Tampereen yliopisto

Ville Penttinen

OHJELMISTON TUOTTEISTAMINEN TUOTERUNKOARKKITEHTUURIA HYÖDYNTÄEN

Informaatioteknologian ja viestinnän tiedekunta
Kandidaatintyö
Syyskuu 2019

TIIVISTELMÄ

Ville Penttinen: Ohjelmiston tuotteistaminen tuoterunkoarkkitehtuuria hyödyntäen
Kandidaatintyö
Tampereen yliopisto
Tietotekniikka
Syyskuu 2019

Tässä työssä tutkitaan tuoterunkojen ja tuoterunkoarkkitehtuurin käyttöä ohjelmiston tuotteistamisessa. Tuotteistamisen tavoitteena on tuottaa markkinoille korkealaatuisia tuotteita kannattavasti.

Tuoterunkoa käyttämällä pyritään mahdollistamaan uusien tuotteiden luominen pienemmillä kustannuksilla ja korkeammalla laadulla verrattuna yksittäisten, asiakaskohtaisten tuotteiden luomiseen. Työssä selvitetään, onko ohjelmiston tuotteistamiseen olemassa valmiita prosesseja tai menetelmiä, joita seuraamalla syntyy tuoterunkoarkkitehtuuria hyödyntävä ohjelmistotuote.

Työssä käydään läpi alan kirjallisuuden pohjalta tuotteistamisen ja tuoterunkojen käsitteitä ja pohditaan ohjelmisto- ja tuoterunkoarkkitehtuurien välisiä yhteyksiä. Työssä tarkastellaan, miten tuotteistaminen ja tuoterungot voidaan sovittaa yhteen. Työssä myös esitetään tapoja tuoterunkoarkkitehtuurin huomioimiseen ohjelmistoarkkitehtuurissa. Ohjelmistoarkkitehtuurin tulee tukea muunneltavuutta, jonka avulla ohjelmistosta voidaan luoda uusia variaatioita erilaisiin tarpeisiin.

Työssä ilmenee miten tuoterunkoa kehittämällä voidaan samalla tukea ohjelmiston tuotteistamista. Lisäksi työssä esitellään joitakin tapoja tuoterunkoarkkitehtuurin huomioimiseen ohjelmiston kehityksessä.

Avainsanat: ohjelmiston tuoterunkokehitys, tuoterunkoarkkitehtuuri, tuotelinja, tuotteistaminen

SISÄLLYSLUETTELO

1 JOHDANTO

Yritysmaailmassa kehitettäessä ohjelmistoja asiakasyrityksille tehdään usein ohjelmistot vain tietyn asiakkaan tarpeita ja käyttöä varten. Tämä asiakaskohtainen ohjelmistokehitys voi johtaa siihen, että ohjelmisto on räätälöity asiakkaan tarpeita varten. Monesti alun perin tiettyä asiakasta varten tehdylle ohjelmistolle on tarvetta ja kysyntää myös muualla. Tällöin haasteena on, että asiakaskohtaiset ohjelmistot voivat olla toteutukseltaan ja arkkitehtuuriltaan nivottu yhteen kyseisen asiakkaan järjestelmien ja käyttötapauksien kanssa.

Asiakaskohtainen räätälöinti on tärkeä osa yrityksille tehtäviä ohjelmistoja ja ratkaisuja, sillä yrityksillä on monesti toisiinsa nähden erilaisia järjestelmiä käytössä, jolloin asiakaskohtaisten integraatioiden ja muutosten tekeminen korostuu. Yksittäiselle asiakkaalle tehtävän ohjelmiston muuttaminen arkkitehtuuriltaan ja toteutukseltaan sellaiseksi, joka tukee muunneltavuutta erilaisiin tarpeisiin, vaatii kehitysorganisaatiolta suunnittelua.

Asiakaskohtaisen ohjelmiston muuttamiseen yleiskäyttöisemmäksi tuotteeksi ei ole yhtä ainoaa tapaa, sillä muuttujia on paljon. Yksi tapa on muuttaa ohjelmiston arkkitehtuuria siten, että se tukee muunneltavuutta. Muunneltavuuden avulla voidaan asiakaskohtaisesta ohjelmistosta tuottaa uusia muunnelmia eli variaatioita, jotka täyttävät sekä uusien asiakkaiden tarpeita että nykyisten asiakkaiden nykyisiä ja uusia tarpeita. Muunneltavuuden toteuttamiseen ja hallinnoimiseen voidaan hyödyntää tuoterunkoa.

Tuoterunkoja ja niihin pohjautuvaa ohjelmistokehitystä on hyödynnetty jo jonkin aikaa, esimerkiksi matkapuhelimien kehityksessä. Tuoterunkoihin pohjautuva ohjelmistokehitys tukee samankaltaisten ohjelmistojen tuottamista pienemmillä kustannuksilla ja korkeammalla laadulla. Tuoterungon avulla kehitettyjä ohjelmistoja, joilla on samankaltainen rakenne ja samankaltaisia ominaisuuksia, nimitetään tuoteperheeksi. Tuoteperheen jäsenten kehittämistä ja ylläpitoa tukevia artefakteja ja prosesseja nimitetään tuotelinjaksi.

Esimerkkejä tuoteperheistä ovat muun muassa Microsoft Office -tuotteet, joilla on hyvin paljon yhteisiä käyttöliittymäelementtejä, vaikka ne ovatkin toiminnoiltaan erilaisia. Toinen esimerkki tuoteperheestä on aikaisemmat Nokian puhelimet. Nokian puhelimet hyödynsivät tuoteperhettä, joka mahdollisti yhteisen ohjelmistoarkkitehtuurin käyttämisen. Tämä yhteinen ohjelmistoarkkitehtuuri tuki muunneltavuutta erilaisten laitteiden ja käyttöliittymien välillä.

Tässä työssä tutkitaan, mitä tuoterungot ja tuoterunkoarkkitehtuuri ovat sekä miten niitä voidaan hyödyntää ohjelmiston tuotteistamisessa. Työssä pohditaan myös, miksi tuoterunkoa ja tuoterunkoarkkitehtuuria tulisi hyödyntää tuotteistamisessa, miten tuoterunkoarkkitehtuurin voi ottaa huomioon ohjelmistokehityksessä ja miten tuoterunkoarkkitehtuuri vaikuttaa ohjelmistoarkkitehtuuriin.

Luvussa 2 esitellään tuotteistamisen määritelmä sekä käydään läpi ohjelmiston tuotteistamiseen käytettäviä prosesseja. Luku 3 käsittelee ohjelmiston tuoterunkokehitystä ja tähän liittyviä konsepteja. Luvussa 4 selvitetään, miten tuoterunkoarkkitehtuuri voidaan ottaa huomioon ohjelmiston kehityksessä, sekä miten tuoterunko vaikuttaa ohjelmistoarkkitehtuuriin. Yhteenvedossa, luvussa 5, käydään läpi tutkielman johtopäätökset.

2 OHJELMISTON TUOTTEISTAMINEN

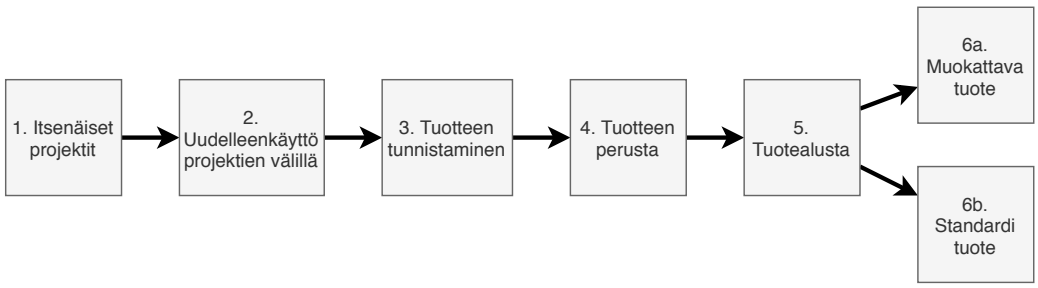
2.1 Tuotteistamisen määrittely

Tuotteistamiselle on olemassa useita määritelmiä. Suominen et al. määrittelevät tuotteistuksen standardoiduksi prosessiksi, jonka tarkoituksena on tuottaa korkealaatuista ja kannattavaa kaupallista tuotetta tai palvelua, joka on kehitetty tuotetun tiedon perusteella [22]. Flamholtz vastaavasti määrittelee tuotteistamisen prosessina, jossa analysoidaan nykyisten ja tulevien asiakkaiden tarpeita tuotteen suunnittelua varten [6]. Flamholtzin määrittelemänä tuotteistaminen sisältää tuotteen suunnittelun lisäksi varsinaisen tuotekehityksen.

