turhaa hämmennystä. Tällainen tilanne voi jopa karkottaa asiakkaita [Piller and Tseng, 2010]. Tällöin puhutaan niin sanotusta massahämmennyksestä. Massahämmennyksen välttämiseksi on tärkeää, että tuotevariaatioiden määrä saadaan pidettyä kohtuullisella tasolla ja kohdennettua markkina-alueelle oikein. Vaikka modulaariset ohjelmistotuotteet ja tuoteperheet pyritään toteuttamaan joustavaksi siten, että lähes rajaton määrä

kombinaatioita olisi mahdollinen, ei tällaisen kombinaatiomäärien tarjoaminen ole kuitenkaan järkevää.

Sopivaa variaatiovaihtoehtojen määrää on vaikea määritellä ja se on aina tapauskohtaista. Tavoitteena on löytää sopiva määrä vaihtoehtoja, jotta ne täyttäisivät asiakkaiden tarpeet. Toisaalta vaihtoehtoja ei saisi olla liikaa, jotta kokonaisuuksista ei tulisi liian monimutkaisia ja vaihtoehtojen määrä ei aiheuttaisi asiakkaille niin sanottua massahämmennystä. [Ahoniemi *et al.*, 2007]

Jos tuotevariaatioiden määrä on hyvin suuri, tulee tuotevariaatioiden, komponenttien ja alijärjestelmien hallinnasta hyvin monimutkaista. Tämä yleensä aiheuttaa lisäkustannuksia ja massaräätälöinnin hyödyt voidaan menettää. Mikäli tuotevariaatioita on hyvin paljon, voidaan kuitenkin hyödyntää erilaisia automaatioratkaisuja, jotka auttavat monimutkaisten tuotekokonaisuuksien hallinnassa [Piller and Tseng, 2010].

Vaikka asiakkaiden vaatimukset ja erilaiset käyttötarpeet ovat arvokasta tietoa, ei jokaista vaatimusta tai toivetta kannata toteuttaa. Käyttäjien odotukset räätälöinnin suhteen ovat yleensä hyvin optimistiset ja, jos tähän kaikkeen pyritään vastaamaan, on todennäköistä, että järjestelmästä ja prosessista tulee liian monimutkainen ja raskas hallita [Meyer and Webb, 2005]. Ohjelmistotuotannossa täytyy ottaa aina huomioon, että monimutkaisuus ja räätälöinti tuovat lisäkustannuksia. Monimutkaisuus ja räätälöinnin määrä täytyy siis suhteuttaa asiakkaiden tarpeisiin ja markkinoihin. Liika räätälöinti ja monimutkaisuus tarkoittavat, että kustannukset kumoavat massaräätälöinnin tuomat hyödyt.

Vaikka tuoteperhemallia ja tuotealustaa hyödyntämällä voidaankin saavuttaa joustavuutta, kilpailukykyä ja kustannustehokkuutta, voi tuoteperhemalli aiheuttaa myös lisäkustannuksia ja muita haittoja liiketoiminnalle [Simpson *et al.*, 2006]. Onnistunut tuoteperhemalli vaatii yleensä selkeää organisaatiorakennetta ja kaikkien liiketoiminnan osa-alueiden sitoutumista malliin. Mikäli kaikki organisaation osat tai prosessit eivät ole sitoutuneita malliin, voi se aiheuttaa ongelmia muun muassa kommunikoinnissa ja dokumentoinnissa. Jos esimerkiksi erilliset yksiköt toteuttavat tuotteen eri komponentteja, olisi yksiköiden välillä syytä toimia jonkinlainen tuki toisten yksiköiden suuntaan, jotta kommunikaatio ja kokonaiskuva monimutkaisista arkkitehtuureista säilyisivät.

