

hyväksymispäivä

arvosana

arvostelija

Ohjelmistotekniikan pro gradu -tutkielma

Johannes Kuusela

Helsinki 20.11.2007

Pro Gradu

HELSINGIN YLIOPISTO

Tietojenkäsittelytieteen laitos

Tekijä — Författare — Author

Johannes Kuusela

Työn nimi — Arbetets titel — Title

Ohjelmistotekniikan pro gradu -tutkielma

Oppiaine — Läroämne — Subject

Työn laji — Arbetets art — Level

Pro Gradu

Aika — Datum — Month and year

20.11.2007

Sivumäärä — Sidoantal — Number of pages

15 sivua + 0 liitesivua

Tiivistelmä — Referat — Abstract

Ohjelmistotuoteperheet eli ohjelmaperheet ovat jatkuvasti kasvava tapa tuottaa ohjelmistoja. Ohjelmistotuoteperhettä perustettaessa voidaan lähteä jo olemassa olevasta tuotteesta tai tuotteista, jolloin niistä pitää tunnistaa ne osat tai ominaisuudet, jotka halutaan uuden tuoteperheen pohjaksi.

ACM Computing Classification System (CCS):

D.2.3 [Coding Tools and Techniques],

D.2.9 [Management],

D.2.13 [Reusable Software]

Avainsanat — Nyckelord — Keywords

Tuoteperheet, perinnejärjestelmät, uudelleenmallinnus

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Sisältö

| | |
|--|-----------|
| 1 Johdanto | 1 |
| 2 Ohjelmaerheet ja ohjelmaerhetuotanto | 3 |
| 3 Perinnejärjestelmät | 5 |
| 3.1 Perinnejärjestelmien synty | 5 |
| 4 Perinnejärjestelmien uudelleenmallinnus | 7 |
| 4.1 Uudelleenmallinnuksen aloittaminen | 7 |
| 4.2 Tekniikat | 9 |
| 5 Ohjelmaerhetuotantoon siirtyminen | 11 |
| Lähteet | 13 |

1 Johdanto

Ohjelmiston uudelleenkäytettävyys on nykyaikaisessa ohjelmistotuotannossa tärkeä kriteeri ohjelmiston laatua määritettäessä. Kiristynvä kilpailu ohjelmistoalalla johtaa tuotantoaikojen jatkuvaan lyhenemispaineeseen. Uudelleenkäytettävyys lisää uusien tuotteiden laatua lyhentämällä niiden tuottamiseen kuluva-aikaa.

Vaikka suuri osa ohjelma-perheiden perustamiseen liittyvistä julkaisuista rohkaisevat rakentamaan ohjelma-perhe olemassa olevien tuotteiden ominaisuuksille, tähän on tarjottu vain vähän johdonmukaisia keinoja citebayer. Voidaan kuitenkin löytää useita syitä, miksi on järkevää aloittaa ohjelma-perhetuotanto tällä tavalla [Bri90]:

- Ohjelmistokehittäjillä on jo vahva tuntemus ja käsitys tuotteesta.
- Ohjelma-perhetuotanto voi olla ohjelmistokehittäjille vierasta, ja raju muutos yhdistettynä puhtaalta pöydältä aloittamiseen saattaa johtaa projektin epäonnistumiseen.
- Pohjatuote on jo käytössä, joten sitä täytyy ylläpitää. Ylläpito voi tapahtua muun ohjelmistokehityksen rinnalla vain pienin lisäkustannuksin.
- Ohjelmistokehittäjät voivat oppia työn ohella. He tottuvat ohjelma-perhetuotannon käytäntöihin ja oppivat näkemään perinnejärjestelmän ohjelma-perheen näkökulmasta.
- Järjestelmään on mahdollisesti jo sijoitettu paljon aikaa ja resursseja.

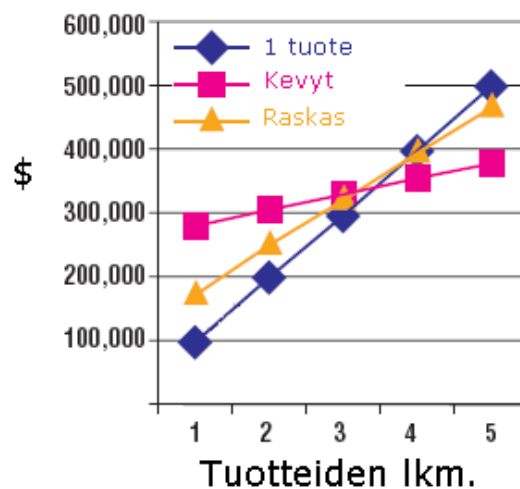
Tässä gradussa esitetään kattava katsaus perinnejärjestelmien käyttämiseen tuoteperheiden pohjana. Aihetta lähestytään esittelemällä ensin ohjelma-perheen perustamiseen sekä perinnejärjestelmien uudelleenmallinnukseen liittyviä ongelmia ja haasteita. Näitä ongelmia ovat esimerkiksi ohjelma-perhetuotantoon siirtymisestä koituvat kustannukset, uudelleenmallinnuksen taloudellinen kannattamattomuus ja usein myös kokemuksen puute organisaatiossa. Yhdistettynä epävarmuuteen lopputuloksesta tämä vähentää niin uudelleenmallinnuksen kuin ohjelma-perhetuotannonkin mielenkiintoa. Tarkoitus onkin pohtia aikaisemman kirjallisuuden pohjalta, kuinka suuri osa näistä ongelmista on yhteisiä ja kuinka suureen osaan voidaan löytää yhteinen ratkaisu ottamalla perinnejärjestelmä tuoteperheen perustaksi.