Ohjelmistojen tuotteistamiseen liittyy ohjelmistokehityksen vaiheita, kuten käytettävien teknologioiden valinta, vaatimustenhallinta ja arkkitehtuurisuunnittelu, mutta myös ohjelmistokehityksen ulkopuolelle jääviä asioita, kuten tuotteen myynti ja hinnoittelu [11]. Tuotteistaminen on monipuolinen prosessi ja sen tavoitteena voidaan pitää sellaista tuotetta tai palvelua, joka on kannattava ja jolla on kysyntää markkinoilla. Ohjelmistojen tapauksessa kannattavuuteen vaikuttaa tuotteen kehitykseen vaadittavat resurssit ja hinnoittelu, kun taas kysyntään vaikuttaa tuotteen markkinointi.

2.2 Tuotteistusprosessi ohjelmistokehityksessä

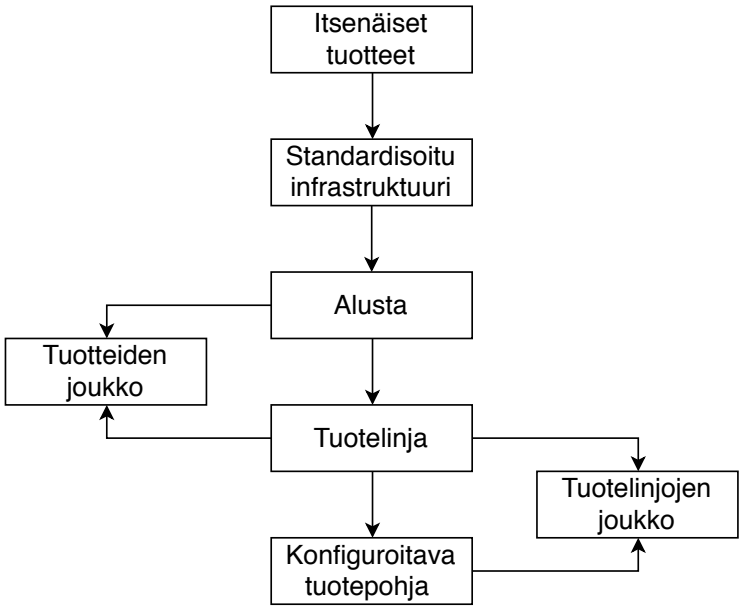
Asiakaskohtaisen ohjelmiston tuotteistamiseen ei ole olemassa yhtä ainoaa tapaa. Artz et al. esittävät [1] kuusivaiheisen tuotteistusprosessin asiakaskohtaisen ohjelmiston tuotteistamiseen, mikä on kuvattu kaaviossa 2.1. Tuotteistusprosessi kuvaa organisaation tuotteistamisen tasoa ja organisaatiot voivat hyödyntää prosessia riippumatta siitä, missä vaiheessa prosessia ne ovat. Esitetyn prosessin lopputuloksena on joko muokattava tuote tai yleiskäyttöinen standardituote.



Kuva 2.1. *Tuotteistusprosessi, muokattu lähteestä [1].*

Leenen et al. tekemässä tapaustutkimuksessa [15], jossa tutkittiin yhden organisaation ohjelmistotuotehallinnan kehitystä tuotteistuksen eri vaiheissa, onnistuttiin yhdistämään tutkimuksen kohteena olevan organisaation ohjelmistokehitysvaiheet Artz et al. esittämän tuotteistusprosessin ensimmäiseen viiteen vaiheeseen. Tapaustutkimukseen osallistunut organisaatio muutti ohjelmistokehityksen lähestymistapaa asiakaskohtaisesta ohjelmistokehityksestä markkinavetoisempaan ohjelmistokehitykseen. Leenen et al. tutkimuksessa tuotteeseen jäi asiakaskohtaisia osioita, minkä seurauksena täysin muokattavaa tai toisaalta aivan standardiakaan tuotetta ei toteutettu. Tutkimus kuitenkin antaa viitteitä tuotteistusprosessin hyödyllisyydestä.

Artz et al. esittämä tuotteistusprosessi on vaiheiltaan lähellä Boschin esittämiä tuotelinjan kypsyyden tasoja [5], jotka on esitetty kuvassa 2.2. Seuraavaksi käydään tarkemmin läpi tuotteistusprosessin eri vaiheita ja niiden yhteyttä Boschin esittämiin tuotelinjan kypsyyden tasoihin.



Kuva 2.2. *Tuotelinjojen kypsyyden tasot, muokattu lähteestä [5].*

Artz et al. esittämän tuotteistusprosessin ensimmäisessä vaiheessa on useita toisistaan erillisiä projekteja, joilla ei välttämättä ole juurikaan yhteisiä ominaisuuksia tai ne eivät sisällä yhteistä lähdekoodia. Nämä erilliset projektit ovat usein asiakaskohtaisia, ja asiakkaat ovat niiden pääsidosryhmä. [1] Boschin mallissa tämä vastaa itsenäisten tuotteiden tasoa [5].

Toisessa vaiheessa projekteja toteutetaan eri tavoilla, mutta ominaisuuksia ja toiminnallisuutta pyritään uudelleenkäyttämään projektien välillä. Toisessa vaiheessa on ensimmäiseen vaiheeseen verrattuna mahdollisuus käyttää näitä yhteisiä ominaisuuksia ja toiminnallisuutta. Tässä vaiheessa kuitenkin asiakaskohtaiset muokkaukset ovat vielä isommassa osassa kehitysprosessia verrattuna yleiskäyttöisiin ominaisuuksiin. [1] Tämän yhteisen toiminnallisuuden voidaan ajatella muodostavan Boschin mallin standardisoidun infrastruktuurin. Boschin mukaan tämä standardisoitu infrastruktuuri tyypillisesti sisältää

esimerkiksi käytettävän käyttöjärjestelmän ja muita kaupallisia komponentteja, kuten tietokannan hallintajärjestelmän. [5]

Vaiheessa 3 uudelleenkäytetään isompaa osaa aikaisemmista projekteista ja tuotteen tuotealue pystytään tunnistamaan tämän uudelleenkäytettävän osan perusteella. Vaihe 3 on ensimmäinen askel ohjelmistotuotteen luomisessa, ja tässä vaiheessa organisaation on hyvä arvioida tarvetta markkinavetoiseen kehittämiseen. Vaatimustenhallinta on keskeisessä osassa tuotteen tunnistamisessa. Kaikkien tulevien asiakasprojektien tulisi käyttää yhteistä järjestelmää vaatimustenhallintaan, jolloin yhteisiin ominaisuuksiin perustuvan tuotteen tunnistaminen helpottuu. Tässä vaiheessa ei voida kuitenkaan puhua tuotteesta tai yleiskäyttöisestä alustasta. [1] Boschin mallissa Artz et al. tuotteen tunnistamisen voidaan ajatella olevan standardisoidun infrastruktuurin ja alustan välillä.

Vaiheessa 4 tunnistetaan kehitettävä tuote, ja sen lisäksi käytössä on selkeästi määritelty tuotteen perusta, jota käytetään tuotteiden kehittämiseen. Tämä tuotteen perusta koostuu niistä yhteisistä ominaisuuksista, jotka aiemmissa vaiheissa on huomattu, ja näiden ominaisuuksien päälle pystytään tehokkaasti toteuttamaan asiakkaiden tarpeisiin muokattuja tuotteita. Vaatimustenhallintaan on tässä vaiheessa hyvä ottaa huomioon markkinan tarpeet pelkkien asiakaskohtaisten tarpeiden lisäksi. Tässä vaiheessa ei kuitenkaan ole kyse ohjelmistotuotteesta, sillä muokattavien osien osuus projektien välillä on vielä suuri. [1] Vastaavasti Boschin mallissa voidaan puhua alustasta. Tämä alusta sisältää kuitenkin vähän muunneltavuutta ja pääosa kehityksestä tapahtuu yksittäisten tuotteiden muokkauksessa [5].

Vaiheessa 5 keskitytään vielä asiakaskohtaisiin toteutuksiin, mutta vaiheeseen 4 verrattuna edetään kuitenkin kohti markkinoille kehitettävää tuotetta ja tuotealustaa. Yhteisten ominaisuuksien muodostaman alustan osuus on suuri, ja niiden päälle voidaan rakentaa yksittäisiä tuotteita nopeasti ja tehokkaasti. [1] Vaiheessa 5 voidaan Boschin mallin mukaisesti puhua jo tuotelinjasta. Tuotealustan päälle rakennettavan tuotelinjan kehittämisessä voidaan hyödyntää tuoterunkoarkkitehtuuria. Tuotelinja tuo mukaan muunneltavuuden, joka mahdollistaa uusien tuotteiden kehittämisen [5].

