Mikko Lampi

MOBIILIRATKAISUT TIETOJÄRJESTELMIEN KEHITYSVÄLINEENÄ

Epicor For Service Enterprises -

toiminnanohjausjärjestelmän kehittäminen

Opinnäytetyö
Tietotekniikan koulutusohjelma

Huhtikuu 2010

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

KUVAILULEHTI

MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences	**Opinnäytetyön päivämäärä** **25.4.2010**
Tekijä(t) Mikko Lampi	**Koulutusohjelma ja suuntautuminen** **Tietotekniikan koulutusohjelma** **Ohjelmistotekniikka**

Nimeke

Mobiiliratkaisut tietojärjestelmien kehitysvälineenä
Epicor For Service Enterprises -toiminnanohjausjärjestelmän kehittäminen

Tiivistelmä

Opinnäytetyöni käsittelee tietojärjestelmien, erityisesti toiminnanohjausjärjestelmien, kehittämistä mobiiliteknologian avulla. Työ toteutettiin toimeksiantona ohjelmistoalalla toimivalle Smart Time Oy:lle. Pääpainotuksena opinnäytetyöni kehitysprojektissa olivat liike-elämä- ja tarvelähtöisyys, projektityöskentely ja uusimman mobiiliteknologian hyödyntäminen.

Nykyään lähes jokaisella organisaatiolla on käytössään yksi tai useampi toiminnanohjausjärjestelmä. Järjestelmien avulla organisaatio ohjaa liiketoimintaansa ja niiden tarjoamaa tietoa hyödynnetään niin päätöksenteossa kuin prosessien kehittämisessäkin. Usein järjestelmien tehokkaaseen käyttöön voi liittyä ongelmia mm. käytettävyyden tai joustavuuden suhteen. Sovellukset ovat raskaita, monimutkaisia ja vaativat käyttökoulutuksen. Toisaalta ne ovat välttämättömyys tehokkaalle toiminnalle.

Opinnäytetyöni kehitysprojektina suunnittelin Epicor For Service Enterprises eli E4SE-toiminnanohjausjärjestelmälle mobiililaitteille tarkoitetun asiakassovelluksen. Sovelluksen avulla käyttäjät voivat syöttää projektikohtaisia tunti- ja kulukirjauksia järjestelmään. Kirjaukset voivat tapahtua missä tahansa ja milloin tahansa käyttäjän ollessa Internet-yhteydessä. Tarkoitukseni oli luoda käyttäjälähtöinen ja moderni sovellus, joka helpottaa järjestelmän jokapäiväistä käyttöä.

Avainasemassa ratkaisun kehittämisessä olivat nykyaikaiset ohjelmistotuotannon menetelmät ja uusin mobiiliteknologia. Toteutusprojekti suunniteltiin Java ME -ohjelmistoalustalle ja ratkaisussa hyödynnettiin palveluorientoitunutta sovellusarkkitehtuuria. Suunnitellun sovelluksen vaatimuksia olivat tietoturvallisuus ja miellyttävä käyttökokemus. Projektin lopputuloksena syntyivät määrittely- ja suunnitteludokumentaatiot, joiden avulla toteutusprojekti suoritetaan vuoden 2010 aikana. Valmistuessaan sovellus on ensimmäinen alustariippumaton E4SE-järjestelmälle toteutettu projektikirjaussovellus.

Työn teoriaosuudessa perehdyin toiminnanohjausjärjestelmiin, ohjelmistotuotantoon, mobiiliteknologian nykytilaan ja mobiiliohjelmointiin.

Asiasanat (avainsanat)

tietojärjestelmät, toiminnanohjaus, Java, J2ME, langaton tekniikka, ohjelmistotuotanto, ohjelmistokehitys

Sivumäärä	**Kieli**	**URN**
60 s. + liitteet 3 s.	Suomi	URN:NBN:fi:mamk-opinn201035623

Huomautus (huomautukset liitteistä)

Ohjaavan opettajan nimi	**Opinnäytetyön toimeksiantaja**
Timo Mynttinen	Smart Time Oy

DESCRIPTION

MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences	**Date of the bachelor's thesis** 25.4.2010
Author(s) Mikko Lampi	**Degree programme and option** Information Technology Software Engineering

Name of the bachelor's thesis

Mobile software solutions in information systems development
Enhancing Epicor For Service Enterprises ERP

Abstract

This thesis is about developing enhancements for information and enterprise resource planning systems by the means of mobile technology. It was carried out as an assignment by Smart Time Ltd. The key concepts of the thesis project were business-centric requirements, project oriented implementation and utilization of the latest mobile technology.

Today almost each organization makes use of some kind of an enterprise resource planning software. The software profoundly enhances the capabilities and efficiency of administrating and planning the business processes in general. However, they tend to be massive and complex systems which are hardly considered dynamic, user friendly or easy to master. At the same time they're pretty much necessity in modern cost-efficiency driven business world.

As a development project I specified and designed a mobile client for E4SE ERP. The application's purpose was to provide easy and convenient functionality for entering time and expense entries against projects in E4SE system. My objective was to design modern user-centred mobile application to ease the daily use of E4SE. When completed the application would be first platform independent mobile client for the system. I had planned to finalize the implementation project during 2010.

The primary means to achieve the objectives were up-to-date and agile development philosophies and intelligent use of the latest mobile technology. The software was planned to be implemented using Java ME platform and service oriented architecture. The essential requirements for the software design were data security and excellent user experience.

The theory sections of this thesis familiarize the concepts of ERP systems, software engineering, current mobile technology and principles of mobile software development.

Subject headings, (keywords)
information systems, enterprise resource planning, Java, J2ME, wireless technology, software engineering, software development

Pages	**Language**	**URN**
60 p. + appendices 3 p.	Finnish	URN:NBN:fi:mamk-opinn201035623

Remarks, notes on appendices

Tutor	**Bachelor's thesis assigned by**
Timo Mynttinen	Smart Time Ltd.

LYHENTEET JA TERMISTÖ

API	*Application Programming Interface*
E4SE	Epicor For Service Enterprises, Epicorin toiminnanohjaus-järjestelmä
ERP	*Enterprise Resource Planning*, toiminnanohjausjärjestelmä
iScala	Epicorin toiminnanohjausjärjestelmä
J2ME	*Java 2 Micro Edition*
JDK	*Java Development Kit*
JVM	*Java Virtual Machine*
KISS	*Keep It Simple and Short* -sovelluskehitysfilosofia
MID	*Mobile Internet Device*
MIDP	*Mobile Information Device Profile*
MRP	*Material Resource Planning*, materiaalinohjausjärjestelmä
MRP II	*Manufacturing Resource Planning*, tuotannonohjausjär-jestelmä
OOP	*Object Oriented Programming*
PSA	*Professional Services Automation*
SaaS	*Software as a Service*
SDK	*Software Development Kit*
SOA	*Service Oriented Architecture*
VPN	*Virtual Private Network*

SISÄLTÖ

1 JOHDANTO

Nykyaikainen organisaatio kerää, hallinnoi ja hyödyntää valtavat määrät erilaista tietoa. Oli sitten kyseessä voittoa tavoitteleva tai yhteisöllinen organisaatio, kustannustehokkuus ja kehittyminen omalla toimialalla määrittelevät liiketoimintaprosesseja ja muuta toimintaa. Ennen tietoteknisten ratkaisujen yleistymistä organisaatioissa tiedonkeruu, analysointi ja hyötykäyttö tapahtuivat manuaalisesti erilaisten kausittaisten raporttien ja lukuisten ihmistyötuntien avulla. Nykyään lähes kaikki organisaatiot hyödyntävät jonkinlaista tietojärjestelmää. Poikkeuksena kenties pienet yritykset, joilla ei ole taloudellisia tai muista resursseista riippuvia mahdollisuuksia tietojärjestelmien käyttöön. Tulevaisuudessa sekin muuttunee, sillä erilaiset SaaS-malliin perustuvat ohjelmistoratkaisut ja tietojärjestelmät yleistyvät nopeasti. Uudet teknologiat kuten pilvipalvelut mahdollistavat järjestelmien joustavan ja edullisen tarjonnan yhä pienemmille organisaatioille.

Yrityksen toimintaa ohjaavat tietojärjestelmät ovat yleensä laajoja, monimutkaisia ja kalliita ohjelmistoratkaisuja. Usein järjestelmän eri komponentit ja osajärjestelmät voivat olla hankittu eri toimittajilta, eri ajankohtina ja ne kohdistuvat erilaisiin tarpeisiin. Yleisimpiä tietojärjestelmätyyppejä ovat mm. toiminnan-, tuotannon-, materiaalin-, ja palvelunohjausjärjestelmät sekä asiakkuudenhallintajärjestelmät. Voidaan olettaa, että jokaisella vähintään keskisuurella tai suuremmalla organisaatiolla on jonkinlainen tietojärjestelmä tai useampia käytössään. Edellä mainituista tietojärjestelmistä aion opinnäytetyössäni perehtyä ERP- eli toiminnanohjausjärjestelmiin.

Ongelma ei nykyään välttämättä ole järjestelmien tai niiden hyödyntämisen puute, vaan erilaisten tietojärjestelmien kirjo ja niiden käyttämiseen liittyvät vaikeudet. Ongelmakohdiksi voivat muodostua mm. järjestelmien vanhanaikaisuus, monimutkaisuus tai vaikkapa niiden käyttö yrityksen sisäverkon ulkopuolelta. Esimerkkitapauksena mainittakoon kuvitteellinen konsulttiyritys. Sen työntekijät ovat useammin asiakkaan tiloissa, tai tien päällä, kuin organisaation sisäverkon tai muun yritysinfrastruktuurin ulottuvissa. Tästä seuraa uudenlaisia ongelmia liittyen mm. ajantasaisiin tunti- ja kulukirjauksiin. Kiinteästi organisaation intranet-verkkoon rajattuun järjestelmään tapahtuvat kirjaukset on tehtävä joko omalta päätteeltä yrityksen tiloista tai vastaavasti jäykähkön VPN-tunnelin kautta rajatuista päätepisteistä. Vaikeasti suoritettavilla kirjauk-

silla on useasti tapana lykkääntyä ja yksityiskohdilla unohtua. Apukeinona voidaan käyttää muita muistiinpanomenetelmiä, mutta ne eivät palvele alkupäistä ideaa.

Pureudun opinnäytetyössäni toiminnanohjausjärjestelmien kehittämiseen käyttäjäystävällisempään ja joustavampaan suuntaan mobiiliteknologian keinoin. Mobiililla päätelaitteella erilaisten kirjausten tekeminen ja tietojen molemminpuolinen päivitys voidaan tehdä käyttäjän ehdoilla sijainnista riippumatta. Kun otetaan huomioon riittävä tietoturva, tarve kirjautua organisaation sisäverkkoon saadaan häivytettyä ja transaktiot voidaan suorittaa joustavasti keventäen tietojensyöttöprosessia. Edellä mainitun kaltainen mobiilisovellus vähentäisi järjestelmän käyttäjän muistinvaraisia kirjauksia, jotka yleensä tehdään takautuvasti enemmän tai vähemmän säännöllisissä jaksoissa. Tällöin tiedot olisivat ajantasaisempia ja tarkempia sekä vältyttäisiin tietyltä mahdollisuudelta inhimillisiin muistiin pohjautuviin virheisiin. Todennäköistä on myös, että kirjaukset hoituisivat tehokkaammin ja vähentäisivät ajankäyttöä vapauttaen kallisarvoista työaikaa olennaisemmille tehtäville.

Haasteina työn toteutuksessa uskon nousevan esiin mm. integraation mobiilisovelluksen ja ERP-järjestelmän välillä. Kaikki tietoliikenne tulisi toteuttaa luotettavasti ja tietoturvallisesti, mutta niin mobiililaitteiden suorituskyky kuin tietoliikennekaistan nopeuskin asettavat omat rajoitteensa. Käytettävyyden ja yhtenäisen käyttökokemuksen suunnittelu laajalle laitevalikoimalle aiheuttaa todennäköisesti myös suunnitteluongelmia ja mahdollisia priorisointeja eri laitealustojen välillä.

Uskon työlläni olevan merkitystä mobiilisovellusten mahdollisuuksien näkemisessä ja ymmärtämisessä myös liiketoiminnallisessa mielessä. Nykyisellä tekniikalla on mahdollista toteuttaa monipuolisia ja kattavia toiminnallisuuksia järjestelmiin. Tarkoituksenani on tuoda esille, kuinka organisaatiot voisivat hyödyntää ohjelmistoalan ratkaisuja saaden samalla teknologian kautta kilpailuetua. Hyöty voi syntyä mm. tehostamalla toimintaa tai tekemällä siitä joustavampaa. Opinnäyteyönä suunnittelemani sovellus on ensimmäinen E4SE-toiminnanohjausjärjestelmälle toteutettu alustariippumaton mobiilikirjaukset mahdollistava ratkaisu. Panostamalla jatkokehitykseen, uskon sovelluksen arvon paranevan ja voivan jopa tarjota tiettyä kilpailuetua. Tavoitteena on erityisesti luoda kehityskelpoinen ja perusominaisuuksiltaan toimiva sovellus. Pääpainona opinnäytetyössäni on projektityö ja tarvelähtöinen sovelluskehitys. Tarkoituksenani on toteuttaa ja esitellä ohjelmistotuotannon menetelmiin ja uusimpaan tietoon

perustuva ratkaisu. Sekä tuottaa käyttäjälähtöisiä sovelluksia liiketoiminnan määrittelemässä viitekehyksessä. Nykyaikaisten ja muuttuvien vaatimusten valossa sovellan toimintamallina mm. ketterän kehityksen periaatteita.

Aloitan opinnäytetyöni lyhyellä katsauksella toiminnanohjausjärjestelmiin pääluvussa kaksi. Kerron niiden toiminnasta, periaatteista ja teknologiasta. Päätän luvun arvioimalla järjestelmien yleisimpiä haasteita ja ongelmakohtia. Perehdyn mobiiliteknologian nykytilaan pääluvussa kolme. Tutkin teknologian ominaisuuksia ja sen tarjoamia mahdollisuuksia. Analysoin lopuksi mobiilialan lähitulevaisuuden trendejä ja yleisiä kehityssuuntia. Tarkastelen mobiiliohjelmointia pääluvussa neljä. Luku tarjoaa tietoa suunnitteluperiaatteista ja syventävän katsauksen Java ME -teknologiaan. Viides luku sisältää katsauksen ohjelmistotyöhön projektinäkökulmasta. Syvennyn myös ohjelmistoratkaisujen toteuttamiseen järjestelmällisenä prosessina Kerron tarkemmin toimintamallien teoriasta luvussa 5.4 ja projektityössäni käytetyistä periaatteista ja projektisuunnitelmasta luvussa 6.5. Esimerkkitapauksena suunnittelen Smart Time Oy:lle mobiilisovelluksen, jonka tarkoitus on parantaa Epicor For Service Enterprises -toiminnanohjausjärjestelmän projektikohtaisia kirjauksia liittyen tunti- ja kulukirjauksiin. Lopuksi analysoin, kuinka hyvin suunniteltu ohjelmisto ratkaisi ongelman, kuinka projekti edistyi ja arvioin sovelluksen jatkokehityskelpoisuutta. Päätän työni pohtimalla mobiiliratkaisujen merkitystä ja tulevaisuutta sekä tietojärjestelmien näkymiä.

2 ERP-JÄRJESTELMÄT

2.1 Mikä on ERP-järjestelmä

Toiminnanohjaus- eli ERP-järjestelmä on laaja-alainen tietojärjestelmä, jonka avulla pyritään hallinnoimaan ja ohjaamaan yrityksen toimintaa kokonaisvaltaisesti. Yleensä toiminnanohjausjärjestelmät kattavat kaikki tarpeelliset osa-alueet yrityksen toimintaa koskien. (Tieke 2008.) Varsinaisen ERP-toteutuksen voivat muodostaa yksi tai useampi modulaarinen järjestelmä. Yleensä myös yhden toimittajan järjestelmät koostuvat monista tiukemmin tai väljemmin integroiduista osajärjestelmistä. (Davenport 2000, 300.)

ERP-järjestelmän toiminnallisuudet riippuvat siihen integroiduista osajärjestelmistä eli moduuleista. Moduuleja voivat olla mm. projektien hallinta, palkanlaskenta, resurssien hallinta, kirjanpito, reskontra ja varastonhallinta. Järjestelmän moduulit voivat olla itsenäisesti kytkettävissä tai vaatia toisia osajärjestelmiä toimiakseen. (Enterprise Resource Planning, 2010). Monet ERP-järjestelmät tarjoavat samankaltaisia toiminnallisuuksia moduuleina. Esimerkiksi maailman suosituin järjestelmä, SAP ERP, tarjoaa myynti- ja jakelumoduulin, materiaalinhallinta-, tuotannonsuunnittelu-, laadunhallinta-, tuotantolaitoksenhallinta-, etuuksienhallinta-, henkilöstönhallinta- ja projektinhallintamoduulin (Monk & Wagner, 2009, 27-28).

Toiminnanohjausjärjestelmä auttaa integroimaan yrityksen prosessit ja tiedot yhdeksi kokonaisuudeksi. Tietojen yhtenäistämisen ja tehokkaan hallinnan takana on aina keskitetty tietovarasto, useimmiten relaatiotietokanta (Davenport 2000, 302). Keskitetyn varastoinnin ja hallinnan lisäksi data on ajantasaista ja sen saanti välitöntä (Monk & Wagner 2009, 17). Järjestelmien hyödyt tulevat ilmi myös liiketoimintaprosessien tehostumisena, selkeytymisenä ja automatisoitumisena. Kun kaikki organisaation tieto on integroitu eri osastojen välillä liiketoiminnan seuraaminen, päätöksenteko, inventoinnit, tuotantoprosessit ja henkilöstönhallinta nopeutuvat ja noudattavat yhtenäistä standardin mukaista menettelyä. On kuitenkin huomioitava etteivät ERP-järjestelmät automaattisesti ratkaise kaikkia ongelmia, vaan vaativat usein räätälöintiä ja organisaatiokohtaista käyttöönottoprojektia. (Wailgum 2008.) Tietoyhteiskunnan kehittämiskeskuksen Tieken mukaan automaation avulla on mahdollista vähentää henkilötyötunteja ja siten säästää kustannuksissa ja työajassa. Esimerkiksi tilaustenhallinta, varastonhallinta, laskutus, reskontra ja kirjanpito voivat toimia reaaliaikaisesti ja välittää keskenään tietoa ilman, että työntekijöiden tarvitsee puuttua toimintaan. (Tieke 2008.) Kuitenkin perimmäinen ja tärkein syy toiminnanohjausjärjestelmien käyttöön on raha; joko säästöt kustannuksissa tai lisävoiton tuottaminen kasvavan tehokkuuden myötä (Monk & Wagner 2009, 34; Tieke 2008).

ERP-järjestelmien perusta syntyi alun perin 60- ja 70-lukujen aikana kehitetyistä MRP-järjestelmistä. Voidaankin sanoa, että organisaatioiden toiminnan tietotekninen hallinta ja integraatio saivat alkunsa teollisuuden tarpeista. (Monk & Wagner 2009, 20-21.) Toiminnanohjausjärjestelmät kehittyivät tekniikan ja vaatimusten kasvaessa, mutta niiden historiassa voidaan puhua kahdesta merkittävää kasvukaudesta: 1980-luvusta ja vuoden 2000 tietotekniikkakriisistä. 1980-luvun taloudelliset vaikeudet loi-

vat uutta uskoa ERP-järjestelmien potentiaaliin ongelmien ratkaisijana ja liiketoiminnan tehostamiskeinona. Vuosituhannenvaihteen ns. Y2K-kriisi puolestaan tarjosi lähes pakon saneleman vaihtoehdon siirtymiseen vakaampiin ja monipuolisempiin tietojärjestelmiin. (Monk & Wagner 2009, 23, 25.)

Tunnettuja toiminnanohjausjärjestelmiä tarjoavia ohjelmistotoimittajia ovat mm. SAP, Microsoft, The Sage Group, Oracle, Infor Global Solutions ja Epicor. Lukuun ottamatta Epicoria, mainitut yritykset olivat Gartnerin tutkimuksen mukaan maailman johtavat ERP-toimittajat vuonna 2005 (Bailor 2006). Epicor on kuitenkin maininnan arvoinen, koska sillä on vahva asema Pohjoismaiden ja Venäjän markkinoilla yritysostojen kautta hankitun, alkujaan Ruotsalaisen, iScala ERP-järjestelmän ansiosta. Vaikka erilaisia toiminnanohjausjärjestelmiä ja niiden toimittajia on markkinoilla paljon, ne ovat useimmiten erikoistuneet ja painottaneet toimintojaan hieman eri tavalla. Osa järjestelmistä on räätälöity tietyn teollisuudenalan tai liiketoiminnan tarpeisiin. Esimerkiksi tässä opinnäytetyössä käsittelemäni Epicorin tuottama E4SE on tarkoitettu palvelualan yritysten tarpeisiin. Palveluyritykseksi tässä kontekstissa voidaan laskea laaja-alaisesti mikä tahansa organisaatio, jonka liiketoiminta muodostuu pääsääntöisesti myydyistä työtunneista tai tuotetuista palveluista.

2.2 ERP-järjestelmien teknologia, laajennettavuus ja käyttöönotto

ERP-järjestelmät ovat käytännössä laajoja, monimutkaisia ja kalliita tietojärjestelmiä. Ne ovat teknisesti haasteellisia johtuen mm. niiden huomattavan pitkästä kehityspolusta, jatkuvasta tuotekehityksestä ja sisäisistä riippuvuuksista eri osien välillä. Monien järjestelmien juuret voivat olla eri ohjelmistoalustalta ja usein eri vuosikymmeneltä. Järjestelmää on myös todennäköisesti paranneltu ja laajennettu vuosien kuluessa. Sen jokaisen osajärjestelmän, moduulin ja ohjelmistokomponentin on silti toimittava luotettavasti ja saumattomasti yhteen. Edellä mainittujen seikkojen valossa ei ole ihme, että ne ovat rakenteeltaan ja teknologiaratkaistuiltaan kompleksisia.

Usein järjestelmien asiantuntijat hallitsevat eri osia ja tasoja ohjelmistosta eikä yksittäinen henkilö pysty hallitsemaan kaikkea riittävällä tarkkuudella (Monk & Wagner 2009, 33). Selkein raja kulkee businesskäyttäjien ja teknisten asiantuntijoiden välillä. Opinnäytetyöni kannalta olennaista on tutustua nimenomaan järjestelmien tekniseen toimintaan. Ohjelmiston tuottaneen organisaation ulkopuolisilla kehittäjillä ei ole

useinkaan mahdollisuutta muokata järjestelmää suoraan järjestelmätasolla esimerkiksi lähdekoodin tai takaisinmallinnuksen avulla. Lähes kaikki järjestelmät kuitenkin tarjoavat joukon erilaisia API-rajapintoja tai muita työkaluja laajennuksien ja räätälöintien tekoon. Esimerkiksi SAP ERP mahdollistaa ohjelmistokomponenttien ja toimintojen kirjoittamisen järjestelmän sisäisellä ohjelmointikielellä (Monk & Wagner 2009, 35). Monesti valmistajat myös haluavat helpottaa käyttöönottoprojektia sekä räätälöintityötä tarjoamalla järjestelmän mukana valmiita konfiguraatioita ja monipuoliset konfigurointityökalut, joiden avulla suurin osa asiakasräätälöinneistä saadaan suoritettua säätötoimenpiteillä. Lisäksi järjestelmiä voidaan laajentaa kolmannen osapuolen sovelluksilla mm. raportointi- ja arkistointiratkaisuilla. (Enterprise Resource Planning 2010.)

Erilaisten teknologisten ratkaisujen kirjo on ERP-järjestelmien keskuudessa laaja, kun ottaa huomioon järjestelmien erilaiset lähtökohdat, historian, sovellusalustat ja monet muut kehitykseen vaikuttaneet seikat. Koska kaupalliset toiminnanohjausjärjestelmät ovat suureta osin suljetun lähdekoodin ohjelmistoja, ei niiden sisäisestä toiminnasta, tarkasta arkkitehtuurista tai ohjelmointiratkaisuista ole yleisesti saatavilla olevaa tietoa. On myös olemassa joukko *open source* -järjestelmiä, mutta ne eivät ole saavuttaneet vastaavaa suosiota. Käytännössä kaikki ERP-järjestelmät noudattavat palvelin-asiakas-arkkitehtuuria lukuun ottamatta aivan vanhimpia keskustietokoneilla toimivia järjestelmiä (Davenport 2000, 299-301). Nykyään järjestelmät ovat kuitenkin kehittymässä SOA-arkkitehtuurin ja selainkäyttöliittymän suuntaan (Monk & Wagner 2009, 41-43). Muutos ei kuitenkaan vaikuta perusarkkitehtuuriin, vaan enemmänkin tapaan kuinka järjestelmä toimii sisäisesti. Tässä työssä käsittelemäni E4SE on täysin selainpohjainen järjestelmä ja uusia Epicor ERP-järjestelmiä markkinoidaan *True SOA* -termillä.