Ohjelma-perhetuotannon (software product line engineering) aloittaminen yrityksessä vaikuttaa tyypillisesti useisiin eri alueisiin yrityksen toiminnassa. Muutoksen suuruus riippuu siitä, valitaan-ko ohjelma-perhetuotantoon siirtymisessä *kevyt* (lightweight) vai *raskas* (heavyweight) strategia

[vdL02]. Molemmat strategiat ovat osoittautuneet menestyksekkäiksi yritysmaailmassa [MNJP02]. Molempien strategioiden puitteissa on mahdollista, että ohjelmaerhe perustetaan *perinnejärjestelmän* (legacy system) päälle. Näitä strategioita käsitellään tarkemmin luvussa 4.

2 Ohjelmaerheet ja ohjelmaerhetuotanto

Ohjelmistotuoteperhe eli *ohjelmaerhe* on joukko ohjelmistotuotteita, joilla on yhteisiä ominaisuuksia tietyn markkina-alueen tarpeisiin. Ne ovat jatkuvasti kasvava tapa tuottaa ohjelmistoja time-to-market -kilpailun ja voittomarginaalien pienenemisen ristipaineessa. Tuoteperheen ohjelmistoja tuotetaan systemaattisilla prosesseilla, jotka käyttävät hyväksi tuotteiden yhtäläisyyksiä mutta sallivat myös eroja tuotteiden välillä [ADH⁺00]. Käyttäjän näkökulmasta ne tarjoavat yhtenäisen ulkoasun lisäksi yhtenäistä toiminnallisuutta ja yhtenäisiä toimintatapoja ohjelmien välillä. Tämä helpottaa siirtymistä ohjelmasta toiseen ja alentaa kynnystä ottaa uusi ohjelma käyttöön. Ohjelmistoliiketoiminnan kannalta on järkevää julkaista ohjelmia, joissa on käyttäjille ennalta tuttuja piirteitä. Lisäksi hyvä tuoterunko mahdollistaa suuren uudelleenikäitettävyyden tuotteiden välillä, joka alentaa uusien tuotteiden valmistusaikaa ja -kustannuksia. Kuvassa 1 verrataan kevyttä ja raskasta siirtymistä ohjelmaerhetuotantoon tavallisen ohjelmistotuotannon kanssa, jossa kehitetään yksi tuote kerrallaan.

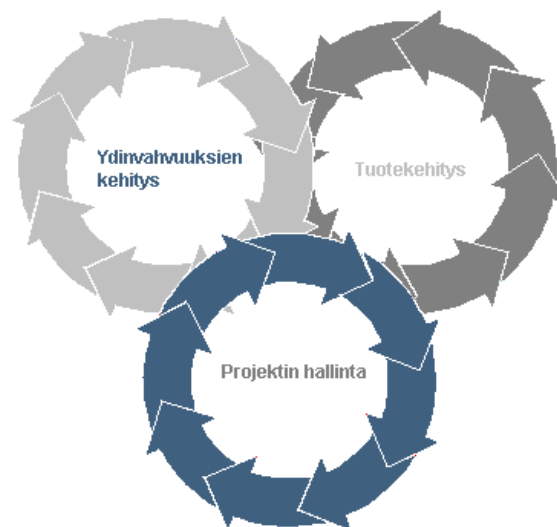


Kuva 1: Kevyen ja raskaan siirtymisen sekä perinteisen ohjelmistotuotannon kustannukset [MNJP02]

Ohjelmaerheet nojaavat vankasti uudelleenikäitettävyyteen. Niillä on runsaasti yhteneväisiä (esim. ulkoasu) ja yhteisiä osia (esim. ohjelmakoodia). Tyypillisesti uuden ohjelmaerheen jäsenen tuottaminen on huomattavasti nopeampaa kuin vastaavan ohjelman tuottaminen muuten. *Ohjelmaerhetuotannon* (software product line engineering) aloittaminen yrityksessä vaikuttaa tyypillisesti

useisiin eri alueisiin yrityksen toiminnassa. Muutoksen suuruus riippuu siitä, valitaanko ohjelma-perhetuotantoon siirtymisessä *kevyt* (lightweight) vai *raskas* (heavyweight) strategia [vdL02]. Molemmat strategiat ovat osoittautuneet menestyksekkäiksi yritysmaailmassa [MNJP02]. Molemmissa tavoissa on mahdollista, että ohjelma-perhe perustetaan *perinnejärjestelmän* (legacy system) päälle. Ohjelma-perhetuotantoon siirtymistä käsitellään tarkemmin luvussa 4.

