OCTOPUS

Ohjelmistotuotannon harjoitustyö

28.4.2013

Annemari Auvinen
Anu Niemi
Jani Sairanen
Aki Suihkonen
Tuukka Tawast

Jyväskylän yliopisto

Tietotekniikan laitos

Sisältö

Esipuhe

Nokiaa kiitellään kirjallisuudessa toistuvasti menestyksekkäästä uuden teknologian markkinoinnista. Vuosien 1996 ja 1998 aikana, jolloin yhden tämän harjoitustyön tekijöistä oli mahdollisuus tutustua tämän menestyksen elementteihin, keskimääräinen työntekijän osuus yhtiön arvonnoususta oli yli miljoona markkaa vuodessa. Tämä oli ennen informaatioteknologia-alan ansiotonta arvonnousua.

Jos tämä menestys onkin johtunut Matti Ala-Huhdan sanoin yhtiön määrätietoisesta suuntautumisesta teknologian murroskohtiin (Agora, 6.11.2001), joukkoon on mahtunut vankan teknologisen osaamisen lisäksi huipputason harhaanjohtamista.

Ilmeisesti jo ennen 90-lukua yrityksen strategioihin oli valittu osuuden kaappaaminen SDH ja PDH tyyppisten runkoverkkojen solmulaite-, ja toteuttamismarkkinoilta. Tältä pohjalta voi ymmärtää yhtiön tutkimuskeskuksessa suunniteltujen pc-pohjaisten tcp/ip-reitittimien vähättelyä markkinoinnin kannalta, erityisesti kun kyseessä oli uusi tuntematon teknologia. Teknologiahuuman ollessa kiihkeimmillään vuoden 2000 maaliskuussa Nokian laskennallinen arvo oli viidenneksi suurin ja ylitti arvoltaan Suomen vuotuisen budjetin moninkertaisesti. Listan kärjessä oli muuan internet-reitittimiin keskittynyt yhtiö.

Vuonna 1995 aloitetussa NAN (New Access Node) Exson N20 projektissa ehti työskennellä toista sataa ohjelmoijaa sekä kymmenkunta laitesuunnittelijaa ennen kuin NTC/FAS (Fixed Access Systems) yksikkö siirtyi englantilaisen tietoliikenneyhtiön Marconin omistukseen vuoden 1999 syksyllä.

Viiden vuoden aikana FAS:n kahdessa projektissa tuotettiin toista miljoonaa koodiriviä verkonhallintasovellukseen, sekä itse tuotteena myytäväksi tarkoitettuun paikallisvaihteeseen (local exchange). NAN:n ohjelmistoarkkitehtuuri samoin kuin suurin osa koodista pohjautui aikaisempaan kahden megabitin SDH päätelaitteen ohjelmistoon, mikä aiheutti suuria adaptoitumisongelmia OCTOPUS-metodologiaa käytettäessä.

1 Johdatus OCTOPUS-metodologiaan

Johtuen sovellusalueiden eri tarpeista, tehokkaan ohjelmistotuotantometodin on oltava sovellusaluekohtainen, joka keskittyy oman sovellusalansa tärkeisiin näkökohtiin. OCTOPUS –metodi tarjoaa systemaattisen lähestymistavan tuotettaessa oliokeskeisiä sulautettuja reaaliaikasovelluksia. Se tarjoaa ratkaisun moniin tärkeisiin ongelmiin kuten samanaikaisuus, synkronointi, kommunikointi, keskeytysten hallinta, ASIC, laitteiston rajapinnat ja end-to-end vasteaika systeemin läpi.

OCTOPUS on hyvin integroitu tuotantoprosessi sisältäen tärkeimmät vaiheet ohjelmistoprosessissa. Inkrementaalinen tuotantoprosessi on tuettu. OCTOPUS lähestyminen tukee myös heterogeenisten systeemien tuotantoa, jossa oliokeskeinen ratkaisu on vain osa ei-oliokeskeistä systeemiä. OCTOPUS metodi käyttää niin pitkään kuin mahdollista OMT-notaatiota ja soveltaa standardeja.

OCTOPUS –metodissa on rakenteiset, funktionaaliset ja dynaamiset mallit systeemi-, alisysteemi-, luokka- ja oliotasoille. Nämä tasot on jaettu seuraaviin tuotantovaiheisiin.

1.1 Vaatimusmäärittelyvaihe

Vaatimusmäärittelyvaihe kuvaa systeemin ympäristön rakenteen kontekstidiagrammilla. Systeemin funktionaalinen ja dynaaminen käyttäytyminen kuvataan käyttötapausdiagrammilla ja useilla käyttötapauksilla. Käyttötapaukset voidaan täydentää skenaarioilla systeemistä ja sen ympäristöstä, joka tuo esiin systeemin dynaamisesta käyttäytymisen. Systeemi on black-box.

1.2 Arkkitehtuurivaihe:

Arkkitehtuurivaihe kuvaa systeemin rakennetta alisysteemidiagrammilla. Tärkeimmät rajapinnat alisysteemien välillä on kuvattu diagrammissa riippuvuuksina. Alisysteemi on black-box.

1.3 Alisysteemin analyysivaihe:

Alisysteemin analyysivaihe tehdään jokaiselle alisysteemille.

1. Rakenteinen malli on alisysteemin luokkadiagrammi lisättynä luokkakuvaustaululla.
2. Funktionaalinen malli on ryhmä toiminnallisuuskaavioita.
3. Dynaaminen malli sisältää tapahtumalistan, tapahtumaryhmädiagrammin, tapahtumakaavion, tilakaaviot, tapahtumataulun, merkitsevyyslistan ja mahdollisesti lista yhdistävistä tapahtumista.