Ohjelmiston tekeminen modulaariseksi on kannattavaa ja jopa välttämätöntä massaräätälöinnin näkökulmasta, mutta toteutuksessa täytyy ottaa huomioon, että

ohjelmiston tekeminen modulaariseksi aiheuttaa lisäkustannuksia [Hoek and Lopez, 2011]. Tästä syystä on tärkeää arvioida, millä tasolla ohjelmisto tai ohjelmistoperhe tehdään modulaariseksi. Aivan täydellisen modulaarisuuden toteuttaminen ei välttämättä ole kannattavaa kaikissa tapauksissa. Täydellinen modulaarisuus ei ole tarpeellista esimerkiksi, jos variaatioiden ja räätälöinnin tarve on vähäinen. On siis tärkeää suhteuttaa modulaarisuuden aiheuttamat kustannukset ja niistä saatavat hyödyt toisiinsa.

Modulaarisuus aiheuttaa helposti myös sen asteista monimutkaisuutta, että siitä voi seurata laatuvaatimusten heikkenemistä [Hoek and Lopez, 2011]. Toisin sanoen moduuleita yhdistelemällä toteutettu järjestelmä ei välttämättä ole yhtä tehokas kuin täysin räätälöidysti tehty järjestelmä. Näin ollen massaräätälöidyt järjestelmät eivät välttämättä sovi kriittisten järjestelmien tehtäviin, joissa vaaditaan ehdotonta suorituskykyä ja laatuvaatimusten täyttymistä. Yksi paljon suorituskykyä vaativa komponentti voi aiheuttaa sen, että koko järjestelmä vaatii paljon suorituskykyä.

Monilla teollisuuden aloilla massaräätälöinnin ongelmana voidaan kokea samaan tuotealustaan pohjautuvien tuotteiden liiallinen samankaltaisuus, jolla voi olla vaikutusta useampien tuotteiden myyntiin ja tuotteiden brändiin [Simpson *et al.*, 2006]. Tämän kaltaiset ongelmat eivät välttämättä ole kovinkaan kriittisiä ohjelmistotuotannossa, mutta erillisten ohjelmistojen ryhmittely omiksi tuotteikseen kannattaa tehdä selkeästi, jotta ne eivät aiheuta asiakkaille hämmennystä. Liian samankaltaiset ohjelmat voivat aiheuttaa tilanteen, jossa vaihteleviin asiakastarpeisiin ei pystytä vastaamaan riittävän hyvin ja toisaalta taas tuotteiden poikkeavuudet jäävät piiloon tuotteiden liiallisen samankaltaisuuden takia [Alsawalqah *et al.*, 2013].

Ohjelmistoperheen eri tuotteiden tulisi toteuttaa omia uniikkeja toiminnallisuuksiaan. Ohjelmistoperheen tuotteiden ei pitäisi sisältää funktionaalista päällekkäisyyttä, jotta asiakkaan olisi helppo tehdä ostos omaan tarpeeseensa. Jos esimerkiksi asiakas tarvitsee ohjelmiston työvuorosuunnitteluun ja työajanhallintaan ja tuoteperheessä on kaksi eri sovellusta, jotka sisältävät tämän kaltaista toiminnallisuutta, voi asiakkaan olla vaikeaa tehdä päätös näiden välillä.

Tuoteperhemalli ei saa aiheuttaa liiallisia rajoitteita laadullisiin ei-toiminnallisiin vaatimuksiin. Nämä ovat vaatimuksia, jotka sisältävät muun muassa ohjelmistolle asetetut turvallisuusvaatimukset, luotettavuusvaatimukset, suorituskykyvaatimukset, saatavuusvaatimukset ja käytettävyysvaatimukset [Sommerville, 2007]. Kun tuotteessa uudelleenkäytetään ja yhdistellään eri tavoin valmiita komponentteja, on mahdollista, että ohjelman suorituskyky laskee merkittävästi tai sen käytettävyys kärsii, vaikka kaikki

komponentit toimisivat luotettavasti muissa tuotteissa. Nämä asiat tulee ottaa huomioon tuotealustan suunnittelussa. Vaikka tuoteperhemallin avulla pystytään automatisoimaan testaamista ja vähentämään testaamisen tarvetta, tulee testaamisen kuitenkin olla riittävää, jotta ohjelmiston laatuvaatimukset täyttyvät.

