LAPPEENRANNAN TEKNILLINEN YLIOPISTO
Tietotekniikan osasto

Kuvausmenetelmän valinta ja käyttöönotto ohjelmistotuoteorganisaatiossa

Diplomityön aihe on hyväksytty tietotekniikan osastoneuvostossa 11.10.2006

Diplomityön ohjaajana on toiminut Juha Sonck
Diplomityön ohjaavana professorina ja ensimmäisenä tarkastajana on toiminut
Professori Kari Smolander ja toisena tarkastajana DI Alexandre Bern

Espoossa 23.2.2007

Seppo Iivonen
Meriusva 5 A 40
02320 ESPOO
050-3074541

TIIVISTELMÄ

Tekijä:	Seppo Iivonen
Työn nimi:	**Kuvausmenetelmän valinta ja käyttöönotto ohjelmistotuoteorganisaatiossa**
Osasto:	Tietotekniikan osasto
Vuosi:	2007
Paikka:	Espoo

Diplomityö. Lappeenrannan teknillinen yliopisto. 66 sivua ja 8 kuvaa.

Tarkastajat:	Prof. Kari Smolander DI Alexandre Bern
Avainsanat:	Kuvausmenetelmä, uudelleendokumentointi, uudelleensuunnittelu, käänteinen suunnittelu

Useiden pitkän kehityskaaren ohjelmistojen ylläpitäminen ja kehittäminen on vaikeaa, sillä niiden dokumentaatio on vajaata tai vanhentunutta. Tässä diplomityössä etsitään ratkaisua tällaisen ohjelmiston ja sen taustalla olevan järjestelmän kuvaukseen. Tavoitteina on tukea nykyisen ohjelmiston ylläpitoa ja uuden työvoiman perehdyttämistä. Tavoitteena on myös pohjustaa uuden korvaavan ohjelmiston suunnittelua kuvaamalla nykyiseen järjestelmään sitoutunutta sovellusalueosaamista. Työssä kehitetään kuvausmenetelmä järjestelmän kuvaamiseen hierarkkisesti laitteistotason yleiskuvauksesta ohjelmiston luokkarakenteeseen sekä toiminnallisuuteen asti. Laite- ja luokkarakennekuvaukset ovat rakenteellisia kuvauksia, joiden tehtävänä on selittää järjestelmän ja sen osien kokoonpano. Toiminnallisuudesta kertovat kuvaukset on toteutettu käyttötapauskuvauksina. Työssä keskityttiin erityisesti kohdejärjestelmän keskeisen ohjelmiston ja tietokannan kuvaamiseen. Ohjelmistosta valittiin tärkeimmät ja eniten sovellusalueen tietotaitoa sisältävät osat, joista työssä luotiin esimerkkikuvaukset. Kuvauksia on kehitettyä menetelmää hyödyntäen helppo laajentaa tarpeiden mukaan paitsi ohjelmiston muihin osiin, myös laitteiston ja järjestelmän kuvaamiseen kokonaisuudessaan syvemmin.

ABSTRACT

Author: Seppo Iivonen

Title: Redocumentative Methods in a Software Product Development Organization

Department: Department of Information Technology

Year: 2007

Location: Espoo, Finland

Master's thesis. Lappeenranta University of Technology. 66 pages and 8 pictures.

Supervisors: Prof. Kari Smolander
M.Sc. Alexandre Bern

Keywords: Redocumentation, reengineering, reverse engineering

Insufficient or out of date documentation in software engineering can lead to a state where maintaining and developing a software product with long life cycle becomes difficult. Using a software product in similar situation as a target, this thesis is trying to find a solution to the problem. The objectives include supporting the maintenance of the current software product and familiarizing new developers with the product. Also, the development of a new software product to replace the existing one should be supported by including the product-enclosed domain knowledge in the documentation models. In this thesis, a redocumentative method of a hierarchical structure is being developed with visual and verbal models of the physical system, the database and the software. The physical structure documentation with software class diagrams forms the structural documentation of the system. The activity of the system is documented with use-case diagrams. The thesis concentrates especially on the central software and database structure of the system. The most significant parts of the software holding the domain knowledge were selected to produce sample models with the method created. The models are easy to expand and deepen as needed to cover the whole system including both its software and hardware.

KÄYTETYT LYHENTEET JA TERMIT

Lyhenteet

ADO	ActiveX Data Objects
BDE	Borland Database Engine
CASE	Computer-aided Software Engineering
RFID	Radio Frequency Identification
RIGHT	Finding the Right Context in Globalizing Software Development
SQL	Structured Query Language
SysML	Systems Modelling Language
UML	Unified Markup Language

Termit

Analyysimallin selvitys	Analysis recovery
Edestakainen suunnittelu	Roundtrip engineering
Etenevä ohjelmistosuunnittelu	Forward engineering
Käänteinen suunnittelu	Reverse engineering
Peritty järjestelmä	Legacy system
Suunnittelumallin palauttaminen	Design recovery
Toteutuksen selvitys	Implementation recovery
Uudelleendokumentointi	Redocumentation
Uudelleenrakentaminen	Restructuring
Uudelleensuunnittelu	Reengineering

SISÄLLYSLUETTELO

1 JOHDANTO 5
2 KOHDEJÄRJESTELMÄ JA TAVOITE 6
2.1 Työajan- ja materiaalihallinnan järjestelmä 7
2.1.1 SpringSoft-ohjelmiston tehtävä ja luonne 9
2.1.2 SpringSoft-ohjelmiston historia ja kehitysvaiheet 10
2.1.3 Nykyinen dokumentointi ja kuvaukset 13
2.2 Kuvauksen tavoitteet 14
3 KUVAUKSEN POHJUSTAMINEN JA PERUSTIEDOT 18
3.1 Etenevä ohjelmistosuunnittelu ja kuvaukset 21
3.2 Käänteisen ohjelmistosuunnittelun menetelmät 26
3.2.1 SpringSystem-järjestelmän kuvauksen rakenne 29
3.2.2 Rakenteen kuvaus 32
3.2.3 Käyttötapauskuvaukset ja toiminnallisuuden kuvaus 35
3.2.4 CASE-työkalut ja sovelluskehitin 38
3.2.5 Visuaaliset kuvausmenetelmät 40
3.3 Järjestelmän kuvausmenetelmät 42
3.4 Tietokannan kuvausmenetelmät 43
4 KOHDEJÄRJESTELMÄN KUVAUSMENETELMÄ 46
4.1 Kuvauksen osat ja kohdejärjestelmän mallikuvaukset 48
5 POHDINTAA 59
6 YHTEENVETO 62
LÄHDELUETTELO 64

1 JOHDANTO

Kaupallisessa ohjelmistotuotannossa ohjelmiston suunnittelua, kehitystä ja käyttöönottoa seuraa aina ylläpito. Ylläpidolle on yleensä sitä enemmän tarvetta, mitä monipuolisempi ja monimutkaisempi ohjelma on. Ylläpito sisältää ohjelman käytön avustamista, ohjelmasta löytyvien virheiden korjaamista sekä ohjelman edelleen kehittämistä. Näitä tehtäviä voi auttaa ohjelman rakenteen ja toiminnallisuuden kuvaaminen sopivalla tasolla. Monissa nykyaikaisissa ohjelmistoprojekteissa ohjelmien kuvaukset syntyvät jo ohjelmiston suunnittelu- ja kehitysvaiheissa, mutta toisaalta joissain ohjelmistoprojekteissa kuvauksia ei ehkä ole tai varsinkin pitkäaikaisissa projekteissa ne ovat voineet vanhentua.

Tässä diplomityössä käsitellään laajaa, pitkän kehityskaaren työajan- ja materiaalinseurannan järjestelmää, jonka dokumentaatio on jäänyt kehitys- ja suunnitteluvaiheissa vähemmälle huomiolle. Järjestelmän pääohjelmistoa on kehitetty yli 10 vuotta kehittämällä ja liittämällä tarpeen mukaan uusia toimintoja vanhaan ydinjärjestelmään. Tämä on johtanut ohjelmiston sisäisen rakenteen monimutkaistumiseen ja ylläpidon vaikeutumiseen. Uuden korvaavan pääohjelmiston kehitystä suunnitellaan, mutta nykyinen ohjelmisto kehittyy vielä ainakin 5 vuoden ajan. Nykyinen ohjelmisto sisältää paljon arvokasta toimialaosaamista, joka haluttaisiin kerätä talteen kuvausten muodossa sekä uuden järjestelmän suunnittelun, että olemassa olevan järjestelmän ylläpidon hyväksi. Tässä työssä valitaan ja kehitetään tarkoitukseen sopivia kuvausmenetelmiä ja luodaan kuvaus tärkeimmistä ohjelman osista. Kehitettyä kuvausmenetelmää on tarkoitus hyödyntää jatkossa kuvauksen laajentamiseen sekä mahdollisen uuden järjestelmän tai ohjelmiston kuvaamiseen.

Työn teksti jakautuu kolmeen pääosioon. Toisessa kappaleessa esitellään aluksi kohdejärjestelmän rakenne ja tehtävät sekä työn tavoitteet. Kappaleessa kolme perehdytään kuvausmenetelmän valintaan ja kehittämiseen vaikuttaneisiin tutkimuksiin ja teorioihin. Viimeisessä osassa kappaleessa neljä kuvaillaan varsinainen käyttöönotettu kuvausmenetelmä ja annetaan esimerkkejä tehdyistä kuvauksista.

2 KOHDEJÄRJESTELMÄ JA TAVOITE

Tämän diplomityön kohteena on teollisen tiedonkeruun SpringSystem-järjestelmä. Kokonaiseen järjestelmään voi asiakkaan tarpeiden sanelemana kuulua mm. henkilöstön työajanseurantaa, kulkuoikeuksien hallintaa, materiaalivuon valvontaa ja materiaalin merkintää RFID- (Radio Frequency Identification) ja viivakooditekniikoita hyödyntäen. Vaikka tässä työssä kuvataan yleisellä tasolla myös kokonaisen järjestelmän rakennetta, keskittyvät työn kuvaukset eniten järjestelmän ytimessä olevan SpringSoft-ohjelmiston tärkeimpien osien kuvaamiseen. SpringSoft on pitkän kehitystyön tulos, josta tässä työssä käytetään uusinta 6.12 versiota. Tarkoituksena on kehittää kuvausmenetelmä järjestelmä- ja ohjelmistotasojen kuvaamiseen sekä luoda mallikuvaukset järjestelmän olennaisimmista osista. Kehitettyä menetelmää käyttäen kuvausta voidaan tämän työn ulkopuolella laajentaa tarpeiden mukaan. Kuvauskohteena oleva järjestelmä esitellään kappaleessa 2.1. ja kuvausten tavoitteista kerrotaan tarkemmin kappaleessa 2.2.

Työ on osa Lappeenrannan teknillisen yliopiston ja Etelä-Karjalan ammattikorkeakoulun RIGHT-projektia (Finding the Right Context in Globalizing Software Development). Tietotekniikan tuotanto ja palvelut keskittyvät yhä enemmän kansainvälisiin ratkaisuihin ja toiminnan globalisointiin, mikä painostaa myös ohjelmistotuotantoa kehittymään kohti ulkoistetun ja jopa kansainvälisen kehitysringin hyödyntämistä. Myös aidosti virtuaalista organisaatiotyyppiä edustavien avoimen lähdekoodin projektien nopea kehitys on aiheuttanut painetta perinteiselle ohjelmistotuotannolle. RIGHT-projektissa tutkitaan näiden muutospaineiden vaikutusta suomalaisiin ohjelmistotuoteorganisaatioihin.

Edullisen työvoiman maiden hyödyntäminen ohjelmistotuotannossa kiinnostaa yrityksiä kustannusten karsimisen vuoksi. Ohjelmistotuotannon ulkoistaminen vaatii kuitenkin monelta yritykseltä paitsi uudelleenorganisointia, myös kehitystä kohti hyvin määriteltyä ja suunnitelmapohjaista prosessimallia, jonka turvin ohjelmistokehityksen avaintekijöistä voidaan sujuvasti keskustella sidosryhmien välillä. Sovellusten

vaatimuksista ja teknisistä yksityiskohdista pitää kyetä sujuvasti ja tyhjentävästi kommunikoimaan ulkoistetun – usein kansainvälisen – ohjelmoijaryhmän kanssa.

Tämä diplomityö perehtyy kehityskaarensa ylläpitovaiheessa olevan ohjelmistotuotteen kuvaamiseen. Vaikka suurimpana tavoitteena kuvaukselle on uuden, korvaavan ohjelmiston kehityksen pohjustaminen, ovat ulkoisen työvoiman mahdollisuus ja tarpeet vahvasti mukana. Alustavia keskusteluja on käyty myös ulkomaille ulkoistetusta ohjelmointityöstä, mikä asettaa tarpeet nykyisen järjestelmän dokumentoinnista ja hallinnasta uudelle tasolle. Vanhan ohjelmiston tiettyjä suurta työvoimaa vaativia muutoksia saatetaan teettää ulkopuolisilla työntekijöillä yrityksen oman erityisosaamisen keskittämiseksi todelliseen ylläpito- ja kehitysideointityöhön. Uuden ohjelmiston ohjelmointi saattaa puolestaan jo kokonaisuudessaan tapahtua ulkoistetusti, mikä asettaa kuvausmenetelmälle vaatimuksen tietotaidon siirtämisestä nykyjärjestelmästä kuvaukseen. Vanhan ohjelmiston kuvauksia pohjana käyttämällä uutta ja ulkoistettua työvoimaa voidaan helpommin perehdyttää uuden ohjelman tarpeisiin.

2.1 Työajan- ja materiaalihallinnan järjestelmä

SpringSystem-järjestelmä voi laajimmillaan koostua henkilöstön kulkua valvovista sähkölukoista, työajan leimaamiseen käytettävistä käytävä- ja seinäpäätteistä, materiaalin jäljitykseen käytettävistä RFID- ja viivakoodilukijoista, tarratulostimista tarra-asettimineen eli applikaattoreineen sekä järjestelmän hallintaan käytettävistä työasemista. Lisäksi keskeinen ohjelmisto sekä tietokanta ovat asennettuina yhdelle tai useammalle palvelimelle. Järjestelmätoimitukseen kuuluu myös ohjelmistot järjestelmän valvontaan ja työajanhallintaan sekä tarvittavat ajuri- ja käyttöliittymäohjelmat päätelaitteille, lukijoille ja applikaattoreille. Vaikka laajimmillaan järjestelmä voi siis olla hyvinkin laaja, on toimitettava lopputuote aina asiakkaan tarpeiden mukaisesti koottu. Pienempiin toimituksiin voi kuulua esimerkiksi vain työajanseurantaan tarvittavat laitteet ja ohjelmistot.

Järjestelmä on tiedonkeruuluonteensa ansiosta fyysisesti hajautettu ja samoin ohjelmiston osia on sijoitettuna fyysisesti eri paikkoihin. Myös hallintaan käytetty pääohjelmisto SpringSoft on sisäisesti hajautettu moduuleiksi toiminnallisuuksien mukaan. Ohjelmiston tuorein julkaistu versio 6.12 koostuu pääohjelmasta sekä kuudesta siihen saumattomasti liittyvästä osamoduulista. Näiden lisäksi ohjelmaan voidaan liittää asiakaskohtaisia moduuleita tarpeiden mukaan. Eri työasemien SpringSoft-ohjelmistot ovat lähiverkon välityksellä yhteydessä palvelimeen, joka sisältää mm. laskennasta huolehtivan SpringServer-ohjelmiston sekä Microsoft SQL Server- tai Oracle-tietokannan. Lähiverkon kautta järjestelmään kytkeytyvät myös tietoa syöttävät lukijat ja päätelaitteet.

Tämän työn pääkohteena olevan ohjelmiston kehittäjänä toimii pieni, noin viiden hengen kehitysryhmä. Koko yrityksen yli 60 hengen organisaatiosta kehitysryhmä on suhteellisen pieni osa. Ryhmän jäsenet ovat toimineet projektin parissa jo useita vuosia aina ohjelmiston ja koko järjestelmän edellisistä sukupolvista asti. Ryhmän kehittäjät ovat erikoistuneet kukin omaan osaansa koko ohjelmistossa ja pyrkivät usein pidättäytymään tässä työnkuvassa. Myös nykyisen ohjelmiston perussuunnittelun ja arkkitehtuurin 90-luvun puolivälissä tehnyt kehittäjä on mukana ryhmässä. Pääsuunnittelijan tietotaito ohjelmistosta on arvokas tuki koko organisaatiolle kehitysryhmästä järjestelmätoimituksiin.

Yrityksen ohjelmistokehityksessä ei ole noudatettu tiettävästi mitään yleisesti tunnettua ohjelmistotuotannon menetelmää, vaan kehitysryhmän toiminta on muotoutunut ajan kanssa nykymuotoonsa. Sekä ohjelmiston kehitys että dokumentointi on ollut pitkälti asiakasvetoista, eli ohjelmisto on kehittynyt uusien ja vanhojen asiakkaiden tarpeiden ja toiveiden mukaan. Ohjelmiston kehittämisestä on niin ikään tehty kirjallista dokumentointia lähinnä asiakasprojektien tarpeisiin. Yrityksen sisällä kehitystä ohjataan omalla projektinhallintaohjelmistolla johon havaitut järjestelmän toimintavirheet ja kehitysideat kirjataan testaajien, käyttäjien ja kehittäjien itsensä toimesta sekä hiljattain käyttöön otetulla versionhallintajärjestelmällä, jolla hallitaan ohjelmien muutoksia lähdekoodin tasolla.

2.1.1 SpringSoft-ohjelmiston tehtävä ja luonne

Kuvauksen pääkohteena oleva SpringSoft on teollisen tiedonkeruun ja merkinnän ohjelmisto, joka auttaa hallitsemaan yrityksen henkilöstöä ja materiaalivirtoja. Ohjelmiston avulla voi hallita monipuolisesti organisaation henkilöiden työvuoroja ja vuorokalentereita, työ- ja tuntikirjauksia, palkkalajeja sekä kulkuoikeuksia. Saman ohjelmiston avulla tapahtuu teollinen materiaalinhallinta ja -jäljitys. Haikalan ja Märijärven [1] esittämän ohjelmistojen luokittelun perusteella SpringSoft voidaan luokitella kaupallis-hallinnolliseksi ohjelmistoksi, joka sisältää myös prosessinohjaustoiminnallisuutta. Pressmanin [2] esittämän rakenteellisen jaon perusteella SpringSoft on arkkitehtuuriltaan tietokeskeistä arkkitehtuuria edustava ohjelmisto, jonka keskeisenä osana toimii tietokanta.

SpringSoft ja sen taustalla palvelimella toimiva laskentasovellus SpringServer ottavat vastaan tietoa useilta lukijoilta ja päätteiltä, suorittavat tarvittavia laskentatoimenpiteitä ja välittävät informaation edelleen erilaisille kolmannen osapuolen tuotannonohjaus- ja palkanlaskentajärjestelmille. Ohjelmiston ja koko järjestelmän tuotteistusaste [1] on matala. Järjestelmä räätälöidään asiakkaan tarpeiden mukaan ja usein myös ohjelmistoon tehdään asiakaskohtaisia muutoksia. Ohjelmisto kehittyy jatkuvasti muuttuvien työehtosopimusten vaatimusten sekä asiakkaiden erilaisten tarpeiden mukaan.

Räätälöinti ja järjestelmän räätälöitävyys ovat kohdejärjestelmälle elinehtoja. Asiakaspiirin muodostavien teollisuusyritysten tarpeet eroavat toisistaan paljon mm. henkilöstön määrän, alakohtaisten työehtosopimusten ja työtapojen osalta. Räätälöitävyyden tulee näin ollen ulottua järjestelmän ja ohjelmistojen sisäiseen toimintaan asti. SpringSoftissa räätälöinti on pääosin pyritty toteuttamaan parametroinnilla. Ohjelmiston käyttämä tietokanta sisältää oman taulun ohjaustiedoille, joilla voidaan asettaa monipuolisia parametreja aina Boolean-tyyppisistä valinnoista ohjelmiston ja tiedon rakenteisiin vaikuttaviin SQL-lauseisiin (Structured Query Language) asti. Ennen ohjelmiston moduulijakoa parametrointi oli myös ainoa keino erottaa asiakkaat toisistaan ohjelmiston sisällä; ohjelmiston

ajotiedosto sisälsi kaikki asiakaskohtaiset ratkaisut, joista parametrein valittiin käyttöön toivotut ominaisuudet. Nykymuotoisena osa asiakaskohtaisista toiminnoista on siirretty omiksi moduuleikseen, jolloin toimitettavan ohjelmiston kokoa on voitu pienentää ja ohjelmiston rakennetta ja sisältöä täsmentää. Parametrointi on silti tarpeen paitsi perusohjelmiston ominaisuuksien, myös asiakaskohtaisten moduulien ominaisuuksien määrittelemiseksi. Myös tietokantaa räätälöidään käyttämällä asiakaskohtaisia näkymiä ja tapahtumista aktivoituvia toimintoja eli triggereitä.