Funtionaalinen ja dynaaminen malli kohtelevat alisysteemiä black-boxina mutta voivat tehdä viittauksia rakenteisen mallin luokkiin

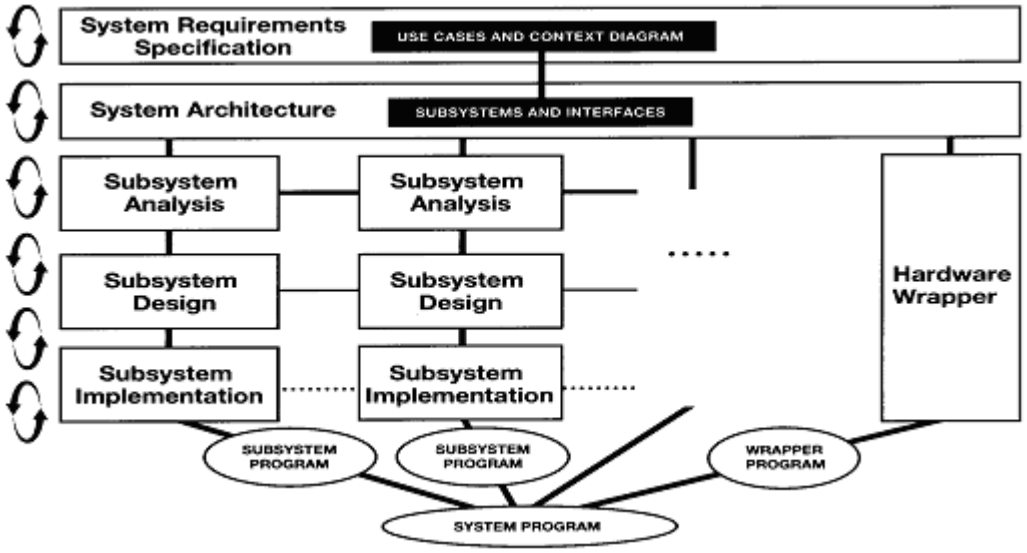
1.4 Alisysteemin suunnitteluvaihe:

Alisysteemin suunnitteluvaihe tehdään jokaiselle alisysteemille. Se on jatkoa alisysteemin analyysivaiheelle. Dynaaminen malli esitetään ryhmällä olioiden interaktio –säikeitä. Nämä ovat tämän jälkeen jaettu sopiviin tapahtumasäikeisiin ja tämän jälkeen olioryhmät on jaettu. Näistä luodaan systemaattisesti ääriviivat prosesseille, luokille ja prosessiensisäisille viesteille.

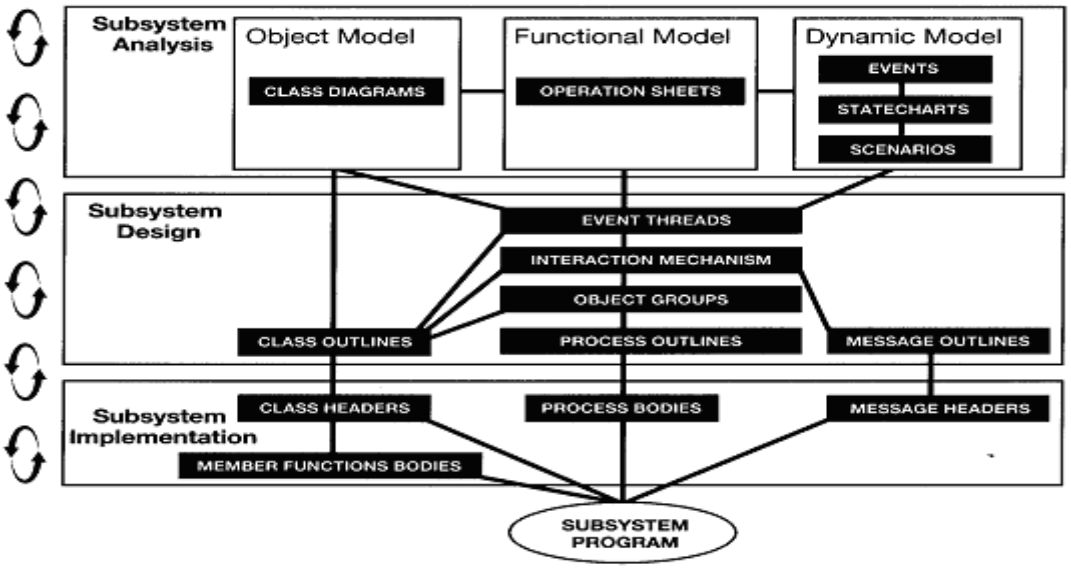
1.5 Alisysteemin toteutusvaihe:

Toteutetut ääriviivat muunnetaan valitun ohjelmointikielen koodiksi.

Prosessi on iteratiivinen. Iteraatiot vähennevät saavuttaessa kohti toteutusvaihetta. Iteraatiot tapahtuvat useammin vaiheen sisällä kuin vaiheiden välillä.



Kuva 1: Ohjelmiston suunnitteluprosessin rakenne.



Kuva 2: Alijärjestelmän kehitysprosessin rakenne.

2 Vaatimusmäärittely

Koska sulautetut järjestelmissä ohjelma on vain osa kokonaisuutta ja se on tehty käyttäjälle näkymättömäksi,vaatimusmäärittely kuvaa koko tuotteen kaikkine puolineen sisältäen elektroniikan ja mekaniikan.

2.1 Käyttötapaukset

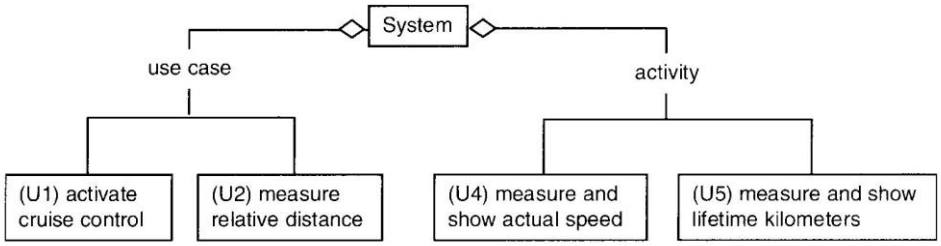
Käyttötapaukset kohtelevat systeemiä black boxina, jokainen tapaus kuvaa erikoistapausta systeemin käytössä. Käyttötappauksen tulisi näyttää, kuinka systeemi toimii aktorin näkökulmasta.

Aktoreita voidaan luokitella usealla eri tavalla:

- Aktiivinen/passiivinen:.Passiivinen aktori ei; koskaan aloita käyttötapausta..
- Asiakas/ei-asiakas: Asiakas käyttää systeemiä tiettyä tarkoitusta varten, ei-asiakas vain vaikuttaa siihen.
- Ensisijainen/toissijainen: Toissijaiset aktorit esiintyvät ainoastaan siksi, että ensisijaiset aktorit voivat käyttää systeemiä.

