Tuotteenhallinta tuoterunkoon pohjautuvissa ohjelmistoissa

Lauri Tuominen

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Ohjaaja: Timo Poranen
8.11.2009

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi
Lauri Tuominen: Tuotteenhallinta tuoterunkoon pohjautuvissa ohjelmistoissa
Pro gradu -tutkielma, 78 sivua + 9 liitesivua
Marraskuu 2009

Tiivistelmä

Tutkielmassa tarkastellaan tuotteenhallintaa tuoterunkoon pohjautuvissa ohjelmistoissa. Tarkastelu keskittyy erityisesti sellaisiin ohjelmistoihin, joissa on erikseen tuoterunko, ja siihen pohjautuvia asiakasversioita. Tutkielmassa kuvataan tuoterungon kehittämiseen tarvittava ohjelmistoprosessi ja esitellään tuotteenhallinnan peruskäsitteitä. Tutkielmassa pohditaan myös tuotteenhallinnan ja ohjelmistokehitysprosessien välistä vuorovaikutusta.

Tuotteenhallintaa käsitellään lähinnä projektin johtamisen näkökulmasta, erityisesti toiminnassa käytettäviä prosesseja. Tutkielmassa esitellään lyhyesti joitakin tuotteenhallintaa tukevia työkaluja ja apuvälineitä. Tutkielman tavoitteena on kartoittaa tuoterunkoon pohjautuvien ohjelmistojen tuotteenhallintaan liittyviä ongelmia, ratkaisuja ja hyviä käytäntöjä. Kirjallisuustarkastelun ohella tutkimuksessa hyödynnetään kahta esimerkkitapausta: toiminnanohjausjärjestelmää ja tuotannonohjausohjelmistoa. Tapaustutkimusten tuotteenhallinnan tasoa verrataan CMMI-kypsyysmallin asettamiin vaatimuksiin.

Avainsanat ja -sanonnat: tuoterunko, tuotteenhallinta, komponenttipohjainen ohjelmistokehitys, CMMI, kypsyysmalli.

Sisällys

1 Johdanto

Monen ohjelmistoyrityksen historiasta löytyy samankaltaisia piirteitä. Aluksi on kehitetty yhden asiakkaan tarpeisiin ohjelmisto ja myöhemmin päädytty tarjoamaan samaa muillekin asiakkaille. Tässä vaiheessa saatetaan puhua tuotteistamisesta, vaikka kyse olisikin lähinnä uudesta asiakaskohtaisesta toteutuksesta. Suppean ohjelmiston monistaminen ja muuttaminen jokaiselle asiakkaalle erikseen onnistuu kohtuullisella työllä ja on noin kolmanteen räätälöityyn versioon asti myös nopeampaa kuin yleiskäyttöisen tuoterungon kehittäminen [Linden *et al.*, 2007, 4].

Asiakkuuksien ja kaikkien eri tuoteversioiden yhteenlaskettujen ominaisuuksien määrän kasvaessa erillisten versioiden ylläpito käy vähitellen erittäin työlääksi. Jotta samankaltaisia piirteitä ei jouduttaisi ylläpitämään useammassa asiakasversiossa erikseen, on yleiskäyttöisiä piirteitä järkevää koota suuremmiksi kokonaisuuksiksi, ja muodostaa niistä tuoterunko, johon räätälöidyt asiakasversiot pohjautuvat. Tuoterunkoon ja siihen pohjautuviin asiakasversioihin kohdistuu jatkuvia muutoksia. Jokin vanhempi asiakasprojekti on toteutettu käyttäen sellaista tuoterunkoa, joka ei sisällä myöhemmin havaittujen vakavien sovellusvirheiden korjauksia. Kuitenkin tuoterunko on muuttunut matkan varrella niin paljon, ettei uusinta versiota tuoterungosta voida ottaa käyttöön sellaisenaan vanhassa asiakasversiossa.

Tuotteenhallinnan vastuualueelle kuuluvat ohjelmistotuotteiden *komponentit*, *konfiguraatiot* ja *toimintatavat*. Tuotteenhallintaan liittyy esimerkiksi sovelluksen eri versioihin tehtävien muutosmenettelyiden ohjeistaminen. Tuoterunkoon pohjautuvan ohjelmiston tuotteenhallinta on haasteellisempaa kuin yksittäisen tuotteen. Tuoterunkoon tehtävillä muutoksilla on suuri vaikutus tuoterunkoon pohjautuviin tuotteisiin. Pieni muutos tuoterungossa voi tehdä siihen pohjautuvista asiakasversiosta toimimattoman. Tuotteenhallintaa helpottaa erilaisten työkalujen ja apuvälineiden käyttö, mutta kokonaisuutta ohjaamaan tarvitaan aina sovitut toimintatavat ja niiden asianmukainen valvonta.

Tutkielmassa pohditaan tuoterungon kehittämistä lähinnä projektin johtamisen näkökulmasta. Tuotteenhallinnan osalta esitellään sekä sen perusteet että toteutuminen CMMI-kypsyysmallissa [CMMI Product Team, 2006]. Tutkielma sisältää myös toiminnanohjausjärjestelmän ja tuotannonohjausohjelmiston tuotteenhallintaa koskevat tapaustutkimukset. Tapaustutkimusten tuotteenhallinnan tasoa verrataan CMMI-kypsyysmallin tavoitteisiin.

Käsitys tuotteenhallinnan nykytilasta vaihtelee varsin paljon. Estublierin [2000] mukaan tuotteenhallinnan tutkimuksessa on jo saavutettu sellainen taso, että sen tuomia hyötyjä voi todella havaita käytännössäkin. Tutkimuksen painopiste on siirtynyt kohti riippumattomuutta mistään ohjelmointikielestä tai ohjelmistotuotteen semantiikasta. Haikala ja Märijärvi [2000, 238] kirjoittavat puolestaan seuraavasti: "Tuotteenhallinta liittyy kiinteästi tuotteeseen ja organisaation tapaan toimia, joten yleisiä reseptejä sen hoitamiseen on vaikea antaa."

Tutkimuksen tavoitteena on selvittää, millaisia tuotteenhallinnan ongelmia ilmenee tuoterunkoon pohjautuvissa ohjelmistoissa, ja miten näitä ongelmia voisi ratkaista. Tutkielmassa tarkastellaan myös tuotteenhallinnan roolia muutamassa ohjelmistokehitysmenetelmässä sekä esitellään lyhyesti joitakin tuotteenhallintaa helpottavia työkaluja. Kirjallisuustarkastelun ohella tutkimusmenetelmänä käytetään selittävää tapaustutkimusta.

Tuotteenhallinnan lähtötilanteena voidaan pitää tuotteenhallintasuunnitelman olemassaoloa. Muun muassa ISO 9001 -laatujärjestelmä [ISO, 2009] ja CMMI-kypsyysmalli [CMMI Product Team, 2006] edellyttävät sitä. Tuotteenhallinnan tukena kannattaa käyttää ensisijaisesti ohjelmistopohjaisia työkaluja, kuten tehtävänhallinta- ja versionhallintajärjestelmää. Tuotteenhallinnan vastuulla olevien muutosten hallinnointia voidaan tehostaa käyttämällä sovelluskoodin kääntämiseen jatkuvan integroinnin järjestelmiä [Fowler, 2009].

Tapaustutkimuksissa molempien tapausten merkittävimmäksi ongelmaksi nousi tiedonkulun ongelmat eri henkilöiden välillä. Toinen haaste liittyy puutteelliseen muutostenhallintamenettelyyn, joka oli havaittavissa erityisesti asiakasprojekteissa molemmissa tapauksissa. Tapausten kehitysehdotuksissa esitettiin tiedonkulun tehostamista säännöllisen sovelluskoodimuutosten läpikäyntiin keskittyvän tilaisuuden avulla sekä työkalujen käyttämistä helpompaan ja visuaalisempaan sovelluskoodimuutosten havainnointiin. Muutostenhallintamenettelyyn esitettiin toiminnanohjausjärjestelmän osalta tuoterungon kehittämisessä hyväksi havaittujen käytäntöjen asteittaista jalkauttamista asiakasprojekteissa. Tuotannonohjausohjelmiston tapauksessa muutostenhallintaan esitettiin tehtävänhallintajärjestelmän työnkulkuun liittyvää muutosta, joka edellyttäisi että joku muukin, kuin muutoksen toteuttaja, hyväksyy ehdotetun muutoksen ennen tapahtuman merkitsemistä valmiiksi.

Tutkielman alkupuoli painottuu tuoterunkoon pohjautuvan tuotteen kehittämisen esittelyyn ja tuotteenhallinnan perusteiden läpikäyntiin. Tutkielman loppupuoli keskittyy kahteen tuotteenhallintaan liittyvään tapaustutkimukseen.

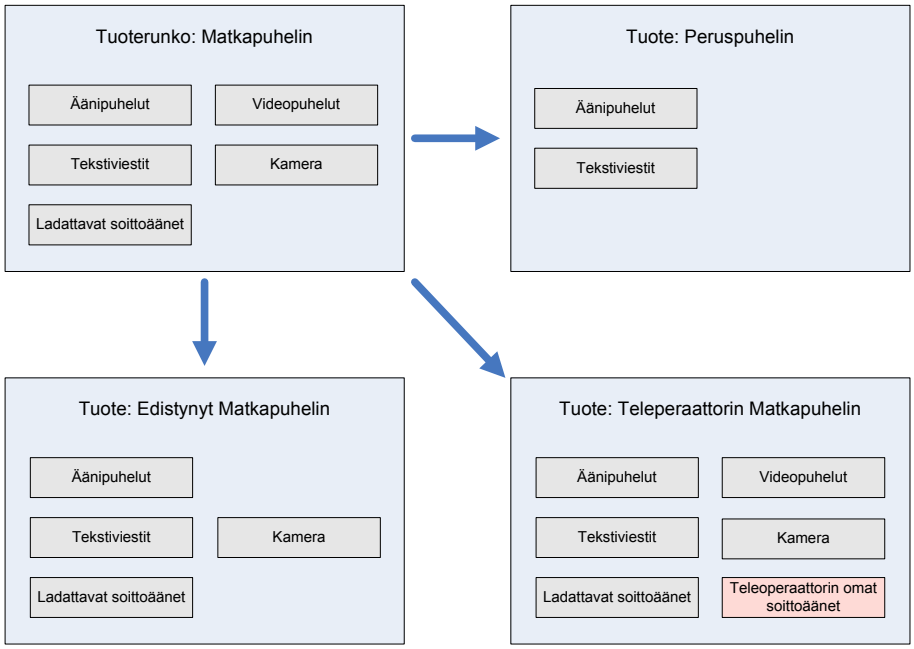
Luvussa 2 käydään läpi tuoterunkoon pohjautuvan ohjelmiston kehittämisen peruskäsitteitä ja menetelmiä. Luku 3 keskittyy tuotteenhallinnan ja sen tavoitteiden esittelyyn sekä havainnollistaa sen toimintaa osana ohjelmistokehitystä. Luvussa 4 esitellään lyhyesti työkaluja, jotka helpottavat tuotteenhallintaa ohjelmistoprojekteissa. Luvussa 5 pohditaan tuotteenhallinnan roolia ISO 9001 -laatujärjestelmässä sekä CMMI-kypsyysmallissa. Luvut 6 ja 7 sisältävät tutkielmaan liittyvät tapaustutkimukset. Luvussa 8 vertaillaan tapauksia CMMI-kypsyysmallin tuotteenhallintaa koskeviin vaatimuksiin. Luvussa 9 on esitetty tutkielman tulokset ja tapaustutkimuksiin liittyvät kehitysehdotukset.

2 Tuoterunkoon pohjautuvan ohjelmiston kehittäminen

Ohjelmistojen toiminnan ja rakenteen ymmärtäminen pelkkien tekstimuotoisten kuvausten perusteella on vaikeaa. Tekstimuotoisten kuvausten tukena on siksi jo pitkään käytetty erilaisia mallinnuskieliä. Ohjelmistokehityksessä mallinnuskielen tarkoituksena on esittää ohjelmistoon rakenteeseen, käyttäytymiseen ja vuorovaikutukseen liittyviä asioita standardoidulla merkintätavalla eli notaatiolla. Tämän hetken todennäköisesti suosituin mallinnuskieli on Unified Modelling Language, UML [OMG, 2009], joka on varta vasten suunniteltu ohjelmistojen kuvaamiseen.

Tässä tutkielmassa käytetään joissakin kuvissa UML-kaavioita havainnollistamaan ohjelmistojen rakenteeseen ja tuotteenhallintaan liittyviä asioita. Helppotajuinen teos UML:n perusteista on esimerkiksi Kimmelin [2006] kirja. Myös virallinen standardi [OMG, 2009] sisältää paljon esimerkkejä eri kaavioista.

Kuva 2.1 havainnollistaa tuoterungon ja siitä johdettujen tuotteiden välisiä suhteita kuvitteellisen matkapuhelin-esimerkin avulla.



Kuva 2.1 Tuoterungon ja tuotteiden yhteys.

Matkapuhelin-tuoterunko käsittää eri ominaisuuksia, kuten äänipuhelut, tekstiviestit ja kamera. Tuotteisiin voidaan valita ominaisuuksia tuoterungosta. Pe-

ruspuhelimessa on ominaisuuksina ainoastaan äänipuhelut ja tekstiviestit. Edistyneessä matkapuhelimessa, on lisäksi kamera ja ladattavat soittoäänet. Teleoperaattorin omalla nimellään lanseeraama puhelinmalli sisältää kaikkien tuoterungon ominaisuuksien lisäksi operaattorin omat soittoäänet. Tuoterunko ei sisällä kyseistä ominaisuutta, joten sitä ei voida valita suoraan uusiin tuotteisiin. Esimerkkitapauksessa tuoterungon pohjalta koostettuun tuotteeseen on lisätty *tuotekohtainen ominaisuus*.

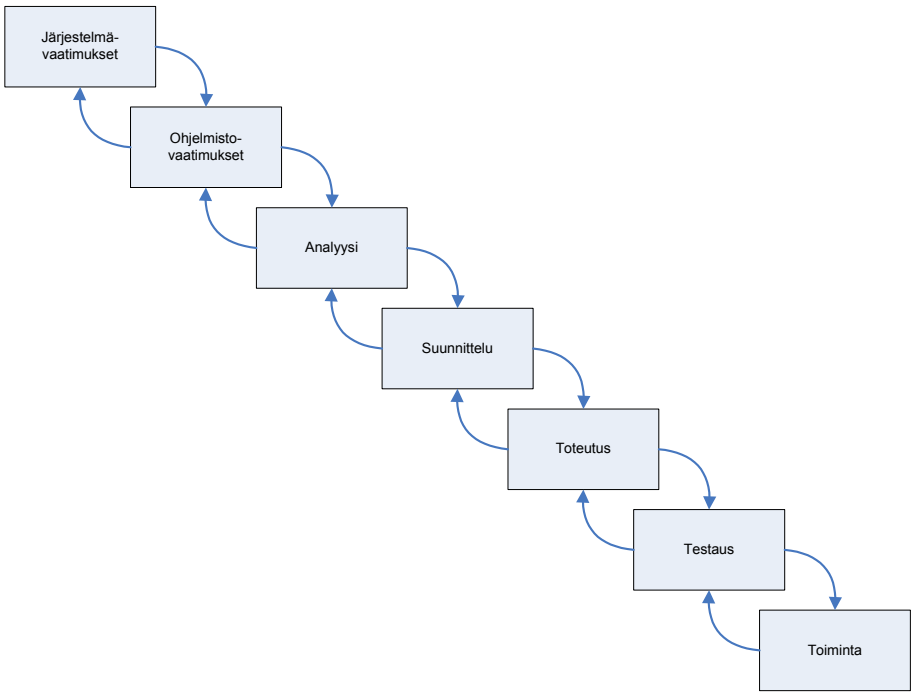
2.1 Ohjelmistokehitysmalleista

2.1.1 Vesiputousmalli

Ohjelmiston kehittäminen ideasta valmiiksi tuotteeksi vaatii usein paljon työtä. Kaikkein pelkistetyimmilläänkin ohjelmiston kehittäminen koostuu sen suunnittelusta ja toteuttamisesta. Vaikka toteutettavan ohjelman tehtävänä olisi tulostaa vain yksi tekstirivi näytölle, joutuu sen toteuttaja suunnitelemaan ohjelman mielessään ennen ohjelmakoodin kirjoittamista. Ohjelmistokehityksen jakamisen vaiheisiin esitti ensimmäisen kerran Benington [1983] jo vuonna 1956. Myöhemmin vuonna 1970 Royce [1987] esitteli nykyisin ehkä tunnetuimman ohjelmistokehitysmallin, jota alettiin kutsua myöhemmin vesiputousmalliksi. Royce pyrki havainnollistamaan prosessin avulla ohjelmistokehitysprosessiin liittyviä haasteita, riskejä ja niiden ehkäisyä. Vesiputousmallista on olemassa useita eri muunnelmia, mutta yleensä niistä voidaan erottaa ainakin määrittely-, suunnittelu- ja toteutusvaiheet [Haikala & Märijärvi, 2000, 25]. Kuva 2.2 on suomennettu versio alkuperäisestä Roycen esittämästä ohjelmistokehitysprosessista.

Ohjelmistokehitysprosessin ensimmäiseksi vaiheeksi kuvataan joissain vesiputousmallien muunnelmissa (esimerkiksi [Haikala & Märijärvi, 2000, 24]) esitutkimusvaihe, jossa selvitetään aiotun ohjelmistotuotteen kehittämisen edellytyksiä. Esitutkimuksessa voidaan arvioida esimerkiksi aiotun tuotteen kehityskustannuksia ja selvittää kilpailevien tuotteiden olemassaoloa ja markkinatilannetta.

Roycen malli lähtee liikkeelle ohjelmiston määrittelyvaiheesta (järjestelmävaatimukset ja ohjelmistovaatimukset). Merkittävä osa ohjelmiston määrittelyä on kartoittaa ja kuvata ohjelmistolle asetettavat vaatimukset. Yksinkertaisimmillaan vaatimukset voivat olla yksittäisiä lauseita, jotka kuvaavat jollakin tapaa järjestelmälle tai ohjelmistolle asetettuja vaatimuksia. Esimerkki järjestelmävaatimuksesta voisi olla ”Järjestelmä voidaan sammuttaa yhtä nappulaa painamalla”. Roycen mallissa ylimmällä tasolla ovat järjestelmävaatimukset, jotka puo-



Kuva 2.2 Ohjelmistokehitysprosessin vaiheet laajan tietokoneohjelman toimittamiseksi asiakkaalle [Royce, 1987, 330].

lestaan ohjaavat ohjelmistovaatimuksia. Esimerkki ohjelmistovaatimuksesta voisi olla ”Ohjelmiston pääkäyttäjä voi tulostaa raportin kaikista järjestelmän käyttäjistä”.

Periaatteessa koko järjestelmän määrittely voitaisiin tehdä pelkästään vaatimusten avulla. Yleensä määrittelyyn liitetään varsinaisten vaatimusten lisäksi myös muuta järjestelmän toimintaa kuvaavaa dokumentaatiota kuten tietosisältökaavioita ja laitteisto- ja ohjelmistorajoitteita. Tietosisältökaaviot kuvaavat sen, millaista tietoa määriteltävällä järjestelmällä käsitellään. Esimerkkinä tästä voisi olla opiskelija-tietue, joka sisältää kentät *etunimi*, *sukunimi*, *sosiaaliturvatunnus*, *aloitusvuosi* ja *koulutusohjelma*. Laitteisto- ja ohjelmistorajoitteet puolestaan rajaavat järjestelmä- ja ohjelmistovaatimuksia. Esimerkki laitteistorajoitteesta: ohjelmiston toimivuutta ei taata muissa puhelimissa kuin Nokia E60. Esimerkki ohjelmistorajoitteesta: ohjelmiston asentaminen edellyttää, että kohdeympäristöön on asennettu Microsoft Windows Vista –käyttöjärjestelmä. Määrittelyvaiheesta syntyy tuloksena vaatimusmäärittelydokumentti, jota käytetään suunnitteluvaiheen syötteenä.

Analyysivaiheessa käydään läpi järjestelmä- ja ohjelmistovaatimukset ja pyritään varmistumaan siitä, että järjestelmän suunnittelu ja toteutus voidaan tehdä asetettujen vaatimusten pohjalta. Jos esitetyissä vaatimuksissa on ristiriitoja, joudutaan palaamaan takaisin ohjelmistovaatimusten kartoitusvaiheeseen.

Ohjelmiston suunnitteluvaiheessa mietitään ohjelmiston tekninen toteutus. Muun muassa määrittelyvaiheessa kuvatut tietosisältökaaviot tarkennetaan lopulliseen muotoonsa ja päätetään tietueiden kenttien tietotyypit (esimerkiksi opiskelija-tietueen *sukunimi*-kenttä on maksimissaan 25 merkkiä pitkä merkkijono). Viimeistään suunnitteluvaiheessa päätetään käytettävästä ohjelmistoarkkitehtuurista. Ohjelmistoarkkitehtuurin nykyisen määritelmän luoneet Bass ja kumppanit [2003, 3] kuvaavat vapaasti suomennettuna ohjelmistoarkkitehtuurin tarkoittavan seuraavaa:

> *Ohjelmiston tai järjestelmän ohjelmistoarkkitehtuuri koostuu rakenteesta tai rakenteista jotka kuvaavat järjestelmän ja sen ohjelmiston osat, näiden osien ulospäin näkyvät ominaisuudet ja riippuvuudet niiden välillä.*
>
> [Bass *et al.*, 2003, 3]

Ohjelmistoarkkitehtuurin valinnalla on suuri merkitys erityisesti tuoterungon kehittämisessä, koska tehty valinta ohjaa myös tuoterunkoon pohjautuvien myöhempien tuotteiden suunnittelua ja toteutusta.

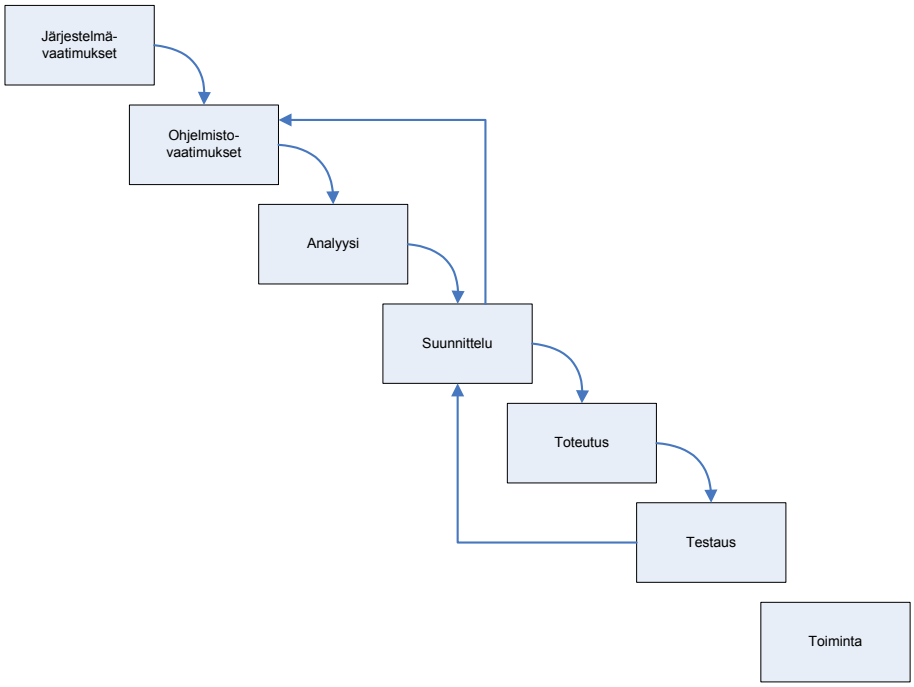
Toteutusvaiheessa kirjoitetaan ohjelmiston sovelluskoodi, luodaan mahdolliset ohjelmiston tarvitsemat tietokannat ja tarvittaessa liitetään eli integroidaan sovellus osaksi muita järjestelmiä tai muut järjestelmät osaksi toteutettavaa ohjelmistoa.

Testausvaiheen syötteenä toimivat määrittelyvaiheessa kerätyt järjestelmä- ja ohjelmistovaatimukset. Testausvaiheessa varmistutaan, että toteutettu ohjelmisto täyttää kaikilta osin nämä vaatimukset. Tarvittaessa ohjelmisto palautetaan takaisin toteutusvaiheeseen havaittujen puutteiden ja virheiden korjaamiseksi. Toimintavaiheella tarkoitetaan ohjelmiston käyttöönottoa ja käyttämistä.

Vesiputousmalli on erikoistapaus yleisestä ongelmanratkaisumallista. Käytännössä ohjelmistokehitys ei koskaan voi edetä kirjaimellisesti sen mukaan, koska muun muassa osa vaatimuksista selviää vasta projektin edetessä ja vaatimukset muuttuvat lähes poikkeuksetta ajan mittaan [Haikala & Märijärvi, 2000, 29].

2.1.2 Evo-malli

Vesiputousmalli ei ole suinkaan ainoa ohjelmistokehityksessä käytetty elinkaarimalli. Se kuitenkin kuvaa hyvin ne vaiheet, jotka käytettävästä ohjelmistokehitysprosessista riippumatta on huomioitava tavalla tai toisella. Royce [1987] havaitsi, ettei ohjelmistoa pystytä käytännössä kehittämään valmiiksi vaiheiden yhdellä läpikäynnillä (kuva 2.3).



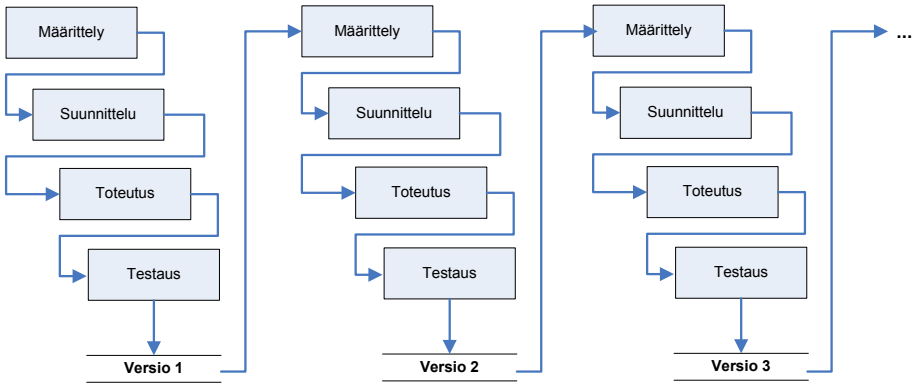
Kuva 2.3 Ohjelmistoprosessin edetessä joudutaan palamaan edellisiin vaiheisiin [Royce, 1987, 330].

Vesiputousmallin testausvaiheessa ohjelmiston *tilaaja* (asiakas) pääsee testaamaan järjestelmää. Vaikka järjestelmä täyttäisikin kaikki määrittelyvaiheessa kerätyt järjestelmä- ja ohjelmistovaatimukset, saattaa asiakas havaita jonkin ensiarvoisen tärkeän toiminnon puuttuvan toteutetusta ohjelmistosta. Tällöin joudutaan palaamaan takaisin vaatimusten keräysvaiheeseen ja päivittämään määrittelyä. Asiakas voi myös todeta jonkin toiminnon täyttävän vaatimusten kirjaimen, mutta toimivan kuitenkin toisella tavoin kuin asiakas tarkoitti. Tällöin joudutaan palaamaan takaisin ohjelmiston suunnittelu- tai toteutusvaiheeseen.

Royce [1987] ehdotti vesiputousmallin läpikäyntiä kahteen kertaan. Ensimmäisellä kierroksella vaiheet käydään läpi kevyemmin, jolloin lopputuloksena saa-

daan prototyyppi valmiista järjestelmästä. Tämän pohjalta käydään kaikki vaiheet vielä uudelleen läpi ja toteutetaan varsinainen tuote.

Evo-malli (engl. evolutionary delivery) pohjautuu samankaltaiseen ajatukseen sillä erotuksella, että kahden läpikäynnin asemesta vesiputousmalli käydään läpi useampia kertoja (kuva 2.4).



Kuva 2.4 Evo-malli [Haikala & Märijärvi, 2000, 30].

