Juhani Haikonen

Olioperustainen ohjelmistokehitys – määrittelyn, suunnittelun ja toteutuksen väliset suhteet

Tietojärjestelmätieteen
kandidaatintutkielma
14.6.2004

Jyväskylän yliopisto
Tietojenkäsittelytieteiden laitos
Jyväskylä

TIIVISTELMÄ

Haikonen, Juhani Mikael

Olioperustainen ohjelmistokehitys – määrittelyn, suunnittelun ja toteutuksen väliset suhteet / Juhani Haikonen

Jyväskylä: Jyväskylän yliopisto 2004

25 s.

Kandidaatintutkielma

Olioperustaisuus on eräs tärkeä lähestymistapa ohjelmistotuotantoon. Olioperustaisuus on niin sanottu olioparadigma, jossa reaalimaailman katsotaan koostuvan olioista.

Olioperustaisesta ohjelmistokehityksestä voidaan tunnistaa ja luokitella toisistaan erotettavia vaiheita. Tässä tutkielmassa perehdytään olioperustaisen ohjelmistokehityksen kolmeen keskeiseen vaiheeseen: oliomäärittelyyn, oliosuunnitteluun ja olio-ohjelmointiin. Vaikka kyseisten vaiheiden lisäksi ohjelmistokehityksen prosessista on tunnistettavissa myös muita vaiheita, ei niitä nähdä yleisesti edellä mainittuja vaiheita tärkeämmiksi. Määrittely-, suunnittelu- ja ohjelmointivaiheessa otetaan eniten kantaa rakennettavien ohjelmien oliopiirteisiin.

Tutkielmassa keskitytään erityisesti näiden kolmen vaiheen välisiin suhteisiin ja suhteiden erityisominaisuuksiin. Tutkielman tärkeimpänä tehtävänä on tuoda lukijalle tietoa siitä, missä laajuudessa edeltävien vaiheiden tulokset voidaan hyväksikäyttää seuraavissa vaiheissa – eli miten vaiheet ovat integroituneet toisiinsa.

AVAINSANAT: Olioperustainen, ohjelmistokehitys, olioanalyysi, oliomäärittely, olio-ohjelmointi.

SISÄLLYSLUETTELO

1 JOHDANTO

Olioperustainen ohjelmistokehitys on ollut eräs tärkeä ohjelmistotuotannon lähestymistapa jo 1980-luvun lopulta asti. Kuitenkin tämä tapa kehittää ohjelmistoja on syntynyt epäloogisessa järjestyksessä, sillä se sai alkunsa toteutustavan synnystä 1960-luvulla, ja vasta myöhemmin 1980-luvulla kehitettiin kyseistä toteutustapaa avustamaan määrittely ja suunnittelu. Vielä nykyäänkin olioperustaisen ohjelmistokehitys –prosessin eteneminen on aika ajoin ongelmallista, siksi onkin tärkeä selvittää sitä haittaavat seikat.

Tämän kandidaatintutkielman tehtävänä on selvittää, mitä oliomäärittely, oliosuunnittelu ja olio-ohjelmointi ovat sekä erityisesti, mitä ominaisuuksia kyseisten vaiheiden suhteet pitävät sisällään. Vaiheet ovat selkeästi tunnistettavissa ja eroteltavissa toisistaan olioperustaisessa ohjelmistokehityksessä.

Luvussa kaksi esitellään lukijalle, mitä olioperustaisuus käytännössä tarkoittaa. Lukija tutustutetaan itse tutkielman aihepiiriin eli olioperustaiseen ohjelmistokehitykseen. Tästä ohjelmistojen kehitystavasta esitellään muun muassa sen eri vaiheet ja selvitetään tutkielman avaintutkimuskohdat: oliomäärittely, oliosuunnittelu ja olio-ohjelmointi.

Luvussa kolme keskitytään tarkemmin edellä mainittuihin kolmeen olioperustaisen ohjelmistokehityksen keskeisiin vaiheisiin. Luvussa jokainen osavaihe esitellään lukijalle riittävällä tarkkuudella, jotta vaiheiden tarkoitukset ja tehtävät eivät jää epäselviksi. Asian konkretisoimiseksi on tutkielmaan liitetty yksinkertainen esimerkki vaiheiden toteutuksista (LIITE 1).

Neljännessä luvussa keskitytään tarkemmin näiden kolmen vaiheen välisiin suhteisiin ja tuodaan esille muun muassa suhteiden ongelma- ja kehityskohtia. Lisäksi luvussa tarkastellaan, kuinka laajasti seuraavissa vaiheissa voidaan käyttää hyväksi edeltävien vaiheiden tuloksia. Toisin sanoen luvussa selvitetään missä määrin vaiheet ovat integroituneet toisiinsa.

Tämän tutkielman saavutukset ja tulokset kirjataan yhteenveto-luvussa. Siinä pohditaan muun muassa, millaisia keinoja hyväksikäyttäen vaiheiden välisiä tuloksia voitaisiin yhtenäistää. Lisäksi luvussa esitellään vaiheiden integraatiota mahdollisesti parantavia toimenpiteitä.

Tämä tutkielma on kirjallisuuskatsaus, joten tutkielmassa monin paikoin viitataan ja luotetaan alan kirjallisuuteen. Käytetty kirjallisuus on koottu tutkielman loppuun.

2 OLIOPERUSTAISUUS

Tämän luvun tarkoituksena on kertoa lukijalle, mitä olioperustaisuus on ja miten se on syntynyt. Lisäksi lukija perehdytetään tämän kandidaatintutkielman aihepiiriin eli olioperustaiseen ohjelmistokehitykseen. Tarkoituksena on selvittää tämän tärkeäksi muodostuneen ohjelmistokehitystavan olemus ja historia.

2.1 Mitä olioperustaisuus on?

Olioperustaisuuden voidaan sanoa saaneen alkunsa 1960-luvulla, kun eräs olio-ohjelmoinnin piirteitä sisältävä ohjelmointikieli, Simula, julkaistiin. Vasta myöhemmin 1970-luvulla alettiin käyttää itse termiä olioperustainen (*engl. object-oriented*) (Berard, 1993, 4).