2.2 Käyttötapausdiagrammit

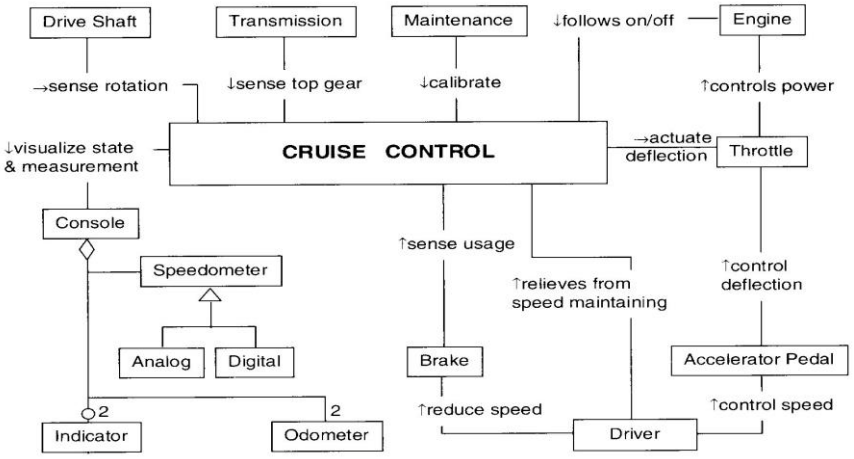
Käyttötapaukset voivat olla hierarkkisesti koostettu. Ne eivät yleensä sisällä alikäyttötapausta, mutta jakavat poikkeuksia ja alitoimintoja. Hierarkkia voidaan visualisoida käyttötapausdiagrammilla käyttämällä oliodiagrammin notaatiota.



Kuva 3: Käyttötapausdiagrammi.

2.3 Systeemin kontekstidiagrammi

Kontekstidiagrammi esittää aktorit ja sen, kuinka ne ovat suhteessa systeemiin. OCTOPUS tehostaa kontekstidiagrammin ilmaisua sallimalla tärkeiden riippuvuuksien näyttämistä aktoreiden välillä. Yksityiskohtaisuuden taso kontekstidiagrammissa voi vaihdella mutta sen tulisi sisältää kaikki aktorit, jotka on mainittu systeemin käyttötapauksissa. Ei ole väliä, onko aktori ensisijainen tai toissijainen, aktiivinen tai passiivinen vaiko asiakas tai ei-asiakas.



Kuva 4: Kontekstidiagrammi.

3 Systeemiarkkitehtuuri

Systeemiarkkitehtuuri koostuu laitteisto- ja ohjelmistoarkkitehtuurista. Käytännössä ohjelmisto- ja laitteistosuunnittelu ovat usein erillisiä ja laitteistosuunnittelu on ennen ohjelmistosuunnittelua. Tämä johtuu siitä, että käytetään standardeja laitteistokomponentteja tai laitteiston aikasempaa versiota. Vaikka laitteisto on erityisesti suunniteltu systeemiä varten, sen suunnittelua ohjaa usein epäfunktionaaliset argumentit kuten komponenttien virrankulutus. Ohjelmiston suunnittelu vaikuttaa laitteistoon rakenteeseen vain silloin, kun haluttua toiminnallisuutta ei voi toteuttaa annetulla laitteistolla. Useimmiten laitteistoa käsitellään vain ohjelmiston ylimääräisinä rajoituksina.

Ulkoiset ja laitteistovaatimukset on eriteltävä. Muuten laitteiston rakenne ohjaisi ohjelmiston suunnittelua ja suunnittelu tulisi hyvin herkäksi laitteiston muutoksille. Kääntääkseen tämän järjestyksen OCTOPUS eristää laitteiston ohjelmiston taakse pinnan avulla, jota nimitetään hardware wrapperiksi. Hardware wrapperin analyysi ja suunnittelu voidaan lykätä kunnes vaatimukset ohjelmistolle ovat tiedossa. Hardware wrapper ohjaa laitteistoa ja tarjoaa palveluja alisysteemeille.

3.1 Modulaarinen rakenne

Ohjelmistoarkkitehtuuri on perinteisesti käsitetty systeemin modulaarisena hajotelmana. Tällöin ohjelmistoarkkitehtuurissa on neljä osaa:

1. systeemin modulaarinen rakenne
2. moduulien rajapinnat
3. kommunikointimekanismit
4. yleinen hallintastrategia

Tämä top-down näkymä ohjelmistoarkkitehtuurista tukee osien aikaista erillistämistä. Jos moduulit ovat heikosti koostettuja, ne voidaan tehdä ja testata lähes riippumattomasti. Todellisuudessa arkkitehtuuriset päätökset tehdään joka tasolla ja usein hienojakoisimmat päätökset muodostavat syteemin laadun. Lisäksi voi olla olemassa säännöstö vähentää näitä riippuvuuksia ja niitä tulisi tarkasti noudattaa.

OCTOPUS sisältää joukon sääntöjä, joita noudattamalla arkkitehtuuri

- perustaa modulaarisuuden ko ohjelmistoalan käsitteisiin
- ohjelmiston rakenteen mekanismina käytetään olioita
- käyttää light-weight prosesseissa eksplisiittista samanaikaisuuta
- perustaa prosessin rakenteen ulkoisten tapahtumien aikavaatimuksille
- perustaa prosessien kontrollimekanismin viesteihin
- erottaa laitteistopinnan sovelluksesta

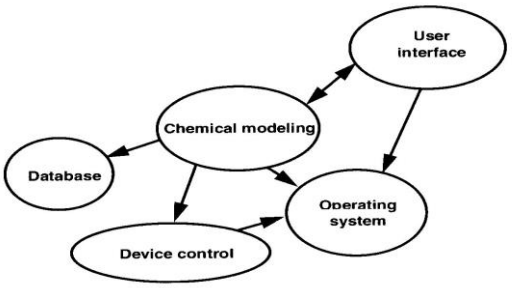
3.2 Aikainen jako alisysteemeihin

Suurissa systeemeissä on järkevää jakaa analyysityö useiden tiimien kesken. Tämä voidaan toteuttaa jakamalla systeemi pienempiin alisysteemeihin ja analysoimalla ja tekemällä

nämä alisysteemit riippumattomasti. Aikainen jako sisältää riskin. Systeemitason vaatimukset eivät ole hyvin määriteltyjä ennen jaon tekemistä ja vaatimukset voi olla vaikea täyttää valitulla alisysteemi rakenteella. Aikainen jako lisäksi lisää työtä. Alisysteemeihin jako ei jaa vaatimusten määrää vaan samat vaatimukset analysoidaan useaan otteeseen. Eri alisysteemeistä tulee samannimisiä luokkia, joilla on kuitenkin eri toteutukset. Tämä on on ongelma systeemin jatkokehityksen kannalta. Lisäksi alisysteemin rajapinnat kehittyvät kun ymmärrys vaatimuksista lisääntyy.