SOA-arkkitehtuuri perustuu mm. w*eb service* -tekniikan hyödyntämiseen. Arkkitehtuurin mukaisesti yksittäiset toiminnallisuudet tai toimintokokonaisuudet on toteutettu verkkopalveluna esimerkiksi lähiverkon tai Internet-verkon päälle. Verkkopalvelu puolestaan tarjoaa joukon määriteltyjä metodeja, joita eri moduulit voivat tarvittaessa kutsua. (Service-oriented architecture 2010.) SOA-arkkitehtuuria noudattavat järjestelmäratkaisut on myös mahdollista tarjota pilvipalveluna tai SaaS-mallin mukaisena palveluna helpommin kuin perinteiset asiakas-palvelin-malliin pohjautuvat tietojärjestelmät. On myös havaittu, että SOA-pohjaisen järjestelmän päälle on nopeampi toteut-

taa uusia toiminnallisuuksia, kuin perinteisten järjestelmien. (Monk & Wagner 2009, 42.) SaaS-malli tarjoaa myös nopeampia käyttöönottoprojekteja, parempaa päivityksien hallintaa sekä halvempia kustannuksia. Uudenlaiseen palvelufilosofiaan liittyy luonnollisesti paljon ennakkoluuloja mm. tietoturvaan, luotettavuuteen ja tiedon saatavuuteen liittyen. (Wailgum 2008.) SOA-ratkaisu kuitenkin parantaa järjestelmän dynaamisuutta, komponenttien uudelleenkäytettävyyttä ja eri osien itsenäisyyttä, mutta voi lisätä uusia toiminnallisuus- ja abstraktiokerroksia ja siten heikentää suorituskykyä (Service-oriented architecture 2010). Voidaankin sanoa, että ratkaisumalli on eräänlainen kompromissi joustavuuden ja suorituskyvyn välillä. Nykyisillä palvelintietokoneiden tehoilla ja skaalautuvilla ohjelmistoratkaisuilla suorituskyky tuskin muodostuu ongelmaksi.

Toiminnanohjausjärjestelmän käyttöönotto on kallis ja pitkä prosessi. Käyttöönoton vaatimaan aikaan ja kustannuksiin vaikuttavat pitkälti organisaation olemassa olevien prosessien laajuus ja monimutkaisuus, käyttäjien koulutus, organisaation koko, räätälöintien määrä, integrointien määrä ja laatu, olemassa olevan datan konversio, ulkopuolisen konsultoinnin tarve ja monet muut seikat (Monk & Wagner 2009, 34; Tieke 2008; Wailgum 2008). Tyypillinen käyttöönottoprojektin kesto vaihtelee kuukausista jopa vuosiin. Projektin kestoa ja kustannuksia on kuitenkin mahdollista pienentää hyödyntämällä SaaS-palveluiden mahdollisuuksia. Wailgum myös painottaa artikkelissaan järjestelmän käyttötavan ja sen käyttöönoton syiden ymmärtämistä. (Wailgum 2008.)

Usein on järkevää sovittaa organisaation liiketoimintaprosessit järjestelmän tarjoamiin toimintamalleihin. Vaikka yrityksessä olisi totuttu suorittamaan prosessit tietyllä tavalla, on kustannuksiltaan halvempaa ja järjestelmän ylläpidon kannalta järkevämpää mukauttaa ne ohjelmiston vaatimusten mukaisiksi. Lopputuloksena järjestelmän päivitys, laajennus ja käyttöönotto on mahdollista suorittaa kustannustehokkaammin ja nopeammin. Laajat räätälöintiprojektit ovat yleensä kalliita ja vaikeasti ylläpidettäviä. (Tieke 2008; Wailgum 2008.)

2.3 Järjestelmien haasteet

Kun organisaatiot alkoivat kehittää ja ohjata toimintaansa tietojärjestelmien avulla kertyi helposti laaja joukko erilaisia tarkasti rajattuihin prosesseihin suunniteltuja jär-

jestelmiä. Vaikka yksittäinen järjestelmä toimi hyvin omalla osa-alueellaan, tieto oli eristäytynyttä ja vaikeasti hyödynnettävissä muissa järjestelmissä, liiketoiminnan seurannassa ja päätöksenteossa. Syitä haasteisiin olivat mm. erilaiset rajapinnat, tiedostoformaatit, standardit ja käytännöt. Seurauksena hajanaisista järjestelmistä olivat korkeat kustannukset, tiedonsiirron hitaus ja virhealttius. (Monk & Wagner 2009, 18-19.)

Pro gradu -tutkielmassaan Mäkipää (2002, 16-17) jakaa ERP-järjestelmiin kohdistuvan kritiikin viiteen tekijään: joustamattomuuteen, hitaaseen käyttöönottoon, ylihierarkkiseen organisaatiokäsitykseen, vanhentuneeseen teknologiaan ja keskinkertaiseen toiminnallisuuteen. Bramorski puolestaan esittää toiminnanohjausjärjestelmien käyttöönoton haasteiksi kalliit investoinnit, vaivalloisen ja pitkäkestoisen toteutuksen ja testauksen sekä korkeat asiantuntemusvaatimukset. Niiden seurauksena käyttöönottoprojektit yleensä moninkertaistuvat aika- ja kustannusarvioihin verrattuna. Lisäksi lopullisia tavoitteita ja strategisia etuja on vaikea arvioida ja ne voivat jopa jäädä saavuttamatta. (Bramorski 2004, 9-10.)

Yleisellä tasolla ERP-järjestelmien käytössä ja käyttöönotossa esiintyy monenlaisia ongelmia, joiden taustalla voi olla monia tekijöitä. Järjestelmien räätälöinti on kallista ja työlästä. Aiemmissa luvuissa esittämäni tiedon valossa on kyseenalaista kannattaako järjestelmää lähteä alun perinkään räätälöimään suuressa mittakaavassa. Käytännössä päätös on organisaatiokohtainen, mutta riskit kannattaa arvioida huolella. Suurin osa epäonnistuneista ERP-toteutuksista voidaan johtaa liialliseen järjestelmän muuttamiseen (Wailgum 2008). Toisaalta olemassa olevien liiketoimintaprosessien mukauttaminen järjestelmän standardiin voi johtaa pahimmassa tapauksessa kilpailuedun menettämiseen tai heikentymiseen (Enterprise Resource Planning 2010). Kuten aikaisemmin tuli ilmi järjestelmien toiminnassa ja hierarkkisuudessa esiintyy tietyn asteista jäykkyyttä ja standardimaisuutta. Ajan kuluessa ja järjestelmän jatkuvan käytön myötä lisääntyvä huolimattomuus voi myös kerryttää ns. likaista dataa järjestelmään. Epätarkka, väärin syötetty tai väärässä muodossa oleva data ei palvele alkuperäistä tarkoitustaan. Tämä on mahdollista nähdä seurauksena liian jäykästä tai vaikeasta käytettävyydestä tai käyttäjien liian vähäisestä koulutuksesta ja tietämyksestä. (Enterprise Resource Planning 2010.)

Kiteytettynä ERP-järjestelmien haasteet voisi jakaa karkeasti kolmeen ryhmään. Käyttöönottoprojektin toteutus kaikkine osa-alueineen ja lähtökohtineen on varmasti mer-

kittävin haaste. Ilman onnistunutta käyttöönottoprojektia ei järjestelmällä ole mahdollisuuksia täyttää sille asetettuja vaatimuksia. Käyttäjien ja muiden avainryhmien koulutus sekä toteutukseen osallistuvien asiantuntijoiden tietämys järjestelmästä ja liiketoimintaprosesseista on toinen haaste. Hyväkään järjestelmä ei voi olla tehokas ellei sitä käytetä oikein ja elleivät liiketoimintaprosessit toimi saumattomasti yhteen sen kanssa. Kolmas haaste on järjestelmän teknologia. Se kattaa mm. räätälöinnit, tekniset asetukset, tekniset ratkaisut, käytettävyyden, rajapinnat ja integraatiot.

Pyrin opinnäytetyöni käytännön osuudessa parantamaan E4SE-järjestelmän käytettävyyttä liittyen henkilöiden suorittamiin tunti- ja kulukirjauksiin. Se on pieni osa-alue massiivisen järjestelmän toiminnoissa. Vaikka yksittäisen toiminnon helppokäyttöisyys ja joustavuus ei muuta koko järjestelmän käyttökokemusta, sen avulla voidaan vähentää aiemmin mainitun likaisen datan kertymistä. Osaltaan se pienentää järjestelmän epäonnistumisen riskejä ja ratkaisee tietojärjestelmiä yleisesti piinaavan käytettävyysongelman yhdellä osa-alueella.

3 MOBIILITEKNOLOGIA

3.1 Mobiililaitteet

Mobiililaite on teknisenä terminä epämääräinen, vaikka yleiskielessä sen merkitys on jo vakiintunut. Suurelle osalle termi luo mielikuvan esimerkiksi kännykästä. Yleistetysti voidaan puhua laitteesta, joka on riittävän pieni ja kevyt kannettavaksi mukana ja sen käyttö on mahdollista paikasta ja ympäristöstä riippumatta. Mobiililaite sisältää myös jonkin tietojärjestelmän. (Mikkonen 2004, 1.) Laitteita on määritelmästä johtuen hyvin paljon erilaisia ja ne käsittävät joukon erilaisia ominaisuuksia, rajoitteita ja toimintoja. Esimerkiksi määritteen toteuttavia laitteita ovat matkapuhelimet, PDA-laitteet, GPS-paikantimet, kannettavat tietokoneet, MP3-soittimet, kämmentietokoneet, elektroniset lukulaitteet tai vaikkapa moderni digitaalinen rannekello. Edellä mainituista esimerkeistä voidaan todeta, että laitteiden käyttötarkoitus voi vaihdella tarkasti rajatusta hyvinkin laaja-alaiseen. Tästä johtuen myös laitteiden toimintaperiaate, resurssit ja ohjelmistot vaihtelevat suuresti.

Yhteistä kaikille mobiililaitteille on niiden pieni koko, vähäinen energiankulutus sekä rajatut laiteresurssit mm. muistin määrä ja prosessorin teho. Mobiililaitteiden fyysinen käyttöliittymä on useimmiten rajattu joko muutamaan näppäimeen, kännykän tapauksessa numeronäppäimistöön lisänäppäimineen tai mahdollisesti minikokoiseen QWERTY-näppäimistöön. Näyttöpääte on myös useimmiten pieni ja tarjoaa suppeamman värimäärän tai resoluution kuin täysikokoinen tietokoneen näyttö. Erilaiset lisälaiteliitännät voivat puuttua kokonaan tai rajoittua yhteen tai useampaan valmistajakohtaiseen tai yleiseen standardiin. Mitään yleistä käytäntöä ei ole syntynyt, vaikka trendi on kohti yhteisiä standardeja ja yhteensopivuutta. Muita yleisiä ongelmia mobiililaitteilla tyypistä ja tarkoituksesta riippumatta ovat mm. kaistanleveys ja viiveet tiedonsiirrossa, tietoturvallisuus, virrankulutus, ulkoiset häiriöt, terveysriskit ja käyttöliittymän soveltuvuus. (Mobile computing 2010; Mobile Phone 2010.)

Tämän opinnäytetyön viitekehyksessä rajaan termin käsittämään vain älypuhelimet ja niihin teknisesti rinnastettavissa olevat MID-laitteet. Erityisen painotuksen saavat nykyaikaiset älypuhelimet ja kämmentietokoneet. Uusimpien ja yhä kehittyneempien laitteiden myötä ero näiden kahden laitetyypin välillä on hämärtynyt ja ei ole mielekästä jakaa mobiililaitteita perinteisesti totuttuihin ryhmiin. Mielestäni parempi jako voisi perustua laitteen käyttötarkoitukseen eikä tyyppiin tai markkinointitermeihin.

Älypuhelinta voidaan kuvata pienoiskokoiseksi tietokoneeksi, jossa on myös puhelinominaisuudet. Termi ei ole tieteellisen tarkasti määritelty ja sen voidaan ymmärtää käsittävän joukon laitteita, jotka sisältävät tiettyjä ominaisuuksia mm. tietoliikennerajapintoja, business- tai tuottavuussovelluksia, laiteliitäntöjä tai jopa täyden käyttöjärjestelmän ohjelmistorajapintoineen. (Smartphone 2010.) Yhä suurempi osa myydyistä uusista puhelimista on älypuhelimia

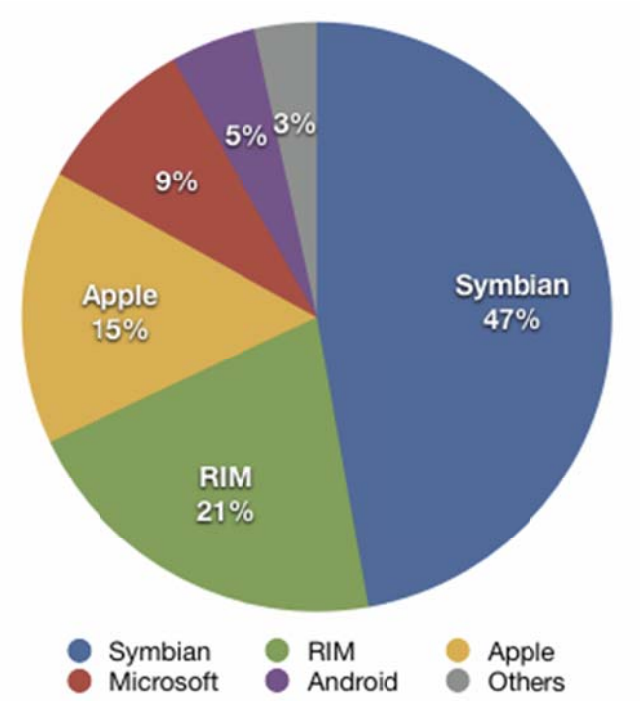
Älypuhelimet ovat teknisessä ja toiminnallisessa mielessä sulautettujen järjestelmien ja työasemien kompromissi. Niistä pyritään tekemään monipuolisia ja laajennettavissa olevia kuten työasemista, mutta samalla niiden tekniset ominaisuudet ja teknologiaratkaisut ovat lähempänä sulautettuja järjestelmiä. Tästä seuraa väistämättä ongelmia liittyen mm. suorituskykyyn, energiankulutukseen, käytettävyyteen, laajennettavuuteen ja ohjelmistokehitykseen. (Mikkonen 2004, 5-9.)

Oman lukunsa laitteiden monimuotoisuuteen tuovat laitevalmistajat, jotka voivat olla samanaikaisesti myös laitteiston pääasiallinen tai yksinoikeutettu ohjelmistovalmistaja. Eri valmistajilla on erilaisia näkemyksiä laitteiden käyttötarkoituksesta ja toteutuksesta. Esimerkiksi Apple iPhone on suljettu ja tarkasti kontrolloitu alusta, joka on tiukasti Applen hallinnassa niin laitteiston kuin ohjelmistojenkin suhteen. Vuoden 2009 lopulla julkaistu Nokia N900 on puolestaan laajalti käyttäjien muokattavissa ja laajennettavissa, koska sen käyttöjärjestelmä perustuu suurelta osin avoimen lähdekoodin Maemo ympäristöön ja sen ohjelmistorajapinnat ovat yleisesti saatavilla (Maemo 2010).

3.2 Mobiilikäyttöjärjestelmät ja -ohjelmistoalustat

Mobiililaitteiden ohjelmistot toteutetaan yleensä joko suoraan tietylle laitteistolle rajapintatasolla tai ohjelmistoalustan avulla. Matalan tason ohjelmointi vaatii usein suoraa tietämystä joko laitteen toiminnasta, käyttöjärjestelmäarkkitehtuurista tai muista teknisistä yksityiskohdista. Vaikka tuotettu ohjelmakoodi on yleensä tehokkaampaa ja toimii kyseisessä laitteessa nopeammin kuin korkean tason menetelmillä toteutetut ohjelmat, se ei ole yleisesti käytetty menetelmä tuottaa yleiseen jakeluun tarkoitettuja sovelluksia. Pääsyinä ovat laitteistoriippuvuus, puutteet dokumentaatiossa, työkaluissa tai rajapinnoissa, taloudelliset syyt tai lisensointiehdot. (Mikkonen 2004, 17.)

Ohjelmistoalustat on kehitetty ratkaisemaan edellä mainittuja ongelmia. Ne voivat olla alustariippumattomia tai sidottuja tiettyyn käyttöjärjestelmään. Seuraavasta kuvasta (kuva 1) selviää vuoden 2009 markkinatilanne eri käyttöjärjestelmien ja ohjelmistotoimittajien suhteen. Sen perusteella voidaan tehdä johtopäätöksien ohjelmistoalustojen ja käyttöjärjestelmien yleisyydestä.



KUVA 1. älypuhelinkäyttöjärjestelmien markkinaosuudet vuonna 2009 (Smartphone 2010)

Esittelen seuraavaksi lyhyesti yleisimpiä älypuhelinten käyttöjärjestelmiä ja sovellusalustoja. Jos tieto on saatavilla, selvitän tukeeko käyttöjärjestelmä tai sovellusalusta Java ME -sovelluksia. Koska opinnäytetyöni projektityön tavoite on toteuttaa mahdollisimman laiteriippumaton Java-sovellus, on mielenkiintoista nähdä, kuinka suuri osa nykyisistä laitteista ja käyttöjärjestelmistä tukee sitä.

Symbian

Symbian OS on Symbian Ltd. kehittämä ohjelmistoalusta ja käyttöjärjestelmä. Vuonna 2008 Symbian OS siirtyi mm. Nokian yritysostojen kautta Symbian Foundation organisaation haltuun. Sen päätarkoituksena on kehittää Symbian OS alustaa voittoa tuottamattomana tahona ja avoimen lähdekoodin ideologialla. Symbianin kehitykseen on sitoutunut laaja joukko maailman suurimpia laite- ja piirivalmistajia, ohjelmisto- ja rahoitusalan yrityksiä ja matkapuhelinoperaattoreita. Symbian on markkinajohtaja älypuhelinten käyttöjärjestelmämarkkinoilla. (Symbian OS 2010.) Symbian^1 sekä entinen Symbian OS voidaan määritellä 32-bittisenä moniajokäyttöjärjestelmänä sekä siihen liitettyinä käyttöliittymä- ja muina kirjastoina (Mikkonen 2004, 12).

Symbian tukee sekä C++ että Java ME -sovelluksia. Tosin sen käyttämä C++-ympäristö sisältää laajennuksia tukeakseen mobiililaitteiden erityispiirteitä. Osa perinteisesti kääntäjän ja ajon aikaisen ympäristön huolehtimista toiminnoista mm. roskien keruu ja muistinhallinta jää ohjelmoijan toteutettavaksi. (Mikkonen 2004, 12.) Aiemmin Symbian OS päälle oli julkaistu useita eri alustoja mm. S60 ja UIQ. Lisäksi useille laitteille oli olemassa erityisesti niille suunnattuja SDK-kehitystyökaluja. Uusi Symbian ympäristö pyrkii selkeyttämään tätä yhdistämällä erilliset alustat yhdeksi. (Symbian OS 2010.) Sovellusten kehittäminen Symbianille on haasteellista ohjelmointikielen vaikeuden ja alustan tuomien haasteiden vuoksi. Java ME -pohjaisten sovellusten kehittäminen Symbianille on huomattavasti nopeampaa ja helpompaa, koska Java on alustariippumaton ja pohjautuu korkean tason abstraktioihin. Symbian käyttöjärjestelmä sisältää nimenomaisesti sille käännetyn Java virtuaalikoneen, jossa Java-sovellukset suoritetaan.

BlackBerry OS

Erityisesti Pohjois-Amerikan markkinoilla menestynyt RIM on julkaissut omalle laitealustalleen suunnitellun BlackBerry OS käyttöjärjestelmän. Sen suunnitteluperiaatteet painottuvat helppokäyttöisyyteen ja liiketoimintalähtöisyyteen. BlackBerry OS on suljetun lähdekoodin käyttöjärjestelmä, joka tarjoaa ohjelmistokehittäjille joukon API-rajapintoja. Käyttöjärjestelmä tukee myös Java MIDP 1.0 ja 2.0 mukaisia sovelluksia. (BlackBerry OS 2010.)

Apple iPhone OS

Apple on toteuttanut iPhone älypuhelimilleen iPhone OS nimellä kulkevan OS X:stä periytyvän käyttöjärjestelmän. Se on myös muiden Applen kannettavien laitteiden esim. iPod Touch ja iPad oletuskäyttöjärjestelmä. Kuten OS X myös iPhone OS pohjautuu Unix käyttöjärjestelmään. iPhone OS on sallinut ohjelmistokehittäjien toteuttaa omia ohjelmistojaan versiosta 2.0 eteenpäin, mutta niitä on saanut julkaista ja asentaa ehtojen mukaan vain Applen App Storen kautta. Ohjelmistojen kehittäminen tapahtuu erityisen SDK:n kautta ja ohjelmia voidaan kirjoittaa Objective-C kielellä. Java ohjelmistoalusta ei ole tuettu eikä Apple ole julkaissut tietoa tuen lisäämisestä tulevaisuudessakaan. Nykyinen lisensointi kieltää Java-virtuaalikoneen toimintaperiaatteen mukaisen ohjelmiston toteutuksen. (iPhone OS 2010.)

Microsoft Windows Mobile

Microsoft on julkaissut Windows Phone -yhteensopiville älypuhelimille ja PDA-laitteille Windows CE -pohjaisen kompaktin mobiilikäyttöjärjestelmän nimeltä Windows Mobile. Windows CE on erityisesti sulautettuihin järjestelmiin tarkoitettu kevytkäyttöjärjestelmä, jolla ei ole työpöytäversioonsa juuri muuta yhtymäkohtaa kuin tuotenimi. Windows Mobile on suosittu etenkin Pohjois-Amerikan markkinoilla, jossa se on kolmanneksi myydyin businesspuhelimien mukana tuleva käyttöjärjestelmä. (Mikkonen 2004, 13-14; Windows Mobile 2010.)

Vaikka useat asiantuntijat ovat ennustaneet Windows Mobile käyttöjärjestelmän jäävän kilpailijoidensa jalkoihin tai poistuvan vähitellen markkinoilta (Crothers 2009; Gold 2009; Hamblen 2009; Perlow 2009), Microsoft itse on sanonut jatkavansa tuotteen kehittämistä ja pyrkivänsä parantamaan asemiaan. (Foley 2010.) Joka tapauksessa, Windows Mobile tarjoaa ohjelmistokehittäjille hyvät työkalut ja monipuolisen sovellusalustan. Se tukee niin C++ ohjelmointikieltä kuin .NET Compact Framework sovellusalustaa. Juuri .NET alustan ansiosta monet olemassa olevat kyseisellä tekniikalla toteutetut Windows-työpöytäsovellukset on helppo siirtää mobiiliympäristöön pienin muutoksin. Tällä hetkellä .NET Compact Framework ei toimi muilla älypuhelimilla kuin Microsoft Phone yhteensopivissa laitteissa ja siinä ei ole virallista Java tukea. Java-virtuaalikoneen tai sitä vastaavan toiminnallisuuden saa lisättyä joihinkin laitemalleihin ja käyttöjärjestelmäversioihin kolmannen osapuolen ohjelmistoilla (ks. WebShpere Everyplace Micro Environment 2010; The Eve Virtual Machine and SDK 2010).

Android

Android on alun perin Android Incin kehittämä, myöhemmin Googlen kautta Open Handset Alliancen omistukseen päätynyt, Linux-pohjainen mobiilikäyttöjärjestelmä. Sen nykyisiin kehittäjiin kuuluvat mm. Google, Intel, HTC, Motorola ja joukko muita nimekkäitä laite- ja ohjelmistovalmistajia. Android perustuu avoimeen lähdekoodiin. (Android 2010.)

Android käyttää omaa Googlen kehittämää JVM implementaatiota, joka on yhtiön mukaan nopeampi ja avoimempi kuin alkuperäinen Sun Microsystemsin kehittämä versio. Ongelmaksi kuitenkin muodostuu Android JVM:n, Dalvikin,epästandardi toteutus sekä laite- ja järjestelmäriippumattoman Javan ideologian katoaminen. (Shankland 2007.) Käyttöjärjestelmä ei tue aidosti alkuperäistä Java-standardia, mutta epäsuora tuki löytyy perinteisen Java ME:n lisäksi myös suurelle osalle standardin Java SE:n kirjastoista (Burnette 2008). J2ME:llä toteutetut sovellukset on käännettävä erityisen MIDP-kääntäjän avulla käyttöjärjestelmän ymmärtämiksi paketeiksi. Sovelluksia Android-alustalle on mahdollista kehittää myös käyttämällä sen omaa SDK-työkalupakettia mm. C-ohjelmointikielellä. (Android 2010.)

