Saimaan ammattikorkeakoulu
Tekniikka Lappeenranta
Tietotekniikan koulutusohjelma
Ohjelmistotekniikan suuntautumisvaihtoehto

Timo Paajanen

INNONET-TOIMINNANOHJAUSJÄRJESTELMÄ

Opinnäytetyö 2011

TIIVISTELMÄ

Timo Paajanen

Innonet-toiminnanohjausjärjestelmä, 97 sivua
Saimaan ammattikorkeakoulu, Lappeenranta
Tekniikka, Tietotekniikan koulutusohjelma
Ohjelmistotekniikan suuntautumisvaihtoehto
Opinnäytetyö 2011
Ohjaajat: Lehtori Martti Ylä-Jussila, Saimaan ammattikorkeakoulu
Toimitusjohtaja Markus Heikkinen, Innotek Oy

Asiakkaana opinnäytetyössä on lieksalainen LVI-alan yritys Innotek Oy. Työn kohteena on Innotek Oy:n käynnistämä Energo-säästöohjelman toimintaprosessin ja siihen liittyvän Innonet-järjestelmän kehittämishanke. Opinnäytetyön tavoitteena oli laatia Innonet-järjestelmän toiminnallinen määrittelydokumentti sekä suunnitella järjestelmän yleisarkkitehtuuri jatkokehitystyötä varten.

Innonet-järjestelmä on tarkoitettu ohjaamaan ja automatisoimaan Innotek Oy:n Energo-säästöohjelman liiketoimintaprosesseja. Innotek Oy:n tavoitteena on tehostaa asentajien työnohjausta, vähentää käsin tehtävää paperityötä ja samalla automatisoida Energo-säästöohjelmaan liittyvää tietojenkäsittelyä Innonet-järjestelmän avulla.

Opinnäytetyö aloitettiin tekemällä alustava tutkimus ja esiselvitys vanhasta Innonet-järjestelmästä sekä keräämällä asiakkaan järjestelmälle asettamat vaatimukset. Esiselvityksessä kuitenkin selvisi, ettei vanhasta järjestelmästä ollut kunnollista dokumentaatiota, eivätkä kerätyt vaatimukset vastanneet järjestelmään toteutettuja toimintoja. Esiselvityksen lopputuloksena päätettiin tehdä järjestelmästä kunnollinen dokumentaatio ja toiminnallinen määrittelydokumentti.

Opinnäytetyön tuloksena asiakkaalle tuotettiin järjestelmän toiminnallinen määrittelydokumentti, yleisarkkitehtuurisuunnitelma sekä suunniteltiin ja toteutettiin tietokanta ja määriteltiin ohjelmoinnissa käytettävät välineet, standardit ja käytännöt.

Järjestelmän toteutustyö jatkuu usean opinnäytetyön puitteissa.

Asiasanat: Innonet, toiminnanohjausjärjestelmä, CodeIgniter, PHP, Energo-säästöohjelma, Innotek Oy, toiminnallinen määrittely

ABSTRACT
Timo Paajanen
Innonet Enterprise resource planning, 97 pages
Saimaa University of Applied Sciences, Lappeenranta
Technology, Degree Program in Information technology
Software Engineering
Bachelor's Thesis, 2011
Instructors: CEO Markus Heikkinen, Innotek Oy
Lecturer Martti Ylä-Jussila, Saimaa University of Applied Sciences

The client in this thesis was Innotek Oy from Lieksa, a company in the HVAC area of business. The purpose of the thesis was to continue the development of Innotek Oy's enterprise resource planning system Innonet. The goal was to form the functional specification and to design the overall software architecture of the Innonet system.

The purpose of the Innonet system is to direct and automate some of the Energo saving program's business processes. Innotek Oy's goals are to enhance work guidance and reduce manual paperwork and material cost involved in the Energo program. Innotek Oy is trying to achieve these goals with the Innonet system.

The thesis was started by doing a preliminary assessment and analysis of the old Innonet system. The analysis yielded a result that there was not adequate documentation about the system's functional and nonfunctional requirements. A decision was made to gather and document these requirements into a software development artifact called functional specification.

As outcome of this thesis, the functional specification and overall system architecture has been designed. System planning and development have also been started.

The development of Innonet system will continue in terms of a couple of bachelor's theses.

Keywords: Innonet, Enterprise resource planning, CodeIgniter, PHP, Energo program, Innotek Oy, functional specification

KÄSITTEET, TERMIT JA LYHENTEET

AJAX	AJAX (Asyncronous JavaScript And XML) on joukko erilaisia tekniikoita, joiden avulla WWW-sovelluksista pyritään tekemään vuorovaikutteisempia, nopeampia ja käytettävämpiä.
CodeIgniter	CodeIgniter on avoimeen lähdekoodiin perustuva PHP-sovelluskehys, jonka rakenne perustuu MVC-arkkitehtuurimalliin. CodeIgniter tarjoaa PHP-sovelluskehittäjälle perusedellytykset täysipainoisten WWW-sovellusten toteuttamiseen.
CSS	CSS (Cascading Style Sheets) on W3C:n kehittämä ja ylläpitämä tyyliohjejärjestelmien laji, joka sisältää periaatteen useiden eri lähteistä peräisin olevien tyyliohjeiden soveltamisen samanaikaisesti tarkoin määriteltyjen preferenssisääntöjen mukaan.
CSS1	CSS1 (Cascading Style Sheet Level 1) on W3C:n kehittämän tyyliohjejärjestelmän ensimmäinen kehitysversio, joka julkaistiin vuonna 1996 HTML:ää varten.
CSS2	CSS2 (Cascading Style Sheet Level 2) on vuonna 1998 julkaistu CSS1 version tyyliohje määritelmiä laajentava toinen kehitysversio.
ERP	ERP (Enterprise Resource Planning) eli toiminnanohjausjärjestelmä on yrityksen liiketoimintaa ohjaava järjestelmä.

HTML	HTML (Hypertext Markup Language) on avoimesti standardoitu kuvauskieli, jolla voidaan kuvata hyperlinkkejä sisältävää tekstiä. HTML:ää käytetään erityisesti WWW-sivujen toteutuksessa.
HTTP	HTTP protokollaa (Hypertext Transfer Protocol) käytetään tiedonsiirtoon WWW-selaimen ja WWW-palvelimen välillä. HTTP-protokolla sijaitsee OSI-mallin ylimmällä kerroksella eli sovelluskerroksella.
JavaScript	JavaScript on alun perin Netscape Communication Corporationin kehittämä WWW-ympäristöissä käytettävä komentosarjakieli. JavaScriptiä käytetään lisäämään WWW-sivuille dynaamista toiminnallisuutta. JavaScript-komennot suoritetaan asiakasympäristössä eli WWW-selaimessa.
MDA	MDA (Model Driven Architechture) on OMG:n kehittämä tehokas visuaalinen suunnittelu-, toteutus- ja ylläpitomenetelmä ohjelmistotuotantoon.
MRP	MRP (Materials Requirement Planning) on 70-luvulla kehitetty, tietojärjestelmä, jolla hallitaan tuotteen rakennetta ja materiaaleja sekä tuotantoaikatauluja.
MRP II	MRP II (Manufacturing Resource Planning) on kehittyneempi versio MRP-järjestelmästä. Se pitää yhdistää tehokkaasti suunnittelukäytännöt kaikista tuotantoorganisaation resursseista.
MySQL	MySQL on avoimeen lähdekoodiin perustuva SQL-tietokannan hallintajärjestelmä.

OMG	OMG (Object Management Group) on kansainvälinen voittoa tavoittelematon organisaatio, jonka tehtävänä on kehittää ja ylläpitää koko yritysmaailmaa yhdistäviä standardeja.
PHP	PHP (Hypertext Preprosessor) on yleiskäyttöinen komentosarjakieli, jota käytetään erityisesti WWW-palvelinympäristöissä dynaamisten WWW-sivujen ja sivustojen kehittämisessä. PHP-komennot suoritetaan palvelinympäristössä eli WWW-palvelimessa.
RDP	RDP (Remote Desktop Protocol) on Microsoftin kehittämä etäkäyttöprotokolla, joka tarjoaa käyttäjälle graafisen käyttöliittymän toisen etätietokoneeseen.
RUP	RUP (Rational Unified Process) on iteratiivisen ja inkrementaalisen ohjelmistokehityksen prosessikehys. Se kehitettiin kolmen suurimman ohjelmistokehitysmetodologin ja laajan yhteisön toimesta Rational Software Corporationin alaisuudessa 1990-luvun lopulla.
SCRUM	Scrum on projektinhallinnan menetelmä, jota käytetään yleisesti ketterässä ohjelmistokehityksessä. Se on kehitetty erityisesti ohjelmistoprojektien hallintaan, mutta sitä voidaan soveltaa myös yleisesti projektinhallinnassa.
SQL	SQL (Structured Query Language) on standardoitu kyselykieli, jolla voidaan hallinnoida relaatiotietokantaa.

UML	UML (Unified Modeling Language) on ohjelmistojen analysointiin ja mallintamiseen tarkoitettu, standardoitu graafinen kuvauskieli. UML standardoitiin vuonna 1997 ja se kehitettiin kolmesta tuolloin vallinneesta oliosuunnittelu menetelmästä (OOSE, BOOCH, OMT).
W3C	W3C (World Wide Web Consortium) kehittää yhteisiä ja yhteensopivia Webin pelisääntöjä ja teknologioita. Työn tavoitteena on ohjata Webin kehittymistä täyteen mittaansa tiedonvälityksen, kaupankäynnin, kommunikaation ja yhteisymmärryksen foorumina.
WampServer	WampServer (Windows, Apache, MySQL, PHP) on avoimen lähdekoodin ohjelmistoihin perustuva ilmainen WWW-palvelin ohjelmisto Windows-alustoille.
XML	XML (eXtensible Markup Language) on rakenteellinen kuvauskieli, joka auttaa jäsentämään laajoja tietomassoja. XML-kieltä käytetään formaattina järjestelmien välisessä tiedonsiirrossa ja erilaisten dokumenttien tallentamiseen.
XP	XP (Extreme Programming) on ketterän ohjelmistokehitysmenetelmä. Sen kehittivät Kent Beck, Ward Cunningham ja Ron Jeffries 1990-luvun lopulla.