Hyvin toteutettu massaräätälöity ohjelmisto vaatii hyvin toteutetun dokumentoinnin. Jos järjestelmää ei ole dokumentoitu hyvin, ovat ongelmat jossain vaiheessa väistämättömiä. Ohjelmistokehitystä tehdään usein erillisissä yksiköissä, jolloin eri yksiköt ovat usein riippuvaisia muiden yksiköiden tuottamien osa-alueiden dokumentoinnista. Jos dokumentointi on huonoa, mennään helposti suuntaan, jossa eri komponentit eivät olekaan enää yhteensopivia tai ohjelmistokokonaisuuden laadullisten vaatimusten täyttymisen taso laskee.

On myös yleistä, että asiakkaat tekevät omia muokkauksia ohjelmistoihin ja liittävät niihin omia järjestelmiään. Tätä varten tarvitaan myös dokumentointia. Jos liittymiä, päivityksiä tai muokkauksia ei tehdä tarkoitetulla tavalla, voivat päivitykset, muokkaukset tai liittymät pahimmillaan korruptoida koko ohjelman tai aiheuttaa laadullisia ongelmia [Meyer and Webb, 2005].

9. Yhteenveto ja johtopäätökset

Useat tutkimukset ja kirjallisuus osoittavat, että massaräätälöinnillä voidaan saavuttaa kustannustehokkuuden parantumista ohjelmistotuotannossa samoin kuin perinteisessä teollisuudessakin. Massaräätälöinti voidaan myös kohdistaa eri osa-alueille ohjelmistotuotannossa samaan tapaan kuin teollisuudessa.

Perinteisessä teollisuudessa ja ohjelmistotuotannossa tuotantoprosessin kokonaiskuva on melko erilainen, koska ohjelmistotuotanto ei sisällä esimerkiksi samanlaisia logistiikkaan tai varastointiin liittyviä tekijöitä kuin perinteinen teollisuus. Siitä huolimatta samoja periaatteita voidaan soveltaa myös ohjelmistotuotantoon ja joissain tapauksessa ohjelmistotuotantoa voidaan pitää jopa massaräätälöinnin äärimallina.

Ohjelmistotuotannossa varsinkin mukautuvuus on ollut alusta alkaen massaräätälöinnin kulmakivi. Mukautuvuutta voidaan tukea ohjelmistotuotannossa monilla eri tavoilla. Muun muassa oikein suunniteltu ja sitoutuneesti toteutettu tuoteperhemalli voi tuoda yritykselle suurta kustannushyötyä.

Tuotteiden ja prosessien modulaarisuudella voidaan tuottaa erilaisia lopputuotteita tehokkaasti kasvattaen samalla joustavuutta ja reagointikykyä asiakkaiden tarpeisiin ja muuttuviin markkinoihin [Simpson *et al.,* 2006]. Joustavuus ja reagointikyky mahdollistavat muun muassa nopean uusien teknologioiden omaksumisen ja uusille markkina-alueille pääsemisen. Tämä tuo suuren kilpailuhyödyn varsinkin niitä yrityksiä kohtaan, jotka tekevät ohjelmistotuotantoa perinteisillä menetelmillä. Muita toimivasta tuoteperhemallista saatavia hyötyjä ovat muun muassa lyhemmät tuotantoajat, pienemmät tuotantokustannukset ja arkkitehtuuriltaan yksinkertaisemmat lopputuotteet [Simpson *et al.,* 2006].

Yhdistelemällä tuotealustoja, standardoituja rajapintoja ja muokattavia komponentteja yritykset voivat luoda tuotelinja-arkkitehtuurin pohjaksi sellaiselle ohjelmistoperheelle, jossa jokainen ohjelmisto on oma tuotteensa ja mahdollisesti suunnattu hyvin erilaisille käyttäjille [Meyer and Webb, 2005]. Massaräätälöinti voidaan viedä vielä tätäkin pidemmälle ohjelmistotuotannossa antamalla asiakkaille mahdollisuus muokata ohjelmistoa erinäisin keinoin. Tällöin asiakas voi itse räätälöidä ohjelmiston juuri omiin tarpeisiinsa sopivaksi ja saavuttaa näin vielä suurempaa hyötyä [Meyer and Webb, 2005]. Asiakkaan tekemä räätälöinti voi kohdistua suoraan ohjelmiston toiminnallisuuteen, liitännäisiin tai ulkoasuun.