Embedded Linux

Embedded Linux eli sulautettu Linux ei ole suoranaisesti oma mobiilikäyttöjärjestelmä, vaan enemmänkin joukko Linux-ytimeen pohjautuvia ohjelmistoratkaisuja. Useat valmistajat ovat julkaisseet tai kehittämässä omaa toteutustaan. Tunnetuimpia Linux-pohjaisia käyttöjärjestelmäratkaisuja ovat mm. Maemo, Openmoko, LiMo, Palm webOS ja Qt Extended. Järjestelmät ovat suosituimpia Kiinassa ja Japanissa, jossa mm. Motorola ja DoCoMo tarjoavat niillä varustettuja laitteita. (Mobile operating systems 2010.) Mobiilin Linuxin vahvuudet vastaavat työasemamarkkinoiden kilpailuetua. Järjestelmät ovat vapaasti muunneltavia, perustuvat avoimeen lähdekoodiin ja niiden käyttö ei sido laitevalmistajia tiettyyn ohjelmistotoimittajaan tai toteutukseen. (Mikkonen 2004, 16.)

Kenties merkittävin Linux-pohjainen mobiilikäyttöjärjestelmä tulevaisuutta ajatellen on Nokian ja Intelin yhteistyöprojekti MeeGo, jossa yhdistyvät Nokian Maemo ja Intelin Moblin käyttöjärjestelmät. Maemon kehittämiseen on osallistunut monia tunnettuja avoimen lähdekoodin projekteja kuten Debian ja Gnome (Maemo 2010). Moblin puolestaan on avoimeen lähdekoodiin perustuva MID-laitteille kohdennettu käyttöjärjestelmä ja ohjelmistoalusta (Moblin 2010).

Riippuen käyttöjärjestelmävalmistajan toteutuksesta Java tuki voi olla saatavilla tai ei. Oletusarvoisesti Linux tukee Java virtuaalikonetta ja se voidaan asentaa erillisenä komponenttina järjestelmään, mutta valmistajat voivat estää käyttäjiä lisäämästä omia

sovelluksia tai olla lisäämättä Java tukea. Todennäköistä nykyisten trendien valossa on sisällyttää Java tuki vähintään Java ME -alustalle.

Palm webOS

Aiemmin Palm Inc. kauppasi Palm OS nimistä mobiilikäyttöjärjestelmää omille ja kumppaniensa PDA- ja älypuhelinlaitteille. Vuoden 2009 kesäkuussa Palm julkisti uuden Linux-pohjaisen webOS nimeä kantavan käyttöjärjestelmän ja ilmoitti lopettavansa vanhan nimikkeen myynnin tulevissa laitteissaan. (WebOS 2010.)

Koska webOS perustuu pohjimmiltaan Linuxiin ja Palmin aiempi käyttöjärjestelmä Palm OS sisälsi Java tuen, on loogista olettaa myös webOS:n sisältävän Java tuen. On myös olemassa artikkeleita, joista saa käsityksen käyttöjärjestelmän Java tuesta (ks. Beers 2009). Kirjoitushetkellä ei yksiselitteistä ja virallista vastausta kuitenkaan löytynyt ja jää nähtäväksi onko ominaisuus toteutettu vai ei. Tällä hetkellä Palm tarjoaa sovelluskehityksen tarpeisiin ainakin oman SDK paketin ja epätavallisen tavan toteuttaa web-sovelluksia ilman selainta (Young 2009).

Brew

Brew on lyhenne sanoista *Binary Runtime for Wireless*. Se on Qualcommin kehittämä ohjelmistoalusta mobiililaitteille (Mikkonen 2004, 15). Brew toteutettiin alun perin CDMA-laitteille, mutta myös nykyisen kaltaiset älypuhelimet ovat myös tuettuja (Brew 2010).

Ohjelmistokehittäjien kannalta tärkeää on, että Brew tarjoaa täydellisen paketin API-rajapintoja dokumentteineen. Sovelluksia on mahdollista toteuttaa monipuolisesti C ja C++ -ohjelmointikielillä. Java on myös käytettävissä, mikäli laite itse tukee sitä. (Brew 2010.)

Brew voidaan nähdä pseudokäyttöjärjestelmänä (Brew 2010), ohjelmistoalustana tai jopa liiketoimintamallina. Ajatuksen perustana on alustan toimintaperiaate, jonka mukaan mobiililaite toimii päätelaitteena sovellukselle siten, että työkuormaa voidaan siirtää tavoitteiden mukaan dataverkon yli palvelimille. (Mikkonen 2004, 15.)

3.3 Tulevaisuus ja trendit

Perustuen edellisen luvun analyysiin ja käyttöjärjestelmien esittelyyn voidaan päätellä muutamia alan yleisiä trendejä, jotka vahvistunevat tulevaisuudessa. Positiivisena trendinä nähdään siirtymä laitevalmistajakohtaisista käyttöjärjestelmä- ja sovellusalustaratkaisuista kohti yhteistyötä ja avoimia standardeja. Se mahdollistaa sovelluksien siirrettävyyden eri järjestelmien ja laitteiden välillä ja luo uudenlaisia kehityspolkuja olemassa oleville ohjelmistoille.

Erilaiset Linux-pohjaiset ratkaisut ovat myös nostamassa suosiotaan erityisesti laajan yhteensopivuuden, laajennettavuuden ja avoimuuden vuoksi. Merkittävä osa uusista mielenkiintoisista alustoista mm. MeeGo ja Android hyödyntävät olemassa olevaa Linux ydintä.

Lähes kaikki suuret mobiilisovellusalustoja kehittävät ja markkinoivat yritykset ovat viime vuosina lanseeranneet verkkopalvelun, jonka kautta sovelluksien myynti ja jakelu pystytään hoitamaan keskitetysti. Ilmiö on luonut uudenlaisia mahdollisuuksia sovelluskehittäjille ja ohjelmistoalan yrityksille. Vuoden 2010 Mobile World Congress -tapahtumassa kävi myös ilmi, että kaksikymmentäneljä suurta mobiilialan toimijaa ovat muodostamassa maailmanlaajuista alustariippumatonta sovelluskauppaa. Muita laitteiden suosioon vaikuttavia tekijöitä ovat kolmansien osapuolten sovelluksien ja työkalujen määrä. Se vaikuttaa merkittävästi kilpailevien ohjelmistoalustojen ja laitteiden ohjelmistotarjontaan ja siten väistämättä myös suosioon. Mm. Adobe on ilmaissut julkaisevansa AIR sovelluskehitystyökalut ainakin Android ja BlackBerry käyttöjärjestelmille. Kansainvälisen televiestintäliiton tutkimuksen mukaan matkapuhelinliittymien määrä kasvaa jopa viiteen miljardiin vuoden 2010 loppuun mennessä. Kasvu avaa aivan uudenlaisia mahdollisuuksia ja markkinoita mobiilisovelluskaupalle. (Mobile World Congress trends… 2010.)

Laitepuolella suunnannäyttäjänä toiminut Apple iPhone on saanut joukon vakavasti otettavia kilpailijoita. Monet yritykset yhdistävät älypuhelimiin aiemmin ns. netbookeista, kämmentietokoneista ja PDA-laitteista tuttuja ominaisuuksia. Älypuhelinmarkkinoiden ohella myös erilaisten minikannettavien ja kämmentietokoneiden markkinat ovat lähteneet kasvuun.

Markkinoille tulee uusia vaikuttajia ja innovaatioita niin laite kuin ohjelmistopuolellekin. Tämän hetkisen tiedon ja tuntuman valossa uskallan arvioida, että laite- ja ohjelmistoalusta, joka kerää parhaat sovellukset ja laajimman valikoiman selviytyy kisasta voittajana. Näkisin, että Java on erittäin vahvoilla pyrkimyksessään universaaliksi teknologiaksi mobiiliohjelmistoissa. Se on erittäin laitteisto- ja alustariippumaton, minkä lisäksi suurin osa nykyisistä valmistajista tukee jo sitä. Tulevaisuudessa yhä harvemmalla on varaa jättää Java huomioimatta.

Esittämiini päätelmiin ja teknologiakatsaukseen pohjautuen uskon, että Java on paras mahdollinen vaihtoehto projektityöni toteutukseen. Seuraavassa pääluvussa käsittelen mobiiliohjelmointia ja erityisesti mobiiliympäristöön soveltuvaa Java ME:tä.

4 MOBIILIOHJELMOINTI

4.1 Suunnitteluperiaatteita

Sovelluksen suunnittelu mobiiliympäristöön on usein haasteellisempaa kuin työpöytä- tai palvelinympäristöön. Edellisessä pääluvussa esitetyt erilaiset rajoitukset ja laitemallien väliset erot määrittelevät käytössä olevat tekniikat. Yleistettynä mobiilisovelluksien suunnittelussa on huomioitava erityisesti rajallinen suorituskyky ja riittävä tietoturva (Mikkonen 2004, 32-33). Suorituskyvyllä tarkoitetaan niin laitteen tarjoamaa muistia, prosessoritehoa kuin tietoliikenneyhteyksiäkin.

Mertasen (2004, 189-190) mukaan tietoliikenneyhteyttä valitessa on syytä miettiä kustannustehokasta vaihtoehtoa, sillä erilaisten yhteystyyppien erot voivat olla merkittävät. Tietoliikenneratkaisujen tulisi olla riittävän kevyitä, koska mobiililaitteiden tiedonsiirtokaista on rajallinen ja hinnoittelu voi perustua siirretyn datan määrään. Toisaalta tiedonsiirron tulisi olla salattua, yksityistä ja turvattua. Salaustekniikat voivat kuitenkin olla raskaita laitteen suorituskykyyn suhteutettuna. (Mikkonen 2004, 33.) Tietoliikenneyhteyksiä turvatessa on huomioitava, että käyttäjän tunnistaminen tehdään aina yhteyttä muodostettaessa. Varmennettu istunto ei myöskään saisi olla voimassa kovin pitkiä aikoja kerrallaan. Istuntojen voimassaoloaika ja tunnistuksen taso ovat kompromissi käyttömukavuuden ja turvallisuuden välillä. VPN-tunneli sopivalla salaustekniikalla on tehokas ja paljon käytetty tapa turvata tietoliikenneyhteydet. Eräs

yleisimmistä salaustavoista on SSL eli *Secure Socket Layer*. (Mertanen 2004, 207-208.)

Tietoturva on syytä huomioida myös kokonaisratkaisuna eikä pelkästään tietoliikenteen näkökulmasta. Laitteen varastamis- tai katoamistilanteessa asianmukainen salaus ja käyttäjäntunnistusmenetelmät voivat pelastaa arkaluontoisen datan tai estää pääsyn suljettuihin verkkoihin ja palveluihin. Suhteellisuuden taju kannattaa tosin pitää mukana. Yleensä sovellustason tunnistusta ei tarvita ja tiedostojakaan ei kannata salata, ellei se ole järkevää suhteessa datan arkaluonteisuuteen, kertoo F-Securen Matias Impivaara. (Mertanen 2004, 204-205.) Tunnistetiedot, salasanat ja muut arkaluontoiset asetukset voi mielestäni salata, sillä niiden sisältämä tietomäärä on yleensä erittäin suppea, joten salaus ei vaikuta merkittävästi suorituskykyyn. Impivaara painottaa myös, ettei tietoturva saisi haitata sovelluksen alkuperäistä tarkoitusta tai tehdä sen käytöstä vaivalloista (mts. 197).

Mobiilisovellus on yleensä jaettu useisiin abstraktiotasoihin (Mikkonen 2004, 39). Se helpottaa eri osien toteutusta ja on osin välttämätöntä, koska eri ohjelmistokomponentit ovat usein eri toimittajien käsialaa. Laitevalmistaja vastaa usein käyttöjärjestelmästä ja ohjelmistoalustasta. Sen päällä voi olla kolmannen osapuolen ohjelmistorajapintoja esimerkiksi Java ME konfiguraatio ja vasta päällimmäisenä ohjelmoijan itse tuottamat komponentit. Tyypilliset abstraktiotasot ovat moduuli-, sovellus-, alusta- ja järjestelmätaso (mts. 39). Riippuen tuotettavasta sovelluksesta ohjelmoija toteuttaa yhden tai useamman kerroksen päälle toimintoja. Tämän opinnäytetyön tapauksessa keskityn pääasiassa sovellustasoon, joka sijoittuu rakenteessa päällimmäiseksi. Sovellustasolla voidaan käsittää sellainen sovellus tai osajärjestelmä, joka on toteutettavissa irrallisena kokonaisuutena alemman tason järjestelmästä (mts. 43).

Mertanen (2004, 234) painottaa erityisesti käytettävyyttä suunnitellessa mobiilisovelluksia. Hyvin usein käyttäjälähtöisyyden puute aiheuttaa ongelmia mobiiliprojekteissa. Se on monesti seurausta kehitysprojektista, jossa asiakas ei osallistu päätöksentekoon tai kykene vaikuttamaan tehtyihin ratkaisuihin. Prototyyppien rakentaminen ja niiden testauttaminen tulevilla käyttäjillä on tehokas keino saavuttaa hyvä käyttöliittymä ja käyttökokemus (Mertanen 2004, 234). Kilpailun koventuessa mobiilimarkkinoilla trendi näyttäisi olevan ainakin kuluttajasovelluksissa kohti parempaa käyttöliittymää ja helppokäyttöisyyttä. Yritysratkaisuiden käyttöliittymät seuraavat kuluttaja-

markkinoiden perässä huomattavasti hitaammin. Hyvä käytettävyys voi siis olla selvä kilpailuetu.

Ratkaisun kehityksessä kannattaa hyödyntää hyväksi havaittuja suunnittelumalleja kuten mm. Mikkonen (2004, 51) suosittelee. Suunnittelumallit eivät ratkaise automaattisesti kaikkia ongelmia, mutta niiden oikeaoppinen käyttö helpottaa ja selkeyttää sovelluskehittäjän päätöksentekoa ja työtä. Klassisia suunnittelumalleja, joita voidaan hyödyntää mobiiliohjelmoinnissa, ovat Mikkosen mukaan mm. välimies, tarkkailija, vaihtuvat pakkaukset, muistirajaus, rajapintarajaus ja muistiallas (mt.) Erilaisia malleja on kymmenittäin ja niiden valikoinnissa on mietittävä lopputulosta ja tulevan sovelluksen ominaisuuksia. Väärin hyödynnettyjä tai vaikutukseltaan negatiivisia malleja on vastaavasti laaja joukko. Ne voivat olla haitallisuudestaan huolimatta yleisesti käytettyjä. Niitä kutsutaan nimellä *anti-pattern*.

Eräs yleinen lähestymistapa mobiiliohjelmointiin tietojärjestelmien kehittämisessä on ns. pilotointi. Pilottiprojektin tarkoitus on selvittää soveltuuko ratkaisu yrityksen käyttöön, millä tavalla sitä pitäisi kehittää ja miten sen käyttö todellisuudessa vaikuttaa. Kokeilujen avulla voidaan ratkaisumalleja parannella tai jopa hylätä minimoiden mahdollinen riski. Pilottikokeilun jälkeen päätetty projekti voi siten olla huomattavasti järkevämpi päätös kuin täysimittainen projekti, joka ei tuota haluttua lopputulosta. Epäonnistunutkin pilotti voi tarjota arvokasta tietoa tulevaisuuden projekteja ajatellen. (Mertanen 2004, 42-45.)

Kaikissa mobiiliprojekteissa kannattaa huomioida tarve, käytännön kokemukset ja asianmukainen suunnittelu. Kokonaisratkaisuihin liittyy useasti paljon muutakin, kuin pelkkä mobiilisovellus. (Mertanen 2004, 47.) Tyypillisesti mukana on erilaisia integraatioita yrityksen muihin tietojärjestelmiin, palvelinpään sovelluksia, tukitoimintoja, käyttöönottoprojekteja, jatkokehitystä ja muita ohjelmistoprojektien liitännäisiä. Mertanen kertoo, etteivät mobiiliprojektit poikkea merkittävästi normaaleista liiketoiminnankehitysprojekteista. Perus projektinhallintamenetelmillä, monipuolisella osaamisella ja asiakkaan aktiivisella osallistumisella saadaan jo hyviä tuloksia. (Mertanen 2004, 233-234.)

4.2 Java ME

Java ME on kehitetty resursseiltaan rajallisten laitteiden ohjelmistoalustaksi. Tyypillisiä laiteratkaisuja ovat sulautetut järjestelmät, älypuhelimet ja muut pienet tai kannettavat laitteet. Sen päätarkoitus on olla mahdollisimman kevyt ja tarjota riittävät toiminnallisuudet kohdelaitteille ollen samalla niin laitteistoriippumaton kuin mahdollista. Myös turvallisuusominaisuuksiin on kiinnitetty suhteellisesti enemmän huomiota kuin sen isoveljissä Java SE:ssä ja Java EE:ssä (Mikkonen 2004, 94-95). Java ME perustuu Sun Microsystemsin, joka on nykyään osa Oracle Corporationia, Java teknologiaperheeseen. Se on Java-ohjelmistoalustoista toiminnallisuuksiltaan suppein ja siten vaatimuksiltaan kevyin. Sen sisältämät ominaisuudet vaihtelevat laiteprofiilien ja konfiguraatioiden mukaan. (Kontio 2002, 12-13, 16.) Java ME on ideaalinen tekniikka mobiiliratkaisujen kehittämiseen, sillä sitä tukevia päätelaitteita on markkinoilla reilusti yli 100 miljoonaa ja sovelluskehitys on suhteellisen helppoa (Mertanen 2004, 142).

4.2.1 Konfiguraatiot ja profiilit

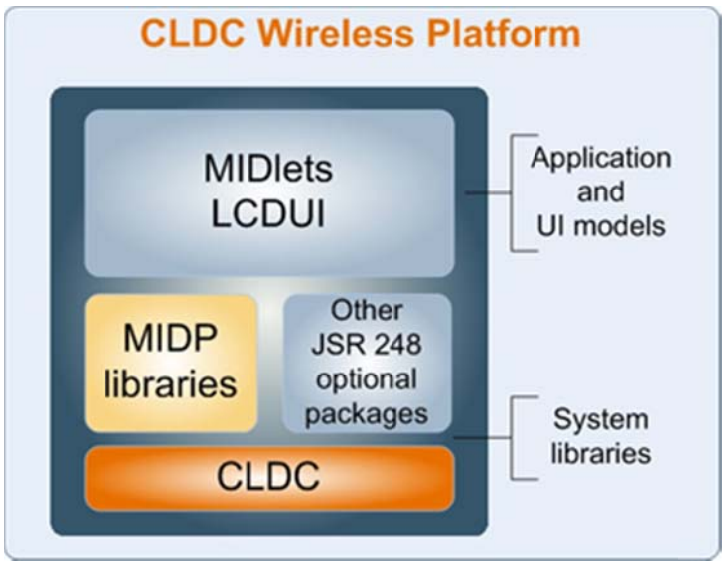
Mertasen (2004) mukaan konfiguraation päätehtävä on toimia rajapintana Java virtuaalikoneelle. Ainoa tapa käyttää virtuaalikoneen toimintoja ja resursseja on konfiguraation tarjoamien rajapintojen kautta. Ratkaisu lisää Javan turvallisuutta mobiililaitteessa. (Mertanen 2004, 144.) Konfiguratiot on suunniteltu resursseiltaan ja fyysisiltä ominaisuuksiltaan samankaltaisille laitteille. Ne määrittelevät sellaisen minimikokoonpanon, jolla kyseinen Java ME ympäristö voi toimia. Konfiguraation vaatimukset sisältävät mm. virtuaalikoneen ominaisuuksia ja ohjelmistokirjastoja. Nykyään on yleisesti käytössä kaksi eri konfiguraatiota: CDC eli *Connected Device Configuration* ja CLDC eli *Connected Limited Device Configuration*. Näistä CDC on tarkoitettu suorituskyvyltään ja ominaisuuksiltaan laajemmille laitteille ja CLDC puolestaan on karsitumpi ja kevyempi versio. (Kontio 2002, 13-15; Mikkonen 2004, 89-90.) Tämän opinnäytetyön kannalta CLDC on mielenkiintoisempi tarkastelukohde, koska kaikki älypuhelimet toteuttavat sen määrittelemät vaatimukset. Jatkossa tarkastelen Java ME -alustaa vain CLDC:n kannalta.

Laiteprofiilit määrittelevät kohdelaitteen tarkat ominaisuudet yksittäisen laitteen tai laiteryhmän tasolla, koska pelkkä fyysiseen samankaltaisuuteen perustuva konfiguraa-

tio ei riitä spesifioimaan laitejoukkoja sovellusten kannalta riittävällä tarkkuudella. Profiili yhdistää laite- tai käyttöjärjestelmätoimittajan implementoiman Java ympäristön ja ohjelmoijan käytössä olevat rajapinnat ja luokkakirjastot. Ensimmäinen ja mahdollisesti tunnetuin profiili on tämän työn kannalta olennainen MIDP eli *Mobile Internet Device Profile*. Profiilien tavoite on, että kaikki samanlaiset laitteet voisivat käyttää keskenään samoja sovelluksia. Profiilit määrittelevät mm. millaisia ohjelmistorajapintoja ja luokkakirjastoja laite tarjoaa ohjelmoijan käyttöön. Java standardin mukaisesti jokainen saman profiilin jakava laite toteuttaa vähintään samat ominaisuudet samalla tavalla. Suurelta osin se on toteutunut, mutta toisinaan poikkeavuuksia esiintyy. Laite ei ole sidottu vain yhteen profiiliin, vaan voi toteuttaa useita eri profiileja ominaisuuksiensa ja käyttötarkoituksensa mukaan. (Kontio 2002, 13-16; Mikkonen 2004, 91-92.) Profiilin tarjoamat tärkeimmät tehtävät sovelluskehittäjän kannalta ovat pääsy käyttöliittymän, tiedostonhallinnan ja sovelluslogiikan vaatimiin kirjastoihin ja rajapintoihin. Sovelluksia kehittäessä kannatta aina käyttää uudempaa ja yrityskäyttöön sopivampaa MIDP 2.0 profiilia, jos kohdelaitteet sitä tukevat. (Mertanen 2004, 144-145.)

4.2.2 Toimintaperiaate ja rakenne

Sisäiseltä rakenteeltaan Java ME on kerroksittainen ja voidaan jakaa kolmeen tasoon. Lisäksi laitteen käyttöjärjestelmä voidaan nähdä rakenteen neljäntenä ja alinpana kerroksena. Java virtuaalikone asettuu laitteen käyttöjärjestelmän päälle. Virtuaalikoneessa puolestaan ajetaan konfiguraatioita, joiden päälle profiilit asettuvat. (Kontio 2002, 13-14.) Käytössä oleva konfiguraatio ja profiili määrittelevät, millaisia sovelluksia laitteella on mahdollista suorittaa. Seuraavassa kuvassa (kuva 2) on esitetty havainnollistava CLDC:n perusrakenne.



KUVA 2. Java ME CLDC konfiguraation rakenne (Java Platform Overview 2010)

Kaikki Java sovellukset mukaan lukien J2ME-sovellukset suoritetaan Java virtuaalikoneessa. CLDC käyttää KVM nimistä virtuaalikonetta. (Kontio 2002, 41.) Jatkossa tarkoitan sitä puhuessani Java virtuaalikoneesta. Kontion mukaan (mt.) KVM on tarkoitettu suorituskyvyiltään heikoille laitteille. Sen lähtökohtana on keveys, siirrettävyys ja muokattavuus. Kaikki Java koodi muunnetaan ensin *byte code* -välikielelle, jota virtuaalikone sitten suorittaa. Tästä periaatteesta johtuva selkeä etu on, että periaatteessa Javalla toteutettuja ohjelmia on mahdollista ajaa missä tahansa laitteessa, joka kykenee ajamaan Java virtuaalikonetta. (Mikkonen 2004, 11, 84-85.) Standardin mukaan suoraan virtuaalikoneen päälle ei kuitenkaan tule ohjelmoida toimintoja, vaan ne on toteutettava profiilin tai konfiguraation päälle.

4.2.3 Sovelluskehitys

Java ME on osittain suoraan yhteensopiva tai pienin muutoksin yhteensopiva Java SE:n kanssa. Tämä on selvä etu muunnettaessa mobiilisovelluksia työasemille tai toisinpäin. Toisaalta sekä CDC että CLDC laajentavat J2ME:n ominaisuuksia siinä määrin etteivät ne automaattisesti ole suoraan siirrettävissä alustalta toiselle. (Kontio 2002, 14-15.) CLDC:n ja MIDP:n rajoitukset on syytä huomioida sovelluksia kehittäessä. Esimerkiksi MIDP ei tue oletusarvoisesti liukulukuja, vaikka CLDC on versiosta 1.1 alkaen sisältänyt tuen niille. Monet osa-alueet kuten virheenkäsittely, säikeet ja luokkien viimeistely on jätetty toteuttamatta tai toteutettu huomattavasti rajallisemmin kuin monipuolisemmissa konfiguraatioissa ja Javan laajemmissa SE ja EE versioissa.

KVM asettaa myös rajoituksia mm. käyttöjärjestelmän natiivikutsujen suorittamiseen, itse toteutetuille luokkalataajille ja *reflection*-tekniikalle. Rajoitukset ovat seurausta suoritusympäristöstä ja tiukemmasta turvastandardista, jonka mukaisesti kaikki KVM sovellukset ajetaan ns. *sandbox* tilassa. (Kontio 2002, 49-54; Mikkonen 2004, 92-97.)