SISÄLTÖ

1 JOHDANTO

Tämän opinnäytetyön kohteena on nykyaikana hyvin tyypillinen yrityksen toimintaprosessin ja palvelutuotteen kehityshanke. Asiakkaana hankkeessa on lieksalainen LVI-alan yritys Innotek Oy.

Yritystoiminnan elinehtona on taloudellinen kannattavuus, joka perustuu siihen, että yritys tuottaa asiakkailleen laadukkaita, kilpailukykyisiä ja kohtuuhintaisia tuotteita. Vapailla markkinoilla on toimialuekohtaista kilpailua, jonka vuoksi yrityksien on jatkuvasti kehitettävä tuotteita ja palveluita sekä tuotava uusia innovaatioita markkinoille. Jatkuvan tuotteiden ja palveluiden kehittämisen tarkoituksena on auttaa yritystä toimimaan taloudellisemmin ja tehokkaammin, jotta pysytään mukana kilpailussa ja pidetään yritystoiminta kannattavana. Markkinoilla vallitseva kilpailu hyödyttää asiakkaita. Kilpailun johdosta tarjolla on parempia tuotteita ja palveluita. Energian, resurssien ja raaka-aineiden säästäminen on ollut kehityshankkeiden lähtökohtana jo vuosikymmeniä.

Nykyaikainen mobiili tietojenkäsittely mahdollistaa yrityksen toimintaprosessien, toiminnanohjauksen ja tietojenkäsittelyn uudistamisen ja tehostamisen, jolla on ratkaiseva vaikutus yrityksen kilpailukykyyn markkinoilla niin laadussa kuin kustannuksissa.

Opinnäytetyön kohteena oleva hanke on tyypillinen esimerkki innovaatiohankkeesta, jossa Innotek Oy pyrkii tuottamaan omalla Energo-palvelutuotteellaan vedenkulutuksen ja energian säästöä omille asiakkailleen eli taloyhtiöille. Palvelutuotteensa ja toimintansa kehittämiseksi Innotek Oy on käynnistänyt kehityshankkeen, jonka tavoittena on automatisoida Energo-ohjelmaan liittyvää tietojenkäsittely ja toiminnanohjaus mobiilin tietojenkeruun ja keskitetyn tietojen loppukäsittelyn avulla.

Hanke on käynnistetty vuonna 2004. Innotek Oy ja Pohjois-Karjalan ammattikorkeakoulu ovat tehneet hanketta vuosien 2004 - 2006 aikana. Työn tuloksena oli saatu aikaan alustavat versiot tietokannasta ja mobiilisovelluksesta sekä WWW-pohjaisesta sivustosta, jolla kerättyjä tietoja hallitaan.

Projektissa oli tarkoituksena jatkokehittää Innotek Oyn vanhaa Energosäästöohjelmaan pohjautuvaa järjestelmää. Asiakkaalla oli selkeät tavoitteet järjestelmän suhteen, mitä sen pitää tehdä ja miten, jotta se täyttää tietyt yrityksen liiketoimintaan liittyvät vaatimukset. Tämä toimii myös hyvänä pohjana kehitettävän järjestelmän hyödyntämisessä. Järjestelmän tarkoituksena on vähentää turhaa paperityötä ja samalla automatisoida säästöohjelmaan liittyviä liiketoimintaprosesseja.

Projektin edetessä tavoitteet kuitenkin muuttuivat alustavan tutkimustyön perusteella, koska vanhasta järjestelmästä ei ollut kunnollista dokumentaatiota, jonka pohjalta liiketoiminnalliset mallit ja prosessit olisivat olleet selkeästi omaksuttavissa.

Liityin Saimaan ammattikorkeakoulun Innotek-asiakasprojektiin suorittamaan opinnäytetyötäni rajaamatta aluksi tarkasti tavoitteita ja tehtäviä, koska projektiin tarvittiin lisää työvoimaa moniin eri tehtäviin. Opinnäytetyöni tehtäviksi muodostuivat alussa järjestelmän toiminnallisten vaatimusten kerääminen ja tarkentaminen, toimintaprosessien suunnittelu ja dokumentointi, kehitysympäristön suunnittelu ja pystytys, kehitysvälineiden valinta, ohjelmistoarkkitehtuurin suunnittelu, ohjelmointistandardien ja malliohjelmien laatiminen sekä uusien suunnittelijoiden työnopastus. Loppuvaiheessa toimin projektipäällikkönä ja projektiryhmän vetäjänä.

2 ASIAKAS

2.1 Innotek Oy

Innotek Oy on lieksalainen, pääasiassa LVI-tuotteita valmistuttava sekä vesikalusteiden kuntoa kartoittava ja asennuksia tekevä yritys. Sen asiakkaat kostuvat pääasiassa huoltoyhtiöistä, isännöintiyrityksistä sekä yksityisistä asiakkaista, ympäri Suomea. Innotek Oy:n päätoimipiste sijaitsee Lieksassa ja muut toimipisteet Helsingissä ja Kempeleellä. Innotek Oy:n tuotevarastot sijaitsevat Lieksassa, Keravalla ja Kempeleellä. Henkilöstö koostuu tällä hetkellä kymmenestä työntekijästä. Liikevaihto on noin 1,2 miljoonaa euroa.

Innotek Oy:llä on käytössä yrityksen sisäinen, sähköinen laskutusjärjestelmä Econet 2000, jolla hoidetaan yrityksen koko laskutus. Päivittäiset työt, kuten raportointi, työtehtävien aikataulutus sekä asiakkaiden tiedotus hoidetaan tällä hetkellä pääasiassa Excel- ja Word-dokumentein. Yrityksen laskutus hoidetaan sisäisesti, mutta kirjanpito on ulkoistettu Tilitoimisto Optimus Oy:lle.

2.2 Energo-säästöohjelma

Innotek Oy:n liiketoiminta sisältää veden ja energian kulutuksen minimointiin keskittyvän toimintamallin, Energo-säästöohjelman. Energo-säästöohjelma koostuu erikoistuotteista ja palveluista, joilla turhaa vedenkulutusta voidaan poistaa vesikalusteista. Sen avulla on saavutettu keskimäärin 20 - 35 % matalampi vedenkulutustaso ja 5 - 15 % pienempi energiankulutustaso.
(Innotek Oy 2000a)

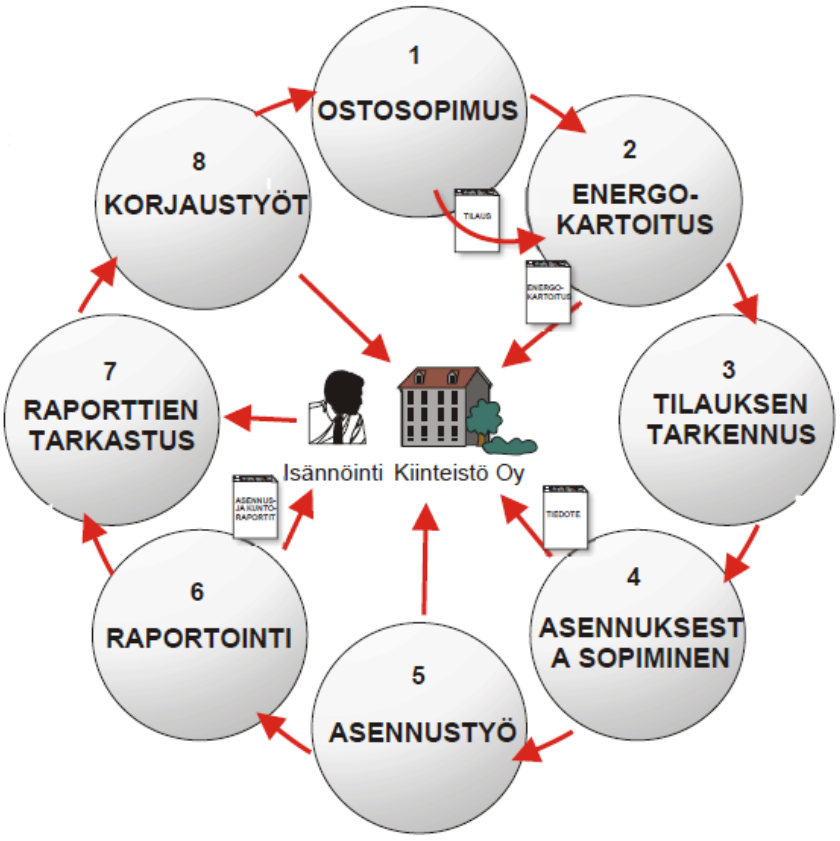
Energo-kartoitus on säästöohjelman ensimmäinen vaihe ja se pitää sisällään kiinteistön vedenkulutuksen kartoituksen ja alustavien toimenpiteiden määrittelyn kiinteistön vedenkulutuksen pienentämiseksi. Energo-kartoituksen avulla täsmennetään tilattujen toimenpiteiden laajuus ja tarkoituksenmukaisuus. Energo-kartoituksesta tehdään kiinteistön isännöitsijälle Energo-kuntoraportti, josta selviää tiloissa ja kalusteissa havaitut viat ja huomiot. Raportin avulla isännöitsi-

ja voi arvioida korjaustöiden tarpeellisuuden, kiireellisyyden ja ajankohdan (Innotek Oy 2000a).