Ohjelmiston vasteaika- ja reaaliaikaisuusvaatimukset ovat merkittävät. Kun työntekijä leimaa työmerkinnän, on hänen kyettävä todentamaan paitsi merkinnän onnistuminen myös sen vaikutus työhön. Käyttöliittymän näkökulmasta ajatellen perinteistä korttileimausta käyttävä henkilö tuntee ja näkee leiman syntymisen konkreettisesti reaaliajassa. Sähköisen järjestelmän olisi kyettävä samaan. Luotettavuus on myös tärkeä tekijä; yhtään merkintää ei saa hukata. Sama vaatimus tulee paremmin esiin materiaalimerkinnöissä, joissa viiveet voisivat johtaa virheellisiin päätelmiin vaikkapa tuotteen kokoonpanossa tai prosessinohjauksessa. Tällaisten virheiden mahdollisuus on huomioitu ohjelmiston toiminnassa, eikä esim. verkkoyhteyden katkeaminen johda heti ongelmiin. Lukijalaitteissa on puskurimuistit merkintöjen väliaikaiseen tallentamiseen yhteyden korjaamisen ajaksi ja tietoa käyttävissä ohjelmissa on toiminnot uudelleenhaulle ja -laskennalle, joilla mahdollisten puuttuvien merkintöjen myötä väärin lasketut tiedot voidaan korjata.

2.1.2 SpringSoft-ohjelmiston historia ja kehitysvaiheet

Ohjelmiston ja sen taustalla olevan järjestelmän laajuus kertoo paljon ohjelmiston kehityksestä ja monipuolisuudesta. Ohjelmiston laajuutta voidaan mitata useilla eri mittareilla, kuten koodirivien määränä, ohjelmiston tai lähdekoodien vaatimana tallennustilana, näyttöjen lukumääränä tai tietokannan kokona ja aktiivisuutena [1]. Työn pääkohteena oleva SpringSoft-ohjelmisto koostuu noin 250 000 ohjelmakoodirivistä ja vaatii käännettynä ja toimintavalmiina kokoonpanosta riippuen 10-60 Mt levytilaa. Tämä määritelmä on kuitenkin kohdejärjestelmälle heikko

ohjelmiston kokoonpanon vaihdellessa paljon eri toimitusten mukaan. Joillain asiakkaista ei esimerkiksi ole tarvetta henkilöstön kulunvalvonnalle tai materiaalin jäljitykselle, mikä muuttaa toimitettavan järjestelmän kokoa. Sama pätee SpringSoftin näyttöjen lukumäärään; eri asiakkaiden erilaiset vaatimukset järjestelmän toimintojen suhteen rajaavat näyttöjen lukumäärän muutamasta kymmenestä yli viiteenkymmeneen.

Järjestelmän ytimessä oleva tietokanta niin ikään vaihtelee paitsi rivien ja aktiivisuuden, myös rakenteensa osalta. Koska esimerkiksi järjestelmällä valvottavien henkilöiden määrä voi vaihdella muutamista kymmenistä työntekijöistä tuhansiin, vaihtelee tietokannankin aktiivisuus muutamasta sadasta päivämerkinnästä jopa kymmeneen tuhanteen merkintään päivässä. Kokonaisuudessaan SpringSystem-järjestelmä koostuu useista laitteista ja ohjelmistoista. Yleensä laitteistotoimituksien laitemäärät ovat parissa kymmenessä laitteessa. Joillain asiakkailla voi olla jopa kymmeniä toimipisteitä ja tällöin myös järjestelmätoimitusten koko voi kasvaa.

Kohteena olevaa ohjelmistoa on kehitetty Borland [3] Delphi Object Pascal kielellä 1990-luvun puolesta välistä alkaen. Kohdeohjelmiston tapauksessa on aiheellista puhua kahdesta merkittävästä aikakaudesta. Ohjelmiston ensimmäinen kehitysvaihe ajoittuu 90-luvun puolivälistä noin vuoteen 2004. Tuona aikana ohjelmisto tunnettiin nimellä SpringSystem.exe. Vuodesta 2004 alkaen ohjelmaa alettiin siirtää vanhentuneesta BDE-tietokantarajapinnasta (Borland Database Engine) ADO-rajapintaan (ActiveX Data Objects) ja samalla ohjelmiston rakennetta järjesteltiin modulaarisemmaksi. Ohjelmiston suunnittelussa voi toisen merkittävän aikakauden laskea alkaneen tästä uudelleenjärjestelystä.

Järjestelmän alkutaipaleen, ensimmäisten noin 10 vuoden aikana ohjelmisto kehittyi pääsääntöisesti rakenteeltaan samanlaisena, eikä täysin uusia ratkaisuja tuon kehityskaaren aikana tehty. Tässä muodossaan SpringSystem.exe tuotenimeä kantanut ohjelmisto ehti saavuttaa hyvin vakaan perusmuodon, joka ei ole vaatinut suuria muutoksia edes laitekannan muutosten myötä. Kyseinen ohjelmiston versio on vielä tällä hetkellä yleisin käytössä oleva versio, sillä uudemman modulaarisen, ADO-

rajapintaa hyödyntävän, ratkaisun testaus oli vielä kesken vuoden 2006 syksyllä. SpringSystem.exen koko ja laajuus kasvoi myös pitkän kehityskaaren uusien ominaisuuksien myötä, mutta kokonaisuus säilyi silti hallinnassa. Tämä on ollut mahdollista, sillä ohjelmointityöstä on alusta asti vastannut pääosin sama ja suhteellisen pieni ryhmä ohjelmoijia. Jokaisella ohjelmoijalla on vankka tuntuma kokonaisuuden perusperiaatteista sekä erinomainen tietotaito omista osaprojekteistansa.

Ohjelmiston uudempi versio on rakennettu modulaariseksi niin, ettei yhteen toimintoon tehtävä muutos vaadi koko ohjelmiston uudelleentoimitusta asiakkaalle. Modulaarisuus toteutettiin käytännössä samalla, kun ohjelmiston lähdekoodi oli käytävä järjestelmällisesti läpi uuteen tietokantajärjestelmään siirtymisen myötä. Myös ohjelmiston päänäytön ulkoasu ja käytettävyys kokivat pieniä kohennuksia, mutta käyttäjälle näkyvät toiminnot ovat samoja ja käyttävät taustalla tietokannan rajapintakonversiota lukuun ottamatta samaa lähdekoodia kuin aiemminkin.

Koko lähdekoodin hidasta läpikäyntiä ja paikoin yksinkertaista etsi ja korvaa työtä vaatineet tietokannan rajapintakonversio sekä modulaarisuuden toteutus tapahtui yrityksen omaa työvoimaa käyttäen osittain siksi, ettei vanhasta järjestelmästä ollut saatavilla riittävää dokumentaatiota ulkopuolisen työvoiman hyödyntämiseksi. Dokumentointi koko järjestelmäkokonaisuudesta on vaillinaista ja usein tietotaito on sitoutunut suoraan järjestelmän suunnittelijoihin ja toimittajiin. Järjestelmätason dokumentteja joistain yksittäisistä ratkaisuista on tehty yleensä asiakkaiden vaatimusten kannustamina. Ohjelmiston osalta sama pätee sekä vanhaan, että uuteen versioon. Rakennedokumentteja tai toimintakaavioita ei ole kuin muutamista hajanaisista ohjelman osista, eikä kehitystyö perustu mihinkään kirjoitettuun ohjeeseen. Jokainen ohjelmoija voi tehdä työssään hyvin itsenäisiä ratkaisuja ja käyttää yhteistä ohjelmointikieltä ja sisäisiä peruskomponentteja suhteellisen vapaamuotoisesti. Tilanne on johtanut vahvoihin riippuvuuksiin tiettyjen ohjelmoijien työpanoksesta. Toisaalta näin lähdekoodi on sen kirjoittaneelle tuttua ja oman tuntuista myös ulkoasultaan, mikä voi vaikuttaa positiivisesti työmoraaliin ja työn nopeuteen.

SpringSoft-ohjelmiston uudenkin version puutteet näkyvät jo. Järjestelmän ohjelmointiin pääsääntöisesti käytetty Borland Delphi 7 ei tue Unicode-merkistön käyttöä, mikä käytännössä rajoittaa järjestelmän markkinat maihin, jotka käyttävät latinalaista merkistöä. Kaikkia järjestelmään kuuluvia ohjelmistoja, mukaan lukien pääohjelmisto SpringSoftia, ei ole toteutettu alun perinkään kielen vaihtamisen ehdoilla. Ongelmia tuottaa tässä mielessä myös alkuperäisenä säilynyt tietokantarakenne, jonka taulujen kääntämistä ei ole huomioitu ja jonka peruskielenä on suomi.

2.1.3 Nykyinen dokumentointi ja kuvaukset

Laadukkaiden dokumenttien tuottaminen on yleisesti ohjelmistotuotannossa käytännön ohjelmointityön heikoimpia lenkkejä, vaikka toki itse ohjelmakoodikin on ajateltavissa dokumentaationa. Aikataulupaineessa dokumentit usein jäävät tekemättä tai hyvin pinnallisiksi. Jos esimerkiksi määrittely- ja suunnitteludokumentit puuttuvat, voi ohjelmiston rakenteeseen tarvittavat muutokset osoittautua niin työläiksi, että koko ohjelmiston uudelleensuunnittelu tulee ajankohtaiseksi. Myös testauksen aikana tehtävä dokumentaatio tuo työnsäästöä seuraavissa samaan ohjelmistoon liittyvissä projekteissa. Dokumenttityypit on jaettavissa kolmeen pääryhmään: laatukäsikirjaan, projektinhallintaan ja tuotedokumentointiin liittyviin dokumentteihin. Laatukäsikirjaan liittyviä dokumentteja ovat lähdekoodin tyyliopas, työtapojen ohjeistukset, dokumenttimallit sekä yrityksen laatujärjestelmän tuottamat dokumentit. Hyvin dokumentoidussa projektissa projektinhallinta tuottaa dokumentteja koko projektin elinkaaren ajan suunnitelmista ja seurannasta loppuraportteihin. Tuotedokumentointiin kuuluu projektikohtaiset sekä tuotekohtaiset käyttöohjeet sekä muut lopputuotteen dokumentit. [1]

Tämän työn kohteena olevan järjestelmän kehitys on tapahtunut pääasiassa asiakkaiden tarpeiden myötä. Uusi asiakas tuo mukanaan uusia tarpeita ja vanhojen asiakkaiden tarpeet kehittyvät ajan myötä. Näin ollen suunnitteludokumentointi on asiakaskohtaista, vaikka lopputuote on kaikille asiakkaille pääsääntöisesti sama.

Suunnitteludokumentaation kattavuus määräytyy usein asiakkaan laatuvaatimusten mukaan, jolloin vaihtelua dokumentoinnissa on paljon. Lisäksi, kun dokumenteista vastaa eri projektien vastuuhenkilöt eikä dokumenttimalleja ole, eroavaisuudet dokumenttien sisällössä ja jopa ulkoasussa ovat suuria. Ohjelmiston toiminnallisuuden ja rakenteen erot asiakkaiden välillä toteuttavan parametroinnin dokumentoinnista jokainen ohjelmiston kehittäjä on vastuussa. Haikalan mainitsemia laatukäsikirjan työohjeita ja malleja ei ole lähdekoodin osalta SpringSoft-ohjelmiston kehityksessä käytetty. Kehitysympäristö sen sijaan on dokumentoitu asetusten ja kolmannen osapuolen komponenttien osalta niin, että kehittäjillä pääsääntöisesti on käytössään samat asetukset koneillaan. SpringSoftin sisäinen versiointi puolestaan auttaa varmistamaan, että testaajilla ja käyttäjillä on käytössään juuri ne osat ohjelmistosta jotka kuuluvat yhteen.

2.2 Kuvauksen tavoitteet

Tässä diplomityössä tuotettavalle kuvausmenetelmälle ja kuvaukselle on asetettu tavoitteita SpringTime Oy:n toimesta ajatellen sekä järjestelmän nykytilaa, että tulevaisuuden kehitystä. Näistä tavoitteista kerrotaan tässä kappaleessa enemmän. Tavoitteet ovat täysin kuvauksen sisältöön ja hyötyihin keskittyviä, eikä yrityksellä ole näkemystä kuvausten lopullisesta muodosta. Tavoitteena on kuitenkin luonnollisesti tehdä kuvauksista mahdollisimman luettavia ja ymmärrettäviä. Kuvausten on myös tarkoitus sisältää tarkkaa tietoa ohjelmasta ja sen osista, mutta toisaalta päivitettävyyden tulee olla mahdollisimman vaivatonta. Tämän vuoksi kuvaus sisältää sekä graafisia että tekstipohjaisia elementtejä sen mukaan, millainen menetelmä kunkin kuvauksen tason esittämiselle parhaaksi havaitaan.

Työn tavoitteena on kehittää kuvausmenetelmä ja luoda kuvaus yrityksen tarjoamasta ohjelmistotuotteesta, sen osista sekä sen osuudesta kokonaisessa järjestelmässä. Kuvausmenetelmän tulee tehostaa lähdekoodin ja ohjelmiston rakenteen sekä toiminnallisuuden ymmärrystä projektin ulkopuolisille henkilöille. Hyvä kuvaus auttaa myös ohjelmiston kehittäjiä hahmottamaan kokonaisrakennetta ja jopa syy-seuraus-

suhteita entistä paremmin. SpringTime Oy:n lopputuotteena oleva SpringSystem-järjestelmä on monimutkainen ja koostuu useista keskenään yhteydessä olevasta ohjelmistosta ja laitteesta. Kuvauksen tulee selkeästi esittää ohjelmistotason ja laitetason yhteenkuuluvuus, mutta se keskittyy ennen kaikkea keskipisteenä olevaan SpringSoft-ohjelmistoon.

Nykyinen SpringSystem-järjestelmä sisältää suuren määrän teollisen työajanseurannan kulunvalvonnan ja materiaalinjäljityksen sovellusalueen tietotaitoa. Tämä tietous on paitsi sitoutunut järjestelmää kehittäneisiin henkilöihin, myös sulautunut järjestelmän ja sen ohjelmiston rakenteeseen ja ratkaisuihin. Tässä työssä kehitettävällä kuvausmenetelmällä on tarkoitus pyrkiä löytämään ja kuvaamaan etenkin SpringSoft-ohjelmistoon sitoutunut tietotaito. Toisaalta kuvaus auttaa näkemään ’suuren kuvan’ kokonaisuudesta ja auttaa näin hahmottamaan potentiaalisia ongelmakohtia entistä paremmin. Luotavan kuvaksen tärkein käyttökohde on pääohjelmiston uudelleensuunnittelun pohjustaminen. Työn lopputulosta onkin mahdollista ajatella uudelleensuunnitteluprosessin (reengineering) ensimmäisenä työvaiheena. Suurin osa järjestelmästä on niin heikosti dokumentoitu, että kuvaus saa myös vahvasti käänteisen suunnittelun (reverse engineering) vaikutteita. Osien rakennetta ja riippuvuuksia voidaan osittain selvitellä hyödyntäen puoliautomaattisia lähdekoodia tulkitsevia mallinnustyökaluja, mutta suurin osa työstä joudutaan tekemään käsin.

Luotavaa ohjelmiston rakenteen kuvausta on tarkoitus käyttää yrityksen vanhojen ohjelmoijien omaan käyttöön sekä uusien ohjelmoijien, ulkoisen työvoiman tai sidosryhmien perehdyttämiseen nykyisen järjestelmän ylläpidossa ja kehittämisessä. Nykyisessä muodossaan dokumentointi ei riitä uuden työvoiman tehokkaaseen käyttöönottoon, sillä pelkästään järjestelmän toiminnan ja rakenteen sisäistämiseen kuluisi uudelta työntekijältä arviolta vähintään kaksi kuukautta.

Kuvauksessa pyritään kehittämään mahdollisimman selkeä ja tähän tapaukseen toimiva kuvausrakenne, jota on mahdollista täydentää myöhemmin. Työssä ei ole tarkoitus kuvata koko järjestelmää täydellisesti, vaan perehtyä kunkin kuvauskerroksen kuvaustapaan käyttäen esimerkkeinä todellisia järjestelmän osia niin, että kuvauksen

laajentaminen on mahdollisimman triviaalia. Lopputuloksena syntyvä kuvaus on tarkka vain joidenkin valittujen järjestelmän osien kohdalla, mutta samaa menetelmää voi hyödyntää jatkossa myös muiden tarpeellisten osien kuvaamiseen.

Kuvauksessa on tarkoitus näkyä laajalla tasolla kaikki kokonaisen myyntituotteen osat ja niiden väliset yhteydet. Tarkemmin kuvauksessa paneudutaan kuitenkin ohjelmistotuotteeseen, jolla hallinnoidaan lukulaitteiden ja päätteiden tuottamaa dataa. Jokaisesta järjestelmän osasta tehdään lyhyt kuvaus joka kertoo laitteen, ohjelmiston tai ohjelmiston osan tehtävän ja siihen välittömästi liittyviä muita tärkeäksi havaittuja tietoja. Kuvausmenetelmässä pyritään mahdollisimman yhtenäiseen kuvaukseen riippumatta siitä, onko kuvattava kohde osa laitteistoa vai ohjelmistoa. Myös tietoyhteydet kuvataan vastaavalla menetelmällä. Tietoyhteyksiä ovat paitsi lähiverkossa olevien laitteiden liittymät, myös eri ohjelmistojen väliset ja ohjelmistojen moduulien väliset yhteydet.

Tässä työssä käsiteltävän ohjelmiston kehittäjät näkevät usein päivittäisessä työssään ohjelmiston lähinnä kooditiedostoina sekä valmiina sovelluksena. Lisäksi kehittäjät tuntevat usein vain rajallisen osan ohjelmistosta kunnolla sillä vastuualueet ovat määräytyneet suhteellisen tarkasti pitkän kehitysjakson aikana. Ohjelmiston käänteinen suunnittelu voi tuottaa lopputuloksena kuvauksen joka auttaa kehittäjiä näkemään koko järjestelmän rakenteen ja heidän työnsä merkityksen paremmin. Kuvaus konkretisoi ohjelmointityötä ja voi parhaimmillaan osoittaa uudenlaisia ratkaisuja tunnettuihin ongelmiin tai löytää ongelmia joita ei ole aiemmin tunnistettu [4]. Tämä auttaa edelleen kehittämään ohjelmistoa jatkossa.

Hyvästäkään kuvauksesta ei ole pitkällistä hyötyä ellei sitä ole helppoa ylläpitää ohjelmiston tai järjestelmän muutosten myötä. Tämän vuoksi kehitettävän kuvausmenetelmän on oltava riittävän yksinkertainen ja suoraviivainen helpon päivitettävyyden aikaansaamiseksi. Samoin kuvaukseen käytettävien työkalujen määrän olisi pysyttävä pienenä ja apuohjelmien keskinäisten roolien kuvauksessa tulisi olla selkeitä. Taloudelliset rajoitteet suosivat niin ikään pientä kuvausten tekemiseen ja

ylläpitämiseen käytettävien ohjelmien määrää, samoin kuin ylläpitoon budjetoitu työmäärä edellyttää ylläpidon vaivattomuutta.

Viimeinen tekninen tavoite on kuvausten siirrettävyys. Tässä työssä kehitettävä kuvausmenetelmä sisältää piirteitä ohjelmien käänteisestä suunnittelusta ja uudelleensuunnittelusta. Ohjelmistosta tehtävien rakennekuvausten yksi käyttömahdollisuus tulevaisuuden käyttökohde on myös ohjelmakoodin generointi kuvauksen pohjalta, jolloin menetelmään liittyy edestakaista suunnittelua (roundtrip engineering). Toimiva edestakainen suunnittelu puolestaan edellyttää muutakin kuin mallien luomista ohjelmakoodista ja päinvastoin. Koska yksittäiset CASE-työkalut (Computer-aided Software Engineering) eivät välttämättä kykene tekemään kaikkia tarvittavia asioita, tulisi kuvausmenetelmä ja mallit suunnitella niin, että luodut kuvaukset olisivat tarpeen niin vaatiessa siirrettävissä eri CASE-työkalujen välillä.

Alla olevaan listaan on kerätty kuvausmenetelmän suorat ja välilliset tavoitteet. Suora tavoitteet ovat kuvausmenetelmälle asetettuja tavoitteita. Välilliset tavoitteet ovat puolestaan tavoitteita menetelmällä luotujen kuvausten hyödyntämiselle.

Suorat tavoitteet kuvausmenetelmälle

- Kyky kuvata järjestelmän kaikki osat
- Kyky kuvata järjestelmän osien yhteydet
- Luettavuus ja yksinkertaisuus
- Päivitettävyys
- Standardinmukaisuus ja siirrettävyys

Välilliset tavoitteet kuvauksille

- Sovellusalueosaamisen kerääminen
- Uuden ohjelmiston kehittäminen
- Ulkoisen työvoiman perehdyttäminen
- Sidosryhmien perehdyttäminen
- Ongelmakohtien paikallistaminen

3 KUVAUKSEN POHJUSTAMINEN JA PERUSTIEDOT

Kohdejärjestelmän, -ohjelmiston ja sovellusalueen tunteminen on tärkeää järjestelmän kuvaamiseksi. Koska tässä työssä keskitytään erityisesti kohdejärjestelmän osana olevan SpringSoft-ohjelmiston rakenteen ja toiminnallisuuden kuvaamiseen, riittää järjestelmäkokonaisuuden ja sovellusalueen perustuntemus. SpringSoft-ohjelmistolla hallitaan pääasiallisesti tietokannassa olevaan tietoon perustuvaa työajanseurantaa, kulunvalvontaa ja materiaalin jäljitystä. Koko järjestelmän osista tiedonkeruun, laskennan ja muiden SpringSystem-sovellusten tuntemus ei tämän työn puitteissa ole ehdotonta kuin pintapuolisesti. Nyt kehitettyjä kuvausmenetelmiä on kuitenkin jatkossa tarkoitus soveltaa myös laajemmin järjestelmän osina toimiviin ohjelmiin ja tällöin myös näiden osien syvempi tuntemus tulee tarpeelliseksi.