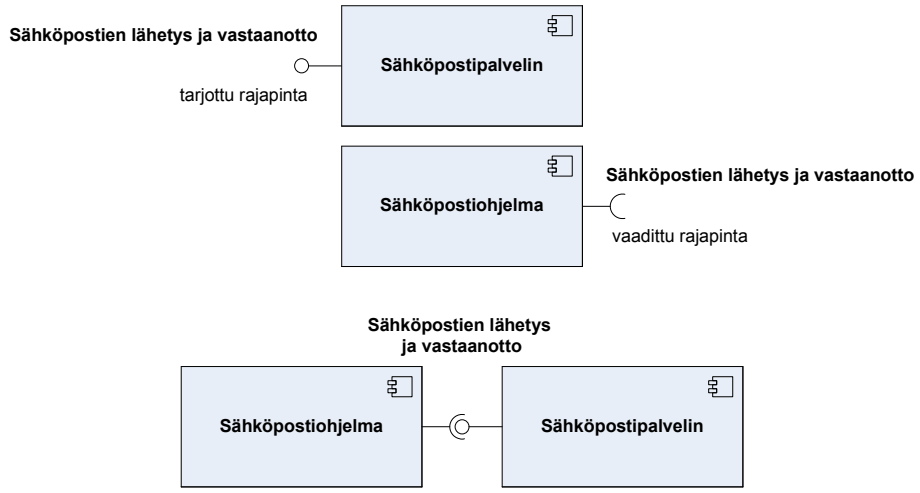
Jokaisen läpikäynnin tuloksena syntyy toimiva järjestelmä, joka ensimmäistä lukuun ottamatta lisää ominaisuuksia edelliseen. [Gilb, 1988] Evo-malli soveltuu erinomaisesti sekä ohjelmistorungon että asiakaskohtaisten versioiden ohjelmistokehitysmenetelmäksi, koska tuoterunko on luonteeltaan jatkuvasti kehittyvä ja asiakasversioihinkin halutaan viedä sekä tuoterunkoon tehtyjä parannuksia että asiakaskohtaisia ominaisuuksia. Tuoterungosta ja asiakasversioista voidaan julkaista uusia versioita tarpeen mukaan. Tuoterungosta voidaan julkaista uusi versio vaikkapa kerran vuodessa ja asiakasversioista tarpeen mukaan tiheämmin tai harvemmin.

2.2 Komponentti

Komponentin määritelmä on varsin väljä. Ohjelmistokehityksessä se voi tarkoittaa mitä tahansa ohjelmistoon kuuluvaa fyysistä, itsenäisesti eroteltavissa olevaa osaa kuten yksittäistä dokumenttia tai sovelluskooditiedostoa. Komponenteilla voidaan mallintaa asioita hyvin karkealla tasolla, esimerkiksi sähköpostipalvelin voi olla yksi komponentti. Toisaalta sähköpostipalvelimen toiminta voidaan jakaa edelleen uusiin alikomponentteihin. Esimerkiksi sähköpostin vastaanottamista ja lähettämistä varten voitaisiin mallintaa omat komponenttinsa.

UML-standardin [OMG, 2009, 146] mukainen komponentin määritelmä vapaasti suomennettuna on: *Komponentti kuvaa järjestelmän osaa kapseloiden sisältönsä ja sen ilmentymä on korvattavissa ympäristössään.*

Yksittäisistä komponenteista voidaan muodostaa ohjelmisto yhdistelemällä niitä toisiinsa. Jokainen komponentti tarjoaa sisältämiään palveluita muiden komponenttien käytettäväksi, tai vaihtoehtoisesti tarvitsee toimiakseen muiden komponenttien tarjoamia palveluita (kuva 2.5). Yksi komponentti voi myös toimia molemmissa rooleissa: sekä tarjota palveluita että käyttää niitä.

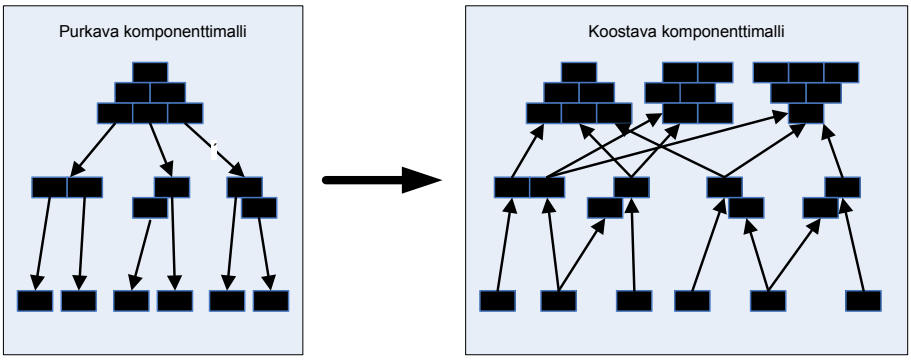


Kuva 2.5 Tarjotut ja vaaditut rajapinnat kuvattuna UML-notaatiolla.

Kuvan sähköpostipalvelinkomponentti tarjoaa rajapinnan sähköpostien lähettämistä ja vastaanottamista varten. Sähköpostiohjelmakomponentti sisältää rajapinnan sähköpostien lähettämiseksi ja vastaanottamiseksi, mutta ei itse pidä sisällään tarvittavaa toiminnallisuutta. Toimiva kokonaisuus saadaan liittämällä yhteen molemmat komponentit.

2.3 Komponenttipohjainen tuote

Perinteisessä ohjelmistokehityksessä on totuttu ajattelemaan järjestelmää suurena kokonaisuutena, joka on purettu (engl. decomposition) alijärjestelmiksi ja edelleen pienemmiksi komponenteiksi (kuva 2.6). Ongelmana on, että täysin sama toiminnallisuus saattaa tulla toteutetuksi ohjelmiston useampaan paikkaan jopa täysin samanlaisena, jolloin muutostarpeiden ilmetessä kaikki saman toiminnon ilmentymät joudutaan päivittämään.



Kuva 2.6 Purkavasta komponenttimallista koostavaan [van Ommering, 2002, 260].

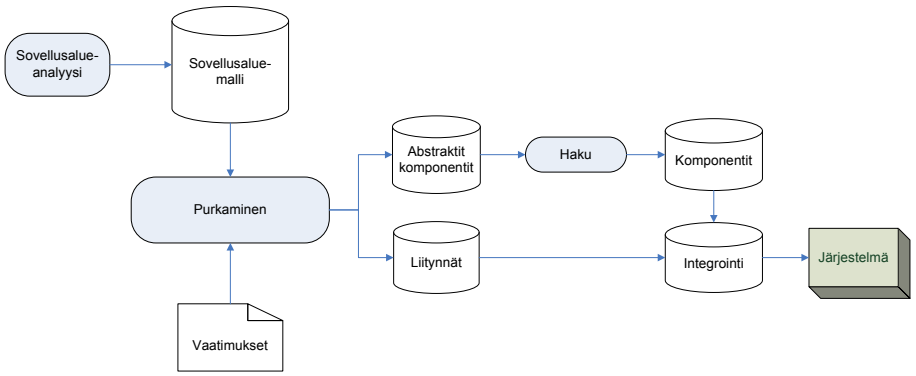
Asiaa voidaan ajatella myös päinvastaisesti, eli kokoamisen kannalta. Tällöin koko järjestelmä rakennetaankin olemassa olevista komponenteista (kuva 2.6). Koostamiseen (engl. composition) pohjautuvassa kehitystavassa on merkittävänä etuna se, ettei sama ominaisuus tule niin helposti toteutetuksi moneen kertaan eri komponentteihin.

Lisäksi komponenttien sisältö ja rakenne on mietittävä kokoamiseen pohjautuvassa menetelmässä paljon paremmin. Purkamiseen perustuvassa tavassa riittää, että ominaisuus on toteutettu jossain järjestelmän komponentissa. Kokoamiseen pohjautuvassa lähestymistavassa on puolestaan huomioitava komponentin erilaiset käyttötavat. Komponentista ei saa tulla liian tuotespesifistä, mutta ei myöskään liian yleiskäyttöistä. Olennaista on, että komponentti voidaan liittää haluttuihin tuotteisiin tai tuoteperheisiin. [van Ommering, 2002, 260]

Yleisesti ottaen useimpiin komponenttipohjaisiin kehitysmalleihin liittyvät lähestymistavat ovat olleet tavalla tai toisella vajavaisia. [Doğru, 2005, 51] Ongelmana on ollut siis esimerkiksi edellä kuvattu purkava kehitysmalli, joka ei ohjaa komponenttisuunnittelua kohti yleiskäyttöisiä komponentteja.

Sovellusalueen samankaltaisten piirteiden löytämiseen on olemassa menetelmä: sovellusalue-analyysi. Sovellusalueanalyysissa ei olla kiinnostuneita yksittäisistä järjestelmän toiminnoista. Sen sijaan sovellusalueanalyysi pyrkii löytämään yhtenevät toiminnot koko sovellusalueelta. [Neighbors, 1980, 1–6] Yksinkertainen esimerkki asiasta on taloushallinnon ohjelmisto, jota suunniteltaessa havaitaan, että tulostustoiminnallisuutta tarvitaan sekä budjettien tulostamisessa että kassapäätetoiminnoissa laskujen tulostamiseksi.

Komponenttien uudelleenkäyttöä voidaan tehostaa sisällyttämällä sovellusalueanalyysi osaksi komponenttipohjaista kehitysmallia (kuva 2.7).



Kuva 2.7 Komponenttisuuntautunut kehitysmalli [Doğru, 2005, 53].

Kehitysmallin ajatuksena on suorittaa ensin sovellusalueanalyysi ja purkaa sen jälkeen sovellusaluemalli ja vaatimukset abstrakteiksi komponenteiksi ja liittymiksi. Seuraavaksi abstraktit komponentit toteutetaan joko hankkimalla valmiita komponentteja tai toteuttamalla uusia. Yhtä abstraktia komponenttia kohden voidaan tarvita useampia valmiita komponentteja. Integrointivaiheessa toteutetut ja hankitut komponentit liitetään yhteen, mistä syntyy valmis järjestelmä.

2.4 Tuotelinjaan pohjautuva tuote

Tuotelinja-arkkitehtuurin ajatuksena on luoda yleiskäyttöinen pohja eli tuotelinja, jota jalostamalla voidaan tehdä useita muita tuotteita. Kokonaisuutta, joka sisältää sekä tuotelinjan että sitä käyttävät tuotteet, kutsutaan tuoteperheeksi. Sinällään tuotelinja-ajattelu ei ole erityisesti ohjelmistotuotantoon liittyvä asia, vaan yhtä hyvin tuotelinja-ajattelua sovelletaan esimerkiksi autoteollisuudessa. Parhaiten tuotelinja-ajattelu soveltuu käytettäväksi sellaisissa tilanteissa, joissa voidaan valmistaa suuria määriä vain hieman toisistaan poikkeavia tuotteita, esimerkiksi pienin muutoksin eri matkapuhelimissa toimivat sovellukset.

Tuotelinja-ajattelun ehkä merkittävin ero ohjelmistokehityksessä verrattuna perinteiseen ajattelutapaan liittyy työskentelyn tavoitteisiin. Perinteisessä ohjelmistokehityksessä keskitytään kehittämään yleensä yhtä järjestelmää ja projektia kerrallaan [Linden *et al.*, 2007, 6], jolloin tavoitteet ovat lähinnä tuotteen toimittamisessa ja aikataulussa pysymisessä.

Tuotelinja-ajattelussa keskitytään sen sijaan tuotteen hyvään ylläpidettävyyteen ja uudelleenkäyttävyyteen [Linden *et al.*, 2007, 70]. Perinteisestä tavasta tuotelinja-ajatteluun siirtyminen on aina strateginen valinta.

Ohjelmistotuotannossa tuotelinja-ajattelu voidaan jakaa neljään pääkohtaan [Linden *et al.*, 2007, 7–8]:

1. Tuotevariaatioiden hallinta: tuotteen ytimessä tapahtuu eri tuotteiden välisten yhtenevyyksien ja variaatioiden hallinta.
2. Liiketoiminnan näkökulma: tuotelinja-ajattelu on liiketoimintakeskeistä ja sen pitää olla linjassa liiketoiminnan strategioiden kanssa ja vastata markkinoiden kysyntään. Kun eri komponentteja rakennetaan eri tuotteisiin, pitää huomioida myös pitkän tähtäimen vaikutukset. Tuotelinjan pitää olla sidoksissa liiketoiminnan strategioihin myös siltä osin.
3. Arkkitehtuurinäkökulma: tuoterungon arkkitehtuuri vaikuttaa keskeisesti siitä johdettujen tuotteiden arkkitehtuureihin. Yleisesti uskotaan että korkea uudelleenkäytettävyyden taso voidaan saavuttaa vain yhteisellä arkkitehtuurilla tuoterungon ja tuotteiden kesken.
4. Kaksivaiheinen elinkaariajattelu: tuotelinja koostuu sovellusalueanalyysiin ja itse toteutusvaiheesta. Uudelleenkäytön kannalta on huomioitava sekä sunnittelu että toteutus siten, että tuotettuja komponentteja voidaan käyttää uudelleen ja toisaalta toteutusvaiheessa pitää pyrkiä käyttämään uudelleen jo aiemmin toteutettuja komponentteja.

2.4.1 Tuotelinja-arkkitehtuuriin siirtyminen

Tuotelinja-arkkitehtuurin voidaan siirtyä kolmella eri tavalla:

- proaktiivisesti
- reaktiivisesti
- inkrementaalisesti.

Proaktiivisessa mallissa tuotelinja määritellään, suunnitellaan ja toteutetaan ensimmäisenä. On havaittu, että tuotelinjan käyttöottokustannukset saadaan peitettyä yleensä noin kolmannen tuotteen kohdalla, joskus aiemminkin. [Linden *et al.*, 2007, 4] Tuotelinjan kehittäminen proaktiivisesti on vaihtoehdoista aluksi eniten työpanosta vaativa, koska tuotelinjan kehitystyö tehdään kokonaisuudessaan yhdellä kertaa.

Reaktiivisessa mallissa kehitetään ensin yksi tai useampia tuotteita, jolloin aloituskustannukset ovat pienemmät kuin proaktiivisessa mallissa. Arkkitehtuuri ja muut ydinseikat pitää saada vakaiksi ja laajennettaviksi, jotta siirtyminen tuotelinja-arkkitehtuuriin olisi tulevaisuudessa helpompaa.

Inkrementaalinen malli on hyvin pitkälti samanlainen kuin proaktiivinen. Siinä suunnitellaan alusta asti vaiheittainen siirtyminen tuotelinja-arkkitehtuuriin. Sen aloituskustannukset ovat pienemmät, koska koko tuotelinjan vaatimaa työtä ei tehdä kerralla.

2.4.2 Soveltuminen pieniin yrityksiin

Tuotelinja-arkkitehtuurin käyttöönoton suurimpia haasteita on tarvittava alkuinvestointi ja riskin ottaminen sen suhteen, että valmistettaville tuotteille löytyy kysyntää. Tuotelinja-arkkitehtuurin käyttöönoton aloituskustannuksia voidaan alentaa huomattavasti lähtemällä liikkeelle reaktiivisella kehitystavalla tekemällä ensin yksi tuote ja jalostamalla siitä muiden tuotteiden pohja.

Yksi tuotelinja-arkkitehtuurin menestystarina pienessä yrityksessä on osakemarkkinoiden seurantaan tarkoitettu tuote Market Maker. [Gacek *et al.*, 2001] Tuotteen kehittämisen aloitti vuonna 1989 yksi henkilö ja vuonna 2001 sitä oli toteuttamassa lähes 60 henkilöä. Varsinaisen tuotelinja-arkkitehtuurin aika alkoi tuotteen kohdalla, kun sen olemassa oleva Windows-työasemakäyttöön toteutettu sovellus kapseloitiin COM-komponentiksi (Component Object Model). Uusi sovelluksen versio kehitettiin Javalla verkko-ympäristöön, mutta varsinainen sovelluksen liiketoimintalogiikka saatiin valmiin COM-komponentin ansiosta lähes valmiina. Tällainen ratkaisu nopeutti verkossa toimivan sovelluksen toteuttamista dramaattisesti. Myöhemmin sovelluksesta tehtiin eri versioita eri tietokannoille, ympäristöille ja kohderyhmille.

2.5 Tuotelinjan haasteet

Tuotelinjan avulla saavutettavissa olevat liiketoiminnan edut ovat merkittäviä. Eräissä tapaustutkimuksissa [Linden *et al.*, 2007, 17] yritykset havaitsivat tuotelinjan käyttöönoton tuoneen merkittäviä hyötyjä: tuotteiden markkinoille saantia tapahtui 50% nopeammin, koodin määrä väheni jopa 70%, ylläpitotyö 20% ja virheiden määrää jopa 50%.

Tuotelinja-ajattelu vaatii kuitenkin aina muutoksia koko organisaation toimintatapoihin ja vaatii siten sekä organisaation ylimmän johdon että muiden tahojen yhteisen tahtotilan tuotelinjan käyttöönottamiseksi. Koska tuotelinjan

avulla saatavien etujen täysimääräinen hyödyntäminen on mahdollista vasta useamman tuotevariaation myötä, riskiä ei ehkä haluta ottaa. Pienessä yrityksessä käyttöönotto onnistuu ainakin päätöksenteon kannalta helpommin; ongelmaksi voivat kuitenkin muodostua aloituskustannukset, jotka ovat lähes aina hieman korkeammat kuin tilanteessa, jossa tuotelinja-ajattelua ei oteta huomioon. Aluksi syntyviä kustannuksia voi pyrkiä minimoimaan lähtemällä liikkeelle reaktiivisen tai inkrementaalisen mallin avulla. Isommissa yrityksissä ongelmaksi muodostunevat kulttuuriset syyt sekä kehitettyjen ohjelmistojen ja ohjelmistorunkojen historian painolasti.

Uuden tuotteen kokoaminen valmiista komponenteista on helpompaa siinä mielessä, että ainakaan aina organisaation toimintatapoihin ei tarvita mitään muutoksia. Kolmannen osapuolen komponenttien käyttöön liittyy nykyisin paljon ongelmia. Komponenttien lisenssiehdot ovat usein monimutkaisia ja useamman eri valmistajan komponenttien liittäminen yhdeksi järjestelmäksi voi osoittautua hyvin hankalaksi. Myöskään komponenttien laatu ei aina ole kohdallaan eikä takeita jatkokehityksestä yleensä anneta, mikä on ongelma erityisesti sellaisten komponenttien osalta, joiden lähdekoodi ei ole vapaasti saatavilla. Jos komponentti sisältää toimintalogiikan lisäksi myös käyttöliittymäelementtejä, voi ohjelmistokokonaisuudesta tulla hyvin epäyhtenäinen niin ulkoasun kuin käytettävyydenkin suhteen.

3 Tuotteenhallinta ohjelmistokehityksessä

3.1 Mitä on tuotteenhallinta

Yhdysvaltain puolustusministeriön tuotteenhallintaa käsittelevä teos [DoD, 2005, 1–3] määrittelee tuotteenhallinnan (engl. Configuration Management, CM) vapaasti suomennettuna seuraavasti:

> *Tuotteenhallinta on prosessi, joka määrittelee ja valvoo tuotteen tehokkuuden, toimintojen ja fyysisten attribuuttien yhtenevyyttä vaatimuksiin, suunnitteluun ja kerättyihin käyttötietoihin.*

Ohjelmiston tuotteenhallinnan (engl. Software Configuration Management, SCM) yksi määritelmä [Pressman, 2001, 225] on vapaasti suomennettuna:

> *Ohjelmiston tuotteenhallinta on joukko toimenpiteitä, joilla hallitaan muutoksia tunnistamalla tuotteet, jotka saattavat muuttua, muodostamalla yhteydet näiden tuotteiden välille, määrittämällä toimintatavat tuotteiden eri versioiden hallinnointiin, kontrolloimalla tapahtuvia muutoksia ja auditoimalla sekä raportoimalla tehdyistä muutoksista.*

Suomenkielisessä ohjelmistotuotantoa käsittelevässä kirjallisuudessa (esimerkiksi [Haikala & Märijärvi, 2000]) käytetään termiä tuotteenhallinta tarkoittamaan ohjelmiston tuotteenhallintaa. Myös tässä tutkielmassa tarkoitetaan tuotteenhallinnalla nimenomaan ohjelmiston tuotteenhallintaa.

3.2 Tuotteenhallinnan tarpeellisuus

Johdatuksena tuotteenhallinnan tarpeellisuuteen sopii hieman lyhennetty versio eräästä kirjallisuudessa [Wahli *et al.*, 2004, 4] kuvatusta tapauksesta.

Vuosia sitten tunnettu autojen valmistaja oli vaihtamassa suunnittelujärjestelmää eräässä toimipisteessään. Järjestelmää käytettiin työryhmissä tuotannon aikataulusuunnitteluun. Ohjelmisto laski aloitus- ja lopetusajat eri työvaiheille.

Siirtyminen uuteen järjestelmään tarkoitti vanhan järjestelmän sovittamista uuteen käyttöjärjestelmään, jolloin se jouduttiin kirjoittamaan uudelleen eri ohjelmointikielellä kuin alkuperäinen ohjelmisto. Uuden järjestelmän toteuttajaksi oli valittu IBM. Yhtenä vaatimuksena oli, että uusi ohjelmisto toimisi käyttöliittymien ja raporttien osalta kuten vanhakin. Tehtävä osoittautui varsin haasteelliseksi. Yhtenä syynä oli reilusti alakanttiin arvioidut työmääräarviot. Toinen on-

gelma liittyi siihen, että olemassa olevaan vanhan järjestelmän dokumentaatioon luotettiin liiaksi.

Puolentoista vuoden uurastuksen jälkeen uuden järjestelmän oletettiin olevan pieniä korjauksia lukuun ottamatta valmis. Toisin kuitenkin kävi. Projektissa oli käytetty menetelmänä vesiputousmallia, ja testausvaiheessa kävi ilmi, että jotain oli pahasti pielessä. Uusi järjestelmä ei toiminut samoin kuin vanha ja lisäksi kävi ilmi, että vanhan järjestelmän dokumentaatio ja toiminta eivät olleet yhteneviä.

Uuden järjestelmän toteuttaminen aloitettiin puhtaalta pöydältä. Vanhan järjestelmän toiminta selvitettiin perinpohjaisesti takaisinmallinnuksen avulla ja sen toiminnasta tuotettiin ajantasainen dokumentaatio. Järjestelmä toteutettiin uudelleen päivitetyn dokumentaation pohjalta. Loppujen lopuksi projekti saatiin onnistumaan ja asiakas oli tyytyväinen. IBM:ltä se vaati kuitenkin paljon rahaa, aikaa ja työtä.

Jos alkuperäisen järjestelmän yhteydessä olisi huolehdittu tuotteenhallinnasta, olisi vanhan järjestelmän dokumentaatio ollut ehkä ajan tasalla uutta järjestelmää tehtäessä. Vähintäänkin olisi nähty, että vanhaan järjestelmään oli tehty joitakin muutoksia dokumenttien päivittämisen jälkeen.

3.3 Tuotteenhallinnan osa-alueet

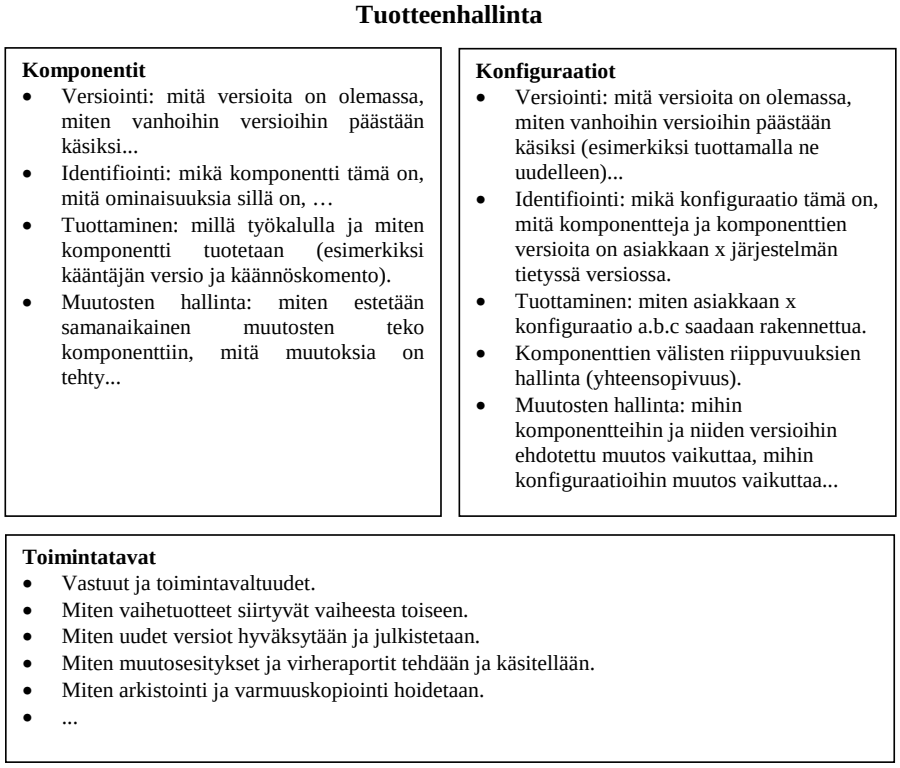
Karkealla tasolla tuotteenhallinta voidaan jakaa komponenttien, konfiguraatioiden ja toimintatapojen hallintaan. Kuva 3.1 havainnollistaa jaotteluun liittyvää vastuujakoa.

3.3.1 Komponentit ja konfiguraatiot

Tuotteenhallinnassa komponentilla tarkoitetaan jotakin hallittavaa asiaa kuten yksittäistä lähdekooditiedostoa tai dokumenttia. Tyypillisesti hallittavia komponentteja ovat muun muassa lähdekieliset ja käännetyt ohjelmamoduulit, vaatimusmäärittely, suunnitteludokumentti ja testaussuunnitelma. Johdetulla komponentilla tarkoitetaan yhdestä tai useammasta toisesta komponentista muodostettua komponenttia, esimerkiksi lähdekooditiedostosta käännettyä ajettavaa tiedostoa.

Konfiguraatioita koostetaan yhdistelemällä eri komponentteja ja niiden versioita laajemmiksi kokonaisuuksiksi (kuva 3.2).

Hallinta-alkio on pienin hallittava yksikkö ja se voi olla joko komponentti tai konfiguraatio. Sekä komponentit että konfiguraatiot versioidaan. Yksi konfiguraation versio käsittää rajoittamattoman määrän (0..*) komponentin versioita.



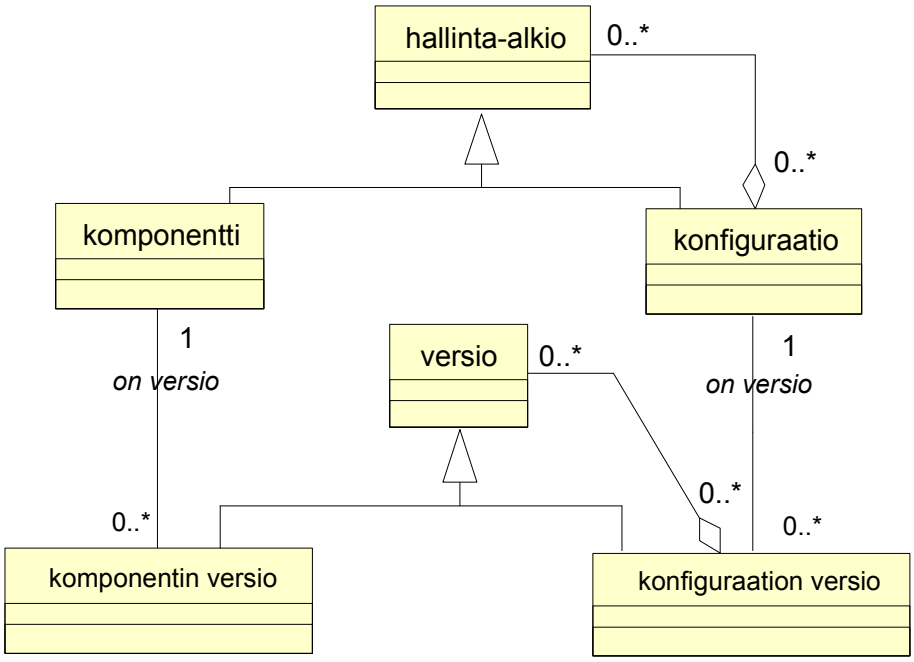
Kuva 3.1 Tuotteenhallinnan osa-alueet. [Haikala & Märijärvi, 2000, 237]

Konfiguraatiolla siis määritetään, mitkä komponentit ja niiden versiot kuuluvat mihinkin konfiguraation versioon.

Tuoterunkoon pohjautuvassa ohjelmistossa komponentteina ovat sekä tuoterungon että asiakasversioiden lähdekoodimuotoiset ja käännetyt ohjelmamoduulit. Tuoterunkoa ja asiakasversiota varten tarvitaan omat konfiguraationsa. Konfiguraatioita voidaan versioida eri tavoin. Yksi mahdollisuus on kiinnittää konfiguraatioiden versiot tuoterungon ja asiakasversioiden versionumeroihin.