Artz et al. esittämä tuotteistusprosessi päättyy kahteen vaihtoehtoiseen lopputulokseen. Ensimmäisen vaihtoehdon tuotoksena syntyy muokattava ohjelmistotuote, jota muokkaamalla voidaan asiakkaille luoda heidän tarpeisiinsa soveltuvia tuotteita. Edellisiin vaiheisiin verrattuna asiakaskohtaiset muokkaukset ovat pienemmässä osassa kehitysprosessia. [1] Muokattavaa ohjelmistotuotetta vastaa Boschin mallissa useampi tuotelinjan kypsyyden tasoista riippuen siitä, miten muunneltavuus ja sen hallinta on alustassa toteutettu. Mikäli alusta on toteutettu niin, että suurin osa asiakaskohtaisista muokkauksista tapahtuu automatisoidusti työkaluja hyödyntämällä, voidaan puhua konfiguroitavasta tuotepohjasta, joka on Boschin mallin mukaisesti korkein tuotelinjan kypsyyden tasoista [5].

Muokattava ohjelmistotuote voi myös vastata tuotelinjojen joukkoa. Boschin esittämässä tuotelinjojen joukossa on kyse useammasta järjestelmän osasta, joista jokainen voi olla oma tuotelinjansa. Näitä eri tuotelinjoja voidaan konfiguroida tyypillisen tuoterunkokehi-

tyksen mallin mukaisesti tai hyödyntämällä yksittäisten tuotelinjojen konfigurointiin aiemmin esitettyä konfiguroitavaa tuotepohjaa. Tuotelinjojen joukko soveltuu etenkin suuren mittakaavan järjestelmien kehittämiseen. [5]

Verrattuna tuotelinjojen joukkoon Boschin mallin mukaisessa tuotteiden joukossa pyritään laajentamaan mahdollisten tuotteiden joukkoa yhteisten artefaktien pohjalta. Tuotteiden joukossa ei ole kyse pelkästään komponenttien sisäisestä muunneltavuudesta, vaan myös siitä, miten järjestelmän eri osia voidaan yhdistää keskenään erilaisten tuotteiden kehittämistä varten. [5] Tuotteiden joukkoa voidaan soveltaa muokattavan ohjelmistotuotteen kehittämisessä.

Kehitysorganisaatio voi myös jättää siirtymättä muihin tuotelinjan kypsyyden tasoihin ja jatkaa tuotteen kehitystä Boschin mallin mukaisen tuotelinjan päälle. Tällöin tuotelinjaa on kehitetty eteenpäin tukemaan enemmän muunneltavuutta.

Toisena tuotteistusprosessin lopputuloksena on standardiohjelmistotuote. Tämä standardiohjelmistotuote ei sisällä enää asiakaskohtaisia muokkauksia, vaan markkinoille tuodaan tuote, jota tarjotaan sellaisenaan asiakkaille. Tuote voi kuitenkin tukea konfiguroimista, esimerkiksi tuotetta asentaessa. [1] Kuten muokattavan ohjelmistotuotteen yhteydessä, voi tässäkin vaiheessa olla kyse Boschin mallin mukaisesta konfiguroitavasta tuotepohjasta. Konfiguroitavasta tuotepohjasta voidaan puhua esimerkiksi siinä tapauksessa, että konfiguroiminen tapahtuu esimerkiksi ohjelmaa asentaessa, kun käyttäjä syöttää lisenssiavaimen, jolloin asennettava ohjelma konfiguroi itsensä lisenssiavaimen tietojen perusteella [5]. Standardin ohjelmistotuotteen etuna on vain yhden tuotteen ylläpito. Vastaavasti jos uusia tarpeita ilmenee, ei niitä voida aina sisällyttää olemassa olevaan tuotteeseen, vaan edessä voi olla kokonaan uuden tuotteen tai tuoteversion kehittäminen.

Tuotteistusprosessin tarkoituksena ei ole suoraan vähentää kustannuksia ja tuotteen kehitykseen vaadittavaa aikaa, vaan prosessin avulla pyritään muuttamaan koko liiketoimintamallia. Asiakaskohtaisten ohjelmistojen kehittämisestä siirrytään ohjelmistotuotteen kehittämiseen, minkä avulla pyritään tarjoamaan ohjelmistoa suuremmalle määrälle asiakkaita. [1] Vastaavasti tuoterunkokehityksen tavoitteena on parantaa tuotteen kehityksen tehokkuutta, tuottaa parempilaatuisia tuotteita ja vähentää kustannuksia [4, 5, 21].

3 OHJELMISTON TUOTERUNKOKEHITYS

Pohl et al. määrittelevät [21, s. 14] ohjelmiston tuoterunkokehityksen (engl. software product line engineering) seuraavasti: Ohjelmiston tuoterunkokehitys on ohjelmistojen kehitysparadigma, jossa hyödynnetään yhteisiä alustoja sekä massaräätälöintiä ohjelmistojen kehittämisessä.

Ohjelmiston tuoterunkokehityksessä luodaan ohjelmistolle yhteinen ohjelmistoalusta, tuoterunko, jonka päälle on mahdollista kehittää eri tarpeisiin räätälöityjä tuotteita hyödyntämällä hallittavaa muunneltavuutta [21, s. 14]. Tuoterungon avulla pyritään kehittämään uusia tuotteita nopeammin, pienemmillä kustannuksilla ja paremmalla laadulla verrattuna yksittäisten tuotteiden kehittämiseen [4, 21]. Tuoterungon perustana olevaa ohjelmistoarkkitehtuuria nimitetään tuoterunkoarkkitehtuuriksi.

Ohjelmiston tuoterunkokehityksessä varsinainen kehitys jaetaan kahteen osaan: alustankehitysprosessiin ja tuotekehitysprosessiin [21]. Tässä luvussa käydään tarkemmin läpi tuoterunkokehityksen eri vaiheita ja käsitteitä sekä tuoterunkoihin liittyviä haasteita.

3.1 Alustankehitysprosessi

Alustankehitysprosessin (engl. domain engineering) aikana määritellään ne ominaisuudet, jotka ovat yhteisiä kaikille tuoteperheen tuotteille. Näiden yhteisien ominaisuuksien päälle rakennetaan käytettävä tuoterunko, joka variaatiopisteiden avulla tukee ohjelmiston muunneltavuutta. [21]. Variaatiopiste on alustan muunneltavuuden esitystapa rikastettuna kontekstuaalisella tiedolla [21, s. 62].

Alustankehitysprosessin aikana kehitettävän tuoterungon tulee huomioida sekä nykyisten että tulevien asiakkaiden vaatimukset ja tarpeet [13, s. 350]. Toimialaosaaminen on olennainen osa asiakkaiden vaatimuksien ja tarpeiden analysoimisessa. Toimialaosaamisesta on myös hyötyä alustan suunnittelussa ja kehittämisessä. Hyödyntämällä sovellusalueen tyypillisiä abstraktioita saadaan muunneltavuudesta ymmärrettävämpää sekä kehittäjille että asiakkaille. [21, s. 17–18]

Tyypillisesti alustankehitysprosessi aloitetaan vaatimusmäärittelyllä. Vaatimusmäärittelyn avulla pyritään tuomaan esiin ja dokumentoimaan kaikki tuoterungon yhteiset ja muuttuvat vaatimukset [21, s. 25–26]. Vaatimusmäärittelyssä voidaan hyödyntää tuotteen etenemissuunnitelmaa (engl. product roadmap), jonka tuotteesta vastuussa oleva taho on toimittanut [21, s. 25]. Vaatimusmäärittelyn yhteydessä luodaan myös muunneltavuusmalli

(engl. variability model), jossa kuvataan tarkemmin alustan muunneltavuusvaatimukset [21, s. 26]. Muunneltavuuteen palataan luvussa 3.3.

Vaatimusmäärittelyn jälkeen vaatimusmäärittelyä ja muunneltavuusmallia hyödynnetään tuoterungon arkkitehtuurin suunnittelussa. Tuoterungon arkkitehtuurin, eli tuoterunkoarkkitehtuurin, täytyy tukea muunneltavuutta. [21, s. 26] Tapoja, miten muunneltavuus voidaan ottaa huomioon ohjelmistoarkkitehtuurissa, käydään tarkemmin läpi luvussa 4.2.1. Alustan tuoterunkoarkkitehtuuria tulee myös testata, vaikka se on usein haastavaa, sillä varsinaista testattavaa tuotetta ei ole vielä tässä vaiheessa [21, s. 26].

Alustankehityksen lopputuloksena on ohjelmistoalusta, jossa variaatiopisteet ja niiden hyödyntäminen on dokumentoitu niin hyvin, että sen perusteella voidaan luoda erilaisia tuotekonfiguraatioita. Uuden tuotekonfiguraation luomista nimitetään tuotekehitysprosessiksi.

3.2 Tuotekehitysprosessi

Tuotekehitysprosessissa (engl. application engineering) hyödynnetään alustankehitysprosessin aikana kehitettyä muunneltavaa alustaa yksittäisen tuotteen ilmentymän kehittämistä varten [21, s. 20–21]. Tuotekehitysprosessin tehokkuus ja kannattavuus ovat riippuvaisia käytettävän alustan toteutuksesta.