Englanninkieliselle termille *object-oriented* esiintyy suomenkielisessä kirjallisuudessa monia eri käännöksiä, mutta ne kaikki tarkoittavat käytännössä samaa asiaa. Sanan *olioperustainen* lisäksi on suomenkielisessä kirjallisuudessa yleisesti käytetty seuraavia vastineita: oliosuuntautunut, oliolähestymistapa, oliokeskeinen ja oliopohjainen. Tässä tutkielmassa keskitytään käyttämään vain ja ainoastaan sanaa *olioperustainen*. Sana *olioperustainen* kuvaa kirjoittajan mielestä parhaiten asian konkreettisuutta.

Mitä sitten olioperustaisuus tarkoittaa? Sanansa mukaisesti se tarkoittaa olioihin perustuvaa/pohjautuvaa. Olioperustaisuus on näkökulma, jonka kautta asiat nähdään ja tunnistetaan olioina.

Itse olio (*engl. object*) käsitetään konseptiksi, jolla on identiteetti, tila ja käyttäytyminen. Berard (1993, 7) määrittelee olion luokan (*engl. class*) näkökulmasta. Luokka on abstrakti tietotyyppi (*engl. abstract data type*), josta olio instantioidaan (*engl. instantiate*). Toisin sanoen luokka määrittelee olion rakenteen ja käyttäytymisen, ja luokasta (yleensä ajonaikaisesti) luodaan instanssi eli varsinainen olio.

Olioperustaisuudessa on yleisesti nähty monia etuja. Olioperustaisuus on jopa nähty todellisena hopealuotina eli revolutiivisena ajatuksena, joka parantaa huomattavasti miltei kaikkea ohjelmiston kehitystoimintaan liittyen. Tällaiset väitteet ovat kuitenkin saaneet kovaakin kritiikkiä vastaansa, sillä niitä ei ole luotettavasti perusteltu.

Eräs tärkeä olioperustaisen lähestymistavan tuoma etu on se, että sen ajatusmaailma on reaalimaailmaa kuvaava. Näin siis sen käsitteet ja konseptit ovat ihmiselle luontaisia. Toisin sanoen, kun kaikki asiat nähdään olioina, niin ne myös sellaisina kuvataan.

Toinen tärkeä etu on se (Booch, 1994, 41), että olioperustaisuuden on nähty nostavan suunnittelun ja toteutuksen abstraktiotasoa, ainakin edeltäviin ohjelmiston kehitystapoihin nähden. Toisaalta myös olioparadigman on nähty pienentävän muun muassa ylläpitokustannuksia (Koskimies, 1997, 4).

Coad ym. (1991) ovat tuoneet esille vielä seuraavia perusteltuja ja konkreettisia etuja. Olioperustaisuus helpottaa projektin edistymisen seurantaa ja parantaa projektinhallintaa. Se myös yksinkertaistaa henkilöstön koulutusta ja parantaa työn tuottavuutta. Lisäksi LaLonde ym. (1998) tuovat selvästi esille sen, että olioparadigma kannustaa iteraatioon.

Olioperustaisuudelle on määritelty joukko mittareita, joita kutsutaan olio–ominaisuuksiksi (tai oliopiirteiksi). Booch (1994) on katsonut seuraavat olio-ominaisuudet tärkeimmiksi ominaisuuksiksi: abstrahointi, kapselointi (tiedon piilottaminen), hierarkia, modulariteetti, identiteetti, polymorfismi ja eheys. Muun muassa Haikala ym. (1998) ja Koskinen ym. (2001) ovat yhtyneet tähän näkemykseen. Kaikki edellä mainitut käsitteet kuvataan seuraavissa kappaleissa.

Abstrahointi (*engl. abstraction*) pyrkii yleistämään ja helpottamaan kokonaisuuksia. Se on paras keino ymmärtää monimutkaisia kokonaisuuksia (”*mitä korkeampi abstraktiotaso, sitä ymmärrettävämpi asia*”). Kapselointi (*engl.*

encapsulation) (tiedon piilotus) tarkoittaa tiedon ja käytöksen sitomista johonkin yhteen kokonaisuuteen (olioon). Modulariteetti (*engl. modularity*) puolestaan tarkoittaa kokonaisuuden jakamista osiin (esimerkiksi komponentteihin tai moduuleihin) (Booch, 1994).

Hierarkia (*engl. hierarchy*) muodostuu olioiden periytymisestä, ja se määrittelee olioiden suhteet toisiinsa nähden. Identiteetillä (*engl. identity*) kuvataan arvoa, jolla oliot voidaan tunnistaa ja erottaa toisistaan. Polymorfismilla (monimuotoisuus) (*engl. polymorphism*) kuvataan sitä, että olio ei ole sidonnainen pelkästään sille määriteltyyn tyyppiin (luokkaan). Eheys (*engl. integrity*) kuvaa sitä, että olion tiedot ovat yksinomaan sen käytettävissä (eli olion tietoihin ei pääse käsiksi kukaan muu) (Booch, 1994).

Olioperustaisuus on siis monessa mielessä hyödyllinen asia, mutta siihenkin liittyy joitakin ongelmallisia piirteitä. Haikala ym. (1998, 317) toteavat, että olioperustaisuudesta on tullut niin sanottu muotituote, jonka mainitseminen auttaa kaupallisessa toiminnassa. Tällöin ei esimerkiksi olioperustaisuuden soveltuvuudella kohdealueeseen saata olla mitään merkitystä. Nerson (1992) mainitsee, että oliojärjestelmillä on taipumus olla liian riippuvaisia rakenteellisista ratkaisuista, jolloin huonot rakenteelliset ratkaisut haittaavat koko järjestelmän toimintaa.

2.2 Olioperustainen ohjelmistokehitys

Olioperustainen ohjelmistokehitys on evolutiivisesti kehittynyt tapa rakentaa ohjelmistoja. Se sai alkunsa olio-ohjelmoinnin synnystä 1960-luvulla. Tämä on mielenkiintoinen lähtökohta, sillä näin olioperustaisen ohjelmistokehityksen kehittyminen eri vaiheita sisältäväksi prosessiksi selvästikin tapahtui takaperoisesti. Olio-ohjelmointia tukemaan syntyi oliosuunnittelu 1980-luvulla ja oliomäärittely kehittyi vasta 1980-luvun loppupuolella. Muut tunnetut olioperustaisen ohjelmistokehityksen vaiheet kehitettiin vasta 1980-luvun jälkeen.