Onnistuneen tuoteperhearkkitehtuurin päätavoite on tuotteiden välisten yhtäläisyyksien uudelleenkäyttäminen, kuitenkin säilyttäen muunneltavuus tuotteiden välillä. Nämä yhtäläisyydet voivat olla laadullisia asioita tai ohjelmakoodia. Ohjelma-perheen tuotteiden yhtäläisyyksistä valitaan koko tuoteperheelle sen *ydinvahvuudet* (core assets). Tuoteperheen ydinvahvuus on sellainen koko ohjelma-perhettä leimaava laadullinen tekijä, ominaisuus tai osa, jonka halutaan olevan perheen tuotteille yhteinen. Yksinkertaistettuna ohjelma-perhetuotanto voidaan jakaa ydinvahvuuksien etsimiseen tai kehittämiseen ja varsinaisten tuoteperheeseen kuuluvien ohjelmien tuottamiseen ydinvahvuuksia käyttäen. Tätä havainnollistaa kuva 2.



Kuva 2: Ohjelma-perhetuotannon osat.

3 Perinnejärjestelmät

Perinnejärjestelmille on kirjallisuudessa useita määritelmiä. Ytimekkäin lienee Oudshoomin esittämä määritelmä "perinnejärjestelmä on sellainen kriittinen järjestelmä, jota ei voida tehokkaalla tavalla ylläpitää"[OG05]. Osuvin puolestaan on ehkä Bennettin epävirallisempi määritelmä "perinnejärjestelmä on laaja ohjelmisto, jonka kanssa meidän on vaikea tulla toimeen mutta joka on meille elintärkeä"[Ben95]. Perinnejärjestelmien, perinnekodein ja vastaavien ohjelmistotuotannon sivutuotteiden määritelmät ja kuvaukset sisältävät eroistaan huolimatta aina tiettyjä yhteisiä piirteitä. Poikkeuksetta perinnejärjestelmät nähdään samaan aikaan sekä hankaluutena että hyötynä, sekä arvokkaana resurssina että resursseja kohtuuttomasti syöväinä. Tämä kaksoisidentiteetti on varsin poikkeuksellinen ja ulottuu kaikkialle, missä käytetään ja luotetaan tietokonejärjestelmiin; päivittäistä toimintaa eniten haittaava tekijä voi olla samalla toiminnan suurin mahdollistaja.

3.1 Perinnejärjestelmien synty

Lehmanin toisen lain mukaan järjestelmän kompleksisuus lisääntyy sen kasvaessa ellei sitä pyritä erikseen vähentämään [Leh96]. Kasvava kompleksisuus ei siis tarkoita, että järjestelmän kehittäjien ammattitaito olisi puutteellinen, tai että järjestelmän suunnittelu olisi laiminlyöty. Kun järjestelmää kehitetään uusien tai muuttuvien vaatimusten pohjalta, sen koko kasvaa. Tämä johtaa myös kompleksisuuden kasvamiseen, mikäli sitä ei erikseen estetä. Useimmiten perinnejärjestelmien syntyä edesauttaa se, että järjestelmällä on useita kehittäjiä, jotka työskentelevät toisistaan tietämättä (tyypillisesti järjestelmän elinkaaren eri vaiheissa). Vaikka yksittäiset kehittäjät tekisivätkin laadukasta ja hyvin suunniteltua työtä, heidän työtapansa ja ratkaisunsa ovat erilaisia, joka johtaa väistämättä järjestelmän kompleksisuuteen. Tämä sopii hyvin siihen ajatukseen, ettei kukaan kysyttäessä kerro ohjelmoivansa perinnejärjestelmiä, tai edes perinnekodeja.

Perinnejärjestelmillä on harvoin yksi tekijä, ja vielä harvemmin ne syntyvät yhdessä ohjelmistotuotantoprojektissa. Yhden projektin perinnejärjestelmien syntyminen on harvinaista ilmeisistä syistä:

- Yhden projektin sisällä tuotettavat järjestelmät ovat tyypillisesti liian hyvin suunniteltuja
- Projektin toteuttajilla on tyypillisesti liian hyvä käsitys järjestelmästä
- Projektien verrattain lyhyt kesto ei tyypillisesti mahdollista informaation katoamista riittä-

vässä määrin

Todellinen perinnejärjestelmä syntyy vuosien saatossa, useiden samaa järjestelmää muokkaavien projektien kuluessa. Projektit voivat olla pieniä tai suuria, samoin muokkaukset järjestelmään. Olennaista on näiden muokkausten tekijöillä oleva tieto. Jotta perinnejärjestelmä voisi syntyä, tietoa täytyy kadottaa (tai sitä täytyy menettää) joltain kautta. Yleisin tiedon katoamistapa on "luonnolliseksi poistumaksi" kutsuttu työntekijöiden siirtyminen uusiin työpaikkoihin. Tiedon katoamisen seurauksena järjestelmässä on laajoja osia, joiden toimintaa kukaan nykyisistä järjestelmän käyttäjistä tai kehittäjistä ei tunne. Hopealuodiksi perinnejärjestelmien syntymistä vastaan mainitaan usein dokumentaatio. Dokumentaatio on kuitenkin kaikissa projekteissa ensimmäinen karsittava aikataulun kiristyessä. Tämän seurauksena dokumentaatiota tehdään retroaktiivisesti ja viime hetkellä tiedon ollessa vielä saatavilla so. työntekijän ajatusten ollessa jo seuraavassa työpaikassa, eläkkeellä tai lomalla. Vaikka nykyään osataan jo panostaa tiedon säilyttämiseen työvoiman siirtyessä, on vastuu tiedon siirtämisestä eteenpäin aina motivaatioltaan kyseenalaisella poistuvalla työntekijällä. Hänen halukkuutensa ja aktiivisuutensa jakaa tietonsa järjestelmästä muovaa ja määrittelee voimakkaasti sen tulevaisuutta. Perinnejärjestelmä alkaa muodostua, kun tarpeeksi tietoa on kadonnut ja puutteellisin tiedoin varustetut henkilöt joutuvat työskentelemään järjestelmän kanssa.

Perinnejärjestelmän syntyminen on seurausta paitsi tiedon puutteesta, myös olemassa olevan tiedon soveltamisesta väärin. Usein ainoa dokumentaatio järjestelmästä on lähdekoodi, joka sekin voi olla vaikeaselkoista tai täysin käsittämätöntä muille kuin alkuperäiselle tekijälle. Lähdekoodin yksityiskohdat on helppo ymmärtää väärin puhumattakaan suuremmasta kokonaisuudesta tai järjestelmän takana olevasta toimintalogiikasta.

4 Perinnejärjestelmien uudelleenmallinnus

"To face tomorrow with the thought of using the methods of yesterday is to envision life at a standstill-James Bell

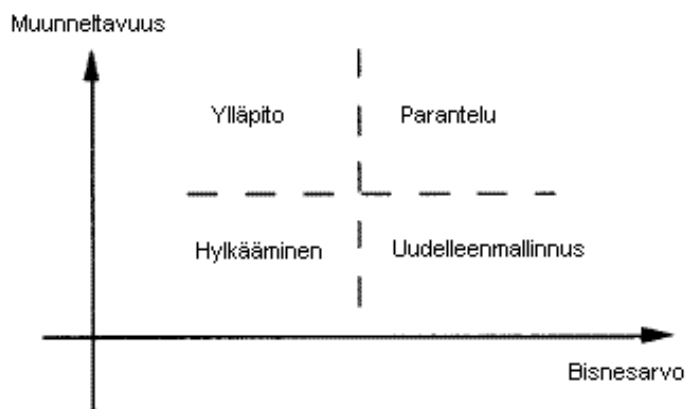
Ohjelmiston *uudelleenmallinnus* (re-engineering) on tapa pidentää ohjelman elinikää. Koska perinnejärjestelmät ovat usein kasvaneet ulos alkuperäisestä tarkoituksestaan niin tilassa kuin ajasakin, niiden uudelleenmallinnukseen on aina painetta. Uudelleenmallinnusprojektit eivät kuitenkaan ole tarpeellisuudestaan huolimatta suosittuja yrityksissä. Perinnejärjestelmän uudelleenmallinnus nähdään yleensä jossain määrin tarpeettomana toimintana, koska järjestelmä kaikesta huolimatta toimii. Lisäksi uudelleenmallinnusprojektit eivät yleensä tuota mitään helposti mitattavaa lisäarvoa yritykselle, ja vievät resursseja muulta liiketoiminnalta. Uudelleenmallinnusprojektin hyväksymiskynnys ylitetään, kun järjestelmän arvioidut ylläpitokustannukset ylittävät projektin arvioidut kustannukset. Molemmat näistä kustannuksista, mutta erityisesti uudelleenmallinnuksesta koituvat kulut arvioidaan usein väärin [OG05]. Weide, Heym ja Hollingsworth väittävät jopa, että perinnejärjestelmän uudelleenmallinnus on projekti, joka ei voi olla kustannustehokas johtuen sen luonteesta [WHH95].

Ohjelmaperheen perustaminen perinnejärjestelmän pohjalta on käytännössä uudelleenmallinnusprojekti. Perinnejärjestelmän elinikää halutaan pidentää ja toisaalta halutaan toimiva tuoterunko. Kyse on siis perinnejärjestelmän ominaisuuksien jalostaminen tuoteperheen tarpeisiin sopivaksi. Tätä käsitellään seuraavassa luvussa ohjelmaperhetuotantoon siirtymisen yhteydessä.