Massaräätälöinnin keinot eivät kuitenkaan suoraan takaa kustannustehokkuuden paranemista ja huonosti toteutettu massaräätälöinti voi jopa lisätä kustannuksia. Massaräätälöinnin toteuttaminen on aina tapauskohtaista ja vaatii tarkkaa suunnittelua. Alkuvaiheessa on hyvin tärkeää pystyä analysoimaan, onko massaräätälöinnin toteuttaminen yleisesti ottaen kannattavaa. Kun massaräätälöintiä lähdetetään tekemään, on tuotealueen ja tuotevariaatioiden rajaaminen erittäin tärkeää, jotta tuotteet pystytään pitämään helposti hallittavissa ja kohdennettavissa. Näin voidaan myös välttää variaatioiden liian suuren määrän aiheuttama massahämmennys ja monimutkaisuus.

Tuotantoprosessilla on tärkeä merkitys onnistuneen massaräätälöinnin toteutuksessa. Koko tuotantoprosessin täytyy olla sitoutunut massaräätälöintiin ja tuotantoprosessi halutaan vakioida mahdollisimman pitkälle, jotta päästään lähelle massatuotannon kustannustehokkuutta. Tämän lisäksi prosessiin täytyy sisällyttää tarkka suunnittelu, hyvä dokumentointi ja tulevien muutosten ennakoiminen. Lisäksi pitkäkestoisista asiakassuhteista kiinnipitäminen on tärkeää. Massaräätälöinti täytyy myös kohdentaa asiakaskohtaisesti niille osa-alueille, mikä kullekin asiakkaalle on tärkeää.

Oikeanlaisessa ympäristössä ja hyvin toteutettuna massaräätälöinti tarjoaa hyvät työkalut siihen, kuinka ohjelmistotuotantoa voidaan tehdä joustavammin, tehokkaammin ja laadukkaammin, jotta voidaan vastata tämän päivän markkinatilanteen ja globaalin kilpailun asettamiin vaatimuksiin. Ohjelmistoala kehittyy koko ajan ja samalla asiakkaiden laatu- ja kustannustietoisuus kasvaa. Näin ollen entistä ketterämpien ja kustannustehokkaampien ohjelmistokehitysmallien löytäminen on tärkeää.

Tässä tutkielmassa on esitelty keinoja, kuinka perinteisessä teollisuudessa ja palveluiden tuottamisessa käytettyjä massaräätälöinnin keinoja voidaan hyödyntää ohjelmistotuotannossa. Tärkeimpinä keinoina on esitelty massaräätälöinnin kohdentaminen, modulaarisuus, tuoteperheet, tuotealustat ja tuotelinjat. Tulevaisuudessa tutkimusta voitaisiin viedä vielä huomattavasti syvemmälle näihin menetelmiin. Tuotelinjojen, ohjelmistoperheiden ja tuotealustojen pohjalta löytyykin paljon valmista kirjallisuutta ja tutkimusmateriaalia. Tätä tutkielmaa voisi syventää muun muassa tapaustutkimuksilla olemassa olevista ohjelmistoratkaisuista, joissa hyödynnetään esiteltyjä menetelmiä. Tämän avulla voitaisiin perehtyä siihen, kuinka massaräätälöinnin keinoja käytännössä toteutetaan yrityksissä.