Energo-kuntoraportissa esille tulevat asiat:

- vesikalusteiden rikkonaisuus
- havaitut vuodot
- kosteiden tilojen pintojen rikkonaisuus
- asukkaiden esille tuomat viat
- selkeät haju- ja siisteyspoikkeamat
- mahdolliset muut viat, joilla on merkitystä isännöitsijälle tai kiinteistönomistajalle(Innotek Oy. 2000a).

Kuvassa 2.1 on esitelty graafisesti Innotek Oy:n toimintamalli Energo-säästöohjelmassa.



Kuva 2.1 Energo-säästöohjelma (Innotek Oy 2000a)

3 TOIMINNANOHJAUSJÄRJESTELMÄT

Toiminnanohjaus ja toiminnanohjausjärjestelmät ovat nykypäivän tietoyhteiskunnassa ja liiketoiminnassa tärkeässä asemassa, koska yhä useammin pienet ja keskisuuret yritykset pyrkivät automatisoimaan sisäisiä ja palveluihin liittyviä toimintojaan. Hyvin toteutettu ja hyvin yrityksen toimintoihin soveltuva toiminnanohjausjärjestelmä tehostaa liiketoimintaa ja alentaa toimintoihin liittyviä kustannuksia. Tämä antaa yrityksille selvää etua kilpailijoihin nähden vapaassa markkinajärjestelmässä, jossa myymällä tai jatkojalostamalla tuotteita tai palveluita tavoitellaan rahassa mitattavia liikevoittoja.

Nykyajan yhteiskunnassa yritysten ja erilaisten organisaatioiden käsittelemän tiedon määrä on jatkuvassa kasvussa. Usein yrityksen menestyminen markkinoilla perustuukin siihen kuinka se sopeutuu tietojen hallinnan, jalostuksen ja jakamisen tuomiin haasteisiin. Nykyisin ohjelmistoyhtiöt ja -yritykset kehittelevätkin yhä monipuolisempia, laajempia ja monimutkaisempia tietojärjestelmiä vastaamaan yrityksien tarpeita. Näitä tietojärjestelmä kokonaisuuksia kutsutaan toiminnanohjausjärjestelmiksi. (Mäkipää 2002)

Toiminnanohjausjärjestelmä tarjoaa yritykselle tai organisaatiolle integroidun ratkaisun tietomassojen ja liiketoimintaprosessien sekä osittain ulkoisien yhteyksien hallintaan. Toiminnanohjausjärjestelmän valinnalla ja käyttöönotolla pyritään tukemaan ja edistämään yrityksen liiketoimintaa. Nykyisistä toiminnanohjausjärjestelmistä on muodostunut ajan kuluessa monimutkaisia kokonaisuuksia, joiden ominaisuudet ja mahdollisuudet riippuvat täysin siitä kuinka suuren osan niiden suunnitellaan kattavan yrityksen tai organisaation prosesseista. (Kettunen & Simons 2001)

3.1 MRP ja MRP II

Toiminnanohjausjärjestelmien kehityksen katsotaan alkaneen 1970-luvun alussa, jolloin aloitettiin kehittämään MRP-järjestelmiä tukemaan yritystoimintaa. MRP-tietojärjestelmien tarkoituksena oli tuottaa materiaalitarvelaskentoja varasto- ja hankintatoimintoja varten. Ohjelmistojen oli tarkoitus automatisoida tilaus-

ten tekemistä ja taloudellisten eräkokojen laskentaa tuotannon suunnittelua varten. Vuosikymmenen lopulla kaupallisten niin sanottujen standardiohjelmistojen valmistus lisääntyi ja ohjelmistojen räätälöinti yhden yrityksen tarpeisiin soveltuvaksi muuttui yleiseksi ohjelmistojen paketoinniksi eli useita yrityksiä yhdistävien toimintojen kattaviksi kokonaisuuksiksi. (Kettunen & Simons 2001)

1980-luvulla varaston ja tuotannonhallinnan hallintaan alettiin kehittelemään MRP II -konseptia. Se perustui vanhaan MRP-järjestelmään laajentaen sen toiminnallisuutta uusien toimintojen, lattiatason toiminnanohjauksen ja jakelunhallinnan osa-alueilla. PC-tietokoneiden yleistyminen ja kehittyminen vauhditti MRP II -järjestelmien kehitystä ja levinneisyyttä. (Kettunen & Simons 2001)

3.2 ERP

1990-luvulla MRP2-järjestelmiin lisättiin ja liitettiin toiminnallisuutta tuotannonohjauksesta ja muista osa-alueista, joiden kehittäminen ohjelmistotuotannossa oli tällöin kulkenut lähes täysin erillään. Tästä päästiinkin MRP- ja MRP II konseptien pohjustamana nykyiseen ERP-konseptiin. Myöhemmin 1990-luvun lopulla konseptiin liitettiin myös toimintoja sähköiseen kaupankäyntiin ja tiedonsiirtoon yritysten välisissä tietojärjestelmissä. Internetin tarjoamat mahdollisuudet ja halventuneet siirtokustannukset ovat vauhdittaneet ERP-järjestelmien laajentumista ja kehittymistä. Tällä hetkellä puhutaankin verkostojen toiminnanohjauksesta ja toimintojen optimoinneista yritysten välillä. Nykyisin useat yritykset ovatkin ruvenneet ottamaan käyttöön toiminnanohjausjärjestelmäkokonaisuuksia, joiden uskotaan kattavan koko yritystoiminnan tarpeet ja tuovan samalla merkittäviä säästöjä tietojen käsittelyyn, hallinnointiin ja jakeluun. Vielä 1990-luvun alussa toiminnanohjausjärjestelmät olivat vain suurien kansainvälisten yritysten käyttöön suunnattuja. Idean huomattiin kuitenkin olevan niin toimiva, että konseptia laajennettiin myös keskisuurten ja pienten yritysten käyttöön sopivaksi. (Kettunen & Simons 2001; Mäkipää 2002)

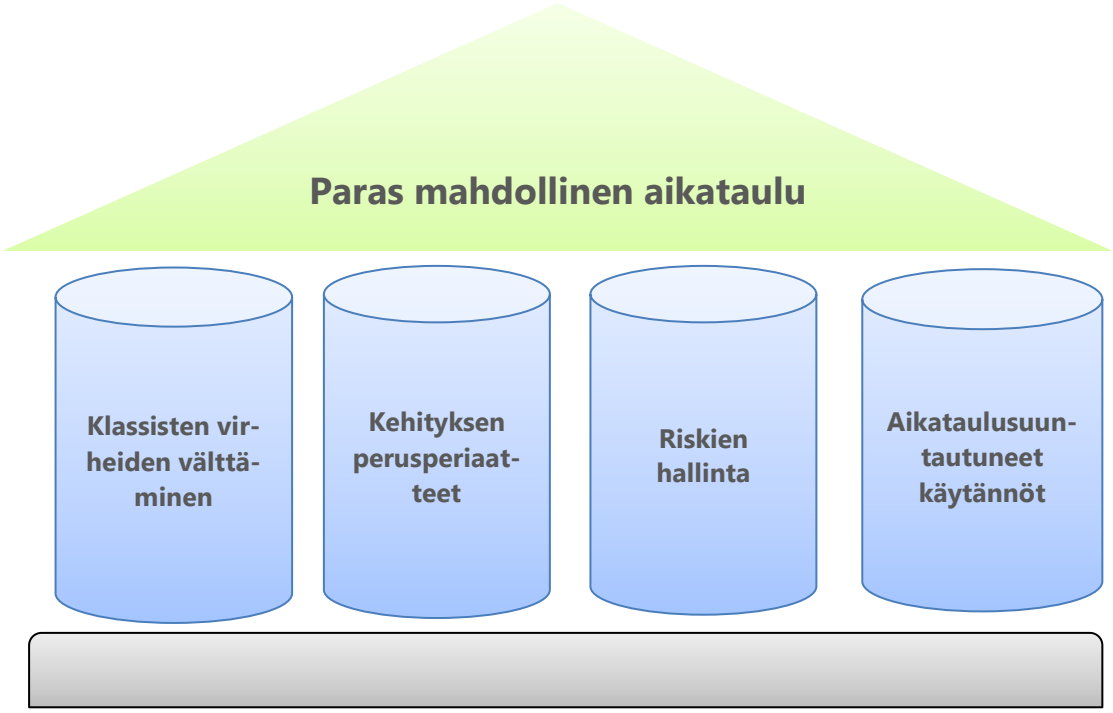
4 OHJELMISTOTUOTANTO

Tässä luvussa perehdytään nopeaan ohjelmistokehittämiseen, ohjelmiston elinkaareen ja ohjelmistotuotannon elinkaarimalleihin.

Nykyisin ohjelmistotuotannossa tavoitellaan nopeaa kehittämisnopeutta, jolla mahdollistetaan kehitettävän ohjelmistotuotteen alhaiset kustannukset ja mahdollinen etu kilpailijoihin nähden, kun tuote saadaan aikaisemmin markkinoille. Ohjelmistotuotanto on nuori ala, joka on jatkuvassa muutoksessa. Uusia menetelmiä ja käytäntöjä kehitetään jatkuvasti, jotta alalla tapahtuviin muutoksiin pystytään vastaamaan nopeasti ja tehokkaasti. Ohjelmistotuotannossa voidaan siis havaita samoja säännönalaisuuksia kuten biologisessa evoluutiossa, jossa parhaiten ympäristöönsä soveltuva menestyy ja välittää perittyjä ominaisuuksia eteenpäin.