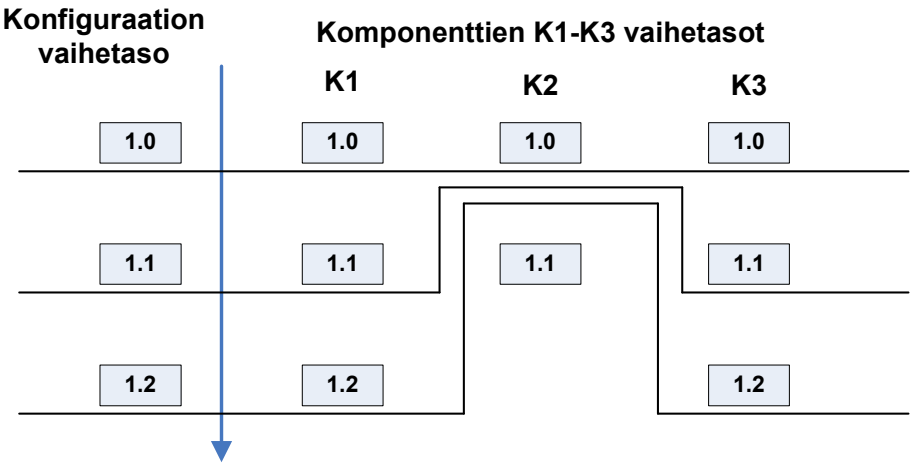
3.3.2 Versiointi

Jotta konfiguraation koostaminen olisi mahdollista, kaikki hallinta-alkiot, eli komponentit, konfiguraatiot ja työkalut on versioitava. Jokaisen hallinta-alkion versionumero voi olla aluksi esimerkiksi 1.0, ja kasvaa sitten lineaarisesti 1.1, 1.2 ja niin edelleen. Versionumerointi voidaan myös liittää tuotteessa käytettyyn versionumeroon ja versioida kolmannella tasolla (1.0.1, 1.0.2, ...) tuotejulkaisujen väliset versiot.



Kuva 3.2 Tuotteenhallinnan käsitteet [Haikala & Märijärvi, 2000, 241].

Kuvassa 3.3 on esimerkki konfiguraatioiden ja komponenttien versioinnista.



Kuva 3.3 Konfiguraatioiden ja komponenttien versiointi.

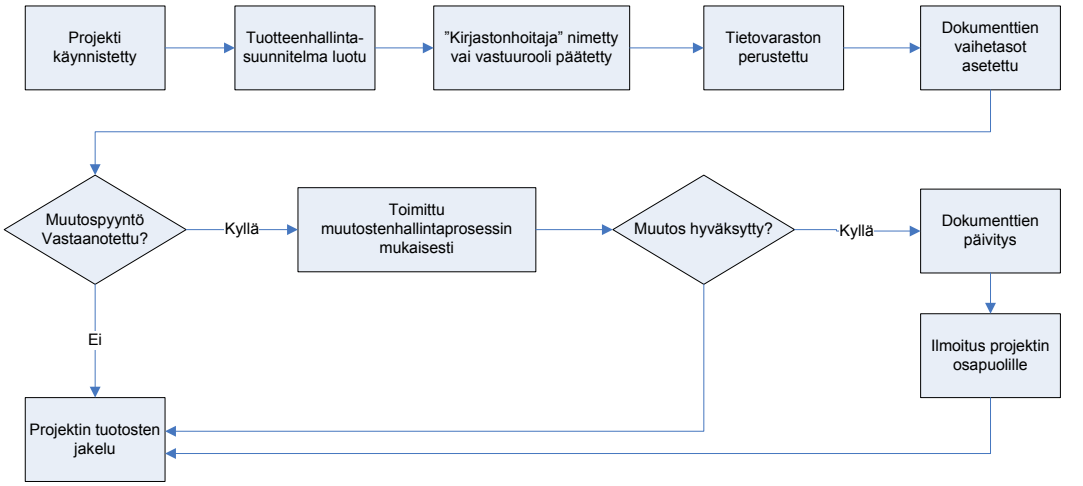
Kuvan 3.3 vasemmassa laidassa on kuvattu konfiguraatiot ja oikealla puolella kuhunkin konfiguraation vaihetasoon (versioon) liittyvät komponenttien K1–

K3 versiot. Konfiguraation vaihetasoon 1.0 kuuluu kaikkien kolmen komponentin versio 1.0. Konfiguraation vaihetasossa 1.1 päivittyvät komponentit K1 ja K3 versioon 1.1, mutta komponentin K2 versio ei muutu. Konfiguraatiossa 1.2 tapahtuu samalla tavalla: komponentit K1 ja K3 päivittyvät, mutta K2:n versio ei muutu.

Kuvasta 3.3 siis nähdään, että konfiguraation vaihetaso määrittelee täsmällisesti siihen kuuluvien komponenttien versiot. Jos halutaan määrittää sellainen konfiguraatio, jossa ovat mukana vain komponentit K1 ja K2, on luotava kokonaan uusi konfiguraatio. Pelkkä vaihetason päivittäminen ei riitä.

3.4 Tuotteenhallinnan prosessi

Luvun alussa perusteltiin tuotteenhallinnan tarpeellisuutta esimerkkitapauksen avulla. Tuotteenhallinnan tehtävänä on muun muassa mahdollistaa erilaisten kokonaisuuksien luonti käytettävissä olevista komponenteista ja tarjota pääsy vanhoihin komponenttien ja konfiguraatioiden versioihin. [Haikala & Märijärvi, 2000, 237–238] Kuva 3.4 esittää tuotteenhallinnan prosessin perusajatuksen.



Kuva 3.4 Tuotteenhallinnan prosessi. [Fantina, 2005, 118]

Projektin käynnistymisen jälkeen laaditaan tuotteenhallintasuunnitelma. Seuraavaksi valitaan henkilö tai rooli, jolla on vastuu projektissa mahdollisten muutospyyntöjen käsittelystä ja arkistoinnista. Projektin konfiguraatioiden ja komponenttien versiointia varten on perustettava tietovarasto ja kaikille projektin dokumenteille on määritettävä ensimmäinen vaihetaso, jotta mahdolliset muutokset voidaan havaita myöhemmin.

Jos projektissa ei tule muutospyyntöjä, voidaan projekti toimittaa muutosten hallinnan kannalta ilman lisätoimenpiteitä. Muutospyynnön tullessa noudatetaan projektin tuotteenhallintasuunnitelman mukaista menettelyä. Jos muutosta ei hyväksytä, se ei vaikuta projektin tuotoksiin. Mikäli muutos hyväksytään, päivitetään tarvittavia dokumentteja ja ilmoitetaan muutoksista kaikille projektin osapuolille. Lopuksi projektin tuottama materiaali toimitetaan sovitusti tilaajalle.

4 Tuotteenhallinnan työkalut

Tässä luvussa esitellään lyhyesti joitakin tuotteenhallinnassa käytettäviä työkaluja. Pääpaino on niiden työkalujen esittelyssä, joita käytetään tapaustutkimuksen tuotteiden hallinnassa (Subversion [Subversion, 2009], Atlassian JIRA [Atlassian, 2009e] ja Atlassian Bamboo [Atlassian, 2009a]).

4.1 Tuotteeseen keskittyvät työkalut

4.1.1 Versionhallintatyökalut

Tuotteenhallinnan kannalta on olennaista pystyä versiomaan konfiguraatiot ja komponentit luotettavasti. Subversion [Subversion, 2009] versionhallintaohjelmiston erään asiakassovelluksen, TortoiseSVN:n [TortoiseSVN, 2009], käyttöohje [Küng *et al.*, 2009] kuvailee versionhallintajärjestelmän toimintaa seuraavasti:

> *TortoiseSVN on ilmainen avoimeen lähdekoodiin perustuva asiakassovellus Subversion-versionhallintajärjestelmälle. Toisin ilmaistuna TortoiseSVN pitää kirjaa tiedostoihin ja hakemistoihin ajan mittaan tehdyistä muutoksista. Tiedostot talletetaan keskitettyyn arkistoon. Arkisto muistuttaa paljolti tiedostopalvelinta, mutta sen lisäksi se muistaa kaikki tiedostoihin ja hakemistoihin koskaan tehdyt muutokset. Tämän ansiosta voit palauttaa vanhoja versioita tiedostoistasi ja verrata sitä, miten ja milloin tiedot ovat muuttuneet, ja kenen toimesta. Tästä syystä Subversionia (ja versionhallintajärjestelmiä yleensä) voidaan pitää eräänlaisena "aikakoneena".*

Eri versionhallintaohjelmistoja on tarjolla niin ilmaisina kuin kaupallisinakin:

- Visual SourceSafe [Microsoft, 2009d] (kaupallinen)
- Perforce [Perforce Software, 2009] (kaupallinen)
- BitKeeper [BitMover, Inc., 2009] (kaupallinen)
- Subversion (SVN) [Subversion, 2009] (ilmainen, avoin lähdekoodi)
- Git [Git, 2009] (ilmainen, avoin lähdekoodi).

Versionhallintaohjelmistot eroavat toisistaan esimerkiksi sen suhteen, tallennetaanko koko muutoshistoria hajautetusti kaikille lähdekoodia käsitteleville tietokoneille vai säilytetäänkö muutoshistoriaa pelkästään keskitetyllä palvelimella. Kaupallisten versionhallintajärjestelmien edut ovat tuotekohtaisia. Esimerkiksi Visual SourceSafen etu on hyvä integroituvuus Microsoft Visual Studio tuoteperheen [Microsoft, 2009b] tuotteisiin.

4.1.2 Jatkuva integrointi

Jatkuva integrointi (engl. Continuous Integration, CI) on ohjelmistokehityksen käytäntö, jossa kehitysryhmä integroi (yhdistää) tuotoksensa säännöllisin väliajoin, yleensä ainakin kerran päivässä. Jokainen integraatio tarkastetaan (yleensä erillisen palvelimen tekemällä) automaattisella käännöksellä ja automaattisten testitapausten ajamisella, jolloin eri kehittäjien tekemät virheet havaitaan mahdollisimman nopeasti. [Fowler, 2009]

Integraatiopalvelin voi noutaa esimerkiksi kerran tunnissa versionhallintajärjestelmästä sovelluskoodien uusimmat versiot, yrittää kääntää niistä toimivan kokonaisuuden, ajaa mahdollisesti sovelluskoodiin liittyvät automaattiset testit, ja virheen sattuessa lähettää sähköpostiviestin niille henkilöille, jotka ovat tehneet muutoksia sovelluskoodiin sitten edellisen onnistuneen käännöksen.

Mainittu automaattisten testien käyttömahdollisuus riippuu paljon käytetyistä sovelluskehitysvälineistä. Pelkkä sovelluksen kääntyminen ei takaa sen toimivuutta. Automaattisen testauksen tarkoituksena on muun muassa [Gao *et al.*, 2003, 158]:

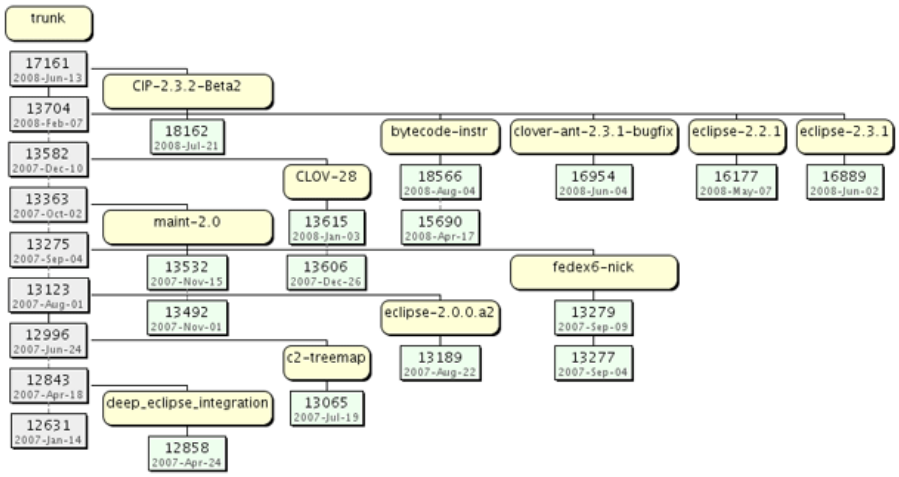
- Vapauttaa sovelluskehittäjä tylsästä ja toistuvasta manuaalisesta testaamisesta.
- Nopeuttaa sovelluksen elinkaarena tapahtuvaa testaamista, ja vähentää sovelluksen testaamisen kuluvaa aikaa ja kustannuksia.
- Parantaa ohjelmiston testausprosessin laatua ja tehokkuutta käyttämällä tarkoituksenmukaisia ennalta määriteltyjä testaussääntöjä.

Muiden työkalujen tapaan jatkuvaan integraatioonkin on tarjolla sekä kaupallisia että ilmaisia vaihtoehtoja, esimerkiksi:

- Atlassian Bamboo [Atlassian, 2009a] (kaupallinen)
- CruiseControl [CruiseControl, 2009] (ilmainen, avoin lähdekoodi).

4.1.3 Sovelluskoodimuutosten näkyvyyden parantaminen

Sovelluskoodimuutoksia voidaan seurata kaikkien versionhallintajärjestelmien muutoslokeista. Pelkästään niiden perusteella voi olla vaikeaa saada hyvää kokonaiskuvaa tehdyistä muutoksista. Markkinoilla on useita ilmaisia ja kaupallisia tuotteita, jotka havainnollistavat tehtyjä sovelluskoodimuutoksia graafisesti. Tällainen tuote on esimerkiksi Atlassian FishEye [Atlassian, 2009c] (kuva 4.1).



Kuva 4.1 Sovelluskoodimuutosten graafinen tarkastelu. [Atlassian, 2009c]

4.2 Tuotteenhallintaprosessin työkalut

Versionhallintatyökalujen ja tuotteenhallintatyökalujen ero on veteen piirretty viiva. Itse asiassa useita versionhallintatyökaluja markkinoidaan tuotteenhallintatyökaluina. Voidaan ajatella, että versionhallintatyökalut keskittyvät tuotteen hallintaan ja tuotteenhallintatyökalut tuotteenhallintaprosessin hallintaan. Tuotteenhallintatyökalut ovat erityisesti räätälöityjä lähdekoodipuiden hallintaan, ja niillä on monia ohjelmistonkehitykseen liittyviä erityisominaisuuksia, kuten tuki tietylle ohjelmointikielen syntaksille tai työkaluja ohjelmistojen rakentamiseen. [Küng *et al.*, 2009]

Esimerkki tuotteenhallintatyökalusta on IBM Rational ClearCase [IBM, 2009a], jonka yksi erityisominaisuus verrattuna vaikkapa Subversioniin on ohjelmistokäännöksen tekotavan (makefile, skripti, jne.) tallennus historiatietoihin.

Nokian omaisuudenhallintaan liittyvässä tapaustutkimuksessa [Linden *et al.*, 2007, 213] havaittiin, että selvityshetkellä markkinoilla ei ollut tuotteenhallintaan saatavilla kaupallisia tuotteita, jotka huomioisivat omaisuuden hallinnan. Saatavilla olevien tuotteenhallintatyökalujen toiminta oli keskittynyt suunnittelu- ja toteutusvaiheeseen, mutta omaisuuden hallinnan näkökulmaa ei oltu otettu riittävästi huomioon.

Versionhallintatyökalujen avulla on mahdollista rakentaa kunkin yrityksen toimintaan soveltuva tuotteenhallintatyökalu, kuten seuraavista alakohdista käy ilmi.

4.3 Tehtävänhallintatyökalut

Tehtävänhallintatyökalua voi ajatella yksinkertaistettuna muistilappuna, johon kirjataan asia, joka pitää tehdä ja tehtävän valmistuttua lappu arkistoidaan. Tehtävänhallintatyökalut ovat yleensä verkkoselaimella käytettäviä ohjelmistoja, joiden avulla voidaan hallita esimerkiksi projektin muutospyyntöjen ja sovellusvirheiden luontia, hyväksymistä, toteutusta ja seurantaa. Tehtävänhallintatyökalut voidaan mieltää sekä tuotteenhallintaprosessin työkaluiksi että tuotteeseen keskittyviksi työkaluiksi. Tehtävänhallintajärjestelmää voidaan käyttää tuotteenhallintaprosessin ohjaamiseen ja seuraamiseen, mutta yhtä hyvin sitä voidaan käyttää uusien tuotetta koskevien kehitysehdotusten kirjaamispaikkana.

Useimmat tehtävänhallintajärjestelmät mahdollistavat työnkulkujen (engl. workflow) konfiguroinnin. Työnkuluilla voidaan luoda erilaisia hyväksymismenettelyjä vaikka siten, että vain päätösvastuussa oleva henkilö voi osoittaa muunnospyynnön toteutettaviksi. Tai että sovellusvirheen korjaus voidaan merkitä valmiiksi vain, jos joku muu kuin korjaaja itse on sen testannut. Työnkulkujen konfiguroitavuuden avulla tehtävänhallinnan toiminta saadaan noudattamaan yrityksen käyttämiä prosesseja.

Monet tehtävänhallintajärjestelmät, kuten Atlassian JIRA [Atlassian, 2009e], soveltuvat myös tuotteenhallintaan, koska ne mahdollistavat julkaisujen (engl. release) hallinnan. Julkaisu voidaan mieltää tuotteenhallintamielessä konfiguraationa. Tiettyyn julkaisuun voidaan liittää halutut tehtävät, ja jo tehtävien luontivaiheessa voidaan määrittää niille aiottu julkaisuversio. Näin julkaisupäivämäärän lähestyessä voidaan tarkkailla vielä tekemättä olevien tehtävien luetteloa, mikä helpottaa projektipäällikön tehtävää. Muutosten hallintaan on tarjolla useita ilmaisia ja kaupallisia vaihtoehtoja, esimerkiksi:

- Atlassian JIRA [Atlassian, 2009e] (kaupallinen, ilmainen avoimen lähdekoodin projekteille)
- Bugzilla [Bugzilla, 2009] (ilmainen, avoin lähdekoodi).

Molemmat tehtävänhallintajärjestelmät integroituvat suoraan ainakin Subversion-versionhallintajärjestelmään. Tämä mahdollistaa esimerkiksi sen, että tehtävähistoriaa läpikäydessä näkee kunkin tehtävän kohdalta, mihin tiedostoihin ja niiden versioihinkin kyseinen tehtävä on vaikuttanut, ja mitä muutoksia tiedostoihin on tehty.

5 Tuotteenhallinnan menetelmät ja laadun valvonta

5.1 ISO 9001

ISO 9001 -laadunhallintajärjestelmä [2009] on toimialariippumaton toiminnan laadun sertifiointiin käytettävä standardi. Standardin käyttö on hyvin yleistä niin Suomessa kuin muuallakin. Vuonna 2007 ISO 9001:2000 -sertifikaatteja oli ISO:n tutkimuksen [ISO, 2008] mukaan käytössä 951 468, joista Suomessa 1804. ISO 9001 -standardi esitellään lyhyesti tässä tutkielmassa, koska se on käytössä kaikissa toiminnoissa yrityksessä, johon tutkielman tapaustutkimukset liittyvät.

ISO 9001 -laadunhallintajärjestelmän merkittävä tavoite on osoittaa organisaation kyky tuottaa vaatimustenmukaisia tuotteita, jotka täyttävät asiakkaan ja lainsäädännön asettamat vaatimukset. [Marttila, 2002, 1]

ISO 9001 -standardi ei tarjoa valmiita prosesseja yrityksen käyttöön vaan ajatuksena on että sertifiointiprosessin osana yrityksen käyttämät prosessit käydään läpi. Tarvittaessa niitä muutetaan siten, että niissä otetaan huomioon poikkeamien käsittely ja ehkäisy. ISO 9001 -standardi soveltuu sekä pieniin parin henkilön yrityksiin että suuriin organisaatioihin. Standardin käyttöönotto on myös joustavaa, koska sertifiointi voidaan määrittää koskemaan vain osaa yrityksen prosesseista, esimerkiksi testausta. Myöhemmin voidaan hankkia sertifiointi esimerkiksi toteutuksessa käytetyille prosesseille.

ISO 9001 -standardin hyödyntämisen perusperiaate on, että ensimmäisessä vaiheessa kuvataan yrityksen käyttämät prosessit ja huolehditaan, että prosesseja suorittavilla henkilöillä on kirjallisen, suullisen tai muun ohjeistuksen välityksellä riittävät tiedot ja taidot prosessien noudattamiseksi. Kun prosessit on saatu kuvattua ja käyttöön, on ISO 9001 -standardin vaatimusten mukaisesti kaikkia sertifioinnin piirissä olevia prosesseja sekä valvottava että mitattava. Muun valvonnan ohella tämä tapahtuu joka tapauksessa standardissa kuvattujen sisäisten ja ulkoisten auditointien avulla. Auditoinneissa käydään läpi prosesseja sekä niiden tuottamia dokumentteja tai muita tuloksia. Pistokoemaisesti voidaan valita vaikka jokin prosessin vaihe, josta on määrätty aina syntyväksi tietty dokumentti ja varmistetaan, että sellainen todella löytyy. Jos toiminnasta löytyy puutteita, niistä kirjataan poikkeamaraportit. Hieman kärjistettynä voidaan sanoa että auditointien tai koko ISO 9001 -standardin tarkoituksena ei ole suoranaisesti varmistaa, että yrityksen tuottamat palvelut tai tuotteet olisivat laadukkaita. Standardin tarkoituksena on ennen muuta varmistaa, että yritys toimii säännön-

mukaisesti itse itselleen asettamien ohjeiden mukaisesti.

ISO 9001 -stardardi soveltuu toimialariippumattomuutensa johdosta hyvin käytettäväksi myös ohjelmistoteollisuudessa. Se soveltuu myös pieneen yritykseen, koska sen käyttöönottoon liittyvä sertifiointiprosessi ei ole kovin raskas.

Tuotteenhallinnan ohjeistaminen on yksi ISO 9001 -standardin edellytyksistä. Ongelmana on kuitenkin se, ettei se ota mitään kantaa tuotteenhallinnan toteutustapaan, vaan lähes mikä hyvänsä muodollinen menettelytapa kelpaa. Siten hyvin alkeellinenkin mekanismi on sertifiointikelpoinen. [Haikala & Märijärvi, 2000, 239]

Tuotteenhallinnan kannalta ISO 9001 -standardi on joka tapauksessa yksi avaintyökalu silloin, kun standardia käytetään muutenkin tai sen käyttöönottoa harkitaan. Koska standardi tarjoaa hyvät työkalut prosessien valvomiseen ja mittaamiseen, soveltuu se hyvin tuotteenhallinnan tehostamiseen ja kehittämiseen. Lähtökohtana on oltava yrityksen selkeä tavoite kehittää tuotteenhallintaa ISO 9001 –standardia hyödyntäen, koska standardi vaatii lähinnä muodollisten menettelyiden olemassaoloa. On siis hyvä tiedostaa, että kahdessa ISO 9001 -sertifioidussa yrityksessä saatetaan hoitaa asioita hyvin eri tavalla, ja loppujen lopuksi hyvin erilaisten laatuvaatimusten mukaisesti.

5.2 CMMI

Ohjelmistokehityksen tueksi on saatavilla paljon erilaista materiaalia liittyen muun muassa standardeihin, metodologioihin sekä erilaisia ohjeistuksia yrityksen liiketoiminnan kehittämiseksi. Monet kehittämisen lähestymistavoista ovat kuitenkin enemmän tai vähemmän puutteellisia ja keskittyvät vain joihinkin liiketoiminnan osa-alueisiin. CMMI-tuotetiimin [CMMI Product Team, 2006, 3] mukaan tämänkaltainen toiminta on vain pahentanut ongelmia, koska tiettyihin osa-alueisiin keskittyminen on usein vienyt huomion pois tyypillisistä useimpien organisaatioiden todellisista ongelmista.

Vaikka CMMI-kypsyysmalli ja ISO 9001 -standardi koskevat useita samoja asioita, on niillä ainakin yksi merkittävä ero. ISO 9001 -standardi asettaa vähimmäisvaatimukset, joiden mukaisesti yrityksen tulee toimia. CMMI-kypsyysmalli keskittyy prosessien kehittämiseen ja auttaa löytämään yrityksen toiminnasta tärkeimpiä kehityskohteita. Joka tapauksessa yrityksen tulisi keskittyä toiminnan kehittämiseen, ei arvosanan tavoitteluun, oli kyse sitten kypsyystasosta tai sertifikaatista. [Paulk, 1994, 23]

CMMI-kypsyysmalli (engl. Capability Maturity Model Integration, CMMI) tarjoaa työkaluja kokonaisvaltaisempaan toimintatapaan. CMMI-kypsyysmallin tuotteisiin ja palveluihin keskittyvä ohjeistus (CMMI for development, [CMMI Product Team, 2006]) kuvaa toimivia käytäntöjä koko tuotteen elinkaaren hallintaan alkaen hahmottelusta ja päätyen aina toimituksiin ja ylläpitoon. CMMI-kypsyysmalli koostuu 22 prosessista, jotka liittyvät muun muassa riskien ja vaatimusten hallintaan, tuoteintegraatioon ja laadun valvontaan. Lisäksi kypsyysmalli sisältää viisi eri tasoa, jotka kuvaavat organisaation edistymistä (kypsyyttä) prosessien käyttämisessä. Tasoilla 1-4 tavoitellaan selkeitä parannuksia prosesseihin ja taso 5 keskittyy prosessien optimointiin. CMMI-kypsyysmalli on esitelty perusteellisesti CMMI-tuotetiimin ohjeistuksessa [CMMI Product Team, 2006].

CMMI-kypsyysmallin käyttäminen tapahtuu siten, että organisaation toiminnasta valitaan kerrallaan yksi kehitettävä prosessi, jonka arvellaan toimivan tehottomasti tai muutoin huonosti. Valitun prosessin tavoitteita verrataan yrityksen noudattamaan nykytilaan esimerkiksi tarkastuslistan avulla. Kuhunkin havaittuun epäkohtaan mietitään korjaavia toimenpiteitä ja toteutetaan ne. Lopuksi tarkastetaan tilanne uudelleen ja siirrytään välillä jonkin toisen prosessin kehittämiseen. Useampaakin prosessia on toki mahdollista kehittää rinnakkaisesti. On kuitenkin selvää, että 22 prosessin läpikäynti ja tehostaminen eivät onnistu yhdessä yössä.

Tässä tutkielmassa keskitytään CMMI-kypsyysmallin tuotteenhallinta-osuuteen. Tuotteenhallintaprosessi on yksilöity CMMI-kypsyysmallissa kypsyystason 2 tukiprosessiksi. CMMI-kypsyysmallin tuotteenhallintaprosessin [CMMI Product Team, 2006, 116] tavoitteet (Specific Goals) ja toimenpiteet (Specific Practices) ovat seuraavat:

- SG 1 Vaihetasojen muodostus (Establish Baselines)
  - SP 1.1 Hallittavien komponenttien tunnistaminen (Identify Configuration Items)
  - SP 1.2 Tuotteenhallintajärjestelmän luonti (Establish a Configuration Management System)
  - SP 1.3 Vaihetasojen luonti tai vapauttaminen (Create or Release Baselines)

- SG 2 Muutosten seuranta ja kontrollointi (Track and Control Changes)
  - SP 2.1 Muutospyyntöjen seuranta (Track Change Requests)
  - SP 2.2 Komponenttien hallinta (Control Configuration Items)
- SG 3 Eheyden varmistus (Establish Integrity)
  - SP 3.1 Konfiguraationhallinnan jäljitettävyys (Establish Configuration Management Records)
  - SP 3.2 Tuotteenhallinnan tarkastukset (Perform Configuration Audits).

CMMI-kypsyysmallin kuvaaman tuotteenhallintaprosessin ensimmäisenä vaiheena on tunnistaa ne komponentit joita tuotteenhallinta koskee. Kuten tuotteenhallintaa yleisellä tasolla esittelevässä luvussa 3 kerrottiin, tyypillisiä komponentteja ovat projektien dokumentit ja sovelluskoodi. Tunnistetuille komponenteille on asetettava vaihetasot (versiot), jotta niihin kohdistuvia muutoksia voidaan seurata. Kun tuotteenhallinnan alaiset komponentit on tunnistettu, tuotteenhallinnan vastuulla on kontrolloida niihin kohdistuvia muutoksia.

Kypsyysmallin tuotteenhallintaprosessi edellyttää vaihetasojen eheyden ylläpitämistä. Tämä ilmenee siten, että kaikki muutokset, jotka tehdään tuotteenhallinnan alaisiin komponentteihin kohdistetaan tarkasti johonkin vaihetasoon. Kun tiedetään eri komponenttien ja vaihetasojen väliset riippuvuudet, osataan tietyn vaihetason muuttuessa myös päivittää muita tarvittavia komponentteja ja niiden vaihetasoja.