4.1 Uudelleenmallinnuksen aloittaminen

Perinnejärjestelmien uudelleenmallinnukseen, kuten kaikkiin ohjelmistotuotantoprojekteihin, liittyy riskejä. Yleensä suurin riski on luonnollisesti projektin epäonnistuminen joka tarkoittaa, että järjestelmää ei onnistuta toteuttamaan joko siinä laajuudessa kuin haluttiin tai ollenkaan. Uudelleenmallinnusprojekteissa kyse on kuitenkin aina olemassa olevan järjestelmän korvaamisesta uudella. Tämä pienentää projektin epäonnistumisen haittavaikutuksia, koska vanha järjestelmä on edelleen käytettävissä. Uudelleenmallinnukselle on olemassa myös vaihtoehtoja, joissa epäonnistumisen riski on pienempi. Oikean vaihtoehdon valinta riippuu järjestelmän bisnesarvon ja muunneltavuuden suhteesta kuva 3.

Uudelleenmallinnusprojekteissa vakavampia riskejä kuin projektin epäonnistuminen edellä mai-



Kuva 3: Perinnejärjestelmän päätösmatriisi[JL91].

nitulla tavalla onkin vääränlaisen järjestelmän luominen vanhan korvaajaksi. Riskianalyysit epäonnistuvat usein perinnejärjestelmien mallinnusprojekteissa juuri tästä syystä. Koska korvattava järjestelmä on ollut olemassa kauan ja on liike- tai muun toiminnan ytimessä, sen vaatimusten katsotaan olevan hyvin tunnetut ja vähemmän alttiit muutokselle kuin tavallisessa ohjelmistotuotantoprojektissa. Tämä oletus pitää kuitenkin paikkansa vain jos uudelleenmallinnusta tekevä ryhmä tuntee perinnejärjestelmän hyvin ja osaa käyttää tätä tietoa epävarmojen vaatimusten aiheuttamien riskien minimointiin [Ado96]. Toinen yleinen oletus on, että perinnejärjestelmässä käytettyjä ratkaisuja tai peräti ohjelmakoodia voidaan käyttää hyväksi uudelleenmallinnuksessa. Tämä on epätodennäköistä, sillä vaikka uudessa järjestelmässä päädyttäisiinkin käyttämään samoja ratkaisumalleja kuin perinnejärjestelmässä, on epätodennäköistä, että niiden toteuttamiseksi käytetään täsmälleen samaa tekniikkaa.

Perinnejärjestelmän uudelleenmallinnusta ei voi aloittaa spontaanisti tyhjältä pöydältä. Ennen projektia tulee kartoittaa järjestelmän kaikki olennaiset (muutoksen alla olevat tai muutoksen muuten koskettamat) osat. Tähän arviointiin on olemassa formaaleja menetelmiä, kuten RENAISSANCE [ESP97] ja uudelleenmallinnuksen suunnittelusta ja riskianalyysistä on kirjoitettu jonkin verran [Sne95], [Sne99]. Olennaista on kuitenkin muistaa, että perinnejärjestelmien uudelleenmallinnusprojektit ovat aina yksilöllisiä ja ne täytyy räätälöidä järjestelmän ja sen toimintaympäristön mukaan. Tämä johtuu ennen kaikkea järjestelmien tärkeästä roolista haltijansa toiminnan ytimessä. Ransom, Sommerville ja Warren tarjoavat joitakin kysymyksiä, joihin tulee vastata projektia suunniteltaessa [JR98]:

- *Onko järjestelmä kriittinen organisaatiolle, jossa se toimii?* Perinnejärjestelmien tapauksessa tämä on perustilanne.
- *Mitkä ovat organisaation liiketoiminnalliset päämäärät?* Jotta projektiryhmä voi arvioida perinnejärjestelmää, sen on tunnettava järjestelmää käyttävän organisaation tavoitteet.
- *Mitkä ovat järjestelmän kehityksen päämäärät?* Mitä järjestelmältä odotetaan tulevaisuudessa, ja miten näihin odotuksiin voidaan vastata.
- *Mikä on järjestelmän odotettu elinaika?* Mikäli järjestelmän käyttämä laitteisto tai tukijärjestelmät ovat vaarassa vanhentua käyttökelvottomiksi, tämä tulee ottaa huomioon uudelleenmallinnuksessa. Toisaalta mikäli järjestelmältä ei edes odoteta pitkäikäisyyttä, voi koko uudelleenmallinnuksen tarpeellisuus olla epävarmaa.
- *Mikä on järjestelmän tekninen tila?* Onko järjestelmä teknisesti heikotasoinen, vaikeasti ymmärrettävä tai epävarmasti toimiva.
- *Kuinka valmis järjestelmää käyttävä organisaatio on muutokseen?* Organisaation suhtautuminen muutokseen on tärkeä tekijä uudelleenmallinnusprojektien onnistumisessa.
- *Millaiset resurssit organisaatiolla on?* Uudelleenmallinnusprojektin kannalta olennaisia resursseja ovat esimerkiksi henkilöstön määrä, henkilöstön tekninen osaaminen, työprosessien kypsyys ja työkalujen laatu.