Mikäli analyysityö on jaettava, hyvä tapa taata riippumattomuus alisysteemien välissä on käyttää erilaisia alueita eri alisysteemeissä. Nämä alueet ovat erillisiä maailmoja sisältäen erillisiä olioita, jotka käyttäytyvät tämän erillisen alueiden sääntöjen ja tapojen mukaan. Koska nämä alueet ovat riippumattomia, alueperusteinen alisysteemeihin jako tuottaa alisysteemeitä, joita voidaan ajaa riippumattomasti ja samanaikaisesti.

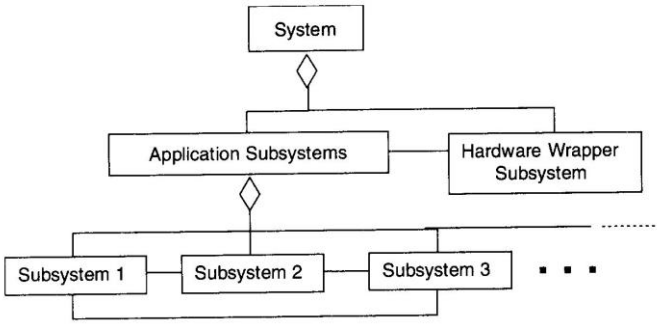
Esimerkisi kemikaalinen prosessi, jonka alueet ovat kemikaalinen alue (kontrolloitavan prosessin malli), laitealue (sensorit, jotka mittaavat prosessia), käyttöjärjestelmäalue, tietokanta-alue ja käyttöliittymäaalue.



Kuva 5: Kemikaalisen prosessin väliset suhteet.

3.3 Alisysteemidiagrammi

OCTOPUS erottaa hardware wrapperin muista alisysteemeistä. Client/server -suhteet alisysteemien välillä voidaan näyttää riippuvuuksina.



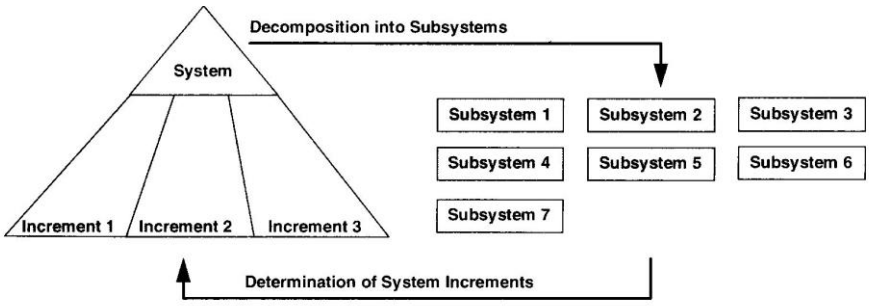
Kuva 6: Alisysteemidiagrammi.

3.4 Inkrementaalinen tuotanto

Alisysteemeihin jako voi laskea ohjelmistoprojektin tekemiseen menevää aikaa, koska alisysteemejä voidaan toteuttaa rinnakkain. Jakamisella on myös muita positiivisia vaikutuksia: tehtävien jako henkilöstön kesken tulee helpommaksi ja joustavammaksi. On suositeltu, että jokaisen alisysteemin tekemiseen määrätään vähintään kaksi suunnittelijaa.

Alisysteemeihin jako tuo uudelleenkäytettävyyttä suuremmalla tasolla kuin luokkatasolla. Kokonainen, hyvin suunniteltu alisysteemi voidaan uudelleenkäyttää toisessa systeemissä.

Alisysteemeihin jako ei ole ilmaista. Se edellyttää alisysteemien välisten liittymien määrittelyä. Täten ei ole hyvä olla rakennetta, jossa alisysteemit ovat suuresti toisistaan riippuvaisia. OCTOPUS suosittelee alisysteemien rinnakkaista analysointia ennen suunnittelua ja jonkun alisysteemin toteutusta, jolloin voidaan selkeästi määritellä alisysteemien väliset rajapinnat. Tällöin voidaan vielä kehittää alkuperäistä alisysteemeihin jakoa. Tällöin jonkin alisysteemin vastuut voivat muuttua, kaksi toisistaan riippuvaa alisysteemiä yhdistetään tai uusi alisysteemi luodaan.



Kuva 7: Alijärjestelmät ja järjestelmälisäykset.

Suuri systeemi täytyy tuottaa inkrementaalisesti. Yksi inkrementti on koostettu yhdestä tai useammasta alisysteemistä, jotka tuottavat osan järjestelmästä, joka voidaan testata. Seuraava inkrementti voidaan lisätä aikaisempiin kunnes täysin toiviva systeemi on valmis.

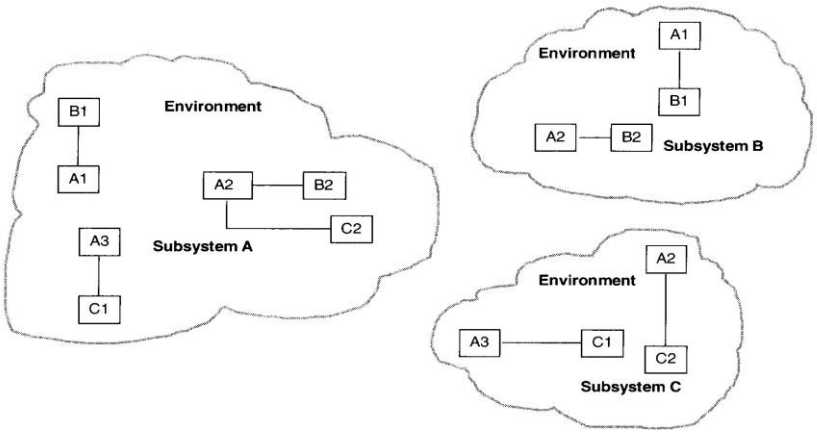
3.5 Rajapinnat

Arkkitehtuuri tulisi kuvata suurella tarkkuudella. On olemassa useita ohjaus- ja data virta mekanismeja ja eri mekanismeja voidaan yhdistellä. Kontrolli voi olla eksplisiittistä tai implisiittistä tai se voi myöskin olla perustettu käyttöjärjestelmän palveluille, mikäli ne ovat sopivia. Tiedon jakaminen voidaan toteuttaa viesteinä, proseduurikutsuina tai yhteiseen muistiin. Alisysteemit tulisi olla ymmärrettävästi riippumattomia. Ylenmääräinen kommunikointi on varoituslippu.