CMMI-kypsyysmallin keskeinen ajatus on ettei minkään prosessin osalta voida päästä täydellisyyteen vaan aina riittää parantamisen varaa. CMMI-kypsyysmallin käyttöönoton alkuvaiheessa epäkohtia löytyy enemmän. Taulukossa 5.1 on listattu joitakin yleisiä tuotteenhallinnan ongelmia, CMMI-kypsyysmallin vaatimia tavoitteita ja korjausehdotuksia [Fantina, 2005, 111] tarvittavista muutoksista. Esimerkit antavat ymmärrystä siitä minkä tyyppisiä ongelmakohtia CMMI-kypsyysmalli pyrkii kehittämään tuotteenhallintaan liittyen. Tutkielman tapaustutkimusosuutta käsittelevissä luvuissa 6 ja 7 kartoitetaan tapausten tuotteenhallinnan tasoa hieman samaan tapaan tarkastuslistojen avulla.

Nykyinen käytäntö	CMM/CMMI	Tarvittava muutos
Tuotteenhallinta tehdään epävirallisesti.	Tuotteenhallintasuunnitelma tehdään jokaiselle projektille.	Valmistellaan tuotteenhallintasuunnitelman pohja.
Projektipäälliköt hallinnoivat työn alla olevia tuotteita eivätkä ne ole jaetulla levyllä.	Tuotteenhallintakirjasto on perustettu versionhallintaan tuotteenhallinnan alaisille tuotteille.	Viedään omilla koneilla olevat työt versionhallintaan ja annetaan kaikille tiimin jäsenille lukuoikeudet.
Tuotteet ovat versionhallinnassa tarpeen mukaan.	Tuotteenhallinnan alaiset tuotteet tunnistetaan projektien käynnistämisvaiheessa ja laitetaan alusta asti versionhallintaan.	Perustetaan erityiset raportointietapit, lisäksi mahdollinen raportointi poikkeustilanteiden varalle.
Kuka tahansa tiimin jäsen voi tehdä muutospyyntöjä tarpeen mukaan.	Käytössä on säännönmukainen toimintamalli vaihetason (engl. baseline) muutospyyntöjen käsittelylle, dokumentoinnille ja seurannalle.	Luodaan ja dokumentoidaan muutostenpyyntöprosessi.
Muutoksia tehdään tarpeen mukaan.	Muutoksia tehdään vaihetason päälle ainoastaan asianmukaisen ja huolellisesti noudatetun muutospyyntöprosessin kautta ja kaikkia tarpeellisia projektin osapuolia informoidaan muutoksesta.	Perustetaan käytäntö muutosten tekemiseen ja sisällytetään siihen muutospyyntöjen käsittelyprosessi.

Taulukko 5.1 Tuotteenhallintaprosessien kehittäminen CMMI-kypsyysmallissa. [Fantina, 2005, 111]

Ensimmäisessä kohdassa ongelmana on tuotteenhallinnan määrämuotoisuuden puute. CMMI edellyttää tuotteenhallintasuunnitelman tekemistä jokaiselle projektille. Muutokseksi ehdotetaan tuotteenhallintasuunnitelman pohjan laadintaa. Yhtä tärkeää on linjata käytettävät työkalut ja toimintamallit sekä valvoa, että sovittuja toimintatapoja noudatetaan.

Toisessa kohdassa tuotteenhallintaan liittyvät tiedot sijaitsevat projektipäälliköiden työasemissa. Ratkaisuna on tuotteenhallintaan liittyvien tietojen vienti yhteiseen versionhallintaan. Ratkaisun avaimena on sopivien työkalujen olemassaolo. Yrityksen on valittava ja otettava käyttöön tuotteenhallintaan tarvittavat työkalut ja edellytettävä niiden käyttämistä kaikissa projekteissa.

Kolmannessa kohdassa vain osa tuotteista on tuotteenhallinnan piirissä. CMMI vaatii tuotteenhallinnan alaisten tuotteiden tunnistamista alusta asti. Ratkaisuksi ehdotetaan raportoinnin kehittämistä. Yksi vaihtoehto voisi olla kaikkien projektien vienti tuotteenhallinnan piiriin riippumatta siitä, onko niiden tuotteenhallinnalle tarvetta.

Neljännessä kohdassa ongelmana on muutospyyntöjen epämääräinen käsittely. CMMI vaatii määrämuotoista muutospyyntöjen käsittelyprosessia, jota ehdotetaan myös ratkaisuksi. Muutosten hallinnan apuvälineenä voidaan hyödyntää tehtävänhallintajärjestelmää.

Viimeisessä kohdassa muutoksia tehdään liian vapaasti. CMMI edellyttää muutoksiin liittyvää päätöksentekoprosessia. On tiedettävä, kuka on vastuussa muutoksen hyväksymisestä. Myös kaikkia projektin osapuolia on informoitava muutoksesta.

6 Toiminnanohjausjärjestelmä

6.1 Esittely

Ensimmäinen tapaustutkimuksen kohde on teollisuuden, tukkukaupan ja taloushallinnon tarpeisiin suunnattu toiminnanohjausjärjestelmä (engl. Enterprise Resource Planning, ERP) [Koch & Wailgum, 2008]. Tässä tutkielmassa tarkoitetaan toiminnanohjausjärjestelmällä nimenomaisesti tätä tapausta, ei toiminnanohjausjärjestelmiä yleisesti. Järjestelmää on kehitetty aktiivisesti yli kymmenen vuotta. Asiakkaina on useita suuria ja keskisuuria suomalaisia yrityksiä erityisesti tukkukaupan ja teollisuuden toimialoilta. Toiminnanohjausjärjestelmä sisältää työkalut yrityksen sisäisten prosessien hallintaan ja se mukautuu erikokoisiin yrityksiin ja liiketoiminnan tarpeisiin. Käytetty ohjelmistoarkkitehtuuri ei ole tiukasti sidottu mihinkään käyttöliittymään eli käyttö onnistuu normaalin työaseman ohella esimerkiksi erilaisilla käsipääteratkaisuilla. Tuotteen tarkoituksena on vähentää manuaalisen työn määrää, vähentää virhetilanteiden mahdollisuutta ja vakiinnuttaa toimintatapoja. Toiminnanohjausjärjestelmä sisältää muun muassa seuraavia komponentteja:

- materiaalivirtojen hallinta
- tuotehallinta
- myyntitoiminnot
- ostotoiminnot
- varaston hallinta
- huoltotoiminta ja kunnossapito
- asiakashallinta
- taloushallinto.

Toiminnanohjausjärjestelmässä on myös kattavat integrointimahdollisuudet muihin järjestelmiin. Tuettuja standardeja ovat muun muassa Finvoice [Finanssialan Keskusliitto, 2009], EDI [UNECE, 2009] ja XML [W3C, 2009].

6.2 Tutkimuksen taustaa

Toiminnanohjausjärjestelmää kehittää tuotekehitystiimi (kuusi henkilöä). Lisäksi asiakasprojektitiimejä on neljä ja niissä työskentelee vakituisesti noin 40-50 henkilöä. Tuotteeseen liittyvien menetelmien ja prosessien kehittäminen on projektitiimin vastuulla. Myös asiakasprojektitiimeissä tehdään aktiivisesti tuotekehi-

tystä vaikka pääpaino onkin asiakaskohtaisissa toteutuksissa. Tässä luvussa toiminnanohjausjärjestelmällä tarkoitetaan siihen liittyvää tuoterunkoa, asiakkaille toimitettavista tuotteista käytetään termiä asiakasversio.

Toiminnanohjausjärjestelmää kehitetään Progress-sovelluskehittimellä [Progress Software Corporation, 2009]. Tuotteenhallinta tehdään JIRA-tehtävänhallintajärjestelmällä. [Atlassian, 2009e] Tehtävien priorisointiin käytetään JIRA:n lisäosaa, GreenHopperia [Atlassian, 2009d]. Lisäosa mahdollistaa helpon tehtävien priorisoinnin hiirellä raahaamalla. Versionhallintajärjestelmänä on Subversion [Subversion, 2009]. Näiden ohella hyödynnetään myös Confluence-tietämyksenhallintajärjestelmää [Atlassian, 2009b].

Yritys on laatinut tuotekehitystä varten prosessit uusien ominaisuuksien lisäämiseen ja muutosten hallintaan. Näiden prosessien ohella tutkimusmateriaali koostuu toiminnanohjausjärjestelmän tuotepäällikön haastattelusta [Henkilö A, 2009] sekä tehtävänhallintajärjestelmästä kerätyistä tiedoista. Haastattelumenetelmänä käytettiin teemahaastattelua [Hirsjärvi & Hurme, 1982], haastattelukysymykset löytyvät liitteestä 1.

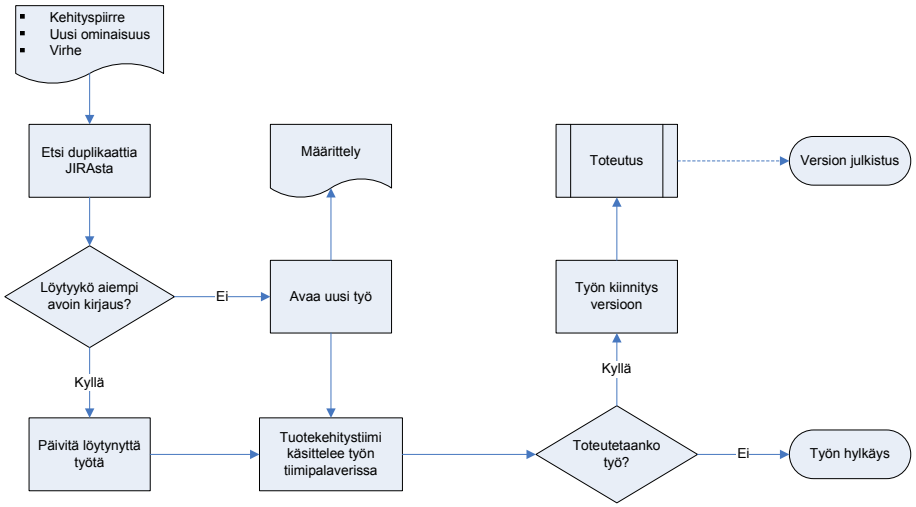
6.3 Prosessit ja toimintatavat

6.3.1 Tuotekehitysprosessi

Toiminnanohjausjärjestelmän tuotekehitysprosessin (kuva 6.1) liikkeellepanijana toimii jokin seuraavista vaihtoehdoista:

- Tuoterunkoon on sovittu tehtäväksi uusi piirre, joka on tyypillisesti lisäys johonkin olemassa olevaan ominaisuuteen. Uuden piirteen kehitystarvetta on voitu käsitellä aiemmin tuotekehitystiimin tiimipalaverissa.
- Tuoterunkoon on sovittu tehtäväksi uusi ominaisuus tai piirre. Tarve on voinut tulla joltakin asiakkaalta tai se on tunnistettu yrityksen sisällä. Tyypillisesti asiakkaalta tulleet kehityspyynnöt käsitellään projektilähtöisen tuotekehityksen prosessilla (alakohta 6.3.2).
- Tuoterungossa on havaittu virhe, joka on korjattava tuoterunkoon, ei asiakasversioihin.

Aluksi tarkastetaan tehtävänhallintajärjestelmästä, löytyykö tuotekehitystarpeeseen liittyen aiempia samaa asiaa koskevia tapahtumia (”Etsi duplikaattia JIRAsta”). Jos aiempia tapahtumia ei löydy, kirjataan kehitystarpeesta uusi tapahtuma. Muussa tapauksessa olemassa olevaa tapahtumaa tai tapahtumia päivite-



Kuva 6.1 Toiminnanohjausjärjestelmä, tuotekehitysprosessi.

tään vastaamaan uutta kehitystarvetta. Jos kehitystarpeesta luodaan uusi tapahtuma tehtävänhallintaan, on se myös määriteltävä. Määrittelyjen taso ja laajuus vaihtelevat suuresti. [Henkilö A, 2009] Toisinaan määrittely on tehty liiankin tarkasti, toisinaan se on aivan liian puutteellista.

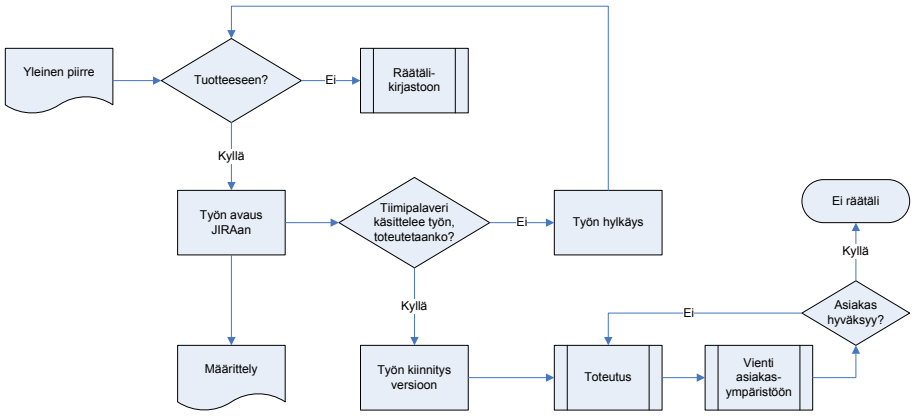
Kun uusi kehitystarve on kirjattu tehtävänhallintaan tai päivitetty olemassa olevaa kehitystarvetta, tuotekehitystiimi käsittelee vielä toteuttavaksi aiotun työn palaverissa. Palaverissa toteutettavaksi aiottua tuotekehitystyötä tarkastelee useampi henkilö, jolloin mahdolliset määrittelyssä olevat puutteet tulevat todennäköisemmin esille. Tuotekehitystiimi päättää työn toteuttamisesta tai hylkäämisestä. Jos työ hylätään, tehdään siitä merkintä tehtävänhallinnan kirjaukseen. Tarvittaessa työ voidaan avata uudelleen myöhemmin. Jos työ päätetään tehdä, päätetään käytettävissä olevien resurssien ja tiedossa olevien versioiden julkaisuaikataulujen pohjalta versio, johon kehitystarve toteutetaan. Kohdeversio merkitään työn tapahtumakirjaukseen ja tapahtumalle osoitetaan tekijä. Samassa yhteydessä tarkastetaan tapahtumaan liitetyn työmääräarvion ajantasaisuus.

Myönteisen toteuttamispäätöksen jälkeen tehtävä toteutetaan. Toteuttaja merkitsee toteutuksen aloittaessaan tapahtuman aloitetuksi. Kun toteutus valmistuu, se siirtyy testattavaksi ja sisäiseen katselmointiin. Katselmointitilaisuuden päätarkoituksena on tunnistaa tuotosten eroavuudet suunnitelmiin nähden [IEEE, 1988].

Pääsääntönä on, että kaikki valmiit toteutukseen liittyvät hyväksytyt tapahtumat on testannut joku toinen henkilö kuin kehittäjä itse, ja ne on katselmoitu sisäisesti. Prosessin viimeisessä vaiheessa julkistetaan uusi versio tuotteesta.

6.3.2 Projektilähtöinen tuotekehitys

Projektilähtöinen tuotekehitys (kuva 6.2) on tilanne, jossa asiakasprojektissa ilmenee tarve kehittää uusi piirre, jonka arvellaan soveltuvan osaksi tuoterunkoa.



Kuva 6.2 Toiminnanohjausjärjestelmä, projektilähtöinen tuotekehitys.

Prosessin alussa ratkaistaan uuden piirteen sijoituspaikka. Tuleeko se tuoterunkoon vai asiakasversioon? Jos ominaisuutta ei oteta tuoterunkoon, sijoitetaan uusi toiminto räätälikirjastoon. Räätälikirjasto tarkoittaa kokoelmaa toimintoja, joita voidaan hyödyntää asiakasprojekteissa, mutta jotka eivät ole tuoterungon osa. Räätälikirjaston toiminnot voivat olla sopivia muutamalle asiakkaalle, mutta tuoterunkoa ajatellen ne eivät ole riittävän yleiskäyttöisiä.

Jos uusi toiminto toteutetaan tuotteeseen, on prosessi varsin samankaltainen kuin tuotekehitysprosessi (alakohta 6.3.1). Projektilähtöisen tuotekehitysprosessin yksi ero on siinä, että uuden piirteen tuotteeseen sijoittamista mietitään kahteen kertaan. Ensimmäisen kerran asiaa käsitellään aivan prosessin alkuvaiheessa, kun päätetään piirteen sijoituspaikkaa (tuote vai räätälikirjasto). Myöhemmin asia tulee uudelleen esille tuotekehitystiimin tiimipalaverissa samaan tapaan kuin tuotekehitysprosessissakin. Projektilähtöisen tuotekehityksen prosessin kautta uusien toimintojen ei pitäisi päätyä liian helposti tuotteeseen.

Toinen ero projektilähtöisen tuotekehityksen ja tuotteeseen liittyvän tuotekehityksen välillä on prosessin loppupäästä. Projektilähtöisessä tuotekehityksessä

uusi toiminto on toimitettava asiakkaalle testattavaksi ja lopuksi tuotantoon.

6.3.3 Konfiguraatioiden ja komponenttien versiointi

Kaikki tuoterunkoon ja asiakasprojekteihin liittyvä materiaali säilytetään yrityksen versionhallintajärjestelmässä lukuun ottamatta joitakin tietokannan parametrointidatoja, joiden varastointi keskitetysti ei ole mielekästä. Versionhallinnan käyttämiseen ei ole liittynyt suuria ongelmia, mutta joskus yksittäinen kehittäjä unohtaa viedä tiedostoja versionhallintaan. Koska kaikki tapahtumat katselmoidaan sisäisesti ennen hyväksyntää, nämä tilanteet on saatu kiinni ennen asiakastoimituksia.

Tuotehallinnan konfiguraatiot ja tuoterungon sekä asiakasversioiden versionumerot ovat käytännössä sama asia, sillä erillistä konfiguraatioiden hallintaa ei tehdä. Versioinnissa oli aiemmin ongelmia asiakastoimituksiin liittyen. Julkaistuihin versioihin vietiin asiakastoimitusten jälkeen muun muassa bugikorjauksia, minkä seurauksena vanhoja versioita ei pystytty jälkikäteen muodostamaan luotettavasti. Tuotepäällikkö uudisti vuoden 2009 alussa tuotteen versiointiin liittyviä käytäntöjä siten, että kaikki asiakastoimituksia seuraavat bugikorjaukset sijoitetaan uuteen aliversioon (tai aliversion aliversioon), eikä jo julkaistuihin versioihin tehdä jälkikäteen enää mitään muutoksia. Nykyisin versioinnissa on käytössä neljä tasoa (taulukko 6.1):

	Pääversio	Aliversio	Aliversion aliversio	Kriittiset korjaukset aliversion aliversioon
Esimerkki	5	4	1	1

Taulukko 6.1 Toiminnanohjausjärjestelmän versiointikäytäntö.

Pääversio, aliversio ja aliversion aliversio ovat aina käytössä. Neljännen tason versiointia tehdään vain tarpeen mukaan, jos edelliseen aliversioon ilmenee tarve tehdä kriittisiä korjauksia.

6.4 Projektidata

Tässä alakohdassa tarkastellaan tuotteenhallinnan prosessien tuottamaa projektidataa. Projektidata toimii taustatietona jäljempänä luvussa esiteltävälle havaintotaulukolle, ja antaa myös viitteitä tuotteenhallinnan prosessien toimivuudes-

ta. Tämän tapauksen osalta analysoitava projektidata on rajattu tuoterungon projektidataan, koska asiakasprojekteihin liittyvät tehtävänhallinnan tiedot eivät olleet käytettävissä. Projektidatan tarkasteluajanjakso on pääasiassa 1.1.2007–30.6.2009. Alkuajaksi on valittu vuoden 2007 alku, koska tapauksen tehtävähallintaan alettiin tuolloin käyttämään täysipainoisesti JIRA-tehtävänhallintajärjestelmää. Loppuajankohdaksi valittiin tutkimuksen toteutushetkeä lähin kokonainen vuosineljännes. Taulukossa 6.2 esitetään toiminnanohjausjärjestelmän tuoterungon tapahtumien jakaantuminen eri tuotteenosien välille.

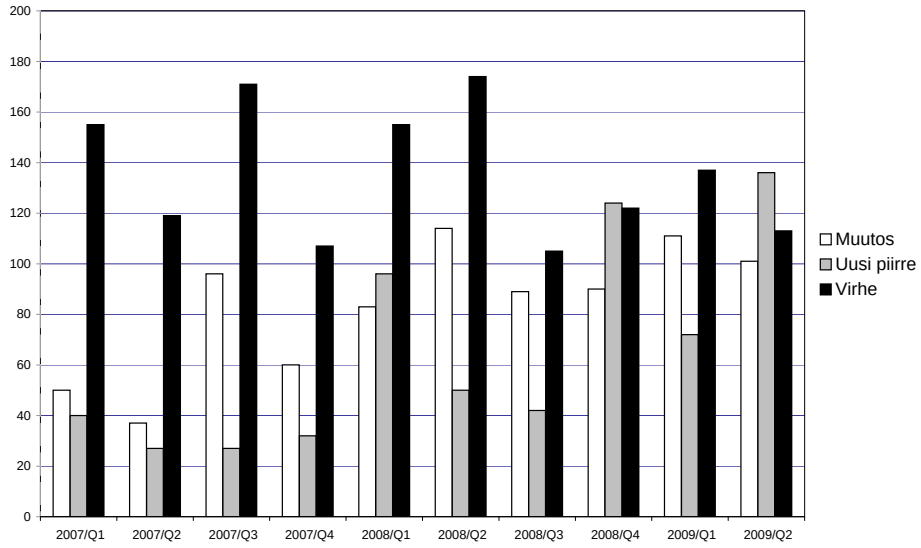
Komponentti	Tapahtumia	Osuus
Yleiset ohjelmat	1652	22 %
Myynti	1020	13 %
Käyttöliittymä, tuotteen ydin ja kehitystyökalut	917	12 %
Varasto	476	6 %
Osto	474	6 %
Tuotanto	392	5 %
Kirjanpito	245	3 %
Tuotehallinta	242	3 %
Perusdokumentaatio	226	3 %
Palkanlaskenta	217	2 %
Ei määritetty	202	2 %
Myyntireskontra	154	2 %
Huolto	152	2 %
Muut	1042	19 %

Taulukko 6.2 Tapahtumakirjausten jakautuminen tuotteen eri osiin.

Tapahtumien jakautuminen osoittaa, että tuoterungon kehittämisen keskittyy yleisten tuotteeseen liittyvien asioiden (yleiset ohjelmat, käyttöliittymä, tuotteen ydin ja kehitystyökalut) lisäksi myyntitoiminnallisuuteen. Yleisiin ohjelmiin liittyviä asioita ovat muun muassa tulostuksen ja liitetiedostojen käsittely, kuten kaksi esimerkkitapauskuvausta osoittaa:

- ”Excel tulostus -kirjoitin ja Excel Pivot taulu -toiminnoista virheilmoituksia”
- ”Liite-dokumentin lataus tietokantaan ei hallitse oikein kolmannen osapuolen tiedoston lukulukkoa”

Seuraavaksi tarkastellaan tapahtumien jakautumista tapahtumalajeittain (uusi piirre, muutos tai virhe). Kuva 6.3 esittää uusien, tarkastelujaksona avattujen tapahtumien, jakaantumista eri tapahtumalajeihin vuosineljänneksittäin ajanjaksolla 1.1.2007–30.6.2009.



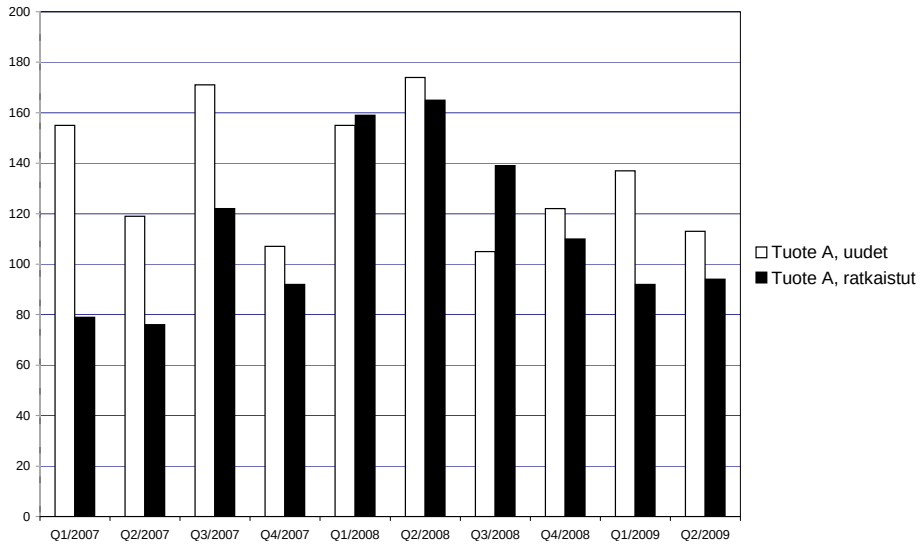
Kuva 6.3 Toiminnanohjausjärjestelmä, luotujen tapahtumien määrä tapahtumalajeittain aikavälillä 1.1.2007–30.6.2009.

Kuvasta voidaan tehdä mielenkiintoinen havainto: vuoden 2008 puoliväliin asti tapahtumalajeista virheiden osuus on ollut merkittävästi suurempi verrattuna uusiin ominaisuuksiin ja muutoksiin. Tämän jälkeen tilanne näyttäisi hieman tasoittuneen. Vuoden 2008 lopulla ja vuoden 2009 toisen neljänneksen osalta uusia ominaisuuksia koskevat tapahtumat jo ohittavat virhetapahtumien määrän. Tämä viestii virheiden korjaamisen tarpeen vähenemisestä ja tekemisen painopisteen muuttumisesta uusien piirteiden toteuttamiseen sekä vanhojen kehittämiseen.

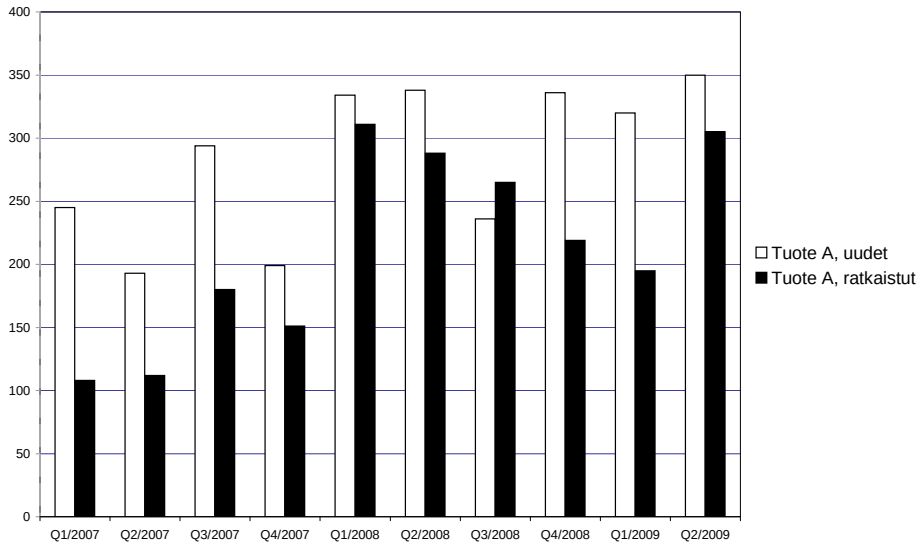
Kuvasta 6.3 ei kuitenkaan käy ilmi, kuinka paljon virheitä on pystytty korjaamaan suhteessa uusiin virhetapahtumakirjauksiin. Sitä asiaa vertailee kuva 6.4, josta käy ilmi luotujen ja ratkaistujen virhetapahtumien määrien suhde eri tarkasteluajanjaksoina.

Pääsääntöisesti virhetapahtumia on luotu enemmän kuin niitä on saatu ratkaistuksi. Poikkeuksen muodostavat neljännekset Q1/2008 ja Q3/2008. Täysin vastaava ilmiö on havaittavissa myös vertailtaessa kaikkien tapahtumalajien uusien ja ratkaistujen tehtävien suhteita (kuva 6.5).

Tarkasteltaessa esimerkiksi Q1/2009:ää, havaitaan uusien ja ratkaistujen ta-



Kuva 6.4 Toiminnanohjausjärjestelmä, luotujen ja ratkaistujen virhetapahtumien määrät aikavälillä 1.1.2007–30.6.2009.