Sovellusaluetuntemuksen lisäksi yksi osa tiettyyn ohjelmaan tutustumista on tuntea ohjelmiston kehitysprosessi. Ohjelmistotuotannossa tunnetaan useita yleisiä ohjelmistotuotannon menetelmiä, samoin kuin ohjelmointimenetelmiä ja -kieliä. Ohjelmiston kehitykseen käytetyt menetelmät ja ohjelmointikielet asettavat perustan kuvaukselle. Esimerkiksi, mikäli ohjelmisto on toteutettu olio-ohjelmointia käyttäen, ovat luokkakuvaukset toimiva ja helpohko tapa kuvata ohjelmiston arkkitehtuuria, kun taas oliottomassa proseduraalisessa ohjelmoinnissa ohjelmiston rakennetta ei voi kuvata luokkakuvauksina luokkien puuttuessa. Ohjelmointikieli paitsi usein määrittää käytetyn ohjelmointimenetelmän, myös kertoo kuvaajalle tai kuvaustyökaluille millaisia lähdekoodirakenteita ohjelmakoodista tulee etsiä.

Laajoja ohjelmia suunniteltaessa yksi keino pyrkiä hallitsemaan suurta ohjelmistokokonaisuutta on Haikalan ja Märijärven mukaan jakaa ohjelma pienempiin osiin, moduuleiksi, jotka yhdessä muodostavat kokonaisen ohjelmiston mutta jotka kuitenkin ovat hallittavissa erillään toisistaan. Jokaiselle moduulille suunnitellaan sisäisen rakenteen lisäksi rajapinta, jolla se on yhteydessä muihin ohjelmiston osiin. Osat pyritään tekemään yksittäin hallittaviksi kokonaisuuksiksi, etteivät niiden sisäisen rakenteen muutokset vaikuttaisi muun ohjelmiston toimintaan. Tämä perusjako pätee

niin proseduraaliseen ohjelmointiin perustuviin ohjelmiin kuin oliopohjaisiin menetelmiin. Nykyisessä ohjelmistokehityksessä yleisessä olio-ohjelmoinnissa ohjelmisto koostuu jo ohjelmointimenetelmän periaatteiden mukaisesti pienemmistä keskenään vuorovaikutteisista osista. Oliopohjaisessa ohjelmistosuunnittelussa voidaan kuvata mm. lähdekoodin ja ohjelman luokkarakennetta sekä olioiden aktiivisuutta ohjelmistoa käytettäessä.

Tämän työn kohdeohjelmiston tapauksessa on käytetty olio-ohjelmointiin perustuvaa, mutta proseduraalisia piirteitä sisältävää ohjelmointia; lähdekoodi rakentuu muutamista pääluokista joista ohjelmiston varsinaiset toiminnalliset osat on periytetty. Nämä osat ovat kuitenkin rakenteeltaan ja olemukseltaan täysin proseduraalisia, eikä olioajattelua ole viety loppuun asti. Syynä epäpuhtaaseen olio-ohjelmointiin on oliopohjaisen kehitystyökalun käyttö, mutta proseduraalisen ohjelmoinnin parempi tuntemus kehittäjien keskuudessa. Kohdejärjestelmän kehittämiseen käytetyillä etenevän ohjelmistosuunnittelun (forward engineering) menetelmillä ja ratkaisuilla oli vaikutusta myös luotuihin kuvausmenetelmiin. Työhön vaikuttaneita ohjelmistosuunnittelun menetelmiä käsitellään kappaleessa 3.1.

Ylläpitovaiheessa olevan ohjelmiston kuvaaminen on suurelta osin käänteistä suunnittelua, eli ohjelman rakenne ja toiminnallisuus pyritään kuvaamaan lähdekoodia ja käännettyä ohjelmistoa hyödyntäen. Käänteisen suunnittelun vaikutusta työn kuvausmenetelmiin käsitellään kappaleessa 3.2. Mikäli ohjelmisto on kehitetty kattavaa dokumentaatiota käyttäen ja ylläpitäen, voivat rakenne- ja toiminnallisuuskuvaukset olla jo olemassa. Haikalan ja Märijärven mukaan [1] käänteisessä suunnittelussa pyritään valmiista lopputuotteesta selvittämään ohjelmiston tällaiset suunnitteluvaiheen mallit ja osat tai, jos suunnitteludokumentaatio löytyy, tarkistamaan toteutuksen ja suunnitelman yhdenpitävyys. Käänteinen suunnittelu voi myös paljastaa suunnitelman heikkouksia, kuten tehtävien lukumäärässä tai koodirivien koossa mitattuna liian suureksi kasvaneita osia tai luokkia [1,4].

SpringSystem-järjestelmän kokonaisuus kuvataan työn hierarkkisessa kuvauksessa ylimmällä tasolla. Järjestelmän ja SpringSoft-ohjelmiston ytimessä toimivan

tietokannan kuvaaminen on fyysisen järjestelmän ja ohjelmiston kuvaamisen ohella tarpeen, jotta kuvauksista tulee yhtenäinen kokonaisuus. Tietokanta sisältää sovellusaluetietoutta siinä missä SpringSoft-ohjelmistokin. Tietokannan kuvaus on lisäksi mukana uuden ohjelmiston kehityksessä; nykyinen tietokanta tarvitsee rakenteeltaan ja suunnittelultaan reilusti uudistusta pärjätäkseen uusien vaatimusten, kuten monikielisyyden, mukaisessa kehityksessä. Fyysisen järjestelmän ja tietokannan kuvaamiseen vaikuttaneita tekijöitä käsitellään kappaleissa 3.3 ja 3.4, vastaavasti.

Kohdejärjestelmän tapauksessa tärkeimmäksi tavoitteeksi koettiin SpringSystem-järjestelmän ja SpringSoft-ohjelmiston sitoman tietotaidon kuvaaminen. Aineeton tietotaito näkyy ohjelmistossa toimintoina ja toisaalta ohjelmiston rakenteena, missä tietyt sovellusalueen osa-alueet on jaettu pienempiin ryhmiin moduuleiksi ja lopulta lähdekoodin yksiköiksi (Delphissä unit) ja tiedostoiksi. Tavoitteen parasta saavutettavuutta varten päätettiin kehittää arkkitehtuurin kuvauksesta hierarkkinen kuvaus joka lähtee liikkeelle kokonaisesta järjestelmästä laitetason kuvauksena ja pureutuu sitten kohdeohjelmiston moduulitason kautta sen luokkarakenteeseen. Vaikka ohjelmisto ei noudata puhdasta olio-ohjelmointia, kykenee luokkakuvaus esittämään tarvittavan rakennekuvauksen toiminnallisuuden jakautumisesta. Toisena tavoitteena on pohjustaa ohjelmiston uudelleensuunnittelua, mitä ajatellen ylimalkainen luokkakuvaus antaa hyvin vapaat kädet miettiä uutta toteutusrakennetta pitäen silti kiinni vanhan järjestelmän tiedon ja toiminnan jaon tuomasta tietotaidosta.

Tärkeä ohjelmiston toiminnallisen kuvauksen käyttökohde on sujuvan kommunikaatiokanavan luominen ohjelmiston suunnittelijoiden, kehittäjien, käyttäjien ja muiden sidosryhmien välille sovellusalatietouden välittämiseksi ja arvioinniksi ohjelmiston kehityskaaren aikana. Erilaisia kuvausmenetelmiä on useita, mutta yleisesti niitä kritisoidaan vahvan teoreettisen pohjan puutteesta ja kykenemättömyydestä välittää sovellusalatietoutta ja todellisen maailman syy-yhteyksiä [5,6]. Tämän puutteen johdosta on mallien pohjalta vaikea ymmärtää, onko jokin osa tai toiminto tarpeellinen tietyn tavoitteen täyttämiseen ja miksi näin on. Tässä työssä rakennekuvaus esittelee ohjelmiston ja järjestelmän rakenteelliset osat ja

toiminnallisuuskuvaus yhdessä sanallisen osakirjaston kanssa pyrkii selvittämään osien roolit ja tarkoituksen.

SpringSoftin toiminnallisuuden kuvaamista ei havaittu yhtä selkeäksi ja suoraviivaiseksi tehtäväksi kuin sen rakenteen kuvaaminen. Koska ohjelmiston toiminnallisten osien kehitys on tapahtunut proseduraalista ohjelmointia käyttäen, ei rakenteen luokkakuvaus anna palautetta toiminnallisuudesta tehtyihin ratkaisuihin. Toisaalta ohjelmiston monimutkainen rakenne sisäänrakennettujen parametroitavien asiakaskohtaisten räätälöintien ja mm. kahden tietokannan tuen myötä on tehnyt lähdekoodista hyvin hankalan tulkita. Ohjelmiston tärkeimmäksi kohteeksi tunnistettu työaikojen hallinta-näyttö mm. sisältää noin 9000 riviä toiminnallista lähdekoodia yhdessä tiedostossa ja näin ollen yhdessä luokassa. Toiminnallisuuden kuvaaminen perinteistä vuokaaviota käyttäen on siis käytännössä mahdotonta, samoin kuin etenkin olio-ohjelmoinnin kuvauksissa yleisesti hyödynnetyn UML:n (Unified Markup Language) sisältämiä oliopohjaisia aktiviteettikaavioita käyttäen.

Toiminnallisuus päädyttiinkin kuvaamaan käyttötapauksina hyödyntäen käyttötapauskaaviota sekä sanallista käyttötapauskuvausta. Käyttötapaukset sitovat kuvaukseen kehitysryhmän tekemät valinnat toiminnoista joita loppukäyttäjällä on käytettävissään sekä menettelyistä millä loppukäyttäjät näitä toimintoja käyttävät. Käyttötapauskaaviot antavat näin yleisen kuvan ohjelmiston sisältämistä ominaisuuksista ja ne saadaan helposti huomioitua uutta ohjelmistoa suunniteltaessa. Lisäksi sanalliset käyttötapauskuvaukset loppukäyttäjän tekemistä työvaiheista tiettyä ominaisuutta käytettäessä kuvaavat nykyisen mallin käytettävyyttä ja antavat näin pohjaa tavoitteille suunnitella parempaa käytettävyyttä uuteen järjestelmään.

3.1 Etenevä ohjelmistosuunnittelu ja kuvaukset

Ohjelmistotuotanto ja alan tutkimus tuntee useita erilaisia menetelmiä ohjelmistokehitysprosessille. Menetelmät perustuvat yleensä ohjelmistotuotannon perusvaiheisiin: määrittelyyn, suunnitteluun, toteutukseen, testaukseen ja ylläpitoon.

Vaiheita voi olla enemmän tai vähemmän projektin tarpeiden mukaan, mutta tärkeimmät erot eri menetelmien välillä tulevat vaiheiden välillä liikkumisesta. Suoraviivainen, perinteinen vesiputousmalli etenee esitetyssä järjestyksessä vaiheesta toiseen kun edellinen vaihe on valmis. Mallin heikkoutena on oletus vaiheiden valmistumisesta yksi kerrallaan ja riippumattomuus seuraavista vaiheista. Toinen perusmalli menetelmille on iteroiva eli kiertävä prosessi, jossa koko prosessi tai osa siitä kiertää vaiheesta toiseen yhä uudestaan kehityskaaren aikana. Näin koko prosessi voidaan jakaa tavoitteellisesti pienempiin kokonaisuuksiin esim. ohjelmiston prototyypin tai perustoiminnallisuuden valmistamiseksi nopeammin.

Kolmannen ryhmän muodostavat markkinalähtöiset ketterät menetelmät (agile methods), joissa pyritään huomioimaan useiden ohjelmistoyritysten realiteetti: korkea tulostavoite pienessä ajassa. Ketterät menetelmät ovat usein iteratiivisia menetelmiä joista on karsittu mahdollisimman paljon manuaalista, aikaa vievää työtä ja joiden mallia on reilusti virtaviivaistettu nopeiden tulosten esiintuomiseksi. Ketterät menetelmät pyrkivät hyödyntämään lähdekoodin uudelleenkäytettävyyttä ja prototyyppejä nopeiden tulosten ja toimivan sidosryhmien kommunikoinnin mahdollistamiseksi. Ketterät menetelmät tuovat asiakkaan lähelle kehittäjiä, jolloin tiedonkulku on nopeaa ja tarpeisiin voidaan reagoida nopeasti. Yhteistä kaikille menetelmille on työn etenemisen dokumentointi. Dokumentoinnin määrä ja laatu riippuu vahvasti menetelmästä. Esimerkiksi ketteristä menetelmistä on karsittu paljon manuaalista dokumentointia prosessin nopeuttamiseksi.

Suunnittelu on perinteisessä ohjelmistotuotannossa monivaiheinen rakenteeseen keskittyvä prosessi, jossa ohjelmiston ja käsiteltävän tiedon rakenne, käyttöliittymän ominaisuudet ja sovelluksen toimintamalli johdetaan vaatimusmäärittelyistä. Pressmannin mukaan [2] ohjelmiston kehitystä ohjaavat kolme suunnittelun avainaluetta:

- tietoalue (data domain),
- toiminnallisuusalue (functional domain) ja
- olemusalue (behavioural domain).

Suunnittelu etenee näiden kolmen alueen mukaisesti tietorakenteiden ja tietokannan suunnittelusta ohjelmiston arkkitehtuuriin ja käyttöliittymän suunnitteluun. Ohjelmiston tarkempi komponenttitason suunnittelu tapahtuu osittain näissä vaiheissa saadun tiedon perusteella. Pressmannin avainalueet kuvaavat yleistä lähestymistapaa ohjelmiston tai sen osan suunnitteluun riippumatta käytetystä prosessimallista.

Tietorakenteen suunnittelussa luodaan ja arvioidaan ohjelman tai järjestelmän tietovirtoja (data flows), tiedon sisältöä (content) sekä tietoyksiköitä (data objects) [2]. Suunnittelu luo käyttäjän näkymän järjestelmän sisältämästä tiedosta. Tästä korkean abstraktiotason mallista kehitetään tietokoneen ymmärtämiä tieto- ja tietokantarakenteita. Usein tietorakenteiden vaatimukset vaikuttavat myös itse ohjelmiston rakenteen suunnitteluun.

Arkkitehtuurin ja toiminnallisuuden suunnittelu voi ohjelmiston toteutustavasta riippuen sisältää komponenttitason suunnittelua, olioiden valitsemista ja niiden välisten suhteiden suunnittelua tai esimerkiksi ohjelmiston toimintalogiikkaa selittävien algoritmien kehittämistä. Yleisesti tavoitteena on Pressmannin mukaan kuvata ohjelmiston rakenne yleisesti tunnettuja ja muokkaantuneita menetelmiä käyttäen virheiden minimoimiseksi. Korkean abstraktiotason yleisistä kuvauksista edetään kohti lähdekoodia eli matalaa abstraktiotasoa. Kuvausmenetelmän tulisi olla yksinkertainen ja helposti muokattavissa sekä selittää ohjelmiston modulaarisuus, rakenne ja logiikka. Parhaimmillaan kuvausmenetelmä mahdollistaisi automaattisen lähdekoodin rakenteellisen ylläpidon rakennekuvauksia käyttämällä.

Käytännössä jokaiselle ohjelmistoyritykselle muokkautuu ajan kuluessa jokin sisäinen ohjelmistotuotannon menetelmä joka määrittää käytetyt tavat ja työkalut sekä luo puitteet työstä tehtävälle dokumentoinnille. Menetelmä voi olla jokin useista tarjolla olevista tutkituista julkisista menetelmistä tai se voi vain perustua johonkin tällaiseen menetelmään. Menetelmä voi myös kehittyä kokonaan käytännön kautta ilman kirjallista taustaa. Tällöin menetelmä kehittyy todennäköisesti käytössä olevien resurssien ja henkilöiden myötä yrityksen omaksi toimintamalliksi. SpringSystem-

järjestelmästä ei ole olemassa suunnitteluvaiheen dokumentaatiota, mutta käytännössä nämä samat vaatimukset voidaan asettaa myös tässä työssä luotavalle käänteisen suunnittelun kuvausmenetelmälle. SpringSoft-ohjelmiston olemusta tai käyttäjätoiminnallisuutta ei myöskään ole suunniteltu tietoisesti malleja hyödyntäen, vaan ohjelmiston olemus on kehittynyt tilannekohtaisten vaatimusten ja asiakkaiden toiveiden perusteella. Paikoin käytettävyyttä on paranneltu käyttäjien palautteen perusteella, mutta toisaalla uusia ominaisuuksia on tuotu mukaan käytettävyyden kustannuksella.

Mitkään tässä kappaleessa mainituista prosessi- tai suunnittelumalleista eivät ole ainoita oikeita ratkaisuita. Käytännössä hyvin harva oikea projekti noudattaakaan puhtaasti mitään olemassa olevaa teoreettista mallia. Vaikka SpringSystem-järjestelmän kehitysmallikaan ei noudata puhtaasti mitään tässä kappaleessa esitettyjä teoreettisia prosesseja, sisältää SpringSoftin sekä muiden SpringTime Oy:n kehittämien sovellusten kehitysmalli kuitenkin piirteitä näistä kaikista. Yleinen toimintatapa on tehdä vaatimusmäärittely asiakkaan kanssa sopien tapauksesta riippuen koko järjestelmän vaatimuksista tai tietyn ohjelmiston vaatimuksista. Projektiin liittyvien sovellusten vaatimusmäärittelyt siirtyvät sitten kunkin ohjelmiston kehitysvastuussa olevalle ohjelmoijalle, jonka toimintatapaa ei yrityksen sisäisesti virallisesti määritellä. Käytettävä ohjelmointikieli sekä käytössä olevat omat tai kolmannen osapuolen ohjelmointikomponentit tosin voivat yhtenäistää varsinaista ohjelmakoodia hieman. Tässä vaiheessa ohjelmiston kehitys muistuttaa vesiputousmallia; tuote pyritään tekemään valmiiksi ennen asiakkaalle toimitusta. Usein SpringSystem-järjestelmän sekä muiden siihen liittyvien sidosjärjestelmien monimutkaisuudesta johtuen ensimmäinen valmis versio ei kuitenkaan täysin täytä asiakkaan toiveita vaatimusmäärittelyistä huolimatta. Syynä voi olla tarkkuudeltaan vajaa vaatimusmäärittely tai väärinymmärrys vaatimusmäärittelyn tulkinnassa. Ohjelmiston toimintaa korjaavaa ja täydentävää työvaihetta voi pitää iteroivana prosessina, jossa tehty työ saa palautetta asiakkaalta kunnes ohjelmisto vastaa täysin vaatimusmäärittelyä. Tässä vaiheessa asiakas saattaa myös muuttaa alkuperäistä vaatimusmäärittelyään, jolloin jatkoprojektia käsitellään kuitenkin jo uutena projektina.

SpringTime Oy:ssä hyödynnetään myös prototyyppi-kehityksen periaatetta osana luonnollista sisäistä kehitystyötä merkittäviä muutoksia tuovissa kehitystapauksissa. Tällaisia kehitysaskeleita on ollut mm. SpringSoftin moduulirakenteeseen siirtyminen sekä saman ohjelmiston ulkoasuun tehdyt merkityksellisemmät uudistukset. Vaikka prototyyppien käyttö toimii kehitysyksikön sisällä hyvänä kommunikaatiokeinona, ei prototyyppejä nähdä toimivimpana menetelmänä koko yrityksen sisäisessä kommunikoinnissa. Yrityksen toiminnassa perinteinen tai portaittain iteroiva vesiputousmalli on niin tuttu toimintamalli, ettei prototyyppejä välttämättä osata mieltää vielä paljon työtä ja kehittelyä vaativina esimerkkeinä tai malleina tuotteesta.

Yrityksen kehitysmalli muistuttaa kokonaisuutena eniten ketteriä menetelmiä asiakasvetoisuutensa ja vähäisen manuaalisen dokumentoinnin osalta. Mitään varsinaista ketterää menetelmää ei ole tietoisesti valittu toimintamalliksi, vaan käytäntö on ohjannut työn samaan suuntaan. Malli on toiminut varsin hyvin jo vuosia, sillä kehitysryhmä on pieni ja erittäin osaava. Muutoksia on voitu hallita asiakaskohtaisesti ja todella nopeasti usein ohjelmoijan omaan tilannekohtaiseen arviointiin perustuen. Ongelmia on kuitenkin tullut eteen, kun ohjelmisto monimutkaistuu, asiakkaiden ja tarvittavan kehityksen määrä kasvaa ja toisaalta vanhentuvan ohjelmiston seuraajaa pitäisi alkaa kehittämään. Yrityksessä ollaankin siirtymässä hallitumpaan versioiden suunnitteluun ja toimittamiseen, mikä tuo myös mukanaan uutta kehitysdokumentointia julkistettujen versioiden ominaisuuksista ja muutoksista kertovien versiokirjeiden sekä aiempaa pidemmän tähtäimen kehitysideoinnin myötä.

Pressmannin kolmijakoista suunnittelua SpringTime Oy:n toimintaan verratessa osat käyvät hyvin yhteen. Painoarvo on tosin selkeästi toiminnallisessa suunnittelussa tiedon ja olemuksen jäädessä huomattavasti pienemmälle huomiolle. Tämä kertoo siitä, että usein ohjelmoijat kehittävät tietorakenteita ja tietokantaa sen mukaan, mitä tarpeita toiminnallisuuden ohjelmoinnissa havaitaan. SpringSoftin taustalla oleva tietokanta on periytynyt edellisistä järjestelmistä jo 90-luvun alusta, eivätkä sen rakenteen tarjoamat mahdollisuudet ole riittäneet täysin nykypäivän vaatimuksille. Vaikka tietokannan

rakenteen muutoksia vältetään useiden paikoin tuntemattomienkin sidonnaisuuksien vuoksi, on kanta kokenut kehitystä vuosien varrella. Kannan rakennetta ei kuitenkaan ole missään vaiheessa ajateltu alusta asti uudelleen modulaarisen SpringSoftin tai edes sitä edeltäneen SpringSystem.exen kehityksen aikana. Samoin on käynyt ohjelmiston olemukselle, eli käyttöliittymälle ja käytettävyydelle.