Tyypillinen arkkitehtuuri sulautetuissa systeemeissä on reagiontiperusteinen: kontrolli on usein tapahtumakeskeinen ja käyttöjärjestelmän viestejä käytetään laukaisemaan tapahtumia. Kannattaa valita mekanismeja, joissa viestien koodaus ja purkaminen vie mahdollisimman vähän resursseja.

Alisysteemien rajapinnat määritellään analyysivaihessa. Funktionaalinen malli kuvaa palvelut ja dynaaminen malli kuvaa miten alisysteemit ovat interaktiossa. Rajapinta voidaan nähdä palvelukokoelmana, jotka tarjoaa alisysteemin ympäristön oliot. Suhteet muihin alisysteemeihin ovat nähtävissä oliomallissa suhteina olioihin ympäristössä. Tämä mahdollistaa näyttää suhteen tyypin alisysteemien välillä. Myöhemmin voidaan esittää tarkka rajapinta, jonka suhde tarvitsee.

Koska alisysteemien liittymien kuvaus toteutetaan analyysivaiheessa mutta alisysteemeihiin jako tehdään ennen analysivaihetta, on tarpeen iteroida arkkitehtuurin suunnittelun ja analyysin välillä.



Kuva 8: Alijärjestelmät rajapinnan ympäristöjen olioina.

4 Analyysivaihe

Octopus menetelmässä käytetään analyysivaiheessa kolmea mallia. Mallit täydentävät toisiaan ja jokaisessa mallissa keskitytään eri piirteiden analysointiin

4.1 Oliomalli

Oliomalli koostuu luokkakaaviosta ja olioiden yksityiskohtaiset tiedot sisältävistä luokkakuvaustaulukoista. Luokkakuvaustaulukossa on luokkakohtaisesti läpikäyty kyseiselle luokalle asetetut tehtävät.

4.2 Toiminnallinen malli

Toiminnallisen mallin tarkoitus on kuvata alijärjestelmän toiminnallinen rajapinta. Tämä rajapinta koostuu alijärjestelmän toisille alijärjestelmille ja ulkoisille toimijoille tarjoamista palveluista.

Toiminnallisen mallin tarkoitus ei ole kuvata, koska ja miksi toiminnot aktivoituvat, päättyykö niiden suoritus ilman toimenpiteitä tai voidaanko niitä suorittaa rinnakkaisesti vai ei. Toiminnallisen vaiheen aikana toiminnot kuvataan Fusion-menetelmässä käytössä olevien operaatiotaulukoiden avulla. Taulukko sisältää:

- *Operaatio (operation):* Operaation nimi
- *Kuvaus (description):* Lyhyt kuvaus operaatiosta.
- *Assosiaatiot (associations):* Assosiaatiot luokkiin ja objekteihin sekä myöskin tapahtumiin ja tiloihin, joihin operaatio liittyy
- *Esiehdot (preconditions):* Esiehdot, jotka mahdollistavat operaation aloittamisen
- *Sisääntulot (inputs):* Operaation suorittamiseen tarvittavat argumentit
- *Muutokset(modifies):* Operaation aiheuttamat muutokset argumentteihinsa tai alijärjestelmän sisäiseen tietoon

4.3 Dynaaminen malli

Dynaaminen malli kuvaa alijärjestelmän toimintaa sen sovellusrajapinnan ylitse ja osoittaa reaaliaikajärjestelmän ominaisuuksia. Malli selvittää missä tilanteissa tapahtumat voivat tapahtua ja kuinka kauan niiden suoritus saa kestää. Mallilla kuvataan alijärjestelmän ja sen ympäristön vuorovaikutusta operaatioiden yhteydessä. Lisäksi dynaaminen malli kuvaa missä tiloissa alijärjestelmän operaatiot ovat mahdollisia ja kuinka ne vaikuttavat toisiinsa.

Octupus menetelmän dynaamisen mallin muodostus on kolmivaiheinen.

4.3.1 Tapahtumien analysointi

Dynaamisen mallin tapahtumien analysoinnissa keskitytään alijärjestelmään saapuvien (input) tapahtumien käsittelyyn. Alijärjestelmästä lähtevät tapahtumat (output) käsitellään funktionaalisen mallin yhteydessä. Saapuvat tapahtumat voivat keskeyttää, aktivoida, tai lopettaa alijärjestelmän toiminnon suorittamisen.

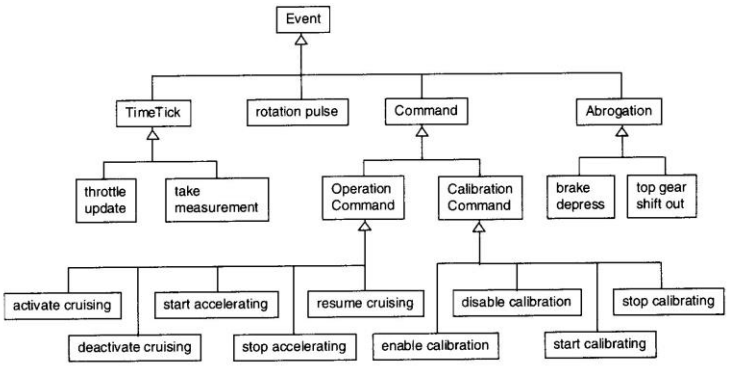
Tapahtumien analysointi suoritetaan kolmessa vaiheessa

Vaihe 1. Loogisten tapahtumien identifiointi ja kuvaus

Dynaaminen vaihe aloitetaan identifioimalla ja kuvaamalla tapahtumat. Tämä saavutetaan parhaiten tutkimalla luokkakaaviota ja tarkastelemalla siinä esiintyviä assosiaatioita. Erityisesti huomiota kiinnitetään assosiaatioihin, jotka ylittävät sovelluksen rajanpinnan. Osa näistä assosiaatiosta ilmaisee tapahtuman.

Vaihe 2. Tapahtumien yhtäläisyyksien etsiminen

Samankaltaisten tapahtumien ryhmittely helpottaa niiden hallintaa. Tapahtumien ryhmittely suoritetaan tapahtumalistaa läpikäymällä ja vertaamalla tapahtumien lähteitä, tietosisältöä, odotettua vastinetta ja aikarajoja. Mitä enemmän tapahtumillla on yhteisiä piirteitä sitä hyödyllisempää on muodostaa niistä yksi ryhmä.