Kuten alustankehitysprosessi, alkaa tuotekehitysprosessi myös vaatimusmäärittelyllä. Tuotekehitysprosessin vaatimusmäärittelyn tavoitteena on huomata sellaiset tuotekohtaiset ominaisuudet, joita alustankehitysprosessissa kehitetty tuoterunko tukee [21, s. 31–32]. Tavoitteena on pyrkiä hyödyntämään tuoterunkoa ja sen tarjoamaa muunneltavuutta mahdollisimman paljon, jotta tuotekohtaisten muutosten määrä olisi pienempi.

Tuotekohtaiset vaatimukset voivat myös vaatia muutoksia alustaan, jolloin tuotekehittäjien tulee kommunikoida sekä tuotteen omistajan että alustankehittäjien kanssa [21]. Tuotteen omistaja voi tehdä arvion tuotekohtaisten vaatimusten tarpeesta muissa tuotteissa ja päättää sen perusteella, muokataanko alustaa vai jätetäänkö kyseinen toteutus tuotekohtaiseksi.

Tuotekohtaisia vaatimuksia hyödynnetään varsinaisessa tuotteen suunnittelussa ja tuotekohtaisessa kehityksessä. Tuotekohtaisessa kehityksessä hyödynnetään alustan tarjoamia komponentteja, rajapintoja ja muita muunneltavuutta tukevia keinoja tuotteen kehittämiseen. Tuotekohtaisten muutosten täytyy kuitenkin noudattaa alustan arkkitehtuuria ja käytäntöjä. [21, s. 32–33]

Tärkeä osa tuotekehitysprosessia on tuotekohtainen testaus. Tuotekohtaiset testit rakennetaan usein alustakohtaisten testien päälle. Tuotekohtaisessa testauksessa täytyy myös huomioida tuotekohtaiset muutokset ja niiden testaus. [21, s. 33–34] Kattavan testauksen avulla tuetaan tuotteen jatkokehitystä ja ylläpitoa.

Tuotekehitysprosessin lopputuloksena on asiakkaalle tai asiakkaille tarjottava tuote, joka vastaa heidän tarpeisiinsa. Suurin osa tuotteen ominaisuuksista tulisi tulla alustan tarjoaman muunneltavuuden kautta. Seuraavaksi käsitellään tarkemmin muunneltavuutta tuoterungoissa.

3.3 Muunneltavuus

Muunneltavuus on keskeinen osa tuoterunkoja. Bachmann ja Clements määrittelevät [2, s. 3] muunneltavuuden järjestelmän kykynä tukea toisistaan suunnitelmallisesti poikkeavien artefaktien luomista. Määritelmässä korostetaan suunnitelmallisuutta, sillä tuoterunkojen tapauksessa muunneltavuus on osa prosessia. Tuoterunkojen muunneltavuudella pyritään maksimoimaan tuotteiden toteuttamisen tuottoaste tietyllä ajanjaksolla tai tietyllä määrällä tuotteita [2, s. 10].

Muunneltavuus tulisi dokumentoida selvästi. Selkeän dokumentaation [21, s. 73–74] tulisi ainakin vastata seuraaviin kysymyksiin:

- Mitkä asiat muuttuvat?
- Minkä takia asiat muuttuvat?
- Miten ne muuttuvat?
- Kenelle dokumentaatio on tarkoitettu?

Dokumentaation luominen ohjaa kehittäjiä esittämään perusteluita uusien variaatiopisteiden lisäämiselle. Samalla dokumentaatiota voidaan hyödyntää asiakkaiden kanssa kommunikoinnissa. Asiakkaille voidaan dokumentaation avulla esittää erilaisia ohjelmiston variaatioita, joista he voivat valita tarpeisiinsa sopivimman. [21, s. 73–74]

Muunneltavuuden hallintaa voi auttaa eri variaatiomekanismien rajaaminen. Vaikka olisikin mahdollista löytää ihanteellinen tapa varioida jokaista muunneltavaa osaa, on helpompaa rajata käytettävät variaatiomekanismit. Pienempi määrä variaatiomekanismeja tarjoaa tuotetta kehittäville kehittäjille pienemmän oppimiskynnyksen, sekä muunneltavuuden ylläpito helpottuu, kun mekanismeja on käytössä rajallinen määrä. Lisäksi muunneltavuuden automatisointi on helpompaa siinä tapauksessa, että käytettävien mekanismien määrä on rajallinen. [2, s. 26]

Alustakehittäjien tulee valita käytettävät variaatiomekanismit, eli tavat, joilla variaatio voidaan ottaa huomioon ohjelmistossa. Mekanismien valintaan vaikuttaa useampia tekijöitä, jotka tulee ottaa huomioon. Tämänlaisia tekijöitä ovat esimerkiksi [2]:

- mekanismin toteuttamiseen vaadittavat taidot
- toteuttamisen kustannukset, sisältäen uusien taitojen oppimisen
- mekanismin käyttöön kuluva aika ja kustannukset, sisältäen tuotekehittäjien kouluttamisen mekanismin käyttöön
- mekanismin käyttäjät, eli tuotekehittäjät

- mekanismin vaikutukset tuotteen laatuun, esimerkiksi suorituskykyyn tai muistinkulutukseen
- mekanismin vaikutukset alustan ylläpidettävyyteen

Näiden tekijöiden lisäksi käytössä oleva tuotantostrategia rajaa käytettäviä variaatiomekanismeja [2, 20]. Tuotantostrategia on kokonaislähestymistapa siihen, miten tuotealusta sekä yksittäiset tuotteet toteutetaan. Lähestymistapaan vaikuttaa esimerkiksi se, aloitetaanko tuotelinjan kehitys tyhjästä vai kehitetäänkö tuotelinjaa olemassa olevien tuotteiden tai yksittäisen tuotteen päälle. [20, s. 17]

3.4 Tuoterungon muutostenhallinta

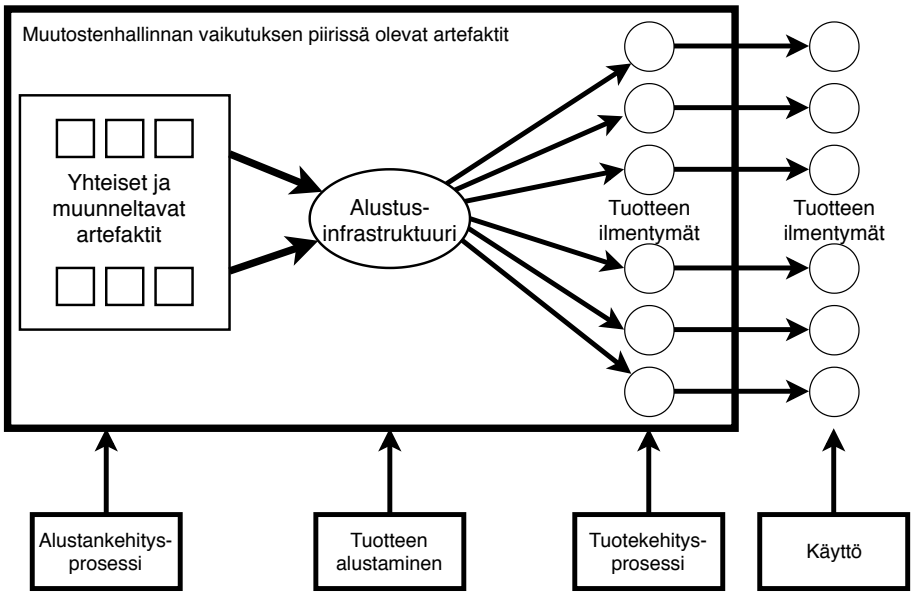
Ohjelmiston kehitysartefaktit kehittyvät ajan myötä, kun teknologiset edistysaskeleet johtavat muutokseen. Tätä kutsutaan muutokseksi ajan suhteen. Muunneltavuus ajan suhteen tarkoittaa eri artefaktien versioiden olemassaoloa eri aikoina. Konfiguraationhallinta on yleinen tapa eri versiota olevien kehitysartefaktien hallintaan. [21, s. 65]

Sekä yksittäisen ohjelmiston kehitys että ohjelmiston tuoterunkokehitys joutuvat hallitsemaan muutosta ajan suhteen. Erona yksittäisen ohjelmiston kehityksen ja ohjelmiston tuoterunkokehityksen välillä on se, että ohjelmiston tuoterunkokehityksessä on ohjelmistoon lisätty valmiiksi variaatiopisteitä, joihin on mahdollista lisätä uusia toiminnallisuuksia. Variaatiopisteet auttavat pitämään muutosten vaikutukset pieninä. [21, s. 65–66]

Muutos tilan suhteen kuvaa artefaktien eri muotoja eri tuotteissa samanaikaisesti [21, s. 66]. Esimerkiksi yritykselle myytävässä ohjelmistossa voi olla variaatiopiste, joka tarjoaa yritykselle mahdollisuuden valita, halutaanko kaksivaiheinen tunnistautuminen (engl. two-factor authentication) käyttöön vai ei. Näin kyseisen variaatiopisteen sisältävästä artefaktista voi olla kaksi eri muotoista versiota käytössä.