4.2 Tekniikat

Perinnejärjestelmien uudelleenmallinnuksesta on kirjoitettu runsaasti ja sen eri vaiheisiin on tarjolla lukuisia tekniikoita. Tärkeä osa näistä ovat tekniikat jotka auttavat paikallistamaan ja siirtämään perinnejärjestelmän toiminnallisuuden uuteen järjestelmään. Perinnejärjestelmän ominaisuuksien selvittäminen voi tuntua implisiittisesti helpolta, onhan järjestelmä osa ydintoimintoja ja jatkuvas-
sa käytössä. Kun ominaisuuksia sitten aletaan etsimään, huomataan usein tehtävän olevan todella vaikea ilman järjestelmällisiä tai jopa formaaleja työkaluja.

On epärealistista olettaa, että perinnejärjestelmä voidaan suoraan korvata uudella, korvaavalla järjestelmällä. Käytännössä siirtyminen tapahtuu vähän kerrallaan. Vanhasta järjestelmästä korvataan pieni osa kerrallaan ja uuteen järjestelmään siirrytään näin pienissä harppauksissa. Tästä on muutamia selviä etuja: Vähän kerrallaan tapahtuva siirtyminen säästää resursseja, koska se ei ole yhtä

raskasta ylläpidon ja riskien hallinnan näkökulmasta. Lisäksi se mahdollistaa uuden järjestelmän soveltuvuuden todentamisen tuotantoympäristössä jo ennen sen lopullista valmistumista.

Ominaisuuksien kaivelu (*feature mining*) on järjestelmän olennaisten ominaisuuksien etsimistä järjestelmästä. Ominaisuudella tarkoitetaan yleensä sellaista kokonaisuutta, joka toteuttaa jonkin järjestelmälle olemassaolevan vaatimuksen. Ominaisuuksien löytäminen vaatii aina jonkin verran järjestelmän takaisinmallinnusta (*reverse engineering*). Takaisinmallinnus on yläkäsite suurelle joukolle toimenpiteitä, joiden päämäärä on järjestelmän komponenttien tunnistaminen ja niiden yhteistoiminnan ymmärtäminen [WC94]. Paitsi, että se on kriittinen osa perinnejärjestelmien uudelleenmallinnusta, sen avulla voidaan lisätä niiden ylläpidettävyyttä mikäli uudelleenmallinnus ei tule kyseeseen.

Kapselointi on perinnejärjestelmän osien uudelleenkäytössä korvaamaton tekniikka. Kapseloinnilla tarkoitetaan jonkin järjestelmän komponentin piilottamista uuden rajapinnan sisään. Tällä menettelyllä komponentin toiminnot voidaan hyödyntää uuteen järjestelmään sopivan rajapinnan kautta. Kapselointityökaluja on olemassa sekä lähdekoodi- että esimerkiksi tietolähdekomponenteille (mm. [Sne96], [LFGP97], [RSI97]).

Coppit ja Sullivan ehdottavat, että lähdekoodin kapseloinnissa lähdettäisiin kolmesta perusoletuksesta. Ensimmäinen on, että on olemassa kaksi osaa ohjelmakoodia; ydin (*core code*) ja ylärakenne (*superstructure*). Ydinosaa on se osa koodista, jossa on varsinainen toiminnallisuus, ja ylärakenne on se koodi, joka integroi ydinosan järjestelmään. Se sisältää rajapinnat tiedon syöttämiseen ja vastaanottamiseen ja niin edelleen. Toinen oletus on, että ydinosan ohjelmakoodi on paitsi kriittistä muulle toiminnalle, myös päätyntä luonnollisen evoluution kautta tilaan, jossa se on vaikeasti ymmärrettävää dokumentaation puutteen ja kompleksisuuden takia. Toisin sanoen kyseessä on perinnekoodi. Kolmas oletus on, että tämän perinnekoodin säilyttämiseen nykyisessä tilassaan ja liittäminen uuteen järjestelmään olemassa olevien rajapintojensa kanssa ei ole mielekästä [CS].