Sun tarjoaa kattavan dokumentaation ja kehitystyökalut MIDP laitteiden ohjelmistokehitykseen. Myös lukuisat muut ohjelmisto- ja laitetoimittajat tarjoavat työkaluja joko yleisesti mobiilisovelluksien tuottamiseen tai rajoitetusti vain omalle laitealustalleen. Virallinen J2ME työkalu MIDP ja CLDC ohjelmistokehitykseen on Sunin Java ME SDK. Se sisältää laite-emulaattorin, IDE-ohjelmiston, dokumentaation, profiileja ja muita keskeisiä työkaluja. (Kontio 2002, 17.)

J2ME-sovellusten kehittäminen on periaatteessa nopeampaa ja helpompaa kuin suoraan laitetasolle tai käyttöjärjestelmälle ohjelmointi. Helppous saavutetaan piilottamalla matalan tason ohjelmistokomponentteja ja muita ominaisuuksia sovelluskehittäjältä. Se tapahtuu erilaisten API-rajapintojen toteuttaman abstraktion avulla. Ohjelmoijan ei tarvitse tuntea alla piilevää käyttöjärjestelmää tai sen rakennetta, vaan Javan tarjoamat toiminnallisuudet ovat suurelta osin alustariippumattomia ja standardeja. (Mikkonen 2004, 81.)

CLDC sisältää kirjastoja J2ME toteutuksesta ja varta vasten konfiguraatiolle itselleen toteutettuja kirjastoja. Sovelluskehittäjän kannalta tärkeimmät luokat kuuluvat paketteihin *javax.microedition, java.lang, java.util* sekä *java.io*. (Kontio 2002, 55-59.) MIDP päälle toteutettuja sovelluksia kutsutaan MIDleteiksi. MIDP laajentaa CLDC:n kirjastovalikoimaa juuri tietylle laiteryhmälle sopivilla paketeilla. Sen tärkeimmät luokat sijaitsevat *javax.microedition.midlet, javax.microedition.rms, javax.microedition.io* ja *javax.microedition.lcdui* paketeissa. Ne sisältävät mm. kansainvälisyystuen, turvallisuusmäärityksiä, verkkokirjastoja ja *I/O*-toimintoja. MIDP määrittelee älypuhelimille ja vastaaville laitteille sopivia käytänteitä mm. käyttöliittymäarkkitehtuurin suhteen. Käytännössä MIDP tarjoaa sekä korkean että matalan tason API:n, jonka avulla käyttöliittymä voidaan toteuttaa. Se myös toteuttaa monia CLDC:n määrittelemiä kehyksiä laitteille sopivalla tavalla. (Kontio 2002, 65-66, Mikkonen 2004, 90-91.) Sovelluksia voidaan konfiguraatioiden ja profiilien lisäksi laajentaa lisäpaketeilla, jotka on kohdennettu tietylle teknologiaspesifille ohjelmointirajapinnalle (Java Platform Overview 2010).

Kuten aiemmin mainitsin, MIDP sisältää erilliset metodit ja luokat käyttöliittymien luomiseen kahdella eri menetelmällä. Korkean tason rajapinta mahdollistaa käyttöliittymäelementtien suoran käytön, kun taas matalan tason toiminnoilla on mahdollista piirtää erilaisia komponentteja ruudulle. (Mikkonen 2004, 111-112.) Useimmiten MIDP luokat määrittelevät graafisen komponentin toiminnallisuudet, mutta käyttöjärjestelmän grafiikkakirjastot huolehtivat sen piirtämisestä ruudulle. Tämän seurauksena sovelluksen ulkoasu voi olla hyvinkin vaihteleva eri laitteilla ja käyttöjärjestelmillä.

Jokaisen Java-sovelluksen tapaan MIDP-sovellus sisältää pääluokan. Pääluokan on tässä tapauksessa periydyttävä MIDlet luokasta ja toteutettava sovelluksen elinkaaren kannalta kolme tärkeää metodia. Ne sisältävät ohjelman käynnistämiseen, sammuttamiseen ja pysäyttämiseen tarvittavat komennot ja vaikuttavat suoraan MIDletin tilaan. (Kontio 2002, 66.)

Asennusta varten Java ME -sovellukset paketoidaan MIDlet pakkauksiin (*midlet suite*). Pakkaus sisältää kaikki tarpeelliset tiedostot kuten sovelluskoodin ja muut resurssit sovelluksen suorittamiseksi. Paketin ja sovelluksen kuvaus on määritelty *manifest* ja JAD tiedostoissa. Muiden Java-sovellusten tapaan kokonaisuus, JAD tiedostoa lukuun ottamatta, on paketoitu JAR tiedostoformaatin mukaiseksi tiedostoksi. (Mikkonen 2004, 100-101.)

5 OHJELMISTOTUOTANTO

Tässä pääluvussa kerron ohjelmistotuotannosta yleisellä tasolla, kuvailen sitä prosessina ja perehdyn ohjelmistojen elinkaareen. Tarkoituksena on tarjota riittävästi tietoa ohjelmistojen järjestelmällisestä ja suunnitelmallisesta toteutuksesta, jotta prosessia voidaan soveltaa käytännön tasolla. Opinnäytetyöni käytännön osuus on toteutettu projektityöskentelynä perustuen yleisiin ohjelmistotuotannon periaatteisiin.

5.1 Määritelmä

Ohjelmistotuotanto on laajakäsitteinen termi. IEEE määrittelee sen sovellukseksi sellaisia lähestymistavaltaan systemaattisia, kurinalaisia ja määrällisiä toimintoja, joita

käytetään ohjelmistojen kehittämiseen, käyttämiseen ja ylläpitoon (IEEE 610.12 1990, 67). Termi voidaan tulkita käsittämään ohjelmistotyötä, joka tuottaa lopputuloksenaan riittävällä tasolla vaatimukset ja tavoitteet täyttäviä järjestelmiä. Se käsittää siten kaikki keskeiset tuotantoprosessit ja niiden osa-alueet. Toisinaan ohjelmistotuotannon synonyymeinä tai lähes samaa tarkoittavina termeinä esiintyvät ohjelmistotekniikka ja systeemityö. (Haikala & Märijärvi 2006, 16.)

Jotta voidaan puhua prosessista ohjelmistotuotannon yhteydessä, on syytä aloittaa määrittelemällä, mikä on käytännössä ohjelmistotuotannollinen prosessi. Aiemmin esitelty määritelmä on hyvin teoreettinen ja ei välttämättä anna käytännönläheistä käsitystä. Ohjelmistoprojekti voi muodostua uudesta tuotekehityksestä, muutostyöstä, ohjelman tai sen osien uudelleenkäytöstä, jatkokehityksestä, ylläpidosta tai mistä tahansa muusta aktiviteetista, jonka lopputuloksena syntyy ohjelmistotuotteita (New Product Development Glossary 2007).

5.2 Ohjelmistoprojektit

Yleisin tapa tuottaa ohjelmistoja ja suorittaa muita ohjelmistotuotannollisia prosesseja on projektipohjainen työtapa (Haikala & Märijärvi 2006, 53). On myös huomioitava, että ohjelmistotoimituksen suhteen varsinainen ohjelmistoprojekti voi muodostaa vain pienen osan työmäärästä ja liiketoiminnasta. Projektityön analysointi, rakenteen määrittely ja kuvaus sekä projektinhallinta ovat laajoja asiakokonaisuuksia ja niistä on julkaistu paljon kirjallisuutta. Tässä työssä on mielekästä paneutua ohjelmistoprojekteihin vain erittäin yleisellä tasolla. Kuvailen seuraavaksi lyhyesti tärkeimmät projektityön osa-alueet ja työkalut. Käytännön sovelluksena kerron luvussa 6.5 opinnäytetyöprojektini suunnittelusta ja hallinnasta.

Projektinhallinnan tarkoitus on varmistaa, että projekti tuottaa halutunlaisen ohjelmiston määrätyssä aikataulussa ja budjetissa siten, että se täyttää asiakasvaatimukset (Luckey & Phillips 2006, 10). Project Management Instituten, PMIn, mukaan projektinhallinta voidaan jakaa yhdeksään ryhmään: laajuudenhallintaan, ajanhallintaan, kustannustenhallintaan, laadunhallintaan, henkilöstönhallintaan, viestinnänhallintaan, riskienhallintaan, ostojenhallintaan ja kokonaisuudenhallintaan (A Guide To The Project Management… 2008, 6-7, 43). Riippuen projektin laajuudesta, monimutkaisuudesta ja muista ominaisuuksista hallintaa voidaan keventää tai soveltaa sopivaksi.

Monesti etenkin suurten projektien kohdalla työvaiheet jaetaan päällekkäisiin tai peräkkäisiin osaprojekteihin mm. määrittely- ja toteutusprojekteihin (Haikala & Märijärvi 53). Se helpottaa hallintaa, kustannusten ja työmäärien arviointia sekä muuttujien ennustamista. Lisäjakona toteutetaan yleensä ositus, eli projekti jaetaan aktiviteetteihin, jotka puolestaan sisältävät tehtäviä ja mahdollisia virstanpylväitä. Lyhyet ja selkeästi määritellyt tehtävät on helppoa arvioida työmäärän, resurssien ja kulujen suhteen. Sijoittamalla tehtävät ja aktiviteetit aikajanalle tai kalenteriin saadaan suuntaa antava arvio aikataulusta. Yleisesti pidetään hyvänä käytäntönä varata aikaa myös ennustamattomille tehtäville ja ongelmille. (Haikala & Märijärvi 2006, 230-236; A Guide To The Project Management… 2008, 18-22.) Lyhyesti voidaan sanoa, että tehokas projektinhallinta perustuu muutamaan perustyökaluun. Näitä ovat mm. projektin avaus, projektisuunnitelman laatiminen ja sen osa-alueiden analysointi ja määrittely, projektin toteuman seuranta ja ohjaus, projektiviestintä ja projektin päättäminen. (Haikala & Märijärvi 2006, 225-228.) Projektisuunnitelma kattaa kaiken projektin hallinnan kannata olennaisen tiedon. Sen tarkoitus on kuvata, kuinka asetetuin ehdoin ja menetelmin päästään tavoiteltuun lopputulokseen. (Haikala & Märijärvi 2006, 240-242.)

Ohjelmistoprojektit eivät aina suju kuten on suunniteltu. Aikataulut venyvät, budjetti paisuu ja toimeksiantaja odottaa selvitystä. Ilmiö on hyvin tyypillinen ohjelmistoalan projekteille ja johtuu useiden osatekijöiden summasta (Haikala & Märijärvi 2006, 54). Selitystä voidaan hakea kolmesta lähteestä: toimeksiantajasta, projektin hallinnasta ja projektitiimistä. Toimeksiantaja ei välttämättä osaa määritellä asiakasvaatimuksia tai tavoitteita, projektin hallinta ei osaa kommunikoida, ohjata projektia tai nähdä kokonaisuutta tai projektitiimin moraali ja osaaminen voi olla heikko. (Luckey & Phillips 2006, 19.) Vaikka ongelmat ja niiden ratkaisut ovat ohjelmistoalan ammattilaisten tiedossa niihin voi olla vaikea puuttua kilpailuedun menettämisen vuoksi. Haikala ja Märijävi (2006, 56) kuvailevat ilmiötä paradoksiksi, jossa ohjelmistokaupan voittaa se toimittaja, joka on arvioinut projektin kustannukset ja aikataulun eniten alakanttiin. Tarkempia epäonnistumisien syitä voivat olla mm. epärealistinen aikataulu, vääränlainen budjetointi, muutokset vaatimuksissa ja niiden riittämätön hallinta, ongelmat projektitiimissä ja tekniikan tai sovellusalan huono tuntemus (Haikala & Märijärvi 2006, 54; Luckey & Phillips 2006, 13-18). Yleensä ohjelmistoprojektien tarkka arviointi etukäteen on mahdotonta, koska jokainen projekti voi olla hyvinkin erilainen. Vastaa-

vanlaisesta projektista ei välttämättä ole olemassa aiempaa tietoa. Lisäksi ohjelmistotuotanto on alana verrattain nuori. (Haikala & Märijärvi 2006, 54-56.)

Yksinkertainen tapa hallita kolmea päämuuttujaa, aikaa, budjettia ja laajuutta, on projektinhallinnan kolmiomalli. Siitä voidaan johtaa vaatimustenhallintatyökalu korvaamalla kustannus, aika ja laajuus niitä vastaavilla positiivisilla adjektiiveilla: nopea (aika), halpa (kustannukset), kattavat ominaisuudet (laajuus). Näistä kolmesta adjektiivista voidaan saavuttaa vain kaksi. (Fried 2005.)

5.3 Ohjelmistokehityksen elinkaari ja vaiheet

5.3.1 Elinkaari

Ohjelmistotuotantoprosessi voidaan jakaa erilaisiin osa-alueisiin mm. projektinhallintaan, tuotteenhallintaan, laatujärjestelmään, laadunvarmistukseen, dokumentointiin, määrittelyyn, suunnitteluun, toteutukseen, testaukseen, käyttöönottoon ja ylläpitoon (Haikala & Märijärvi 2006, 35-36). Osa-alueita voidaan tarkentaa tai yleistää painotuksesta, resursseista, aikataulusta, budjetista ja muista muuttujista riippuen. Ne kannattaa kuitenkin käsitellä lopputuloksen kannalta riittävällä tarkkuudella.

Ohjelmiston elinkaari on aikajana, jolle kaikki ohjelmistoa koskevat tapahtumat sijoittuvat. Elinkaari alkaa ohjelmistoprojektin aloittamisesta ja päättyy sen käytöstä poistumiseen. Sen sisältämiä tapahtumia voidaan jakaa tarkemmin erilaisiin vaiheisiin riippuen käytetystä vaihejakomallista. Kuitenkin riippumatta mallista, eri vaiheisiin liittyy laatujärjestelmän mukaisia toimenpiteitä mm. laadunvarmistuksia. (Haikala & Märijärvi 2006, 36-37.) Tyypilliset vaiheet ohjelmiston elinkaaren aikana ovat esitutkimus, määrittely, suunnittelu, toteutus, testaus, dokumentointi, käyttöönotto ja ylläpito (Haikala & Märijärvi 2004, 36-39; Kosonen ym. 2008, 20-21).

5.3.2 Esitutkimus ja määrittely

Esitutkimus on periaatteessa ohjelmistoprojektin tärkein vaihe. Väite kuulostaa rohkealta, mutta sitä voidaan perustella alan veteraanin Gerald Weinbergin sanoin: ”Mikään ei korvaa ratkaistavan ongelman perusteellista ymmärtämistä - joskus voi tosin käydä hyvä tuuri” (Haikala & Märijärvi 2006, 31). Esitutkimuksessa selvitetään tärkeimmät

asiakas- ja järjestelmävaatimukset yleisellä tasolla. Siinä selvitetään miksi ohjelmistoprojekti kannattaa tai ei kannata toteuttaa. Jos esitutkimus epäonnistuu, on olemassa riski koko projektin ajautumisesta väärään suuntaan, koska ongelmilla on tapana eskaloitua. Pahimmillaan lähdetään toteuttamaan mahdotonta projektia. (Luckey & Phillips 2006, 42-43.) Esitutkimus voidaan nähdä omana vaiheenaan tai osana määrittelyvaihetta. Ero määrittelyn ja esitutkimuksen välillä voi olla huomaamaton tai sitä ei ole ollenkaan, koska esitutkimuksen tuloksia tarkennetaan määrittelyvaiheessa. Tietyissä tapauksissa esitutkimus voidaan suorittaa osana tukitoimintoja, erityisesti vaatimustenhallintaa. (Haikala & Märijärvi 2006, 37.)

Määrittelyvaiheessa tulevan ohjelmiston vaatimukset analysoidaan ja esitetään yleensä kirjallisessa muodossa (Kosonen ym. 2008, 20). Toisinaan määrittelyvaihetta kutsutaan vaatimusmäärittelyksi, koska sen tärkein tavoite on asiakasvaatimusten muuntaminen ohjelmistovaatimuksiksi. Vaatimusten avulla määritellään, mitä ohjelman pitää tehdä ja mitä ominaisuuksia siinä on oltava. Vaatimusmäärittely ei yleensä ota kantaa tekniseen toteutukseen, mutta voi tarvittaessa asettaa sille vaatimuksia ja reunaehtoja. Määrittelydokumentaatio kattaa tyypillisesti toiminnalliset ja ei-toiminnalliset vaatimukset, käyttötapauksia, rajoitukset ja vaatimukset täyttävän ratkaisumallin tai järjestelmän kuvauksen. Vähimmäisvaatimus määrittelyvaiheelle on tuottaa ainakin toiminnallinen määrittely dokumentti. (Haikala & Märijärvi 2006, 39, 78-81.)

Määrittelyvaihe tuottaa siis erilaisia spesifikaatioita suunnittelun, toteutuksen ja testauksen ohjaukseen. Se on myös erittäin tärkeä suorittaa huolellisesti, jotta päädytään oikeanlaiseen ohjelmistoon, joka sisältää kaikki vaaditut ominaisuudet ja toiminnot eikä mitään ylimääräistä. Tarkoituksenmukaisesti ja riittävällä tarkkuudella toteutettu esitutkimukseen pohjautuva toiminnallinen määrittely takaa, että projekti pysyy hallinnassa ja onnistuu arvioiden sekä suunnitelmien mukaisesti. (Haikala & Märijärvi 2006, 91-94.)

5.3.3 Suunnittelu

Suunnittelu on vaihe, jossa määrittelyn tuottamat asiakas- ja järjestelmävaatimukset käsitellään, analysoidaan ja muunnetaan toteutettavan järjestelmän teknisiksi vaatimuksiksi ja toteutussuunnitelmiksi. Tekniset vaatimukset vastaavat kysymykseen, miten järjestelmä tai ohjelmisto, joka kuvattiin määrittelyssä, toteutetaan käytännössä.

Riippuen projektin mittakaavasta, suunnitteluvaihe voi olla jaettu useampiin alivaiheisiin ja tasoihin tai se voidaan toteuttaa tuottamalla minimissään tekniset vaatimukset kattava dokumentti. Tyypillisiä suunnittelutasoja ovat arkkitehtuuri- ja moduulisuunnittelu. (Haikala & Märijärvi 2006, 40.) Suunnittelun tavoitteena on kuvata toimintamenetelmien, tietorakenteiden, liittymien ja ominaisuuksien tekninen toteutus. Ohjelmistojen suunnittelussa sanonta ”hyvin suunniteltu on puoliksi tehty” on erittäin paikkansa pitävä. (Kosonen ym. 2008, 20-23.) Hyvän suunnitelman tunnusmerkkejä ovat yksinkertaisuus, suoraviivaisuus, osittaminen, abstraktioiden käyttö sekä yhtenäinen ratkaisutapa ja -filosofia kautta suunnitelman (Haikala & Märijärvi 2006, 313).

Ohjelman tai tietojärjestelmän suunnittelu voidaan toteuttaa muodollisilla menetelmillä, kuvausmenetelmillä, vapaamuotoisesti sanallisesti tai pseudokoodin avulla. Jotta suunnittelu ja sen lopputulokset olisivat tarkoituksenmukaisia, on tunnettava riittävällä tasolla käytettävät tekniikat, ohjelmien perusrakenteet ja ratkaisumallit. (Kosonen ym. 2008, 23-27.) Ohjelmistojen ja tietojärjestelmien suunnitteluun on olemassa lukuisia erilaisia periaatteita ja toimintaohjeita. Yleisesti hyväksi havaittu tapa on jakaa arkkitehtuurisuunnittelussa kokonaisuus mahdollisimman pieniin, mutta mielekkäisiin kokonaisuuksiin ja suunnitella niiden väliset rajapinnat sekä riippuvuudet. Sen jälkeen moduulien sisäinen toteutus ja toiminnallisuudet on helpompi ja yksinkertaisempi suunnitella. Ohjelmistot suunnitellaan nykyään pitkälti OOP-periaatteiden mukaisesti, jos se on teknisesti järkevää ja mahdollista. Muunlaiselle rakenteelle on omat paikkansa, mutta oliokeskeinen suunnittelu on muodostunut hyväksi havaituksi käytännöksi. (Haikala & Märijärvi 2006, 305-310.) Määrittelyn aikana luotuja käyttötapauksia voidaan soveltaa myös suunnitteluvaiheessa analysoimalla niitä teknisestä näkökulmasta. Tehokas työkalu ohjelmistosuunnittelun avuksi on UML-kuvauskieli. Sen avulla suunnitelman eri osa-alueet voidaan visualisoida ja esittää standardin mukaisesti ja yksiselitteisesti.

5.3.4 Toteutus

Kun ohjelmiston vaatimukset on määritetty ja suunnittelu suoritettu, seuraa toteutusvaihe. Vaihejakomallista ja projektisuunnitelmasta riippuen eteneminen ei välttämättä tapahdu lineaarisesti, vaan vaiheet voivat olla päällekkäisiä ja iteratiivisia. Toteutusvaiheeksi kutsutaan varsinaista ohjelmointia, käyttöliittymän graafista toteutusta ja yleisesti määritellyn järjestelmän toteuttamista. Suppeassa mielessä se on toimintaoh-

jeiden ja komentojen kirjoittamista valituilla ohjelmointikielillä. (Kosonen ym. 2008, 20-21.) Yleensä toteutusvaihe tapahtuu ainakin osittain samanaikaisesti testauksen kanssa. Yksikkötestaus voidaan suorittaa moduulin toteutuksen yhteydessä ja integraatiotestaus samalla, kun moduulien välisiä rajapintoja toteutetaan. Testausta ei kuitenkaan kannata jättää yksinomaan moduulin tai liittymän ohjelmoijan harteille eikä suorittaa sitä vain toteutuksen lomassa.

Ohjelmistojen toteutukseen on olemassa lukuisia menetelmiä ja filosofioita, joita voidaan käyttää sellaisenaan, sovelletusti tai yhdistelemällä. Suosituimpiin filosofioihin kuuluvat olio-ohjelmointi ja menettelytavoista suosiotaan ovat nostaneet mm. ketterä kehitys *(agile software development)* ja nopea kehitys *(rapid application development)* (Guigova, 2009; Vanderboom 2009). Valitettavan usein käytössä ei ole mitään organisoitua toimintatapaa, vaan sitä kuvaa enemmänkin paradigma ”koodaa ja korjaa” (Fowler 2000). Kerron tarkemmin ohjelmistotuotannon metodiikasta ja toimintamalleista luvussa 5.4. Ohjelmia toteuttaessa kannattaa noudattaa hyväksi havaittuja käytänteitä mm. lähdekoodin kommentointia, selkeää nimeämiskäytäntöä, versionhallintaa, tarkastuksia, arviointeja ja ohjelmointikielikohtaisia standardeja. Monet asiantuntijat suosittelevat edellä mainittuja laatua parantavia menetelmiä (ks. Martin 2009, 2-12). Haikala ja Märijärvi (2006, 276-277) painottavat tarkastuksien merkitystä työn laatuun. On kuitenkin muistettava, että aina löytyy parannettavaa ja optimoitavaa. On järkevää tehdä rajaveto niin, että saavutetaan hyvä hyötysuhde. Hyvän ohjelmakoodin ja laadun tarkkailun merkitystä ei silti voi korostaa liikaa, sillä sen avulla minimoidaan myöhempien vaiheiden, erityisesti testauksen, ja mahdollisten jatkoprojektien työmäärää (Haikala & Märijärvi 2006, 275-276).

5.3.5 Testaus

Joskus ajatellaan, että testaus suoritetaan vasta kun toteutettava ohjelma tai järjestelmä on valmis. Tämä ajattelumalli on käytännön kannalta väärä ja johtaa usein huonoihin lopputuloksiin. (Haikala & Märijärvi 2006, 287.) Testaus tapahtuu monella tasolla ja sitä voidaan mallintaa esim. V-mallin avulla (mts. 40). Sen tarkoitus on suunnitelmallisesti etsiä ja korjata ohjelmistossa esiintyviä virheitä ja muita epäkohtia. Se, että virheitä ei löydy, ei ole hyvän ohjelman mittari, vaan huonon testauksen. Yleensä ohjelmakoodin tai moduulin toteuttajan ei ole järkevää suorittaa testausprosessia. Testauksen merkitystä ohjelmistoprojekteissa on suuri. (mts. 283-284.) Testaus ja siitä seu-

raavan virheiden ja vikojen korjaus voi aiheuttaa jopa 50 - 80 % ohjelmistoprojektin kuluista (mts. 40-41).