4.1 Nopea ohjelmistokehitys

Ohjelmiston kehittäminen on vielä nykyaikanakin hyvin pitkälle projektilähtöistä kehitystyötä. Usein ohjelmistojen kehittämisessä tavoitellaan mahdollisimman nopeaa kehittämisnopeutta ja aikataulusuuntautuneisuutta. Tämä voi usein tuottaa ongelmia matkan varrella ja aikatauluista myöhästytään yllättävän usein. Hyvän aikataulun saavuttamisessa täytyy kiinnittää huomiota neljään tärkeään tekijään. Nämä neljä tekijää ovat nopean ohjelmistokehittämisen tukipilareita. Kun kaikki tukipilarit ovat hyvässä tasapainossa, on paras mahdollinen aikataulu saavutettavissa. Kuvasta 4.1 nähdään tukipilarit tasapainoa kuvaavassa rakennelmassa. (McConnell 2002)



Kuva 4.1 Nopean kehittämisen neljä tukipilaria (McConnell 2002)

Kuten kuvan 4.1 rakennelmasta nähdään saa paras mahdollinen aikataulu parhaan tuen, kun kaikki neljä tulipilaria ovat tasapainossa. Jos tasapainottavista tekijöistä lähdetään tinkimään ja keskitytään esimerkiksi pelkästään aikataulusuuntautuneisiin käytäntöihin, on parhaan mahdollisen aikataulun toteutuminen vaarassa. (McConnell 2002)

Ohjelmiston kehittäminen on monivaiheinen prosessi, joten klassisia virheitä tulisi välttää koko ohjelmiston elinkaaren aikana. Hyvänä tekniikkana voidaan pitää listausta klassisista virheistä, joita tarkkaillaan koko ohjelmistoprojektin elinkaaren ajan. Tarkkailu tapahtuu koko projektiryhmän voimin ja näin pyritään välttämään listalla olevien klassisten virheiden tapahtuminen.
(McConnell 2002)

Ohjelmisto kehittämisen perusperiaatteisiin on myös kiinnitettävä huomiota. Perusperiaatteet voidaan jaotella hallinnollisiin, teknisiin ja laadunvalvonnallisiin periaatteisiin. Hallinnolliset periaatteet sisältävät työmäärien ja aikataulutuksen arvioinnin, projektisuunnittelun, seurannan ja mittaamisen, jonka avulla suoritettuja projekteja voidaan verrata keskenään ja käytäntöjä parantaa seuraaviin projekteihin. Teknisiin perusteisiin sisältyvät vaatimusten hallinta, toteutuksen

suunnittelu, rakentaminen eli itse toteuttaminen ja rakennettujen osien kokoonpano. Laadunvalvonnalliset perusteet käsittävät vikojen etsinnän moduuleista, moduulien ja järjestelmän osien erilaisen testaukset sekä tekniset katselmoinnit. Tärkeintä ohjelmiston kehittämisen perusperiaatteissa on niiden ohjeellinen noudattaminen. Koska eri osa-alueet tukevat toinen toistaan luoden kokonaisuuden, jossa heikoin lenkki määrittelee lopputuloksen. Esimerkkinä huonosti sovelletut tekniset periaatteet voivat aiheuttaa sen, ettei ohjelmisto valmistu ajallaan, koska tekijöillä menee enemmän aikaa virheiden korjaamiseen kuin uuden toiminnallisuuden tekemiseen. Samankaltainen vaikutus voi olla myös heikosti hoidetulla hallinnoinnilla; jos aikataulutus ja työmääräarviot ovat epärealistisia, kääntyy tekemisen luonne nopeasti kiirehtimiseen ja ylitöiden tekemiseen. Tämä taas johtaa siihen, että tekijät tekevät enemmän virheitä ja aikaa kuluu lopulta enemmän virheiden korjaukseen kuin uuden tekemiseen. Ohjeiden seuraaminen on siis tärkeässä roolissa ja varsinkin suurissa ohjelmistoprojekteissa, joissa ne todella säästävät aikaa. (McConnell 2002)

Ohjelmistojen kehittämisprojektit sisältävät suuren määrän riskejä. Riskit tulee tunnistaa, käsitellä ja minimoida mahdollisimman aikaisessa vaiheessa. Riskejä tulee myös seurata koko projektin ajan. Tietyssä vaiheessa projektia riskejä voi poistua kokonaan ja uusia riskejä ilmaantua, joten myös riskien seuranta kuuluu osana riskien hallintaan. (McConnell 2002)

Kun edellä mainitut osa-alueet ovat hallinnassa, voidaan valita aikataulusuuntautuneita käytäntöjä, joilla voidaan nopeuttaa ohjelmiston kehitysprojektia. Aikataulusuuntautuneista käytännöistä on myös hyvä muistaa se, etteivät ne sovellu kaikenlaisien ohjelmistoprojektien nopeuttamiseen. Tietyntyyppisissä projekteissa voi olla tehokasta käyttää esimerkiksi case-välineitä ja toisissa taas nopeaa prototyyppien rakentamista. Käytettävien aikataulusuuntautuneiden tekniikoiden valinta voi riippua rakennettavasta tuotteesta ja sen luonteesta. Esimerkiksi onko tuote kohdistettu ihmisen elintoimintoja ohjaavaksi sulautetuksi järjestelmäksi, jolla luotettavuus on elintärkeässä asemassa, tai onko kehitettävä tuote verkkosovellus, joka tulee rajatun ryhmän käyttöön.
Kummankin tuotteen kehittämiseen valitaan siihen parhaiten soveltuva käytäntö. (McConnell 2002)

4.2 Ohjelmiston elinkaari

Ohjelmiston elinkaari käsittää kaikki ohjelmiston kehittämisessä läpikäytävät tehtävät: esitutkimus, vaatimusmäärittely, arkkitehtuurisuunnittelu, yksityiskohtainen suunnittelu, toteutus, testaus ja käyttöönottoon sekä ylläpito. Näitä edellä mainittuja tehtäviä kutsutaan ohjelmiston elinkaaressa vaiheiksi ja useat elinkaari mallit ovatkin niin sanottuja vaihejakomalleja. Vaihejakomallit on jaoteltu perinteisiin, ketteriin ja inkrementaalisiin malleihin. (McConnell 2002)

4.3 Koodaa ja korjaa

Koodaa ja korjaa -malli on harvoin hyödyllinen, siitä huolimatta se on yleinen ja sitä käytetään, kun uusia ohjelmointikieliä opiskellaan. Se on myös oletusarvoinen elinkaarimalli silloin, kun mitään muuta elinkaarimallia ei ole erikseen valittu. Yhtenä erittäin huonona puolena tässä elinkaarimallissa on sen yhdistäminen erittäin lyhyeen aikatauluun, jolloin siitä voi muodostua niin sanottu "koodataan pirusti" -malli. (McConnell 2002)

Kuva 4.2 Koodaa ja korjaa -elinkaarimalli

Koodaa ja korjaa -malli omaa kuitenkin kaksi selvää etua. Ensimmäisenä on ajankäyttö eli yhtään ylimääräistä aikaa ei kulu projektin ja kehitettävän ohjelman tukitoimintoihin esimerkiksi määrittelyyn, suunnitteluun, dokumentointiin tai laadunvalvontaan. Toisena etuna koodaa ja korjaa -mallissa on se, ettei sen

käyttäminen vaadi ammattitaitoa, eli kuka tahansa, joka on kirjoittanut tietokoneohjelman voi käyttää sitä ja tuntee mallin. (McConnell 2002)

Koodaa ja korjaa -malli on epävirallinen malli ja se on yleisessä käytössä, koska se on yksinkertainen. Kuvasta 4.2 nähdään koodaa ja korjaa -mallin rakenne. (McConnell 2002)

4.4 Klassinen vesiputousmalli

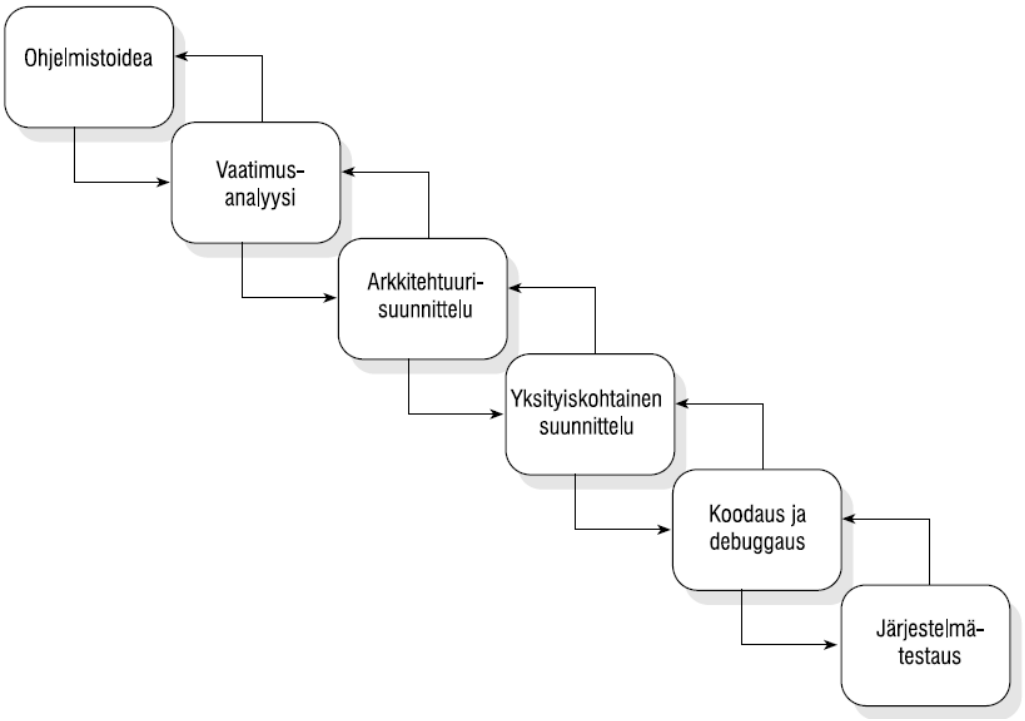
Vesiputousmalli on tunnetuin elinkaarimalli ohjelmistotuotannossa, ja se toimii myös pohjana useimmille kehittyneemmille elinkaarimalleille. Vesiputousmallissa ohjelmiston kehitys kulkee tiettyjen vaiheiden läpi ohjelmistoideasta järjestelmätestaukseen ja käyttöönottoon sekä lopulta ohjelmiston ylläpitoon. Vesiputousmallissa projektissa pidetään aina katselmus jokaisen vaiheen lopussa ennen kuin voidaan siirtyä seuraan vaiheeseen. Jos tarkastuksessa ilmenee, ettei nykyinen vaihe ole vielä valmis, pysyy projekti nykyisessä vaiheessa, kunnes kaikki siihen liittyvät tehtävät on suoritettu. (McConnell 2002)