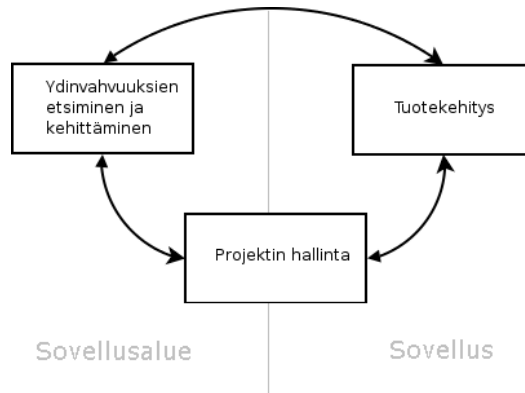
5 Ohjelmaperhetuotantoon siirtyminen

Ohjelmaperhetuotantoon siirryttäessä on tunnistettava tuoteperheen *ydinvahvuudet*. Nämä ovat sellaisia tuoteperheen läpileikkaavia ominaisuuksia, jotka ovat yhteisiä useammalle tuoteperheen jäsenelle (ja näinollen kuuluvat tuoterunkoon). Tuoteperheen ydinvahvuus on sellainen laadullinen tekijä, ominaisuus tai osa, jonka halutaan olevan perheen tuotteille yhteinen. Se on koko tuoteperhettä leimaava tekijä. Ohjelmaperheen ydinvahvuuksien etsimiseen on kaksi lähestymistapaa. Lähestymistavan valinta riippuu siitä, onko tuoteperheen pohjana olemassa olevia tuotteita vai perustetaanko tuoteperhe tyhjästä. Näitä lähestymistapoja on kutsuttu myös *kevyeksi* ja *raskaaksi* tavaksi perustaa ohjelmaperhe [MNJP02] [vdL02].

Lähestymistapojen välillä on useita eroja, mutta pohjimmiltaan kyse on alkupanostuksen suuruudesta. Raskaassa siirtymisessä koko ohjelmistotuotantoprosessi valjastetaan heti kättelyssä ohjelmaperhetuotantoon ja tuoterunko suunnitellaan huolellisesti palvelemaan kaikkia ennalta nähtäviä variaation tarpeita, kun taas kevyessä ohjelmaperhetuotannossa tuoterungon tuki variaatiolle lisätään vasta tarpeen ilmetessä.

Ohjelmaperheen ydinvahvuuksien etsiminen ja itse tuotekehitys ovat yhtä, iteratiivista prosessia. Tuotteita kehitetään ydinvahvuuksien pohjalta, mutta usein uusia ydinvahvuuksia kehittyä tai löydetään tuotekehityksen kautta (kuva 4). Kun ohjelmaperhettä perustetaan perinnejärjestelmän pohjalta, ydinvahvuuksien etsiminen ja kehittäminen on luonteeltaan erilaista kuin tavallisesti. Siinä missä ydinvahvuudet normaalisti löydetäisiin vaatimusmäärittelyn avulla, ne löydetään perinnejärjestelmän vaatimusten avulla. Toki näiden vaatimusten löytäminen saattaa myös edellyttää vaatimusmäärittelyprosessia. Ydinvahvuuksien kehittäminen tapahtuu rinnakkain tuotekehityksen kanssa kuten normaalistikin, mutta niiden pohjana käytetään perinnejärjestelmän osia. Tästä koi-tuu ohjelmistokehitykselle niin hyötyjä kuin haittojakin.

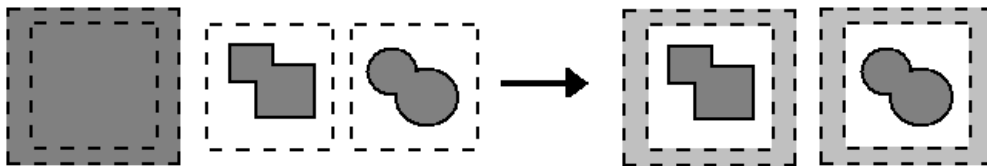
Jotta perinnejärjestelmä voisi toimia osana ohjelmaperhettä, saati sen pohjana, on siitä eroteltava ne ominaisuudet, jotka ovat käyttökelpoisia ohjelmaperheessä. Tuoreessa ja hyvin dokumentoidussa järjestelmässä tämä olisi helppoa. Ominaisuudet löytyisivät vaatimusanalyysin dokumentaatiosta ja niiden tekniset toteutukset olisivat helposti paikallistettavissa jäljitettävyyden ansiosta. Valitettavasti perinnejärjestelmissä tämä pitää harvoin paikkansa. Järjestelmän evoluution myötä dokumentaatio jää jälkeen todellisuudesta, ohjelmakoodia siirretään, muutetaan ja järjestellään uudelleen. Vaatimuksia tulee lisää, katoaa ja muuttuu. Menetelmiä ominaisuuksien etsimiseen perinnejärjestelmistä on ehdotettu useita [WS95]. Jotain toiminnan vaikeusasteesta kertoo se, että



Kuva 4: Ydinvahvuudet ja tuotekehitys.

näitä menetelmiä verrataan tavalliseen tekstihakuun [WBP⁺03].

Yksi ongelma on olemassa olevien tuotteiden jalostaminen ohjelmaperheen pohjaksi on rajapintojen yhteensopimattomuus. David Parnas esitti, että ohjelmaperheen jäsenten samankaltaisuudet olisivat niiden näkyvä osa eli rajapinnat, kun taas eroavaisuudet olisivat piilossa [DLP84]. Miksei sama toimisi myös toisin päin; ohjelmaperheen perustaksi valittujen tuotteiden erilaiset rajapinnat voidaan piilottaa tuoteperheen rajapinnan taakse (kuva 5). Toisaalta on täysin mahdollista, että jonkin tuotteen ottaminen ohjelmaperheen perustaksi ei ole mielekästä johtuen sen arkkitehtuurin puutteellisuudesta.