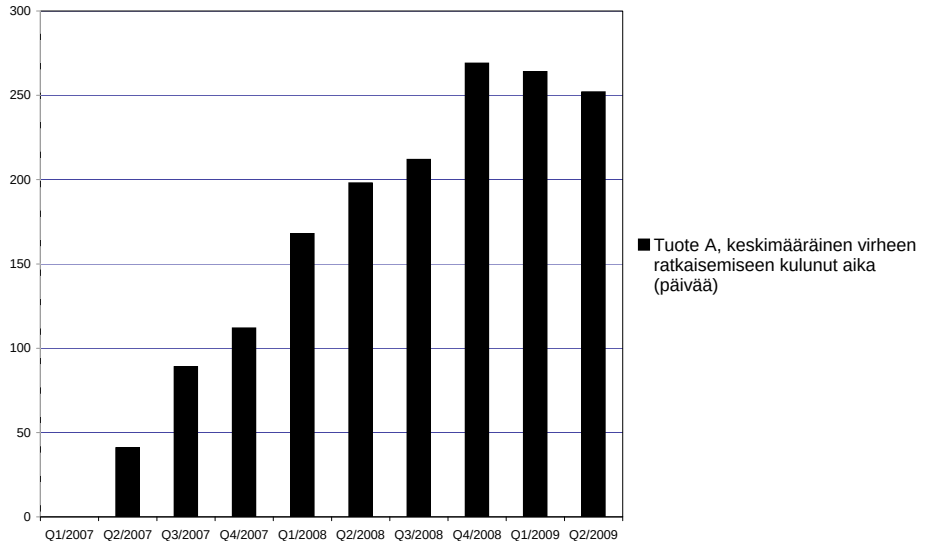


Kuva 6.5 Toiminnanohjausjärjestelmä, luotujen ja ratkaistujen tapahtumien määrät aikavälillä 1.1.2007–30.6.2009.

pahtumien eron olevan 125 tapahtumaa. Vertailun vuoksi pelkkä uusien ja korjattujen virheiden ero oli samalla ajanjaksolla 45 tapahtumaa (kuva 6.4). Sinällään suuri ero luotujen ja ratkaistujen tapahtumien välillä ei ole välttämättä huono asia. Ilmiön yksi selitys on se, että asioita kirjataan tehtävänhallintajärjestelmään

enemmän ja järjestelmällisemmin kuin ennen. Kaikkia uusia tapahtumia ei oteta heti työn alle vaan myös vanhoja tapahtumia pyritään aktiivisesti ratkaisemaan. Voi olla, että aiemmin tehtävien asioiden muistilistoja on ylläpidetty manuaalisesti työpöydällä sijainneessa muistilapuissa. Jatkuvaa luotujen ja ratkaistujen tehtävien välimatkan kasvua voidaan perustella lisäksi virhetapahtumia lukuun ottamatta sillä, että tulevaisuuden kehitystarpeita kirjataan tehtävänhallintaan, ja vain jotkut niistä toteutetaan.

Virheiden käsittelyä voidaan mitata myös toisella mittarilla: keskimääräisellä virhetapahtumien käsittelyyn kuluneella ajalla (kuva 6.6). Keskimääräinen aika kertoo, kuinka kauan tarkastelujaksolla on kestänyt keskimäärin virhetapahtuman luontihetkestä hetkeen, jolloin tapahtuma on merkitty ratkaistuksi.



Kuva 6.6 Toiminnanohjausjärjestelmä, keskimääräinen virheen ratkaisemiseen kulunut aika (päivää) aikavälillä 1.1.2007–30.6.2009.

Kuvaajan perusteella nähdään, että keskimääräinen virhetapahtumien käsittelyyn kuluva aika on lähtenyt laskemaan vuoden 2009 ensimmäisestä neljänneksestä lähtien. Ilmiö on merkillinen, koska todennäköisempää olisi keskimääräisen ajan jatkuva kasvaminen. Jatkuva kasvaminen olisi luonnollista siksi, että vakavuustasoltaan vähäisiä, esimerkiksi kosmeettisia virheitä, korjataan ehkä vasta pitkänkin ajan kuluttua niiden havaitsemisesta. Kuten aiemmin havaittiin (kuva 6.4), virheiden kokonaismäärä (uusia virheitä enemmän kuin ratkaistuja), ja siten myös vakavuustasoltaan pienempien virheiden määrä on kuitenkin kasvanut tarkastelujaksolla.

Virheiden ratkaisemiseen kuluneen keskimääräisen ajan todennäköinen selitys on se, että vuoden 2009 ensimmäisellä ja toisella neljänneksellä on korjattu enemmän uusia virhetapahtumia kuin vanhoja. Tämän tapauksen osalta ei tarkasteltu asiakasprojektien projektidatoja, mutta selitystä tukisi ajatus siitä, että asiakasprojektien kautta on tullut suuri määrä pikaisia korjauksia vaativia virhetapahtumia vuoden 2009 alussa.

6.5 Havainnot

Tuotteenhallinnan kannalta hyvin on hoidettu seuraavat asiat:

- muutostenhallintaprosessi: asiakasversioiden piirteet eivät päädy vahingossa tuoterunkoon
- tehtävänhallinnan käyttö töiden hallinnassa ja priorisoinnissa
- tuoterungon versiointi ja hallinta
- versionhallinnan käyttö versioinnissa.

Taulukkoon 6.3 on kerätty tapauksen tuotteenhallintaan liittyviä ongelmia. Ongelmien vakavuutta on arvioitu asteikolla 1-5 (1 = käytännössä ei juurikaan merkitystä, 5 = erittäin vakava ongelma).

Havainnot pohjautuvat tuotepäällikön haastatteluun, tuotteen kehitysprosessin käytön sisäisen koulutustapahtuman sisältöön ja analysoituun projektidataan.

Vakavuusluokittelu pohjautuu subjektiiviseen näkemykseen, joten luokittelua voidaan pitää lähinnä suuntaa antavana. Havainnoissa on mukana myös asiakasprojekteihin liittyviä huomioita. Tarkkaan ottaen kaikki ongelmat eivät kuulu suoranaisesti tuotteenhallinnan vastuulle, mutta vaikuttavat silti ainakin välillisesti tuotteenhallinnan toimivuuteen.

Ongelma	Vakavuus
Ominaisuuksien hallinnan prosesseja noudatetaan ainoastaan tuotteen osalta, asiakasprojekteissa käytäntö kirjavaa.	5
Tiedonkulussa on ongelmia tuotekehitystiimin ja asiakasprojekteissa työskentelevien välillä. Tieto ei kulje viereiseen huoneeseen. Tuotepäällikön mukaan valtaosa ongelmista ratkeaisi, jos kaikki puhutut muutokset saataisiin jalkautettua käytännön toimintatapoihin.	5
Toimituksia tehdään satunnaisesti, eli tuotteen ja asiakasversioiden versioiden synkronointi ei toimi nykyisin.	5
Puutteet toimitusten versioinnissa ja hallinnassa. Asiakkaan käyttämän version tarkka sisältö on joskus lähes mahdotonta selvittää asiakkaan ilmoittaessa virheestä.	4
Tuotekehityksen ja asiakasprojektien yläpuolella ei ole tällä hetkellä sellaista tahoa, joka käytännössä ajaisi yhteisiin ohjelmistoprosesseihin liittyviä asioita aktiivisesti eteenpäin.	4
Automaattisten testien käyttömahdollisuus on olemassa, mutta tutkimuksen lähtötietojen hankintahetkellä käytössä oli ainoastaan yksi tällainen testi.	4
Asiakkaille toimitetaan kerralla isompi kokonaisuus, josta asiakas haluaa monesti käyttöönsä vain osan. Kun loput toiminnoista poistetaan jälkikäteen, vaikutuksia kokonaisuuteen ei tiedetä.	4
Yleisesti ottaen koodikatselmointeja ei tehdä säännöllisesti asiakasprojekteissa (tuoterungon osalta tehdään).	4
Toteuttaminen lähtee nykyisin etenemään liian helposti. Isojen kokonaisuuksien kohdalla pitäisi ensin miettiä tarkemmin kokonaisvaikutuksia.	4
Kulttuuriongelmat kehitystiimeissä. Iskostuneet perinteet ja toimintatavat vähintäänkin hidastavat tehokkaampien ja parempien toimintatapojen käyttöönottoa.	4

Ongelma	Vakavuus
Asiakasversioiden osalta ei nykyisin aina tiedetä, mitä tuoterungon versiota käytetään.	4
Tuoterungon prosessien mukaista katselmointia ja testausvaihetta ei tehdä asiakasprojektien asiakaspiirteille eli niille piirteille, jotka eivät tule tuoterunkoon. Näiden piirteiden testaamisesta vastaa lähinnä toteuttaja.	4
Ohjelmistoon tehtyjen muutosten dokumentointia ei tehdä aina käytetyissä prosesseissa sovitulla tavalla. Tekniseltä kirjoittajalta kuluu kohtuuttomasti aikaa puutteiden selvittämiseen.	4
Asiakasversioihin alun perin toteutettujen ominaisuuksien siirtäminen tuotteeseen on kallista. Työmäärät arvioidaan näissä tapauksissa yleensä todellista alhaisemmiksi. Työvaiheen kesto on tyypillisesti vähintään sama kuin ominaisuuden toteuttaminen asiakasversioon.	3
Asiakasprojekteissa ei tehdä ominaisuuksien järjestelmällistä testausta.	3
Asiakkaat testaavat uusia ja muuttuneita ominaisuuksia hyvin vaihtelevasti. Pienet asiakkaat eivät välttämättä edes ymmärrä testauksen merkitystä tai ehdi testaamaan, suuremmilla saattaa olla jopa testaamiseen erikoistuneita henkilöitä.	3
Puutteet tuotekehityssatsauksissa. Projektiryhmissä saattaa joillakin henkilöillä ”pyöriä peukalot”, eikä vapaata aikaa läheskään aina voida hyödyntää tuotekehityksessä, koska tuotekehitykseen myönnetty henkilötyötuntimäärä ei sitä mahdollista.	3
Tuotekehitystiimi ei pysty viemään helposti läpi suurempia kokonaisuuksia, koska tiimin jäseniä tarvitaan alituiseen muuallakin, esimerkiksi myynnin tuen tehtävissä.	3
Testaamiseen ei ole aina käytettävissä henkilöä. Testaajat saattavat olla esimerkiksi koulutustehtävissä. Toteutettujen ominaisuuksien testaaminen viivästyy. Kun niitä viimein testataan paljon myöhemmin, kuluu kehittäjältäkin paljon aikaa palautella mieleen jonkin jo aikaa sitten toteutetun piirteen yksityiskohtia.	3

Ongelma	Vakavuus
Nykyisten versionhallinnan tapahtumien kirjaustarkkuus vaihtelee. Toisinaan asioita ei ole kuvattu riittävän tarkasti, minkä seurauksena asioita voidaan ymmärtää väärin.	3
Jotkut versionhallinnan kirjaukset ovat aivan liian isoja kokonaisuuksia, yhden kirjauksen tekemiseen voi kulua aikaa jopa kaksi viikkoa.	3
Testiympäristössä ei ole käytössä asiakkaan tietomassoja, osa ongelmista tulee esille vasta tuotantokäytössä.	3
Ehkä noin kerran viikossa joltain kehittäjältä unohtuu viedä versionhallintaan jokin tiedosto.	2
Päätös siitä, tuleeko uusi ominaisuus tuotteeseen vai asiakasversioon, ei toimi nykyisin aina prosessissa kuvatulla tavalla. Myyjä saattaa myydä asiakkaalle jonkin piirteen tuotteeseen tehtäväksi, jolloin asiakas säästää rahaa. Tällöin ominaisuutta ei enää voida valita asiakasversioon tehtäväksi, mikä olisi järkevämpää joissakin tapauksissa.	2
Tietokannan parametrointitietoja on kehittäjien omilla koneilla (eli niitä ei hallita keskitetysti).	1

Taulukko 6.3 Toiminnanohjausjärjestelmä, tuotteenhallinnan ongelmakohtia.

Tuotteenhallinnasta ei löytynyt tuoterungon osalta pahoja puutteita. Projektidatan perusteella tuoterungon konfiguraatioiden ja komponenttien hallinta tehdään esimerkillisesti. Osoituksena siitä on esimerkillinen versiointikäytäntö. Havaitut tuotteenhallinnan puutteet liittyvät lähinnä toimintatapoihin. Havaintoja analysoidaan tarkemmin tutkielman yhteenvedossa, luvussa 9.

7 Tuotannonohjausohjelmisto

7.1 Esittely

Toinen tapaustukimuksen kohde on tuotannonohjausohjelmisto (engl. Manufacturing Execution System, MES) [ITtoolbox Popular Q&A Team, 2009]. Tässä tutkielmassa tarkoitetaan termillä tuotannonohjausohjelmisto nimenomaisesti tätä tapausta, ei tuotannonohjausohjelmistoja yleisesti. Järjestelmää on kehitetty hieman laskentatavasta riippuen noin 10–15 vuotta. Järjestelmän pääkohderyhmänä ovat suuret ja keskisuuret elintarvike- ja einestoimialueen yritykset. Tuotannonohjausohjelmisto on teollisuuden kokonaisvaltainen ohjelmistoratkaisu tuotannon materiaalivirtojen ja inventaarien kustannustehokkaaseen prosessihallintaan. Tuotannonohjausohjelmisto kattaa tuoterakenteiden ja resurssien optimointitoiminnot sekä tuotekehityksen välineet. Useimmiten tuotannonohjausohjelmistoa käytetään toiminnanohjausjärjestelmien kanssa parantamaan tuotannon tehokkuutta ja järjestelmien käytettävyyttä. Ratkaisu kattaa kaikki tuotannon prosessit sekä integroi keskeiset toiminnot toiminnanohjausjärjestelmään ja muihin tietojärjestelmiin. Ratkaisun tavoitteena on:

- alentaa liiketoiminnan kustannuksia
- taata tuotantoprosessin jäljitettävyys
- varmistaa lopputuotteen laadun tasaisuus kustannustehokkaimmalla mallilla
- parantaa liiketoiminnan laatua.

7.2 Tutkimuksen taustaa

Alun perin tuote kehitettiin osana asiakasprojektia vastaamaan yksittäisen asiakkaan tarpeita. Myöhemmin sen pohjalta jalostettiin yleiskäyttöisempi tuoterunko, johon asiakasprojektit pohjautuvat. Tuoterunko ei ole sellaisenaan käytössä millään asiakkaalla.

Asiakasprojekteja toteutetaan laajentamalla tuoterunkoa uusilla yleisillä ominaisuuksilla ja toteuttamalla asiakaskohtaisia piirteitä asiakasversioihin, jotka pohjautuvat tuoterunkoon.

Tuotannonohjausohjelmistoa kehittää 10 henkilön projektiryhmä. Kuulun tällä hetkellä itse tähän ryhmään. Sama ryhmä vastaa sekä tuoterungon että

siihen pohjautuvien asiakasprojektien toteuttamisesta. Verrattuna toiseen tutkittavaan tapaukseen, toiminnanohjausjärjestelmään, tuotannonohjausohjelmiston tuotekehitysresurssit ovat vaatimattomat. Käytännössä koko tuotannonohjausohjelmiston kehitystiimi toteuttaa asiakasprojekteja, joiden ohessa ylläpidetään myös tuoterunkoa. Tästä johtuen tuotannonohjausohjelmiston tapaustutkimuksessa ei erotella tuoterunkoa ja asiakasprojekteja toisistaan, vaan tapausta käsitellään yhtenä kokonaisuutena.

Koska ohjelmistolla ohjataan tyypillisesti tuotannon toimintaa, on sen toimivuus asiakkaiden ympäristöissä erittäin kriittistä. Mahdollisten vikatilanteiden ilmetessä ongelman selvittäminen on yleensä aloitettava välittömästi. Asiakkaasta riippuen ohjelmistoon joudutaan toimittamaan muutoksia joskus useinkin, jos esimerkiksi tuotannon prosessit muuttuvat.

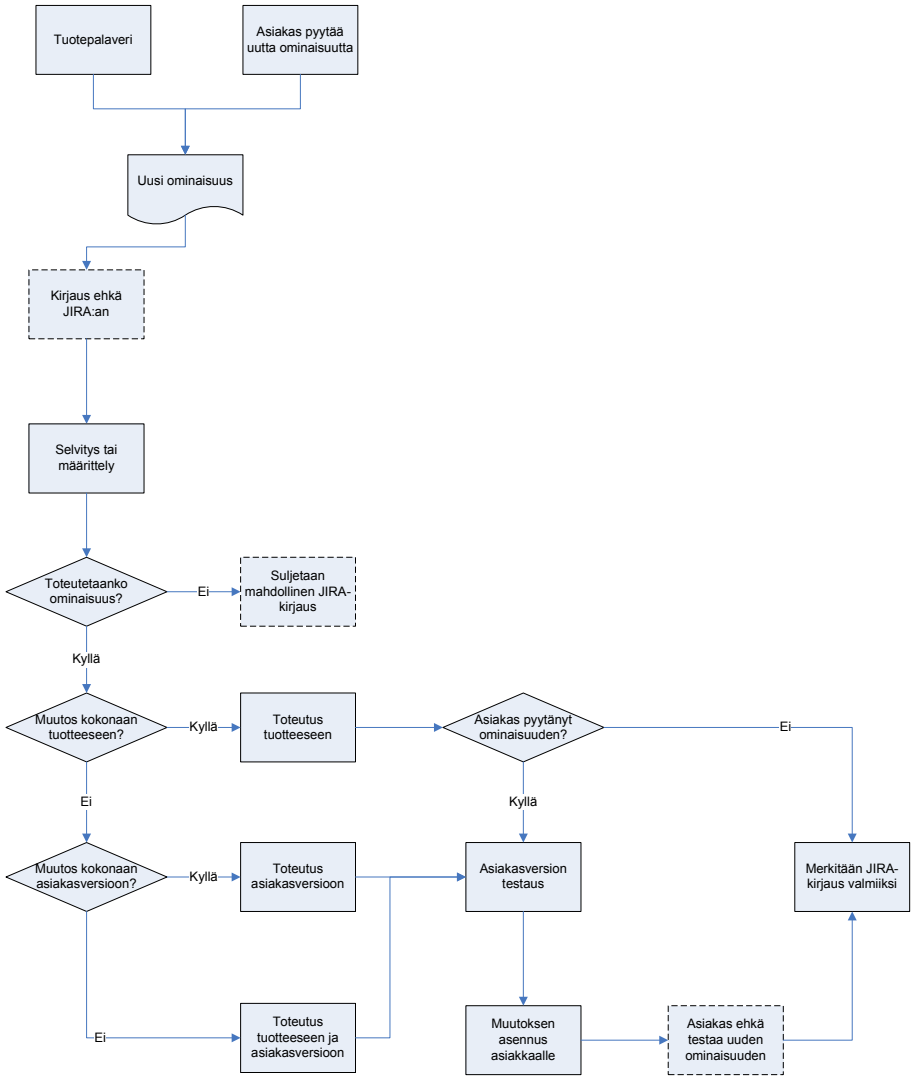
Tuotannonohjausohjelmisto pohjautuu Microsoftin .NET -teknologioihin [Microsoft, 2009c]. Sovelluskehitysvälineinä on pääasiassa Microsoft Visual Studio [Microsoft, 2009b] ja Microsoft SQL Server [Microsoft, 2009a]. Sovelluskoodin automaattiseen kääntämiseen (jatkuvaan integrointiin) käytetään Atlassian Bamboo -järjestelmää [Atlassian, 2009a]. Muut kehityksen tukena käytettävät työkalut (tehtävänhallinta, versionhallinta ja tietämyksenhallintajärjestelmä) ovat samat kuin Toiminnanohjausjärjestelmän tapauksessa (JIRA [Atlassian, 2009e], Subversion [Subversion, 2009] ja Confluence [Atlassian, 2009b]).

Tuotannonohjausohjelmiston kehittämisessä käytettävät prosessit pohjautuvat suulliseen ohjeistukseen ja tehtävänhallinnan sisältämään työkulun ohjaukseen. Osana tapaustutkimusta mallinnettiin tärkeimmät tuotteenhallintaan liittyvät prosessit kaaviokuviksi analysointia varten. Näiden prosessien ohella tutkimusmateriaali koostuu tuotannonohjausohjelmiston järjestelmäarkkitehdin haastattelusta [Henkilö B, 2009] ja tehtävänhallintajärjestelmästä kerätyistä tiedoista. Haastattelumenetelmänä käytettiin toisen tapauksen tapaan teemahaastattelua, haastattelukysymykset (Liite 1) olivat samat molemmissa tutkittavissa tapauksissa.

7.3 Prosessit ja toimintatavat

7.3.1 Uuden ominaisuuden lisääminen tuoterunkoon

Uuden ominaisuuden (tai piirteen) lisääminen tuoterunkoon (tuotteeseen) (kuva 7.1) lähtee liikkeelle joko asiakastarpeesta tai sisäisesti havaitusta kehitystarpeesta. Sisäiset kehitystarpeet käsitellään tuotepalaverissa. Samassa palaverissa tehdään myös päätökset ominaisuuksien toteuttamisista.



Kuva 7.1 Tuotannonohjausohjelmisto, uuden ominaisuuden lisääminen tuoterunkoon.

Jotkut asiakkaat saattavat kirjata uuden ominaisuuden toteutuspyynnön itse tehtävänhallintajärjestelmään (JIRA). Ilmoitus uudesta ominaisuustarpeesta voi tulla myös puhelimitse tai sähköpostitse. Ominaisuuspyynnön vastaanottamisen jälkeen se kirjataan tehtävänhallintajärjestelmään. Joskus kirjausta ei muisteta tehdä lainkaan. Uuden ominaisuuden tarve voi myös olla aluksi epävarmaa, mutta varmistuu myöhemmin. Kuitenkaan kirjausta tehtävänhallintajärjestelmään ei välttämättä tehdä tarpeen varmistumisenkaan jälkeen.

Jos uuden toiminnon toteuttamispyyntö on tullut asiakkaalta, selvitetään tässä uuden ominaisuuden toteuttamisen edellytykset eli mitä uuden ominaisuuden toteuttaminen maksaa ja miten se kannattaisi toteuttaa, jotta asiakas olisi tyytyväinen lopputulokseen. Luonnollisesti selvitetään myös, onko uuden toiminnon toteuttaminen ylipäätään teknisesti mahdollista.

Asiakas ei päätä siitä, tuleeko uusi ominaisuus osaksi tuotetta vai asiakaskohtaista versiota. Uuden ominaisuuden kohdalla joudutaan aina miettimään, tehdäänkö uusi ominaisuus kokonaan asiakasversioon, kokonaan tuotteeseen vai jaetaanko siitä osia kumpaankin. Asiakaskohtaisten versioiden osalta projektipäällikkö voi päättää, että uusi ominaisuus tehdään kokonaan asiakaskohtaiseen versioon. Jos ominaisuus halutaan tehdä tuotteeseen, sopii asianomainen projektipäällikkö sen osittaisesta tai kokonaisesta sisällyttämisestä tuotteeseen tuotepäällikön kanssa pidettävässä palaverissa. Palaveri saattaa olla hyvin epämuodollinen ”käytäväkeskustelu”, eikä siitä välttämättä laadita mitään kirjallista dokumenttia.

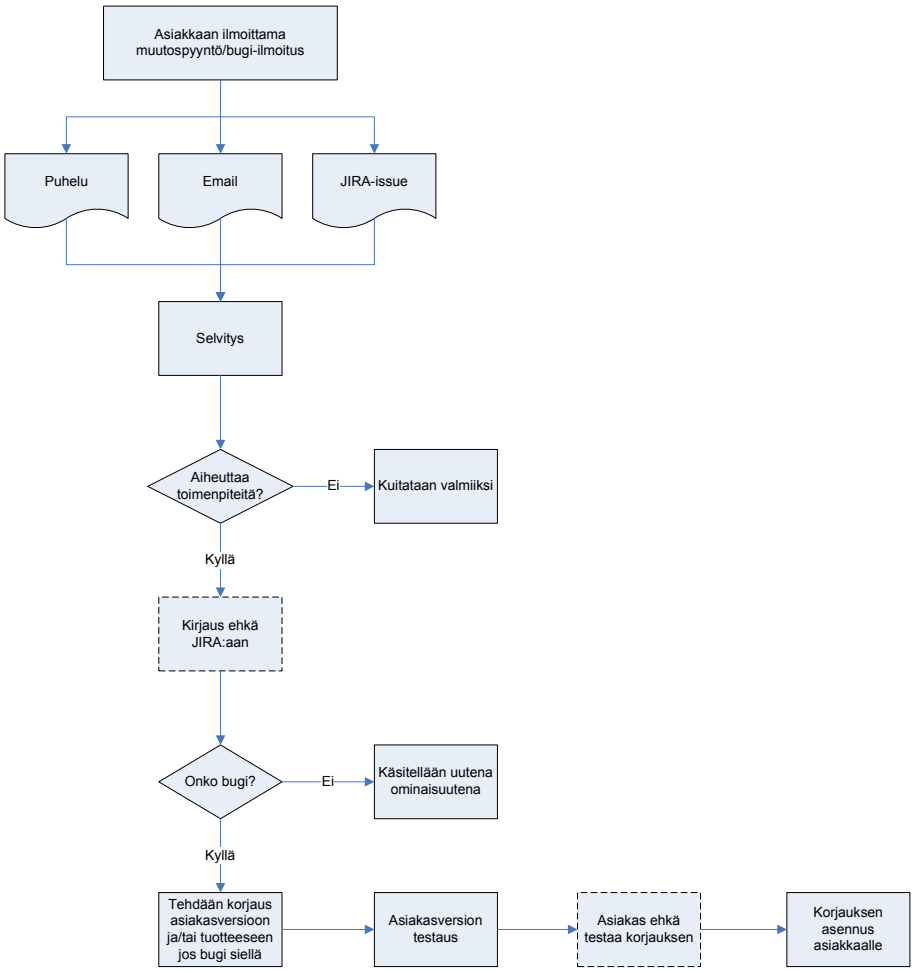
Jos uusi ominaisuus ei ole asiakkaan ilmoittama, toteutetaan ominaisuus pelkästään tuoterunkoon.

Tämän jälkeen ominaisuus toteutetaan asiakasversioon, tuotteeseen tai molempiin. Ominaisuuden toteuttaja testaa toteutetun muutoksen asianomaisen asiakkaan asiakasversiossa. Yleensä tuoterunkoa tai muita asiakasversioita ei testata uuden ominaisuuden toteuttamisen yhteydessä muutoin. Käytetyt ohjelmistokehitysvälineet tukevat kuitenkin automaattisten testien käyttöä, joka korvaa osittain manuaalisen testauksen vähyyttä. Uusien automaattisten testien kirjoittaminen on kuitenkin uuden ominaisuuden kehittäjän vastuulla, ja ne saattavat jäädä tekemättä kiireessä.

Uusi ominaisuus toimitetaan asiakkaan testiympäristöön tai asiakkaan niin halutessa suoraan tuotantoon. Jos ominaisuus toimitetaan asiakkaan testiympäristöön, testaa asiakas uutta ominaisuutta yleensä jossain määrin ennen sen asentamista tuotantoympäristöön. Asiakkaiden testaamisen taso ja kattavuus vaihtelevat suuresti.

7.3.2 Muutos ominaisuuteen

Uuden ominaisuuspyynnön tapaan asiakas saattaa kirjata muutospyynnön (kuva 7.2) itse suoraan tehtävänhallintajärjestelmään. Muutospyyntö voi tulla myös sähköpostitse tai puhelimitse.



Kuva 7.2 Tuotannonohjausohjelmisto, korjaus tai muutos toimintoon.

Muutospyynnön vastaanottaja (yleensä asiakasversion projektipäällikkö) tekee lyhyen selvityksen muutospyynnön sisällöstä. Selvitettävät asiat ovat samat kuin uudenkin ominaisuuden yhteydessä: tekniset ja taloudelliset edellytykset ja rajoitteet. Joskus käy ilmi, että asiakkaan pyytämää muutosta ei tarvitse tehdä, vaan pyydetyn ominaisuumuutoksen asia voidaan tehdä jollakin jo olemassa olevalla toiminnolla.

Jos muutospyynnön todetaan edellyttävän sovelluskoodimuutoksia, saatetaan se kirjata tehtävänhallintajärjestelmään. Muutos saattaa jäädä kirjaamatta, jos kyseessä on esimerkiksi yksinkertainen tietokannan siivoukseen liittyvä asia.