Massaräätälöinnin keinojen lisäksi tässä tutkielmassa on pohdittu, minkälaisia vaatimuksia massaräätälöinti asettaa ohjelmistolle ja tuotantoprosessille. Tutkimuksessa

käytiin läpi pintapuolisesti arkkitehtuurille asetettuja vaatimuksia sekä keinoja massaräätälöityvän arkkitehtuurin toteuttamiseen. Tärkeimmät esitellyt keinot olivat modulaarinen arkkitehtuuri, tuoteperhearkkitehtuuri, tuotealusta-arkkitehtuuri, tuotelinjan arkkitehtuuri ja mallintaminen. Lisäksi tutkielmassa esitettiin tuotantoprosessille asetettavia vaatimuksia, jotka mahdollistavat massaräätälöinnin toteutumisen. Tuotantoprosessit, mallintaminen ja erilaiset arkkitehtuuriratkaisut ovat suuria asiakokonaisuuksia, joten tutkimusta voitaisiin syventää käsittelemään tarkemmin erilaisia arkkitehtuuritekniikoita sekä mallinnusmenetelmiä. Myös näistä asiakokonaisuuksista löytyy jo tänä päivänä paljon kirjallisuutta ja tutkimusmateriaalia. Olemassa olevia arkkitehtuuritoteutuksia ja prosessiviitekehyksiä tarkastelemalla voitaisiin perehtyä siihen, kuinka massaräätälöinnin mahdollistava arkkitehtuuri ja prosessimalli voitaisiin käytännössä toteuttaa.

Viiteluettelo

[Ahoniemi *et al.*, 2007] Lea Ahoniemi, Markus Mertanen, Marko Mäkipää, Matti Sievänen, Petri Suomala ja Mikko Ruohonen, *Massaräätälöinnillä kilpailukykyä*, Teknologiateollisuus Ry, 2007.

[Alfnes and Skjelstad, 2010] Erlend Alfnes, Lars Skjelsta, How to implement a mass customization strategy: guidelines for manufacturing companies. In: Frank Piller and Mitchell Tseng, *Handbook of Research in Mass Customization and Personalization.* World Scientific Publishing Co. Pte. Ltd. 44-64.

[Alsawalqah *et al.*, 2013] Hamad Alsawalqah, Sungwon Kang, Danhyung Lee, A method for software product platform design based on features. In: *Proc. of SPLC '13 Workshops Proceedings of the 17th International Software Product Line Conference co-located workshops*, 18-25.

[Aoyama, 1998] Mikio Aoyama, New Age of Software Development: How Component Based Software Engineering Changes the Way of Software Development? In: *Proc. of 1998 International Workshop on CBSE*, 1-5.

[Blecker and Friedrich, 2007] Thorsten Blecker, Gerhard Friedrich, Guest Editorial: Mass Customization Manufacturing Systems. *Engineering Management, IEEE Transactions on* **54**, 1 (Feb. 2007), 4-11.

[Brown, 2000] Alan W. Brown, *Large-Scale, Component-Based Development*, 1th ed. Prentice Hall PTR, 2000.

[Gao *et al.*, 2009] Fei Gao, Gang Xiao, Timothy W. Simpson, Module-scale-based product platform planning. *Research in Engineering Design* **20**, 2 (Jul. 2009), 129-141.

[Hoek and Lopez, 2011] André van der Hoek, Nicolas Lopez, A design perspective on modularity. In: *Proc. of AOSD '11 Proceedings of the tenth international conference on Aspect-oriented software development*, 265-280.

[Jiao and Tseng, 1999] Jianxin Jiao, Mitchell M. Tseng, A methodology of developing product family architecture for mass customization. *Journal of Intelligent Manufacturing* **10**, 1 (Mar. 1999), 3-20.

[Johannesson and Gedell, 2010] Hans Johannesson and Stellan Gedell, Knowledge based configurable product platform models. In: Frank Piller and Mitchell Tseng, *Handbook of Research in Mass Customization and Personalization.* World Scientific Publishing Co. Pte. Ltd. 357-375.

[Jørgensen, 2010] Kaj A. Jørgensen, Product family modeling: working with multiple abstraction levels. In: Frank Piller and Mitchell Tseng, *Handbook of Research in Mass Customization and Personalization.* World Scientific Publishing Co. Pte. Ltd. 319-337.