Testaus kuuluu laadunvarmistustoimenpiteisiin. Prosessiin liittyviä laadullisia vaatimuksia voidaan määritellä jo organisaation laatujärjestelmässä. (Haikala & Märijärvi 2006, 267-268.) Yleensä siihen katsotaan kuuluvaksi testauksen suunnittelu, testiympäristön luonti, varsinaiset testaustoiminnot ja tulosten analysointi (mts. 283). Ilmenneiden virheiden määrään voidaan vaikuttaa suoraan toteutusvaiheen menetelmillä ja hyvällä laadunvarmistuksella. Testausta ovat myös toteutusvaiheeseen yhdistetyt tarkastukset, katselmukset ja muut analyysit. (Mts. 275-276.) Kuten aiemmin mainitsin, testaus suoritetaan yleensä eri tasoissa. Testauksen tasoja ovat mm. järjestelmätestaus, integrointitestaus, moduulitestaus, hyväksymistestaus ja käytettävyystestaus. Hyvä testaus on suunnitelmallista ja tapahtuu laadittuja spesifikaatioita vasten. Se voidaan suorittaa lasilaatikko-, harmaalaatikko tai mustalaatikkotestauksena riippuen halutaanko ohjelman tai moduulien sisäinen toiminta häivyttää vai sisällyttää prosessiin. Testaussuunnitelmassa tulisi määritellä kriteerit testauksen hyväksymiselle tai hylkäämiselle sekä ohjeistus tulosten arviointiin. Työkaluja testauksen automatisointiin ja suorittamiseen ovat mm. profiloijat, testipetigeneraattorit, vertailijat, testitapausgeneraattorit, emulaattorit ja simulaattorit. (Mts. 284-292.) Kaiken kattavia menetelmiä ei ole olemassa ja testaus on aina kompromissi käytettyjen resurssien ja saatujen tulosten välillä (mts. 293).

5.3.6 Käyttöönotto ja ylläpito

Kun ohjelmasta tai järjestelmästä toteutetaan riittävä kokonaisuus, voidaan suorittaa käyttöönotto. Käyttöönotto tarkoittaa kaikkia sellaisia tapahtumia, joiden avulla toteutettu ohjelma tai järjestelmä toimitetaan asiakkaalle ja saatetaan sellaiseen tilaan, että sitä voidaan hyödyntää halutulla tavalla. Käyttöönoton jälkeen alkaa tyypillisesti ylläpitovaihe. Samanaikaisesti ylläpidon kanssa voidaan kehittää seuraavaa versiota jatkokehitysprojektina tai uutena iteraationa. Ylläpito jakautuu suurpiirteisesti kolmeen ryhmään: korjaavaan, sopeutuvaan ja täydentävään ylläpitoon. Korjaava ylläpito kattaa ohjelman virheiden, eli bugien, korjaamisen. Sopeutuva ylläpito muuntaa ohjelmaa muuttuneiden vaatimusten tai ympäristön vuoksi. Täydentävässä ylläpidossa puolestaan ohjelman toiminnallisuutta parannetaan tai siihen lisätään uusia ominai-

suuksia. Monesti täydentävä ylläpito ja tuotteen jatkokehitysprojekti kulkevat käsi kädessä. (Haikala & Märijärvi 2006, 41.)

Olennaisia työkaluja ylläpidossa ja käyttöönotossa ovat projektin aikana syntyneet dokumentaatiot, joista tehdään ko. vaihetta tukevat versiot. Dokumentteja ovat mm. käyttöohjeet, asennus- ja ylläpitodokumentit, koulutusmateriaali ja tekninen dokumentaatio. (Haikala & Märijärvi 2006, 75.) Ohjelmistojen toimitusketjut eivät ole vielä nykyisin samalla tasolla fyysisten tuotteiden toimitusprosessien kanssa, vaikka ohjelmistotuotanto on yli 50 vuotta vanha tuotantoala. Tulevaisuudessa uudenlaiset mallit ohjelmistokehityksessä ja toimitusketjuissa muokkaavat alaa. Muutamia nousevia trendejä ovat mm. avoimen lähdekoodin toimitukset, web- ja yhteisöpohjaiset jakelut, SOA-mallin mukainen toimitusketju ja globaali ohjelmistokehitys toimitusketjuineen. (Sabbah 2007.)

5.3.7 Dokumentointi

Dokumentointi on tärkeä osa ohjelmistotuotantoprosessia. Kaikki työvaiheet ja osa-alueet tulisi dokumentoida osana kyseisiä toimintoja eikä erillisenä työvaiheena. Haikala ja Märijärvi (2006, 65) kuvailevat hyvän dokumentaation ja spesifikaation tunnusmerkkejä ominaisuuksilla täydellisyys, tarkkuus, virheettömyys, ymmärrettävyys, testattavuus ja jäljitettävyys. Useimmiten kaikkien edellä mainittujen ominaisuuksien saavuttaminen on mahdottomuus. Yleisiä ongelmia ovat puutteellisuus, vaikea ymmärrettävyys ja moniselitteisyys. (Mts. 65.) On havaittu, että usein projektin aikataulupaineiden kasvaessa dokumentointi kärsii ensimmäisenä. Dokumentoinnin puutteellisuus voi pahimmillaan johtaa koko ohjelman uudelleenkirjoittamiseen. (Mts. 70-71.)

Hyvä dokumentaatio palvelee niin kehittäjiä, käyttäjiä kuin ylläpitäjiäkin (Kosonen ym. 2008, 20-21). Projektidokumentaation tulisi sisältää ainakin eri vaiheiden tuottamat dokumentit kuten suunnittelu-, määrittely- ja testausdokumentit sekä projektisuunnitelman. Dokumentteja voi kuitenkin projektin laajuudesta riippuen syntyä kymmeniä (Haikala & Märijärvi 2006, 51). Muita yleisiä projektin aikana syntyviä dokumentteja ovat mm. sopimukset, raportit, pöytäkirjat, ennusteet ja toteumat (mts. 71).

Myös ohjelmakoodin sisältämät kommentit ovat tärkeä osa dokumentointia. Lähdekoodin kommenttien tarkoitus on kompensoida sen puutetta luonnollisessa ilmaisussa. Hyvä kommentointi selkeyttää monimutkaisia rakenteita, määrittelee standardeja, havainnollistaa, kuvailee ja esittää varoituksia tai muita olennaisia huomautuksia. (Martin 2009, 53-59.)

5.4 Vaihejakomallit ja metodiikka

Ohjelmistoprojektit voidaan suorittaa erilaisilla järjestelyillä, painotuksilla ja työvaiheilla. Vaiheistus tapahtuu vaihejakomallin avulla. Mallia voidaan käyttää sellaisenaan tai soveltaa eri osia eri malleista. Yleisimpiä vaihejakomalleja ovat mm. vesiputousmalli, prototyyppimalli, EVO-malli, spiraalimalli, V-malli, RUP, RAD sekä joukko ketterän kehityksen menetelmiä (Haikala & Märijärvi 2006, 42-47). Luckeyn ja Phillipsin (2006, 119) mukaan on olemassa kymmenittäin erilasia ohjelmistotuotannon menetelmiä, malleja ja sovelluksia. Jokaisessa mallissa on hyvät ja huonot puolensa ja ne soveltuvat erilaisiin projekteihin ja tarpeisiin. Tärkeintä on, että mallia noudatetaan kaikkien projektiin osallistuvien toimesta.

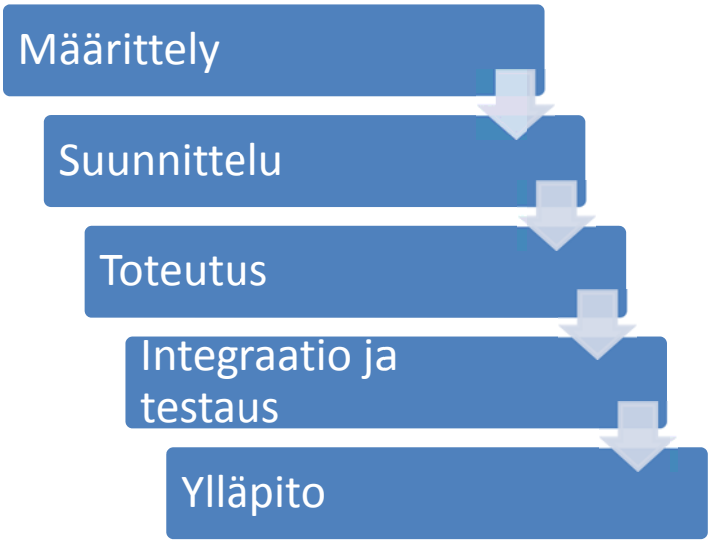
Esittelen seuraavaksi tunnetuimmat vaihejakomallit ja niiden keskeiset menetelmät. Ohjelmistokehityksen metodiikka voidaan myös karkeasti jakaa kolmeen ryhmään: oliokeskeiseen, funktionaaliseen ja strukturaaliseen. Keskityn tässä opinnäytetyössä vain oliokeskeiseen ohjelmistokehitykseen, mutta kaikki mallit ovat yleisesti sovellettavissa muun tyyppiseen kehitykseenkin.

5.4.1 Vesiputousmalli

Klassinen vesiputousmalli on eräs vanhimmista vaihejakomalleista. Vaikka se on saanut osakseen paljon kritiikkiä mm. jäykkyydestä ja vanhanaikaisuudesta, lähes kaikki uudemmat mallit perustuvat jollain tapaa siihen. Puhtaimmillaan vesiputousmalli on jaettu tarkasti rajattuihin vaiheisiin: määrittely, suunnittelu, toteutus, integraatio ja testaus ja ylläpito. Kun edeltävä vaihe on suoritettu, voidaan seuraava vaihe aloittaa. Yleensä vaiheen loppuun liitetään testausta, joka voi antaa palautteen edellisen tason uudelleen käynnistämisestä, jos vaatimuksia ei ole saavutettu. Yleensä vaihe tuottaa lopputuloksenaan joukon erilaisia dokumentteja seuraavan vaiheen syötteeksi. (Goodliffe 2007, 427-429) Haikala ja Märijärvi (2006, 41) toteavat, että vesiputousmalli on

erikoistapaus yleisestä järjestelmällisestä ongelmanratkaisumallista, mutta käytännön ohjelmistokehitys ei koskaan noudata sitä kirjaimellisesti.

Vesiputousmalli on klassinen esimerkki ohjelmistokehityksestä. Malli on yksinkertainen, suoraviivainen ja eteenpäin suuntautuva prosessi. Suurin osa sen ongelmista johtuu ohjelmistotyön luonteesta, joka ei täysin istu mallin periaatteisiin (Haikala & Märijärvi 2006, 41). Projektin aikana ilmenneitä muutoksia on työläs ja vaikea hallita, edellisiin vaiheisiin palaaminen on raskas prosessi eikä malli skaalaudu optimaalisesti suurille projekteille. Ratkaisun kokonaisvaltainen testaus tai esittäminen asiakkaalle on niin ikään mahdotonta ennen kuin projekti on edennyt vähintään integraatiovaiheeseen saakka. Jos tässä vaiheessa havaitaan ongelmia tai muutostarvetta on niiden korjaus ja toteutus yleensä hyvin kallista ja aikaa vievää. Vaikka mallia on kritisoitu paljon, joka toinen kehitysprojekti perustuu siihen tavalla tai toisella. (Goodliffe 2007, 429, 431) Alla esitetty kuva (kuva 3) havainnollistaa yksinkertaista vesiputousmallin mukaista etenemistä.



KUVA 3. Vesiputousmalli

5.4.2 Prototyyppimalli

Prototyyppimalli on yleisnimike sellaisille lähestymistavoille, joissa ratkaisua testataan osatoteutuksilla ennen varsinaista toteutusta. Mallin ideaalisia sovellusalueita ovat uuden teknisen ratkaisun kokeilu ja epäselvien asiakasvaatimusten selkeyttäminen. Sen vahvuuksia ovat myös käyttöliittymien määrittely sekä ratkaisun suorituskyvyn testaus. Protoilemalla syntynyttä tuotetta voidaan jatkokehittää valmiiksi ratkai-

suksi tai toteuttaa lopullisen ohjelmiston määrittely prototyyppiin perustuen. Prototyyppimalli voi sisäisesti muistuttaa vesiputousmallia tai muuta vaihejakomallia. (Haikala & Märijärvi 2006, 42-44.) Yleensä prototyypit kehitetään nopeasti, korkean tason ohjelmointikielillä, automaation avulla tai muilla nopeilla työkaluilla. Prototyypin pääpaino on usein toimintojen sijaan käyttöliittymässä. Jos projektiin liittyy merkittäviä riskejä, voi prototyyppilähestymistapa auttaa hallitsemaan ja kartoittamaan niitä. (Goodliffe 2007, 431.)

Myös prototyyppimallisessa lähestymistavassa on ongelmansa. Ratkaisu näyttää helposti valmiimmalta kuin se oikeasti onkaan, erityisesti asiakkaan näkökulmasta. Vaarana on myös jatkaa protoilua parantelemalla tuotetta loputtomasti. Ongelmat voivat johtaa projektinhallinnallisiin tai kustannuksellisiin ongelmiin. (Haikala & Märijärvi 2006, 43-44.) Prototyypin muuttaminen suoraan tuotteeksi pienin muutoksin voi olla houkutteleva vaihtoehto, mutta johtaa yleensä huonolaatuiseen tuotteeseen ja ylläpidollisiin ongelmiin (Goodliffe 2007, 432).

5.4.3 EVO-malli

EVO-mallin nimi tulee termistä *evolutionary delivery*. Se lähtee liikkeelle järjestelmän ytimen ja päätoimintojen toteuttamista. Uusia toiminnallisuuksia ja osajärjestelmiä kehitetään jatkoprojekteissa tai myöhemmissä sykleissä. Ohjelmisto kehittyy ikään kuin luonnollisen evoluution mukaisesti, jolloin aiemmin toteutettuja osia voidaan parantaa ja niistä pystytään oppimaan. Perinteisesti EVO-mallin syklit ovat kestoltaan lyhyitä verrattuna mm. vesiputousmalliin. Hyötynä tästä ovat vaatimusten, riskien ja testauksen tehokas hallinta. Yleensä ensimmäiset iteraatiot eivät tuota vielä määrittelyssä kuvattua järjestelmää, vaan lopputulos saavutetaan esimerkiksi viidennellä iteraatiolla. Lopputulos on kuitenkin hiotumpi ja viimeistellympi kuin yhdellä kertaa toteutettuna. Malli on oikeastaan sateenvarjotermi erilaisille inkrementaalisille menetelmille. (Haikala & Märijärvi 2006, 45.)

Yleinen ongelma kaikissa EVO-malliin perustuvissa iteratiivisissa menetelmissä liittyy eri vaiheissa toimitettujen tuotteiden ylläpitoon. Asiakkaan ongelmien selvittäminen ja toimitetun tuotteen ongelmien korjaus voi olla projektinhallinnallisesti haastava tehtävä, joka vie aikaa seuraavan mallin toteuttamiselta. Kyseinen skenaario luo myös ongelmia versionhallinnallisesti, jos korjattu toimitus poikkeaa suuresti kehitteillä

olevasta seuraavasta versiosta. Liian lyhyitä syklejä ei tulisi myöskään suorittaa, koska se voi rapauttaa ohjelmiston arkkitehtuurin ja aiheuttaa laadultaan heikomman lopputuloksen. (Haikala & Märijärvi 2006, 47.)

5.4.4 Spiraalimalli

Spiraalimalli on läheistä sukua aiemmin mainitulle EVO-mallille. Nimensä mukaisesti spiraalimallin mukainen projekti on jaettu samankaltaisiin vaiheisiin, jotka toteutetaan sen jokaisella kierroksella. (Haikala & Märijärvi 2006, 45.) Mallin mukaisesti toteutettu ohjelmisto on aluksi karkean tason toteutus, joka tarkentuu syklien lisääntyessä. Jokainen täyden ympyrän muodostava kierros spiraalissa vastaa periaatteessa vesiputousmallin mukaista toteutusta. Yleensä myöhemmät toistot suoritetaan laajemmin kuin aikaisemmat. Alussa kierrokset voivat olla hyvinkin lyhyitä ja nopeita, mutta vaativat ajan kuluessa yhä enemmän aikaa ja resursseja. (Goodliffe 2007, 432-433.)

Malli painottaa erityisesti riskienhallintaa. Riskit ja kriittiset prosessit on helppo hallita, koska spiraalimallin mukaisesti vaatimukset on määritelty tärkeysjärjestyksessä ja ne voidaan toteuttaa priorisoidusti. Jokaisessa uudessa kierroksessa vaatimukset, määritykset, suunnitelmat ja muut dokumentit tarkentuvat ja niitä voidaan parannella alkuperäisestä. (Goodliffe 2007, 432-433.)

5.4.5 V-malli

V-malli kehitettiin alun perin vesiputousmallin pohjalta Saksassa. Nimi johtuu mallin visuaalisesta esitystavasta, joka muistuttaa isoa V-kirjainta. Kirjaimen vasemmalla puolella on yleensä esitetty kehitysvaiheet ja oikealla puolella testaus sekä hyväksymistoiminnot. (Goodliffe 2007, 430). Koska V-malli on hyvin testauspainotteinen, mm. Haikala ja Märijärvi (2006, 289) käyttävät siitä nimeä testauksen V-malli. Sen toteutus- ja testaustasot jaetaan toisiaan vastaaviin kerroksiin. Määrittelyä vastaa järjestelmätestaus, arkkitehtuurisuunnittelua vastaa integrointitestaus, moduulisuunnittelua vastaa moduulitestaus ja ohjelmointia vastaa yksikkötestaus. Jokainen taso sisältää testauksen suunnittelun ja sen tulosten arvioinnin. (Mts. 288-289.)

Selvin ero vesiputousmalliin on, että vastaavan tason testaus ja hyväksymistoiminnot suoritetaan rinnakkain toteutuksen kanssa. Niillä on myös samanlainen painotus ja

tärkeys prosessissa. Sen avulla saavutetaan erityisesti testauksen tehokkuus ja asianmukainen suoritus. Tyypillisesti testausvaihe on sijoitettu projektin loppuun, jolloin testauksessa ilmenneiden ongelmien korjaaminen on kallista ja hidasta verrattuna välittömään tai lähes välittömään havaintoon. V-mallin selkeimmät edut verrattuna vesiputousmalliin näkyvät testauksen suunnittelussa ja tehokkuudessa, tuotteen laadussa ja ylläpidon minimoinnissa. Testaus ja virheiden korjaus voivat muodostaa jopa puolet projektin työmäärästä, joten niiden painottaminen alusta alkaen on mielekästä. (Goodliffe 2007, 430.)

5.4.6 RUP

Eli *rational unified process* on kaupallinen vaihejakomalli, joka pohjautuu EVO- ja spiraalimalliin (Haikala & Märijärvi 2006, 45). Vuodesta 2003 eteenpäin se on ollut IBM:n omistuksessa. Vaikka RUP on kokonaisuutena raskas toimintamalli, se on sisäiseltä toiminnaltaan joustava ja soveltuu hyvin oliokeskeiseen ohjelmistotuotantoon (Goodliffe 2007, 435). RUP mukainen prosessi jaetaan neljään päävaiheeseen: *inception*, *elaboration*, *construction* ja *transition*. Jokainen vaihe puolestaan koostuu tarpeen mukaan lyhyistä iteraatioista ja vesiputouksista. Siirryttäessä vaiheesta toiseen edellinen vaihe tuottaa projektisuunnitelman seuraavaa varten. (Haikala & Märijärvi 2006, 45-47.) Keskeisiä työkaluja toimintamallissa ovat mm. UML-standardin mukaiset kuvaukset ja käyttötapausten perusteella suoritettu suunnittelu (Goodliffe 2007, 435).

Inception-vaihe käsittää keskeisten projektinhallinnallisten ja ratkaisun kannalta olennaisten seikkojen analysoinnin ja määrittelyn. *Elaboration* sisältää perusarkkitehtuurin suunnittelun ja toteutuksen ja tarkennuksia edelliseen vaiheeseen. Se tarjoaa myös alustavia suunnitelmia ja dokumentteja eteenpäin. *Construction* on varsinainen toteutusvaihe, jossa jokainen iteraatio tuottaa uuden ns. beta-version ohjelmistosta. Tuotetut ohjelmistoversiot voidaan halutessa julkaista rajatulle testiryhmälle parantamaan testaustuloksia. Kun *construciton*-vaihe on tuottanut määritykset täyttävän tuotteen, siirrytään *transition*-vaiheeseen, jossa ratkaisu tuotteistetaan tarpeellisella tasolla ja toimitetaan kaikkine oheistuotteineen ja dokumentteineen asiakkaalle. (Haikala & Märijärvi 2006, 45-47.)

Mallin kaupallisuudesta huolimatta sillä on selviä etuja. Sen taustayritykset tarjoavat mm. konsultointia, koulutusta ja sovelluskehitystyökaluja. IBM rahoittaa ja suorittaa mallin kehitystä.

5.4.7 RAD

RAD eli *rapid application development* on toimintamalli, joka keskittyy nimensä mukaisesti ohjelmistojen nopeaan tuottamiseen ja julkaisuun. Sen pääperiaatteita ovat pienet tuotantotiimit, asiakkaiden osallistuminen kehitysprosessiin, prototyypit ja pitkälle viety automaatio. Projektin alussa määritelty aika-arvio on sitova ja sovellus toteutetaan sen puitteissa. Ominaisuudet arvioidaan esitutkimuksen avulla ja toteutetaan tärkeysjärjestyksessä. Usein joitakin toimintoja karsitaan lopullisesta ohjelmistosta. (Goodliffe 2007, 435; McConnel 2002, 18-20.)

RAD perustuu neljään perustekijään, jotka ovat klassisten virheiden välttäminen, aiemmin mainitut periaatteet, riskien hallinta ja aikataulusuuntautuneet käytännöt. Nopean ohjelmistokehityksen lopputulos on hyvin pitkälti riippuvainen tehokkaasta ja ammattitaitoisesti tiimistä sekä projektinhallinnasta. Oikeanlaiset teknologiavalinnat vaikuttavat myös ratkaisevasti, koska työkalujen on tuettava käytettyjä periaatteita ja niiden on palveltava ideologiaa ja tarpeita, jotta tavoitteet on mahdollista saavuttaa. Muita olennaisia vaikuttimia ovat kehitettävä tuote, erityisesti sen laadulliset tekijät, ja tuotantoprosessi. (McConnel 2002, 8-20.)

5.4.8 Ketterä kehitys

Ketterä kehitys on yksi viime vuosien suosituimmista ohjelmistokehityksen malleista (Haikala & Märijärvi 2006, 47). Se ei ole yksittäinen teoria, vaan yleisnimike erilaisille malleille ja niitä yhdistävälle ideologialle. Se on eräänlainen ohjelmistoalan vastaisku byrokraattisille ja jäykille perinteisille menetelmille. Ketterän kehityksen lähtökohta on, ettei ohjelmistoprojektia voida etukäteen arvioida tai suunnitella kaiken kattavasti, vaan sen on muista teknisistä prosesseista poikkeava ja dynaaminen kokonaisuus. (Goodliffe 2007, 433.) Se on saanut vaikutteita erityisesti EVO-mallista ja RAD-menetelmistä. Termi kattaa paljon erilaisia malleja, joista suosituimpia ovat mm. Scrum, XP (*extreme programming*), UP (*unified process*), DSDM (*dynamic systems development model*), Crystal Methods, FDD (*feature driven development*), LD (*lean*

development), ASD (*adaptive software development*) ja useimmat muut EVO-mallin johdannaiset. Tunnetuimpia ja käytetyimpiä malleja ovat nykyisin XP ja Scrum. (Highsmith 2002, xxxi-xxxiv; Larman 2009, 35-39.)

Vaikka ketterän kehityksen alle voidaan niputtaa laaja joukko erilaisia malleja ja menetelmiä, niitä kaikkia yhdistävät samat perusperiaatteet. Ne ovat iteratiivisia ja sykleiltään hyvin lyhyitä; jopa niin lyhyitä, että kehitysprosessia voidaan pitää jatkuvana. Hyvin yleinen lähestymistapa on myös testitapausten automatisointi ja niitä vasten ohjelmointi. (Haikala & Märijärvi 2006, 47.) Projektit toteutetaan tiiviissä yhteistyössä asiakkaiden ja tiimin välillä. Eräitä tärkeimpiä ketterän kehityksen periaatteista ovat myös nopea muutostenhallinta, pyrkimys yksinkertaisimpaan mahdolliseen ratkaisuun, sopeutuva suunnittelu, yhteistyö ja kevyt projektinhallinta. Ketterä kehitys pyrkii minimoimaan riskejä tuottamalla julkaisukelpoisia tuotteita jokaisen syklin päättyessä ja korostamalla testauksen merkitystä ja jatkuvuutta. Se pyrkii myös vähentämään byrokraattista dokumentointia ja korostamaan hyvän koodin merkitystä. Koodausstandardeja ja kommentointia noudattamalla ketterällä kehityksellä tuotetun ohjelmakoodin tulisi olla itseään kuvaavaa ja selkeää, joten sillä voidaan korvata suuri joukko suunnittelu- ja toteutusdokumentteja. Projektinhallinta ja projektin etenemisen seuranta tapahtuvat yleisesti asiakastapaamisten ja tuotteen todellisen toimivuuden arvioinnin avulla. Yleensä prosessia ei kuvata perinteisillä formaaleilla keinoilla vaan apuna käytetään ihmisläheisiä menetelmiä ja apuvälineitä. (Goodliffe 2007, 434; Highsmith 2002, 338-344; Larman 2009, 25-39.)