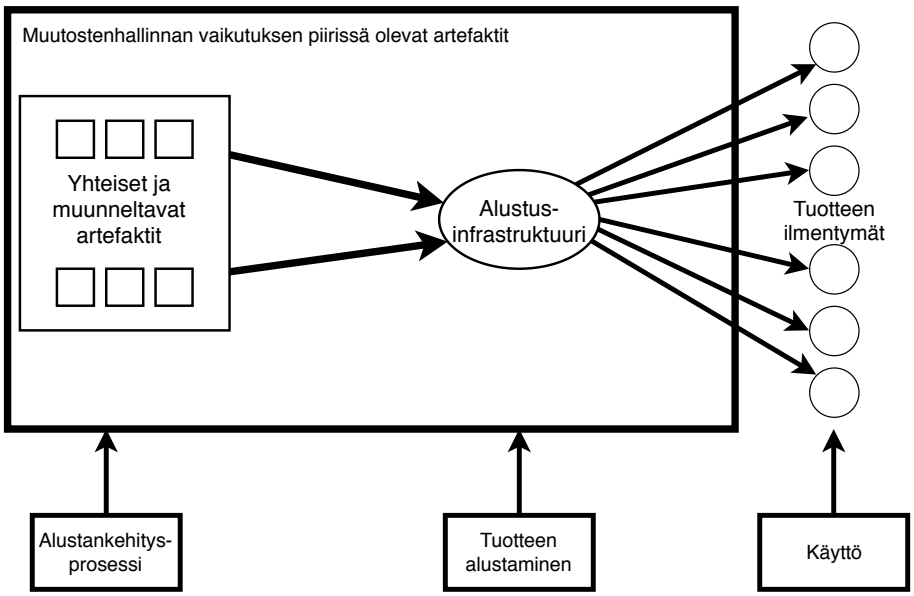
Ohjelmiston tuoterunkokehityksessä on kyse muutoksesta sekä ajan että tilan suhteen. Tämän vuoksi muutostenhallinta on tuoterunkojen tapauksessa moniulotteinen konfiguraationhallinnan ongelma. [14]

Tyypillisesti tuotelinjan muutostenhallinnan kannalta jokainen tuotteen ilmentymä on oma klooninsa luomisensa jälkeen. Esimerkiksi sadan tuotteen ilmentymä tarkoittaa sitä, että muutostenhallinnassa on sata erilaista tuotteen evoluutiota. Kloonien tapauksessa muutokset yksittäisiin tuotteisiin eivät heijastu takaisin yhteisiin artefakteihin tai muihin tuotteen ilmentymiin. Muutosten tuominen tuotteen muihin ilmentymien vaatii lisätyötä ja mahdollisesti refaktorointia, eli ohjelman lähdekoodin muuttamista siten, että se mahdollistaa muutosten käyttöönoton. Lisäksi alustaan tehtyjä muutoksia ja parannuksia ei voida välttämättä tuoda kaikkiin tuotteen ilmentymiin, sillä niihin on voitu toteuttaa sellaisia muutoksia, jotka eivät ole suoraan yhteensopivia alustaan tehtyjen muutosten kanssa. [14] Tyypillinen muutostenhallinta tuotelinjassa on esitetty kuvassa 3.1.



Kuva 3.1. *Muutostenhallinta tyypillisessä tuotelinjassa, muokattu lähteestä [14, s. 41].*

Krueger esittää [14] vaihtoehtoisen tavan muutostenhallintaan tuotelinjassa. Kruegerin mukaan muutostenhallinnan tulisi hallita tuotantolinjaa (engl. production line) tuotelinjan sijasta. Tuotantolinjassa muutostenhallinnassa olisi vain yhteiset ja muuttuvat artefaktit sekä tuotteen alustukseen käytettävä infrastruktuuri. Yksittäiset tuotteen ilmentymät jäävät muutostenhallinnan ulkopuolelle ja ne voidaan aina alustaa uudestaan. Yksittäisten tuotteiden kehitys poistuu ja kaikki kehitys tapahtuu yhteisissä ja muuttuvissa artefakteissa sekä alustukseen käytettävässä infrastruktuurissa. [14, s. 41–42] Kuvassa 3.2 on esitetty tuotantolinja osana muutoksenhallintaa.



Kuva 3.2. *Muutostenhallinta tuotantolinjaa hyödyntäessä, muokattu lähteestä [14, s. 42].*

Muutostenhallinnan kannalta tuotantolinja tarjoaa seuraavanlaisia [14, s. 42] etuja:

- Muutostenhallinnan vaikutuksen piirissä on vain yksi tuote, mikä vähentää hallittavien variaatioiden määrää.
- Kaikki muutokset tehdään yhteisiin ja tuotekohtaisiin artefakteihin, joten muutoksia ei tarvitse toistaa yksittäisiin tuotteen ilmentymiin.
- Muutokset ja parannukset alustaan ovat saatavilla kaikkiin tuotteen ilmentymiin.

Tuotantolinja ei kuitenkaan sovellu kaikkiin tuoterunkoihin. Se vaatii keskitetyn kehityksen, jotta saadaan luotua yleiset ja muuttuvat artefaktit sekä alustusinfrastruktuuri. Tuotantolinja ei myöskään tue jaettua kehitystä, jossa osa organisaatiosta kehittää alustaa ja osa yksittäisiä tuotteita, sillä kaikki kehitys tapahtuu yleisiin tai muuttuviin artefakteihin tai käytettävään alustusinfrastruktuuriin [14, s. 43]. Tuotantolinjan kanssa voidaan saavuttaa parempi muutostenhallinnan ylläpidettävyys, mutta samalla menetetään osa tuoterunkojen tarjoamasta joustavuudesta.

3.5 Tuoterunkojen haasteita

Tuotealustan omistajien tulee yhdessä tuotteen kehittäjien kanssa päättää, mitkä tuotekohtaiset ominaisuudet voidaan siirtää tuotealustan alaisuuteen [3, 21]. Alustan tulee tarjota riittävästi joustavuutta, jotta tuotteen eri variaatiot voidaan toteuttaa, mutta tämä kuitenkin lisää alustan monimutkaisuutta. Tuotealustan käyttämän tuoterunkoarkkitehtuurin suunnittelu ja toteutus ovat keskeisessä roolissa tuoteperheen menestyksen ja kannattavuuden kannalta [20].

Ylläpidon kannalta tuoterungot voivat olla haastavia. Esimerkiksi yksittäisessä tuotteessa havaitun virheen korjaaminen voikin vaatia muutoksia alustaan, joten korjaus voi vaikuttaa myös toisiin alustan päälle rakennettuihin tuotteisiin [18]. Lisäksi tuoterunkojen muunneltavuus asettaa omat haasteensa ylläpidettävyyteen, sillä uusien variaatiopisteiden lisääminen voi vaatia muutoksia alustan arkkitehtuuriin, mikä taas vaikuttaa kaikkiin tuotteen ilmentymiin [3].

Näiden vaikutusten vähentämiseksi on suositeltavaa, että variaatiopisteet suunnitellaan mahdollisimman itsenäisiksi toisiinsa nähden, eli variaatiopisteiden välillä ei ole merkittäviä riippuvuuksia. Samalla on tärkeää myös eristää yksittäisiin tuotteen ilmentymiin tehtävät muutokset. [18]

Kuten tyypillisesti ohjelmistokehityksessä, testaus on tärkeä osa tuoterunkojen kehitystä. Kehitettävän alustan ja varsinaisten tuoteversioiden kattava testaus on työlästä, sillä mitä enemmän muunneltavia ominaisuuksia tuotteesta löytyy, sitä enemmän on testattavia variaatioita. Tuoterunkojen testausta varten kannattaa testeille luoda oma testiarkkitehtuuri, joka ymmärtää tuoterunkojen muunneltavuutta. Hyvä tuoterungon testiarkkitehtuuri tukee testien automatisointia, jolloin testejä voidaan ajaa jatkuvasti. [17]

4 TUOTERUNKOARKKITEHTUURIN HYÖDYNTÄMINEN

Tuoterunkoarkkitehtuuri asettaa omat vaatimuksensa ohjelmiston arkkitehtuurille. Ohjelmiston yhteisessä osassa tulee nyt huomioida mahdolliset variaatiopisteet, jotta muunneltavuus olisi mahdollista [3]. Muunneltavuuden avulla mahdollistetaan tuoterunkoarkkitehtuurin tehokas hyödyntäminen, minkä seurauksena uusia tuotteita voidaan tuoda nopeammin ja tehokkaammin markkinoille. Tässä luvussa käydään läpi tuoterunkojen ja tuotteistamisen yhteensovitusta, sekä käytäntöjä, joiden avulla muunneltavuusvaatimukset voidaan ottaa huomioon ohjelmiston arkkitehtuurissa ja sen toteutuksessa.

4.1 Tuoterunko ja tuotteistaminen

Ohjelmiston tuoterunkokehitys on periaatteiltaan lähellä tuotteistamisen perusajatusta. Molempien tavoitteena on pyrkiä tuottamaan kannattavaa palvelua tai tuotetta, jolle on kysyntää markkinoilla [11, 21].