Vesiputousmalli on dokumenttiohjautuva. Tämä tarkoittaa sitä, että jokaisessa vaiheessa syntyy työn tuloksena artifakteja eli ohjelmistodokumentteja. Kun puhutaan klassisesta tai niin sanotusta puhtaasta vesiputousmallista on huomioitava, etteivät siihen sisältyvät vaiheet ole päällekkäisiä eli ne ovat täten epäjatkuvia. (McConnell 2002)

Vesiputousmalli auttaa minimoimaan suunnitteluun kuluvan ylimääräisen ajan, koska kaikki suunnittelu tehdään ennen toteutusta. On myös huomattava, ettei konkreettisia tuloksia synny ennen kuin projekti etenee toteutusvaiheeseen. Mutta ne, jotka ovat tutustuneet projektiin, voivat pitää alkuvaiheiden tuottamia dokumentteja selvinä edistymisen merkkeinä. (McConnell 2002)

Kuvasta 4.3 nähdään klassisen vesiputousmallin kulku ja eri vaiheet sekä muutamia vaiheiden tuottamia ohjelmistodokumentteja. Klassinen vesiputousmalli lähtee liikkeelle ohjelmistoideasta ja etenee seuraavaksi vaatimusmäärittelyyn, arkkitehtuurisuunnitteluun, yksityiskohtaiseen suunnitteluun ja lopulta toteutuk-

seen. Kun toteutus on saatu valmiiksi, suoritetaan rakennetun ohjelmistotuotteen testaaminen. Jos virheitä löytyy, palataan takaisin toteutukseen ja virheet korjataan. Tämä pätee myös kaikkiin muihinkin vesiputousmallin vaiheisiin: jos virheitä havaitaan edellisen vaiheen tuottamissa dokumenteissa tai suunnitelmissa, on palattava edelliseen vaiheeseen korjaamaan virheet ennen kuin työ voi jatkua. Kun ohjelmisto tuote on testattu ja todettu riittävän virheettömäksi omassa kontekstissaan, suoritetaan käyttöönotto. Jokaiseen ohjelmistotuotteeseen liittyy myös ylläpidollisia tehtäviä ja piilevien virheiden korjausta myös käyttöönoton jälkeen. (McConnell 2002)



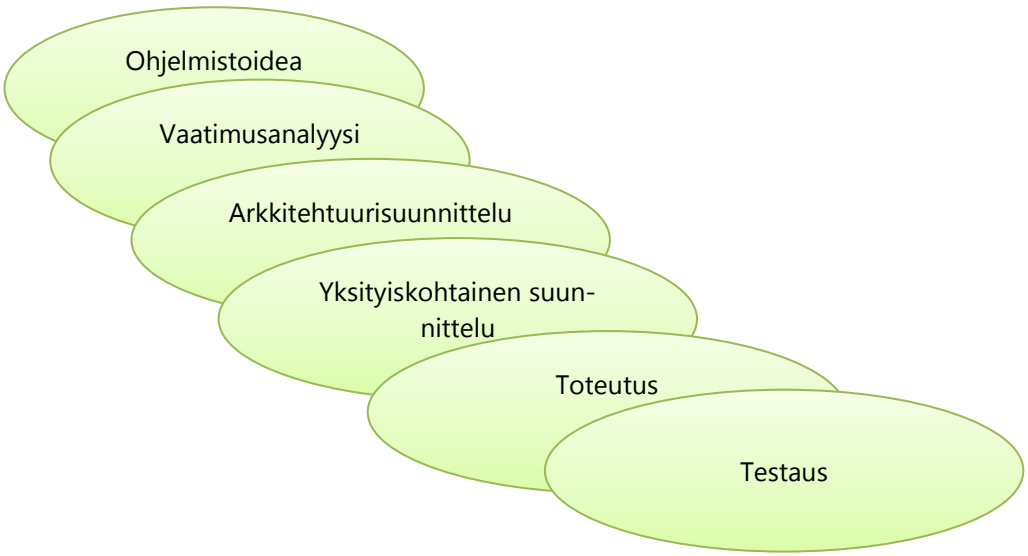
Kuva 4.3 Klassinen vesiputous -elinkaarimalli (McConnell 2002)

4.5 Muunnellut vesiputousmallit

Klassisessa vesiputousmallissa olevat vaiheet ovat sisäänrakennettuja osia ohjelmistokehityksessä eikä niiltä voida välttyä. Jostain on saatava ohjelmistolle vaatimukset, määritellylle ohjelmakonseptille on myös suunniteltava arkkitehtuuri sekä jaoteltava tarvittavat toiminnot osiin myötäillen arkkitehtuurisuunnitelmaa. (McConnell 2002)

Klassisen vesiputousmallin ongelmat eivät yleensä johdu vaiheissa olevista ongelmista vaan niiden käsittelystä erillisinä ja peräkkäisinä osioina. Näitä heikkouksia voidaan korjailla tekemällä vesiputousmallille pieniä muutoksia ja sovelluksia. Esimerkiksi limittämällä vaiheita osittain samanaikaisiksi, vähentämällä dokumentaatiopainotusta ja helpottamalla taaksepäin menemistä voi syntynyt lopputulos soveltua paremmin projektin tarpeisiin ja nopeuttaa ohjelmistokehitystä. (McConnell 2002)

Vesiputousmallin muunnelmia on useita ja seuraavaksi tutustutaan yhteen niistä. Vesiputous limittäisin vaihein on yksi muunnelluista vesiputousmalleista. Siinä vaiheiden päällekkäisyys mahdollistaa että voidaan olla esimerkiksi pitkällä arkkitehtuurisuunnittelussa ja aloittamassa yksityiskohtaista suunnittelua ennen kuin vaatimus analyysiä on saatettu täysin loppuun (ks. kuva 4.4).
(McConnell 2002)



Kuva 4.4 Limittäinen vesiputousmalli

Klassisessa vesiputousmallissa tietyn vaiheen tuottama dokumentti voidaan antaa suoraan toiselle ryhmälle, joka jatkaa työtä edellisen ryhmän tuotoksen perusteella ja tekee oman tuotoksensa omasta vaiheestaan, joka niin ikään voidaan antaa taas eri ryhmälle eteenpäin. Tästä herää kysymys, miksi pitäisi tehdä näin. Onko kyseinen tapa käytännöllinen, jos voidaan pitää esimerkiksi sama henkilöstö ja ryhmä koko projektin ajan työskentelemässä saman ohjelmisto-

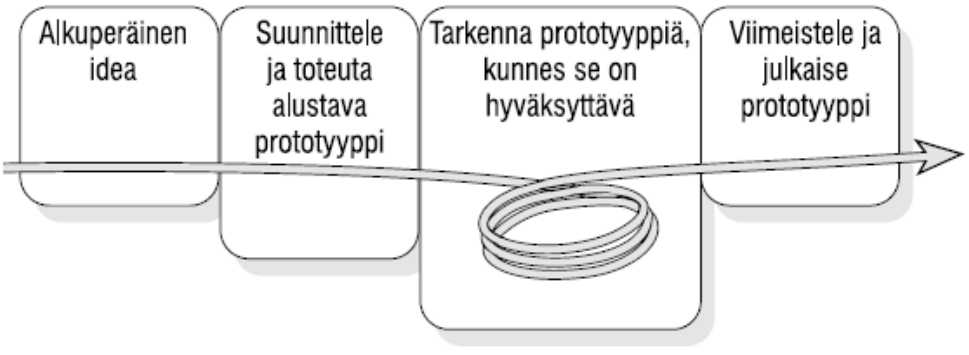
tuotteen ääressä? Tällöin ei välttämättä tarvita yhtä paljon dokumentaatiota ja vaiheita voidaan limittää. (McConnell 2002)

Limitetyssä vesiputousmallissa on myös omat ongelmansa, jotka liittyvät tarkastuspisteisiin, jotka ovat epämääräisempiä limittäisyyden ja päällekkäisyyksien takia. Edistymisen seuranta voi myös olla vaikeampaa ja tämä voi johtaa kommunikointivirheisiin ja vääriin olettamuksiin sekä tehottomuuteen.
(McConnell 2002.)