SpringSoft on ohjelmistona monipuolinen, ja se näkyy ohjelman käyttöliittymässä. Tehtävittäin tai tehtäväalueittain jaettuja käyttöliittymän ikkunoita on täydennetty ajan myötä kasvaneiden vaatimusten ja ideoiden mukana. Paikoin uudet toiminnot on lisätty nopeasti käyttöliittymään ja elementtien sijoitteluun on panostettu vasta myöhemmin. Ohjelmiston toisiinsa liittyvät toiminnot ovat säilyneet kiitettävästi yhdellä näytöllä, mutta varsinkin uudelle ohjelmiston käyttäjälle näytöt voivat paikoin olla ahdistavan täysiä. Ulkoiseen olemukseen ja osittain käytettävyyteen liittyy myös tiedon esitystavat. Vanha kehitysympäristö ei ole itsessään tarpeeksi joustava moniin graafisen esitystavan tarpeisiin, joten selkeyttäviä ja intuitiivisia graafisia esityksiä ei ole kuin muutamissa tärkeimmissä osissa. Kaiken kaikkiaan käyttöliittymäkehityksen toimintatapa tuottaa pitkällä aikavälillä ilmeeltään hyvin erilaisia näyttöjä, joiden käytettävyys ei ole parasta mahdollista.

3.2 Käänteisen ohjelmistosuunnittelun menetelmät

Kun yritykselle tärkeät ohjelmistojärjestelmät vanhenevat, tulee käänteisestä suunnittelusta yhä tärkeämpää niiden ylläpidolle [7]. Yleensä käänteiselle suunnittelulle voi olla tarvetta joko silloin, kun ohjelmiston lähdekoodia ei ole saatavilla tai kun lähdekoodi on saatavilla, mutta dokumentaatio on vajaata tai puuttuu kokonaan. Käänteinen suunnittelu voi tulla apuun myös silloin, kun olemassa olevien suunnitelmien havaitaan eroavan kehitetystä ohjelmistosta, tai vastaavuudesta ei voida olla varmoja.

Käänteinen suunnittelu on prosessi jossa analysoidaan kohdejärjestelmä, tunnistetaan järjestelmän komponentit ja niiden suhteet sekä luodaan järjestelmästä esitys toisessa

muodossa tai korkeampaa yleistys- eli abstraktiotasoa käyttäen. Käänteisen suunnittelun päätarkoitus on edistää järjestelmän yleistä ymmärrettävyyttä ja ylläpitoa etenkin ohjelmiston suunnittelutasolla sekä helpottaa ohjelmiston kehittämistä edelleen [4,8,9]. Käänteinen suunnittelu auttaa

- hallitsemaan monimutkaistuvaa järjestelmää ja sen muutoksia,
- luomaan erilaisia näkökulmia järjestelmän kuvauksesta,
- luomaan korkeamman abstraktiotason kuvauksia järjestelmästä,
- täydentämään suunnittelu- ja ylläpitovaiheiden vajaita dokumentaatioita,
- paljastamaan riskialttiita suunnitelmista poikkeavia rakennemuutoksia,
- rakentamaan järjestelmän osia uudelleen,
- siirtämään järjestelmää uuteen modernimpaan kehitysympäristöön ja
- löytämään mahdollisia uudelleenkäytettäviä ohjelmiston osia.

Käänteinen suunnittelu on lähtöisin mekaanisten ja elektronisten laitteiden tutkimisesta, missä motiivina on ollut selvittää kilpailevan laitteen toimintaperiaate. Myös ohjelmistotuotannossa vakoilu on mahdollista esim. suljettujen järjestelmien rajapintojen selvittämisessä, mutta useammin ohjelmien käänteisessä suunnittelussa on kuitenkin kohteena yrityksen oman työn tulos [2,8].

Erilaisissa ohjelmistotuotteen elinkaarimalleissa kehityksen aikaisemmat vaiheet käsittelevät luotavaa järjestelmää yleisellä tasolla, suunnitelmina ja ratkaisuina jotka eivät ole toteutustapaan sidottuja. Käänteistä suunnittelua voi ohjelmiston tutkimisena ajatella väärinpäin eteneväksi kehitystyöksi, jossa normaalin kehitystyön lopputulosta, lähdekoodia ja käännettyä ohjelmaa tutkimalla edetään kohti alkuperäistä määrittelyä eli kohti abstraktimpaa mallia järjestelmästä. Käänteisessä suunnittelussa on ala- ja sivuprosesseja, kuten uudelleendokumentointi (redocumentation) ja suunnittelumallin palauttaminen (design recovery). Uudelleendokumentoinnissa tuotetaan kuvauksia jonkin abstraktiotason näkymästä helpottamaan ohjelmiston toiminnan seuraamista eri tilanteissa. Suunnittelumallin palauttaminen puolestaan siirtyy alemmalta abstraktiotasolta ylemmäs hyödyntäen laajalti saatavilla olevia resursseja, kuten

lähdekoodia, olemassa olevaa dokumentaatiota ja alan asiantuntijoiden tietotaitoa. [8,9]

Käänteisellä suunnittelulla ei pystytä selvittämään täydellisesti kohteena olevan järjestelmän suunnitteluprosessia. Kehitystyössä on voitu liikkua erilaisten suunnitelmien ja mallien välillä kunnes lopullisen tuotteen toimintamalli on löydetty. Käänteinen suunnittelu ei voi palauttaa tietoa tehtyjen valintojen syistä ellei valintoja ole dokumentoitu suunnitteluaikana. Toisaalta käyttämällä lopputuotteen suunnittelumallia ja lähdekoodia käänteinen suunnittelu voi paljastaa seikkoja, joita alkuperäisessä suunnittelutyössä ei osattu huomioida, esim. koodauksen aikana tehtyjä kiertoteitä tai ohjelman ylläpidon tuomia rakenteellisia muutoksia, nk. sivuvaikutuksia [8].

Käänteisen suunnittelun kohteena olevaa järjestelmää ei prosessin aikana muuteta. Järjestelmän muuttamiseen pyrkivät uudelleensuunnittelu ja uudelleenrakentaminen (restructuring) kuitenkin sisältävät käänteistä suunnittelua jolla selvitetään kohteen yleistetyn tason kuvaus. Käänteistä suunnittelua seuraavat sitten etenevän suunnittelun työvaiheet, joissa joko suunnitellaan koko järjestelmän toteutus uudelleen tai rakennetaan yksittäinen ohjelman osan rakenne uusiksi. Vaikka ohjelmiston lähdekoodi muuttuu, pyritään vanhan järjestelmän toiminnallisuus säilyttämään. Myös uutta toiminnallisuutta voidaan samalla kehittää [8].

SpringSystem-järjestelmä ja sen pääohjelmana toimiva SpringSoft ovat monipuolisia ja laajoja järjestelmiä. SpringSoft on SpringTime Oy:lle tärkeä ohjelmistojärjestelmä, jonka ylläpitäminen on nykyisellään yritykselle elinehto. SpringSoft on kuitenkin kehittynyt tilannekohtaisia ratkaisuja käyttäen ja heikkoa dokumentointia noudattaen, mikä on epäedullista ylläpidon toimivuutta ajatellen. Näin ollen SpringSoftin kohdalla joudutaan turvautumaan käänteisen suunnittelun menetelmiin joidenkin ohjelmiston ominaisuuksien kuvaamiseksi. Kuvauksen tarkoitus eroaa siis perinteisimmästä käänteisen suunnittelun ajatuksesta eli ohjelmistoteollisuusvakoilusta tai vaikkapa suljettujen tiedostomuotojen selvittämisestä. SpringSoft on SpringTime Oy:n oma tuote ja sen kehitysryhmä on pääpiirteittäin edelleen toiminnassa, joten käytännössä

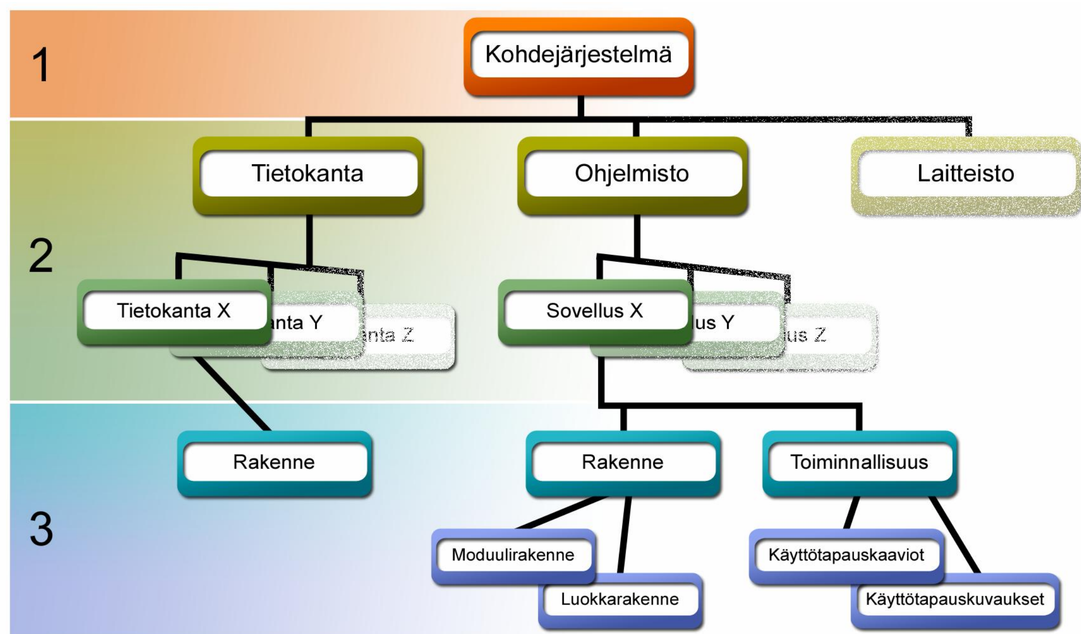
kaikki tietotaito edelleen on yrityksen käytössä. Pääasiallisena tavoitteenakin on uuden järjestelmän suunnittelun pohjustaminen ja vasta toissijaisena tarkoituksena nykyjärjestelmän ylläpito. Käänteistä suunnittelua onkin näin ollen tavoitteena käyttää nopeaan ja tarkkaan rakenteen kuvaamiseen sekä monipuoliseen toiminnallisuuden kuvaamiseen. Esimerkiksi rakenne on pääasiallisesti tuttu ohjelmiston kehittämisen kanssa tekemisissä oleville henkilöille, mutta sidosryhmille rakenne ei ole selvä. Usein myös kehitysryhmän jäsenille tutuinta ovat tietyt omat osat ja alueet järjestelmässä, eikä muiden osien roolista tai oikeellisuudesta ole aina täyttä varmuutta.

Käänteinen suunnittelu on vielä nuorehko tutkimusala etenkin ohjelmistotuotannossa. Käänteisen suunnittelun tavoitteet ovat vielä yleisesti epäselviä ja menetelmät enemmän tai vähemmän tilannekohtaisia [7]. Tässä työssä SpringSystem-järjestelmän kuvaamiseksi kehitetyn kuvausmenetelmän käänteisen suunnittelun osat eivät noudata suoraan mitään mahdollisesti olemassa olevaa käänteisen suunnittelun mallia. Käytännössä sovellusten kehitysvaiheiden erot kehitystyökalujen valinnassa ja käytetyissä menetelmissä ja ratkaisuissa sekä toisaalta käänteisen suunnittelun tavoitteiden erot tekevät yleishyödyllisten käänteisen suunnittelun menetelmien suunnittelusta vaikeaa. Työtä varten on perehdytty useaan erilaiseen käänteisen suunnittelun menetelmään ja case-tapauksiin. Näiden pohjalta on paitsi tutkittu mahdollisuutta käyttää automaattisia tai puoliautomaattisia CASE-työkaluja, myös poimittu vaikutteita kuvauksen tärkeimmistä kohteista ja esitystavoista. Lopputulokseen vaikuttaneita menetelmiä esitellään pääpiirteissään seuraavissa kappaleissa.

3.2.1 SpringSystem-järjestelmän kuvauksen rakenne

SpringSystem-järjestelmän kuvaus on kerroksittainen, hierarkkinen rakenne, joka etenee korkean abstraktitason järjestelmäkuvauksesta kohti matalan tason luokkakuvausta. Tällainen rakenne auttaa kuvausta lukevaa ensin ymmärtämään järjestelmän perusrakenteen ja -periaatteet ja saamaan näin valmiuden tulkita yhä tarkempia kuvauksia. Kuvassa 1 ensimmäinen taso on koko järjestelmän yleiskuvaus,

joka sisältää järjestelmän osat laitteistosta ohjelmistoon. Kuvausmenetelmän toinen taso kuvaa kunkin järjestelmän osakokonaisuuden yleistasolla. Tietokannassa tämä voi olla luettelo tauluista, ohjelmistossa kuvaus sovelluksista ja niiden riippuvuuksista. Kolmannella tasolla kuvataan ohjelmiston rakennetta ja toiminnallisuutta. Taso voi jakautua useampaan kerrokseen kuvattavan kohteen laajuuden asettamien tarpeiden mukaan. Kohdejärjestelmän SpringSoft-ohjelmisto jakautui rakenteellisesti moduulikuvaukseen sekä kahteen eritasoiseen luokkakuvaukseen. Toiminnallisuuden kuvaaminen jakautui SpringSoftissa kuvallisiksi käyttötapauskaavioiksi sekä sanallisiksi käyttötapauskuvauksiksi. Kuvausmenetelmää voisi laajentaa myös laitteiston kuvaamiseen, mutta tähän ei kohdejärjestelmän tapauksessa havaittu tarvetta.



Kuva 1. Kuvausmenetelmän hierarkia

Koko järjestelmän kuvaaminen ja kaikkien tavoitteiden saavuttaminen yhdellä menetelmällä on vaikeaa, sillä vaihtelu kuvauksen tarkkuudessa ja kuvauskohteissa on tässä työssä suurta. Rakennekuvaukset ovat käytännössä suunnittelumallin palauttamisen prosessia. SpringSystem-järjestelmää tai SpringSoft-ohjelmistoa luotaessa järjestelmällisiä kuvauksia rakenteesta ei ole tehty, joten ne on luotava käänteisesti. Ylimpänä tasona toimii järjestelmän fyysinen rakennekuvaus. Sen tehtävänä on esittää kokonaisen SpringSystem-järjestelmän kokoonpano ja toimivat

osat. Järjestelmän lähellä toimineille rakenne on usein tuttu, mutta kaavio helpottaa kokonaisuuden hahmottamista etenkin sidosryhmille. Samalla järjestelmätason kaavio osoittaa, mihin mikäkin ohjelmistokehitysyksikön ohjelmisto tai ohjelmakomponentti fyysisesti sijoittuu. Kun tässä työssä tehtävä ohjelmistokuvaus laajennetaan jatkossa myös toissijaisiin osiin järjestelmässä, toimii järjestelmäkuvaus ikään kuin karttana näille kuvauksille.

Ohjelmistotasolla voi olla ohjelmiston rakenteesta riippuen yksi tai kaksi kuvausta. SpringSoftin tapauksessa ensimmäinen taso kertoo ohjelmiston modulaarisen rakenteen ja toinen taso ohjelmiston luokkarakenteen. Vaikka laajuuteensa ja monipuolisuuteensa verraten ohjelmisto sisältää hyvin vähän luokkia – yhden jokaista näyttöä tai raporttia kohden – tulee luokkarakenteen kuvauksesta todella massiivinen. Tämän vuoksi koko SpringSoftia esittävään luokkakuvaukseen on merkitty vain periytyvien luokkien nimet ilman luokkien ominaisuuksia. Näin myös luokkakuvaus toimii karttana josta nopeasti näkee tietyn luokan – eli SpringSoftin tapauksessa tiedoston – luonteen ja sijainnin modulaarisessa järjestelmässä. Jokaisesta päätason luokasta on sitten periytettyine luokkineen tehty tarkempi, attribuutit ja metodit esittelevä kuvaus.

Järjestelmän elementtien kirjallinen dokumentointi eri tasoilla on prosessina järjestelmän uudelleendokumentointia. Vastaavaa osakohtaista dokumentointia ei SpringSystem-järjestelmästä ole tehty tai sen ylläpitäminen on unohtunut. Dokumentointi helpottaa osien roolien ymmärtämistä tarjoamalla selkokielisen selityksen kustakin osasta. Toisaalta dokumentoinnilla voidaan tallentaa monipuolisesti tarpeelliseksi havaittuja tietoja mm. sidonnaisuuksista joita rakennekuvaus ei kykene esittämään.

Järjestelmän mahdollisen uudelleensuunnittelun kannalta kuvauksen olisi hyvä olla laajalti standardien mukainen. Tämä helpottaisi erilaisten CASE-työkalujen käyttöä ohjelmiston kehityksessä, kun mallin voisi sellaisenaan siirtää työkalujen välillä. Lisäksi kuvauksen hyödyntämisen kannalta olisi tärkeää, että kuvaus sisältäisi

ohjelmiston osien ja suhteiden lisäksi mitattavaa tietoa. Tällainen tieto voi olla hyödyksi järjestelmän heikkoja kohtia ja pullonkauloja etsittäessä.

3.2.2 Rakenteen kuvaus

Työssä kehitetään kattava rakenteellinen kuvausmenetelmä kohteena olevan järjestelmän toiminnan kuvaamiseksi. Rakenteen huomioiminen on luonnollista järjestelmässä joka koostuu useista ohjelmisto- ja laiteyksiköistä. Työn pääkohteena on kuitenkin järjestelmän keskeinen ohjelmisto, jonka rakenteen dokumentointi on niin ikään tärkeää. Rakennetieto ei ole loppukäyttäjän tai järjestelmän itsensä käyttämää tietoa vaan ohjelmiston sisältämää sovellusaluetietoutta sekä ohjelmiston arkkitehtuuriin liittyvää ja siihen sisältynyttä tietoa [9]. Suurin osa ohjelmistojen kehitysdokumentaatioista kuvaa tyypillisesti varsinkin proseduraalista ohjelmistoa algoritmien ja tietorakenteiden tasolla. Ylläpidon mahdollistamiseksi pitkän kehityskaaren ohjelmistoille rakenteen ja arkkitehtuurin kuvaaminen on kuitenkin tärkeämpää [10]. Tällaiset ohjelmistot ovat usein suuria, kypsiä ja monimutkaisia järjestelmiä, jotka ovat pitkäaikaisen työpanoksen tulosta. Ohjelmistoon käytetty suuri ajan ja rahan panos vaatii sen kehittänyttä yritystä pitkittämään ohjelmiston elinkaarta vaativin ylläpitotoimenpitein jotta ohjelmisto voi maksaa itsensä takaisin.

Useimmat pitkän kehityskaaren ohjelmistot kärsivät samantyyppisistä ongelmista. Ne on alun perin kirjoitettu 10-25 vuotta sitten eikä perityissä järjestelmissä (legacy systems) ohjelmiston alkuperäinen kehitysryhmä ehkä enää ole käytettävissä. Ohjelmointikäytännöt ja -kielet ovat vanhentuneet ja ohjelmisto voi olla heikossa kunnossa pitkällisen ylläpidon ja siitä johtuvan kasvaneen monimutkaisuuden vuoksi. Ohjelmistosta tehty dokumentaatio kohdistuu vain joillekin järjestelmän osille ja on usein vanhentunutta tai hukkunut. Näiden ongelmien vuoksi ohjelmiston käänteinen suunnittelu on hankalaa, mutta toisaalta sitä vaaditaan toimivaan ylläpitoon, uudelleensuunnitteluun ja kehittymiseen. Suurille pitkän kehityskaaren ohjelmistoille järjestelmän rakenteen uudelleendokumentointi ja rakenteellisten seikkojen ymmärtäminen on yksittäisten ratkaisujen algoritmeja tärkeämpää [4,10].

Käytännössä rakennedokumentoimattoman ohjelmiston arkkitehtuurin kuvaus ja ylläpito vaatii lähdekoodin läpikäyntiä ja merkityksellisten järjestelmän osien tunnistamista suoraan lähdekoodista. Tällaisessa prosessissa lähdekoodista etsitään yleisiä ohjelmointirakenteita ja tunnistetaan komponenttien sisäinen rakenne [2]. Usein yksittäisen komponentin toiminta muistuttaa kolmijakoa jossa ensin valmistellaan syötteenä saatu tai haettu tieto, käsitellään tietoa ja lopuksi valmistetaan se ulosantia tai vientiä varten. Matalan tason lähdekoodin läpikäynti ja osien liittäminen korkean tason kuvaukseen on käsityönä vaativa tehtävä, mutta toisaalta juuri tällaisesta dokumentaatiosta käänteisen suunnittelun prosessissa on kyse. Suurille järjestelmille käytetäänkin yleensä puoliautomaattisia työkaluja perusrakenteiden selvittämiseksi. Wongin [10] mukaan ohjelmiston rakennekuvaus sisältää kolme avaintekijää: *komponentit* kuten ohjelmiston funktiot, moduulit, liittymät ja alijärjestelmät, *riippuvuudet* komponenttien välillä, kuten periytys ja ohjausvuo (control flow) sekä *määritteet* jotka selittävät mm. komponenttien tyypin tai liittymien koon.