Puhuttaessa olioperustaisesta ohjelmistokehityksestä on kuitenkin painotettava sitä, että perinteisesti eri olioperustaisen ohjelmistokehityksen vaiheilla ei ole ollut mitään tekemistä toistensa kanssa [Sekä Berard (1993, 3 – 5) ja Haikala ym. (1998, 316 – 317)]. Vasta viime aikoina on alettu nähdä olioperustainen ”ohjelmiston kehitys” todellisena yhtenäisenä olioperustaisena ohjelmistokehitys –prosessina.

Koskimies (1997, 4) on tiivistänyt asian hyvään määritelmään: ”Olioperustaisella ohjelmistokehityksellä tarkoitetaan systemaattista koko ohjelmiston elinkaaren käsittävää prosessia, joka perustuu kaikissa vaiheissaan ohjelmistojen kuvaamiseen joukkona keskinäisessä vuorovaikutuksessa olevia olioita. Toisin sanoen sovellusohjelma simuloi kyseiseen sovellukseen liittyvän maailman konkreettisia tai abstrakteja tapahtumia, jolloin ohjelmiston rakenne heijastaa tätä ajattelutapaa”.

Perinteisestä ”puhtaasta” olioperustaisesta ohjelmistokehitys –prosessista voidaan tunnistaa selvästi seuraavat toisistaan eroavat vaiheet (*KUVA 1*):

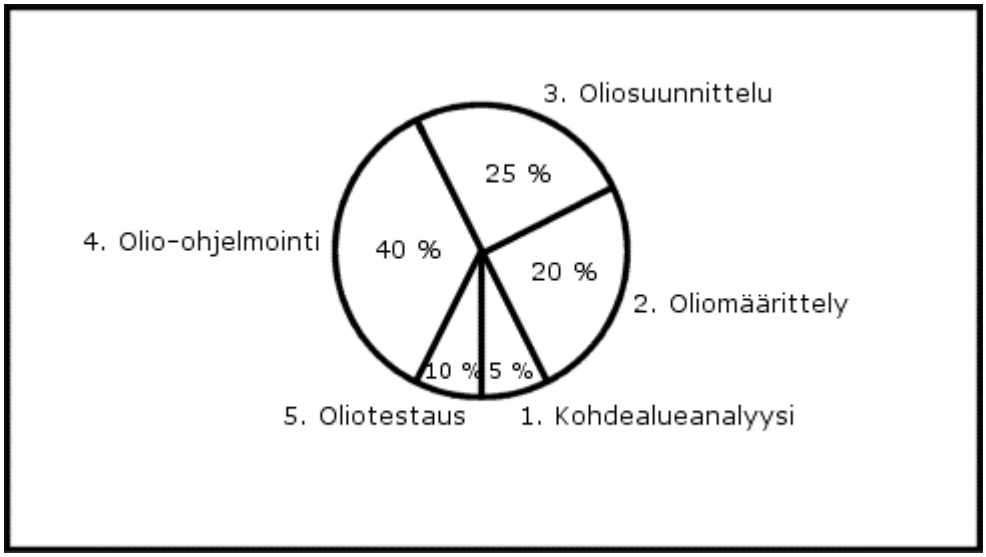
1. Olioperustaisessa kohdealueanalyysissä (*engl. object-oriented domain analysis*) pyritään identifioimaan uudelleenkäytettäviä ja toistuvia asioita olioissa (Berard, 1993, 185).

2. Oliomäärittelyssä (*engl. object-oriented analysis*) tunnistetaan järjestelmän tarpeelliset ja hyödylliset oliot.

3. Oliosuunnittelussa (*engl. object-oriented design*) määritellään kaikkien oliomäärittelyssä tunnistettujen olioiden väliset suhteet ja toiminnallisuudet.

4. Olio-ohjelmoinnissa (*engl. object-oriented programming*) suunniteltu ja mallinnettu rakenne toteutetaan toimivaksi ohjelmaksi.

5. Oliotestauksessa (*engl. object-oriented testing*) rakennetun ohjelman toiminta ja rakenne testataan.



KUVA 1: Esimerkki olioperustaisesta ohjelmistokehitysprosessista.

Perinteisesti edellä mainitut vaiheet nähdään enemmän tai vähemmän loogisesti toisiaan seuraavina vaiheina. Kuitenkin tämän perinteisessä muodossa mielletyn olioperustaisen ohjelmistokehityksen voi nähdä myös räätälöitynä (*engl. tailored*) prosessina. Tällaiset perinteisestä mallista poikkeavat kehitystavat on kuvattu prosessimalli- ja/tai menetelmäkohtaisesti. Näissä eri prosessimalleissa muun muassa vaiheiden järjestys ja rakenne voi olla kuvattu toisistaan poikkeavasti. Esimerkiksi V-prosessimallin ohjelmistokehitystavassa testaus nähdään jatkuvana tehtävänä koko prosessin ajan.

Koskimies (1997, 5) on tuonut esille sen, että olioperustaisuus ei sinällään nopeuta ohjelmistokehitystä, vaan jopa pikemminkin pidentää sitä. Näin siksi, että määrittely ja suunnitteluvaihe vievät perinteisiä menetelmiä (esimerkiksi rakenteellista analyysia ja suunnittelua) kauemmin aikaa. Koska olioparadigma kannustaa iteraatioon (LaLonde ym., 1998) (toistettavuuteen/uudelleenmääritettävyyteen), on selvää, että iteratiivinen prosessi on ajallisesti pidempi ”suoraviivaiseen” prosessiin nähden.

3 OLIOPERUSTAISEN OHJELMISTOKEHITYKSEN KOLME KESKEISTÄ VAIHETTA

Kuten jo aiemmin on todettu, olioperustainen ohjelmistokehitys voidaan nähdä vaiheisena prosessina. Oliomäärittely, oliosuunnittelu ja olio-ohjelmointi ovat tärkeimpiä vaiheita erityisesti oliopiirteisiin kantaa ottavina vaiheina, toisin sanoen ko. vaiheissa vaikutetaan eniten rakennettavien ohjelmien oliopiirteisiin. Tässä tutkielmassa ei perehdytä muihin edellisessä luvussa mainittuihin vaiheisiin, kuten olioperustaiseen kohdealueanalyysiin tai oliotestaukseen.