Kuva 9:Luokkakaaviota voidaan käyttää myös tapahtumien ryhmittelyn kuvaamiseen.

Vaihe 3. Tapahtuma taulujen täyttäminen

Tapahtumataulu sisältää tapahtuman kannalta oleelliset tiedot:

- *Tapahtuma (event):* Tapahtuman nimi.
- *Vastine (response):* Järjestelmän suunniteltu vaste.
- *Assosiaatiot (assosiations):* Luokan ja olion assosiaatiot sekä mahdollisesti myös tapahtumaan liittyvät operaatiot.
- *Lähde (source):* Tapahtuman herättäjä.
- *Asiasisältö (contents):* Tapahtumaan liittyvät atribuutit.
- *Vasteaika (response time):* Maksimi- ja minimiaika vasteen lähettämiseksi.
- *Nopeus (rate):* Kuinka usein tapahtuman arvioidaan ilmenevän.

4.3.2 Tilojen analysointi

Tämän vaiheen ensimmäinen askel on identifioida tärkeimmät tilakaaviot. Identifionti perustuu aikaisempiin tapahtumien analysoinnin yhteydessä muodostettuihin operaatiotaulukoihin sekä fyysisen ympäristön tutkimukseen.

Kun tilakaaviot on saatu identifioitua on seuraavana vaiheena tutkia niiden yhteistoimintaa. Aikaisemmin tuotettuja tilakaavioita ei ole tarkoitus yhdistää yhdeksi suureksi tilakaavioksi vaan yhteistoiminnan tutkiminen suoritetaan tilojen listauksen avulla, jolloin tiloihin yhdistetään luokkakaavion assosiaatiot.

Tämän jälkeen seuraavana vaiheena on toimintataulukoiden täyttäminen. Toimintataulukko sisältää yhdessä vaikuttavien tilakaavioiden tiedot.

4.3.3 Skenaariot

Dynaamiseen malliin kuuluvat myös skenaariot, jotka kuvaavat alijärjestelmän ja sen ympäristön välistä vuorovaikutusta. Dynaamisen malliin kuuluvissa skenaarioissa alijärjestemä esiintyy kokonaisuutena eikä sen sisäistä toimintaa käsitellä yksityiskohtaisesti. Skenaario voi käsittää yksittäisen operaation, operaatioryhmän tai vain osan siitä. Skenaariossa voidaan kuvata aikarajoitukset tapahtumien välillä, alijärjestelmän tilan muutokset sekä myöskin oletetut alijärjestelmän tekemät alioperaatiot.

4.4 Hardware Wrapperin analyysi vaihe

Luokat, jotka ilmenevät yhdessä tai useammassa alijärjestelmän ympäristössä, mutta jotka eivät kuulu itse alijärjestelmään kuulluvat Hardware Wrapperiin. Nämä luokat edustavat joko Hardwaren ulkoisia tai sisäisiä käsitteitä. Esimerkiksi auton nopeuden näyttävä mittari on auton hardwaren ulkoinen osa kun taasen itse nopeusmittari on sisäisesti yhteydessä auton järjestelmäään.

Hardware wrapperin analyysi vaihe koostuu samojen mallien rakentamisesta kuin itse soveeluksenkin analyysivaiheessa. Perusperiaatteetkin näiden mallien rakentamisessa ovat samat näkökulma on vain hieman toinen.

Sovelluksen analyysivaiheessa ympäristöön liittyvat luokat kuvataan yhtenä luokkana, hardware wrapperin oliomallissa tämä abstrakti luokka hajoitetaan omiksi konkreettisemmiksi luokiksi. Nyt alijärjestelmien sovellukset ymmärretään yhtenä abstraktina luokkana.

Sovelluksen analyysivaiheen aikana hardwareen liittyvät liittyvät yksityiskohdat sivuutettiin.Dynaamisen mallin rakentamisen aikana ei kiinnitetty huomiota siihen kuinka tapahtumat syntyvät. Hardware wrapperin dynaaminen malli käsittelee tätä kysymystä.

5 Suunnitteluvaihe

Octopus-mallissa suunnitteluvaiheessa on tarkoitus luoda kuvaus, kuinka järjestelmä toimii ohjelmointikieliä varten. Suunnittelija kuvaa tarkasti olioiden viestimisen tapahtumakäsittein ja havainnollistaa sen oliokaaviomerkinnällä. Kuten analyysivaihekin, suunnitteluprosessi käydään läpi vähintään kaksi kertaa. Kertaalleen sovellussysteemille ja hardware wrapperille. Jos systeemi on jaotettu alisysteemeihin, jokainen niistä suunnitellaan erikseen rajojensa ja ympäristönsä mukaan. Hardware wrapperin suunnittelu sisältää samat käsitteet, mekanismit ja merkinnät kuin sovelluksen alisysteemi.

Suunnittelu muuttaa analyysivaiheessa luodun abstraktin määritelmän kuvaukseksi, joka selvittää kuinka yksityiskohdat ja funktiot peritään. Kun kanssakäyntitapa on päätetty, saadaan olioryhmät. Sen jälkeen käyttöjärjestelmäprosessi liitetään joka ryhmään ja ohjelmakoodi hahmotellaan. Jaetut oliot tutkitaan synkronisoinnin yhdenaikaisuuden vuoksi ja tapauskohtaiset ratkaisut liitetään.

5.1 Alisysteemin suunnittelu

Alisysteemien suunnitteluvaihe on jatkoa analyysivaiheelle. Se tehdään erikseen jokaiselle alisysteemille.

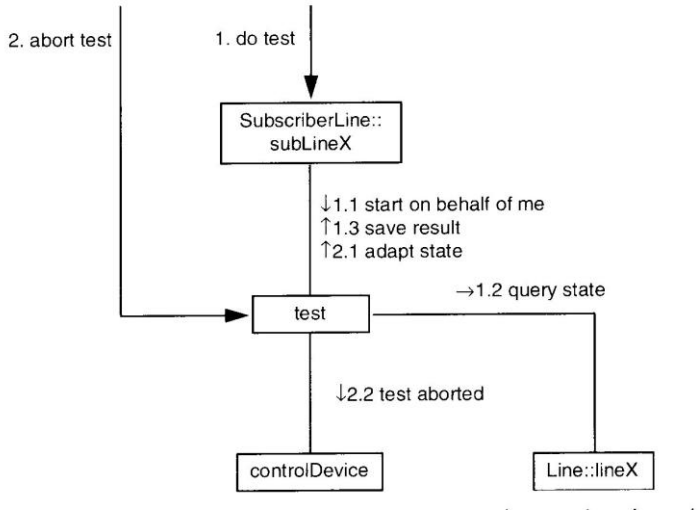
5.1.1 Olioiden suunnittelu

Analyysivaiheessa tuotettu oliomalli kuvaa pääasiassa luokkia. Luokkakaavio ja luokan kuvaustaulu ovat standardi tapa tallentaa informaatiota. Luokkadiagrammi näyttää luokat ja niiden väliset yhteydet. Suunnittelun päätehtävänä on suunnitella olioita. Jokainen suunniteltu olio on esiintymä vastaavasta luokasta. Suunniteltaessa olioita, vaikka kukin olio on esiintymä luokasta, voidaan jokainen tarkasti eritellä. Jokaisen suunnittelun olion tarkoituksena on ymmärtää kokonaissuunnittelu.