Ohjelmiston normaalisti etenevä kehityskaari alkaa korkean tason yleistetyistä malleista ja päättyy tarkkaan rakenteen määrittelyyn lähdekoodissa. Käänteisessä suunnittelussa pyritään luomaan rakennedokumentti järjestelmästä tunnistamalla järjestelmän nykyiset komponentit riippuvuuksineen ja määritteineen sekä luomalla näistä tarvittavia yleistyksiä monimutkaisuuden karsimiseksi ja toteutusriippumattomuuden saavuttamiseksi [8,10]. Rakenteellinen uudelleendokumentointi on siis järjestelmän arkkitehtuurin käänteistä suunnittelua, mutta ei sisällä järjestelmän uudelleenrakentamista luodun mallin pohjalta. Toki kuvaus on hyvä pohja myös rakenteen uudistamiselle. [10]

Yksi keino selvittää ohjelman rakenne on käyttää metamallia [9] joka Knodelin mukaan myös auttaa sovellusaluetiedon selvittämisessä järjestelmästä. Metamalli on staattinen ohjelman perusmalli joka esittää lähdekoodissa olevat rakenteet sekä niiden suhteet ohjelmoijan ja lähdekoodin näkökulmasta. Malli ei esitä varsinaista kehitystyön lopputuloksena olevaa ohjelmistoa vaan toimii yleisenä rakenteen mallina. Metamalli voi olla kehityskielestä riippumaton yleinen malli tai kielikohtainen

rakenne. Kieliriippumattomat mallit eivät aina kykene tarjoamaan kaikkia erityyppisten kielten rakenteita tai sisältävät ko. kielelle turhia rakenteita, joten esimerkiksi Knodel on luonut Delphille oman metamallin joka yleisiä malleja paremmin kuvaa Delphin oliomallia ja ohjelmarakennetta. Toimivaa metamallia voidaan hyödyntää manuaalisessa ja automatisoidussa rakennetiedon eristämisessä.

SpringSystem-järjestelmää ajatellen metamalleja on olemassa sekä olio-ohjelmoinnin piirteiden kuvaamiselle että Delphi ohjelmointikielelle, mutta käytännössä näiden käyttäminen tämän työn pohjana osoittautui potentiaalisesta hyödyllisyydestään huolimatta vaikeaksi. Vaikka ensisilmäys SpringSoftin lähdekoodiin antoi kuvan oliopohjaisesta mallista, ei ohjelmiston käytännön toteutus noudata metamallien oliomallia. Samantyyppinen ongelma tuli eteen myös käänteisen kehityksen työkalujen kanssa. Tutkimusraporttien perusteella on olemassa useita nopeita, apuohjelmiin perustuvia käänteisen suunnittelun prosesseja. Usein nämä työkalut mainitaan kieliriippumattomiksi, mutta käytännössä lupaus osoittautui ainakin Delphin osalta epätäydelliseksi. Eri ohjelmointikielten erilaiset lähestymistavat ja syntaksit eivät käytännössä toimi yhdellä metamallilla oikein eikä suoraa tukea Delphin käyttämälle mallille usein löydy. Lisäksi apuohjelmat tähtäävät tyypillisesti hyvin pienen sektorin työhön, tiettyyn tarpeeseen tai tehtävään. Apuohjelmat ja menetelmät joihin tämän työn taustatutkimuksissa perehdyttiin, eivät käytännössä toimineet laajan tietotaidollisen informaation keräämiseen eivätkä ne toisaalta tukeneet ohjelmiston kehityksessä käytettyä osittaista olio-ohjelmointia.

SpringSoft on verrattain vanha ohjelmisto, joka sisältä myös perittyjen järjestelmien piirteitä. Tätä kuvausta tehdessä alkuperäinen kehitysryhmä on suurimmaksi osaksi ennallaan ja käytettävissä, mutta ohjelmiston rakenne itsessään on periytynyt saman ryhmän aiemmista sovelluksista. Monipuolistuneen kokonaisuuden ylläpitäminen vaikeutuu tasaisesti ajan myötä, sillä ohjelmiston pitäminen ajantasaisena uusia ominaisuuksia tai uutta olemusta kehittämällä haastaa aina vanhaa lähdekoodia ja odottamattomia ongelmia ilmenee helposti. Ohjelmiston rakenne on monipuolisuudestaan huolimatta tuttu ja selvä kehitysryhmän avainhenkilöille. Kuvauksessa luodaan silti rakenteellinen kuvaus helpottamaan ylläpitoa ja kehittämään

sidosryhmien ymmärrystä ja kommunikaatiota järjestelmästä. Yksittäisten ratkaisujen toteutuksiin ei kuvauksessa algoritmitasolla mennä, sillä pitkän kehityskaaren, proseduraalisen ohjelmoinnin ja dokumentoimattoman lähdekoodin johdosta lähdekoodin tulkinta osoittautui liian hitaaksi prosessiksi. Ohjelmiston tärkeimpiä osia päädyttiinkin kuvaamaan ulkoisesti käyttötapauksina sekä ohjelmiston toimintalogiikkana tärkeimpiä toimintoja käytettäessä. Toimintalogiikka selvitetään hyödyntämällä kunkin ohjelmiston osan toimintaan parhaiten perehtyneen kehittäjän selvitystä.

3.2.3 Käyttötapauskuvaukset ja toiminnallisuuden kuvaus

Käyttötapaukset kuvaavat loppukäyttäjien todellisia käyttötilanteita järjestelmän kanssa ja kuvaavat näin järjestelmän toiminnallisia vaatimuksia. Käyttötapauskuvaukset voivat olla sanallisia, hyvinkin tarkkoja kertomuksia ihmisen ja koneen vuorovaikutuksesta tietyssä tilanteessa [14] tai ylimalkaisempia graafisia kuvauksia erilaisista toiminnoista joita käyttäjillä on käytettävissään. Käyttötapauksia voidaan hyödyntää ohjelmiston etenevässä kehityksessä ohjelmiston rakenteen ja tarvittavien luokkien hahmottamiseen, mutta toisaalta myös valmiin tuotteen toiminnallisuuden testaamiseen.

Käänteisellä suunnittelulla ei voida tuottaa täsmälleen vastaavia käyttötapauskuvauksia kuin mitä ohjelmiston etenevässä suunnittelussa on käytetty. Olemassa oleva ohjelmisto ja sen toiminnot vaikuttavat luotavaan kuvaukseen, jolloin käyttötapauskuvaus kuvaakin ohjelmiston toteutunutta toiminnallisuutta suunnitellun toiminnallisuuden sijaan. Tällaista toteutumaa on esimerkiksi tämän työn puitteissa mahdotonta verrata vaatimuksiin, sillä koko järjestelmän kattavia vaatimusmäärittelyitä tai käyttötapauskuvauksia ei ole olemassa ja eri asiakkaiden vaatimukset ja käyttötapaukset eroavat merkittävästi toisistaan. Yleistä käsitystä pienimmästä mahdollisesta ”perus” SpringSystem-järjestelmästä tai laajimmasta ”täydellisestä” SpringSystem kokonaisuudesta ei myöskään ole olemassa sillä

käytännössä järjestelmä sisältää aina joitakin räätälöitäviä osia ja osa asiakkaiden räätälöinneistä menee ristiin toistensa kanssa.

Käyttöliittymän ja käyttäjän välinen toiminta on SpringSoftin tapauksessa pääosin kuvailtu ohjelmiston käyttöoppaassa, mutta käyttöopas ja tämän työn kuvausmenetelmän käyttötapauskuvaukset eroavat merkittävästi luonteeltaan. Graafisen käyttötapauskaavion tarkoitus on luoda nopeasti lähestyttävä yleiskartta ohjelman toiminnoista tehtävälähtöisesti. Siinä missä käyttöopas kertoo *miten*, on käyttötapausten tarkoitus kertoa *mitä*. Tämä ero havaittiin tärkeäksi mietittäessä nykyjärjestelmän toiminnallisuuden siirtoa uuden järjestelmän toiminnoiksi. Uuden ohjelmiston on osattava suorittaa vähintään samat tehtävät kuin vanhankin, mutta mielellään entistä tehokkaammin. Yleistäen, uuden järjestelmän yleiset käyttötapauskuvaukset voivat olla hyvinkin lähellä vanhaa järjestelmää, kun taas käyttöopas on kirjoitettava kokonaan uusiksi.

Syvyyttä käyttötapauskuvauksiin haluttiin toisaalta kokeilla tuottamalla vuokaavio ohjelman toimintalogiikasta tiettyjä tärkeimpiä toimintoja käytettäessä. Luotujen logiikkakuvausten hyödynnettävyys ei kuitenkaan ollut itsestään selvää. Nykyisen ohjelmiston toiminnan selittämiseksi niillä on selkeä tarkoitus, mutta toisaalta ohjelmisto on jo pääasiallisesti ylläpitovaiheessa ja nykyinen kehitysryhmä tuntee ohjelmiston logiikan tarpeeksi hyvin ylläpidon tarpeisiin. Uuden järjestelmän suunnittelun pohjana looginen kuvaus sitoo nykyistä enemmän tietotaitoa tuomalla mukaan ohjelmiston pinnan alla tapahtuvia tarkistusrutiineita ja niihin liittyviä toimijoita, mutta toisaalta saattaa vaikuttaa uuden ohjelmiston suunnittelun valintoihin hillitsemällä uusien näkökulmien tuomista kehitystyöhön.

SpringSystem-järjestelmän toiminnallisuuden kuvausta mietittäessä havaittiin, ettei käytettyjen työkalujen ja joidenkin ohjelmointimenetelmien suosimat toiminnallisuuskaaviot toimi tässä tapauksessa. Koska ohjelmistoa ei ole suunniteltu oliopohjaisesti, ei ohjelmiston tai toimialan toimijoita ole kuvattu luokkina ja olioina lähdekoodissa. Käytännössä pienin ohjelmiston elementti on jonkin näytön tai raportin sisältämän lähdetiedoston yksi funktio. Näiden välistä kanssakäymistä on kuitenkin

hyödytöntä kuvata teknisesti kutsujen muodossa toiminnallisuuden esittämiseksi. Samoin mahdottomaksi toiminnallisuuskuvaukseksi osoittautui perinteisesti proseduraalisessa ohjelmoinnissa käytetty funktioiden kutsuja kuvaava vuokaavio, sillä ohjelmiston monipuolisuus tekee mahdollisten valintojen määrästä suuren. Näin ollen päädyttiin kuvaamaan SpringSystemin tai tarkemmin SpringSoftin toiminnallisuutta käyttötapauskuvauksina.

Käyttötapauskuvauksista voidaan SpringSystem-järjestelmän tapauksessa saada eniten hyötyä käyttämällä kahta eri kuvaustasoa. Ensimmäinen kuvaustaso on em. graafisen kuvauksen mukainen ylimalkainen tehtäväkuvaus. Se kertoo graafisessa käyttötapauskaaviossa eri käyttäjäryhmien tehtävistä ja toiminnoista tietyllä SpringSoftin näytöllä. Kuvauksesta saa nopeasti mielikuvan kunkin näytön tarkoituksesta ja koko ohjelmiston toiminnoista ja mahdollisuuksista. Tämän tason kuvaus edesauttaa etenkin uuden ohjelmiston suunnittelua, sillä kaikkien nykyisen SpringSoftin toimintojen tulisi löytyä myös uudesta ohjelmistosta. Vaikka SpringSoftissa tai sitä seuraavassa uudessa ohjelmistossa tietyn tehtävän suorittamiseen tarvittavat työvaiheet muuttuisivat, ei itse tehtävän tarkoitus yleensä muutu. Yleisen tason käyttötapauskuvaus on siis liki muuttumaton toiminnallisuuskuvaus.

Tarkemmalla tasolla käyttötapauskuvaus tehdään tekstimuotoisena selittäen käyttäjän työvaiheet tietyn tehtävän suorittamiseksi. Käyttötapaus kuvataan ihmisen ja tietokoneen kanssakäymisenä esittäen paitsi ihmisen, myös ohjelmiston kommunikointi vaiheittain. Tarkat käyttötapauskuvaukset tehdään tämän työn puitteissa SpringSoftin valituista tärkeimmistä osista, mutta kuvausta laajennetaan tarpeiden mukaan näytöittäin uuden järjestelmän toiminnallisuutta suunniteltaessa. Tarkempaa käyttötapausta voidaan tarvittaessa syventää käyttöliittymän pinnan alle loogisilla kuvauksilla ohjelmiston tekemistä loogisista päättelyistä, tarkistuksista ja muista rutiineista. Looginen kuvaus tehdään sanallisessa muodossa, jolloin luonteeltaan erilaiset loogiset ratkaisut on helpompi esittää ilman graafisten menetelmien rajoitteita. Lisäksi looginen kuvaus on näin helpommin syvennettävissä ja tarkennettavissa.

3.2.4 CASE-työkalut ja sovelluskehitin

CASE-työkalut ovat ohjelmistopohjaisia apuvälineitä, jotka helpottavat tai edesauttavat jonkun ohjelmistotyön vaiheen suorittamista. Useat alkuvaiheen CASE-työkalut yrittivät kattaa koko ohjelmistokehitysprosessin, mutta käytäntö on osoittanut, etteivät tällaiset ratkaisut pärjää useamman kohdistetun työkalun käytölle [11]. Suunnittelussa ja kuvauksessa käytetään nk. edustatyökaluja (front-end tools), jotka ovat yksinkertaisimmillaan ohjelmakaavioiden piirto-ohjelmia ja monipuolisimmillaan eri kuvauksia ja näkymiä yhdistävään kuvauskantaan perustuvia työkaluja. Monipuolisemmat edestakaista suunnittelua tukevat työkalut voivat sisältää erilaisia näkymiä ohjelmarakenteeseen sekä tulkita ja luoda ohjelmakoodia. Teoriassa tällaiset sovellukset voivat tuoda merkittävää tehostusta, mutta käytännössä työkalut tulee valita todellisen tarpeen mukaan tekemään esimerkiksi työ mikä joka tapauksessa tehtäisiin, mutta mikä vaatisi käsin tehtynä paljon työtä. Toisaalta hyväkään CASE-työkalu ei ainakaan toistaiseksi tee kaikkea itse. [1]

Jotta CASE-työkaluja voidaan hyödyntää ohjelmointiprojektissa kumpaankaan suuntaan, on tuki käytetylle ohjelmointikielelle tärkeä ominaisuus. Käänteisessä ja edestakaisessa suunnittelussa kuvauksen ja ohjelmakoodin tulee olla lähellä toisiaan, eikä tämä onnistu ilman käytetyn ohjelmointikielen tukea. Tässä työssä kohteena olevan järjestelmän sovellukset on pääsääntöisesti toteutettu Delphi 7 sovelluskehittimellä käyttäen Delphi Pascal kieltä. Delphi on Borlandin 1970-luvun lopulta kehittämä kehitysympäristö joka pohjautuu Pascal-kieleen.

Delphi on tarkoitettu Microsoft Windows ja Linux ohjelmien kehittämiseen ja se tukee sekä proseduraalista että oliopohjaista ohjelmointia. Delphissä ei ole C++ kaltaisia header tiedostoja, vaan kaikki lähdekoodi kirjoitetaan .pas päätteisiin lähdetiedostoihin. Sovellusten käyttöliittymän ikkunoiden lähdekoodi luodaan automaattisesti sovelluskehittimellä .dfm tiedostoihin. Jokaisessa .pas lähdekooditiedostossa on vähintään yksi ohjelmayksikkö (unit), jota voi ajatella yksiselitteisenä nimiavaruutena. Ohjelmayksiköt ovat ohjelmistokehittäjien oman työn tuloksia ja nimenomaan niistä on ohjelmiston rakenne ja toiminta selvitettävissä.

Delphissä on myös järjestelmäyksiköitä, jotka ovat ohjelmiston käyttämiä kolmannen osapuolen luomia osia. Näiden eri yksiköiden erottaminen on käänteisessä suunnittelussa tärkeää oleellisen tiedon löytämiseksi. [9]

Delphi on tällä hetkellä etenkin Länsi-Euroopassa ja Pohjois-Amerikassa melko harvinainen kehitysympäristö, aktiivisimman käyttöalueen ollessa Itä-Eurooppa ja Venäjä. Pienehkö käyttäjäkunta hankaloittaa hyvien CASE-työkalujen löytämistä kuvauksen tueksi. Tämän vuoksi tässä työssä luotavat kuvaukset voivat vaatia enemmän manuaalista työtä kuin vastaavat projektit muissa kehitysympäristöissä mahdollisesti vaatisivat. Lähdekoodin tuottamiseen käytetyn kehitysympäristön ja ohjelmointikielen käytännön tuntemus osoittautui paikoin manuaalisesti tapahtuneen työn johdosta tärkeäksi. Delphiin tutustuminen tapahtui kehittämällä pieni ja suhteellisen yksinkertainen liittymäohjelma SpringSystem-järjestelmään. Projekti tutustutti onnistuneesti Delphin tietorakenteisiin ja olio-malliin sekä auttoi paikoin ymmärtämään yrityksen sisäistä toimintamallia omien ja kolmannen osapuolen komponenttien käytöstä. Tutustumisprojekti myös osoitti oikeaksi epäilyn pitkästä uuden työvoiman perehdyttämisajasta – Delphiä ja työskentelytapoja ymmärtävältä ohjelmoijalta työhön olisi kulunut arviolta viikosta kahteen, mutta nyt projekti vei jopa neljä viikkoa aikaa. Toisaalta sen tuoma ymmärrys joidenkin ratkaisujen motiiveista Delphi-ohjelmoinnista johtuen oli arvokas lisä ohjelmakokonaisuuden ymmärtämiselle.

Kuvauksen työstämisen pohjaksi löytyi lopulta käytännössä vain yksi toimiva vaihtoehto. Vaikka kuvauksen tekeminen käsin olisi jollain tasolla voinut toimia, olisi kuvauksen ylläpito todennäköisesti kärsinyt yhdessä edestakaiseen suunnitteluun hyödynnettävyyden kanssa. Tässä työssä käytetty CASE-työkalu on neljänä perättäisenä vuonna sittemmin lopetetun Delphi Developers lehden parhaaksi valitsema ModelMaker [12]. ModelMaker on Borland Delphi Pascal- sekä Microsoft Visual Studio C# .NET [13] kehityskieliä tukeva ohjelma. ModelMakerin uusin versio 9.0 hallitsee mm. lähdekoodin olio-mallin analysoinnin luokkakuvaukseksi sekä lähdekoodin uudelleenluomisen kuvausten perusteella. ModelMaker integroituu

kehitysympäristön kanssa tuoden kuvaukset lähemmäksi ohjelmoijia pyrkien näin helpottamaan kuvausten hyödyntämistä ohjelmistotuotannossa.

Käytännössä ModelMakerin valintaan vaikutti eniten tuki sekä Delphille että Visual Studiolle, sillä molempia kehitysympäristöjä käytetään parhaillaan SpringSystem-sovellusten kehittämiseen ja toisaalta kumpikin on ehdolla seuraavan sukupolven ohjelmien kehitysalustoiksi. ModelMakerin kyvystä luoda rakennekuvaus lähdekoodista osoittautui toissijaiseksi ominaisuudeksi, sillä nykyinen SpringSoftin ohjelmakoodi ei noudata puhdasta olio-ohjelmointia eikä ModelMakerin luoma Delphi-lähdekoodin metamalliin perustuva malli voi näin olla rakenteeltaan täydellinen. ModelMaker tarjoaa kuitenkin laajat työkalut paitsi puoliautomaattisesti luotujen kuvausten muokkaamiseen käsin, myös täysin omien kuvausten laatimiseen ja niiden sitomiseen olemassa oleviin kuvauksiin.

3.2.5 Visuaaliset kuvausmenetelmät

Graafisten kuvausmenetelmien käyttämisen on todettu helpottavan korkean abstraktiotason mallien ymmärtämistä monimutkaisenkin ohjelma- tai järjestelmärakenteen sisäistämistä ja niitä käytetään hyvin paljon [4,11,15]. Tällaisten näkymien luominen ja ylläpitäminen on kuitenkin työlästä ja muodostuu helposti pullonkaulaksi ellei käytössä ole apuohjelmia joilla työtä voi helpottaa. Käänteisen suunnittelun työkalut voivat luoda automaattisesti tai puoliautomaattisesti graafisia kuvauksia lähdekoodista sekä luoda vaihtoehtoisia näkymiä samasta abstraktiotasosta helpottamaan mallin tutkimista ja tarkastamista [8].

Sovellusten graafiset kuvausmenetelmät auttavat pääsääntöisesti analysoimaan ohjelmien eri osien riippuvuuksia ja näin ohjelman sisäistä toimintaa [11]. Michele Lanzan ja Stéphane Ducassen kehittämä polymetrinen näkymä on European Espritin FAMOOS projektin mukainen ohjelmiston kuvausmenetelmä joka pyrkii näyttämään käyttäjälleen nopeasti tärkeimpiä ohjelmiston piirteitä yhdessä kehittyneessä kuvaajassa.

FAMOOS projektin punainen lanka on kiteytetty kolmeen avainsanaan [4]:

- *Yksinkertaisuus*. Mallin on oltava toteutettavissa nopeasti ilman monimutkaisia ja kalliita apuohjelmia.
- *Skaalautuvuus*. Mallin on kyettävä esittämään myös suuria teollisia ohjelmistoja.
- *Kieliriippumattomuus*. Mallin on kyettävä käsittelemään eri ohjelmointikielillä toteutettuja ohjelmistoja.