3.1 Yleistä

Tarkasteltaessa olioperustaista ohjelmistokehitystä puhtaana suoraviivaisena prosessina nähdään oliomäärittelyn tehtäväksi tarjota oliosuunnittelulle järjestelmän mallipohja, josta suunnittelu voidaan aloittaa. Oliosuunnittelun lopputulokset käytetään hyväksi olio-ohjelmointi –vaiheessa, jossa ohjelma toteutetaan olio-ohjelmointimenetelmin (Booch, 1994, 39-40).

Selvästikin sekä oliomäärittely että oliosuunnittelu keskittyvät järjestelmän ymmärtämiseen ja mallintamiseen, kun taas olio-ohjelmointi keskittyy tämän kyseisen järjestelmän toteutukseen (tuottamiseen ajettavaksi ohjelmaksi). Oliomäärittelyn ja oliosuunnittelun tärkeimpänä tuotoksena on dokumentaatio, kun taas olio-ohjelmoinnin tuloksena on dokumentaation lisäksi ohjelmakoodi.

Oliomäärittely ja oliosuunnittelu nojaavat tekstikuvausten lisäksi erilaisiin graafisiin esityksiin. Eri notaatiot ja notaatiokielet ottavat kantaa tarkasti näiden graafisten esitysten rakenteisiin ja sisältöihin. Esimerkiksi Koskinen ym. (2001, 35) mainitsevat OMT (*Object Modeling Technique*) –oliomenetelmän käsittävän kolme mallia (oliomalli, dynaaminen malli ja toiminnallinen malli), joilla kuvataan mallinnettavaa järjestelmää sen eri näkökulmista. Toisaalta kuitenkin Mathiassen ym. (2000) väittävät, että eri notaatiokielistä UML (*Unified Modelling Language*) –kielestä on nopeasti tulossa olioperustaisen ohjelmistokehityksen notaatiostandardi.

Perinteinen lähestymistapa ohjelmistokehityksessä perustuu niin sanottuun ylhäältä-alas–ositukseen (*engl. top-down*). Osituksen periaatteena on aloittaa mallinnus suurista kokonaisuuksista, ja kun nämä tunnistetut kokonaisuudet on saatu mallinnettua, niitä lähdetään tarkentamaan. Mallintamisen jälkeen suunnitelmat konkretisoidaan toteutusvaiheessa.

Tutkielman lopussa on liitteenä konkretisoitu esimerkki näiden kolmen vaiheen toteutuksista (LIITE 1). Liitteen esimerkki on erittäin yksinkertainen koska sen tarkoituksena on havainnollistaa asiaa.

3.2 Oliomäärittely

Oliomäärittely sai alkunsa 1980-luvun loppupuolella, kun tarve eritellä ohjelmistotuotteiden vaatimuksia oliomaisesti sai suurta huomiota. Tyytymättömyys muun muassa rakenteista analyysiä kohtaan kasvoi ja näin oliomäärittely kehitettiin vastaamaan järjestelmän ymmärtämystä olioperustaisessa ohjelmistokehityksessä (Berard, 1993, 4).

Coad ym. (1990, 3) esittelevät kirjassaan seitsemän syytä, miksi oliomäärittelyä ylipäätään tarvitaan. Tärkein syy on selvästi se, että oliomäärittelyn kautta voimme määritellä ja tunnistaa järjestelmän vaatimukset hyväksikäyttäen kolmea ihmiselle luonnollista abstraktiota – olioita (objekteja), luokittelua ja koostamista. Toinen huomattava syy on se, että oliomäärittely keskittyy vain ja ainoastaan kohdealueen ymmärtämiseen.

Tarkkaan ottaen oliomäärittelyssä etsitään olioluokat, niiden attribuutit ja operaatiot sekä luokkien väliset suhteet (Haikala ym., 1998, 327 – 328). LaLonde ym. (1993) täydentävät, että määrittely keskittyy vain olioiden analysointiin sekä käyttötapausten selvittämiseen olioiden vastuita painottaen.

Mathiassen ym. (2000) kiteyttävät tarkemmin oliomäärittelyn seuraavaan kontekstiin. Määrittelyvaiheessa olioita käytetään organisoimaan järjestelmän kontekstin ymmärrystä. Toisin sanoen olio on ilmentymän abstraktio

järjestelmän kontekstissa (kuten esimerkiksi asiakas). Olio kuvaa käyttäjän todellisuutta. Tämän lisäksi järjestelmän vaatimukset tunnistetaan olioiden avulla. Oliomäärittely tarkoittaa siis toimintaa, jossa jokin asia eristetään ja kuvataan.

Voidaankin siis todeta, että määrittelyvaiheessa mallinnettavaa järjestelmää tarkkaillaan ulkopuolisesti. Siinä pyritään tunnistamaan ja määrittelemään kaikki vaikuttavat oliot ja kokonaisuudet käyttäen hyväksi kohdealueen termistöä.

3.3 Oliosuunnittelu

Oliosuunnittelu sai alkunsa 1980-luvun alkupuolella, kun Booch ensimmäisenä esitti oliosuunnittelun käsitteen elinkaariprosessina, jossa määriteltiin ohjelmiston komponenttien vuorovaikutukset (Berard, 1993, 4). Tosin Haikala ym. (1998, 317) kertovat Boochin esitelleen oliosuunnittelun periaatteita jo 1970-luvun loppupuolella.

Mitä siis oliosuunnittelu tarkoittaa? Booch (1994, 39) esittää perinteiselle menetelmälleen seuraavanlaisen määritelmän: ”Oliosuunnittelu on suunnittelun keino sitoa oliokeskeisyyden hajottava prosessi ja notaatio, jolla kuvataan järjestelmän sekä loogisia ja fyysisiä että staattisia ja dynaamisia malleja”.