[Kumar, 2004] Ashok Kumar, Mass Customization: Metrics and Modularity. *International Journal of Flexible Manufacturing Systems* **16**, 4 (Oct. 2004), 287-311.

[Li *et al.*, 2006] Yiyuan Li, Jianwei Yin, Jinxiang Dong, A Component Management System for Mass Customization. In *proc. of Computer and Computational Sciences, 2006. IMSCCS '06. First International Multi-Symposiums **2***, 398-404.

[Ma and Tan, 2006] Jianhong Ma, Runhua Tan, Handling Variability in Mass Customization of Software Family Product. In: Kesheng Wang, George L. Kovacs, Michael Wozny, Minglun Fang, *Knowledge Enterprise: Intelligent Strategies in Product Design, Manufacturing, and Management.* Springer US. 996-1001.

[Mathiassen and Sandberg, 2014] Lars Mathiassen, Anna Börjesson Sandberg, Process Mass Customization in a Global Software Firm. *Software, IEEE* **31,** 6 (Jan. 2014), 62-69.

[Merle et *al.*, 2010] Aurelie Merle, Jean-Louis Chandon, Elyette Roux, Why consumers are willing to pay for mass customized products: dissociating product and experiential value. In: Frank Piller and Mitchell Tseng, *Handbook of Research in Mass Customization and Personalization.* World Scientific Publishing Co. Pte. Ltd. 208-225.

[Meyer and Webb, 2005] Marc H. Meyer, Peter H. Webb, Modular, Layered architecture: the necessary foundation for effective mass customisation in software. *International Journal of Mass Customisation.* **1** (2005), 14-36.

[Michiels, 2003] Cindy Michiels, Monique Snoeck, Wilfried Lemahieu, Frank Goethals, Guido Dedene, A Layered Architecture Sustaining Model-Driven and Event-Driven Software Development. In: Manfred Broy, Alexandre V. Zamulin, *Perspectives of System Informatics.* Springer Berlin Heidelberg. 58-65.

[Mohabbati et al., 2011] Bardia Mohabbati, Marek Hatala, Dragan Gašević, Mohsen Asadi, Marko Bošković, Development and configuration of service-oriented systems families. In: *Proc. of SAC '11 Proceedings of the 2011 ACM Symposium on Applied Computing, 1606-1613.*

[Moon et *al.*, 2010a] Seung Ki Moon, Xiaomeng Chang, Janis Terpenny, Timothy W. Simpson, Soundar R.T. Kumura, towards a knowledge support system for product family design. In: Frank Piller and Mitchell Tseng, *Handbook of Research in Mass Customization and Personalization.* World Scientific Publishing Co. Pte. Ltd. 297-318.

[Moon et al., 2010b] Seung Ki Moon, Timothy W. Simpson, Soundar R.T. Kumara, market-based strategic platform design for a product family using a bayesian game. In: Frank Piller and Mitchell Tseng, *Handbook of Research in Mass Customization and Personalization.* World Scientific Publishing Co. Pte. Ltd. 338-356.

[Murwantara, 2011] I Made Murwantara, Initiating layers architecture design for Software Product Line. In: *proc. of 2011 International Conference on Uncertainty Reasoning and Knowledge Engineering (URKE),* 48-51.

[Paunu, 2014] Pasi Paunu, Design Configurator – Managing the Order Engineering Challenge in ETO Companies. Pro gradu -tutkielma. School of Information Sciences. Tampereen yliopisto: Tampere.

[Piirainen and Gonzalez, 2013] Kalle Piirainen, Rafael Gonzalez, Constructive Synergy in Design Science Research: A Comparative Analysis of Design Science Research and the Constructive Research Approach. *Liiketaloudellinen Aikakauskirja.* **3-4**, (2013), 206-234.

[Piller, 2004] Frank Piller, Mass customization: reflections on the state of the concept. *International Journal of Flexible Manufacturing Systems*. **16**, 4 (2005), 313-334.