Samassa yhteydessä, kun muutospyyntö kirjataan tehtävänhallintajärjestelmään, tehdään muutoksen luokittelu: Onko kyseessä bugi vai muutospyyntö? Taloudellisesti luokittelulla voi olla suurikin merkitys, erityisesti asiakasversion takuuaikana. Jos kyseessä ei ole bugi, se käsitellään samoin kuin uusi ominaisuus (kuva 7.1). Jos kyseessä on bugi, korjataan se asiakasversioon, tuoterunkoon, tai molempiin.

Bugikorjaus testataan aina sen asiakkaan versiolla, jossa se havaittiin. Bugikorjauksen tekijä testaa bugikorjauksen yleensä uusimmalla asiakasversion kehitysversiolla, eikä siten välttämättä sillä versiolla, joka asiakkaalla on käytössä. Bugikorjauksen vaikutuksia muiden asiakkaiden versioihin tai tuoterunkoon ei yleensä erikseen testata. Vastaavasti kuten uudenkin ominaisuuden yhteydessä, bugikorjauksenkin (ehkä) testaa asiakas. Jos korjauksen viemisellä tuotantoon on kova kiire, asiakkaan kanssa voidaan sopia korjauksen asentamista suoraan asiakkaan tuotantoympäristöön.

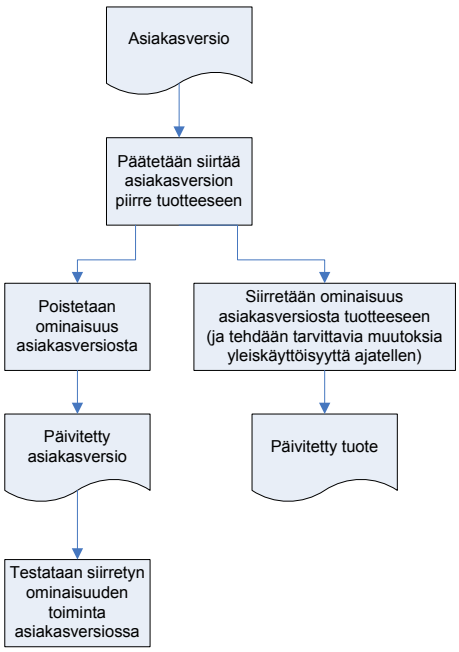
7.3.3 Ominaisuuden siirtäminen tuoterunkoon, nykytila

Uusien ominaisuuden viennissä tuoterunkoon ei yleensä hätiköidä. Yksi syy tähän on se, että uusi ominaisuus voi olla asiakaskohtainen tai ominaisuuden toimitusaikataulu on asiakkaan vaatimuksesta niin kireä, ettei yleiskäyttöistä (eli tuoterunkoon soveltuvaa) ominaisuutta ehditä miettimään riittävästi. Taloudellisetkin seikat vaikuttavat asiaan. Uuden ominaisuuden toteuttaminen asiakasversioon on lyhytkatseisesti halvempaa, koska ominaisuuden toiminnallisuutta ja toteutusta ei tarvitse miettiä tuotteen yleiskäyttöisyyden kannalta.

Asiakkaan pyytämä uusi ominaisuus saattaa olla jo toteuttuna jollekin toiselle asiakkaalle. Tällaisissa tilanteissa on usein järkevää käyttää jo aiemmin toteutettua toiminnallisuutta pohjana ja siirtää se osaksi tuoterunkoa siten, että se täyttää vähintäänkin näiden kahden asiakkaan tarpeet.

Nykytilan mukaisesti (kuva 7.3) ominaisuuden siirtäminen tuoterunkoon tehdään siirtämällä alkuperäisestä asiakasversiosta ominaisuuteen liittyvä sovelluskoodi tuoterunkoon sopivaan komponenttiin. Yleensä asiakaskohtainen ominaisuus ei sovellu aivan sellaisenaan tuoterungon osaksi, vaan yleiskäyttöisyyttä ajatellen sitä joudutaan laajentamaan ja lisäämään siihen konfigurointimahdollisuuksia.

Lopuksi ominaisuuden siirtämisen tehnyt kehittäjä testaa siirron onnistumisen sillä asiakasversiolla, josta ominaisuus siirrettiin tuoterunkoon.



Kuva 7.3 Tuotannonohjausohjelmisto, asiakasversion ominaisuuden siirtäminen tuoterunkoon, nykytila.

7.3.4 Ominaisuuden siirtäminen tuoterunkoon, tavoitetila

Asiakasversion ominaisuuksien siirtäminen tuoterunkoon ei ole aina suoraviivaista. Ehkä suurin ongelma liittyy siihen, että kaikkia asiakaskohtaisen ominaisuuteen liittyviä sovelluskoodin kohtia on toisinaan vaikeaa löytää, etenkin jos tehtävänhallinnan kirjaukseen liitetyt versionhallintatapahtumat ovat puutteellisia. Toimintoihin liittyvät sovelluskoodit saattavat sijaita useissa komponenteissa ja asiakaskohtaiset ominaisuudet voivat muuttaa tuoterunkoon toteutettuja toimintoja. Asiakasversion toiminto voi kääntyä ja käynnistyä ongelmitta tuoterunkoon siirtämisen jälkeen, mutta se ei välttämättä toimi oikein.

Tuotannonohjausohjelmiston järjestelmäarkkitehdin mukaan [Henkilö B, 2009] uusien toimintojen osalta pitää miettiä karkean tuotteeseen vai asiakasversioon -jaottelun lisäksi myös sitä, voisiko asiakasversioon toteutettu ominaisuus tulla myöhemmin osaksi tuoterunkoa. Tätä ajatusta tukee jo osittain käytössä oleva ominaisuuksien siirtämiseen liittyvä prosessi (kuva 7.4).

Prosessin alkutilanteessa pohditaan, tuleeko uusi ominaisuus tuoterunkoon vai asiakasversioon. Jos ominaisuus toteutetaan tuotteeseen, prosessi ei eroa nykytilasta.

Jos ominaisuus päätetään toteuttaa asiakasversioon ja se on sellainen, että se voitaisiin ottaa myöhemmin osaksi tuoterunkoa, merkitään kaikki asiakasversioon toteutettava sovelluskoodi siten, että se löydetään helposti myöhemmin. Teknisesti tämä toteutetaan .NETin attribuuttien avulla.

Toinen muutos nykytilaan on ominaisuuksien tuoterunkoon siirtämisen käynnistäjä. Nykyisin ominaisuuksia on siirretty asiakasversiosta tuoterunkoon silloin, kun asiakas on pyytänyt omaan versioonsa sellaista ominaisuutta, joka löytyykin jo jonkin toisen asiakkaan versiosta. Tavoitetilassa tuotetta kehitetään itsenäisesti ja etsitään aktiivisesti asiakasversioista sellaisia toimintoja, joita kannattaisi siirtää osaksi tuotetta.

7.3.5 Konfiguraatioiden ja komponenttien versiointi

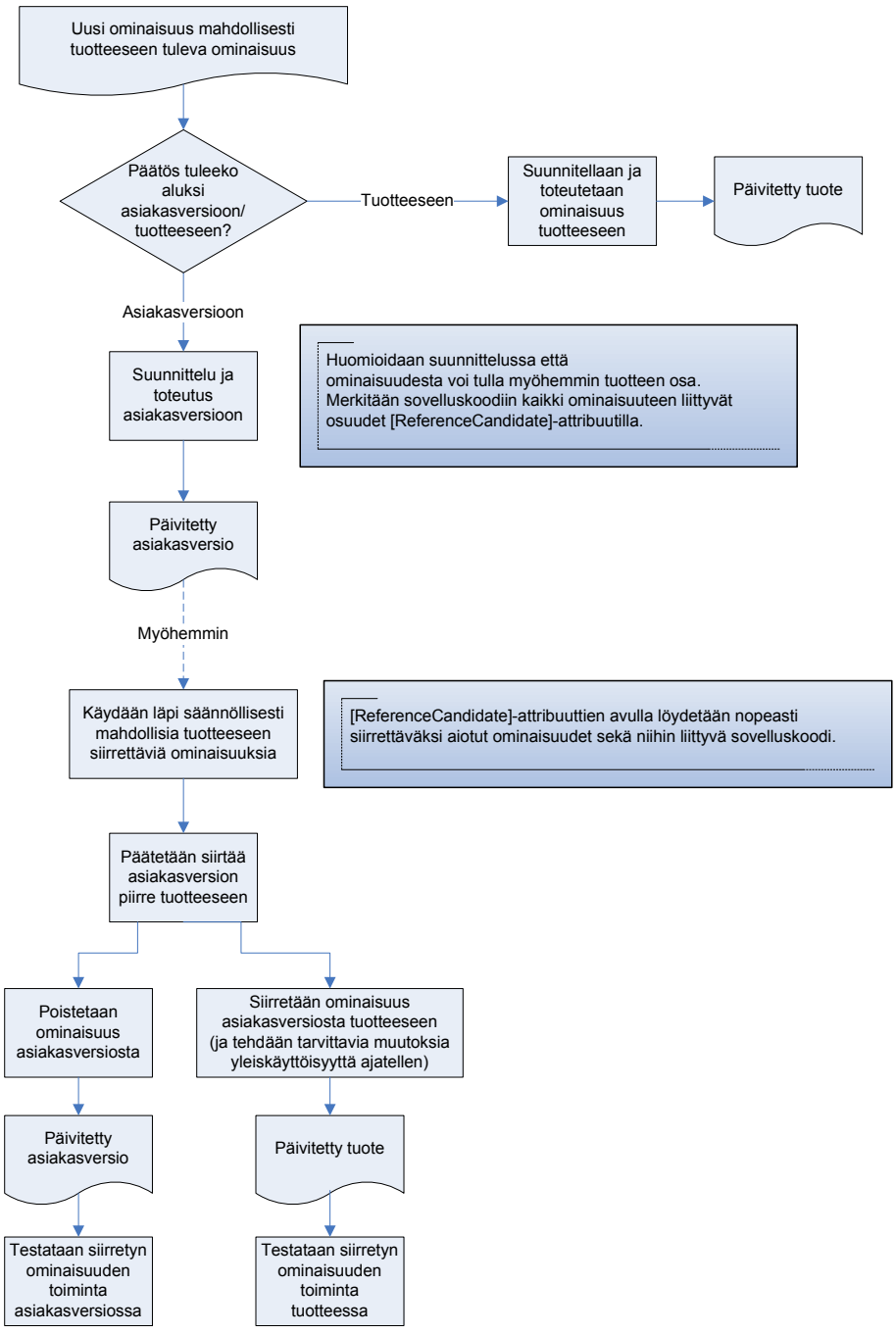
Kaikki tuoterunkoon ja asiakasprojekteihin liittyvä materiaali säilytetään yrityksen versionhallintajärjestelmässä lukuun ottamatta joitakin tietokannan parametrointidatoja, joiden varastointi keskitetysti ei ole mielekästä. Versionhallinnan käyttämiseen ei ole liittynyt suuria ongelmia. Mahdolliset virhetilanteet saadaan nopeasti kiinni, koska jatkuvan integroinnin palvelin ilmoittaa asianomaisille sovelluskehittäjille sähköpostitse mahdollisista virheistä. Tuoterungon versiojulkaisusta vastaa tuotteen järjestelmäarkkitehti. Asiakasprojektien versiojulkaisuista vastaa kunkin projektin projektipäällikkö.

Tuotehallinnan konfiguraatiot ja tuoterungon sekä asiakasversioiden versionumerot ovat käytännössä sama asia, erillistä konfiguraatioiden hallintaa ei tehdä. Versioinnissa on käytössä kolme tasoa (taulukko 7.1):

	Pääversio	Aliversio	Kriittiset korjaukset aliversioon
Esimerkki	5	4	1

Taulukko 7.1 Tuotannonohjausohjelmiston versiointikäytäntö.

Pääversio ja aliversio ovat aina käytössä. Kolmannen tason versiointia tehdään vain tarpeen mukaan, jos ilmenee tarve tehdä edelliseen aliversioon kriittisiä korjauksia.



Kuva 7.4 Tuotannonohjausohjelmisto, asiakasversion ominaisuuden siirtäminen tuoterunkoon, tavoitetila.

7.4 Projektidata

Tässä alakohdassa tarkastellaan tuotteenhallinnan prosessien tuottamaa projektidataa. Projektidata toimii taustatietona jäljempänä luvussa esiteltävälle havain-

totaulukolle ja antaa myös viitteitä tuotteenhallinnan prosessien toimivuudesta. Tämän tapauksen osalta analysoitava projektidata koostuu sekä varsinaisen tuoterungon että siihen pohjautuvien tuotteiden yhteiseen projektidataan. Projektidatan tarkasteluajanjakso on pääasiassa 1.1.2007–30.6.2009. Alkuajaksi on valittu vuoden 2007 alku, koska tapauksen tehtävähallintaan alettiin tuolloin käyttää täysipainoisesti JIRA-tehtävänhallintajärjestelmää. Loppuajankohdaksi valittiin tutkimuksen toteutushetkeä lähin kokonainen vuosineljännes. Taulukossa 7.2 on esitetty tuotannonohjausohjelmiston tapahtumien jakaantuminen eri tuotteenosien välille.

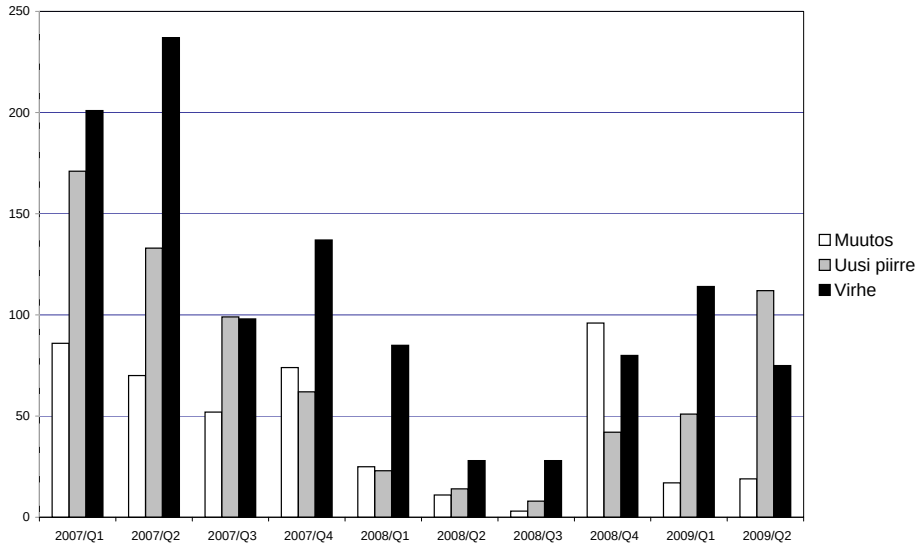
Komponentti	Tapahtumia	Osuus
Windows käyttöliittymät	210	28 %
Tuoterakenne	121	16 %
Liiketoimintalogiikka	103	14 %
Perustiedot	51	6 %
Ei määritetty	47	6 %
Varasto	40	5 %
Kustannuslaskenta	31	4 %
Tietokanta	27	3 %
Dokumentaatio	21	2 %
Tuotannon suunnittelu	20	2 %
Tuotanto	13	1 %
Laiteliittymät	13	1 %
Simulointi ja optimointi	12	1 %
Tiedostoliittymät	8	1 %
Raportointi	7	0 %
Logiikkaliittymät	5	0 %
ASP.NET	1	0 %

Taulukko 7.2 Tapahtumakirjausten jakautuminen tuotteen eri osiin.

Tapahtumien jakautumisesta eri komponenttien välille nähdään, että suurin osa tapahtumista liittyy Windows-käyttöliittymiin, tuoterakenteiden hallintaan ja liiketoimintalogiikkaan. Tuoterakenteista käytetään myös termiä reseptit. Tuoterakenne kertoo, mitä tuotteita ja kuinka paljon tarvitaan jonkin toisen tuotteen valmistamiseen. Liiketoimintalogiikka sisältää kaiken järjestelmän toimintalogii-

kan ja tietokantakäsittelyn.

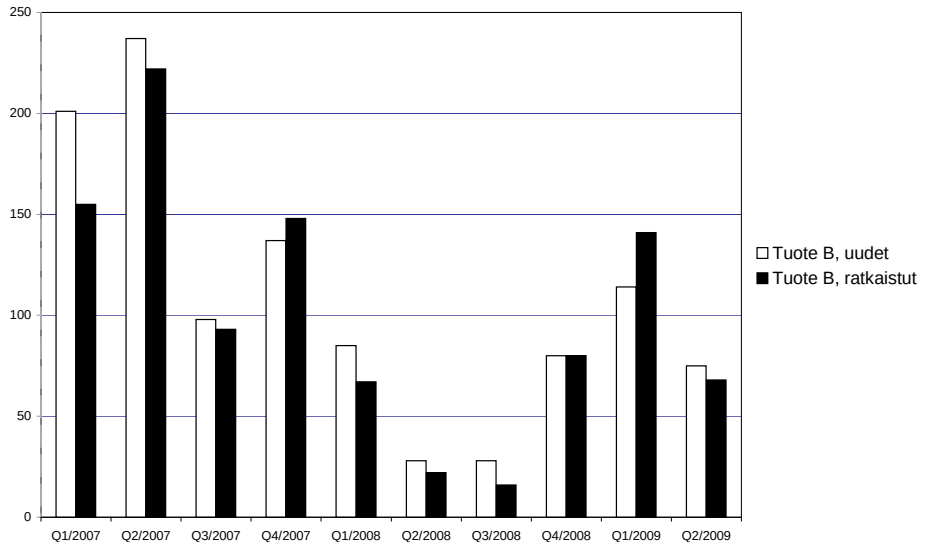
Seuraavaksi tarkastellaan tapahtumien jakautumista tapahtumalajeittain (uusi piirre, muutos tai virhe). Kuva 7.5 esittää uusien, tarkastelujaksona avattujen tapahtumien, jakaantumista eri tapahtumalajeihin vuosineljänneksittäin ajanjaksolla 1.1.2007–30.6.2009.



Kuva 7.5 Tuotannonohjausohjelmisto, luotujen tapahtumien määrä tapahtumalajeittain aikavälillä 1.1.2007–30.6.2009.

Kuvasta nähdään, että vuosineljänneksillä 2007/Q3 ja 2009/Q2, uusien piirteiden tapahtumamäärät ovat olleet suurempia kuin virhemäärien. Lisäksi vuoden 2008 neljännellä vuosineljänneksellä luotujen muutostapahtumien määrä on ollut suurempi kuin virheiden. Muina vuosineljänneksinä virheitä on kirjattu enemmän kuin muita tapahtumalajeja. Vuoden 2008 toisella ja kolmannella vuosineljänneksellä tehtävänhallinnan käyttö on ollut hyvin vähäistä verrattuna muihin vuosineljänneksiin. Tapahtumalajilta suurin kirjausluokka, virheet, sisältää 28 kirjausta molempina neljänneksinä. Tuotteenhallinnan kannalta havainto on hälyttävä! Jos kaikista virheistä ja muutoksista ei pidetä kirjaa asianmukaisesti, aiheutuu tästä varmuudella ongelmia myöhemmin, kun jäljitettävyys tehtyihin asioihin katkeaa.

Kuvasta 7.5 ei käy ilmi, kuinka paljon virheitä on pystytty korjaamaan suhteessa uusiin virhetapahtumakirjauksiin. Sitä asiaa vertailee kuva 7.6, josta käy ilmi luotujen ja ratkaistujen virhetapahtumien määrien suhde eri tarkasteluajanjaksoina.



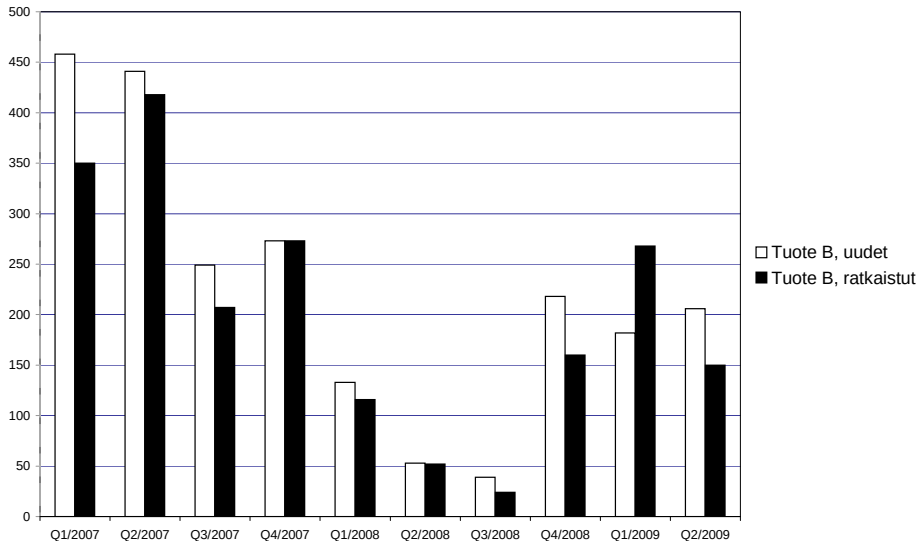
Kuva 7.6 Tuotannonohjausohjelmisto, luotujen ja ratkaistujen virhetapahtumien määrät aikavälillä 1.1.2007–30.6.2009.

Jos kuvaa 7.6 vertaa vastaavaan toiminnanohjausjärjestelmän kuvaan 6.4 havaitaan, että tuotannonohjausohjelmistossa virheitä ratkaistaan huomattavasti enemmän suhteessa uusiin havaintoihin kuin toiminnanohjausjärjestelmän tapauksessa. Tilanteet eivät tosin ole täysin vertailukelpoisia, koska toiminnanohjausjärjestelmän osalta kuvaajassa esitettiin ainoastaan tuoterunkoon liittyvät virheet. Tuotannonohjausohjelmiston osalta mukana ovat myös asiakasprojektit.

Joka tapauksessa tuotannonohjausohjelmiston kuvaaja kertoo järjestelmän kriittisyydestä. Sovellusvirheet voivat pahimmillaan pysäyttää tehtaan toiminnan, mutta myös lievemmät sovellusvirheet on korjattava. Korjaamatta jäävät lähinnä jotkut kosmiset virheet.

Samankaltainen ilmiö on havaittavissa myös vertailtaessa kaikkien tapahtumalajien uusien ja ratkaistujen tehtävien suhteita (kuva 7.7).

Kuvaajasta nähdään, että useamman vuosineljänneksen osalta uusien ja ratkaisujen tehtävien ero ei ole kovinkaan suuri, vaikka vertailuun otetaan mukaan kaikki tehtävälajit. Perustelu ilmiölle löytyy toimintatavoista: tehtävänhallintaan tehdään enimmäkseen sellaisia kirjauksia, jotka myös toteutetaan. Tapauksen

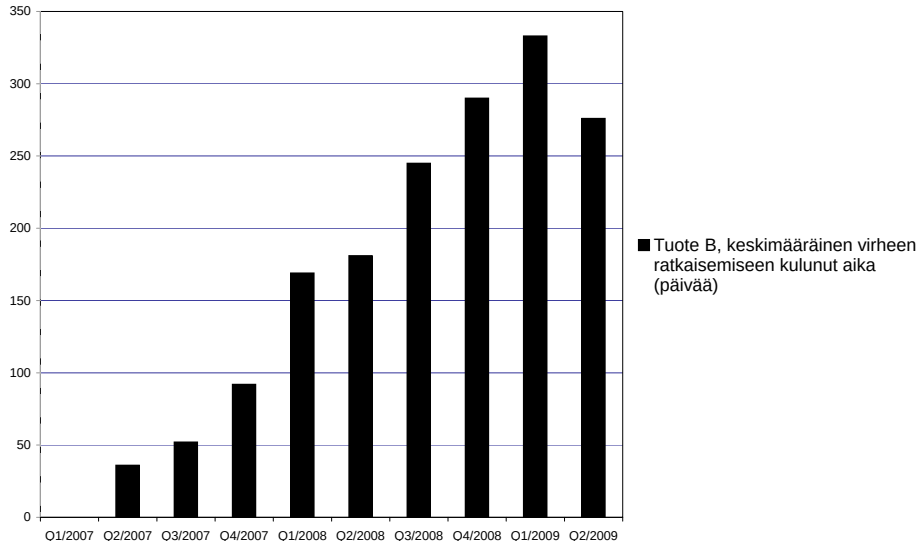


Kuva 7.7 Tuotannonohjausohjelmisto, luotujen ja ratkaistujen tapahtumien määrät aikavälillä 1.1.2007–30.6.2009.

tuotteenhallintaan liittyvissä prosesseissa esitettiin uusien ominaisuuksien kehitystarpeiden tulevan joko asiakkailta tai yrityksen sisältä. Jos asiakas esittää muutostarpeen, se pyritään toteuttamaan mahdollisimman pikaisesti. Vastaava menettelytapa koskee myös muutospyyntöjä.

Virheiden käsittelyä voidaan mitata myös toisella mittarilla: keskimääräisellä virhetapahtumien käsittelyyn kuluneella ajalla (kuva 7.8). Keskimääräinen aika kertoo, kuinka kauan tarkastelujaksolla on keskimäärin kestänyt virhetapahtuman luontihetkestä hetkeen, jolloin tapahtuma on merkitty ratkaistuksi.

Kuvan 7.8 perusteella nähdään, että keskimääräinen virhetapahtumien käsittelyyn kuluva aika pudonnut vuosineljänneksen Q2/2009 osalta alhaisemmaksi kuin vuosineljänneksellä Q4/2008, vaikka neljänneksellä Q1/2009 keskimääräinen virheidenkäsittelyaika on ollut kumpaakin korkeampi. Sinänsä tämä havainto ei ole kovin yllättävä kuvan 7.6 perusteella. Vuosineljänneksellä Q1/2009 on ratkaistu virheitä huomattavasti enemmän kuin kirjattu uusia (114 uutta, 141 ratkaistua). Tällöin on suljettu huomattavan paljon vanhentuneita virhetapahtumia, eli sellaisia tapahtumia, joiden korjaaminen ei enää ole tarkoituksenmukaista. Sulkemisten yhteydessä tapahtumille on saatu määritettyä käsittelyaika, joka puolestaan kasvattaa keskimääräistä virheiden ratkaisemiseen kulunutta aikaa huomattavasti. Vuosineljänneksen Q2/2009 alhaisempi ratkaisuaika selittyy puolestaan sillä, että Q1/2009:lle osuneen tehtävänhallinnan läpikäynnin seurauksena



Kuva 7.8 Tuotannonohjausohjelmisto, keskimääräinen virheen ratkaisemiseen kulunut aika (päivää) aikavälillä 1.1.2007–30.6.2009.

noussut keskimääräinen virheen ratkaisemiseen kuluva aika oli helppo alittaa.

7.5 Havainnot

Tuotteenhallinnan kannalta hyvin on hoidettu seuraavat asiat:

- versionhallinnan käyttö versioinnissa
- versiointi asiakasprojekteissa
- integraatiopalvelimen käyttö automaattisiin käännöksiin ja –testitapauksiin (nähdään esimerkiksi etteivät tuoterungon muutokset riko asiakasversioita).

Taulukkoon 7.3 on kerätty tapauksen tuotteenhallintaan liittyviä ongelmia. Havainnoissa on mukana sekä tuoterunkoon että asiakasprojekteihin liittyviä huomioita. Tarkkaan ottaen kaikki ongelmat eivät kuulu suoranaisesti tuotteenhallinnan vastuulle, mutta vaikuttavat silti ainakin välillisesti tuotteenhallinnan toimivuuteen.

Ongelmien vakavuutta on arvioitu asteikolla 1-5 (1 = käytännössä ei juurikaan merkitystä, 5 = erittäin vakava ongelma). Havainnot pohjautuvat tuotepäällikön haastatteluun ja analysoituun projektidataan. Vakavuusluokittelu pohjautuu subjektiiviseen näkemykseen, eli luokittelua voidaan pitää lähinnä suuntaa antava.