Ohjelmiston tuoterunkokehitys voi toimia tuotteistamisen osana, jos siihen liitetään myös markkinointi, myös yksittäisen asiakaskohtaisen projektin tapauksessa. Tuoterunkokehitys ei aseta rajoitteita siihen, miten kehitys pitäisi aloittaa.

Tuoterunkokehitys ja tuoterunkoarkkitehtuuri tarjoavat variaatiopisteiden ja hyvin dokumentoidun muunneltavuuden avulla mahdollisuuden luoda laajan skaalan tuotteita erilaisiin asiakkaiden tarpeisiin [21]. Yksittäisessäkin tapauksessa voidaan hyödyntää muunneltavuutta ja samalla varautua tulevaisuuteen.

4.2 Ohjelmiston muunneltavuus

Tässä luvussa käsitellään muunneltavuutta ohjelmistoarkkitehtuurissa, sekä ominaisuuslippuja tapana hallinnoida ja toteuttaa muunneltavuutta.

4.2.1 Muunneltavuus ohjelmistoarkkitehtuurissa

Tuoterunkoarkkitehtuuriin pohjautuvan ohjelmistoarkkitehtuurin tulee mahdollistaa erilaisten variaatioiden tekemisen ohjelmistosta. Tapoja, miten muunneltavuus voidaan huomioida ohjelmistoarkkitehtuurissa, on monia. Gacek ja Anastasopoulos esittävät [10] useam-

man tavan variaatiomekanismien toteuttamiseen ohjelmistoarkkitehtuurissa. Seuraavaksi käydään läpi joitakin tapoja, joiden avulla muunneltavuus voidaan ottaa huomioon ohjelmistoarkkitehtuurissa. Valittujen tapoja on rajattu sillä perusteella, mitä voidaan hyödyntää tyypillisissä olio-ohjelmointia tukevissa ohjelmointikielissä ilman erillisiä työkaluja.

Yksi käytetty tapa muunneltavuuden toteuttamiseen on hyödyntää perintää (engl. inheritance) [10, s. 111]. Perimisen avulla voidaan yleiskäyttöisiin kantaluokkiin toteuttaa perustoiminnallisuutta ja näistä kantaluokista periyttämällä voidaan toteuttaa variaatiokohtaisia muutoksia aliluokkiin. Haasteena on kuitenkin se, että mitä enemmän muunneltavaa toiminnallisuutta on, sitä enemmän tarvitaan aliluokkia, minkä seurauksena luokkahierarkian selkeys ja ylläpidettävyys heikkenevät [10, s. 111].

Toinen tapa muunneltavuuden toteuttamiseen on aggregaation ja delegoimisen hyödyntäminen [10, s. 111]. Aggregaatiossa ja delegaatiossa oliot jaetaan kahteen tyyppiin, delegoivaan ja delegoitavaan olioon. Delegoiva olio toimii eräänlaisena palvelunvälittäjänä ja kutsuu delegoitavan olion metodeja [20, s. 48]. Hyödyntämällä aggregaatiota oliot voivat tukea monipuolista toiminnallisuutta siirtämällä osan toiminnallisuudestaan delegoitavalle oliolle [10, s. 111]. Muunneltavuus voidaan toteuttaa siten, että delegoiva olio toteuttaa yhteisen toiminnallisuuden ja variaatiokohtainen toiminnallisuus toteutetaan delegoitavassa oliossa, joka voidaan antaa esimerkiksi delegoivan olion rakentajan parametrissa.

Delegoiminen toimii hyvin valinnaisten ominaisuuksien kanssa. Vastaavasti vaihtoehtoisten ominaisuuksien tukeminen voi olla haastavaa, sillä tyypillisesti delegoiva olio kutsuu vain yhtä delegoitavan olion jäsenfunktiota, jolloin vaihtoehtoinen toiminnallisuus joudutaan toteuttamaan omassa delegoitavassa oliossa, joka samalla toimii delegoivana oliona. Tuettavien variaatioiden määrän kasvaessa voi luokkien lukumäärä kasvaa niin suureksi, että ylläpidettävyys heikkenee. [10] Tyypillisesti aggregaatio tapahtuu ohjelman käännöksen aikana eikä sitä voida muuttaa ohjelmistoa ajettaessa, mutta on myös mahdollista hyödyntää esimerkiksi riippuvuusinjektiota olioiden riippuvuuksien selvittämiseen ohjelmiston ajon aikana.

Riippuvuusinjektiossa (engl. dependency injection) pyritään siirtämään riippuvuuksien alustaminen luokan sisältä sen ulkopuolelle siten, että riippuvuudet välitetään niitä käyttävälle luokalle. Tämä tekee luokista itsenäisempiä ja vähentää niiden välistä sidostuneisuutta. Riippuvuusinjektio on yksi tapa toteuttaa hallinnan muutossuunnan vaihtaminen (engl. inversion of control). [23] Hallinnan muutossuunnan vaihtamisessa pyritään siihen, että käytetty ohjelmistoviitekehys kutsuu kehittäjän toteuttamia metodeja sen sijaan, että kehittäjä kutsuisi viitekehyksen tarjoamia metodeja [7, 8].

Riippuvuusinjektio mahdollistaa ohjelmiston variaatioiden muokkaamisen ohjelmiston ajon tai sen käynnistyksen aikana. Esimerkiksi yksittäinen tuote voi toteuttaa oman versionsa alustan tarjoamasta rajapinnasta, joka ohjelmaa ajettaessa otetaan käyttöön alustakohtaisten toteutuksen sijasta. Tuotteen toteuttama rajapinta voi myös hyödyntää alustassa olevaa toteutusta omassa toteutuksessaan, tällöin täytyy vain toteuttaa kyseisen variaa-

tion vaatimat muutokset.

Alustan kehittäjät voivat myös hyödyntää viitekehyksen kehittämistä muunneltavuuden mahdollistamiseksi. Viitekehystä hyödyntämällä ja hallinnan muutossuuntaa vaihtamalla voi alusta tarjota tuotekehittäjille mahdollisuuden toteuttaa omia tuotekohtaisia variaatioita tietyissä paikoissa lähdekoodia. Sen sijaan, että tuotekehittäjä kutsuisi alustan tarjoamaa toiminnallisuutta, kutsuukin alustan tarjoama viitekehys tiettyjä metodeja tietyissä paikoissa, jotka tuotekehittäjä voi halutessaan toteuttaa.

Alustaan voidaan toteuttaa esimerkiksi olion tallennuksen yhteyteen tietokantaheräte, jolloin tuotekehittäjä voi lisätä tapahtumaan tarkempaa lokitusta. Toinen esimerkki on erilaiset rajapintakutsut. Alustassa voi olla oma viitekehys rajapintakutsujen tekemiseen, joka tarjoaa tuotekehittäjille mahdollisuuden reagoida rajapintakutsun eri vaiheisiin. Tällöin tuotekehittäjä voi lisätä omaa koodiaan, jota esimerkiksi ajetaan ennen jokaista rajapintakutsua tai jokaisen rajapintakutsun jälkeen. Hyödyntämällä viitekehystä alustan toteuttamisessa tekee alustasta laajennettavan.

Muita tyypillisiä tapoja huomioida muunneltavuus ohjelmistoarkkitehtuurissa ovat parametrisointi ja ylikuormitus [10, s. 111–112]. Parametrisoinnissa komponentit muuttuvat niiden saamien parametrien perusteella [10, s. 111]. Parametrisointiin voidaan hyödyntää esimerkiksi joissakin ohjelmointikielissä olevaa geneeristä ohjelmointia. Geneeristä ohjelmointia voidaan hyödyntää muun muassa yleiskäyttöisten tietorakenteiden toteutuksessa [19]. Esimerkkinä yleiskäyttöisestä tietorakenteesta on monissa ohjelmointikielissä oleva dynaaminen lista tai taulukko, esimerkiksi C#-kielen List<T>-luokka [16]. Tästä listasta voi olla ohjelmassa eri variaatioita olemassa riippuen listan alkioiden tyypistä. Geneerinen ohjelmointi mahdollistaa näiden eri variaatioiden olemassaolon ilman, että listan kehittäjän on tarvinnut tehdä jokaisesta eri variaatiosta oma toteutuksensa. Geneerinen ohjelmointi tukee siis parametrisointia ja täten muunneltavuutta.

Funktion ylikuormituksessa käytetään samaa nimeä, mutta muutetaan toiminnallisuutta esimerkiksi muuttamalla funktion parametreja [10, s. 111]. Ylikuormituksen hyödyntäminen vaatii tuen käytettävältä ohjelmointikieleltä. Ylikuormitus voi kuitenkin tehdä lähdekoodista vaikeammin ymmärrettävää kehittäjille, sillä käytettävä nimi ei välttämättä vastaa toteutettua toiminnallisuutta [10, s. 111].