5.1.2 Olioiden viestintä

Järjestelmän suunnittelua ei voi toteuttaa riittävästi keskittymällä erikseen yksittäisiin olioihin. On tarpeellista ottaa laajempi näkökanta ja nähdä miten palaset toimivat yhteistyössä. Ennen kuin toteutusvaihe voi alkaa, olioiden viestintä tulee määritellä. Olioiden kanssakäynti tapahtuu pisteestä pisteeseen. Pyynnön esittäjä on asiakas (client) ja vastauksen tai palvelun tuottaja on palvelin (server).

Oliotoimintakaavio esittää yleiskuvan olioiden välisestä toiminnoista. Kuva koostuu laatikoista ja niitä yhdistävistä viivoista. Laatikot ovat suunniteltuja olioita, viivat kuvaavat toimintoja eri olioiden välillä. Nuoli määritelmän vieressä näyttää pyynnön suunnan ja luku toiminnon järjestysnumeron.



Kuva 10: Olioiden kanssakäyntikartta kahdella säikeellä.

Reaaliaikajärjestelmissä toiminnan kannalta voi olla tarpeellista järjestää tietty viive synkronisoinnin kannalta. Tämä kuvataan kaavioon ympyrällä, johon on merkitty toteutettava viive.

Analyysivaiheessa kehitettiin muutamia tilakarttoja kuvaamaan riippuvaisia toimintoja, jotta toiminta selkeytyisi. Octopus-mallissa jokainen toiminta säie voi sisältää tilakartan (statechart). Olioiden kanssakäyntikaaviossa, tilakartta esitetään laatikkona linkitettynä toisiin olioihin. Avainsana tilakartta kuviomäärittelyssä erottaa kartan muista olioista.

Tapahtumasäikeiden suunnittelu on päätehtävä alisysteemien suunnittelussa. Suunnittelu on onnistunut, mikäli kaikki tapahtumat on pystytty kartoittamaan ja merkitsemään. Suunnittelussa käytetään 7 kohtaista listaa. Huomattavaa on, että suunniteltava tapahtuma voi olla perus-, yli- tai yhdistelmätapahtuma.

- *Vaihe 1:* Valitse tapahtuma, jota ei ole vielä toteutettu.
- *Vaihe 2:* Havainnoi olio, joka on mukana prosessissa valitun tapahtuman kanssa.
- *Vaihe 3:* Suunnittele ja tallenna myöhemmät kanssakäynnit.
- *Vaihe 4:* Yhdistä uuden olion säikeet jo olevien kanssa.
- *Vaihe 5:* Toista vaihetta 3 niin kaun kun mukana olevia olioita löytyy.
- *Vaihe 6:* Toista vaihetta 1 kunnes kaikki tapahtumat on tehty.
- *Vaihe 7:* Kertaa ja tasapainota olioiden käyttö ja tilakartta.

5.1.3 Luokan ulkoasu

OCTOPUS-metodissa luokan ulkoasua käytetään tallentamaan suunnitteluyksityiskohtia. Luokan ulkoasu kuvaa olion yleistä toiminnallisuutta ja käyttäytymistä. Suunnitteluvaiheen lopussa luokan ulkoasu on tarkka kuvaus olioiden sisäisistä toiminnoista ja se toimii samalla lähtökohtana toteutusvaiheelle.

Luokan ulkoasu on analyysivaiheen jälkeen pelkkä tekstikopio luokan kaaviosta. Suunnittelussa sitä laajennetaan muuttujilla ja funktioilla. Oliot kommentoidaan varsinaisia ohjelmoijia varten, jotta nämä ymmärtäisivät suunnittelijan ajatuksen. Kommentointi tapahtuu samoin, kuin normaalin ohjelmakoodin kommentointikin. Luokan rakenne on jo hieman varsinaisen ohjelmakoodin mukainen.

```
Class SubscriberLine
  Int number;
  Line Line;          //identifies the corresponding, physical line
  Ref(TestResult) latestResult; // reference to
  Statechart SubscriberLine state; //local, refer to its graph

  function doTest(requestPriority)
    if line.state() != busy //
    then state.Testing = Ongoing; //
      Statechart sltSystem.Request = requestPriority;
      if  test.startOnBehalfOf(this) == connectionFailed
      then state.Testing = None.NoConnection;
        statechart sltSystem = Idle; //
        controlDevice.reportFault(noConnection);
      endif;
    endif;
  endfunction;

  function testFinished(resultUpdate)
    free = latestResult;
    lastResult =  resultUpdate;
    state.Testing = None.Tested
    state.TestResults = Valid;
    return free;
  endfunction;
   …
endclass
```

Listaus: Esimerkkiluokka.

5.1.4 Samanaikaisuuden suunnittelu

Ehdottomassa samanaikaisessa mallissa eri oliot suorittavat tehtäviään äärettömän nopeasti pyydettäessä. Tällä periaatteella Octopus-malli voi olla mm. käyttöjärjestelmä-, ja ohjelmointikieliriippumaton. Kuitenkin Octopus olettaa, että standardia olio-ohjelmointikieltä ja sovittua realiaikajärjestelmää käytetään.

Olioryhmä-käsite on on keskeisessä osassa tätä lähestymistapaa. Tämä termi on määritelty antamalla muodolliset säännöt, kuinka ryhmät muodostetaan. Olioryhmissä prosessien rajat on johdettu yksinkertaisella tavalla. Olioryhmät määrittelevät, miltä osin jokainen olio on osallinen prosessien toimintasäikeisiin.