[Piller and Tseng, 2010] Frank Piller, Mitchell Tseng, Mass customization thinking: moving from pilot stage to an established business strategy. In: Frank Piller and Mitchell Tseng, *Handbook of Research in Mass Customization and Personalization.* World Scientific Publishing Co. Pte. Ltd. 1-18.

[Pressman, 2005] Roger S. Pressman, *Software Engineering: A Practitioner's Approach*, 6th ed. McGraw-Hill, 2005.

[Sam *et al.*, 2006] Yacine Sam, Omar Bouccelma, Mohand-Saïd Hacid, Dynamic Web Services Personalization. In: *proc. of E-Commerce Technology, 2006. The 8th IEEE International Conference on and Enterprise Computing, E-Commerce, and E-Services, The 3rd IEEE International Conference,* 86-92.

[Sand, 2012] Antti Sand, Ohjelmistotuotannon massaräätälöinnistä tietojärjestelmätuotannossa. Pro gradu -tutkielma. Informaatiotieteiden yksikkö. Tampereen yliopisto: Tampere.

[Siems and Walcher, 2010] Florian Siems and Dominik Walcher, Modularity as a base for efficient life event cycle management. In: Frank Piller and Mitchell Tseng, *Handbook of Research in Mass Customization and Personalization.* World Scientific Publishing Co. Pte. Ltd. 263-274.

[Simpson *et al.*, 2006] Timothy W. Simpson, Zahed Siddique, Jianxin Roger Jiao, Platform-based product family development. In: Timothy W. Simpson, Zahed Siddique, Jianxin Roger Jiao, *Product Platform and Product Family Design.* Springer Science+Business Media, LLC. 1-15.

[Sommerville, 2007] Ian Sommerville, *Software Engineering*, 8th ed. Addison-Wesley, 2007.

[Stotko and Snow, 2010] Christof Stotko, Andy Snow, e-Manufacturing – Making extreme mass customization real by Laser-Stingering. In: Frank Piller and Mitchell Tseng, *Handbook of Research in Mass Customization and Personalization.* World Scientific Publishing Co. Pte. Ltd. 555-567.

[Tuck et *al.*, 2010] Christopher Tuck, Min-Huey Ong, Helen Wagner, Richard Hague, Extreme customization: rapid manufacturing products that enhance the consumer. In: Frank Piller and Mitchell Tseng, *Handbook of Research in Mass Customization and Personalization.* World Scientific Publishing Co. Pte. Ltd. 537-554.

[Verdouw *et al.*, 2014] Cor Verdouw, Adrie Beulens, Sjaak Wolfert, Towards Software Mass Customization for Business Collaboration. In: *Proc. of Global Conference (SRII), 2014 Annual SRII*, 106–115.

[W3C, 2015a] w3.org verkkosivut. Saatavilla osoitteessa http://www.w3.org/TR/wsdl20/. Viitattu 1.5.2015.

[W3C, 2015b] w3.org verkkosivut. Saatavilla osoitteessa http://www.w3.org/2001/03/WSWS-popa/paper08. Viitattu 1.5.2015.

[W3C, 2015c] w3.org verkkosivut. Saatavilla osoitteessa http://www.w3.org/TR/soap/. Viitattu 1.5.2015.

[Watson, 2010] Andrew Watson, A mass of customizers: the WordPress software ecosystem. In: Frank Piller and Mitchell Tseng, *Handbook of Research in Mass Customization and Personalization.* World Scientific Publishing Co. Pte. Ltd. 717-728.

[Wijnstra, 2002] Jan Gerben Wijnstra, Critical factors for a successful platform-based product family approach. In: Gary J. Chastek, *Software Product Lines.* Springer Berlin Heidelberg. 68-89.

[WordPress, 2015] WordPress.org verkkosivut. Saatavilla osoitteessa https://wordpress.org/. Viitattu 7.3.2015.

[Zakál *et al.*, 2011] Dávid Zakál, László Lengyel, Hassan Charaf, Software product lines-based development. In: *proc. of Applied Machine Intelligence and Informatics (SAMI), 2011 IEEE 9th International Symposium on*, 79-81.