Ketterä kehitys soveltuu parhaiten pieniin tai keskisuuriin projekteihin. Tosin sen toimivuudesta suurissa projekteissa ei ole juuri tietoa puolesta eikä vastaan. (Haikala & Märijärvi 2006, 47.) Kritiikkinä ja ongelmina voidaan kuitenkin nähdä joitakin ketterän kehityksen vaatimuksia. Yleisesti voidaan sanoa, että mallien soveltaminen käytännössä vaatii asiantuntevan, kokeneen ja motivoituneen tiimin, jolla on mahdollisuus työskennellä projektin keston ajan samassa työympäristössä. Yleensä vaaditaan myös paljon asiakastapaamisia, joissa osapuolten olisi hyvä olla fyysisesti läsnä. (Goodliffe 2007, 434)

6 MOBIILIRATKAISUN KEHITTÄMINEN EPICOR FOR SERVICE ENTERPRISES -TOIMINNANOHJAUSJÄRJESTELMÄÄN

Opinnäytetyöni käytännön osuus on E4SE-toiminnanohjausjärjestelmän käyttöä helpottamaan tarkoitettu sovellus älypuhelimille ja muille yhteensopiville MID-laitteille. Tässä pääluvussa esittelen projektin toimeksiantajan, muut osapuolet, E4SE-järjestelmän sekä varsinaisen projektin. Sovellan aiemmin esittelemiäni ohjelmistotuotannon malleja sekä uusinta mobiiliteknologiaa ratkaisun kehittämisessä. Luvun tarkoitus on kuvata ohjelmistoprojektin elinkaari ja vaiheet riittävällä tarkkuudella luomaan realistinen kuva käytännön ohjelmistotyöstä.

6.1 Toimeksiantaja sekä muut osapuolet

Smart Time Oy on ohjelmistokehitykseen, ohjelmistoratkaisujen konsultointiin ja myyntiin erikoistunut yritys. Vuonna 2004 perustettu yritys on kasvanut voimakkaasti viime vuosien aikana. Samalla sen toiminta on monipuolistunut ja laajentunut.

Yritys perustettiin Internet-pohjaisen ajanvarauspalvelu haircut.fi ympärille, mutta sen toiminta laajeni käsittämään monipuolisesti ohjelmistoalan konsultointitehtäviä. Samaan aikaan haircut.fi palvelusta alettiin kehittää uutta monipuolisempaa versiota, joka valmistui nimellä aika24.fi. Vuodesta 2008 alkaen Smart Time Oy on toiminut kansainvälisen ohjelmistojätti Epicor Software Corpirationin Channel Partner -yhteistyökumppanina. Yhteistyö laajensi yrityksen toimintaa koskemaan Epicorin ohjelmistotuotteiden jälleenmyyntiä, konsultointia, koulutusta ja ylläpitoa. Toiminnan laajeneminen mahdollisti uuden henkilöstön rekrytoinnin. Aloitin urani yrityksen alaisuudessa samana vuonna ja tutustuin ensimmäistä kertaa E4SE ERP-järjestelmään.

Epicor Software Corporation on kansainvälinen toimija ohjelmistoalalla. Sen toimialaan kuuluvat erilaiset suurten ja keskisuurten yritysten liiketoimintaohjelmistot mm. tuotannon- ja toiminnanohjausjärjestelmät, taloushallinto-, jakelu- ja palvelualakohtaiset ohjelmistot. Yritys on perustettu Yhdysvalloissa vuonna 1984 nimellä Platinum Software Corporation. Se on sittemmin laajentanut toimintaansa kehittämällä omia tuotteita ja suorittamalla onnistuneita yritysostoja. Nykyisin se toimii yli 140 maassa, palvelee kymmeniätuhansia asiakkaita ympäri maailman ja työllistää tuhansia työnte-

kijöitä. Suomessa Epicor toimii nimellä Epicor Software Finland. Sen tunnetuimpia tuotteita Suomessa ovat mm. E4SE ja iScala -toiminnanohjausjärjestelmät.

6.2 Epicor For Service Enterprises - E4SE

Epicor For Service Enterprises eli E4SE on erityisesti palvelualalle suunniteltu ja toteutettu toiminnanohjausjärjestelmä. E4SE soveltuu hyvin projektilähtöisen liiketoiminnan tarpeisiin. Vaikka E4SE voidaan nähdä PSA-järjestelmänä, se on täysimittainen ERP. Sen sisältämät osajärjestelmät kattavat mm. asiakkuuden-, toimitusketjun-, toimittajasuhteen-, projektin-, suorituskyvyn-, koulutuksen- ja liiketoimintaprosessienhallinnan lisäksi myös taloushallinnon, raportoinnin ja yritysportaalin. (ERP Enterprise Solutions.) Se on mahdollista integroida valmiilla komponenteilla mm. Microsoft Project -projektinhallintaohjelmistoon, Microsoft Exchange -sähköposti palvelimeen ja Epicor iScala ERP-järjestelmään. Lisäksi se integroituu olemassa olevaan Active Directory -hakemistopalveluun hyödyntäen sitä mm. käyttäjien tunnistamiseen.

E4SE on web-pohjainen Windows-alustalle tarkoitettu asiakas-palvelin-järjestelmä. Sen pohjana on Epicor ICE -arkkitehtuuri, joka perustuu Microsoft .NET -sovelluskehykseen ja SOA-arkkitehtuurimalliin. Epicorin (Epicor ICE Business Architecture) mukaan Epicor ICE -arkkitehtuuri erottaa liiketoimintasovellukset ja teknologiakehyksen toisistaan. Sen päätarkoituksena on tarjota uuden sukupolven tietojärjestelmille yhtenäinen ja joustava perusta. Epicor ICE -kokonaisarkkitehtuurin muita hyötyjä ovat selkeät integrointirajapinnat, järjestelmän joustavuus, hyvä skaalautuvuus, tietoturvallisuus ja parempi käyttökokemus. (Epicor ICE Business Architecture.) Toteutukseltaan E4SE perustuu SOA-malliin, jota Epicor kutsuu nimellä Epicor True SOA. Arkkitehtuurimallin mukaisesti sisäiset ja ulkoiset liittymät on toteutettu pääsääntöisesti *web services* -tekniikalla. Tiedonsiirto tapahtuu järjestelmän määrityksien mukaisilla XML SOAP sanomilla.

Epicor tarjoaa työkaluja sovelluksien ja businesslogiikan muokkaamiseen monella eri tasolla. Järjestelmään on mahdollista luoda uusia näkymiä, muokata olemassa olevaa toiminnallisuutta, toteuttaa täysin uutta logiikkaa tai kutsua sen verkkopalveluja E4SE:hen integroiduista ulkopuolisista sovelluksista.

6.3 Lähtötilanne ja ongelma

Opinnäytetyöni lähtökohtana toiminutta ongelmaa on helpoin lähestyä lyhyen esimerkin avulla. Oletetaan, että organisaatio XYZ on projektityötä tekevä konsulttiyritys. Sen konsultit viettävät työaikansa yleensä asiakkaiden tiloissa tai matkustaessa. Tarvetta työskentelyyn oman yrityksen toimistossa ei välttämättä ole ja organisaatio suosii etätyötä päivinä, jolloin ei ole tarvetta vierailla asiakkaalla. Konsultit käyvät toimistollaan sattumanvaraisesti kerran viikossa tai harvemmin. Asiakasprojekteja voi olla menossa useita, joten matka-, tarvike- ja edustuslaskuja kertyy viikon mittaan paljon. Olisi tarpeen kirjata kulut ja tunnit järjestelmään mahdollisimman pian, jotta korvaukset eivät viivästyisi ja yrityksen hallintajärjestelmä pysyisi ajan tasalla. Konsultti joutuu pitämään kirjaa kuluista ja tunneista muistikirjassa, Excel tiedostoissa, muistilapuilla tai vaikkapa sähköpostiohjelman kalenterissa kunnes käy yrityksensä toimistolla kirjaamassa merkinnät järjestelmään. Aina ei ehdi tai muista kirjoittaa kaikkia yksityiskohtia ylös, muistilaput voivat olla hukassa tai Excel tiedosto korvautunut väärällä versiolla. On myös työlästä kirjoittaa merkinnät yksityiskohtaisesti ensin toiseen ohjelmaan ja siirtää ne manuaalisesti lopulliseen järjestelmään. Konsultti voi kuitenkin seurata sähköpostejaan ja kontaktejaan suoraan uudesta älypuhelimestaan. Miksi puhelimella ei voisi myös suorittaa kirjauksia tai kirjoittaa ne muistiin ohjelmaan, joka siirtäisi ne järjestelmään automaattisesti? Kysymystä on pohdittu myös Smart Timella. Ennen projektin aloitusta oli tullut esille tarve joustavampaan tapaan suorittaa tunti- ja kulukirjauksia. Ongelma oli selkeä ja koskettaa varmasti monia organisaatioita.

Usein toiminnanohjausjärjestelmien ja muiden suurten tietojärjestelmien tapauksessa erilaisten kirjausten tekeminen on kankea ja jäykkä prosessi. Se täytyy suorittaa määrätyillä työasemilla, tietyillä tunnuksilla ja yrityksen sisäverkosta käsin. Ratkaisuksi on yleensä tarjottu mm. VPN-yhteyksiä, joilla vaikkapa kotitoimiston saa helposti liitettyä osaksi yritysverkkoa. Jotkin järjestelmät tarjoavat myös mahdollisuuden kirjautua julkisen Internet-verkon kautta selaimella, mutta hyvin usein selaintuki on rajoitettu vain tietyn toimittajan ohjelmistoihin.

Edellä mainitun kaltaiset keinot eivät kuitenkaan ratkaise ongelmaa kuin osittain. Ne voivat jopa asettaa uusia rajoituksia ja vaatimuksia ratkaistujen ongelmien tilalle. VPN vaatii kiinteän alku- ja päätepisteen, joten työntekijän sijainti ei voi vaihdella ja

paikkasidonnaisuus säilyy. Kannettavaa tietokonetta voi toki käyttää missä tahansa, mutta voi olla turhan työlästä kaivaa kannettavaansa laukusta esimerkiksi odottaessa tapaamista toimiston aulassa, matkustaessa taksissa tai istuessa paikallisjunassa. Jos VPN-ratkaisu lisäksi vaatii laitepohjaisia ratkaisuja, vaihtoehto ei vaikuta kovinkaan houkuttelevalta liikkuvan työntekijän kannalta. On myös olemassa puhtaasti ohjelmistopohjaisia VPN-ratkaisuja, joilla laitevaatimuksia ei ole, mutta toteutuksen määrää käytännössä yrityksen tietoturvapolitiikka sekä muut käytänteet. Yleistettynä vaatimuksena on jokin kannettava laite, jolla verkkoyhteys luodaan. Kaikki yritykset eivät tarjoa työntekijöidensä käyttöön kannettavia tietokoneita tai salli omien laitteiden käyttöä. Näissä tapauksissa on etsittävä vaihtoehtoja muualta.

Selaimella tapahtuvat kirjaukset ovat epäkäytännöllinen ratkaisu, koska suurin osa järjestelmistä on yhteensopivia vain tietyllä selaimella. Esimerkiksi E4SE:n tapauksessa vaaditaan Microsoft Internet Explorer -selain. Ainoastaan Windows Mobile -älypuhelimet tarjoavat mahdollisuuden sen käyttöön. Matkapuhelimien selaimet ovat toiminnoiltaan ja teknisiltä ominaisuuksiltaan huomattavasti vaatimattomampia kuin tietokoneelle saatavat versiot. Toimintojen ja suorituskyvyn puutteet voivat muodostaa ongelmia. Mobiililaitteen pieni näyttö on epäkäytännöllinen tietokoneen ruudulle suunniteltujen käyttöliittymien tarkasteluun ja tehokkaaseen käyttöön. Yleensä laitteiden suorituskyky on huomattavasti heikompi kuin perinteisen työaseman tai kannettavan tietokoneen. Tästä johtuen on myös mahdollista, että järjestelmä ei ole käytettävissä heikkotehoisella älypuhelimella. E4SE:n tapauksessa lisäesteitä muodostuu järjestelmän vaatimuksesta, että työaseman on oltava osa Windows Domain -verkkoa ja sille on kirjauduttava henkilökohtaisilla tunnuksilla. Ongelma on mahdollista osittain kiertää lisämoduulin avulla, joka mahdollistaa web-pohjaisella lomakkeella kirjautumisen.

Kirjauksien suorittamisen ongelma korostuu tapauksissa, joissa työntekijä on huomattavan osan ajasta muualla, kuin oman yrityksensä tiloissa tai yhteydessä yrityksen sisäverkkoon. Monissa ammateissa joutuu matkustamaan merkittävästi, työtehtävät vaativat työskentelyä asiakkaan tiloissa tai työtahti on muista syistä hektinen. Vastaavasti useasti työpäivän aikana kertyy niin sanottuja hukkaminuutteja mm. matkustaessa tai odottaessa muusta syystä. Lähtökohtana projektille oli ajatus, voisiko tätä aikaa hyödyntää tehokkaasti ja samalla ehkäistä kirjausten suorittamisen viivästymistä. Edellä mainitun kaltaiset ongelmat poistuisivat, jos työntekijä voisi suorittaa kirjauk-

set missä tahansa, itse määrittelemässään tahdissa ja riippumatta siitä onko sillä hetkellä yrityksen verkossa tai verkoyhteydessä ollenkaan. Käytännössä jokaisella työntekijällä on käytössään jatkuvasti Internet-yhteydessä oleva älypuhelin, jonka suorituskyky riittäisi kevyen sovelluksen käyttöön.

6.4 Tavoitteet

Projektin tavoitteena on kehittää joustava ja helppo tapa suorittaa oleellisimmat kirjaukset E4SE järjestelmään. Mainitut kirjaukset käsittävät tunti- ja kulukirjaukset. Alustavassa esitutkimuksessa pohdimme Smart Timen projektipäällikön ja teknisen johtajan kanssa miten kirjauksista saisi joustavan ja miellyttävän toimenpiteen, mitä ratkaisulta vaaditaan sekä vaihtoehtoja toteutukselle. Haikalan ja Märijärven (2006, 37) mukaan esitutkimuksen tehtäviin kuuluu yleisten järjestelmätason vaatimusten kerääminen ja analysointi. Analyysi vastaa kysymykseen miksi ratkaisu tulisi kehittää ja onko se mielekästä.

Pohdintojemme perusteella päädyimme toteuttamaan ratkaisun älypuhelimille suunnattuna sovelluksena. Se mahdollistaisi ongelman joustavan ratkaisun ilman uusien järjestelmätason lisävaatimusten syntymistä. Mobiilisovelluksella on kuitenkin omat vaatimuksensa, jotka vaativat lisäselvitystä ja analysointia ennen toteutusvaihetta. Tällä hetkellä E4SE järjestelmään ei ole saatavilla alustariippumatonta asiakasohjelmaa älypuhelimille. Järjestelmän toimittaja tarjoaa vastaavien toimintojen sisältävää sovellusta Microsoft Windows CE tai Palm OS käyttöjärjestelmää käyttäviin laitteisiin. Ratkaisu on perustelu, koska Epicor on Yhdysvaltalainen ohjelmistotoimittaja ja edellä mainittujen käyttöjärjestelmien markkinaosuus on Pohjois-Amerikassa huomattavasti suurempi kuin Euroopassa, jossa Nokian Symbian OS dominoi älypuhelimien käyttöjärjestelmämarkkinoita. Pohdimme mahdollisuutta toteuttaa sovellus, joka toimisi suurimmassa osassa mobiililaitteita riippumatta käyttöjärjestelmästä. Java ME -ympäristö osoittautui vahvimmaksi vaihtoehdoksi juuri järjestelmäriippumattomuutensa vuoksi. Java on myös tunnettu ja ominaisuuksiltaan rikas sovellusalusta. Javateknologian kehitys, tuki ja tulevaisuus vaikuttavat hyviltä, koska sen taustalta löytyy luotettava ja pitkäikäinen ohjelmistoyritys.

Myöhemmin tavoitteena on tarjota sovellusta myös olemassa oleville E4SE asiakkaille Suomessa ja projektin onnistuessa laajemmassa mittakaavassakin. Sovelluksen tuli-

si siis olla toiminnaltaan virheetön, helppokäyttöinen, laajennettavissa ja tietysti myyvä. Smart Time Oy voi ottaa sovelluksen ensimmäisen version testikäyttöön ja kehittää siitä valmiin tuotteen. Opinnäytetyöni käsittää projektin ja sovelluksen analyysin, esitutkimuksen, määrittelyn ja suunnittelun. Ohjelmistoprojekti sisältää lisäksi testikäyttöön tulevan version toteutuksen, testauksen ja dokumentoinnin sekä tarpeelliset välineet jatkokehitykselle. Toteutus-, testaus- ja käyttöönottovaiheet sijoittuvat ajallisesti varsinaisen opinnäytetyöni ulkopuolelle, mutta kuuluvat projektin kannalta saumattomasti yhteen.

6.5 Projektin suunnittelu ja hallinta

Projektinhallinta toteutettiin luomalla projektisuunnitelma, pitämällä tarvittaessa kokouksia ja katsauksia sekä viikoittaisella seurannalla. Viikkokatsaukset sisälsivät projektin etenemisen analysoinnin edelliseltä viikolta, seuraavan viikon tavoitteiden asettamisen, mahdollisten ongelmien ja muutoksien läpikäynnin. Tarkoituksena on kehittää projektisuunnitelmaa ja muita hallintadokumentteja iteratiivisesti tarpeen vaatiessa. Projektinhallinta oli jatkuva prosessi, jota suoritettiin koko projektin keston ajan. Hallinnalla pyrin ylläpitämään ajantasaista ja realistista näkemystä projektin tilasta ja suunnitelmien paikkansapitävyydestä. Sen avulla pyrin minimoimaan riskejä ja yllätyksiä projektin toteutuksessa. Opinnäytetyötä varten luomani projektisuunnitelma sisältää mm. organisoinnin ja ohjauksen, teknologian ja toimintatapojen määrittelyn sekä aikataulutuksen ja kustannusarvion. Edellä mainittujen päälukujen lisäksi sisältö on jaoteltu tarkemmin pienempiin osa-alueisiin. Projektisuunnitelman tarkempi sisältö on esitetty opinnäytetyön liitteenä (liite 1) ja sen johdantoon ja projektin tavoitteisiin voi tutustua toisessa liitteessä (liite 2).

Projektinhallinnan dokumentointi tapahtui ylläpitämällä projektisuunnitelma dokumenttia sekä projektisivustoa Smart Time Oy:n intranetissä. Projektidokumentit julkaistiin yrityksen henkilöstön käyttöön verkkojaolla. Intranetissä toimiva projektisivusto toteutettiin MediaWiki ohjelmiston päälle muokattavana wiki-sivustona. Kaikki projektidokumentaatio toteutettiin Smart Time Oyn: käytänteiden mukaisesti englannin kielellä. Opinnäytetyön suorittajana olin vastuussa projektin hallinasta yrityksen projektipäällikön ohjauksella.

Jotta ohjelmistoprojektia pystytään seuramaan ja hallitsemaan mielekkäästi, täytyy se osittaa. WBS-menetelmällä saadaan hierarkkinen kuvaus erilaisista aktiviteeteistä ja etapeista (Haikala & Märijärvi 2006, 230). Suoritin projektin suunnitteluvaiheessa alustavan osittamisen ja tarkensin sitä määrittelyvaiheen edetessä. Projekti jaettiin viiteen osaprojektiin:

I. *Project Setup*
II. *Definition and Analysis*
III. *Build*
IV. *Finalization*
V. *Control Project*

Näistä osista neljä ensimmäistä ovat selviä työvaiheita, jotka sisältävät omat aktiviteettinsa. Jokaisen valmistuminen myös muodostaa etapin projektin etenemisessä. Viides vaihe eli projektinhallinta on liitetty omaksi osakseen selkeyden vuoksi ja havainnollistamaan sille varattua työmäärää. Perinteisessä yksisuuntaisessa projektimallissa, kuten vesiputousmallissa, työ etenisi systemaattisesti vaiheesta toiseen, kun edellinen etappi on saavutettu. Sovelsin työssäni dynaamisempaa ketterän kehityksen mallia ja osia muista iteratiivista mallista. Näin ollen eri vaiheita voitiin suorittaa iteratiivisesti ja tarkentaa tarpeen vaatiessa. Edelliseen vaiheeseen voitiin myös palata myöhemmin luomalla siitä uusi iteraatio ja sen valmistumisesta uusi etappi.

Jos kyseessä olisi normaali ohjelmistoprojekti, resurssi- ja vastuusuunnittelulla olisi huomattavasti suurempi merkitys. Projektin ollessa opinnäytetyö, vastuu ja tehtävät olivat pääasiallisesti omilla harteillani. Suunnitelmassa määrittelin erilaisia rooleja soveltuvin osin. Yrityksen muita resursseja osallistui lähinnä testaukseen ja projektin ohjaukseen. Työn ohjauksesta vastaava tekninen johtaja osallistui kuitenkin myös määrittely- ja suunnitteluvaiheeseen neuvonantajan roolissa. Muiden resurssien osallistumisprosentin määritteli lähinnä heidän oma aikataulunsa sekä akuutti tarve projektin onnistumisen kannalta.

Riskienhallinta on toteutettu ennakoivalla seurannalla, riittävällä kommunikaatiolla sekä sovelletulla ketterän kehityksen mallilla. Ennakoiva seuranta perustui viikkokatsauksiin, joissa analysoin esiin nousseita ongelmia ja arvioin uuden tiedon valossa tulevia aktiviteettejä. Pitämällä tarkastusten ja katsauksien aikavälin lyhyenä ongelmat ehditään havaita ja niihin pystytään reagoimaan ennen kuin ne muodostavat varteenotettavaa riskiä projektille. Jos katsauksissa havaittiin riskitekijä, johon ei ole varauduttu tai joka on huomioitava tai arvioitava uudelleen voitiin järjestää palaveri, johon

osallistui tarpeen mukaan yrityksen henkilöstöä riippuen riskin vakavuudesta ja osa-alueesta. Soveltamalla joustavia ohjelmistokehityksen malleja, voitiin riskienhallinta jakaa iteratiiviseksi prosessiksi kuten muukin projektinhallinta. Vaatimuksena oli, että perustavanlaatuiset, vakavat ja todennäköiset riskit oli kartoitettu ja analysoitu. Suorittaessani riskianalyysia pyrin minimoimaan niiden vaikutukset jo suunnitelmaa tehdessäni.

Projektin vaatimusten- ja muutostenhallinta on syytä toteuttaa asianmukaisesti, koska niiden vaikutus projektin onnistumiseen on merkittävä. Vaatimukset määrittelevät projektin tavoitteet, toteutuksen ja aikataulun. Muutokset määrittelyvaiheessa tehtyihin vaatimuksiin ovat tavallisia ja niiden hallinta on oltava tehokasta ja dokumentoitua, jotta kokonaisuus säilyy mielekkäänä ja uusia riskejä ei pääse syntymään. Vaatimustenhallinta kattaa kaikkien toimenpiteiden seurannan ja varmistuksen asiakasvaatimuksista valmiiseen ohjelmistoon. (Haikala & Märijärvi 2006, 91.) Tässä opinnäytetyössä vaatimusten- ja muutostenhallinta oli integroitu osaksi projektisuunnitelmaa ja hallintaprojektia. Muutokset kirjattiin ja esitettiin sitä mukaa, kun niitä nousi esiin. Ne arvioitiin ja käsiteltiin viikkokatsauksien yhteydessä, ellei ollut tarvetta välittömiin toimenpiteisiin. Viikoittain suoritetussa käsittelyssä päätettiin, mitkä muutoksista otettiin projektiin mukaan ja miten ne vaikuttivat vaatimuksiin, aikatauluun tai muihin projektin kannalta oleellisiin seikkoihin. Kaikki muutokset verifioitiin projektin laajuusmäärittelyä vasten, jotta kokonaistyömäärä ei ylitä odotuksia tai aiheuta aikataulun tarpeetonta venymistä. Iteratiivisen projektimallin mukaisesti määrittelyitä ja spesifikaatioita voitiin muuttaa missä vaiheessa projektia tahansa, jos muutos tapahtuu hallitusti.

Projektin toteutuksessa sovelletaan ketterän kehityksen mallia sekä joukkoa muita dynaamisia menetelmiä. Valinnan avulla pyrin minimoimaan jäykkää ennalta määrättyä toteutusta, suunnitelman joustamattomuutta ja pienentämään riskejä ja muutostoimenpiteiden vaikutuksia. Koska ohjelmistoprojekti ei suunnitellusti pääty opinnäytetyön puitteissa, iteraatioita voidaan jatkaa suunnitelman mukaisesti tarkentamalla ja laajentamalla määrittelyjä. Ketterän kehityksen mallien metodiikasta sovellan mm. lyhyen aikajänteen iteraatioita, maksimoitua tehtävien osittamista, toimituskelpoisuuteen keskittyvää toteutusvaihetta, koodin tai muun tuotoksen säännöllistä arviointia ja uudelleenjäsentelyä, muutostietoisuutta, KISS-kehitysfilosofiaa, tehokasta yksikkötestausta ja tarvittaessa testauslähtöistä toteutusta. Ohjelmoinnin osalta painotan kyseisel-

le kielelle ominaisia standardeja, selkeää ulkoasua ja koodin hyvää laatua. Dokumentaatio toteutetaan luonnollisella kielellä, informatiivisesti ja sitä voidaan havainnollistaa tarvittaessa erilaisilla UML standardin mukaisilla kaavioilla.