Polymetrisen näkymän kantavana ideana on ottaa käyttöön kaksiulotteisen kuvaajan kaikki erottavat tekijät kuvaajan tulkinnan hyväksi. Näin ollen esimerkiksi luokkakuvauksessa luokkaa kuvaavan suorakaiteen leveys, korkeus ja väritys kertovat jotain luokan ominaisuuksista. Näin hyvin nopeakin vilkaisu kuvaukseen voi antaa nopeasti palautetta lähdeohjelman rakenteesta tai heikoista kohdista. Tällaisia mitattavia suureita voivat olla esimerkiksi luokan suuruus koodiriveinä, attribuuttien ja metodien määrä ja suhde, tai luokan riippuvuuksien määrä muihin luokkiin nähden. Suureita eri tavoin yhdistelemällä voidaan luoda näkymiä jotka auttavat uudelleen suunnittelua erilaisissa tilanteissa koko järjestelmän tasolta aina pieniin yksityiskohtiin asti.

SpringSoft-ohjelmiston kuvaamisessa visuaalinen kuvaaminen havaittiin mielenkiintoiseksi ja hyödylliseksi lähestymistavaksi. Rakenteen visuaalinen kuvaus on käytännössä ainoa järkevästi hyödynnettävissä oleva kuvaustyyli, kun puolestaan ohjelman toiminnallisuuden ja joidenkin ominaisuuksien kuvaus ei visuaalisesti ole yhtä selkeää. Myös tärkeimpien ominaisuuksien sisällyttäminen rakennekuvauksiin polymetristen kuvausten tyyliin herätti kiinnostusta, mutta polymetristen kuvausten tekemiseen suunnatut työkalut Codecrawler ja Moose [4] eivät toimineet Delphi-lähdekoodin kanssa. Näin ollen luodussa kuvausmenetelmässä päädyttiin käyttämään polymetrisen kuvauksen tuomia ideoita tärkeimpien seikkojen esilletuomiseksi. ModelMakerilla tehdyissä rakennekuvauksissa tämä tavoite näkyy luokkien järjestämisessä niiden käyttökohteen ja tarkoituksen perusteella sekä luokkien

värittämisessä niiden isäntämoduulin perusteella. Myös lähdeohjelmien koodipituudet laskettiin, mutta käytännössä tämän määreen esittäminen visuaalisesti unohdettiin pian ModelMakerin muotoilurajoitusten vuoksi.

3.3 Järjestelmän kuvausmenetelmät

Tässä työssä käsiteltävä ohjelmisto on osa laajempaa järjestelmää joka koostuu tiedonkeruulaitteista ja mahdollisista tarra-asettimista, tietoa käsittelevistä ja tallentavista työasemista sekä palvelimista ja näiden ohjelmista. Eri kokoonpanot eroavat merkittävästi eri toimitusten välillä asiakkaiden erilaisista tarpeista johtuen. Järjestelmän perusrakenteen ymmärtäminen on tärkeää pääohjelmiston kehittämiselle, joten tässä työssä kuvataan myös järjestelmätaso jolla ohjelmisto toimii.

Järjestelmäsuunnittelussa on kyse yleiskuvan luomisesta, täydellisten järjestelmäkokonaisuuksien tai sisäkkäisten järjestelmien kehittämisestä ja kuvaamisesta. Laajojen järjestelmien sekä järjestelmäperheiden ja sisäkkäisten järjestelmien kehittäminen on nykyään tavallista. Järjestelmäsuunnittelussa otetaan huomioon kaikki järjestelmien toimijat: laitteisto, ohjelmisto ja ihmiset. Kehittäjien ja muiden järjestelmään liittyvien tahojen vaivaton ymmärrys kokonaisjärjestelmästä on tärkeää. Kehittäjien kannalta järjestelmän osien uudelleenkäyttö helpottuu kun olemassa olevia ja kehitettäviä järjestelmiä voidaan järjestelmällisesti verrata keskenään. Myös hajautettu kehittäminen hyötyy kuvauksesta. Kuvaamiseen voidaan hyödyntää ohjelmistotuotannosta tutumpaa UML kuvauskielen 2.0 versiota, joka sisältää riittävät ominaisuudet järjestelmäsuunnittelun kehittämiseksi. UML 2.0 kykenee esittämään useita eri abstraktiotasoja kaikissa kaaviotyypeissä paremmin kuin UML 1.x määritys. Tästä on hyötyä järjestelmäsuunnittelussa [15], mutta myös tämän työn hierarkkisessa kuvauksessa.

UML 2.0:sta edelleen järjestelmäsuunnittelun tarpeisiin kehitetty SysML (Systems Modelling Language) laajentaa ominaisuuksia edelleen tarjoten kielilaajennuksia jotka tekevät peruskielestä täydellisen järjestelmäkeskeisen kielen. SysML:n kehitys on

kuitenkin vielä luonnosvaiheessa, eikä sille ole olemassa työkaluja. Koska SysML perustuu vahvasti UML 2.0 määritykseen, on UML 2.0 ja sitä hyödyntävät työkalut kykeneviä tukemaan järjestelmäsuunnittelua [15]. Tässä työssä UML 2.0 kuvauskieltä käytetään kohdejärjestelmän olemassa olevan laite-ohjelmistotason kuvaamiseen.

3.4 Tietokannan kuvausmenetelmät

Relaatiotietokanta on yksinkertaisimmillaan kuvattavissa tauluina ja sarakkeina. Taulujen nimeäminen ja luonne kertovat tiedon roolista järjestelmän kokonaisuudessa ja sarakkeet kuvaavat tauluun sijoitettavaa tietoa ja tiedon tyyppejä sekä mahdollisia riippuvuuksia taulujen välillä. Taulun rivit ovat varsinaista säilöttyä tietoa, minkä kuvaaminen ei ole tarpeen tietorakenteiden selvittämiseksi. Puhtaassa relaatiotietokannassa jokaisella taululla on pääavain (primary key) joka toimii taulun tietorivien yksilöivänä tunnisteena. Kun taulun jokin sarake viittaa toisen taulun pääavaimeen, puhutaan relaatiosta ja viitatun taulun pääavaimesta tulee viittaavan taulun vierasavain (foreign key).

Tietokantojen suunnittelu on usein vajaata ja kannat saavat huonon suunnittelupohjan päälle monia jälkikäteen tehtyjä optimointeja jotka voivat sotkea kantaa entisestään. Blahan tekemien tutkimusten mukaan [16] useimmista relaatiokannoista puuttuu muodolliset vierasavainten määrittelyt ja monesti rivien tyhjien arvojen käsittely on epämääräisesti tehty. Vierasavainten määrittelyn puuttuminen hankaloittaa tiedon rakenteen kuvaamista. Vierasavaimia voi löytää päättelemällä riippuvuuksia eri taulujen välillä sarakkeita ja taulujen pääavaimia vertailemalla. Jos tietorakennetta käyttävän ohjelman lähdekoodi on saatavilla, voi riippuvuuksia päätellä myös kenttien yhdistämisten (join) kautta [16].

Tietokannan käänteinen suunnittelu etenee päinvastaisissa vaiheissa kuin kannan normaali etenevä suunnittelu yleensä tapahtuu. Kolmeen vaiheeseen jaettuna käänteisen suunnittelun osat ovat:

- toteutuksen selvitys (implementation recovery),
- suunnittelumallin palauttaminen ja
- analyysimallin selvitys (analysis recovery).

Ensimmäisessä vaiheessa esitetään taulut luokkina tai käsitteinä ja taulujen sarakkeet luokkien attribuutteina. Tässä vaiheessa huomioidaan indeksit, datatyypit, tyhjien arvojen käsittely sekä pää- ja vierasavaimet sekä mahdolliset avainehdokkaat. Mallin on tarkoitus olla mahdollisimman puhdas ja selkeä. Suunnittelun selvitys vaiheessa päätavoitteena on selvittää vierasavainten viittaukset. Viimeisessä vaiheessa luodusta mallista tehdään yleisempi, abstraktimpi. Tarkoituksena on siistiä mahdolliset tietokantasuunnittelun virheet ja tarpeeton tieto. Vaiheet eivät seuraa toinen toistaan suoraviivaisesti vaan prosessi liikkuu vaiheiden välillä vapaasti. [16]

Tietokannan kuvaamista voi hidastaa ohjelmien sisäiset tietokannat tai kryptaamalla suojatut kannat, jolloin käänteistä suunnittelua jouduttaisiin lähestymään kantaa käyttävän sovelluksen toimintojen selvittämisen kautta. Tässä työssä ei tietokannalle tarvitse tehdä varsinaista käänteisen suunnittelun tutkimusta kuten ohjelmiston lähdekoodille, sillä tietokanta on vapaasti käytettävissä tietokantamanagerin kautta. Tärkeimpänä tehtävänä tiedon kuvaamisessa onkin Blahan mallin kolmen vaiheen mallien tuottaminen pääsääntöisesti olemassa olevasta tiedosta.

SpringSystem-järjestelmän tietokanta periytyy järjestelmän vanhimmalta ajalta asti. Kantamuutoksia on pyritty välttämään, jotta kantaa käyttävien sovellusten toiminta ei hajoaisi. Relaatiokanta on kuitenkin suunniteltu alusta alkaen tuntematta kunnolla relaatiomallin vaatimuksia. Näin ollen monista kannan tauluista puuttui jopa pääavaimet puhumattakaan vierasavainten määrittelystä. Paikoin taulujen ja viittausten nimitykset olivat myös epäselvät, eivätkä noudattaneet mitään sääntöä. Toteutusmallin ja suunnittelumallin luomiseksi päädyttiin korjaaman tietokannan taulujen rakennetta

niin, että se noudattaa relaatiomallin pääsäätöjä pää- ja vierasavaimista. Vasta tämän toimenpiteen jälkeen tietokannan kuvaus voitiin luoda tietokantamanagerin omia työkaluja käyttäen.

Tietokannan analyysimallin luomisessa havaittiin ongelmaksi paitsi kannan väärä rakenne, myös yleinen epävarmuus kannan joidenkin osien roolista. Kannan taulut voitiin rajata suhteellisen vaivattomasti SpringSystemin ja SpringSoftin perustoimintojen mukaisiksi tauluiksi, mutta taulujen joidenkin sarakkeiden tärkeydestä ei ollut varmuutta. Koska ohjelmisto on rakentunut pitkällä aikavälillä uusia ominaisuuksia tuomalla, on kehitys voinut tuoda käyttöön entistä parempia, vanhoja toimintoja korvaavia osia. Jos vanha osa on käyttänyt apunaan tietokannassa tiettyjä tauluja ja sarakkeita joita uusi osa ei tarvitse, voi tietokannassa hyvinkin olla turhia kenttiä. Näiden karsiminen on käytännössä mahdollista kehitysryhmän yhteisen päättelytyön kautta, mutta toisaalta karsimiselle ei ole välitöntä tarvetta. Luotu tietokannan kuvaus kuitenkin tuo kannan rakenteen ja viittaukset mahdollisine karsittavine ominaisuuksineen selkeästi esille, joten tarpeen tullen siistiminen on mahdollista aloittaa kuvauksen pohjalta.

4 KOHDEJÄRJESTELMÄN KUVAUSMENETELMÄ

Aikataulullisesti kuvausmenetelmän kehittäminen ja kohdejärjestelmän kuvaaminen koostui viidestä osasta. Vaiheet ovat järjestyksessä

- tutustuminen,
- tarve- ja tavoitemääritys,
- teoriatutkimus,
- kuvausmenetelmän laatiminen ja
- kuvauksen tekeminen.

Käytännössä osat eivät seuranneet toisiaan puhtaan kronologisesti vaan iterointia osien välillä tapahtui paljon. Tämä johtui osittain lähdeohjelman toteutusrakenteen väärästä tulkinnasta projektin alkaessa, mikä johti odotettua pidempiaikaiseen olemassa olevien teorioiden ja menetelmien tutkimukseen. Toisaalta varsinkin järjestelmään tutustuminen ja tarpeiden määrittely tapahtuivat käytännössä lähes päällekkäin. Tarpeiden ymmärrys kehittyi järjestelmän tuntemisen mukana. Myös kuvausmenetelmien kehittäminen ja kuvauksen tekeminen kulkivat käsi kädessä ja menetelmät kehittyivät paikoin kokeilun ja erehdyksen kautta.

Kuvausmenetelmän kehittämisen alkaessa sovellusaluetuntemus oli jo saavuttanut aiemman työkokemuksen ansiosta tyydyttävän perustason, mikä auttoi heti keskittymään SpringTime Oy:n tarpeiden ja tavoitteiden määrittelyyn. Ennen varsinaista tarvemääritystä oli kuitenkin perehdyttävä pinnallisesti SpringSoft-ohjelmiston rakenteeseen ja toteutukseen, sillä ilman alustavaa tietoutta lähtötilanteesta mahdollisten tarpeiden selvittäminen olisi voinut olla vaikeaa. Samoin olemassa oleviin dokumentointeihin ja kuvauksiin tutustuminen ajoittui heti projektin alkuun.

Kuvaustyö osoittautui pian suurelta osin käänteiseksi suunnitteluksi. Järjestelmästä tai SpringSoft-ohjelmistosta ei ollut olemassa kattavaa dokumentaatiota rakenteellisesti tai toiminnallisesti, joten lähtökohtana oli valmis ohjelmistotuote sekä lähdekoodi.

Arvokkaana tukena projektin läpiviennille ohjelmiston kehitysryhmä oli niin ikään käytettävissä. Käänteinen suunnittelu on kohtalaisen yleinen tehtävä ohjelmistotuotannossa, joten sen tueksi löytyy useita tutkimuksia ja dokumentoituja menetelmiä. Tämän työn teoriatutkimus kuitenkin ajautui lupaavan alun jälkeen umpikujaan, kun mitkään tutustutuista menetelmistä eivät toimineet kohdejärjestelmän kanssa. Ongelmana oli ohjelmiston epäpuhdas olio-ohjelmointi. Jollain tasolla yleisimmin teorioissa käsitellyt olio-ohjelmoinnin käänteisen suunnittelun menetelmät auttoivat kohdejärjestelmänkin tilanteessa, mutta käytännössä näitä menetelmiä käyttäen kuvaus ei olisi kyennyt esittämään pintarakennetta syvempiä toiminnallisia tai rakenteellisiakaan seikkoja. Toisaalta suurin osa teoriassa kieliriippumattomistakin menetelmistä suuntautui C++- ja Java-kielien oliomalleihin.

Teoriatutkimus osoitti, että kuvausmenetelmä oli käytännössä luotava itse. Tehty teoriatutkimus ei kuitenkaan mennyt missään nimessä hukkaan, sillä tutkitut menetelmät sisälsivät monia hyviä ideoita ja auttoivat löytämään järjestelmän tärkeimmät kuvattavat piirteet tavoitteiden täyttämiseksi. Kuvausmenetelmä keskittyy kuvaamaan SpringSoft-ohjelmiston rakennetta sekä toiminnallisuutta. Rakenne on ohjelmiston fyysinen ominaisuus ja kohdejärjestelmän tapauksessa ohjelmiston kehittämisessä osittain käytetty olio-ohjelmointi auttoi hyödyntämään oliopohjaista luokkakaavioita ohjelmiston perusrakenteen kuvaamiseen. Kukin ohjelman toimintokokonaisuuksista ja näytöistä on sijoitettu lähdekoodissa omaan luokkaansa. Myös toiminnallisuutta voisi puhtaasti oliopohjaisessa ohjelmistossa kuvata luokkarakenteen avulla ja olioiden välisenä kanssakäymisenä, mutta tähän kohdejärjestelmän sisäinen rakenne ei kyennyt yksittäisten luokkien laajuuden vuoksi. Toiminnallisuuden kuvaamiseen päädyttiin sen sijaan käyttämään näkökulmaa ohjelmiston ulkopuolelta, käyttötapausten muodossa.

Itse kuvaus päätettiin järjestelmän laajuuden vuoksi tehdä tämän työn puitteissa vain pienelle alueelle ohjelmaa. Koska kuvausmallista muodostui kerroksittainen, sopi päätös hyvin malliin. Ylimmän tason järjestelmä kuvattiin kokonaisuudessaan ja toisen ja kolmannen tason ohjelmatasoilla kuvattiin vain SpringSoft-ohjelmisto sekä tietokanta. Toiminnallisuuden kuvauksen voi laskea yhdessä tarkan rakennekuvauksen

kanssa hierarkian alimmaksi tasoksi, vaikka kuvausmenetelmä vaihtuukin täysin. Käyttötapauskuvaukset päätettiin tehdä SpringSoftin tärkeimmästä ja toisaalta monipuolisimmasta osaohjelmasta, työaikojen hallinta -näytöstä.

Tietokannan kuvaaminen oli SpringSoft-ohjelmistosta irrallinen prosessi vaikka ohjelmisto ja kanta toimivat saumattomasti yhdessä. Kannan kokonaisrakenteen kuvaaminen sijoittuu kuvauksen kerrosmallissa toiselle tasolle järjestelmän alapuolelle. Tietokannan syvempi tutkimus sijoittuu kolmannelle tasolle eli ohjelmiin verraten tarkan luokkakaavion ja käyttötapauskaavioiden tasolle (kuva 1). Tällä tasolla selvitettiin tietokannan ja ohjelmiston yhteydet. Tieto ohjelmiston käyttämistä tietokannan tauluista ja soluista koettiin tärkeäksi ennen kaikkea tietokannan kehittämiselle. Kannan ja ohjelmiston riippuvuudet merkittiin osakirjastoon, jolloin tietojen ylläpitäminen ja laajentaminen eivät vaadi kuvallisten yleiskuvausten päivittämistä.

Kaikista kuvauksen osista laadittu osakirjasto yhdistää kuvauksen kerroksia. Osakirjaston ideana on toimia kuvauksen komponenttien hakulistana koko järjestelmälle aina fyysisistä laitteista ohjelmistojen moduuleihin ja luokkiin asti. Esimerkiksi ohjelmistopuolella luokkien paikoin epäselvät nimilyhenteet eivät aina kerro selkeästi luokan tarkoituksesta tai tehtävästä. Osakirjastoon kirjattiin kaikista kuvatuista järjestelmän osista lyhyt tehtäväkuvaus sekä muita käytännöllisiksi havaittuja tietoja. Kirjastoa voisi ajatella kortistona, jossa jokaista kuvauksissa näkyvää osaa vastaa yksi kortti.

4.1 Kuvauksen osat ja kohdejärjestelmän mallikuvaukset

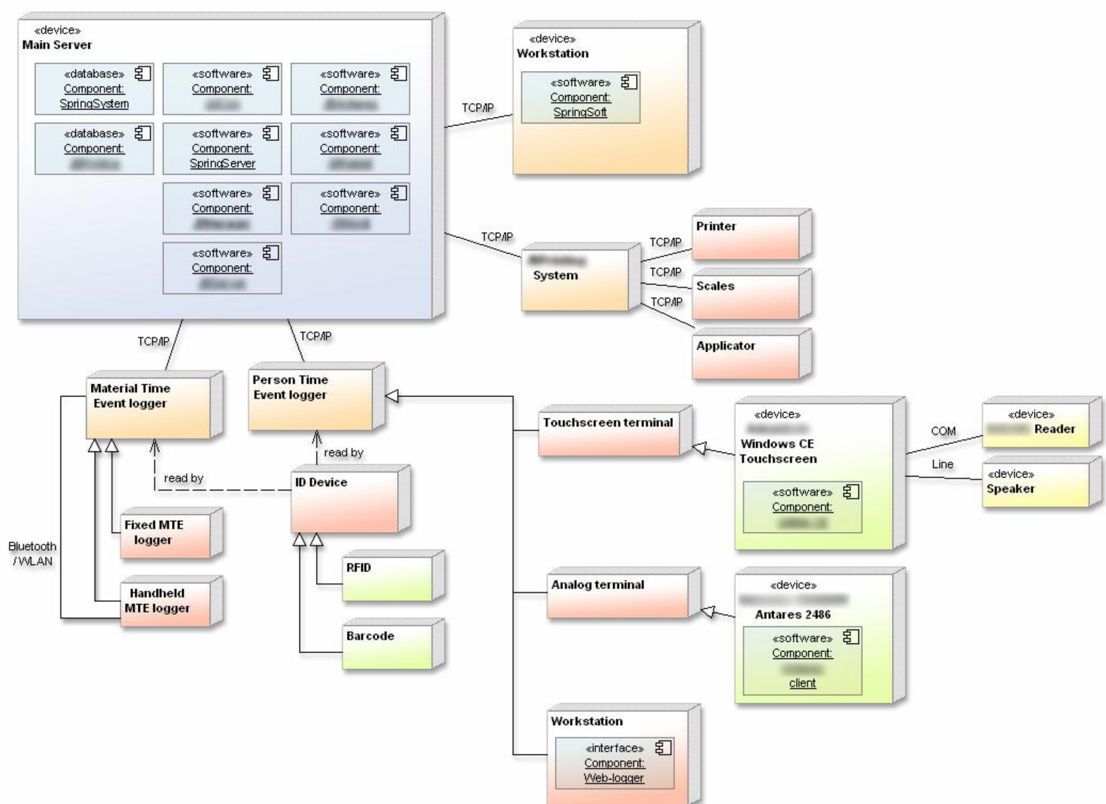
Käytännössä käytetystä kuvausmenetelmästä on mahdotonta esittää ohjelmistotuotannon vesiputousmallin tapaista vaiheittain etenevää mallia tai ohjetta. Kun tiedon määrä järjestelmästä kasvaa, mahdolliset puutteet jo tehdyissä kuvauksen osissa tulevat esiin ja kuvauksia joudutaan tutkimaan uudestaan. Tämän lisäksi tämän työn kuvausta aloittaessa ei ollut täysin varmaa, mitä tietoja kuvaukselta lopulta

halutaan. Koska nämä seikat ovat hyvin tavallisia tilanteita tutkivassa työssä, tilanne osattiin ennakoida. Kuvausmenetelmän kerrosmallin jokaisesta kerroksesta luotiin kuvausvaiheen aluksi prototyyppikuvaus joka antoi käsityksen kokonaisuudesta. Prototyypistä ja sen tekemisestä selvinneiden tietojen perusteella työn taustalla olevaa teoriatutkimusta pystyi tarkentamaan ja prototyyppiä tämän jälkeen parantamaan. Kuvauksen valmistuminen muistuttaa siis ohjelmistotuotannon iteroivaa kehitysmallia. Tässä kappaleessa esitellään malleja työssä kehitetyllä kuvausmenetelmällä luoduista kuvauksista. Kappaleen kuvat ovat otteita todellisista kuvauksista eivätkä näytä välttämättä kokonaista kuvausta, sillä todelliset kuvaukset ovat fyysisesti hyvin suurikokoisia. Kuvien tarkoitus onkin helpottaa luomaan mielikuva siitä, millaisia kuvauksia kohdejärjestelmästä luotiin.