4.6 Inkrementaaliset mallit

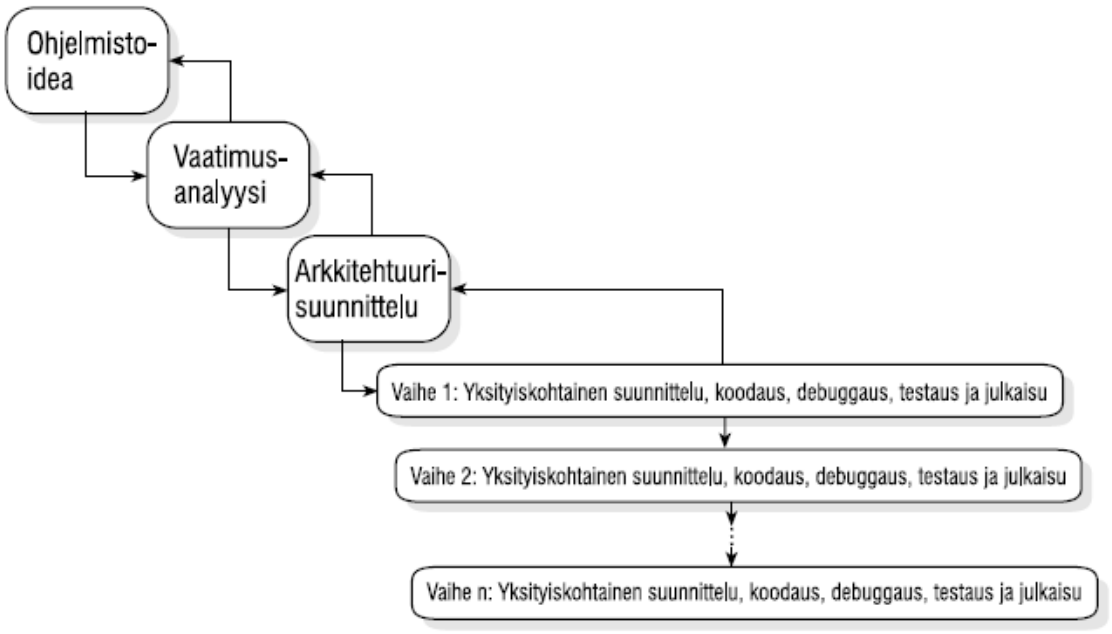
Klassisen ja muunneltujen vesiputousmallien lisäksi on muitakin yleisiä elinkaarimalleja. Yhteen ryhmään kuuluvat niin sanotut inkrementaaliset mallit, joita ovat esimerkiksi evolutiivinen protoilu, vaiheistettu toimitus, evolutiivinen toimitus, aikataulun mukainen suunnittelu ja spiraalimalli (McConnell 2002).

Evolutiivinen protoilu -mallissa (ks. kuva 4.5) järjestelmän konsepti kehitetään projektin edistyessä. Mallissa projekti lähtee liikkeelle alkuperäisestä ideasta ja ensimmäisenä vaiheena tehdään alustava prototyyppi, johon toteutetaan järjestelmän näkyvimmät piirteet. Prototyyppi demotaan asiakkaalle, jonka jälkeen prototyyppiä aletaan kehittää pienissä osissa ja useissa sykleissä. Jokaisen syklin päätteeksi tarkennettu prototyyppi demotaan asiakkaalle. Asiakas antaa palautetta ja seuraava sykli keskittyy täydentämään / korjaamaan prototyyppiä palautteen mukaisesti. Lopuksi kun asiakas toteaa prototyypin olevan "tarpeeksi hyvä", se viimeistellään ja julkaistaan. (McConnell, S. 2002)



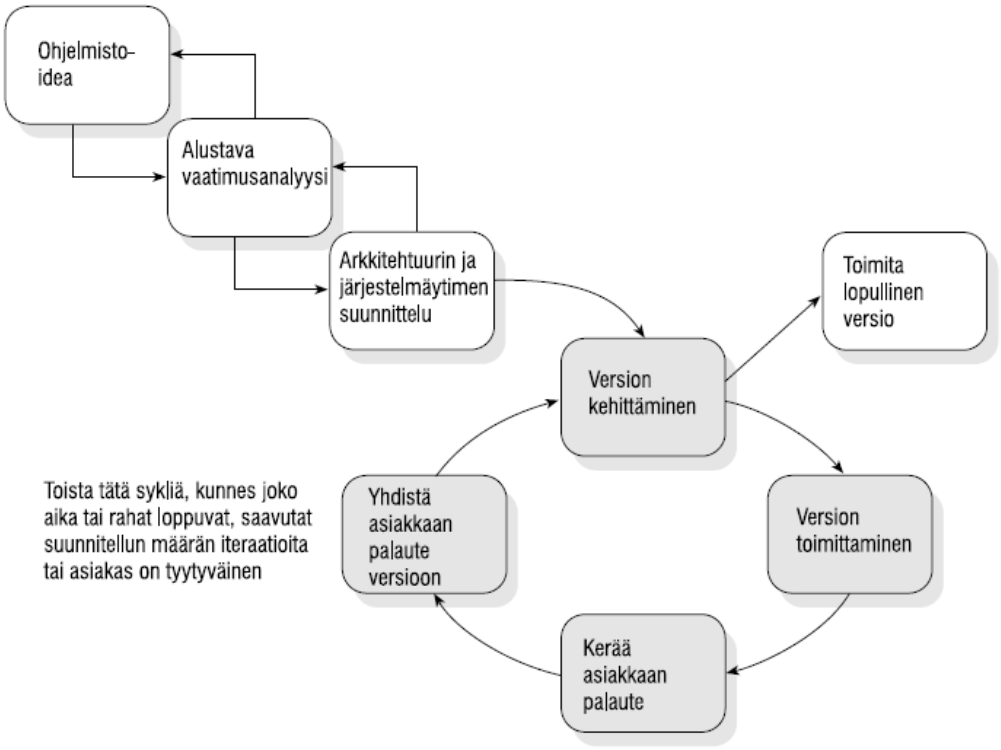
Kuva 4.5 Evolutiivinen protoilu -malli (McConnell 2002)

Vaiheistettu toimitus (ks. kuva 4.6) kulkee ohjelmistoideasta vaatimusanalyysiin ja arkkitehtuurisuunnitteluun vesiputousmallin mukaisesti, mutta yksityiskohtainen suunnittelu, toteutus ja testaus jaotellaan useampiin vaiheisiin. Toisin sanoen jokaisen vaiheen tuloksena on julkaistu ohjelma, joka toimitetaan asiakkaalle. Jokainen vaihe täydentää edellisen vaiheen tuotetta uusilla toiminnoilla. Lopullinen tuote siis toimitetaan asiakkaalle osissa projektin kuluessa.
(McConnell 2002)



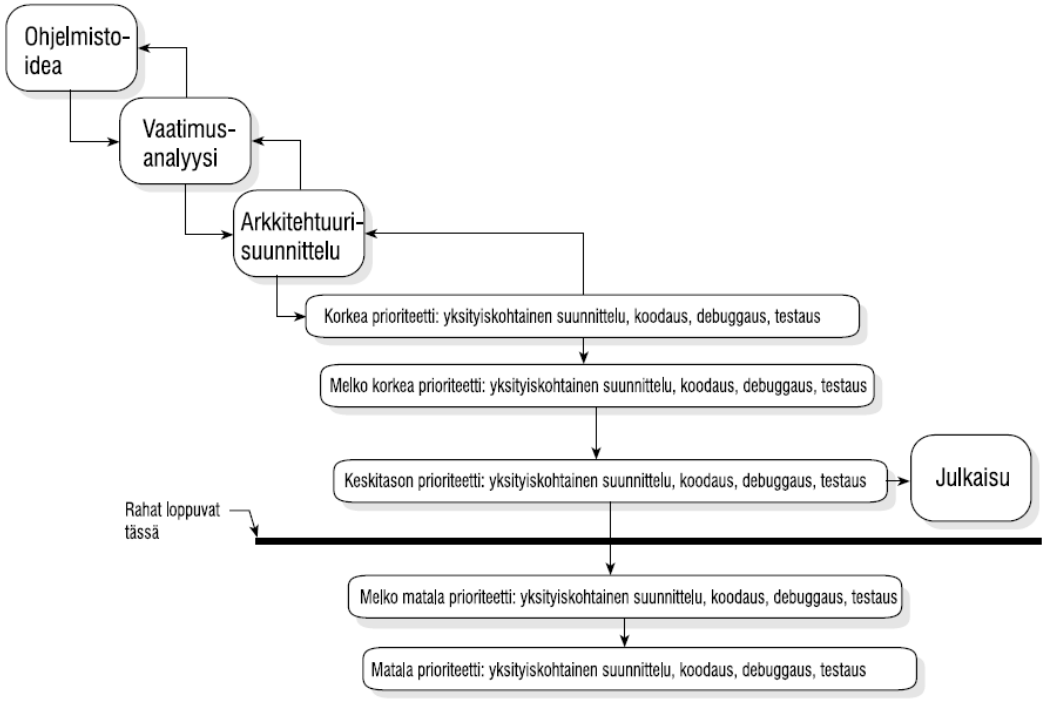
Kuva 4.6 Vaiheistettu toimitus -malli (McConnell 2002)

Evolutiivinen toimitus on elinkaarimalli (ks. kuva 4.7), joka tarjoaa välimallin evolutiiviselle protoilulle ja vaiheistetulle toimitukselle. Malli lähtee liikkeelle ohjelmistoideasta siirtyen alustavaan vaatimusanalyysiin ja tämän jälkeen arkkitehtuurisuunnitteluun ja järjestelmäytimen suunnitteluun. Seuraavaksi mennään sykleissä protoilua kunnes aika, rahat loppuvat tai suunniteltu iteraatioiden määrä täyttyy tai asiakas on tyytyväinen. Yhden iteraation pituus ja laajuus riippuu siitä, kuinka tarkasti asiakkaan palautteeseen reagoidaan. Kehitetäänkö kaikki vaatimukset joka syklillä vai käytetäänkö lyhyempiä syklejä, joihin valitaan vain tietyt ominaisuudet, jotka toteutetaan seuraavaan prototyyppiin?
(McConnell 2002)