6.6 Määrittely

Projektin määrittelyvaihe on osa vaatimustenhallinta toimintoa. Vaatimustenhallinnan ja sen osa-alueiden tavoitteena on tuottaa asiakasvaatimuksia vastaava lopputulos. Se voidaan toteuttaa joko tukitoimintona, rinnakkaisena aliprojektina tai sulauttaa soveltuviin vaiheisiin, kuten määrittelyyn, ohjelmistoprojektissa. (Haikala & Märijärvi 2006, 91-94.) Opinnäytetyöni tapauksessa vaatimustenhallinta on osa hallintaprojektia sekä määrittelyvaihetta. Järjestelyn tavoitteena on vähentää työmäärää ja byrokratiaa. Projektin tavoitteena ei ole tuottaa valmista myyntikelpoista sovellusta, vaan sisäiseen käyttöön tarkoitettu sovellus, jossa on riittävät ominaisuudet jatkokehitystä ajatellen. Määrittelyä voidaan myöhemmin jatkokehitysprojektin vaatiessa muuttaa ja parannella.

Suoritin projektin vaatimusmäärittelyn yhteistyössä yrityksen projektipäällikön ja teknisen johtajan kanssa. Analysoimme, millaisia vaatimuksia sovelluksella on käyttäjätoimintojen, järjestelmätoiminnallisuuksien ja mielekkyyden kannalta. Arvioimme myös ehtoja, jotka toteutuksen on täytettävä, jotta se olisi toteutuskelpoinen. Lopuksi selvitimme millaisia ehtoja määritelty ratkaisu asettaa, jotta se olisi toteuttamiskelpoinen. Lopputuloksena syntyneeseen vaatimusmäärittelydokumenttiin kirjattiin eriteltyinä osa-alueina yleiset määritykset sekä toiminnalliset ja ei-toiminnalliset vaatimukset. Dokumentti käsittää sekä asiakas- että järjestelmävaatimukset, jotta kokonaisuus olisi selkeästi hahmotettavissa. Yleiset määritykset sisältävät mm. ympäristön ja käyttäjäryhmien määrittelyn, toteutuksen reunaehdot sekä rajoitukset ja olettamukset, joiden suhteen vaatimukset on laadittu. Niiden tarkoitus on määritellä riittävällä tarkkuudella se pohja ja perustelut, joiden varaan toiminnalliset ja ei-toiminnalliset vaatimukset rakentuvat. Toiminnalliset vaatimukset käsittävät joukon käyttäjän suorasti tai epäsuorasti suoritettavissa olevia toimintoja, jotka ohjelman on tarjottava vaaditulla tavalla. Toiminnot on arvioitu toteutusprioriteetin ja ohjelmiston kohdeversion mukaisesti. Näin ollen dokumentti tarjoaa kartan, missä järjestyksessä vaatimukset on toteuttava. Ei-toiminnalliset vaatimukset ovat ohjelmiston ominaisuuksia, jotka eivät suoraan liity suoritettaviin toimintoihin, mutta jotka on toteutettava, jotta ohjelma toimisi halutulla

tavalla ja täyttäisi esitutkimuksessa ilmenneet asiakasvaatimukset. Esimerkki ei-toiminnallisesta vaatimuksesta on sovelluksen käyttöliittymän vasteaikavaatimus päätelaitteella.

Analysoin ja määrittelin sovelluksen toiminnallisia vaatimuksia tarkemmin luomalla käyttötapauksia ja lyhyitä sanallisia kuvauksia toiminnosta. Esimerkki dokumentoidusta käyttötapauksessa on esitetty liitteenä (liite 3). Käyttötapaus kuvaa järjestelmän yksittäisen toiminnon perättäisinä käyttäjän suorittamina tapahtumina. Käyttötapaukset ovat erityisen hyödyllisiä määritellessä asiakasvaatimuksia. (Haikala & Märijärvi 2006, 158.) Tapaukselle voidaan tarvittaessa määritellä erilaisia vaatimuksia ja ehtoja, jotka on täytettävä ennen kuin käyttötapaus voidaan suorittaa. Monimutkainen tai vaikeasti lähestyttävissä oleva toiminto on mahdollista pilkkoa pieniin osatapahtumiin, joista kokonaisuus muodostuu. Käyttötapauksien perusteella voidaan määritellä erilaisia testitapauksia, joilla suoritetaan sovelluksen testausta toteutusvaiheen aikana. Kirjoitin sovelluksen testaussuunnitelman suorittaessani määrittelyvaiheen toimenpiteitä. Sisällytin siihen asiakas- ja järjestelmävaatimuksia vastaavien toimintojen testauksen sekä niille jo selvillä olevat ehdot ja rajoitukset. Täydensin myös erilaisia laatuvaatimuksia ja määrittelin monimutkaisille ja kriittisille ominaisuuksille joukon testitapauksia, jotka on testattava ja läpäistävä ennen kuin kyseinen moduuli voidaan hyväksyä sovellukseen. Testaussuunnitelma käsittelee testitapauksien lisäksi testausmenetelmiä, testauksen hallintaa ja laajuutta, laadullisia ominaisuuksia ja hyväksymistestausta.

Sisällytin projektin määrittelyprosessiin myös teknisen esitutkimuksen E4SE:n liittymistä ja relevanteista toiminnallisuuksista. Niiden vaikutus järjestelmämäärittelyyn, koskien erityisesti olettamuksia ja rajoituksia, on huomioitava alusta lähtien, koska toteutettavan ohjelmiston on noudatettava E4SE:n toimintatapoja ja sen asettamia vaatimuksia. On huomattavasti helpompaa määritellä uuden sovelluksen toiminta yhteensopivaksi jo olemassa olevan ja merkittävästi laajemman kokonaisuuden kanssa kuin toisinpäin. Lähtökohtana määrittelystä alkaen on, että E4SE järjestelmään ei tehdä mitään muutoksia, vaan kehitettävä ohjelma sopeutuu siihen.

E4SE kommunikoi ulkoisten järjestelmien kanssa *web service* -rajapintojen kautta. Tunnistautuminen E4SE-järjestelmään tapahtuu joko suoraan Active Directory integraation kautta Windows autentikoinnilla tai *web service* -sanomaan upotettuina para-

metreina. Kehitettävän sovelluksen palvelinpään on kyettävä vastaanottamaan ja lähettämään E4SE:n määrityksiä noudattavia *web service* -sanomia ja tunnistauduttava oikein, jotta integraatio olisi mahdollinen. Määriteltyjen toiminnallisuuksien mukaiset sanomat on sovelluksen suunnittelua varten listattu ja analysoitu. Sanomien tarvitsema data kerätään joko mobiilisovelluksessa käyttöliittymän kautta tai generoidaan automaattisesti palvelinpäässä, mikäli tieto on luonteeltaan staattista.

6.7 Suunnittelu

Sovelluksen suunnittelu tapahtui projektisuunnitelman mukaisesti kahdella periaatteella. Toteutin järjestelmätason arkkitehtuurisuunnittelun sekä alustavan moduuli- ja rajapintasuunnittelun määrittelyprosessin yhteydessä. Tarkempi olio-, tietorakenne-, käyttöliittymä- ja rajapintasuunnittelu tapahtuu toteutusvaiheen yhteydessä ketterän kehityksen ja prototyyppimallin periaatteiden mukaisesti. Tällä jaolla pyrin keventämään suunnitteluprosessia ja lisäämään sen joustavuutta. Kokosin suunnittelussa syntyneen informaation *Solution Design* -dokumentiksi. Projektin vaihejako on luonteeltaan organisointia ja hallintaa helpottava. Sen tarkoitus ei ole määritellä kiinteästi vaiheiden välisiä eroja tai rajoittaa niiden suoritusta. Pyrin muodostamaan suunnittelusta dynaamisen prosessin, joka jatkuu koko toteutusvaiheen ajan muutostenhallinnassa määritellyillä ehdoilla.

Sovelluksen tuotekehitys on jaettu eri versioihin. Tärkeimmät versiot ovat 1.0 ja 2.0 sekä niiden testauskelpoiset väliversiot 0.5 sekä 1.5. Opinnäytetyöni tavoite on suunnitella toteutuskelpoinen 1.0 julkaisuversio. Projektisuunnitelmassa arvioin, että sen toteutusvaihe olisi valmis vuoden 2010 loppuun mennessä. Sovelluksen suunnittelu kattaa pääasiallisesti vain version 1.0 toteuttamiseen tähtäävät osa-alueet, mutta pyrin ottamaan huomioon tulevaisuuden tarpeet, laajennettavuuden ja hyvät suunnitteluperiaatteet, jotta tulevien versioiden kehittäminen olisi mahdollisimman helppoa. Versio 1.0 tarkoittaa sovelluksen versiota, jossa on toteutettu vain *online*-toiminallisuudet. Käytännössä se tarkoittaa, että MID-laitteen oletetaan olevan jatkuvassa yhteydessä sovelluspalvelimeen ja mitään konfliktitilannetta synkronoinnissa ei pääse syntymään. Ennen myyntikelpoisuutta sovellus on kehitettävä versioon 2.0, joka sisältää myös *offiline*-toiminnallisuudet ja paremman virheensietokyvyn.

Arkkitehtuurisuunnittelun tehtävänä oli tarjota riittävä perusta sovelluksen toteutuksenvaiheen aloittamiseksi. Lisäksi se määritteli teknisellä tasolla tarvittavat osajärjestelmät ja suunniteltavat kokonaisuudet. Tavoitteenani oli luoda riittävän kattava kuvaus sovellusarkkitehtuurista ja teknisestä kokonaisratkaisusta, jotta sitä ei olisi tarpeen muuttaa yksityiskohtien tai yksittäisten toiminnallisuuksien vuoksi. Tuleva sovellus pitää sisällään kaksi erillistä komponenttia: mobiilisovelluksen sekä sovelluspalvelimen. Dokumentaatiossa viittaan niihin väliaikaisilla nimillä *Mobile Client* ja *Communications Server*. Valitsin ratkaisun toteutuskieleksi Javan, koska se kattaa sekä mobiiliohjelmoinnin Java ME -alustalla että palvelinohjelmoinnin Java EE -alustalla. Se on myös riippumaton MID-laitteen tai palvelimen käyttöjärjestelmästä, helposti siirrettävissä ja nopeasti laajennettavissa.

Mobiilisovelluksen tehtävänä on tarjota käyttäjärajapinta kirjausten käsittelyyn ja kaikki käyttäjälle näkyvä toiminnallisuus. Mobiilisovellus kommunikoi *web services* -tekniikalla sovelluspalvelimen kanssa. Sovelluspalvelin huolehtii sanomien reitityksestä MID-laitteen ja E4SE-palvelimen välillä. Se huolehtii myös käyttäjän tunnistamisesta ja tarvittavien tietojen synkronoinnista päätepisteiden välillä. Lopullisessa versiossa sovelluspalvelin sisältää myös hallinta- ja lokitoimintoja. Suunnittelin tässä vaiheessa myös sovelluspalvelimen kokoonpanon ja tarpeelliset kolmannen osapuolen sovellusvaatimukset. Sovelluspalvelin sijoitetaan DMZ-verkkoon julkisen Internet-verkon ja organisaation sisäverkon väliin. Tällöin kommunikointi tapahtuu turvallisesti ja organisaation tietoturvallisuusperiaatteiden mukaisesti. Kehitysprojektia varten käyttöön otettiin virtuaalipalvelin.

Alustava järjestelmäsuunnittelu käsitti myös tarpeellisten sanomien rakenteen teknisen analysoinnin ja niihin liittyvän pakollisen toiminnan suunnittelun. Määrittelyvaiheessa kerätty tieto E4SE:n rajapinnoista toimi syötteenä rajapintojen ja tietorakenteiden suunnittelussa. Niiden tarkempi dokumentointi muodostuu kuitenkin vasta erilaisten prototyyppien testaamisen jälkeen. Voi olla mahdollista, että E4SE:n käyttämät sanomarakenteet ovat liian raskaita käsiteltäviksi suoraan mobiililaitteella. XML-tiedostojen koko voi myös osoittautua pullonkaulaksi mobiililaitteen tiedonsiirtokapasiteetille. Käytännössä se, muodostavatko edellä mainitut seikat ongelmaa, selviää parhaiten prototyyppien ja suorituskykytestauksen avulla. Sen vuoksi en nähnyt mielekkääksi lyödä lukkoon rajapinnan lopullista toteutusta vielä alustavassa suunnitteluvaiheessa. Mikäli sanomat ovat liian raskaita, ne voidaan muuntaa sovelluspalveli-

messa kevyempään muotoon XSLT-muunnoksen avulla. Tällöin mobiilisovelluksen ja sovelluspalvelimen välillä voidaan kommunikoida vain tarpeellisen tiedon sisältävien XML SOAP -sanomien välityksellä.

Projektin toteutusvaiheen edetessä täydennän suunnitteludokumentaatiota perustuen prototyyppien ja suorituskykytestien lopputuloksiin. Pyrin soveltamaan hyväksi todettuja suunnittelumalleja, kommunikoimaan ohjausryhmän kanssa ja vastaamaan muutoksiin nopeasti. Tavoitteenani on samalla tutkia kuinka joustavaa kehitystä ja suunnittelua voidaan soveltaa ohjelmistokehityksessä. Vaikka etukäteen suoritetun suunnittelun määrä on huomattavasti pienempi verrattuna suunnitelmapainotteisiin malleihin, kuten vesiputousmalliin, se ei tarkoita suunnitelmallisuuden poissaoloa tai *cowboy coding* -lähestymistapaa. Toisaalta pyrin minimoimaan ketterään kehitykseen liitettyjä riskejä toteuttamalla alustavan arkkitehtuuri- ja järjestelmäsuunnittelun, mutta jättämällä siihen reilusti liikkumavaraa optimaalisen toteutuksen saavuttamiseksi.

Toteutusvaiheen avuksi tuotettu *Solution Design* -dokumentti piti sisällään yleiskatsauksen toteutettavasta sovelluksesta ja osajärjestelmistä, arkkitehtuurisuunnittelun, versionhallintasuunnitelman, selvityksen vaadituista ohjelmistokomponenteista, alustavan rajapintasuunnittelun, alustavan tietorakennesuunnittelun ja alustavan käyttökokemussuunnittelun. Eri osien tarkkuus vaihtelee sen mukaan, miten kriittisiä ne ovat onnistuneen toteutuksen kannalta, kuinka paljon tietoa määrittely- ja esitutkimusvaihe tarjosivat ja kuinka olennaisesti suunnitelman muuttuminen vaikuttaa kehitysprojektin ominaisuuksiin.

6.8 Toteutus ja testaus

Sovelluksen toteutus- ja testausvaihe sijoittuu ajallisesti opinnäytetyöni ulkopuolelle, mutta koin mielekkääksi esitellä sen tässä yhteydessä, koska se on osa projektikokonaisuutta ja vastaan sovelluksen kehittämisestä ensimmäiseen käyttökelpoiseen versioon saakka. Versionhallintasuunnitelman mukaisesti toteutan yrityksen sisäiseen käyttöön pätevän ja jatkokehityskelpoisen tuotteen. Nimesin sen versio 1.0:ksi, koska se on projektin ensimmäinen vaihe, joka käsittää valmiin kokonaisuuden. Projektisuunnitelman kannalta toteutus on erillinen vaihe, mutta käytännössä se tapahtuu samanaikaisesti suunnittelun kanssa. Suoritin toteutusvaiheen avauksen ja osia siitä jo tehdessäni alustavaa suunnittelua.

Toteutusta varten tarvitaan kaksi erillistä ympäristöä. Kehitysympäristöksi valitsin virtualisoidun Windows 7 -työaseman, jossa suoritan sovelluksen ohjelmoinnin, testauksen ja simuloinnin. Parhaaksi vaihtoehdoksi sovelluskehitykselle osoittautui Sunin Java ME SDK 3.0, joka sisältää NetBeans-pohjaisen IDE:n, emulaattorin, CLDC-konfiguraation, MIDP-profiilin ja kaikki tarpeelliset dokumentaatiot. Valinta oli helppo, sillä paketti on Sunin virallinen kehitystyökalu, hyvin dokumentoitu, tuettu ja asennuksen jälkeen välittömästi käyttövalmis. Toinen ympäristö on kehitystä, testausta ja jatkokehitysprojektia varten tarvittava sovelluspalvelin. Palvelimen käyttöönotto ei onnistunut yhtä helposti, koska sen vaatimat ohjelmistoratkaisut oli suunniteltava erilaisten vaatimusten valossa. Projektin aloitushetkellä organisaatiolla ei ollut tarpeisiin sopivaa sisäistä virtualisointialustaa. Kirjoitushetkellä DMZ-verkkoon sijoitettu ratkaisu oli jo olemassa, mutta sovelluspalvelimen käyttöönotto viivästyi. Sovelluspalvelin ei kuitenkaan ole välttämätön kehityksen alkuvaiheissa, sillä XML-sanomia voidaan simuloida manuaalisesti muodostetuilla testisanomilla. Mobiilisovelluksesta voi tehdä varsin pitkälle vietyjä prototyyppejä rajapintoineen ennen integraation toteutusta. Myös sovelluspalvelinta on mahdollista kehittää erillään ja siirtää se lopulliselle alustalle myöhemmin. Sovelluspalvelimen alustaksi valikoitui alustavasti Windows Server ja soveltuva Java EE sovelluspalvelin, esimerkiksi JBoss tai Glassfish. Lisäksi myöhemmin lisättävää toiminallisuutta, mm. lokia, ajatellen tarvitaan reititys keskitetylle tietokantapalvelimelle tai sovelluksen omaan käyttöön tarkoitettu tietokanta. Kehitysympäristöjen lisäksi otin käyttöön versionhallintaohjelmiston ja kehitys-wikin. Versionhallintaan käytän Visual SVN nimistä Windowsille tarkoitettua Subversion toteutusta ja TortoiseSVN-asiakasohjelmaa.

Suoritan yksikkötestausta samanaikaisesti toteutuksen kanssa. Sen tavoite on eliminoida virheet välittömästi ja vähentää myöhemmän moduuli-, integraatio- ja hyväksymistestauksen kestoa. Wikipedian yhteisöllisen tiedon (Unit testing 2010) mukaan yksikkötestaus on pienin osa, jota voidaan mielekkäästi testata; tyypillisesti yksittäinen proseduuri tai metodi. Kun yksittäinen moduuli tai rajapinta on toteutettu sellaiseen vaiheeseen, että sen toimintaa on mielekästä testata, se voidaan siirtää moduulitestaukseen. Suoritan moduulitestauksen keskitetysti niin, että kaikki tietyn ajanjakson sisällä valmistuneet moduulit voidaan testata kerralla. Tällä vähennetään testauksen hajanaisuutta ja parannetaan tehokkuutta. Ennen julkaistavien testiversioiden, käytännössä 0.5 ja 1.0, viimeistelyä suoritan integraatiotestauksen testaussuunnitelman mu-

kaisesti. Sen tarkoitus on varmistaa jokaisen osan saumaton yhteispeli ja testiversion toimivuus. Kirjaan kehitysprojektin wiki-sivulle testitulokset, lukuun ottamatta yksikkötestausta. Määrittelin testaussuunnitelmassa kriteerit testien läpäisylle, kattavuudelle ja muille ominaisuuksille. Testaussuunnitelman tarkoitus on tehdä testauksesta systemaattinen ja laadunvarmistusta tukeva toimenpide.

Projektin lopputuloksien näkyvin osa on mobiilisovelluksen käyttöliittymä. Pyrin luomaan hyvännäköisen, modernin ja käytettävyydeltään hyvän rajapinnan käyttäjän ja toiminnallisuuden välille. Mielestäni harmillisen usein liiketoimintasovelluksien käyttöliittymä on vanhahtava tai jätetty liian vähäiselle huomioille sovellusta toteutettaessa. Henkilökohtainen tavoitteeni on osoittaa, että järkevällä työmäärällä saadaan toteutettua näyttävä ja relevantti käyttöliittymä. Työkaluina tavoitteeseen pyrkiessäni on löytää mahdollisimman moderni Java ME yhteensopiva käyttöliittymäkirjasto sekä hyödyntää prototyyppimallia.

Varsinaisen sovelluksen lisäksi toteutusvaihe tuottaa joukon dokumentaatioita. Kirjoitan sovelluksen kehittämisen aikana teknistä dokumentaatiota sovelluksen ylläpitoa ja jatkokehitystä ajatellen. Kun käyttöliittymä ja lopulliset toiminnot yksityiskohtineen ovat selvillä versioon 1.0 saakka, kirjoitan käyttöohjeen ja ylläpito-ohjeen. Niiden kirjoittaminen ei ole mielekästä, jos on olemassa merkittävä mahdollisuus käyttöliittymän muutoksista. Ensimmäisen organisaation sisäiseen käyttöön tulevan version kannalta ohjeistus ei ole ensiarvoisen tärkeää, mutta se tarjoaa lähtökohdan kaupallista versiota ajatellen.

Toteutus- ja testausvaiheen suoritus tapahtuu noudattaen projektisuunnitelmassa määriteltyjä menetelmiä ja laadullisia vaatimuksia. Koska ketterän kehityksen mallit on suurimmaksi osaksi tarkoitettu tiimityöskentelyyn, täytyy niitä soveltaessa huomioida menetelmän soveltuvuus tähän projektiin. Pyrin noudattamaan hyviä standardeja, tuottamaan laadukasta ja selkeää koodia, kommentoimaan kaikki lähdekoodin toiminnot asianmukaisesti ja kompaktisti sekä arvioimaan ja parantamaan tuotettua koodia tai muuta tuotosta ns. *refactoring*-menetelmällä. Noudatan ohjelmistokehityksessä alan asiantuntijoiden oppaita ja suosituksia (ks. Martin 2009).

6.9 Käyttöönotto

Viimeinen vaihe kehitysprojektin aikajanalla on sovelluksen käyttöönotto. Tässä tapauksessa se tarkoittaa yrityksen sisäistä käyttöä, joka toimii eräänlaisena palautekanavana ja väliversiona kaupallisen version kehittämiselle. Samalla sovellusta kuitenkin voidaan hyödyntää yrityksen omiin tarpeisiin.

Ennen käyttöönottoa sovelluksen on läpäistävä hyväksymistestaus. Se on kuvailtu vaatimuksineen ja läpäisyehtoineen testaussuunnitelmassa. Hyväksymistestauksen funktio on viimeistään varmentaa, että toteutettu sovellus vastaa esitutkimuksessa ilmenneitä tavoitteita ja määrityksessä kuvailtua toiminallisuutta. Hyväksymistestauksen läpäisyn jälkeen sovellus tullaan asentamaan yrityksen omaan käyttöön. Käytön aikana ilmenee varmasti parannusideoita, bugeja tai muita huomioitavia seikkoja. Ne ovat ensiarvoisen tärkeitä kaupallista versiota ajatellen ja kannattaa kerätä talteen.

Käyttöönotto on osa projektisuunnitelmassa esitettyä *Finalization*-vaihetta. Se sisältää myös tarpeellisen dokumentaation julkaisun, tuote-wikin perustamisen ja lopetusraportin. Lopetusraportissa arvioin mm. projektin onnistumista ja etenemistä sekä koostan yhteenvedon projektista kokonaisuutena. Tarvittaessa vaiheeseen voidaan liittää lyhyt käyttökoulutus tai workshop tyyppinen tilaisuus.

7 POHDINTA

7.1 Toimeksiannon arviointi ja työn merkitys

Alkuperäinen tavoite suunnitellessani toimeksiantoprojektia oli toteuttaa se kokonaisuudessaan samanaikaisesti opinnäytetyöni kanssa. Tarkempi projektisuunnittelu, ongelman analysointi ja ratkaisuvaihtoehtojen selvitys kuitenkin pakotti arvioimaan aikataulutusta uudelleen. Uuden aika-arvion mukaisesti toteutin vain määrittely- ja suunnitteluprojektin opinnäytetyöni puitteissa. Vastaavasti uusi määräaika toteutusprojektille on asetettu siten, että version 1.0 käyttöönotto tapahtuu vuoden 2010 aikana. Mielestäni uudelleen arvioitu aikataulu on huomattavasti realistisempi ja tarjoaa paremmat mahdollisuudet projektin onnistumiselle. Liian hätäinen toteutus vähentäisi helposti kokonaisvaltaista ymmärtämistä ja vaikuttaisi negatiivisesti luovuuteen. Aika-

taulun muutoksiin vaikutti merkittävästi myös se, että suoritan projektin kokonaisuudessaan työajan ulkopuolella. Vaikka tein keinotekoisen kahtia jaon määrittely- ja tuotantoprojektiin, ne kuuluvat käytännössä samaan kokonaisuuteen.