LaLonde ym. (1993) kuvaavat, että oliosuunnittelu pyrkii erittelemään oliomäärittelyssä tunnistettuja olioita lisää ottamalla huomioon muun muassa toteutusluokkia ja esittelemällä abstraktit luokat, perintähierarkiat, käyttötapaukset (suunnittelun kannalta) ja näkyvyydet. Mathiassen ym. (2000) lisäävät, että oliosuunnittelu on erityisesti rakentavaa toimintaa, jossa tunnistetut osat yhdistetään.

Voidaankin todeta, että suunnitteluvaiheessa mallinnettavaa järjestelmää tarkkaillaan sisäpuolelta, ja siinä pyritään jakamaan suunnitteluvaiheessa

tunnistettuja kokonaisuuksia edelleen osakokonaisuuksiin. Lisäksi näiden tunnistettujen olioiden/komponenttien rakenne, toiminta ja interaktio mallinnetaan sekä pyritään jaottelemaan niille eri tehtäväkokonaisuuksia.

De Champeaux ym. (1992) esittävät, että oliosuunnittelu on ikään kuin silta (*engl. bridge*) määrittelyn ja toteutuksen välillä. Suunnittelun luonne on erilainen olioperustaisessa ohjelmistokehityksessä, sillä kun määrittelyvaiheen malleilla ja toteutusvaiheen järjestelmällä on sama rakenne, on suunnittelun perusolemus muotoa muuttava (*engl. transformative*). Oliosuunnittelun tehtävänä on siis muuntaa kuvaavat määritelmät toteutettavaan muotoon. Muun muassa tämän vuoksi oliomäärittelyn ja oliosuunnittelun raja on häilyvä.

3.4 Olio-ohjelmointi

Olio-ohjelmointi sai alkunsa jo 1960-luvulla ensimmäisen olio-ohjelmointikielen synnystä. 1970-luvulla olio-ohjelmointi alkoi saavuttaa laajaa tuntemusta, mutta vasta 1980-luvun puolivälissä se kypsyi laajaan mittaansa (Berard, 1993, 5).

Booch (1994, 38) kuvaa olio-ohjelmoinnille seuraavan määritelmän: ”Olio-ohjelmointi on toteutuksen keino järjestää oliot yhteistyötä tekeväksi ohjelmaksi, joista jokainen olio on jonkin luokan instanssi ja sen kaikkia luokkia yhdistää periytyminen muodostaen toisiinsa kytkeytyvän hierarkian”.

Yksinkertaisemmin voidaan sanoa, että olio-ohjelmoinnin tehtävänä on toteuttaa jokin ohjelmakokonaisuus tietokoneella ajettavaksi ohjelmaksi. On kuitenkin hyvä huomioida, että olio-ohjelmointikielet ovat niin sanottuja korkean abstraktiotason ohjelmointikieliä, jotka on luotu matalampien tasojen (ohjelmointi-) kielten päällä toimiviksi. Näin ollen toteutettu ohjelma ei ole suoraan tietokoneella ajettavassa muodossaan ilman käännöstä (*engl. build*).

Olio-ohjelmointikielet jakautuvat oliokieliksi ja olioperustaisiksi kieliksi. Oliokieliä pidetään puhtaina (olio-) kielinä, joissa kaikki yleisesti määritellyt oliopiirteet ovat toteutettuina. Puhtaita oliokieliä ovat esimerkiksi Smalltalk ja

Eiffel. Olioperustaiset kielet ovat puolestaan kieliä, joissa on joitakin oliopiirteitä käytössä, mutta ei kuitenkaan niitä kaikkia. Yksi tällainen kieli on Ada83 (Haikala ym., 1998, 325).

Ensimmäiset olio-ohjelmointikielet eivät olleet puhtaita olio-ohjelmointikieliä, vaan niissä oli käytetty hyväksi joitakin nykyisiä olio-ominaisuuksia. Lisäksi on olemassa niin sanottuja hybridikieliä, jotka ovat sekä toisaalta puhtaan oliokielen että toisaalta jonkin muun kielen –risteytyksiä. Näistä kielistä tunnetuin on C++, jossa yhdistyy kyseisen kielen oliopiirteet ja C-kielen rakenteinen ohjelmointitapa (Haikala ym., 1998, 325 – 326).

Olio-ohjelmointi on yksi ohjelmointiparadigma eli tapa kuvata erilaisten järjestelmien rakennetta ja toimintaa. Muita yleisiä ohjelmointiparadigmoja ovat esimerkiksi proseduraalinen ohjelmointi ja funktionaalinen ohjelmointi (Koskimies, 1997, 1 – 2). On erityisen hyvä huomata, että pohjoismaisen koulukunnan mukaan ohjelmointikielten kehitys on evolutiivista, ei revolutiivista. Tämä siis poikkeaa amerikkalaisen koulukunnan näkemyksestä.

Olio-ohjelmointi-vaiheessa on erotettava selkeästi termit olio ja luokka. Koska oliot ovat luokkien niin sanottuja instansseja, yleensä ajonaikaisia ilmentymiä. Vain siis olio-ohjelmoinnin yhteydessä voidaan puhua puhtaasti olioista niitä selkeästi tarkoittaen. Kaikissa muissa vaiheissa ainakin ennen olio-ohjelmointia on periaatteessa käytettävä olioista sanaa luokka. On kuitenkin yleisesti hyväksytty sanan *olio* kuvaavan myös sanaa *luokka*, *luokan* ollessa *olio*n ”muotti”. Muun muassa LaLonde ym. (1993) selvästi kuvaavat, että olioperustaisen ohjelmistokehityksen vaiheiden termistö on sama.

Miksi sitten olio-ohjelmointi on ollut niin suosittua jo vuosikymmeniä ja mitä hyötyjä se tarjoaa ohjelmistokehittäjille? Budd (1997, 1 – 2) kuvaa olio-ohjelmoinnin hyötyjä seuraavasti. Ihmiset uskovat, että olio-ohjelmointi helposti ja nopeasti parantaa tuottavuutta ja tuo ohjelmistotyöhön lisävarmuutta. Myöskin siirtymisen vanhemmista kielistä olio-

ohjelmointikieliin nähdään olevan on yksinkertaista, koska ongelmanratkaisun ajatusmallit eivät muutu suuresti siirryttäessä olio-ohjelmointiin.