Järjestelmätason kuvaus toteutettiin UML 2.0 kuvauskielen sijoittelukaaviota (deployment diagram) käyttämällä. SpringSystem-järjestelmä koostuu useista fyysisistä laitteista, jotka tuottavat ja hyödyntävät tietoa. Järjestelmä ei kuitenkaan ole sidottu tietyn laitemerkin tiettyihin laitemalleihin, joten järjestelmän laitteet voidaan yleistää laiteluokiksi, joista yhtä jokainen todellinen laite edustaa. Edelleen laiteluokat voitaisiin yleistää tietoa tuottaviin tiedonkeruulaitteisiin sekä tietoa käyttäviin päätteisiin, mutta käytännössä suurin osa laitteista osallistuu kumpaankin. Laiteluokat SpringSystem-järjestelmässä ovat:

- Palvelin
- Työasema
- Leimauspääte
- Sähkölukko
- Materiaalin lukulaitteet
    - RFID-lukija
    - Viivakoodilukija
- Tarratulostin
- Tarra-asetin

Kuvassa 2 näkyvä järjestelmätason kuvaus esittää näiden laiteluokkien keskinäisen rakenteen, laitteiden välisen kommunikaation ja mahdollisesti laitteissa olevat SpringSystem-järjestelmään kuuluvat ohjelmistot. Kuvaukseen on myös liitetty esimerkkejä todellisista laitteista joita järjestelmässä käytetään. Esimerkit auttavat järjestelmän rakenteeseen tutustuvaa saamaan konkreettisemman mielikuvan järjestelmän muokattavuudesta ja toisaalta auttaa mahdollista uutta työntekijää ymmärtämään yrityksen sisäistä puhekieltä, jossa laitteisiin viitataan usein vain mallinumeroilla.



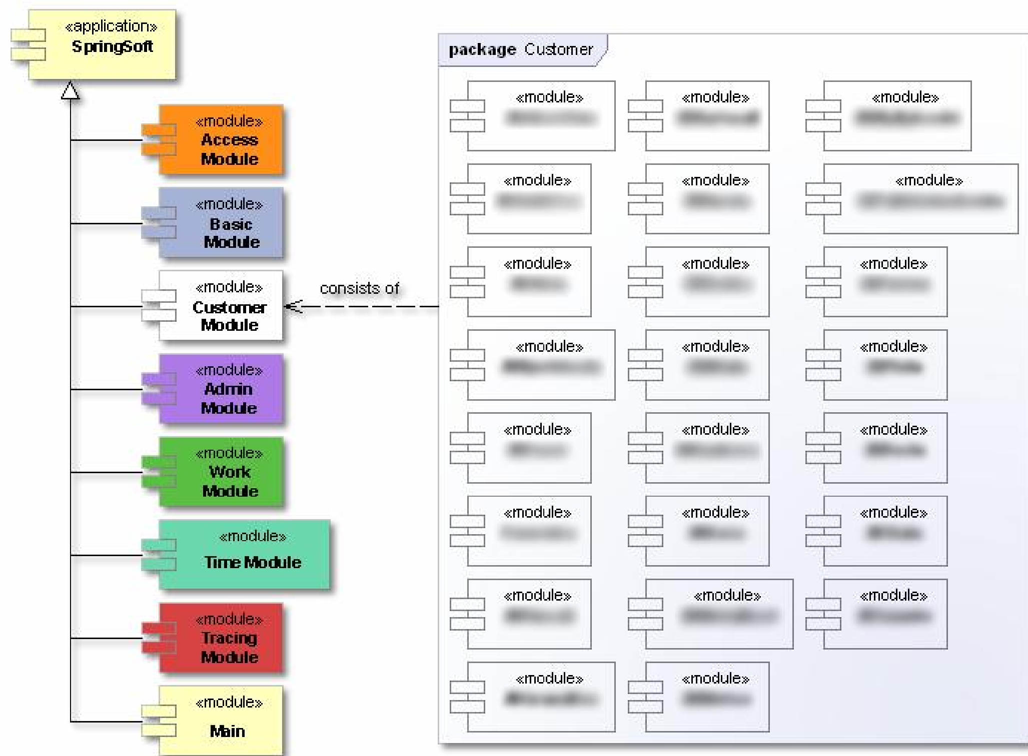
Kuva 2. SpringSystem-järjestelmä, kuvauksen ensimmäinen taso

Ohjelmistotason rakennekuvauksista voidaan tehdä tarkkuudeltaan useamman tasoisia kuvauksia. Esimerkkinä käytetyn SpringSoftin tapauksessa tarvetta on moduulitasolle, joka kuvattiin UML 2.0 pakkauskaavion (package diagram) avulla. Moduulitaso yksinkertaisesti esittää ohjelmistokokonaisuuteen kuuluvat ohjelmamoduulit. Kuvausten luettavuuden parantamiseksi moduulit väritettiin eri värein. Moduulitason kuvauksessa värityksestä ei ole hyötyä, mutta luokkatasolla hyöty tulee nopeasti esille.

SpringSoft koostuu kuudesta päämoduulista sekä vaihtelevasta määrästä asiakaskohtaisia räätälöintejä toteuttavista moduuleista. SpringSoftin osat ja niiden tehtävät ovat:

- SpringSoft – pääohjelma
- AccessModule – kulunvalvontamoduuli
- BasicModule – perustoimintomoduuli
- AdminModule – ylläpitomoduuli
- WorkModule – työaikamoduuli
- TimeModule – ajanhallintamoduuli
- TracingModule – materiaaliseurannan moduuli

Asiakasmoduulit tunnetaan järjestelmässä yleisnimellä CustomerModule. SpringSoftin pakkauskaaviossa kuvassa 3 CustomerModule on virtuaalinen pakkaus, joka koostuu asiakaspakkauksista. Varsinaiseen järjestelmätoimitukseen voidaan asiakasmoduuleista valita ko. asiakasta vastaavat lisätoiminnot. Jokainen asiakasmoduuli vastaa yhtä asiakasta, mutta jokaisella SpringTime Oy:n asiakkaalla ei välttämättä ole räätälöintejä, eikä tällöin omaa asiakasmoduuliakaan.

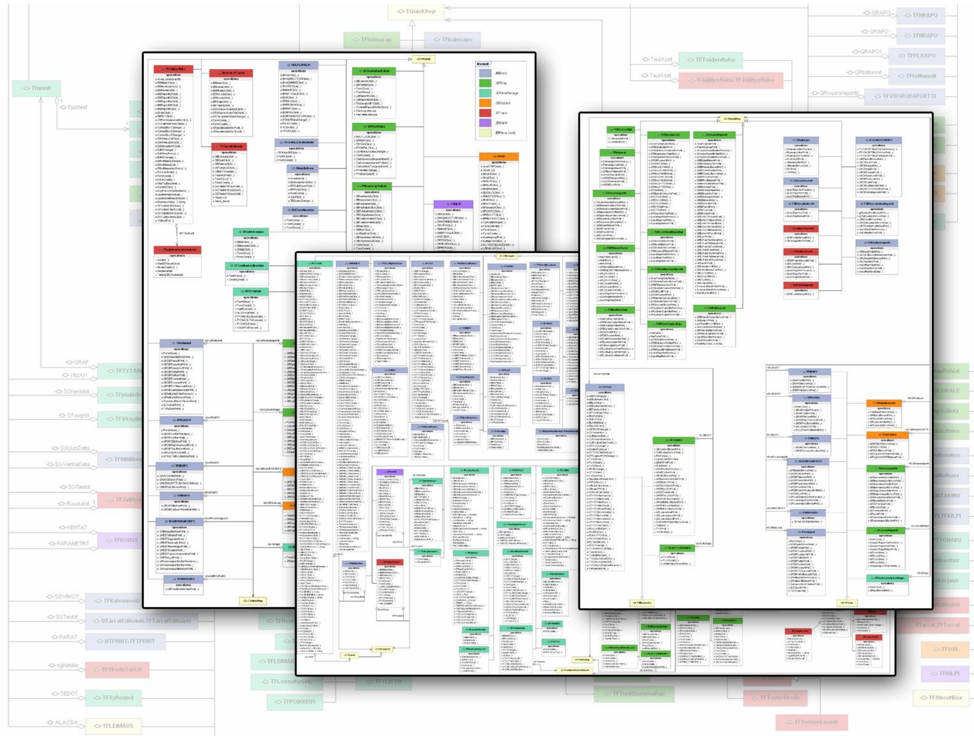


Kuva 3. SpringSoft-ohjelmiston moduulirakenne

Moduulitason kuvausta ei tarvita kaikista järjestelmän sovelluksista. Ohjelmistotason toinen taso on luokkakuvaus. Luokkakuvaus voisi periaatteessa olla hierarkkisessa kuvausmenetelmässä yhdelläkin tarkkuustasolla, mutta käytännössä SpringSoft-ohjelmistoa kuvatessa järjestelmän laajuus teki heti ilmeiseksi tarpeen karttamaisesta yleiskuvauksesta ja tätä tarkentavista luokkakuvaukista, joissa luokkien ominaisuudet tulevat ilmi. Karttatason kuvaus (kuva 4) näyttää ohjelmiston jakautumisen muutamiin tärkeimpiin pääluokkiin, joista suurin osa ohjelmiston toiminnallisista luokista on periytetty. Esimerkkeinä kuvauksessa näkyvä TSSLomake on yrityksen oma peruskomponentti ohjelmaikkunoiden luomiseen. TQuickRpt puolestaan on kolmannen osapuolen Quick Report komponentti, jota hyödynnetään SpringSoftin tuottamien raporttien ulosantiin. Tällä tasolla moduulien värittäminen tulee myös hyödylliseksi. Koska luokkarakenne kuvaa koko SpringSoft-ohjelmiston rakennetta pääluokkiin – eli luokkien luonteeseen – perustuen, on väritys toimiva tapa erotella moduulit toisistaan. Vastaava moduulijakoinen kuvaus osoittautui monimutkaisemmaksi toteuttaa. SpringSoftin luokkakuvauksen karttatasolla jokainen pääluokka on avattu niin, että niiden mahdolliset ominaisuudet tulevat esille.

Tarkempi luokkakuvaus näyttää pääluokista peritettyjen luokkien metodit. ModelMaker-sovellus rajoitti yhden kuvauksen suurimmaksi kooksi A2-paperia vastaavan koon, joten käytännössä kuvaus on jaettu useammaksi kuvaukseksi (kuva 5). Jako on tehty pääluokkien perusteella; TSSLomake ja TQuickRpt luokkia sekä muita SpringSoftin pääluokkia vastaa siis kutakin oma luokkakuvaus, jossa pääluokasta periytetyt luokat esitellään. Moduulien värikoodaus on mukana myös tarkemmassa luokkakuvauksessa.

Kuva 4. SpringSoft-ohjelmiston luokkakuvauksen ylin taso

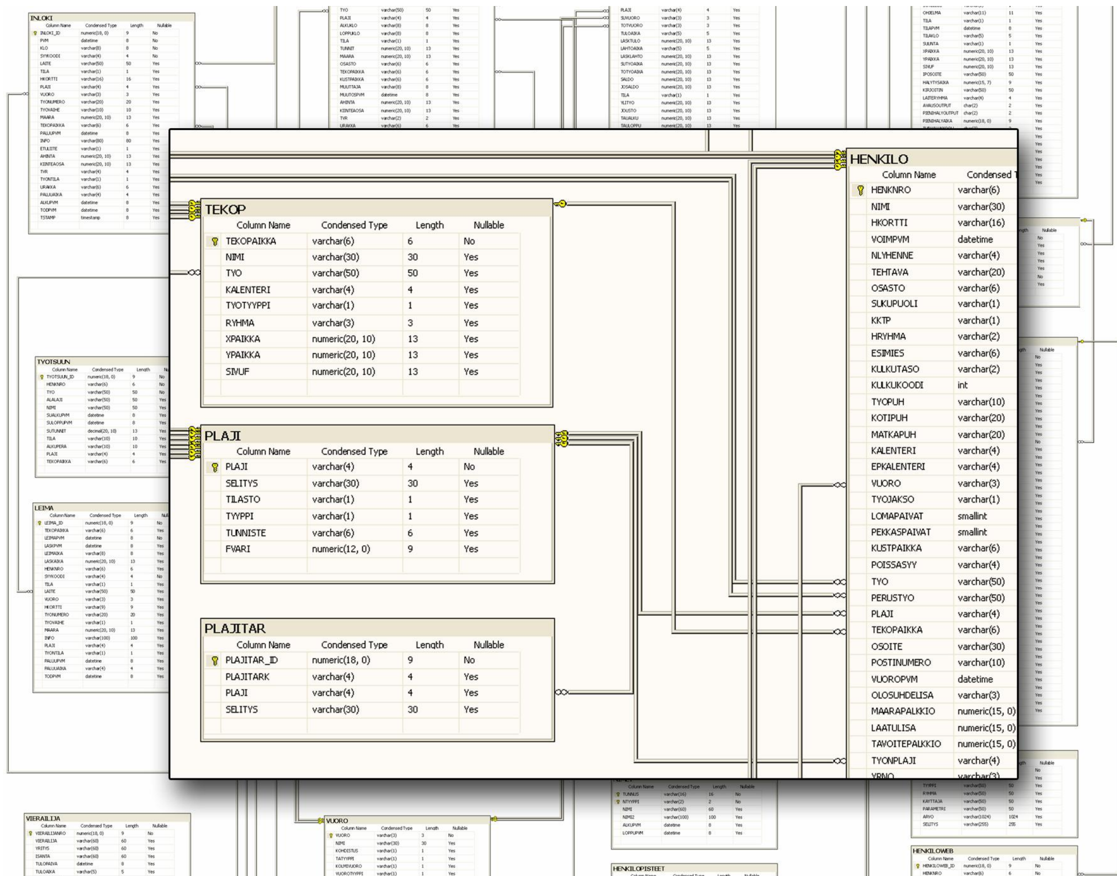


Kuva 5. SpringSoft-ohjelmiston tarkat luokkakuvaukset

SpringSoftin lisäksi SpringSystem-tietokanta kuvattiin rakenteen osalta. SpringTime Oy:n sisäisessä käytössä olevan MSSQL-tietokannan rakenteen kuvaaminen tehtiin Microsoft Visual Studion tietokantatyökaluilla puoliautomaattisesti. Kaikki tietokannassa olleet taulut eivät olleet SpringSystem-järjestelmän toiminnalle tarpeellisia, joten ensimmäisenä valittiin merkitykselliset taulut hyödyntäen ohjelmiston kehittäjien tietoutta järjestelmän rakenteesta. Visual Studion tietokantamanageri tuotti tauluista kuvauksen rungon, johon valittiin näkyviin taulun sarakkeet tietotyyppeineen ja pituuksineen.

Tietokanta ei sisältänyt vierasavaimien määrityksiä, joten tietokantamanagerin runko oli käytännössä pelkkä luettelo kannan tauluista ilman mitään riippuvuuksia taulujen välillä. Taulujen ja sarakkeiden nimistä päättelemällä taulujen väliset yhteydet oli kuitenkin nyt mahdollista selvittää. Ennen vierasavainten määrittämistä tauluista mahdollisesti puuttuvat pääavaimet luotiin taulujen yksilöiväksi tekijäksi. Korjauksella ei ollut suoraa vaikutusta tietokannan nykyiseen toimintaan tai kuvauksen sisältöön, mutta ilman pääavaimia tietokantamanagerilla ei voinut hallita riippuvuuksia ja vierasavaimia eikä näin ollen tehdä kuvausta. Tietokannan valmistelun jälkeen taulujen

riippuvuudet merkittiin vierasavaimina tietokantaan taulu kerrallaan, ja kuvaus tietokannasta oli valmis (kuva 6).



Kuva 6. Tietokannan taulujen ja vierasavainten rakennekuvaus

Vaikka yo. kuvaus kertoo paljon tietokannan sisäisestä rakenteesta ja viittauksista taulusta toiseen, se ei kerro mitään ohjelmiston ja tietokannan välisestä toiminnasta. Koska joihinkin tietokannan taulujen sarakkeisiin voidaan viitata hyvin monesta sovelluksesta ja ohjelmakomponentista, ei viittausten merkitseminen visuaalisesti olisi järkevää. Koska kuvauksen taulut ja sarakkeet ovat muiden järjestelmän osien tyylisesti mukana tekstimuotoisessa osakirjastossa, päätettiin riippuvuudet ohjelmiston ja tietokannan välillä merkitä osakirjastoon. Tietokantakyselyt ilmenevät ohjelmakoodissa SQL-kielisinä merkkijonoina, joten yhteydet etsittiin lähdekooditiedostoista ensin taulujen ja sitten sarakkeiden nimillä etsien. Tietokantakyselyt löytyvät myös Delphi kehitystyökalulla tietokantakomponenttien parametreista, mutta menetelmä vaatisi jokaisen lähdekooditiedoston läpikäyntiä yksitellen ja kantaviittausten etsiminen näin todettiin hankalammaksi kuin

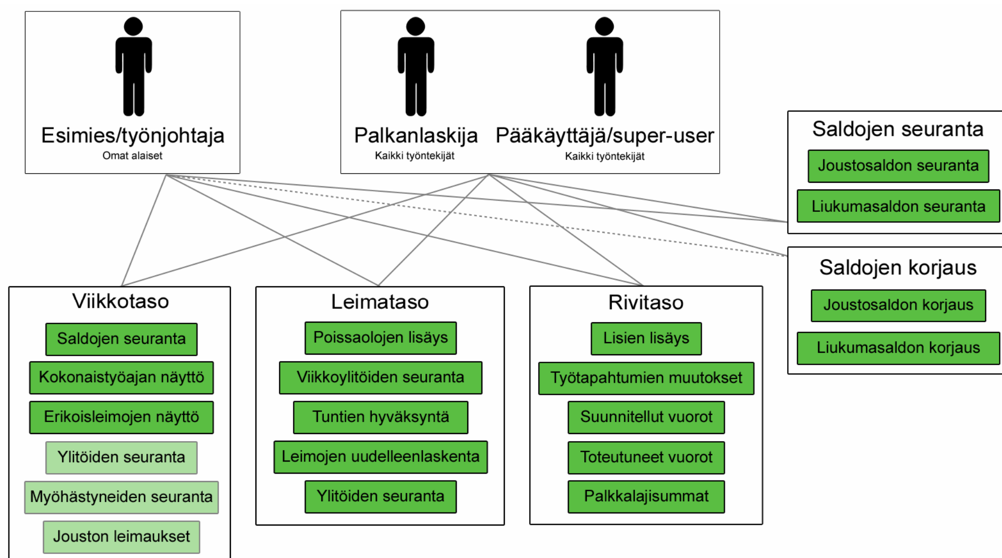
yksinkertainen tekstihaku. Löydetyt riippuvuudet merkittiin osakirjaston riippuvuus-sarakkeeseen.

Järjestelmän toiminnallinen kuvaus toteutettiin käyttötapauskaavioina. Järjestelmän laajuus pakotti rajoittamaan tämän työn merkeissä tehtävän mallikuvauksen SpringSoft-ohjelmiston tärkeimmiksi koettuihin osiin, työaikojen hallinta -näyttöön sekä raportinlaadintaan. Molempien näyttöjen kohdalla kaikki mahdolliset toiminnot kerättiin aluksi yhdeksi listaukseksi varmistamaan, että varsinaisia näytöillä suoritettavia tehtäviä kartoitettaessa suorituksen eri vivahteet tulisivat esille. Esimerkiksi näyttöjen taulukkonäkymissä rivien järjestely ja monivalinta on joissain paikoissa mahdollista ja toisaalla taas ei. Tällaisten seikkojen sisällyttäminen toiminnalliseen kuvaukseen voi auttaa laajentamaan ajatusmallia toiminnoista ja näin kehittämään nykyistä järjestelmää yhä käytettävämpään suuntaan. Toisaalta jollain tietyllä näytöllä esiintyvällä rivien järjestämisen mahdollisuudella on monessa tapauksessa harkittu tausta, mikä lisää toiminnallisuuskuvauksen arvoa toimialaosaamisen ja uuden ohjelmiston suunnittelun saralla.

Varsinaiset näytöillä suoritettavat tavoitteelliset tehtävät käytiin läpi SpringTime Oy:n järjestelmäkouluttajan kanssa yhteistyössä. Osa tehtävistä selviää myös käyttöliittymää tai ohjelmiston käyttöohjetta tutkimalla, mutta kouluttajan tietotaito ja kokemus erilaisten asiakasyritysten tarpeista toi kartoitukseen syvyyttä. Lisäksi kullekin tehtävälle voitiin kouluttajan avulla määritellä tekijä tai kohderyhmä, mille ko. tehtävä on suunnattu. Esimerkiksi yleisellä tasolla SpringSoft-ohjelmistoa käyttävät vain hallinnolliset työntekijät sekä esimiesasemassa olevat henkilöt, jotka voivat kuitenkin hallita ohjelmistolla yleensä vain omien suorien alaistensa työaikakirjauksia ja muita tietoja. Toisaalta esimerkiksi monet työtiedot ovat tärkeitä työnjohtajille sekä työntekijöille. SpringSoftin ja muun järjestelmän käyttäjärajoitteiden liittäminen toiminnallisuuskuvaukseen auttaa kehittämään nykyistä ja uutta ohjelmistoa vastaavasti kuin näyttöjen toiminnallisuuserojen kuvaaminen.