Ongelma	**Vakavuus**
Ohjelmistoprosessi on liian joustava, muutosten tekeminen ohjelmistoon on liian vapaata, yksittäinen kehittäjä saattaa toteuttaa muutoksia jopa kysymättä asiaa keneltäkään toiselta.	5
Tiedonkulussa on ongelmia. Tieto ei aina kulje samojenkaan asioiden parissa työskentelevien välillä.	5
Kaikkia muutoksia tai korjauksia ei kirjata yrityksen käyttämään versionhallintajärjestelmään, jolloin joidenkin muutosten jäljitettävyysketju katkeaa (ongelma erityisesti henkilöstövaihdokset).	5
Yksittäisillä henkilöillä on paljon ”hiljaista tietoa” ja asiakasosaamista, josta muut eivät tiedä mitään.	4
Puutteellinen laadun valvonta, testaus on lähinnä kehittäjien ja asiakkaan vastuulla.	4
Koodikatselmointeja ei tehdä säännönmukaisesti.	4
Tuotekehityksestä puuttuu selkeä pitkän tähtäimen suunnitelma ja budjetti.	4
Asiakkaat eivät osaa kertoa vaatimuksista yksiselitteisesti ja johdonmukaisesti vaan yksi vaatimus kumoaa toisen.	3
Tuotteiden ominaisuuksia määriteltäessä ei ole käytettävissä välttämättä kuin yksi asiakasnäkökulma, loppuihin pitää käyttää mielikuvitusta.	3
Asiakkaalle tehdään pieniä korjaustoimituksia ilman asianmukaista versioinnin tekoa versionhallintaan.	3

Ongelma	Vakavuus
Kehittäjät eivät korjaa systemaattisesti aiheuttamiaan virheitä. Automaattisen ohjelmistokoodin käännöspalvelimen (jatkuva integrointi) ilmoittamat käännösvirheet eivät välttämättä tule korjatuksi ilman erillistä valvontaa.	3
Ominaisuuksien suunnittelun laajuus vaihtelevaa: Välillä suunnittelua ei juuri tehdä ja toisinaan suunnitellaan enemmän kuin olisi tarpeen.	3
Ohjelmistossa on hyvin paljon erilaisia parametreja ja testaus kaikilla kombinaatioilla on vaikeaa.	3
Versionhallintajärjestelmään kirjattujen tapahtumien kuvausten tarkkuudet vaihtelevat paljon. Aina ei kuvausten perusteella saa riittävää kuvaa tehdystä asiasta.	3
Kehittäjät unohtavat satunnaisesti joidenkin ohjelmistokoodimuutosten ja korjausten viennin versionhallintaan.	2

Taulukko 7.3 Tuotannonohjausohjelmiston tuotteenhallinnan ongelmakohtia.

Tarkastelussa ilmeni, että tapauksen osalta tuotteenhallinta toimii hyvin versionhallinnan käytön osalta. Itse versiointi on hoidettu hyvin asiakasprojekteissa, ja teknisessä mielessä myös tuoterungon osalta. Tuoterungon osalta ongelmana on se, että tulevien versiojulkistusten osalta ei varsinaisesti tehdä suunnittelua, eli ennakkoon ei tiedetä varmasti, mitkä ominaisuudet tulevat tuoterunkoon missäkin versiossa.

Muut ongelmat liittyvät ennen muuta toimintatapoihin, tiedonkulun haasteisiin ja tuotekehitysbudjetin puutteeseen. Havaintoja analysoidaan tarkemmin tutkielman yhteenvedossa, luvussa 9.

8 Tapausten vertailu CMMI-kypsyysmalliin

Tässä luvussa verrataan tapaustutkimusten tapausten tuotteenhallinnan nykytilaa CMMI-kypsyysmallin tuotteenhallintaprosessin tavoitteisiin. Vertailu pohjautuu Olsonin ja kumppaneiden [1994] tarkastuslistoihin. Niiden avulla voidaan selvittää, täyttävätkö organisaation käyttämät ohjeistukset, standardit, prosessit, käytännöt ja työkalut CMMI-kypsyysmallin vaatimukset. Tarkastuslistojen käytöstä on myös muita etuja.

Tarkastuslistat

- auttavat löytämään henkilöt, joiden tulisi osallistua prosessien kehittämiseen
- helpottavat eniten kehittämistä tarvitsevien prosessien löytämistä
- helpottavat organisaation roolien ja vastuualueiden määritystä
- auttavat organisaation sisäisten prosessien ohjeistamisessa ja prosessimallien luonnissa.

Seuraavassa arvioidaan tapaustutkimuksen tapausten tuotteenhallinnan nykytilaa edellä mainittujen tarkastuslistojen avulla. Arviot pohjautuvat tapaustukimusten haastatteluihin ja analysoituun projektidataan.

Komponenttien vaihetasot on tallennettu dokumentoidun menettelyn mukaisesti.

- Tuotteenhallintatapahtumat on kirjattu riittävän tarkasti siten, että yksittäisten komponenttien tila ja sisältö on tiedossa ja aiemmat versiot voidaan palauttaa.

  Toiminnanohjausjärjestelmä: *Tuotteen osalta sovelluskoodin nykytila ja historia on selvillä. Vanhoja dokumenttiversioita ei ehkä pystytä palauttamaan.*

  Tuotannonohjausohjelmisto: *Kaikki sovelluskoodimuutokset on tallennettu versionhallintaan. Vanhoja dokumenttiversioita ei ehkä pystytä palauttamaan.*

- Jokaisen komponentin nykytilaa ja historiatietoja (muutokset ja muut toimenpiteet) ylläpidetään.

  Toiminnanohjausjärjestelmä: *Toteutuu sovelluskoodin osalta. Kaikkien dokumenttien muutoshistorian tarkkaa sisältöä ei pystytä selvittämään.*

  Tuotannonohjausohjelmisto: *Toteutuu sovelluskoodin osalta. Kaikkien dokumenttien muutoshistorian tarkkaa sisältöä ei pystytä selvittämään.*

Vaihetasojen tarkastukset tehdään dokumentoidun prosessin mukaisesti.

- Vaihetasojen tarkastuksiin valmistaudutaan.

  Toiminnanohjausjärjestelmä: *Vaihetasojen tarkastuksia ei tehdä.*

  Tuotannonohjausohjelmisto: *Vaihetasojen tarkastuksia ei tehdä.*

- Vaihetasojen eheyttä valvotaan.

  Toiminnanohjausjärjestelmä: *Jossain määrin kyllä, sovelluskoodit käännetään automaattisesti. Dokumentteja ei aina päivitetä muiden komponenttien päivittyessä. Valvonta puutteellista.*

  Tuotannonohjausohjelmisto: *Jossain määrin kyllä, sovelluskoodit käännetään automaattisesti. Dokumentteja ei aina päivitetä muiden komponenttien päivittyessä. Valvonta puutteellista.*

- Tuotteenhallintakirjaston rakennetta ja sisältöä seurataan (komponentit, riippuvuudet).

  Toiminnanohjausjärjestelmä: *Rakennetta ja sisältöä seurataan. Kaikkia riippuvuuksia ei ole tunnistettu.*

  Tuotannonohjausohjelmisto: *Ei seurata aktiivisesti. Tarkkoja riippuvuussuhteita ei ole tiedossa.*

- Tuotteenhallintakirjaston oikeellisuutta ja täydellisyyttä tarkkaillaan (kaikki tarvittavat komponentit ovat tuotteenhallinnan alaisuudessa).

  Toiminnanohjausjärjestelmä: *Kaikki sovelluskoodit ovat tuotteenhallinnan alaisuudessa. Dokumenteista 90% on tuotteenhallinnan alaisuudessa. Loput verkkolevyillä.*

  Tuotannonohjausohjelmisto: *Kaikki sovelluskoodit ovat tuotteenhallinnan alaisuudessa. Dokumentteja on verkkolevyillä ja joitain dokumenttien versioita saattaa jäädä yksittäisille työasemille.*

- Tuotteenhallintastandardien noudattamista ja toimintatapoja tarkkaillaan.

  Toiminnanohjausjärjestelmä: *Tuotteen osalta käytössä on selkeä prosessi ja sen noudattamista valvotaan. Asiakasprojekteissa käytäntö on vaihtelevaa. Varsinaisesti ei tosin noudateta mitään valmista tuotteenhallintastandardia.*

  Tuotannonohjausohjelmisto: *Mitään valmista tuotteenhallintastandardia ei noudateta. Toimintatapoja tarkkaillaan lähinnä pistokoemaisesti, säännönmukainen valvonta puuttuu.*

- Tarkastusten tulokset raportoidaan vastuuhenkilöille.

  Toiminnanohjausjärjestelmä: *Erillisiä tarkastuksia ei toistaiseksi tehdä.*

  Tuotannonohjausohjelmisto: *Erillisiä tarkastuksia ei toistaiseksi tehdä.*

- Varmistutaan siitä, että sovitut korjaustoimenpiteet on viety loppuun asti.

  Toiminnanohjausjärjestelmä: *Koska erillisiä tarkastuksia ei tehdä, ei sovita korjaustoimenpiteistäkään.*

  Tuotannonohjausohjelmisto: *Koska erillisiä tarkastuksia ei tehdä, ei sovita korjaustoimenpiteistäkään.*

Tuotteenhallintasuunnitelma on laadittu jokaiselle projektille dokumentoidun käytännön mukaisesti.

- Tuotteenhallintasuunnitelma on laadittu ajoissa, projektin suunnitteluvaiheessa.

  Toiminnanohjausjärjestelmä: *Tuotteenhallinnan ohjeistus on yhdistetty muihin toimintaohjeisiin. Projektikohtaista tuotteenhallintasuunnitelmaa ei yleensä tehdä.*

  Tuotannonohjausohjelmisto: *Tuotteenhallinta pohjautuu suullisiin ohjeisiin. Erillisiä suunnitelmia ei ole.*

- Tuotteenhallintasuunnitelman ovat katselmoineet tahot joita se koskettaa.

  Toiminnanohjausjärjestelmä: *Varsinaisia tuotteenhallintasuunnitelmia ei ole.*

  Tuotannonohjausohjelmisto: *Varsinaisia tuotteenhallintasuunnitelmia ei ole.*

- Tuotteenhallintasuunnitelmaa hallitaan ja kontrolloidaan.

  Toiminnanohjausjärjestelmä: *Varsinaisia tuotteenhallintasuunnitelmia ei ole.*

  Tuotannonohjausohjelmisto: *Varsinaisia tuotteenhallintasuunnitelmia ei ole.*

Komponentteja koskevat muutospyynnöt ja bugiraportit avataan, tallennetaan, katselmoidaan ja hyväksytään dokumentoidun prosessin mukaisesti. Muutoksia seurataan ja kontrolloidaan dokumentoidun prosessin mukaisesti.

- Katselmointeja tai regressiotestejä tehdään sen varmistamiseksi etteivät muutokset aiheuta ei-toivottuja vaikutuksia vaihetasoihin.

  Toiminnanohjausjärjestelmä: *Tuotteen osalta jokainen muutos katselmoidaan ja testataan ennen hyväksyntää. Asiakasprojekteissa käytäntö on ainakin toistaiseksi kirjavaa.*

Tuotannonohjausohjelmisto: *Muutosten vaikutusten testaus on lähinnä automaattisten testien (integraatiopalvelin) vastuulla. Manuaalisia katselmointeja ei tehdä kuin harvakseltaan pistokoemaisesti.*

- Ainostaan ne komponentit, jotka muutoksista vastaava työryhmä on hyväksynyt viedään vaihetasokirjastoon.

  Toiminnanohjausjärjestelmä: *Kyllä. Tuotteen osalta ainoastaan katselmoidut muutokset voidaan hyväksyä. Asiakasprojekteissa käytännöt vaihtelevat.*

  Tuotannonohjausohjelmisto: *Erityistä hyväksymismenettelyä ei ole.*

- Yksittäisten komponenttien lisääminen ja muuttaminen säilyttää tuotteenhallinnan alaisten konfiguraatioiden eheyden ja oikeellisuuden.

  Toiminnanohjausjärjestelmä: *Tuotteen osalta vaikutuksia arvioidaan jo muutosten suunnitteluvaiheessa. Asiaan kiinnitetään huomiota. Asiakasprojekteissa käytännöt vaihtelevat.*

  Tuotannonohjausohjelmisto: *Toimivuutta tarkkaillaan automaattisten käännösten (integraatiopalvelin) avulla. Toimivuutta ei yleensä muutoin testata.*

Tuotteet koostetaan ja niiden julkaisua kontrolloidaan tuotteenhallinnan alaisista komponenteista dokumentoidun käytännön mukaisesti.

- Muutoksista vastaava työryhmä hyväksyy tuotteenhallintakirjastosta koostettavat tuotteet.

  Toiminnanohjausjärjestelmä: *Tuotteen osalta julkaisumenettely on dokumentoitu ja julkaisut suunnitellaan etukäteen.*

  Tuotannonohjausohjelmisto: *Julkaisuja ei suunnitella etukäteen eikä käytäntöjä ole dokumentoitu.*

- Sekä sisäiseen että ulkoiseen käyttöön tarkoitetut sovellukset rakennetaan vain sellaisista komponenteista jotka ovat tuotteenhallinnan alaisuudessa.

  Toiminnanohjausjärjestelmä: *Periaatteessa kyllä, koska kaikki sovelluskoodit sijaitsevat versionhallinnassa.*

  Tuotannonohjausohjelmisto: *Periaatteessa kyllä, koska kaikki sovelluskoodit sijaitsevat versionhallinnassa.*

9 Tulokset ja keskustelu

9.1 Tulokset ja päätelmät

Erillisiä tuotteenhallintasuunnitelmia ei laadittu kummassakaan tapauksista. Toiminnanohjausjärjestelmän osalta tuotteenhallinnan asioita oli kuitenkin muutoin dokumentoitu.

9.1.1 Työkalut

Koska molemmat vertailtavat tapaukset oli valittu saman yrityksen sisältä, molemmissa käytetyt tuotteenhallinnan työkalut olivat lähes samat. Toiminnanohjausjärjestelmän yhteydessä käytetään tehtävien priorisointiin tarkoitettua JIRA:n lisäosaa, GreenHopperia [Atlassian, 2009d], jota ei käytetä tuotannonohjausohjelmiston kehittämisessä.

Toiminnanohjausjärjestelmän versionhallinnassa muodostetaan sellaisenaan käytettävä testiympäristö joka yö. Tuotannonohjausohjelmiston tapauksessa automaattista testiympäristöä ei muodosteta, mutta sovelluskoodimuutosten toimivuus varmistetaan jatkuvaan integrointiin tarkoitetulla Bambookäännöspalvelimella [Atlassian, 2009a], joka myös ajaa kaikki sovelluskoodiin liittyvät automaattiset testit vähintään kerran vuorokaudessa.

9.1.2 Konfiguraatioiden hallinta

Molempien tapausten konfiguraatioiden hallinta oli hyvin organisoitu ja toteutettu. JIRA-tehtävänhallinnan [Atlassian, 2009e] kautta pystyy helposti tarkastelemaan vanhoja versioita (versiot vastaavat molemmissa tapauksissa konfiguraatioita) ja niiden sisältöä. Versioihin liittyvien tapahtumien kautta on edelleen mahdollista nähdä niihin liittyvät versionhallinnan tapahtumat sekä tehdyt sovelluskoodimuutokset.

Sovelluskoodimuutosten kirjautuminen osaksi tehtävänhallintajärjestelmän tapahtumaa edellyttää manuaalista tapahtumanumeron syöttämistä kommenttikenttään, kun sovelluskoodimuutoksia viedään versionhallintajärjestelmään. Pistokoemaisen tarkastelun perusteella Tuotteen A osalta jäljitettävyys oli versiojulkaisusta mahdollista aina yksittäisiin sovelluskoodiriveihin asti. Tuotteen B osalta löytyi kirjauksia, joissa lähdekoodimuutoksia ei ollut nähtävissä suoraan tapahtuman kautta.

Versionhallintajärjestelmään tallentuu jokaisesta sovelluskoodimuutoksesta ajankohta, tekijä ja itse muutos, joten manuaalisesti muutoksen jäljille on koh-

talaisen helppoa päästä käyttämällä tehtäväkirjauksesta löytyvää ajankohtaa ja muutoksen tekijää.

Toiminnanohjausjärjestelmän tuoterungon osalta versiojulkaisujen suunnittelu osoittautui toimivaksi. Seuraavia versiojulkaisuja suunnitellaan hyvissä ajoin ennen julkaisujen tekemistä. Toiminnanohjausohjelmiston osalta tuoterungon tulevien julkaisujen suunnittelu on puutteellista, koska tuleviin versioihin sisällytetään lähinnä korjauksia ja asiakasprojekteista tulleita muutoksia, mutta tarkkaa sisältöä ei mietitä etukäteen.

9.1.3 Toimintavat

Suurimmat kehityskohteet löytyvät molempien tapausten osalta toimintatapojen kehittämisestä. Molemmille tapauksille yhteinen suuri ongelma on tiedonkulku. Seinän takana työskentelevälle kollegalle ei aina välity tarpeellisia tietoja. Ratkaisu on kommunikaation lisääminen.

Tapausten välillä on samankaltaisia haasteita myös tehtävänhallinnan tapahtumakirjausten tarkkuustason hallinnassa. Toiminnanohjausjärjestelmän osalta yksittäinen tapahtuma sisältää toisinaan liian suuren kokonaisuuden ja tapahtumat on toisinaan kirjattu liian karkealla tasolla. Myös tuotannonohjausohjelmiston tapauksessa tehtävien kirjaustaso vaihtelee suuresti. Toisinaan tapahtumiin on kirjattu pelkkä otsikko, joskus tehtävät on suunniteltu jopa liian tarkalla tasolla.

Toiminnanohjausjärjestelmän tuoterungon osalta muutosten hallinnassa on käytössä selkeä prosessi, joka on myös käytännössä osoittanut toimivuutensa. Tuotannonohjausohjelmiston osalta muutosprosessi toimii heikommin, koska yksittäiset sovelluskehittäjät saattavat tehdä muutoksia jopa keskustelematta asiasta kenenkään muun kanssa. Tämä pätee erityisesti asiakasprojekteihin. Tuotannonohjausohjelmiston tuoterungon muutoksissa konsultoidaan pääsääntöisesti järjestelmäarkkitehtia ennen muutosten tekemistä, joten puutteellinen muutosprosessi ei tältä osin ole käytännössä aiheuttanut merkittäviä ongelmia.

Suurempi ongelma tuotannonohjausohjelmiston tapauksessa on se, että kaikki tehdyt muutokset eivät projektidatasta tehtyjen havaintojen perusteella (alakohta 7.4) ehkä päädy tehtäviksi tehtävänhallintajärjestelmään, joka on kuitenkin ainoa paikka, jossa on yksikäsitteisesti määritelty julkaisuversioiden sisällöt.

Toiminnanohjausjärjestelmän tuoterungon osalta tehtävien testaaminen ja katselmointi on toteutettu erinomaisesti, sillä se tehdään lähes kaikille tapahtumille. Tuotannonohjausohjelmiston osalta tuoterunkoon tehtäviä tapahtumia

ei pääsääntöisesti katselmoida eikä niitä useinkaan testaa yrityksen sisällä kuin toteuttaja.

Toiminnanohjausjärjestelmän osalta ei varsinaisesti tutkittu asiakasprojekteja, mutta haastattelussa [Henkilö A, 2009] kävi ilmi, että siihen liittyvissä asiakasprojekteissa testauskäytäntö on vaihteleva. Välttämättä muutoksia ei testaa kuin toteuttaja, sillä asiakkaat testaavat vaihtelevasti. Jotkut testaavat paljon, jotkut eivät lainkaan. Tilanne on käytännössä sama tuotannonohjausohjelmiston asiakasprojekteissa: asiakkaat ehkä testaavat muutoksia, toimittajaorganisaatiossa niitä testaa usein vain toteuttaja.

Tuotannonohjausohjelmiston kehittämisessä on alettu käyttää automaattisia testejä, joiden käyttäminen osittain korvaa manuaalisen testauksen puutteita. Toiminnanohjausjärjestelmässä automaattisia testejä oli tutkimuksen tekohetkellä käytössä vain yksi, mutta niiden käyttöä aiotaan lisätä lähitulevaisuudessa.

Tuoterungon kehittämismahdollisuuksiin vaikuttaa suuresti käytettävissä oleva budjetti. Molemmissa tapauksissa haastateltavat pitivät käytettävissä olevaa budjettia liian pienenä [Henkilö A, 2009, Henkilö B, 2009]. Toiminnanohjausjärjestelmän osalta tuotekehitykseen on varattu erillinen budjetti. Tuotannonohjausohjelmiston kehitysbudjetti on sidoksissa asiakasprojekteihin: jos asiakkaat tilaavat uusia ominaisuuksia, saatetaan niistä kehittää yleiskäyttöisempiä versioita tapauskohtaisesti harkittavat tuotekehityspanostuksen myötä. Tuotannonohjausohjelmiston tuoterungon kehittämiseen ei ole myönnetty kiinteää budjettia.

9.2 Kehitysehdotukset

9.2.1 Toiminnanohjausjärjestelmä

Toiminnanohjausjärjestelmän ongelmakohtaluettelossa (taulukko 6.3) vakavimmiksi arvioidut ongelmat liittyvät tuoterungon ja sitä käyttävien asiakasversioiden yhteiseloon. Tuoterungon kehittämisprosessit on todettu käytännössä toimiviksi ja erityisen tyytyväisiä kehittäjät ovat olleet tuotepäällikön lanseeraamaan kaikkia uusia ominaisuuksia ja muutoksia koskevaan koodikatselmointikäytäntöön. [Henkilö A, 2009].

Tuoterungon kehittämisessä käytettävät prosessit eivät ole toistaiseksi käytössä asiakasprojekteissa eikä niiden vienti jo käynnissä oleviin asiakasprojekteihin ole ehkä mahdollista. Pienillä askelilla eteneminen tuottaa varmemmin tulosta kuin kaiken muuttaminen kerralla. Kun uusia asiakasprojekteja käynnistyy, tulisi niistä valita jokin ”protoprojektiksi” ja jalkauttaa tuoterungon kehittämi-

sessä käytettyjä prosesseja sen osaksi. Jos kaikkien prosessien vienti yhdellä kertaa yhteenkin asiakasprojektiin osoittautuu liian suureksi haasteeksi, voisi tuoterungon prosessien hyviä käytäntöjä siirtää asiakasprojekteihin pala kerrallaan. Yksi ensimmäisiä asiakasprojekteihin siirrettäviä prosessin osia voisi olla katselmointikäytäntö, jossa kaikki toteutetut uudet piirteet ja muutetut ominaisuudet katselmoitaisiin ennen asiakkaalle toimittamista.

Toinen asia johon tulisi kiinnittää huomiota, on tuoterungon ja asiakasversioiden välisten suhteiden tarkempi määritys. Asiakasversioon saatetaan tuoda korjauksia uudemmasta tuoterungon versiosta kuin mihin asiakasversio pohjautuu. Myöhemmin ei aina tiedetä, liittyykö tuotu sovelluskoodi tuotteeseen vai asiakaskohtaisiin ominaisuuksiin. Ongelmaan tuskin on yhtä ainoaa oikeaa ratkaisua. Jos asiakaspiirteisiin ja tuotteeseen liittyvät sovelluskoodit on mahdollista sijoittaa eri tavoin nimettyihin tiedostoihin, tuotteen ja asiakasversioiden erot olisivat helposti nähtävissä. Toinen mahdollisuus on käyttää tehtävänhallintajärjestelmää tai versionhallinnan kommenttikenttää asian dokumentointiin.

Kolmas parannusehdotus koskee tiedonkulun parantamista. Koska tieto ei kulje seinän toiselle puolelle itsestään, voisi tiedonkulkua kehittää erityisen säännöllisesti järjestettävän tilaisuuden avulla. Aluksi sellaisen voisi pitää esimerkiksi joka toinen kuukausi. Toiminnanohjausjärjestelmän kehittämisen tukena on jo nyt erityisiä koulutustapahtumia; käytännön voisi ottaa osaksi niitä. Tilaisuuden sisältö koostuisi ennen muuta suurimpien sovelluskoodiin tehtyjen muutosten läpikäynnistä. Muutosten visuaalista hahmottamista helpottaa huomattavasti hyvä työkalu. Harkitsemisen arvoinen vaihtoehto on Atlassian FishEye [Atlassian, 2009c]. Jos kaikki tilaisuuteen osallistujat näkisivät selkeän visuaalisen esityksen koodimuutoksista, muodostuisi toivon mukaan kaikille parempi kokonaiskuva järjestelmään tehdyistä muutoksista. Tilaisuus myös aktivoisi selvittämään paremmin muutosten vaikutuksia omaan työhön. Jos muistaisi nähneensä, että joku toinen oli viime kuussa tehnyt muutoksia samoihin lähdekooditiedostoihin, joita itsellä on tarve muuttaa, heräisi uteliaisuus katsoa tehtävän- ja versionhallinnasta, mitä asiaa edellinen muutos koski. Jos ehdotettua muutosten läpikäyntitilaisuutta ei ole käytännön syistä mahdollista järjestää, kannattaisi kuitenkin ehdotetun työkalun kokeilemista ja hankkimista harkita. Sen avulla voisi yksittäinen sovelluskehittäjäkin saada helposti tietoa sovelluskoodiin tehdyistä muutoksista.

Neljäs parannusehdotus liittyy tehtävänhallinnan tapahtumien määrittelyiden (epä)tarkkuuteen. Koska on hyvin tapauskohtaista, millä tasolla määrittelyä tarvitaan, tulisi muutamaan yleisempään tapaukseen varautua ennalta. Määrittelyjen tekemistä varten voisi laatia vaikkapa kaksi erillistä pohjaa: suppea mää-

rittely ja laajempi määrittely. Yhdelläkin selkeällä pohjalla pääsee silti alkuun. Määrittelypohjassa olisi kerrottu, mitä asioita tapauksesta pitää vähintään mainita. Ymmärtämistä edelleen auttaisi, jos määrittelypohjaan olisi annettu kaikista mainittavista kohdista hyvät ja huonot esimerkit. Hyvä tavoite määrittelylle on se, että kokenut kollega pystyisi tapahtumakuvauksen perusteella toteuttaman sen sisällön ilman lisäkysymyksiä.

Viimeinen parannusehdotus koskee automaattisten testien lisäämistä. Tapaukseen liittyvän teemahaastattelun tekohetkellä käytössä oli vain yksi automaattinen testi. Automaattisia testejä ei ole mielekästä lähteä toteuttamaan kerralla koko järjestelmään. Tuoterungosta voisi valita jokin komponentin, johon lähdettäisiin aina uusien piirteiden, muutosten ja bugikorjausten yhteydessä järjestelmällisesti toteuttamaan automaattisia testejä. Saatujen kokemusten perusteella automaattisten testien jalkauttaminen muihinkin komponentteihin ja asiakasprojekteihin olisi myöhemmin mahdollista.

9.2.2 Tuotannonohjausohjelmisto

Tuotannonohjausohjelmiston ongelmakohtaluettelossa (taulukko 7.3) vakavimmiksi arvioidut ongelmat liittyvät muutosprosessin liian suureen joustavuuteen ja tiedonkulun ongelmiin. Yleisesti ottaen toiminnanohjausjärjestelmää varten annettuja kehitysehdotuksia kannattaa yrittää soveltaa tuotannonohjausohjelmistoonkin.

Tapaukseen liittyvän projektidatan analysoinnissa (kohta 7.4) nousi esille, ettei kaikkia muutoksia aina kirjata välttämättä tehtävänhallintajärjestelmään. Nykyisin vietäessä koodimuutoksia versionhallintaan, on kommenttikentän yhteyteen mahdollista syöttää käsin tehtävänhallinan tapahtuman tunniste. Usein tunniste on syötetty, mutta ei ilmeisesti aina. Yksinkertainen korjaus ongelmaan on muuttaa tehtävänhallintajärjestelmän tehtävätunnisteen syöttäminen pakolliseksi jokaisen versionhallintaan viennin yhteydessä.

Muutosprosessin kehittämiseen yksi realistinen tavoite on varmistaa, että kaikista muutoksista kirjataan tehtävänhallintaan tapahtuma. Hyvä tapa muutosten käsittelyyn olisi muuttaa tehtävänhallinnan työnkulkua siten, ettei tehtävää voi merkitä valmiiksi, jollei sitä ole toteuttajan lisäksi hyväksynyt esimerkiksi muutosten käsittelyyn erikoistunut henkilö tai työryhmä.

Tuoterungon versiojulkistuksia tulisi kehittää siten, että tiedettäisiin jo hyvissä ajoin, mitä ominaisuuksia mihinkin versioon tulisi. Koska tuoterungon kehitysresurssit ovat tällä hetkellä varsin vaatimattomat, tulisi versiopäivitysten si-

sältökin suhteuttaa vallitsevaan tilanteeseen. Tärkeää olisi saavuttaa säännölliset julkaisuaikataulut ja niiden sisällön tarkka suunnittelu etukäteen.

Työkalujen osalta kannattaa tehtävänhallinnan apuvälineeksi harkita toiminnanohjausjärjestelmän yhteydessä käytettyä GreenHopperia [Atlassian, 2009d]. Samoin sovelluskoodimuutosten havainnollistamiseen voisi harkita FishEye-työkalua [Atlassian, 2009c].