4 KOLMEN VAIHEEN VÄLISET SUHTEET

Vaikkakin olioperustaisen ohjelmistokehityksen vaiheiden tarkoitus on tukea toisiaan, ei niiden suhde ole aina täysin selkeä. Esimerkiksi oliomäärittelyn ja -suunnittelun erot ovat häilyvät, vaikka molempien kohdistus on selkeä (Booch, 1994). Lisäksi on havaittu, että suora siirtyminen oliomäärittelystä oliosuunnittelun kautta oliototeutukseen ei suinkaan ole mutkatonta (Haikala ym., 1998. 329).

Tässä luvussa tutkitaan, missä laajuudessa vaiheiden integraatio toteutuu, vai onko se puutteellista joka suhteessa. On hyvä pitää mielessä, että vaiheiden rajat ovat saman termistönsä vuoksi olioperustaisissa järjestelmissä häilyvät. Lisäksi on muistettava olioperustaisuuden kannustavan iteratiiviseen toimintaan, joka omalta osaltaan hämärtää vaiheiden keskinäistä eroavaisuutta.

4.1 Yleistä

Todellisten olioperustaisten CASE-välineiden (*Computer Aided Software Engineering*) puute haittaa täysin integroidun tuotantoprosessin toteutumista. Yksi huomattava syy tällaisten välineiden puutteelle on selkeästi olio(-perustaisten-)menetelmien laaja kirjo ja vaihtelevaisuus (Berard, 1993, 5).

Tästä huolimatta on ympäri maailman esitelty monia erilaisia olioperustaista ohjelmistokehitystä avustavia ohjelmia. Tällaiset ohjelmat keskittyvät perinteisesti kuitenkin vain johonkin tiettyyn kokonaisuuteen tai vaiheeseen, kuten esimerkiksi mallinnuksen avustamiseen. Yksi tunnetuin mallinnusta avustava ohjelma on Rationalin ROSE –mallinnusohjelma. Vastaavasti olio-ohjelmointia toteuttavista kielistä on kaikista tunnetuin C++ –ohjelmointikieli.

Lisäksi LaLonde ym. (1993) väittävät, että nykyisellään oliomäärittely, oliosuunnittelu ja olio-ohjelmointi ovat suvaitsemattomia muutoksille. Esimerkiksi kohtuullisen suuri muutos vaatimuksissa kesken toteutusvaihetta, vaatii miltei aina ylhäältä-alas –prosessin uudelleenkäynnistämistä

aikaisemmalta elinkaaren tasolta lähtien. Toisin sanoen on suuri epäkohta, ettei ole olemassa mitään metodiikkaa, joka mahdollistaisi integroidun samanaikaisen aktiviteetin toimittamista.

LaLonde ym. (1993) myös väittävät, että ei ole olemassa työvälineitä, jotka pystyisivät toimittamaan peruutusta eli suorittamaan niin sanottua alhaalta-ylös (*engl. bottom-up*) –prosessia. Tarve peruutukseen syntyy esimerkiksi muutoksista, jotka säteilevät toteutuksesta takaisin suunnitteluun (tai jopa määrittelyyn) saakka.

4.2 Oliomäärittely ja -suunnittelu

Muun muassa Haikala ym. (1998, 329) esittävät, että oliomäärittelyn ja oliosuunnittelun rajanveto on häilyvä. On siis hyvin helppo periaatteessa siirtyä vaiheesta toiseen sitä huomaamatta. Määrittelyvaiheessa määriteltävää järjestelmää on toki suunniteltava, mutta vaiheet erottaa toisistaan asiaan paneutumisen taso.

Oliomäärittelyn ja -suunnittelun suurimman sudenkuopan voidaan todeta olevan vaiheiden tunnistamisen ongelma. Koska määrittelyn ja suunnittelun suhde on läheinen, voidaan suhteen läheisyyden nähdä olevan iteraation ja evolutiivisen kehittymisen kannalta toivottu asia. Muita ongelmakohtia ei näiden vaiheiden välillä ole kritisoitu. Oliomäärittelyn tuloksia voidaan siis käyttää hyvinkin tehokkaasti ja laajasti oliosuunnittelussa.

4.3 Oliosuunnittelu ja -ohjelmointi

Suurin ongelma suunnittelun ja toteutuksen välillä on vaiheiden keskinäinen erilaisuus, sillä jo pelkästään käsitteet suunnittelu ja toteutus eroavat toisistaan suuresti. Suunnittelu on jonkin kokonaisuuden tarkastelemista ja sen rakenteen ja toiminnan selvittämistä. Toteutus puolestaan on toimintaa, jossa jokin kokonaisuus muodostetaan toiminnalliseksi ja rakenteelliseksi ratkaisuksi.

LaLonde ym. (1993) kritisoivat, että ei ole olemassa sellaista yleistä työvälinettä, joka tukisi yhtäaikaista toimintaa näiden kolmen vaiheen suhteen. Heidän näkemyksen mukaan tämä on suuri vahinko, sillä olioperustaisuus pyrkii suunnittelun ja toteutuksen yhtäaikaisuuteen. Edelleen he kritisoivat, että olio-ohjelmointi ymmärretään nykyään erittäin hyvin, mutta oliomäärittelyä ja oliosuunnittelua ei vieläkään täysin ymmärretä eikä niiden rooleja mielletä.

Winograd (1990) kritisoi vuoden 1990 OOPSLA –konferenssissa alkupuhujan roolissa sitä, että ihmiset eivät ole tietoisia todellisesta tahdostaan. Hänen mukaansa muun muassa eräässä OOPSLA –julkaisussa peräänkuulutettiin oliosuunnittelua, mutta silti puhuttiin olio-ohjelmoinnista. Tämä perustavaa laatua oleva ristiriita on osaltaan luonut näiden kahden vaiheen välille suuria paineita.

Winograd (1990) painottaa, että ilman suunnittelua ei ole toteutusta. ”Suunnittelu ei ole tekniikka, vaan pikemminkin se konkreettinen asia, jonka perusteella voidaan ylipäätään jotakin toteuttaa”. Lisäksi hän kritisoi sitä, että perinteisesti tietojenkäsittelyn opetuksessa keskitytään opettamaan vain eri tekniikoita eikä niinkään sitä, kuinka tekniikoita voitaisiin hyväksikäyttää eri tarpeiden täyttämiseksi.