Kuva 5: Eroavaisuuksien piilottaminen rajapinnalla.

Lähteet

- ADH⁺00 Ardis, M., Daley, N., Hoffman, D., Siy, H. ja Weiss, D., Software product lines: a case study. *SoftwarePractice and Experience*, 30,7(2000), sivut 825–847.
- Ado96 Adolph, W. S., Cash cow in the tar pit: Reengineering a legacy system. *IEEE Software*, 13,3(1996), sivut 41–47.
- Ben95 Bennett, K., Legacy systems: Coping with success. *IEEE Software*, 12,1(1995), sivut 19–23.
- Bri90 Britcher, R. N., Re-engineering software: a case study. *IBM Syst. J.*, 29,4(1990), sivut 551–567.
- CS Coppit, D. W. ja Sullivan, K. J., Experiences 'unwrapping' two legacy systems for reuse. URL citeseer.ist.psu.edu/379517.html.
- DLP84 D. L. Parnas, P. C. Clements, D. M. W., The modular structure of complex systems. *ICSE '84: Proceedings of the 7th international conference on Software engineering*. IEEE Press, 1984, Orlando, Florida, United States, sivut 408–417.
- ESP97 ESPRIT, The renaissance method, 1997. URL <http://tinyurl.com/3cp8nz>.
- JL91 Jacobson, I. ja Lindström, F., Reengineering of old systems to an object-oriented architecture. *OOPSLA '91: Conference proceedings on Object-oriented programming systems, languages, and applications*, New York, NY, USA, October 1991, ACM, sivut 340–350.
- JR98 J. Ransom, I. Somerville, I. W., A method for assessing legacy systems for evolution. *Proceedings of the Second Euromicro Conference on Software Maintenance and Reengineering*, March 1998, Florence, Italy, sivut 128–134.
- Leh96 Lehman, M. M., Laws of software evolution revisited. *European Workshop on Software Process Technology*, Nancy, France, October 1996, sivut 108–124.
- LFGP97 Lucca, G. A. D., Fasolino, A. R., Guerra, P. ja Petruzzelli, S., Migrating legacy systems towards object-oriented platforms. *ICSM '97: Proceedings of the International*

- Conference on Software Maintenance*, Washington, DC, USA, October 1997, IEEE Computer Society, sivut 122–129.
- MNJP02 McGregor, J., Northrop, L., Jarrad, S. ja Pohl, K., Initiating software product lines. *IEEE Software*, 19,4(2002), sivut 24–27.
- OG05 Oudshoorn, M. ja Graham, M., Reengineering software: A case study. *SOMET 05: The 4th International Conference on Software Methodologies, Tools and Techniques*, September 2005, Tokyo, Japan.
- PMC06 P. M. Clements, L. M. N., A framework for software product line practice, 2006. URL <http://www.sei.cmu.edu/productlines/framework.html>.
- RSli97 Roth, M., Schwarz, P., It, S. ja it, W., A wrapper architecture for legacy data sources. *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases*. Morgan Kaufmann, August 1997, Athens, Greece, URL citeseer.ist.psu.edu/roth97wrapper.html.
- Sne95 Sneed, H. M., Planning the re-engineering of legacy systems. *IEEE Software*, 12, sivut 24–34.
- Sne96 Sneed, H. M., Encapsulating legacy software for use in client/server systems. *WCRE '96: Proceedings of the 3rd Working Conference on Reverse Engineering (WCRE '96)*, Washington, DC, USA, November 1996, IEEE Computer Society, sivu 104.
- Sne99 Sneed, H. M., Risks involved in reengineering projects. *Proceedings of the Sixth Working Conference on Reverse Engineering*, October 1999, Atlanta, GA, USA, sivut 204–211.
- vdL02 van der Linden, F., Software product families in europe: The esaps & café projects. *IEEE Software*, 19,4(2002), sivut 41–49.
- WBP+03 Wilde, N., Buckellew, M., Page, H., Rajlich, V. ja Pounds, L., A comparison of methods for locating features in legacy software. *J. Syst. Softw.*, 65,2(2003), sivut 105–114.
- WC94 Waters, R. G. ja Chikofsky, E., Reverse engineering: progress along many dimensions. *Commun. ACM*, 37,5(1994), sivut 22–25.

- WHH95 Weide, B. W., Heym, W. D. ja Hollingsworth, J. E., Reverse engineering of legacy code exposed. *ICSE '95: Proceedings of the 17th international conference on Software engineering*, New York, NY, USA, April 1995, ACM, sivut 327–331.
- WS95 Wilde, N. ja Scully, M. C., Software reconnaissance: mapping program features to code. *Journal of Software Maintenance*, 7,1(1995), sivut 49–62.