Tuotannonohjausohjelmiston kehitysprosesseissa uusia ominaisuuksia, muutoksia ja bugeja testaa toteuttajan lisäksi vain hyvin harvoin joku muu yrityksen sisällä. Aiemmin käytössä on ollut toimintatapa, jossa muutokset testaa toteuttajan lisäksi aina toinen henkilö. Resurssien vähyyden vuoksi manuaalista testaamista ei ole mahdollista lisätä nykyisestä kovin paljon. Avattaessa uusia tapahtumia olisi kuitenkin hyvä, jos tapahtumaan voisi liittää ehdon, että toteuttajan lisäksi jonkun toisen on se testattava. Henkilötyönä tehtyä testausta korvaamaan voisi edelleen lisätä automaattisten testien osuutta. Niitä on jo nyt käytössä, mutta paljon enemmänkin niitä voitaisiin hyödyntää. Esimerkiksi käyttöliittymän toimintaa ei toistaiseksi juuri testata automaattisilla testeillä. Tuotteenhallinnan kannalta automaattiset testit helpottavat esimerkiksi vanhojen versioiden ylläpitoa. Automaattisten testien avulla nähdään ettei jokin uudemmasta versiosta viety piirre tai bugikorjaus riko jotain toista asiaa vanhemmasta versiosta.

Tuotannonohjausohjelmiston osalta tuskin milloinkaan päästään siihen tilanteeseen, että jokainen tehtävänhallinnan tapahtuma koodikatselmoitaisiin. Kuitenkin samankaltaisen sovelluskoodimuutosten läpikäyntitilaisuuden, jota ehdotettiin toiminnanohjausjärjestelmään, voisi soveltaa tuotannonohjausohjelmistoonkin. Tilaisuudessa voisi käydä sovelluskoodimuutoksia läpi myös pistokoemaisesti ja antaa tarvittaessa tunnustusta tai kehitysehdotuksia liittyen tehtyihin toteutusratkaisuihin.

9.2.3 Tutkimuksen arviointi ja luotettavuus

Tuotteenhallintaan liittyvän kirjallisuustarkastelun (luvut 2-5) osalta väittämät pohjautuvat joko suoraan lähteisiin tai tehtyihin päätelmiin. Päätelmien lähteenä toimivat myös tapaustutkimuksen (luvut 6-7) yhteydessä kerätyt tiedot. Tapaustutkimuksen pohjalta tehtyjä päätelmiä voi pitää vertailukelpoisina lähinnä samankaltaisiin tapauksiin.

Tapaustutkimukseen liittyvät havainnot pohjautuvat suurimmaksi osaksi teemahaastattelujen tuloksiin. Haastateltaviksi valittiin tapauksiin liittyvien tuotteiden avainhenkilöt. Havainnot olisivat varmasti olleet toisenlaisia, jos haasteltavia

olisi ollut useampia. Tutkimuksen päätavoitteena ei ollut löytää täysin objektiivisia näkemyksiä tapausten tuotteenhallinnan tilasta vaan pikemminkin selvittää, millaisia haasteita niitä kehitettäessä on havaittu.

Tuotteenhallinnasta voisi tehdä Suomessakin laajemman tutkimuksen, jossa kartoitettaisiin esimerkiksi kyselytutkimuksella tuotteenhallintaan liittyviä asioita: käytettyjä työkaluja ja toimintatapoja. Erityisen mielenkiintoista olisi saada selville, millaisissa yrityksissä havaitaan vähiten toimintatapoihin liittyviä ongelmia.

Viiteluettelo

[Atlassian, 2009a] Atlassian. Bamboo Continuous Integration Server. http://www.atlassian.com/software/bamboo/, 2009.

[Atlassian, 2009b] Atlassian. Confluence Enteriprise Wiki. http://www.atlassian.com/software/confluence/, 2009.

[Atlassian, 2009c] Atlassian. FishEye. http://www.atlassian.com/software/fisheye/, 2009.

[Atlassian, 2009d] Atlassian. GreenHopper. http://www.atlassian.com/greenhopper/, 2009.

[Atlassian, 2009e] Atlassian. JIRA Bug and Issue Tracker. http://www.atlassian.com/software/jira/, 2009.

[Bass *et al.*, 2003] Len Bass, Paul Clemets, and Rick Kazman. *Software Architecture in Practice*, 2nd edition. Addison-Wesley, Boston, 2003.

[Benington, 1983] Herbert D. Benington. Production of Large Computer Programs. *IEEE Annals of the History of Computing*, 5(4):350–361, 1983.

[BitMover, Inc., 2009] BitMover, Inc. BitKeeper. http://www.bitkeeper.com/, 2009.

[CMMI Product Team, 2006] CMMI Product Team. CMMI for Development, Version 1.2. Technical Report CMU/SEI-2006-TR-008, Carnegie-Mellon Software Engineering Institute, 2006.

[DoD, 2005] DoD. *MIL-HDBK-61A(SE) Military Handbook: Configuration Management Guidance*. Department of Defense, USA, 2005.

[Doğru, 2005] Ali Doğru. *Toward a Component-Oriented Methodology to Build-by-Integration in Development of Component-Based Information Systems*. M. E. Sharpe, Armonk, NY, 2005.

[Estublier, 2000] Jacky Estublier. Software configuration management: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 279–289, ACM, NY, 2000.

[Fantina, 2005] Robert Fantina. *Practical Software Process Improvement*. Artech House Publishers, 2005.

[Finanssialan Keskusliitto, 2009] Finanssialan Keskusliitto. Finvoice-verkkolasku. http://www.fkl.fi/verkkolasku/yrityksen_verkkolasku/tekniset_kuvaukset_yrityksen_verkkolasku.htm, 2009.

[Fowler, 2009] Martin Fowler. Continuous Integration. http://www.martinfowler.com/articles/continuousIntegration.html, 2009.

[Gacek *et al.*, 2001] Christina Gacek, Peter Knauber, Klaus Schmid, and Paul Clemets, 2001. A Case Study: Successful Software Product Line Development in a Small Organization.

[Gao *et al.*, 2003] Jerry Zayu Gao, Jacob Tsao, Ye Wu, and Taso H.-S. Jacob. *Testing and Quality Assurance for Component-Based Software*. Artech House, Norwood, MA, 2003.

[Gilb, 1988] Tom Gilb. *Principles of Software Engineering Management*. Addison-Wesley Longman, Boston, MA, 1988.

[Git, 2009] Git. Git - Fast Version Control System. `http://git-scm.com/`, 2009.

[Haikala & Märijärvi, 2000] Ilkka Haikala ja Jukka Märijärvi. *Ohjelmistotuotanto*, 7. painos. Satku - Kauppakaari Oyj, 2000.

[Henkilö A, 2009] Henkilö A. haastattelu, 1.7.2009.

[Henkilö B, 2009] Henkilö B. haastattelu 1.6.2009.

[Hirsjärvi & Hurme, 1982] Sirkka Hirsjärvi ja Helena Hurme. *Teemahaastattelu*. Gaudeamus, Helsinki, 1982.

[IBM, 2009a] IBM. Rational ClearCase. `http://www.ibm.com/developerworks/rational/products/clearcase/`, 2009.

[IBM, 2009b] IBM. Tietoa IBM:stä. `http://www.ibm.com/ibm/fi/fi/`, 2009.

[IEEE, 1988] IEEE. IEEE Standard for Software Reviews and Audits (IEEE STD 1028-1988), IEEE Computer Society, 1988.

[ISO, 2008] ISO, 2008. The ISO Survey of Certifications 2007 (basic results).

[ISO, 2009] ISO. ISO 9001:2000 Requirements. `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=21823`, 2009.

[ITtoolbox Popular Q&A Team, 2009] ITtoolbox Popular Q&A Team. Difference Between MES and ERP. `http://erp.ittoolbox.com/documents/difference-between-mes-and-erp-15776`, 2009.

[Kimmel, 2006] Paul Kimmel. *UML Demystified*. McGraw-Hill, NY, 2006.

[Koch & Wailgum, 2008] Christopher Koch and Thomas Wailgum. ERP Definition and Solutions. `http://www.cio.com/article/40323/ERP_Definition_and_Solutions`, 2008.

[Küng *et al.*, 2009] Stefan Küng, Simon Large, and Lübbe Onken. Subversion-käyttöliittymä Windows-ympäristöön. `http://tortoisesvn.net/docs/release/TortoiseSVN_fi/`, 2009.

[Linden *et al.*, 2007] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag, NJ, 2007.

[Marttila, 2002] Ari Marttila. *ISO 9001:2000 Vaatimukset laatujärjestelmälle*. AJ Marttila Laatupaja Oy, 2002.

[Microsoft, 2009a] Microsoft. Microsoft SQL Server. `http://www.microsoft.com/sqlserver/2005/en/us/default.aspx`, 2009.

[Microsoft, 2009b] Microsoft. Microsoft Visual Studio. `http://www.microsoft.com/visualstudio/en-gb/default.mspx`, 2009.

[Microsoft, 2009c] Microsoft. .NET Framework. `http://www.microsoft.com/NET/`, 2009.

[Microsoft, 2009d] Microsoft. Visual SourceSafe. `http://msdn.microsoft.com/en-us/library/3h0544kx(VS.80).aspx`, 2009.

[Neighbors, 1980] James M. Neighbors. Software Construction Using Components, University of California, CA, 1980.

[Olson *et al.*, 1994] Timothy G. Olson, Neal R. Reizer, and James W. Over. A Software Process Framework for the Capability Maturity Model CMU/SEI-94-HB-1, 1994.

[OMG, 2009] OMG. UML 2.2 Superstructure Specification. Technical report, Object Management Group (OMG), February 2009.

[Paulk, 1994] Mark C. Paulk. A Comparison of ISO 9001 and the Capability Maturity Model for Software. Technical report, Software Engineering Institute, Pittsburgh, PA, 1994.

[Perforce Software, 2009] Perforce Software. The Perforce SCM System. `http://www.perforce.com/perforce/products.html`, 2009.

[Pressman, 2001] Roger Pressman. *Software Engineering: A Practioner's Approach*. McGraw Hill, Boston, 2001.

[Progress Software Corporation, 2009] Progress Software Corporation. Progress OpenEdge. `http://web.progress.com/index.html`, 2009.

[Royce, 1987] Winston Royce. Managing the Development of Large Software Systems. In *ICSE '87: Proceedings of the 9th International Conference on Software Engineering*, pages 328–338, IEEE Computer Society Press, Los Alamitos, CA, 1987.

[Subversion, 2009] Subversion. Subversion Version Control System. `http://subversion.tigris.org/`, 2009.

[Bugzilla, 2009] Bugzilla. The Bugzilla team. `http://www.bugzilla.org/`, 2009.

[CruiseControl, 2009] CruiseControl. The CruiseControl team. `http://cruisecontrol.sourceforge.net/`, 2009.

[TortoiseSVN, 2009] TortoiseSVN. The TortoiseSVN team. `http://tortoisesvn.tigris.org/`, 2009.

[UNECE, 2009] United Nations Economic Commission for Europe. EDI. `http://www.unece.org/trade/untdid/welcome.htm`, 2009.

[van Ommering, 2002] Rob van Ommering. Building product populations with software components. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 255–265, ACM, NY, 2002.

[W3C, 2009] W3C. Extensible Markup Language. `http://www.w3.org/XML/`, 2009.

[Wahli *et al.*, 2004] Ueli Wahli, Jennie Brown, Matti Teinonen, and Leif Trulsson. *Software Configuration Management: A Clear Case for IBM Rational ClearCase and ClearQuest UCM*. IBM Corp., Riverton, NJ, 2004.

Liite 1: Tapaustutkimuksen teemahaastattelun kysymykset ja vastaukset

Seuraavassa on esitetty teemahaastattelun kysymykset ja vastaukset tiivistetysti. Haastatelluista [Henkilö A, 2009] on toiminnanohjausjärjestelmän tuotepäällikkö ja [Henkilö B, 2009] tuotannonohjausohjelmiston järjestelmäarkkitehti (ja tuotepäällikkö).

Taustatiedot:

1. Mikä on pääasiallinen roolisi ja vastuualueesi suhteessa tuotteeseen ja asiakasversioon?

   [Henkilö A, 2009]: *Toimin tuotepäällikkönä, vastaan tuotekehityksestä ja tuotehallinnasta tuotteen (tuoterungon) osalta.*

   [Henkilö B, 2009]: *Toimin järjestelmäarkkitehtina.*

2. Työskenteletkö pääosin tuotteen vai asiakasversioiden parissa?

   [Henkilö A, 2009]: *Tuotteen.*

   [Henkilö B, 2009]: *Työskentelen suunnilleen yhtä paljon sekä tuotteen että asiakasversioiden parissa.*

3. Kuinka kauan olet työskennellyt tuotteen ja/tai asiakasversioiden parissa?

   [Henkilö A, 2009]: *Olen toiminut vuoden 2009 alusta tuotepäällikkönä ja toiminut tuotteen parissa noin neljä vuotta. Aiemmin olen toiminut asiakkaan roolissa pääkäyttäjänä 2 vuotta. Yhteensä siis noin 6 vuotta.*

   [Henkilö B, 2009]: *Noin 8 vuotta.*

4. Kuinka kauan tuotetta on kehitetty?

   [Henkilö A, 2009]: *Yli 10 vuotta.*

[Henkilö B, 2009]: *Laskentatavasta riippuen noin 10-15 vuotta.*

Konfiguraation hallinta

1. Säilytetäänkö kaikki tuotteeseen/asiakasversioihin liittyvä ohjelmistokoodi, ym. oheismateriaali versionhallinnassa?

   [Henkilö A, 2009]: *Pääsääntöisesti kyllä. Tietokannan skeema ja järjestelmään liittyvä parametrointidata ei ole. Dokumentaatiosta on versionhallinnassa ehkä 90% (loput verkkolevyllä). Koska tuotetta yritetään tehdä pitkälti parametroimalla, osa parametrointitiedoista on omalla koneella.*

   [Henkilö B, 2009]: *Kyllä.*

2. Säilytetäänkö kaikki muu tuotteeseen/asiakasversioihin liittyvä materiaali keskitetysti?

   [Henkilö A, 2009]: *Kyllä.*

   [Henkilö B, 2009]: *Kyllä.*

3. Onko ollut ongelmia sen kanssa, että olisi unohdettu laittaa jotakin versionhallintaan/muuhun tietovarastoon?

   [Henkilö A, 2009]: *Ehkä kerran viikossa käy niin, että joku on unohtanut viedä versionhallintaan muuttamansa tiedoston. Tapaukset on havaittu kuitenkin nopeasti. Asiakkaille ei ole aiheutunut unohduksista ongelmia.*

   [Henkilö B, 2009]: *Aiemmin on ollut enemmän ongelmia. Nykyisin unohdukset ovat yksittäisiä tapauksia. Tavat ovat parantuneet henkilövaihdosten myötä.*

4. Versioidaanko tuote/asiakasversiot aina ennen asiakkaalle toimittamista?

   [Henkilö A, 2009]: *Kyllä, tuotteen (tuoterungon) osalta toimitaan näin. Asiakasprojekteissa käytännöt eivät ole vielä yhtä selviä.*

[Henkilö B, 2009]: *Versiointia ei aina tehdä. Esimerkiksi bugikorjauksia voidaan toimittaa "välistä", tekemättä versiointia. Tavoite on, että versiointi tehtäisiin aina riippumatta siitä kuinka pienestä asiasta on kyse.*

5. Onko ollut vaikeuksia tunnistaa käytettävä versio, kun asiakas on ilmoittanut virheestä?

   [Henkilö A, 2009]: *Tuotteen osalta versiointi tehdään nykyisin kurinalaisesti. Version tunnistamisen haasteet riippuvat asiakasprojektista. Jos versioinnit on tehty asiakasprojekteissa asianmukaisesti ennen toimituksia, versiot pystytään tunnistamaan jälkikäteen. Käytännössä asiakasprojekteissa on vielä haasteita tämän osalta.*

   [Henkilö B, 2009]: *Aiemmin on ollut vaikeaa tunnistaa käytettävä versio. Nykyisin tämä ei enää ole kovin suuri ongelma, koska asiakastoimitukset merkitään ("tägätään") versionhallintaan toimitushetkellä.*

Ohjelmistoprosessi

1. Tapahtuuko uusien ominaisuuksien/ominaisuusmuutosten hallinta tuotteessa (ja asiakasversioissa) tällä hetkellä selkeästi määritellyn prosessin avulla?

   [Henkilö A, 2009]: *Tuotteen osalta kyllä (prosessi on myös kuvattu), asiakasprojekteissa käytännöt vaihtelevat.*

   [Henkilö B, 2009]: *Lyhyesti kuvattuna prosessi toimii seuraavasti: määritellään toiminto pääpiirteittäin, toteutetaan ketterästi ja kierretään iteratiivisesti kehää kunnes tulokseen ollaan tyytyväisiä.*

2. Kuinka pitkään prosessia on käytetty?

   [Henkilö A, 2009]: *Prosessi on ollut jossain muodossa käytössä jo pitkään. Sitä on kehitetty matkan varrella.*

   [Henkilö B, 2009]: *Muunlaista prosessia ei tietääkseni ole käytetty.*

3. Onko prosessia muutettu matkan varrella? Osaatko sanoa millä tavalla?

   [Henkilö A, 2009]: *Vuoden 2009 alusta tuotteen kehittämisen kiinteäksi osaksi tulivat katselmointikäytännöt ja automaattiset buildit.*

   [Henkilö B, 2009]: *Prosessi on pysynyt pääpiirteittäin samana.*

4. Missä kohdissa ja kuinka usein tiedät/arvelet tapahtuvan poikkeamia prosessin noudattamisessa?

   [Henkilö A, 2009]: *Prosessioikomiset liittyvät dokumentointivaiheeseen, asioita ei tehdä niin kuin pitäisi. Online-dokumentointi on iso osa tuotetta ja sitä pitäisi päivittää aina, kun tuotteeseen tehdään muutoksia. Tekniseltä kirjoittajalta menee kohtuuttomasti aikaa selvittäessä, miten muutos vaikuttaa dokumentaatioon.*

   [Henkilö B, 2009]: *En osaa sanoa tarkasti, mutta arviolta jonkin verran.*

5. Onko prosessi mielestäsi tällä hetkellä toimiva vai pitäisikö sitä muuttaa joltain osin?

   (a) Mikä toimii tällä hetkellä hyvin?

   [Henkilö A, 2009]: *Oman tiimin sisällä prosessi on saatu hiottua toimivaksi. Ongelmana on sen vieminen projektitiimeihin. Erilaiset toimitavat aiheuttavat turhaa työtä.*

   [Henkilö B, 2009]: *Prosessi on joustava niin teknisesti kuin muutenkin.*

   (b) Mikä toimii tällä hetkellä huonosti?

   [Henkilö A, 2009]: *Jos tuotteen (tuoterungon) kehitysprosessiin tehdyn parannuksen jälkeen saa muut innostumaan asiasta, onnistuu valutus myös projektiryhmiin. Tämä vaatii kuitenkin paljon aikaa ja kärsivällisyyttä.*

[Henkilö B, 2009]: *Prosessi joustaa joskus liikaakin ja liian helposti.*

(c) Jos muuttaisit prosessia, mitä asioita muuttaisit?

[Henkilö A, 2009]: *Kiinnittäisin huomiota prosessin alkupäässä isojen muutosten tai uusien ominaisuuksien miettimiseen. Ennen toteutusta pitäisi miettiä kokonaisuutta paremmin. Alusta asti pitäisi saada mukaan myös testaaja ja tekninen kirjoittaja. Kaikki yhdessä voisivat miettiä ominaisuuden todellista tarvetta ja toteutustapaa.*

[Henkilö B, 2009]: *Lisäisin prosessiin hieman enemmän ryhtiä. Sooloilua tulisi ehkäistä ja ohjata enemmän tekemistä. Tekemisellä pitäisi olla enemmän vaatimuksia.*

(d) Miten ja miksi arvelisit muutosten parantavan tilannetta?

[Henkilö A, 2009]: *Varsinainen tarve saattaa unohtua tehdessä. Uuden ominaisuuden valmistuessa kukaan ei välttämättä osaa ottaa kantaa tai kertoa, toimiiko se täysin niin kuin pitäisi*

[Henkilö B, 2009]: *Saataisiin yhdenmukainen toimintatapa. Uusien henkilöiden hyödyntäminen projekteissa olisi helpompaa kun ei tarvitsisi opetella useita erilaisia tapoja tehdä asioita.*

6. Onko tuotteenhallinta tuoterungossa (tai asiakasversioissa) muuttunut ajan myötä?

   (a) Onko se ollut helpompaa/vaikeampaa aikaisemmin verrattuna nykytilanteeseen?

   [Henkilö A, 2009]: *Asiakaspiirteen ominaisuus vietiin ennen tuotteeseen yhdellä heitolla, kukaan ei testannut sitä. Nykyään käytössä koodin jäädytykset. Tavoitteena on, että kuukausi ennen julkaisua jäädytetään lähdekoodi, jonka jälkeen tehdään vain bugikorjauksia.*

   [Henkilö B, 2009]: *Nykyisin siihen kiinnitetään enemmän huomiota.*

(b) Mikä on toiminut aiemmin paremmin?

[Henkilö A, 2009]: *Ei mikään.*

[Henkilö B, 2009]: *Ei mikään.*

(c) Mikä on toiminut aiemmin huonommin?

[Henkilö A, 2009]: *Tämä käy ilmi ylemmästä vastauksesta.*

[Henkilö B, 2009]: *Tämä on kehittynyt ajan myötä. Nykyisin on helpompi kehittää laajoja kokonaisuuksia kuin 5 vuotta sitten. Nyt meillä on teknisesti osaavampaa porukkaa ja tekniikka on tutumpaa kuin ennen.*

(d) Arveletko tuotteenhallinnan helpottuvan tulevaisuudessa? Miksi?

[Henkilö A, 2009]: *Kyllä, toimivat käytännöt ovat vietävissä vähitellen myös asiakasprojekteihin.*

[Henkilö B, 2009]: *Kyllä, suuntana on formaalimpi malli, julkaisupolitiikan parannus on ensisijainen tie. Tuoterungon ominaisuuksia aiotaan myös paketoida eli tuotteistaa jatkossa nykyistä enemmän.*

7. Arvioita syistä

(a) Mitkä ulkopuoliset tekijät vaikuttavat onnistumiseen/epäonnistumiseen (esim. asiakkaan toimintatavat)?

[Henkilö A, 2009]: *Asiakkailla pitäisi olla kurinalainen testaus. Asiakkaan pitäisi pystyä hyväksymään testiin asennettu toimitus kokonaisuudessaan, eikä poimia "kirsikoita kakusta". Jos toimituksesta hyväksytään vain osa, kokonaisuutta ei kyetä hallitsemaan. Riippuen asiakkaiden resursseista jotkut asiakkaat testaavat lukuisilla resursseilla tehokkaasti ja hyvin. Pienet asiakkaat eivät täysin ymmärrä mitä on testaaminen eivätkä panosta siihen.*

[Henkilö B, 2009]: *Asiakkaat eivät osaa kertoa vaatimuksia siten, että ne olisivat yksiselitteisiä ja johdonmukaisia. Yksi vaatimus kumoaa toisen.*

(b) Mitkä oman yrityksen tekijät vaikuttavat onnistumiseen/epäonnistumiseen?

[Henkilö A, 2009]: *Kommunikaation lisääminen. Parannusten jalkauttaminen on iso ongelma. Jos puhutut muutokset saataisiin ajettua läpi, ongelmista ratkeaisi 90%. Tiimien ja projektipäälliköiden yläpuolella pitäisi olla taso, joka voisi ajaa yhteisesti asioita läpi. Pitkällä tähtäimellä tuote lähtee rönsyilemään, jos tehdään vain asiakaslähtöistä tuotekehitystä. Tuotekehitykseen pitäisi laittaa nykyistä enemmän paukkuja. Projektiryhmissä väki pyörittää tällä hetkellä toisinaan peukaloita, kun saman ajan voisi hyödyntää tuotekehitykseen.*

[Henkilö B, 2009]: *Onnistumisissa on parantamisen varaa. Yrityksessä on erityisen ammattitaitoista porukkaa, jotka voisivat oman työnsä ohella auttaa muitakin.*

(c) Mitkä oman tiimin tekijät vaikuttavat onnistumiseen/epäonnistumiseen?

[Henkilö A, 2009]: *Tiimin kokemus vaikuttaa onnistumiseen. Tuotekehitystiimissä henkilöillä on keskimäärin 8 vuoden kokemus tuotteen kehittämisestä. Jäsenet ovat erittäin kokeneita koodareita, melko kurinalaisia ja ylpeitä omasta tuotteestaan. Kokemus on myös haittapuoli, koska sen myötä käytössä on pinttyneitä toimintatapoja, joille ei ole perusteita. Tuotekehitystiimin jäseniä kysellään myös vähän väliä myynnin tukitehtäviin ja muihin tarpeisiin, mikä hankaloittaa laajojen kehityshankkeiden läpivientiä.*

[Henkilö B, 2009]: *Pitkään tiimissä työskennelleillä henkilöillä on paljon hiljaista tietoa, joka tulisi saada muidenkin käyttöön.*

(d) Mitkä yksilön tekijät vaikuttavat onnistumiseen/epäonnistumiseen?

[Henkilö A, 2009]: *Yksittäisillä henkilöillä on kova kiire ja tekemistä moneen suuntaan levällään. Tämä syö työmotivaatiota ja vähentää onnistumisia. Seurauksena on noidankehä. Tiimi on miettinyt omaa työtilaa, joka rauhoittaisi ympäristöä.*

[Henkilö B, 2009]: *Jokainen voi vaikuttaa lopputulokseen. Tuotetta kehittävä tiimi on aika pieni ja tekijät samoja. Yksi henkilö tekee pitkälti yhden palasen, työtä ei juurikaan tehdä pienryhmissä.*

8. Rinnakkaisten versioiden ylläpito ja resurssivaikutukset

   (a) Joudutaanko samaa ominaisuutta ylläpitämään sekä asiakasversiossa että tuotteessa? Miksi?

   [Henkilö A, 2009]: *Tuotteesta ylläpidetään useampaa versiota (asiakkaiden kanssa sovittu että kahta viimeisintä versiota ylläpidetään). Kuitenkin niinsanotut vuodenvaihdepaketit tehdään vanhempiinkin, jopa 10 vuotta vanhoihin asennuksiin. Niiden osalta muutosten tekeminen ole ole suoraviivaista vaan voidaan joutua tekemään hyvinkin erilaisia versiokohtaisia ratkaisuja.*

   [Henkilö B, 2009]: *On kokeiltu, ei tehdä mielellään. Rinnakkaisten versioiden ylläpito on erittäin työlästä ja aikaa vievää.*

Laadun valvonta

1. Kuvaile tuotteenhallinnassa käytettyjen ohjelmistoprosessien laadun valvontaa.

   [Henkilö A, 2009]: *Testaus on osana tekemistä. Tuotteen kouluttajaporukka testaa kaikki muutokset. Joskus muutoksen alkuperäisessä jira-kirjauksessa (tehtävänhallinnan tapahtuma) ei ole kerrottu, miten sitä pitäisi testata. Testaaja saattaa testata liian laajasti tai väärää asiaa.*

   [Henkilö B, 2009]: *Laadun valvontaa tehdään hyvin vähän ja se on lähinnä kehittäjien vastuulla. Automaattisia testejä on käytössä, mutta kehittäjät*

eivät korjaa aiheuttamiaan käännösten rikkoontumisia aina. Tarvittaisiin henkilö, jonka päätehtävänä on hoitaa laadun valvontaa. Käytetyt työkalut tukevat laadun valvontaa. Loppujen lopuksi laadun pitäisi lähteä tekijöistä, valvojan tehtävänä on katsoa perään, että homma toimii ja ohjata tarvittaessa.

Muuta

- Mitä muuta haluat kertoa?

  [Henkilö A, 2009]: *Kaikkien ominaisuuksien toteutus pitäisi tehdä yhdenmukaisella prosessilla niin tuotteen kuin asiakasprojektienkin osalta. Yksi vaihtoehto olisi tuotekehitystiimin hajottaminen eli tuotekehitystiimin jäsenet menisivät asiakasprojekteihin ohjaamaan tekemistä. Projektitiimeissä ei tiedetä tai haluta tietää tuotekehityksen asioista. Joka tapauksessa tuotekehityksen ja asiakasprojektitiimien pitäisi olla hyvin läheisissä tekemisissä keskenään. Tämä parantaisi tekemisen laatua. Tuotekehitys ja tuotteeseen liittyvät asiat saataisiin yhteen koottua yhdeksi kokonaisuudeksi.*

  [Henkilö B, 2009]: *Tuotteen tekemisen haasteet liittyvät siihen, että tuotteesta saataisiin riittävän monikäyttöinen erilaisiin asiakastarpeisiin. Aina kun tehdään tuotteeseen ominaisuuksia, pitäisi olla puoli tusinaa asiakastoteusta vertailukohteena. Kun näin ei ole, joudutaan turvautumaan mielikuvitukseen.*