Sovelluskehitysprojektin tarkoitus on luoda sisäiseen käyttöön, pilotointiin ja jatkokehitykseen soveltuva ohjelmistoratkaisu. Näihin lähtökohtiin perustuen oli mielestäni tärkeää painottaa ratkaistavan ongelman ja käytettävän teknologian ymmärtämiseen. Pyrkimykseni ei ole tuottaa sovellusta mahdollisimman pienellä työmäärällä tai mahdollisimman nopeasti. Haluan ymmärtää teknologian mahdollisuudet ja toteuttaa ratkaisun, joka tarjoaisi muutakin, kuin pelkän ensisijaisen lopputuloksen. Painotuksen valinta on mahdollista, koska en sido siihen työaikaani tai organisaation tuotekehitysbudjettia. Otan samalla tietoisen riskin kehittäessäni tuotetta itsenäisesti, tavallaan *risk and reward* -mallisesti.

Projektin puolesta asetettuja tavoitteita olivat toteutetun E4SE-asiakasohjelman lisäksi tietotaito mobiilisovelluskehityksestä ja uudelleenkäytettävä Java ME sovellusrunko. Hyvin toteutettuna sovelluksen perusrakenteita on mahdollista hyödyntää tulevissa projekteissa joko sellaisenaan tai valikoimalla kulloiseenkin tilanteeseen parhaiten sopivat moduulit ja oliot. Sen vuoksi ratkaisun tulisi olla mahdollisimman hyvin suunniteltu ennen muuta oliokeskeisyyden ja laajennettavuuden kannalta. Lähdekooditason kommentointi on myös olennainen tekijä laajennettaessa työn merkitystä.

Lopputuloksen arviointi tässä vaiheessa kokonaisprojektia on mahdotonta. Odotukseni tavoitteiden saavuttamisesta on kuitenkin optimistinen. Vaatimuksena on projektin painotuksen muistaminen ja sen hallittu toteutus. Määrittely- ja suunnitteluprojekti sujui omasta näkökulmastani hyvin. Pääsin tavoitteisiini, toteutuksen lähtökohdat ovat selvillä ja edessä on mielenkiintoinen urakka. Onnistuminen ei kuitenkaan tarkoita ettei kehittämisen varaa ja parannettavaa olisi. Kuten monissa tapauksissa, projektinhallinta oli alkupainotteinen. Edistymisen seuranta, suunnitelman kehittäminen, kommunikointi ja projektin kokonaisvaltainen hallinta oli tehokasta ja ajantasaista. Projektin edetessä ne alkoivat lipsua ja projektin varsinainen suoritus vei aikaa tukitoiminnoilta, jotka ovat olennainen osa toimivaa projektia. Tämä näkyi konkreettisimmin projektin wiki-sivun päivityksien aikavälien kasvamisessa. Etenemistä hidasti myös oma sekä muiden projektiresurssien työkiire. Eräs projektin kahtia jakamisen selvistä hyödyistä on vaiheiden välillä tapahtuva arviointi ja hengähdystauko. Alkuvaiheesta

on mahdollista oppia ja pyrkiä parantamaan heikommin onnistuneita osa-alueita. Kriittinen, mutta positiivinen tutkiskelu antaa varmasti uusia eväitä projektin loppuun saattamista ajatellen.

Sovelluksen kehitys ei lopu projektini valmistumiseen. Sen tuottama versio 1.0 on riittävä perusratkaisu rajattuihin toimintatilanteisiin, mutta ei tarjoa kokonaisvaltaista ratkaisua. Kuitenkin tekemiini määrittelyihin ja suunnitelmiin pohjautuen jatkokehitys on todennäköisesti pienempi projekti niin ajallisesti kuin työmäärällisestikin. Moni teknologialähtöinen ongelma on ratkaistu ja vaatimukset on analysoitu. Uskon, että versio 2.0 on kilpailukykyinen mobiilisovellus, jolle löytyy markkinarako. Ensimmäisen version onnistuessa hyvin, otaksun, että riskit seuraavan version epäonnistumiselle ovat merkittävästi pienemmät.

Opinnäytetyöni tulokset tarjoavat näkemyksen tämän hetkisestä tilanteesta mobiilimaailmassa. Vaikka se on nopeasti kehittyvä tekniikan ala, tietyt perustekijät tuskin muuttuvat hetkessä. Oli mielenkiintoista huomata Javan vahva asema mobiilisovelluskehityksessä. En havainnut sille selvää haastajaa, joka pystyisi olemaan yhtä aikaa monipuolinen ja alustariippumaton. On selvää, ettei Java ole yhtä tehokas kehitysväline kuin laitekohtaiset SDK:t. Niiden heikkouksia ovat kuitenkin heikko siirrettävyys ja riippuvuus spesifistä alustasta. Mielestäni avoimuuteen ja tehokkuuteen pyrkivässä ohjelmistotuotannossa on olennaista valmistajariippumaton kehitys.

Työni aihepiiri tarjoaisi kehityskelpoisia ideoita monen opinnäytetyön tai muun selvityksen aiheiksi. Koska lähes jokainen organisaatio hyödyntää tieto- ja ohjausjärjestelmää, ne tarjoavat monipuolisia tarvelähtöisiä projekteja. Tietotekniikan osa-alueelta mielenkiintoisia ja hyödyllisiä aiheita voisivat olla mm. käyttöönotto, sovelluskehitys, integraatiot, ylläpito tai vaikkapa palvelun tarjoaminen pilvenä. Lisämausteen tuovat liiketaloudelliset, projektinhallinnalliset ja kokonaisuuksia käsittelevät teemat.

7.2 Mobiiliteknologian kehitys

Kuten jo luvussa 3.3 arvioin, mobiiliteknologian tulevaisuudesta on nähtävillä tiettyjä trendejä. Nopeasti muuttuvan alan luotettava ennustaminen on. Mielestäni selviä suuntia ovat kuitenkin käyttäjä- ja käyttöliittymäpainotteisuus, helppokäyttöisyys, keskitetyt sovelluskaupat, sovellusalustojen sekä käyttöjärjestelmien yhä suurempi

avoimuus ja yhteisöllisyys. On melkeinpä itsestään selvää, että teknologia kehittyy jatkuvasti. Uudet laitteet ovat suorituskyvyltään ja ominaisuuksiltaan vanhoja parempia, mutta kaikki kehitys ei välttämättä ole itseisarvoisesti kohti parempaa. Esimerkiksi laitteiden fyysisten mittojen jatkuva pienentyminen on aiheuttanut hankaluuksia etenkin ikääntyvälle väestölle.

Opinnäytetyöni näkökulmasta mielenkiintoisinta on pohtia mobiilin sovelluskehityksen tulevaisuutta. Avoimen lähdekoodin projektien nostaessa suosiotaan on jännittävää nähdä, millainen on kaupallisten sovellusten tulevaisuus aikana, jona monet käyttäjät ovat tottuneet saamaan sovellukset, elokuvat, musiikin ja tiedon ilmaiseksi Internetin vertaisverkoista tai muista *online*-lähteistä. Laatu ja helppokäyttöisyys ovat tekijöitä, joiden avulla on mahdollista saavuttaa merkittävä kilpailuetu. Kosketusnäytölliset laitteet tarjoavat yhä intuitiivisemman käyttöliittymän ja kenties tulevaisuudessa laitteita voisi todella käyttää ihmisen ajatusmalleihin perustuen tietämättä tekniikan tai ohjelmoijan edesottamuksista yhtään mitään.

Tiukasti sovelluskehittäjän näkökulmasta katsottuna työkaluvalikoima, niiden laatu ja alustojen sekä teknologian dokumentaatio ovat parantuneet huomattavasti. Sovelluskehitystyökaluja löytyy usein laitevalmistajan lisäksi myös monien yhteisöllisten lähteiden tai avoimen lähdekoodin projektien kautta. Emulaattorit ovat kehittyneempiä ja muistuttavat fyysisiä perikuviaan huomattavasti paremmin kuin vielä muutama vuosi sitten. Sovelluskehityksen ja kehittäjien määrällisesti lisääntyessä sosiaaliset verkostot ja Internet-foorumit tarjoavat nopean keinon ratkaista ongelmia ja jakaa tietoa. Vertailukohtana sovelluskehitysvälineistä haluan mainita kokemuksistani Symbianille ja S60-alustalle ohjelmoinnista joitakin vuosia takaperin. Silloiset kehitysympäristöt olivat jokseenkin bugisia tai osittain vasta beta-asteella. Vaikka siitä ei ole kovin monta vuotta aikaa, kehitys on ollut huima. Vasta muutamia vuosia vanhat Java ME oppikirjat kuvailivat kehitystyökaluja vanhahtaviksi ja rajoitetuiksi. Käyttäessäni nykyistä Java ME SDK 3.0 sovelluspakettia, havaitsin sen olevan kehittynyt täysverinen IDE ja miellyttävä käyttää. Tuntuu, kuin valmistajat haluaisivat kilpailla myös siitä kenellä on eniten sovelluskehittäjiä. Mielestäni se on järkevää, sillä enemmän sovelluskehittäjiä tarkoittaa enemmän sovelluksia. Se puolestaan kasvattaa laitteen potentiaalia.

Kulki kehitys mihin suuntaan tahansa, uskon mobiiliteknologian lisääntyvän rajusti tulevina vuosina ja vuosikymmeninä. Laitteista tulee langattomia yhä kiihtyvällä

vauhdilla, ja onpa tiedelehdissä kirjoiteltu jo langattomasta sähkönsiirrostakin. Kun fyysiset alustat on irrotettu kaapeleista, sovelluksia voidaan käyttää missä tahansa; toivottavasti myös käyttäjän ehdoilla. Sovellusten käyttötavat ja tarpeet muuttuvat varmasti, mutta en epäile etteivätkö ohjelmistokehittäjät ja markkinamiehet pysyisi perässä.

7.3 Toiminnanohjaus- ja tietojärjestelmien tulevaisuus

Aina tulee olemaan tarvetta organisaation toimintaa ohjaaville järjestelmille. Eri liiketoiminnan osa-alueet ovat kasvaneet vaatimuksiltaan niin paljon, että päätöksenteko ja hallinto tarvitsevat tehokkaat työvälineet. Kun tieto on kerran opittu keskittämään ja hyödyntämään tehokkaasti, ei paluuta entiseen enää ole.

Järjestelmät todennäköisesti kehittyvät ohjelmistoalan yleisten kehityssuuntien mukaisesti. Muutokset eivät varmasti tule olemaan nopeita tai noudata jokaista viimeisintä trendiä, johtuen järjestelmien massiivisesta koosta ja monimutkaisuudesta. Vanhat arkkitehtuuri- ja teknologiaratkaisut seuraavat uusien versioiden mukana, ellei koko järjestelmää suunnitella ja toteuteta uusiksi. Joissain tapauksissa se voi varmasti olla järkevämpi ratkaisu, kun olemassa olevan koodin ylläpito osoittautuu liian kalliiksi tai työlääksi.

Uusi kehityssuunta, jonka uskon kasvattavan suosiotaan tulevaisuudessa, on jo aiemmin mainittu pilvipalvelumalli. Pilven avulla järjestelmien koko ei ole ongelma ja ne voidaan skaalata juuri niihin tarpeisiin kuin kulloinkin vaaditaan. Se tuo ratkaisut yhä pienempien organisaatioiden ulottuville. Pilven ylläpito on vaivatonta palvelua käyttävälle osapuolella ja tehokasta toimittajalle. Suurin ongelma pilvipalvelussa on, ettei sitä voida räätälöidä samalla tasolla kuin perinteisesti toteutettua järjestelmää. Toisaalta suurimmat ongelmat ERP-projekteissa johtuvat räätälöinneistä. Nähtäväksi jää osuuko arvioni oikeaan. Lähes kaikki suuret ohjelmistotalot ovat kuitenkin investoineet pilvipalveluiden ja muun relevantin teknologian kehittämiseen, joten jotain merkittävää siitä varmasti syntyy.

LÄHTEET

A Guide To The Project Management Body of Knowledge - PMBOK Guide - Fourth Edition. 2008. Newtown Square: Project Management Institute, Inc.

Android. Verkkodokumentti. http://en.wikipedia.org/wiki/Android_%28operating_system%29. Päivitetty 23.2.2010. Viitattu 23.2.2010.

Bailor, Coreen. 2006. For CRM, ERP, and SCM, SAP Leads the Way. Verkkodokumentti. http://www.destinationcrm.com/Articles/ReadArticle.aspx?ArticleID=47784. Päivitetty 5.7.2006. Viitattu 13.4.2010.

Beers, David. 2009. Is that a Java application server running on your Palm Pre? Verkkodokumentti. http://www.pikesoft.com/blog/index.php?itemid=207. Päivitetty 13.1.2009. Viitattu 23.2.2010.

BlackBerry OS. Verkkodokumentti. http://en.wikipedia.org/wiki/BlackBerry_OS. Päivitetty 21.2.2010. Viitattu 22.2.2010.

Bramorski, Tom. 2004. A Case Study of ERP Implementation Issues. Powerpoint-esitys. Wisconsin: University of Wisconsin.

Brew. Verkkodokumentti. http://en.wikipedia.org/wiki/BREW. Päivitetty 23.10.2009. Viitattu 23.2.2010.

Burnette, Ed. 2008. Java vs. Android APIs. Verkkodokumentti. http://blogs.zdnet.com/Burnette/?p=504. Päivitetty 14.1.2008. Luettu 23.2.2010.

Crothers, Brooke. 2009. First iPhone, now Droid. Who needs Windows? Verkkodokumentti]. http://news.cnet.com/8301-1001_3-10392926-92.html?tag=col1;post-11516. Päivitetty 8.11.2009. Viitattu 23.2.2010.

Davenport, Thomas. 2000. Mission Critical: Realizing the Promise of Enterprise Systems. Harvard Business School Press.

Enterprise Resource Planning. 2010. Verkkodokumentti. http://en.wikipedia.org/wiki/Enterprise_resource_planning. Päivitetty 12.4.2010. Viitattu 13.4.2010.

Epicor ICE Business Architecture. Verkkodokumentti. http://www.epicor.com/Solutions/Pages/ICE-BusinessArchitecture.aspx. Päivitystietoa ei saatavilla. Viitattu 18.4.2010.

ERP Enterprise Solutions. Verkkodokumentti. http://www.epicor.com/PRODUCTS/Pages/E4SE.aspx. Päivitystietoa ei saatavilla. Viitattu 18.4.2010.

Foley, Mary Jo. 2010. Microsoft already setting a high expectation bar for Mobile World Congress. Verkkodokumentti. http://blogs.zdnet.com/microsoft/?p=4913&tag=col1;post-4913. Päivitetty 7.1.2010. Viitattu 23.2.2010.

Fowler, Martin. 2000. The New Methodology. Verkkodokumentti. http://martinfowler.com/articles/newMethodology.html. Päivitetty 13.12.2005. Viitattu 26.2.2010.

Fried, Jason. 2005. Getting Real: Pick two - scope, timeframe, or budget. Verkkodokumentti. http://37signals.com/svn/archives2/2005/04/getting_real_pi.php. Päivitetty 1.4.2005. Viitattu 25.2.2010.

Gold, Jack. 2009. Microsoft's Windows Mobile: Time to Hang Up? Verkkodokumentti. ttp://www.businessweek.com/technology/content/aug2009/tc2009087_110164.htm. Päivitetty 9.8.2009. Viitattu 23.2.2010.

Goodliffe, Pete. 2007. Code Craft. The Practice of Writing Excellent Code. San Francisco: No Starch Press Inc.

Guigova, Ilka. 2009. Approaches, Styles, or Philosophies in Software Development. Verkkodokumentti. http://www.codeproject.com/KB/architecture/sdlcstyles.aspx. Päivitetty 3.12.2009. Viitattu 26.2.2010.

Haikala, Ilkka; Märijärvi Jukka. 2006. Ohjelmistotuotanto. Jyväskylä: Talentum Media Oy.

Hamblen, Matt. 2009. Windows Mobile worries mount as competition heats up. Verkkodokumentti. http://www.computerworld.com/s/article/9139841/Windows_Mobile_worries_mount_as_competition_heats_up?taxonomyId=75&pageNumber=1. Päivitetty 26.10.2009. Viitattu 23.2.2010.

Highsmith, Jim. 2002. Agile Development Ecosystems. Indianapolis: Addison Wesley.

IEEE Standard Glossary of Software Engineering Terminology. PDF-dokumentti. http://www.engr.uvic.ca/~seng380/IEEE_Standard_Glossary_of_Software.pdf. Päivitetty 28.9.1990. Viitattu 24.2.2010.

iPhone OS. Verkkodokumentti. http://en.wikipedia.org/wiki/Iphone_os. Päivitetty 22.2.2010. Viitattu 22.2.2010.

Java Platform Overview. 2010. Verkkodokumentti. http://java.sun.com/javame/technology/index.jsp. Päivitystietoa ei saatavilla. Viitattu 21.3.2010.

Kontio, Mikko. 2002. Inside Mobiili Java - J2ME. Helsinki: IT-Press.

Kosonen, Pekka; Peltomäki, Juha; Silander, Simo. 2008. Java 2. Docenco.

Larman, Craig. 2009. Agile & Iterative Development - A Manager’s Guide. Westford: Addison Wesley.

Luckey, Teresa; Phillips Joseph. 2006. Software Project Management for Dummies. Indianapolis: Wiley Publishing Inc.

Maemo. Verkkodokumentti. http://en.wikipedia.org/wiki/Maemo. Päivitetty 22.2.2010. Viitattu 22.2.2010.

Martin, C. Robert. 2009. Clean Code. A Handbook of Agile Software Craftmanship. Prentice Hall.

McConnel, Steve. 2002. Ohjelmistotuotannon hallinta. Helsinki: IT-Press.

Mertanen, Juha. 2004. Pane yritys liikkeelle. Mobiiliratkaisut liiketoiminnan tukena. Helsinki: Talentum Media Oy.

Mikkonen, Tommi. 2004. Mobiiliohjelmointi. Jyväskylä: Talentum Media Oy.

Mobile computing. Verkkodokumentti. http://en.wikipedia.org/wiki/Mobile_computing. Päivitetty 21.2.2010. Viitattu 22.2.2010.

Mobile operating systems. Verkkodokumentti. http://en.wikipedia.org/wiki/Mobile_operating_system. Päivitetty 21.2.2010. Viitattu 22.2.2010.

Mobile phone. Verkkodokumentti. http://en.wikipedia.org/wiki/Mobile_phone. Päivitetty 18.2.2010. Viitattu 22.2.2010.

Mobile World Congress trends: applications thrive, mobile operators join forces. 2010. Vverkkodokumentti. http://www.independent.co.uk/life-style/gadgets-and-tech/news/mobile-world-congress-trends-applications-thrive-mobile-operators-join-forces-1900421.html. Päivitetty 15.2.2010. Viitattu 23.2.2010.

Moblin. Verkkodokumentti. http://en.wikipedia.org/wiki/Moblin. Päivitetty 16.2.2010. Viitattu 23.2.2010.

Monk, Ellen F; Wagner, Bret J. 2009. Concepts in Enterprise Resource Planning, Third Edition. Boston: Course Technology Cengage Learning.

Mäkipää, Marko. 2002. Toiminnanohjausjärjestelmän käyttöönotto - teoreettinen metodi ja empiirinen koettelu kahdessa case-yrityksessä. Tampere: Tampereen yliopisto.

New Product Development Glossary. 2007. Verkkodokumentti. http://www.npd-solutions.com/glossary.html. Päivitystietoa ei saatavilla. Viitattu 24.2.2010.

Perlow, Jason. 2009. In Smartphone Wars, Darwinism Triumphs Over Intelligent Design. Verkkodokumentti. http://blogs.zdnet.com/perlow/?p=11516. Päivitetty 8.11.2009. Viitattu 23.2.2010.

Sabbah, Danny. 2007. The Future of Software Delivery. PDF-dokumentti. IBM White Paper.

Service-oriented Architecture. 2010. Verkkodokumentti. http://en.wikipedia.org/wiki/Service-oriented_architecture. Päivitetty 13.4.2010. Viitattu 15.4.2010.

Shankland, Stephen. 2007. Google's Android parts ways with Java industry group. Verkkodokumentti. http://news.cnet.com/8301-13580_3-9815495-39.html. Päivitetty 12.11.2007. Viitattu 23.2.2010.

Smartphone.Verkkodokumentti. http://en.wikipedia.org/wiki/Smartphone. Päivitetty 18.2.2010. Viitattu 22.2.2010.

Symbian OS. Verkkodokumentti. http://en.wikipedia.org/wiki/Symbian_os. Päivitetty 21.2.2010. Viitattu 22.2.2010.

The Eve Virtual Machine and SDK. Verkkodokumentti. http://www.ewesoft.com/eve/TheEveVirtualMachineAndSDK.htm. Päivitystietoa ei saatavilla. Viitattu 23.2.2010.

Tieke. 2008. ERP luultua tärkeämpi pk-yritykselle. Verkkodokumentti. http://www.tieke.fi/tieke/tieken_tiedotteet_2008/erp_luultua_tarkeampi_pk-yrityks/. Päivitetty 7.2.2008. Viitattu 14.4.2010.

Unit testing. 2010. Verkkodokumentti. http://en.wikipedia.org/wiki/Unit_test. Päivitetty 14.4.2010. Viitattu 18.4.2010.

Vanderboom, Dan. 2009. Software Development Methods. Verkkodokumentti. http://dvanderboom.wordpress.com/2009/08/19/software-development-methods/. Päivitetty 19.8.2009. Viitattu 26.2.2010.

Wailgum, Thomas. 2008. ERP Definition and Solutions. Verkkodokumentti. http://www.cio.com/article/40323/ERP_Definition_and_Solutions. Päivitetty 17.4.2008. Viitattu 14.4.2010.

WebOS. Verkkodokumentti. http://en.wikipedia.org/wiki/WebOS. Päivitetty 22.2.2010. Viitattu 23.2.2010.

WebShpere Everyplace Micro Environment. Verkkodokumentti. http://www-01.ibm.com/software/wireless/weme/. Päivitystietoa ei saatavilla. Viitattu 23.2.2010.

Windows Mobile. Verkkodokumentti. http://en.wikipedia.org/wiki/Windows_Mobile. Päivitetty 22.2.2010. Viitattu 22.2.2010.

Young, Andrew. 2009. David Beers talks on webOS, web apps, and Java. Verkkodokumentti. http://www.weboshelp.net/all-webos-news-articles/117-david-beers-talks-on-webos-web-apps-and-java. Päivitetty 14.2.2009. Viitattu 23.2.2010.

LIITE 1.

Projektisuunnitelman sisältö

Table of Contents

LIITE 2.
Projektisuunnitelman johdanto ja projektin tavoitteet

1 Introduction

1.1 General description

E4SE Mobile Client is a Java ME based application for mobile Internet devices. The application makes possible to enter time and expense entries against an E4SE project. It is developed to ease the process of entering transactions by making it possible regardless of the time and place.

Currently E4SE lacks a platform independent mobile client and thus project is worthwhile completing.

1.2 Project plan maintenance

The project plan will be maintained and reviewed on a weekly basis. The maintenance actions include reviews and optionally meetings to better understand and define different changing aspects, requirements, timeline and other properties. The project plan will be kept as accurate and up to date as possible.

2 Organizing and steering

2.1 Objectives and requirements

The goal of the project is to develop the first operational version of the E4SE Mobile Client software. This version includes complete documentation and specifications of the software, development project and end user manual. The application should meet the requirements defined in *Functional Specification* and follow the principles described in *Project Plan* and implemented as described in detail in *Solution Design*. The primary objectives should be completed during 2010.

| The original timeframe has been reviewed and updated to meet the current situation.

The project will be determined completed and closed when the requirements are fulfilled and project steering group validates the software against the *Functional Specification* and project scope described in next section. Furthermore the software is required to pass each predefined test case described in *Testing Plan* and the acceptance testing.

Other requirements and critical functionality will be discussed in more detail in *Functional Specification* document.

4.1.4 Select a specific date

Priority
High

Target version
0.5

Description
The user has to be able to conveniently select a date for which to perform actions. The selection method has to be simple, consistent and logical. There should be no more than three steps from application startup screen to view of the selected a date. The date selection component should have a calendar like interface.

Use case / user story

1. Business user, e.g. Ron, starts up the E4SE Mobile Client application on the MID.
2. He selects *Calendar* menu option to launch the appropriate screen.
3. Application displays a calendar screen with current day selected on default.
4. He uses the calendar navigation options on the user interface to select a date. Calendar user interface provides a quick way for selecting desired year and month. The user interface displays also week numbers. The standard view displays a whole month on the screen.
5. He opens the selected date from the user interface.
6. The application displays a day view of the selected date with previous entries and options for toggling in between the time and expense entries. By default the time entries are shown. The user interface also contains menu options for adding a new entry, selecting entries and modifying or deleting the selected entries.