Muita keinoja muunneltavuuden toteuttamiseen on olleet esimerkiksi C-pohjaisissa kielissä käytettävät esikääntäjän #ifdef ja #endif -lausekkeet, joilla voidaan ehdollistaa koodin kääntämistä riippuen näistä määrittelyistä [10, 12, 13, 21]. Näitä esikääntäjän lausekkeita voidaan kuitenkin hyödyntää vain ohjelmiston käännöksen aikana, joten ominaisuuksien muokkaaminen vaatii aina uuden käännöksen ohjelmasta.

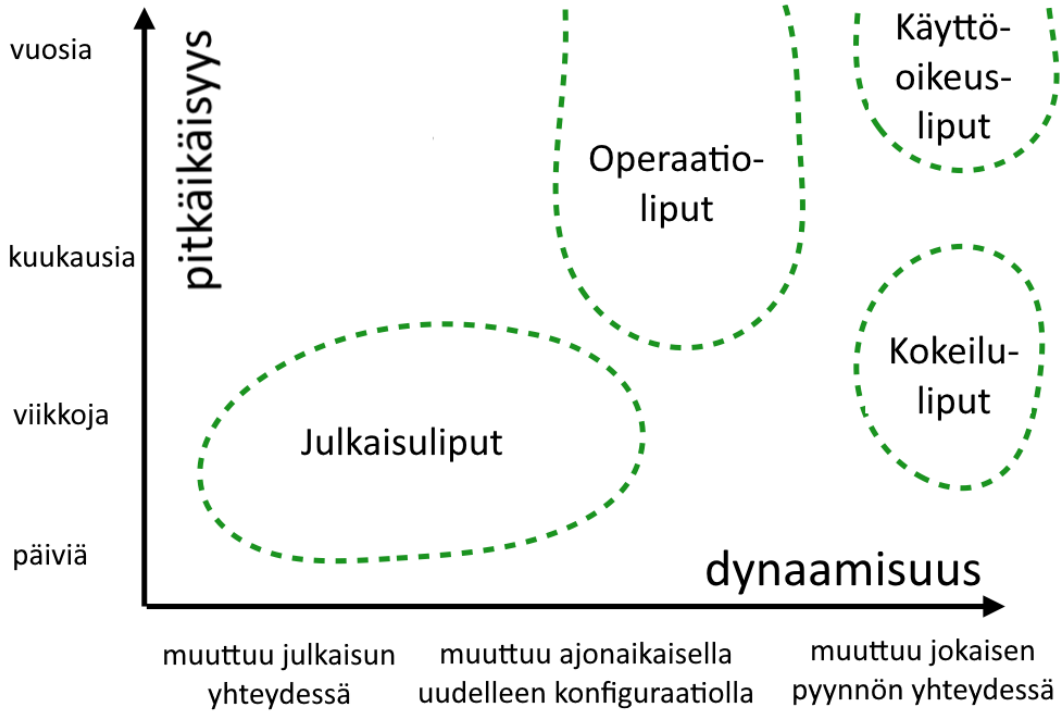
Muunneltavuus on keskeinen osa tuoterunkoarkkitehtuuria. Hyvin toteutetun muunneltavuuden avulla voidaan samaa tuoterunkoa käyttää useampaan erilaiseen tuotteeseen. Tällöin uudelleenkäyttö on huomattavasti parempaa ja kehitys on usein nopeampaa.

4.2.2 Ominaisuusliput

Yksi tapa toteuttaa muunneltavuutta ohjelmakoodissa on hyödyntää ominaisuuslippuja (engl. feature flag). Ominaisuuslippujen avulla voidaan muuttaa ohjelman käyttäytymistä ilman muutoksia ohjelmakoodiin. Ominaisuusliput lisäävät ohjelmakoodin monimutkaisuutta, jolloin niiden hyvä hallinnointi on tärkeää. [9]

Ominaisuuslippuja voidaan kategorisoida kahdella akselilla: ominaisuuslipun pitkäikäisyydellä ja sillä, kuinka dynaamisen ominaisuuslipun vaikutuksen tulee olla. Nämä eivät ole ainoat huomioitavat asiat, mutta näiden kahden avulla voidaan helpommin hallita erilaisia ominaisuuslippuja. [9]

Fowler esittää neljä erilaista ominaisuuslippua, jotka asettuvat dynaamisuuden ja pitkäikäisyyden mukaan omille paikoilleen. Nämä ovat: julkaisuliput, kokeiluliput, operaatioliput sekä käyttöoikeusliput. [9] Ominaisuusliput on esitetty kuvassa 4.1.



Kuva 4.1. *Ominaisuusliput jaoteltuna niiden pitkäikäisyyden ja dynaamisuuden mukaan, muokattu lähteestä [9].*

Julkaisuliput (engl. release flag) ovat staattisia ominaisuuslippuja, joiden elinkaari on usein muutamia viikkoja [9]. Julkaisulippujen avulla voidaan esimerkiksi julkaista uusia päivityksiä ohjelmistoon ilman, että julkaisulippujen takana olevat ominaisuudet ovat käytössä. Julkaisulippujen hyöty on siinä, että ne mahdollistavat ominaisuuksien julkaisemisen erillisenä niitä toteuttavan ohjelmakoodin julkaisusta.

Kokeiluliput (engl. experiment flag) ovat pääasiassa hyödyllisiä vain A/B-testauksessa. Tällöin näiden kokeilulippujen takana olevia ominaisuuksia voidaan dynaamisesti muuttaa siten, että eri tilanteissa käytetään eri ominaisuuksia. Tarkoituksena on kerätä tietoa uudesta ominaisuudesta ennen lopullisen päätöksen tekoa. [9]

Operaatiolippuja (engl. operations flag) käytetään hallinnoimaan ohjelmiston käyttäytymistä. Niitä voidaan hyödyntää esimerkiksi siinä tapauksessa, että uuden ominaisuuden tai toteutuksen suorituskyvystä ei olla vielä täysin varmoja. Jos uusi toteutus aiheuttaa ongelmia, voidaan se helposti kytkeä pois päältä. Operaatioliput ovat usein lyhytikäisiä, mutta poikkeuksiakin on, ja joskus operaatiolippu voi jäädä pitkäaikaiseksi. Operaatiolippujen tulee olla helposti muunnettavissa. [9]

Käyttöoikeusliput (engl. permissioning flag) muuttavat ohjelmiston ominaisuuksia tai käyttäytymistä käyttäjästä ja tämän käyttöoikeuksista riippuen. Tällaisia voivat esimerkiksi olla erilaiset maksulliset ominaisuudet tai uudet kehitteillä olevat ominaisuudet, jotka ovat saatavilla vain tietyille käyttäjille tai käyttäjäryhmille. Käyttöoikeusliput voivat olla erittäin pitkäikäisiä, mutta ne ovat kuitenkin dynaamisia. Käyttäjälle voidaan myöntää uusia käyttöoikeuksia, jolloin käyttöoikeuslippujen takana olevien ominaisuuksien tulee olla saatavilla. [9]

Ominaisuuslippujen avulla voidaan kehittää ja testata uusia ominaisuuksia ja variaatioita siten, että tarvittaessa voidaan poistaa uusi ominaisuus käytöstä ja ottaa käyttöön aiempi toteutus. Ominaisuuslippujen elinkaarta on tärkeä hallita hyvin, sillä ne lisäävät koodin monimutkaisuutta ja voivat tehdä koodista vaikealukuisempaa. Etenkin julkaisulippujen tulisi olla lyhytikäisiä. Ominaisuusliput ovat yksi tapa toteuttaa muunneltavuutta ohjelmistokoodissa ja ohjelmistossa. Ne täytyy dokumentoida muun muunneltavuuden osana, jotta tuotekehitysprosessin aikana voidaan niitä hyödyntämällä luoda ohjelmiston eri variaatioita.

5 YHTEENVETO

Tässä työssä tutkittiin, mitä tuoterungot ja tuoterunkoarkkitehtuuri ovat sekä miten niitä voidaan hyödyntää ohjelmiston tuotteistamisessa. Tuoterungot ovat ohjelmistoalustoja, joiden päälle voidaan muunneltavuuden avulla luoda uusia variaatioita tuotteesta. Tuoterungon ohjelmistoarkkitehtuuria kutsutaan tuoterunkoarkkitehtuuriksi.

Asiakaskohtaisen ohjelmiston tuotteistamiseen voidaan hyödyntää ohjelmiston tuoterunkokehitystä ja tuoterunkoarkkitehtuuria. Tuotteistamisella pyritään tuottamaan korkealaatuisia tuotteita kannattavasti. Tuoterunkojen avulla pyritään luomaan uusia tuotteita nopeasti, pienemmillä kustannuksilla ja korkeammalla laadulla verrattuna yksittäisten tuotteiden luomiseen.