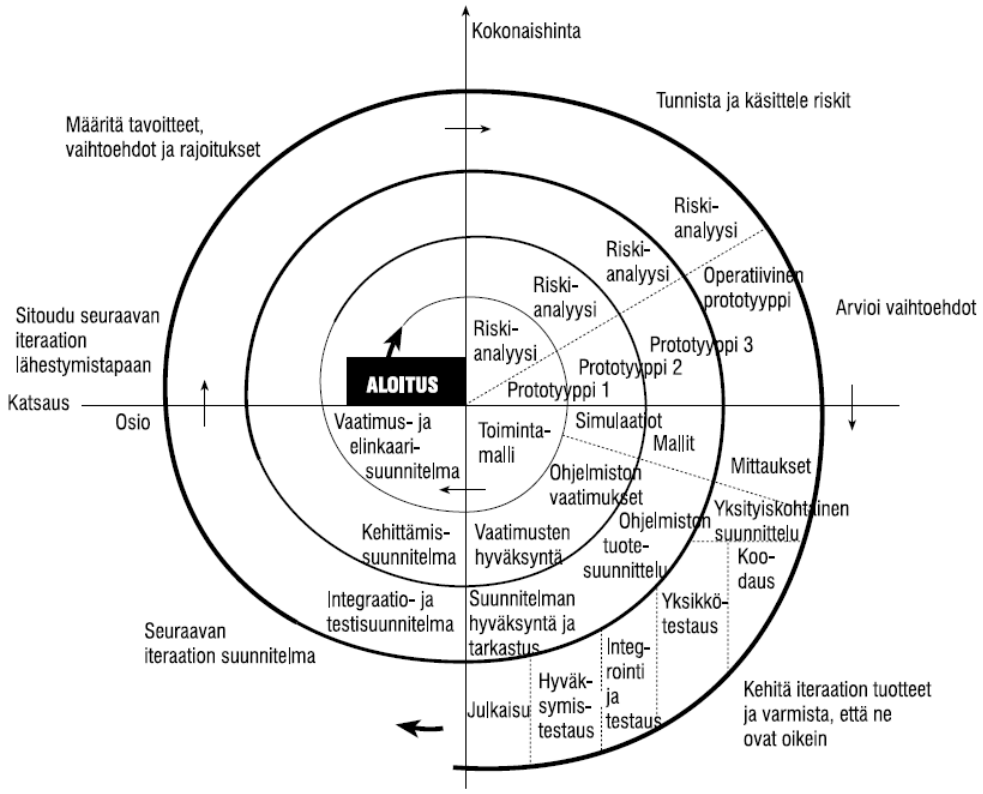
Kuva 4.7 Evolutiivinen toimitus -malli (McConnell 2002)

Aikataulun mukainen suunnittelu (ks. kuva 4.8) käyttää samaa kaavaa vaiheistetun toimituksen kanssa. Ainoana erona vaiheistettuun toimitukseen on, ettei alussa välttämättä tiedetä, päästäänkö viimeiseen suunniteltuun sykliin. Esimerkiksi suunnittelussa on sovittu tietty kiinteä aika tai päivämäärä, johon mennessä lopullinen tuote on julkaistava. Jaotellaan tuote vaikkapa 5 osaan ja ruvetaan kehittämään. Lopulta huomataan, että päästiin 4 vaiheen loppuun, kunnes aikaraja tuli vastaan. Tätä mallia tai strategiaa käyttämällä voidaan varmistaa, että tiettyyn päivämäärään mennessä on julkaisukelpoinen tuote valmiina. Tämä elinkaari malli on myös erittäin hyödyllinen niihin tuotteen osiin, joita ei haluta kriittiselle polulle projektisuunnitelmassa. (McConnell 2002)



Kuva 4.8 Aikataulun mukainen toimitus -malli (McConnell 2002)

Spiraalimalli suuntautuu riskeihin. Se pilkkoo ohjelmistoprojektin useammiksi pieniksi projekteiksi. Jokainen pienempi projekti keskittyy yhteen tai useampaan riskiin, kunnes pääasialliset riskit on hoidettu. Tämän jälkeen spiraalimalli päättyy kuten vesiputousmallikin. Spiraalimallin kaavakuva (kuva 4.5) on monimutkainen, mutta sen perusideana on aloittaa pienellä mittakaavalla keskustassa ja edetä inkrementaalisesti laajentaen mittakaavaa. Kuvasta 4.5 voidaan tutustua tarkemmin spiraalimallin kaavaan. (McConnell 2002)



Kuva 4.9 Spiraalimalli (McConnell 2002)

4.7 RUP-malli

RUP eli Rational Unified Process perustuu pitkälti kolmen tunnetuimman metodologian kehittäjän, Ivar Jacobsonin, Grady Boochin ja James Rumbaugh'n koostettuun työhön ja sen tuottamaan ohjelmistokehittämisen metodologiaan. Edellä mainitut kolme kehittäjää ja laaja kehittäjien yhteisö liittyivät yhteistyöhön Rational Corporationin kanssa muodostaen yhdistetyn, yhtenäisen ja kokonaisvaltaisen kehyksen tietojärjestelmien ohjelmistokehittämiseen. Ryhmän työ perustui aiempien ja testattujen metodologioiden yhdistämiseen ja standardien muodostamiseen, joihin kuuluvat Case-tekniikat ja UML-kieli.
(The Menlo Institute LLC, 2002)

RUP sisältää kolme yksilöivää ominaisuutta:

1. Käyttötapauslähtöinen
2. Arkkitehtuurikeskeinen
3. Iteroiva ja lisäävä prosessi

RUP koostuu iteraatioista, jotka voivat toistua useasti järjestelmän elinkaaren aikana. Iteraatio koostuu neljästä perusvaiheesta, jotka ovat aloitus-, kehittely-, rakennus- ja siirtymävaihe. (The Menlo Institute LLC, 2002)

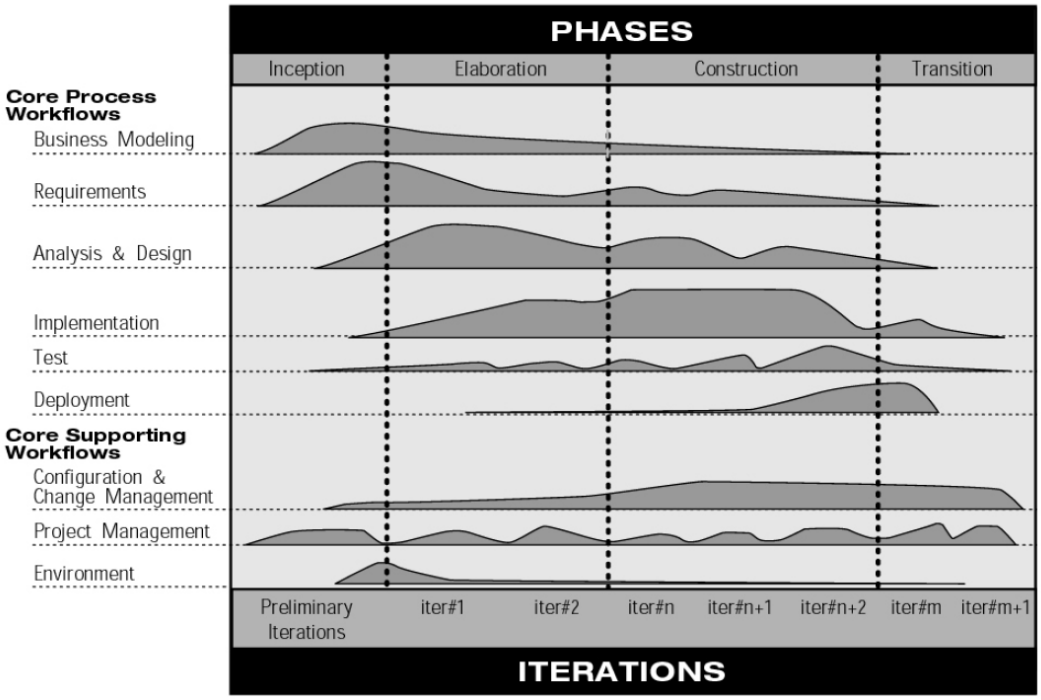
Aloitusvaiheen aikana keskitytään määrittämään projektin laajuus, kuvaamaan pahimmat riskit, alustavat liiketoimintamallit ja pohtimaan, onko projektin suorittamiselle edellytyksiä liiketoiminnan kannalta. Aloitusvaiheen aikana tehdään myös arviot hinnasta, tuotosta ja aikataulusta. Projektille kerätään myös perustiedot tulevan järjestelmän arkkitehtuurista ja sovellusalueesta.
(Kampman 2001)

Kehittelyvaiheessa määritellään projektin perusarkkitehtuuri ja toiminnallisuus. Suurin osa vaatimuksista on tässä vaiheessa selvillä, ja rakennettava järjestelmä analysoidaan ja suunnitellaan niiden pohjalta. Toteutettavasta järjestelmästä voidaan tehdä useita prototyyppejä ja toteutusympäristö valitaan kehittelyvaiheen aikana. (Kampman 2001)

Rakennusvaiheessa tuleva järjestelmä toteutetaan määritellyn perusarkkitehtuurin perusteella. Järjestelmää toteutetaan suorittamalla iteraatioita, jotka sisältävät analyysi-, suunnittelu-, toteutus- ja testausvaiheet. Rakennusvaiheen lopussa kehitettävä järjestelmä on valmiina beta-testaukseen. (Kampman 2001)

Siirtymävaiheessa järjestelmä valmistellaan toimituskuntoon, jotta se voidaan toimittaa asiakkaalle dokumentteineen ja koulutuksineen. Järjestelmä asennetaan ja otetaan käyttöön tuotantoympäristössä. Järjestelmä myös versioidaan siirtymävaiheen aikana. Tällöin ei enää tehdä mitään suuria muutoksia vaan pelkästään pieniä hienosäätöjä käyttäjäkokemuksien perusteella.
(Kampman 2001)

Kuvassa 4.10 on pystyakselilla RUP:n kehitysprosessin työnkulut ja vaaka-akselilla vaiheet. Kaaviossa on myös kuvattu, kuinka ajankäyttö painottuu eri osa-alueille ja myös kuinka iteraatiot liittyvät kokonaisuuteen.