5.1.5 Synkroninen vs. asynkroninen kanssakäynti

Tapahtumasäikeet kuvaavat kontrollivuon, joka käynnistyy pyynnöstä. Jokainen olio tapahtumasäikeessä voidaan ajatella itsenäisenä ja äärettömän nopeana prosessorina, jota

ohjaavat pyynnöt. Tämä herättää kysymyksen kanssakäyntitavasta. Yleisesti on kaksi erilaista tapaa kanssakäymiseen olioiden välillä.

1) Synkroninen kanssakäynti. Jäsenfunktiota kutsutaan suoraan tai kohdeolion tietoja käsitellään suoraan.

2) Asynkroninen kanssakäynti. Lähettäjä lähettää viestin vastaanottajalle käyttöjärjestelmän kautta ja jatkaa sen suoritusta.

Valittavalla kommunikointitavalla vaikutetaan tapahtumien vastausten saantiin ja järjestelmän toiminnan vakauteen.

5.1.6 Olioryhmät

Normaalisti termi olioryhmä tarkoittaa käsitteellistä liittymää olioista, joilla on synkrooninen kanssakäynti. Huomattavaa on, että olioryhmään kuuluvat sekä oliot että kanssakäyntiviivat. Jatkuva synkronine kanssakäynti muodostaa apupuun: olioryhmä kuuluu yhteen tai useampaan tällaiseen apupuuhun. Olioryhmän ominaisuus on, että se liittää yhden käyttöjärjestelmäprosessin suorituskutsun kulkemaan halki koko alipuun.

5.2 Hardware Wrapperin suunnittelu

Hardware wrapperin suunnittelussa tulee ottaa huomioon sovelluksen loogisten ja primitiivisten tapahtumien erot. Suositeltavaa on tehdä wrapperin sunnittelun 1 kierros ennen varsinaista sovelluksen suunnittelua. Wrapperin suunnittelua tulee täydentää ja päivittää, kun sovelluksen suunnittelu selventää yksityiskohtia.

Järjestelmän keskeytysrutiini voidaan suorittaa miltei milloin tahansa. Jos tapahtuma on luokiteltu ns. “kovaksi keskeytykseksi”, alullepanija on keskeytyspalvelurutiini. Tapahtuman ollessa ns “pehmeä”, alullepanija on käyttöjärjestelmän ajastin. Molemmat alullepanijat kuuluvat hardware wrapperille ja piirrettäessä tapahtumasäikeen vuorovaikutuskaaviota ne näytetään normaalissa oliolaatikossa varustettuna luokan nimellä.

Vuorovaikutuksesta voidaan tehdä asynkroninen tai synkroninen. Synkroninen vuorovaikutus toteutetaan kutsumalla kontrolloivan olion jäsenfunktiota. Asynkroninen käyttää käyttöjärjestelmän toiminnallisuutta.

Keskeytyksistä alkavia toimintoja kuvataan nuolin. Numerot pohjustavat järjestystä.

6 Loppusanat

Ohjelmistotuotannon voidaan katsoa koostuvan neljästä tekijästä: ihmisten, tuotteen, projektin ja prosessin hallinnasta [Pressman]. Jälkikäteen voidaan vastaavasti katsoa käytetyn metodologian olleen konfliktissa kaikkien osatekijöiden kanssa.

FAS:n käyttämä prosessi ei soveltunut inkrementaaliseen suunnitteluun. Viiteen virstanpylvääseen (milestones) jaettu prosessimalli oli perusteltu lähinnä henkilökohtaisen tulospalkkauksen takia [Adams]. Todellisuudessa laajan uudentyyppisen ohjelmistoprojektin toteuttaminen vanhasta periyttämällä ei onnistunut kitkatta. Toinen merkittävä asia ohjelmistoprosessissa oli yhden ihmisen vastuualueen määrittely moduleittain, jolloin sama henkilö oli vastuussa alijärjestelmän vaatimusmäärittelystä, suunnittelusta, toteutuksesta ja testauksesta. Octopus-metodologiaa voidaan teoriassa seurata toteutusvaiheessa henkilön *tietämättä* – tämä pelkästään hyvän suunnittelun turvin. Todellisuudessa valtaosa suunnittelustakin toteutui metodologiaa tuntematta, koska se oli joka adoptoitu sellaisenaan aiemmasta tuotteesta tai vastuuhenkilö ei ollut hyötynyt riittävästi octopus-koulutuksesta. Tämä on sopusoinnussa ohjelmistotuotannon keskeisimmän haasteen kanssa: miten tehdä hyvä tuote ei aivan parhaalla henkilöstöllä. Haasteellisen metodologian seuraaminen aiheutti konflikteja koodausorientoituneiden ihmisten henkilökohtaisten päämäärien kanssa useita vuosia kestäneessä projektissa. Suunnittelu ja toteutus degeneroitui code-and-fix metodologiaksi ja vuoden 98 lopussa projekti olikin jo toista vuotta aikataulusta jäljessä.

Vaikka OCTOPUSta on käytetty tekijöiden mukaan menestyksellä vakaiden ja luotettavien tietoliikennesovellusten suunnittelussa, NAN perhe ei kuulunut tähän ryhmään. Jos monien vuosien kokemukset reaaliaikajärjestelmien suunnittelusta ovatkin antaneet oman osuutensa octopus-metodiin, on hyvinkin mahdollista, ettei metodi viime kädessä ollut eduksi suunniteltavalle tuotteelle.

Octopus-metodin tarkoitus oli poistaa kuilu laitteistoläheisten reaaliaikajärjestelmien ja oliokeskeisten tekniikoiden väliltä – missä tehtävässä metodologia todellakin osoitti menestyksensä. C++ osoittautui luontevaksi toteutuskieleksi reaaliaikajärjestelmässä vastoin useimpien ammatilaisten ennakko-odotuksia. Lisäksi voidaan huomioida metodin kehityksen yhdessä KISS case-työkalun kanssa tuoneen arvokkaita visualisointivälineitä rinnakkaisten järjestelmien dokumentointiin.

7 Lähteet

Awad, Kuusela, Ziegler: Octopus-tutorial saatavilla html-muodossa <URL: http://www-nrc.nokia.com/octopus/tutorial/index.html>

Awad, Kuusela, Ziegler: Object-Oriented Technology for Real-Time Systems, Prentice Hall, 1996

Scott Adams: Dilbert Principle

Roger S. Pressman: Software Engineering, McGraw Hill Higher Education, 2000

William Brown, etc.: Anti-Patterns, Wiley&Sons, 1998