Työssä pohdittiin myös, miksi tuoterunkoa ja tuoterunkoarkkitehtuuria tulisi hyödyntää tuotteistamisessa, miten tuoterunkoarkkitehtuurin voi ottaa huomioon ohjelmistokehityksessä ja miten tuoterunkoarkkitehtuuri vaikuttaa ohjelmistoarkkitehtuuriin.

Luvussa 2.2 vertailtiin Artz et al. esittämää tuotteistusprosessia Boschin esittämään tuotelinjan kypsyyden tasoihin. Tuotelinjan kypsyyden tasot sopivat hyvin yhteen esitetyn tuotteistusprosessin vaiheiden kanssa. Näin ollen kehittämällä tuoterunkoa voidaan samalla siirtyä seuraavalle tuotteistamisen vaiheista. Tuotteistamisessa on tärkeää kuitenkin analysoida markkinaa ja kehittää tuotetta myös potentiaalisten asiakkaiden tarpeiden mukaisesti. Tuotteistamisprosessista oli myös tehty tapaustutkimus, jossa onnistuttiin validoimaan Art et al. esittämän tuotteistusprosessin viisi ensimmäistä vaihetta.

Tuoterunkoarkkitehtuurin toteutukseen on ohjelmistoarkkitehtuurin näkökulmasta useita tapoja. Tuoterunkoarkkitehtuurilla on vaikutuksia ohjelmiston arkkitehtuuriin ja yleisesti ohjelmiston dokumentaatioon. Hallittavan muunneltavuuden tulee olla hyvin dokumentoitu, jotta tuotekehitys pysyy kannattavana.

Hyödyntämällä luvussa 4 esitettyjä tapoja voidaan tukea muunneltavuutta ja sen hallintaa. Ominaisuusliput ovat yksi mahdollinen tapa hallinnoida ja toteuttaa muunneltavuutta ohjelmistossa. Erilaisten ominaisuuslippujen avulla voidaan toteuttaa erilaista muunneltavuutta ohjelmistoon. Lyhytikäiset julkaisuliput soveltuvat uusien ominaisuuksien julkaisuun siten, että ne saadaan tarvittaessa pois käytöstä. Vastaavasti pitkäikäiset käyttöoikeusliput soveltuvat ohjelmiston muunteluun riippuen käyttäjän käyttöoikeuksista.

Ohjelmistosta tulee voida luoda uusia variaatioita erilaisiin tarpeisiin nopeasti ja hyvällä laadulla. Tuotteen laadulla on merkitystä kannattavuuden kannalta.

Jatkoa ajatellen tuoterungot ovat mielenkiinnon arvoisia. Ohjelmiston tuoterunkokehitys vaatii sen omaksuvalta organisaatiolta panostusta, mutta sillä voidaan päästä korkealaatuiseen ja kannattavaan ohjelmistokehitykseen.

LÄHTEET

[1] P. Artz, I. van de Weerd, S. Brinkkemper and J. Fieggen. Productization: Transforming from Developing Customer-Specific Software to Product Software. *Software Business*. Ed. by P. Tyrväinen, S. Jansen and M. A. Cusumano. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, 2010, 90–102. ISBN: 978-3-642-13633-7.

[2] F. Bachmann and P. C. Clements. *Variability in Software Product Lines*. CMU/SEI-2005-TR-012. Software Engineering Institute, Carnegie Mellon University, Sept. 2005. URL: https://apps.dtic.mil/docs/citations/ADA450337 (viitattu 4.7.2019).

[3] L. Bass, P. Clements and R. Kazman. *Software Architecture in Practice, Third Edition*. 3rd ed. Book, Whole. Addison-Wesley Professional, 2012.

[4] D. Batory. Product-Line Architectures. *Smalltalk and Java Conference* (1998), 12.

[5] J. Bosch. Maturity and Evolution in Software Product Lines: Approaches, Artefacts and Organization. *Software Product Lines*. Ed. by G. J. Chastek. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, 257–271. ISBN: 978-3-540-45652-0.

[6] E. Flamholtz. Managing organizational transitions: Implications for corporate and human resource management. *European Management Journal* 13.1 (Mar. 1, 1995), 39–51. ISSN: 0263-2373. DOI: 10.1016/0263-2373(94)00056-D. URL: http://www.sciencedirect.com/science/article/pii/026323739400056D (viitattu 27.7.2019).

[7] M. Fowler. *Inversion of Control Containers and the Dependency Injection pattern*. martinfowler.com. 2004. URL: https://martinfowler.com/articles/injection.html (viitattu 29.8.2019).

[8] M. Fowler. *Inversion Of Control*. martinfowler.com. 2005. URL: https://martinfowler.com/bliki/InversionOfControl.html (viitattu 1.9.2019).

[9] M. Fowler. *Feature Toggles (aka Feature Flags)*. martinfowler.com. 2017. URL: https://martinfowler.com/articles/feature-toggles.html (viitattu 3.6.2019).

[10] C. Gacek and M. Anastasopoules. Implementing Product Line Variabilities. *Proceedings of the 2001 Symposium on Software Reusability: Putting Software Reuse in Context*. SSR '01. event-place: Toronto, Ontario, Canada. New York, NY, USA: ACM, 2001, 109–117. ISBN: 978-1-58113-358-5. DOI: 10.1145/375212.375269. URL: http://doi.acm.org/10.1145/375212.375269 (viitattu 31.8.2019).

[11] J. Hietala, J. Kontio, J. Jokinen and J. Pyysiainen. Challenges of software product companies: results of a national survey in Finland. *10th International Symposium on Software Metrics, 2004. Proceedings.* 10th International Symposium on Soft-

ware Metrics, 2004. Proceedings. Sept. 2004, 232–243. DOI: 10.1109/METRIC.2004.1357906.

[12] C. Kästner, S. Apel and M. Kuhlemann. Granularity in software product lines. *2008 ACM/IEEE 30th International Conference on Software Engineering*. 2008 ACM/IEEE 30th International Conference on Software Engineering. May 2008, 311–320. DOI: 10.1145/1368088.1368131.

[13] C. Kästner and S. Apel. Feature-Oriented Software Development. *Generative and Transformational Techniques in Software Engineering IV: International Summer School, GTTSE 2011, Braga, Portugal, July 3-9, 2011. Revised Papers*. Ed. by R. Lämmel, J. Saraiva and J. Visser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, 346–382. ISBN: 978-3-642-35992-7. DOI: 10.1007/978-3-642-35992-7_10. URL: https://doi.org/10.1007/978-3-642-35992-7_10 (viitattu 3.6.2019).

[14] C. W. Krueger. Variation Management for Software Production Lines. *Software Product Lines*. Ed. by G. J. Chastek. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, 37–48. ISBN: 978-3-540-45652-0.

[15] W. Leenen, K. Vlaanderen, I. van de Weerd and S. Brinkkemper. Transforming to Product Software: The Evolution of Software Product Management Processes during the Stages of Productization. *Software Business*. Ed. by M. A. Cusumano, B. Iyer and N. Venkatraman. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, 2012, 40–54. ISBN: 978-3-642-30746-1.

[16] *List<T> Class (System.Collections.Generic)*. URL: https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1 (viitattu 6.9.2019).

[17] J. D. McGregor, L. M. Northrop, S. Jarrad and K. Pohl. Initiating software product lines. *IEEE Software; Los Alamitos* 19.4 (Aug. 2002), 24–27. ISSN: 07407459. DOI: 10.1109/MS.2002.1020282. URL: http://search.proquest.com/docview/215828194/abstract/D02729A134B7406FPQ/1 (viitattu 29.8.2019).

[18] K. Mohan and B. Ramesh. Change Management Patterns in Software Product Lines. *Commun. ACM* 49.12 (Dec. 2006), 68–72. ISSN: 0001-0782. DOI: 10.1145/1183236.1183269. URL: http://doi.acm.org/10.1145/1183236.1183269 (viitattu 3.6.2019).

[19] D. R. Musser and A. A. Stepanov. Generic programming. *Symbolic and Algebraic Computation*. Ed. by P. Gianni. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1989, 13–25. ISBN: 978-3-540-46153-1.

[20] L. M. Northrop and P. C. Clements. A Framework for Software Product Line Practice, Version 5.0. *SEI* (2012). URL: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=495357 (viitattu 4.9.2019).

[21] K. Pohl, G. Böckle and F. v. d. Linden, eds. *Software product line engineering: foundations, principles, and techniques*. OCLC: 254631989. Berlin: Springer, 2005. 467 pp. ISBN: 978-3-540-24372-4.

[22] A. Suominen, J. Kantola and A. Tuominen. Reviewing and Defining Productization. *The Proceedings of The International Society for Professional Innovation Manage-*

ment (ISPIM 2009) Conference: The Future of Innovation, Vienna, Austria (Jan. 1, 2009), 21–24.

[23] A. Yli-Sipilä. Riippuvuusinjektio : Joustavuutta arkkitehtuuriin löyhillä sidoksilla. AMK-opinnäytetyö. Metropolia Ammattikorkeakoulu, 2013. URL: http://www.theseus.fi/handle/10024/56745 (viitattu 1.9.2019).