Lisäksi olio-ohjelmoinnin synty ennen oliosuunnittelua (ja oliomäärittelyä) on luonut osaltaan väärää kuvaa vaiheiden tehtävistä. Oliosuunnittelu on siis kehitetty olio-ohjelmoinnin tarpeita silmälläpitäen eikä omaehtoisesti.

Kaikki nämä mainitut asiat huomioon ottaen, on toteutuksen ja suunnittelun integraatio kaikista suurimman turbulenssin kohteena olioperustaisessa ohjelmistokehityksessä. Konseptinen kielimuuri vaiheiden välillä on toki kohtalaisen suuri, mutta tosiasiassa vaiheiden tavoitteet ovat samat. Voidaan siis todeta, että vaiheilla on kuitenkin enemmän yhteistä kuin eroavuutta. Yleisesti oliosuunnittelun tuloksia pystytään käyttämään hyväksi olio-ohjelmoinnissa riittävissä määrin.

4.4 Integroitu määrittely, suunnittelu ja toteutus

Olio-ohjelmointi saa siis syötteenä sitä edeltävistä vaiheista tuotetut tulokset. Ilman minkäänlaista etukäteen tehtyä määrittelyä tai suunnittelua on olio-ohjelmointi todella raskas tehtäväkokonaisuus, varsinkin suurissa järjestelmissä ja ohjelmistoissa. Koska olio-ohjelmointi luotiin ennen oliosuunnittelua, ja oliosuunnittelu ennen oliomäärittelyä, on periaatteessa mahdollista siirtyä määrittelystä suoraan toteutukseen tai jopa aloittaa suoraan toteutuksesta. Näin ei kuitenkaan ole suositeltavaa toimia, koska silloin lopputulos ei todennäköisesti ole toivottu.

Korson ym. (1990) esittävät, että vaiheiden integraation lisäämiseksi olisi luotava niin sanottu ”olioperustainen kieli”, joka kattaisi määrittelyn, suunnittelun ja toteutuksen. Aiemminhan jo todettiin, että vaiheiden sanasto on sama. Ehdotuksessa mennään astetta pidemmälle, sillä se ei tarkoita perinteisessä mielessä käsitettyjä notaatiokieliä, jotka ovat vain ja ainoastaan mallinnukseen (toisin sanoen määrittelyyn ja suunnitteluun) keskittyviä.

Kuinka määrittely, suunnittelu ja toteutus sitten loppujen lopuksi integroituvat toisiinsa? Nykyisellään voidaan integraation todeta toimivan kohtalaisessa määrin. Tässä suhteessa on kuitenkin vielä paljon parantamisen varaa. Tärkein yksittäinen nimetty asia, jolla integraatiota voidaan huomattavasti parantaa, on yhtenäisten työvälineiden kehittäminen [Sekä Berard (1993) että LaLonde ym. (1993)].

5 YHTEENVETO

Tässä kandidaatintutkielmassa on käsitelty olioperustaista ohjelmistokehitystä ja erityisesti sen kolmea keskeistä vaihetta – oliomäärittelyä, oliosuunnittelua ja olio-ohjelmointia. Tutkielmassa on tutkittu myös vaiheiden välisiä suhteita ja tuotu esille suhteiden erityispiirteitä. Aihepiiriksi valitut vaiheet ovat sekä konkreettisimmin kehitettäviin ohjelmiin vaikuttavia että oliopiirteisiin eniten kantaa ottavia vaiheita. Tutkielman tärkeimmät havainnot ja johtopäätökset on tiivistetty seuraavaan.

Olioperustaisuudessa on yleisesti nähty monia etuja. Tärkein konkreettinen hyöty on siinä, että se pyrkii kuvaamaan reaalimaailmaa yksiselitteisesti. Lisäksi olioperustainen ohjelmistokehitys nähdään yhtenä merkittävänä lähestymistapana ohjelmistojen tuottamiseen. Sen on esitetty muun muassa parantavan tuottavuutta ja lisäävän järjestelmien ymmärrystä.

Vaikka olioperustaisen ohjelmistokehityksen vaiheiden tarkoitus on tukea toisiaan, ei niiden suhde aina ole täysin selkeä. Oliomäärittelyssä pyritään mallintamaan reaalimaailmaa. Oliosuunnittelussa keksitään abstraktiot ja mekanismit, jotka tarjoavat määritellyn mallin käytöksen. Olio-ohjelmoinnissa määritelty ja suunniteltu järjestelmä toteutetaan.

Vaiheiden välillä on monenlaisia asioita haittaamassa niin sanotun integroidun prosessin toteutumista. Ensinnäkin on huomioitava, että vielä nykyäänkin on tarvetta olioperustaista ohjelmistokehitystä avustaville CASE –välineille. Kunnollisten ohjelmistojen elinkaaren kattavien ja ohjelmistoprosessin eri vaiheita tukevien välineiden puute on ollut yksi merkittävimmistä tekijöistä, joka vaikeuttaa vaiheiden integraatiota. Mikäli tällainen väline luotaisiin, olisi tilanne monissa määrin ihanteellinen.

Suunnittelua ja määrittelyä jo tukevat notaatiokielet tulisi laajentaa koskemaan määrittelyn ja suunnittelun lisäksi toteutusta. Nykyisin käytetty notaatiokieli on olio-ohjelmoinnissa käytettyyn kieleen nähden monilta osin erilainen. Olisi

tärkeää standardoida yksi yleinen (perus-)notaatiostandardi kattamaan kaikkia kolmea keskeistä vaihetta. Eniten tätä tavoitetta lähestyy UML –notaatiokieli, joka kuitenkin toistaiseksi keskittyy vain määrittelyn ja suunnittelun tukemiseen.

Olioperustainen ohjelmistokehitys on nykyään eräs tärkeä ja huomattava lähestymistapa ohjelmistotuotannossa ja sitä se varmasti tulee olemaan vielä monen vuoden ajan. Jotta olioperustaista ohjelmistokehitystä voitaisiin parantaa, on edellä esitetyt asiat tärkeää huomioida. Vaikka nykyiselläänkin oliomäärittely, -suunnittelu ja -ohjelmointi toimivat keskenään tietyssä määrin, niiden keskinäisen integroidun toiminnan parantaminen tehostaisi ja helpottaisi koko olioperustaisen ohjelmistokehitysprosessin hallintaa.