Toiminnallisuuskuvauksen ensimmäinen osa on graafinen käyttötapauskaavio (kuva 7), jonka tarkoituksena on esittää visuaalisesti ja mahdollisimman selvästi ohjelmiston

eri käyttäjäryhmien ulottuvilla olevia tehtäviä ja niiden rajoitteita. Esimerkiksi esimies voi tarkastella vain alaistensa työkirjauksia, mutta palkanlaskija voi nähdä kaikkien työntekijöiden merkinnät. Graafisessa esityksessä eri toimintojen ja tehtävien sijoittelu tiettyihin ohjelmiston moduuleihin havaittiin myös hyödylliseksi, joten kaavioon sovellettiin rakennekuvauksissa käytettyä moduulien väritystä. Lisäksi kaavioon voidaan sisällyttää parametrein ohjelmistoon lisättävissä olevia toimintoja. Kuvassa 7 tällaisia toimintoja on viikkotason kolme alinta, vaaleammalla värillä merkittyä toimintoa. Useat toiminnot ovat lisäksi parametroitavissa tietyille käyttäjäryhmille. Tällainen parametroitava yhteys käyttäjäryhmän ja toiminnon välille on esitetty katkoviivana. Tarkempaan toiminnallisuuden kuvaukseen käytettiin sanallisia käyttötapauskuvauksia ja ohjelmiston logiikan kuvauksia, joita kuitenkin luodaan vain tärkeimmistä, tarkempaa analysointia vaativista käyttöliittymän osista. Sanallisten käyttötapauskuvausten laatiminen on hitaampaa ja monesti vastaava tieto löytyy järjestelmän käyttöoppaista. Graafiseen kuvaukseen verraten sanalliset kuvaukset voivat myös muuttua merkittävästi paitsi uuden ohjelmiston myötä, myös nykyisen järjestelmän kehittyessä.



Kuva 7. Työaikojen hallinta -näytön käyttötapauskaavio

Kuvauskokonaisuuden viimeinen osa on kaikkia muita kuvauksen osia ja kerroksia yhdistävä kuvaus. Tekstimuotoinen osakirjasto sisältää lyhyen sanallisen selityksen jokaisesta kuvauksen osasta kokonaisista ohjelmistoista yksittäisiin komponentteihin.

Osakirjasto auttaa syventämään kuvausten sisältöä ilman, että graafisia kuvauksia pitää uusia. Koska graafisten kuvausten ylläpitäminen on aina monimutkaisempaa ja hitaampaa, takaa Microsoft Office Excelillä [17] toteutettu osakirjasto (kuva 8) kuvauskokonaisuudelle myös paremman päivitettävyyden. Uudet tarpeet on helppo toteuttaa ensin osakirjastoon ja vasta sen jälkeen tarvittaessa päivittää graafisia kuvauksia.

280	TFUudKasittelyTark	TimeModule	UUudKasittelyTark.pas	Kehitys	Delphi 7
281	TFUudKasittelyTarkRapo	TimeModule	UUudKasittelyTarkRapo.dfm	Kehitys	Delphi 7
282	TFUudKasittelyTarkRapo	TimeModule	UUudKasittelyTarkRapo.pas	Kehitys	Delphi 7
283	TFVKOYLI	TimeModule	UVKOYLI.dfm	Kehitys	Delphi 7
284	TFVKOYLI	TimeModule	UVKOYLI.pas	Kehitys	Delphi 7
285	TPAIVA	TimeModule record	UVKOYLI.pas		Delphi 7
286	HenkTun	TimeModule record	UVKOYLI.pas	Kehitys	Delphi 7
287	TFVKOYLI2	TimeModule	UVKOYLI2.dfm	Kehitys	Delphi 7
288	TFVKOYLI2	TimeModule	UVKOYLI2.pas	Kehitys	Delphi 7
289	TFVkoylimuutos	TimeModule	uvkoylimuutos.dfm		Delphi 7
290	TFVkoylimuutos	TimeModule	uvkoylimuutos.pas	Kehitys	Delphi 7
291	TFvkylitapahtumat	TimeModule	Uvkylitapahtumat.dfm	Kehitys	Delphi 7
292	TFvkylitapahtumat	TimeModule	Uvkylitapahtumat.pas	Kehitys	Delphi 7
293	TFYLITYO	TimeModule	UYLITYO.dfm	Kehitys	Delphi 7
294	TFYLITYO	TimeModule	UYLITYO.pas	Kehitys	Delphi 7
295		TracingModule MITÄTÖN	CNUJaljitysHaku.dfm		Delphi 7
296	TFJaljitysHaku	TracingModule	JaljitysHaku.pas	Kehitys	Delphi 7
297		TracingModule MITÄTÖN	MversioUJaljitysHaku.pas		Delphi 7
298	UJaljitysHaku.TFJaljitysHaku	TracingModule	UJaljitysHaku.dfm	Kehitys	Delphi 7
299	UJaljitysHaku.TFJaljitysHaku	TracingModule	UJaljitysHaku.pas	Kehitys	Delphi 7
300	TFJaljitysSelaus	TracingModule	UJaljitysSelaus.dfm	Kehitys	Delphi 7
301	TFJaljitysSelaus	TracingModule	UJaljitysSelaus.pas	Kehitys	Delphi 7
302		TracingModule pääohjelma	USSTracing.pas	Kehitys	Delphi 7
303	TFKilvet	TracingModule	UKilvet.dfm	Kehitys	Delphi 7
304	TFKilvet	TracingModule	UKilvet.pas	Kehitys	Delphi 7
305	TTaulukoidenAsetukset	TracingModule	UNKentat_TR.dfm	Kehitys	Delphi 7
306	TTaulukoidenAsetukset	TracingModule	UNKentat_TR.pas	Kehitys	Delphi 7
307	TFNaytettavatKentat	TracingModule	UNKentat_TR.pas		Delphi 7
308	TQJaljitysRaportti	TracingModule	UQRJaljitysRaportti.dfm	Kehitys	Delphi 7
309	TQJaljitysRaportti	TracingModule	UQRJaljitysRaportti.pas	Kehitys	Delphi 7
310	TFRivinMuokkaus	TracingModule	URivinMuokkaus.dfm	Kehitys	Delphi 7
311	TFRivinMuokkaus	TracingModule	URivinMuokkaus.pas	Kehitys	Delphi 7

Kuva 8. Otos osakirjastosta

5 POHDINTAA

Työssä selkeästi suurimpaan rooliin asettui SpringSystem-järjestelmän keskuksena toimiva SpringSoft-ohjelmisto. Tarkoituksena oli kuvata SpringSoft-ohjelmiston lähdekooditason rakennetta ja toiminnallisuutta sekä sen osuutta koko SpringSystem-järjestelmässä. Yleisesti kuvauksilta toivottiin nykyistä helpompaa lähestymistapaa ohjelmiston ja koko järjestelmän toimintaan niin ohjelmoijille kuin muillekin sidosryhmille nykyisen ohjelmiston ylläpitoa ja kehittämistä ajatellen. Tavoitteina oli järjestelmään sitoutuneen sovellusalueosaamisen kuvaaminen, mahdollisuus käyttää kuvauksia uuden työvoiman perehdyttämiseen, sekä uuden ohjelmiston suunnittelun pohjustaminen. Toissijaisia tavoitteita oli kuvauksen ylläpidettävyys, yhtenäisyys ja standardinmukaisuus.

Työn päätavoitteet saavutettiin hyvin. Järjestelmän rakenteen ja toiminnallisuuden kuvaaminen tehtiin erillään toisistaan, mikä tukee tavoitetta kuvata sovellusalueosaamista. Rakenteen ratkaisuihin vaikuttaa tietotaito eritellä sovellusalueen eri osia ohjelman rakenteellisiksi osiksi. Toiminnalliset valinnat puolestaan kertovat sovellusalueen tehtävien sekä syy-seuraus-yhteyksien ymmärtämisestä. Molemmat käyvät esille luoduista kuvauksista. Rakenteen ja toiminnallisuuden erottaminen toisistaan auttoi myös pitämään kuvauksen selkeänä ja helppolukuisena. Tämä helpottaa paitsi vähemmän ohjelmiston rakennetta ymmärtävien sidosryhmien perehdyttämistä, myös kuvausten ylläpitoa.

Kuvaus lähtee liikkeelle koko fyysisen järjestelmän kuvauksesta tarkentuen siitä ohjelmistoihin sekä tietokantaan ja niiden rakenteeseen. Tämä helpottaa hahmottamaan kunkin ohjelmiston, tietokannan taulun ja komponentin paikkaa ja roolia koko järjestelmässä. Graafisissa rakennekuvauksissa käytettiin myös SpringSoft-ohjelmiston kohdalla värikoodausta ohjelmiston moduulien erottamiseksi toisistaan. Tämä ratkaisu tukee ratkaisujen näkemistä sekä luettavuutta, sillä SpringSoftin luokkakuvauksesta voi nopeasti nähdä koko ohjelmiston tasolla esim. ohjelmistoon sisältyvät raportit tai

mistä pääluokasta mikäkin näyttö on periytetty, sisällyttäen silti myös moduulijaon näkyvyyden samassa kuvauksessa.

Toiminnallisuuden kuvaaminen päädyttiin toteuttamaan ohjelmiston ulkopuolisesta näkökulmasta. Menetelmän käyttötapauskaaviot ovat paitsi helppolukuisia, myös helposti täydennettävissä. Toiminnallisuuskuvaukset eivät kuitenkaan kykene esittämään luokka- tai oliotasolla tiettyihin toimintoihin osallistuvia luokkia. Käytännössä SpringSoft kuitenkin rakentuu niin, että yleensä yhden loppukäyttäjälle näkyvän näytön takana toimii vain yksi luokka. Näin ollen kuvauksen viimeisenä osana oleva osakirjasto kykenee usein vastaamaan ongelmaan. Osakirjasto kertoo sanallisessa muodossa kunkin ohjelmiston osan ja komponentin tehtävän sekä mm. komponentin vastuukehittäjän ja riippuvuudet.

Rakenteellinen ja toiminnallinen kuvaus yhdessä auttavat varmistamaan, että uutta järjestelmää suunniteltaessa tietotaito tarvittavista toiminnoista ja järjestelmän osista on olemassa. Uuden ohjelmiston rakenne voidaan alusta asti suunnitella tietäen tarkasti nykyisen järjestelmän rakenne ja toiminnallisuus. Uuden järjestelmän toimintalogiikka ja rakenne voidaan miettiä kokonaan uudestaan sitoutumatta vanhan ohjelmiston ratkaisumalleihin. Samalla voidaan varmistua siitä, että uusi järjestelmä tukee kaikkia vanhassa järjestelmässä tarpeellisiksi havaittuja ominaisuuksia. Lisäksi etenkin toiminnallinen kuvaus voi auttaa arvioimaan tulevaisuuden tarpeita ja laajentamaan nykyisen järjestelmän toiminnallisuutta kiertämällä havaittuja rajoitteita uuden rakennesuunnittelun avulla.

Toissijaisista tavoitteista ylläpidettävyys nähtiin lopulta tärkeimmäksi. Graafisten kuvausten ylläpito vaatii aina manuaalista käsittelyä etenkin osien sijoittelussa, joten osakirjasto helposti ylläpidettävänä Excel-tiedostona on hyvä lisä kokonaisuuteen. Yhtenäisyys eri järjestelmän osien välillä toteutui kuvausmenetelmässä onnistuneesti, vaikka toisaalta tämän työn merkeissä kuvattiinkin vain pieni osa koko järjestelmästä. Rakennekuvauksen kerroksittaisuus on kuitenkin helposti sovellettavissa isommista ohjelmistoista pienimpiin ja mahdollisuus moduulien tai muiden rakenteellisten osien erotteluun värikoodauksella on olemassa. Rakennekuvaus soveltuu periaatteessa myös

laitteiston kuvaamiseen, vaikkei sille toistaiseksi olekaan tarvetta. Toiminnallisuuden kuvaaminen ei myöskään ole sidottu pelkästään ohjelmistoihin – käyttötapaukset soveltuvat monipuolisesti kaikkeen toiminnallisuuden kuvaamiseen. Siirrettävyyden tavoite standardien mukaisen kuvauksen kautta ei sen sijaan toteutunut tavoitteiden mukaisesti. Ohjelmisto on rakennettu osittaista olio-ohjelmointia käyttäen, mikä johti kuvausstandardien tapauskohtaiseen soveltamiseen päätavoitteiden saavuttamiseksi.

Luotu kuvausmenetelmä mahdollistaa tavoitteen mukaisesti järjestelmän laajemman kuvaamisen tulevaisuudessa. Käytännössä ohjelmiston lisäksi käytetyllä menetelmällä voitaisiin kuvata myös laitejärjestelmää, mutta SpringTime Oy:n tavoitteet asettuvat lähiaikoina lähinnä uuden ohjelmiston suunnitteluun. Käytettyjä menetelmiä ja saatua kuvauksesta kokemusta voidaan lisäksi hyödyntää suoraan uuden järjestelmän kehittämisessä. Rakenteelliseen kuvaamiseen käytetty ModelMaker-ohjelmisto mahdollistaa uuden järjestelmän suunnittelun visuaalisesti luokkakuvauksia käyttäen, ja ohjelmiston kuvausta voidaan pitää yllä kehityksen aikana alusta alkaen. Työkalu mahdollistaa lähdekoodin ja kuvauksen saumattoman yhteiskäytön kehityksen aikana, mutta nähtäväksi jää, kuinka hyödylliseksi ja toimivaksi tällainen edestakaisen suunnittelun toimintamalli koetaan. Toiminnallisuuden kuvaamiseen käytetyt käyttötapauskaaviot käyvät sellaisenaan myös uuden järjestelmän kuvaukseen ja kuvauksia voidaan luonnollisesti laajentaa ja kehittää uuden järjestelmän tarpeiden mukaisesti. Kuvaukset siis auttavat uuden ohjelmiston suunnittelijoita hahmottamaan nykyisen järjestelmän laajuutta ja sen osien yhteyksiä, mutta toisaalta käytettyjä menetelmiä ja työkaluja soveltamalla voidaan luoda kuvauksia uudesta ohjelmistosta jo sen suunnitteluvaiheessa.

6 YHTEENVETO

Tämä diplomityö on osa Lappeenrannan teknillisen yliopiston ja Etelä-Karjalan ammattikorkeakoulun RIGHT-projektia, jonka tarkoituksena on tutkia tietotekniikan ja ohjelmistotuotannon palveluiden kansainvälistyvän luonteen merkitystä suomalaisessa yritysmaailmassa. Työssä tutkitaan kuvausmenetelmiä erään pitkän kehityskaaren teollisen työajan- ja materiaalinseurannan ohjelmiston kuvaamiseen. Ohjelmiston ja sen taustalla olevan järjestelmän kuvaaminen on tullut ajankohtaiseksi, sillä monimutkaistuneen järjestelmän ylläpidettävyys on vaikeutunut ja toisaalta uutta korvaavaa ohjelmistoa aletaan pian suunnitella. Ohjelmistoa kehitettäessä on myös dokumentointi jäänyt vähäiseksi, minkä vuoksi järjestelmästä ei ole olemassa kattavia rakenteellisia tai toiminnallisia kuvauksia.

Kuvausmenetelmän tavoitteina oli hyödynnettävyys uuden ohjelmiston suunnittelussa, nykyisen järjestelmän ylläpidossa sekä uuden työvoiman perehdyttämisessä. Uuden ohjelmiston kehityksessä aiotaan hyödyntää ulkoista työvoimaa, ja myös ulkomaisen työvoiman käyttämistä on harkittu. Tämän mahdollistamiseksi nykyisen järjestelmän rakenteen ja toiminnan ymmärtämistä on tehostettava; nykymenetelmin uuden työvoiman perehdyttämiseen kuluisi jopa kaksi kuukautta. Uuden ohjelmiston suunnitteluun puolestaan keskittyy kuvausmenetelmän tavoite kuvata järjestelmään ja ohjelmistoon sitoutunutta sovellusalueosaamista.

Kehitetty kuvausmenetelmä muodostuu graafisista ja sanallisista osista, jotka muodostavat yhdessä hierarkkisen, kerroksittain syvenevän kuvauksen. Ylimmällä tasolla kuvataan fyysinen laitteiden ja ohjelmistojen muodostama järjestelmä. Toisella tasolla kuvataan järjestelmän osat yleisellä tasolla. Tämän työn SpringSystem-kohdejärjestelmässä keskityttiin järjestelmän keskeisen SpringSoft-ohjelmiston sekä järjestelmän taustalla toimivan tietokannan kuvaamiseen, joten kuvauksen toisella tasolla ovat SpringSoft-ohjelmiston sekä tietokannan yleiskuvaukset. SpringSoftin yleiskuvaukseen kuuluu ohjelmiston moduulirakenteen selvittävä kuvaus sekä yleisen tason luokkakuvaus. Yleinen luokkakuvaus toimii karttana tarkemmalle lähdekoodin

luokkakuvaukselle, joka esittelee luokkien ominaisuudet ja funktiot. Luokkakuvauksissa on käytetty värikoodausta, joka sitoo kuvauksen luokat SpringSoft-ohjelmiston toiminnallisiin moduuleihin. Tietokannan kuvauksessa kuvataan taulujen rakenne yhteydet pää- ja vierasavainten muodossa. Kuvausmenetelmän viimeinen osa on osakirjasto, joka yhdistää kaikkia kuvauksia kertomalla kuvauksissa esiintyvien osien ja komponenttien tehtävistä sanallisessa muodossa. Osakirjastossa kuvataan myös ohjelmiston ja tietokannan välillä esiintyvät riippuvuudet.

Kehitettyä kuvausmenetelmää käyttäen järjestelmästä luotiin diplomityössä esimerkkikuvaus valituista järjestelmän osista. Kuvausta on tulevaisuuden tarpeiden mukaan helppo laajentaa paitsi SpringSoft-ohjelmiston sisällä, myös järjestelmän muihin sovelluksiin. Käytännössä hierarkkinen kuvausmenetelmä mahdollistaisi myös laitteiston tarkemman kuvaamisen samaa menetelmää käyttäen, mutta tähän ei kohdejärjestelmän kohdalla nähty tarvetta.

LÄHDELUETTELO

[1] Haikala & Märijärvi; Ohjelmistotuotanto. Suomen Atk-kustannus Oy, 1998. ISBN 951-762-696-7.

[2] Pressman, R. & Ince; Software Engineering – A Practitioner's Approach, European Edition. D. McGraw-Hill, 2000. ISBN 007-709-677-0.

[3] Borland, URL: http://www.borland.com (viitattu 27.12.2006)

[4] Lanza, M., Ducasse, S.; Polymetric Views – A Lightweight Visual Approach to Reverse Engineering. IEEE Transactions on Software Engineering, volume 29, issue 9, IEEE September 2003, pp 782 – 795.

[5] Parsons, J., Cole, L.; What do the pictures mean? Guidelines for experimental evaluation of representation fidelity in diagrammical conceptual modelling techniques. Data & Knowledge Engineering, volume 55, issue 3, Elsevier December 2005, pp 327-342.

[6] Eynard, B., Gallet, T., Roucoules, L., Ducellier, G.; PDM system implementation based on UML. Mathematics and Computers in Simulation, volume 70, issues 5-6, Elsevier February 2006, pp 330-342.

[7] Anquetil, N.; Lethbridge, T.C.; Comparative study of clustering algorithms and abstract representations for software remodularization. IEE Proceedings – Software, volume 150, issue 3, IET June 2003, pp 185 – 201.

[8] Chikofsky, E.J.; Cross, J.H., II; Reverse Engineering and Design Recovery: A Taxonomy. IEEE Software, volume 7, issue 1, IEEE January 1990, pp 13 – 17.

[9] Knodel, J., Calderon-Meza, G.; A Meta-Model for Fact Extraction from Delphi Source Code. Electronic Notes in Theoretical Computer Science, volume 94, Elsevier May 2004, pp 19-28.

[10] Wong, K.; Tilley, S.R.; Muller, H.A.; Storey, M.-A.D.; Structural Redocumentation: A Case Study. IEEE Software, volume 12, issue 1, IEEE January 1995, pp 46 – 54.

[11] Demeyer, S., Ducasse, S., Tichelaar, S.; Why Unified is not Universal – UML Shortcomings for Coping with Round-trip Engineering. UML'99 Proceedings, Software Composition Group, University of Berne.

[12] ModelMaker Tools, URL: http://www.modelmakertools.com (viitattu 14.2.2007)

[13] Microsoft Visual Studio, URL: http://msdn.microsoft.com/vstudio/vshome.aspx (viitattu 15.2.2007).

[14] Stone, D. et. al., User Interface Design and Evaluation. Morgan Kauffman, 2005. ISBN 012-088-436-4.

[15] Willard, B.; UML for systems engineering. Computer Standards & Interfaces, volume 29, issue 1, Elsevier January 2007, pp 69-81.

[16] Blaha, M.R.; The Case for Reverse Engineering. IT Professional, volume 1, issue 2, IEEE March-April 1999, pp 35 – 41.

[17] Microsoft Office Excel, URL: http://office.microsoft.com/en-us/default.aspx (viitattu 28.2.2007).