LÄHDELUETTELO

Berard E.V. 1993. Essays on Object-Oriented Software Engineering Volume 1. Prentice-Hall, Inc.

Booch G. 1994. Object-Oriented Analysis And Design With Applications (2nd ed.). The Benjamin/Cummings Publishing Company, Inc.

Budd T. 1997. An Introduction to Object-Oriented Programming (2nd ed.). Addison Wesley Longman, Inc.

De Champeaux D., Lea D., Faure P. 1992. The Process of Object-Oriented Design. Conference proceedings on Object-oriented programming systems, languages, and applications. Vancouver, British Columbia, Canada. Pages: 45 - 62. The ACM Digital Library.

Coad P., Yourdon E. 1990. Object-Oriented Analysis. Prentice-Hall, Inc.

Haikala I., Märijärvi J. 1998. Ohjelmistotuotanto (5. Painos). Suomen Atk-kustannus Oy.

Korson T., McGregor J. D. Understanding Object-Oriented: A Unifying Paradigm. Communications of the ACM. September 1990, Volume 33, No. 9. Pages: 40 – 60. The ACM Digital Library.

Koskimies K. 1997. Pieni oliokirja. Suomen Atk-kustannus Oy.

Koskinen J, Sakkinen M., Paakki J. 2001. Ohjelmistotekniikka (Opetusmoniste) (2. Painos). Jyväskylän yliopiston yliopistopaino.

LaLonde W., Pugh J., White P., Corriveau J.P. 1993. Towards Unifying Analysis, Design, and Implementation in Object-Oriented Environments. Proceedings of the 1993 conference of the centre for Advanced Studies on Collaborative research: software engineering – Volume 1. Toronto, Ontario, Canada. Pages: 536 - 569. The ACM Digital Library.

Mathiassen L., Munk-Madsen A., Nielsen P.A., Stage J. 2000. Object-Oriented Analysis & Design. Marko Publishing ApS.

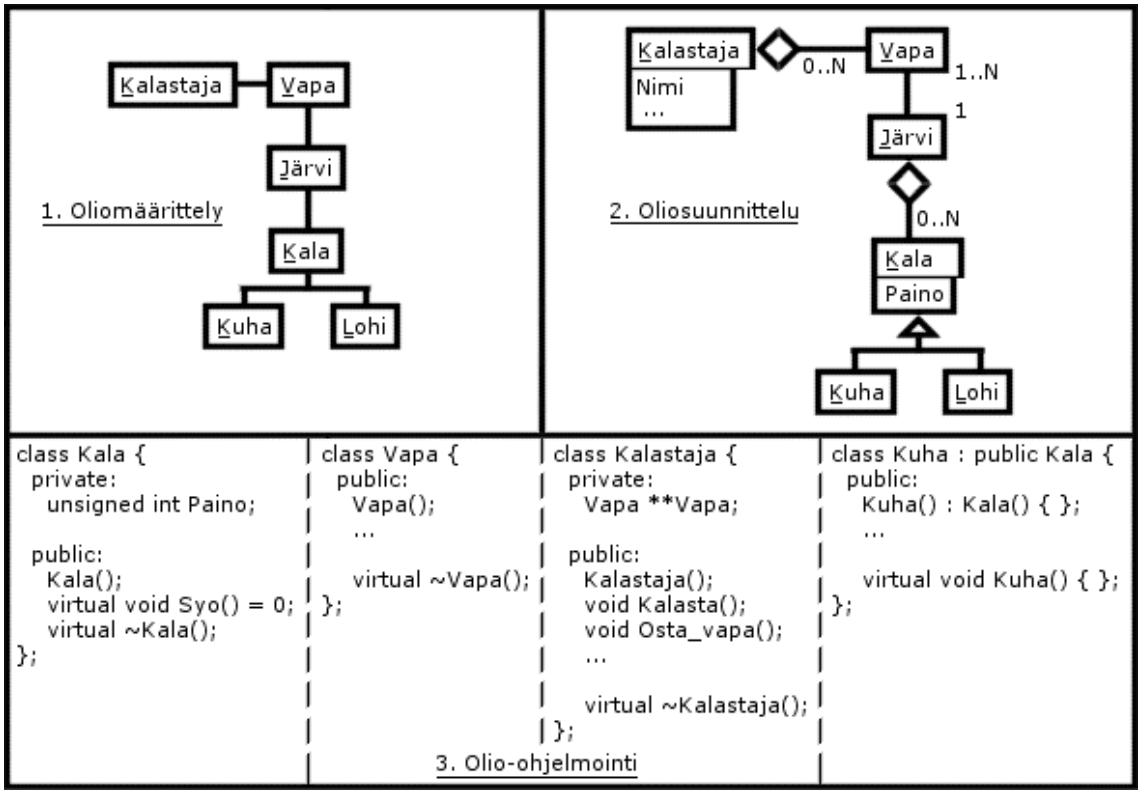
Nerson J. - M. 1992. Applying Object-Oriented Analysis and Design. Communications of the ACM. September, Volume 35, No. 9. Pages: 63 – 74. The ACM Digital Library.

Winograd T. 1990. Object-Oriented Programming, Systems, Languages, And Applications (OOPSLA) [Keynote/Addendum to the Proceedings], 21-25 October 1991. Pages: 7 – 8. Ottawa, Canada.

LIITE 1

ESIMERKKI

Alla on kuvattu yksinkertainen ja konkreettinen esimerkki olioperustaisen ohjelmistokehityksen keskeisten vaiheiden – oliomäärittelyn, oliosuunnittelun ja olio-ohjelmoinnin – toteutuksista. Oliomäärittelyn ja oliosuunnittelun esitysmuoto on UML –notaatiokieltä mukaileva ja esimerkki olio-ohjelmoinnista on esitetty C++ –ohjelmointikielellä.



Kuva 2: Olioperustaisen ohjelmistokehityksen kolme vaihetta -esimerkki.