Kuva 4.10 Rational Unified Prosess:n työnkulut ja vaiheet (Rational Software, 2001)

Rational Unified Process sisältää yhdeksän perustyönkulkuprosessia, jotka esittävät työntekijöiden ja aktiviteettien jaottelun loogisiin ryhmiin kiinnostusalueiden tai alojen mukaisesti. Jokainen työnkulku sisältää esittelyn, kaaviokuvauksen, siihen kuuluvat aktiviteetit, artifaktit, suoritusohjeet ja työnkulkuun liittyvien käsitteiden kuvaukset. (Rational Software, 2001)

Perustyönkulkuprosessit on jaoteltu kuuteen tekniseen työnkulkuun ja kolmeen avustavaan työnkulkuun.

Ensimmäinen tekninen työnkulku on liiketoiminnan mallintaminen, joka keskittyy kohdeorganisaation liiketoiminnallisten prosessien ja organisaatiorakenteen selvittämiseen. Tarkoituksena on ymmärtää niiden ongelmat ja yrittää löytää niihin parannusehdotuksia. Tärkeää on, että kaikki projektiin osallistujat ovat mukana loppukäyttäjistä, suunnittelijoihin ja projektipäällikköön. Osallistujien on ymmärrettävä, kuinka kohdeorganisaatio toimii. (Kampman 2001)

Seuraava työnkulku keskittyy vaatimusten määrittelemiseen ja tavoitteena on saada kaikkien projektiin osallistujien välille selkeä yhteisymmärrys, siitä mitä kehitettävän järjestelmän tulisi tehdä. Järjestelmän kehittäjät saavat tietoa vaatimuksista, joita järjestelmälle asetetaan. Samalla saadaan käsitys siitä, mikä kuuluu järjestelmään ja mikä ei. Työnkulussa pyritään myös saamaan peruskäsitys iterointien suorittamisesta ja arviot projektin kustannuksista, aikataulusta ja tarvittavista resursseista. (Kampman 2001)

Analyysi ja suunnittelu -työnkulku kuvaa, miten järjestelmä toimii. Järjestelmän on täytettävä kaikki käyttötapauksissa määritellyt tehtävät ja vaatimusmäärittelyssä esitetyt vaatimukset. Tämä työnkulku toimii suunnitelmana toteutukselle, jonka pohjalta kehitettävä järjestelmä pystytään toteuttamaan.
(Kampman 2001)

Toteutuksessa järjestelmä rakennetaan toteutettavista komponenteista. Komponentit yksikkötestataan toteutusvaiheen aikana. Järjestelmän toteutuksessa käytetään myös usein hyväksi aikaisemmin toteutettuja komponentteja. Kaikki toteutuksessa syntyvät uudet komponentit suunnitellaan ja rakennetaan ajatellen niiden mahdollista uudelleen käyttöä seuraavissa projekteissa.
(Kampman 2001)

Testaus-työnkulun aikana testataan, kuinka järjestelmän oliot kommunikoivat keskenään ja että komponentit toimivat oikein yhdessä. Testauksessa myös tarkistetaan, että kaikki järjestelmälle asetetut vaatimukset on toteutettu oikein. Testauksella on kolme laadullista ulottuvuutta: luotettavuus, toiminnallisuus ja järjestelmän suorituskyky. (Kampman 2001)

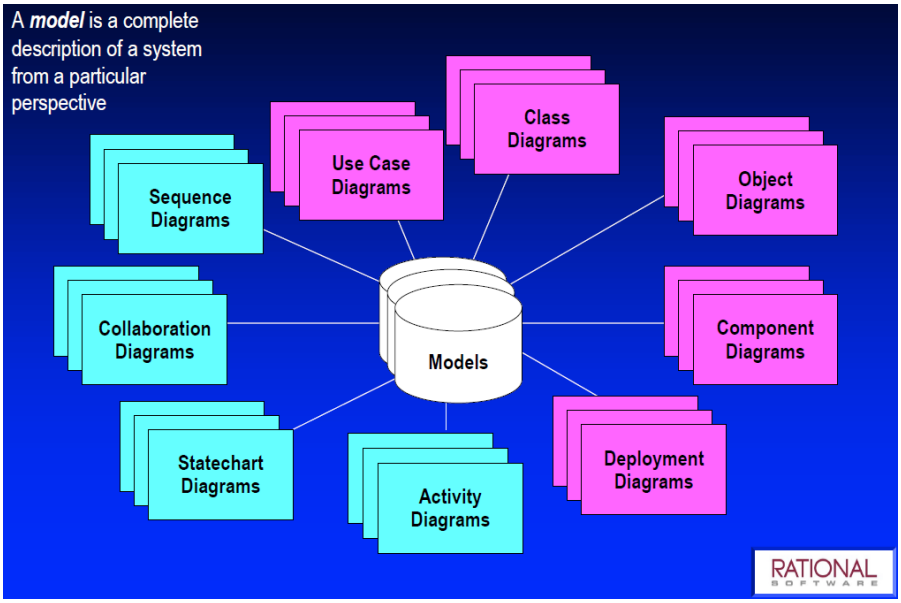
Seuraavaksi on vuorossa sijoittelu, joka on viimeinen tekninen työnkulku. Sen aikana valmistellaan järjestelmästä versioitu tuote, joka toimitetaan loppukäyttäjille. Versioitu tuote pakataan ja jaetaan asiakkaille. Se voidaan tarvittaessa asentaa tuotantoympäristöön. Käyttöönotossa tarjotaan apua järjestelmän loppukäyttäjille. Sijoittelu-työnkulku voi sisältää myös järjestelmän beta-testauksen ja muodollisen hyväksymistestauksen. (Kampman 2001)

Ensimmäinen avustava työnkulku on konfigurointi ja muutosten hallinta. Avustavien työnkulkujen tarkoituksena on tukea ohjelmiston kehitystä. Konfigurointi ja muutosten hallinta kuvaa toiminnot, joiden avulla artifaktit saadaan pysymään järjestyksessä. Se varmistaa myös, ettei kukaan tee artifakteihin päällekkäisiä päivityksiä ja että kaikki osalliset ovat tietoisia tehdyistä muutoksista.
(Kampman 2001)

Seuraavana on projektin hallinta, sen tarkoituksena on auttaa hallitsemaan riskejä, jotta ohjelmisto voidaan toimittaa onnistuneesti asiakkaalle. Projektin hallinta keskittyy pääasiassa ohjelmiston näkökulmaan eikä ota kantaa budjettiin, sopimuksiin tai työntekijöiden hallintaan. (Kampman 2001)

Ympäristö-työnkulku on viimeinen avustava työnkulku RUP:ssa ja sen tehtävänä on varmistaa kehitysorganisaation käyttämä kehitysympäristö prosesseineen ja työkaluineen, joiden avulla projektiorganisaatio pystyy toteuttamaan projektin onnistuneesti. Se kuvaa myös tehtävät, jotka on suoritettava projektin ohjeistuksen suunnittelussa. Ympäristö-työnkulku sisältää myös mahdollisesti tarvittavan prosessin muokkauksen. (Kampman 2001)

RUP:ssa käyttetään kaikissa työnkuluissa ja vaiheissaan hyväksi UML-kieltä. Kuvassa 4.11 on esitelty UML:n tarjoamat kaaviot, joita RUP:n vaiheissa esiintyy.

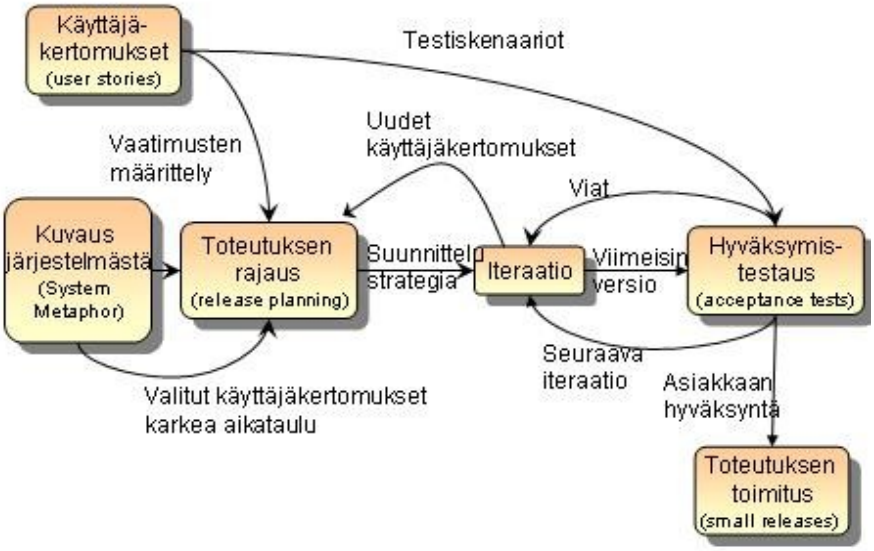


Kuva 4.11 Rational Unified Process -mallit ja UML-kaaviot (Jacobson)

4.8 Ketterät mallit

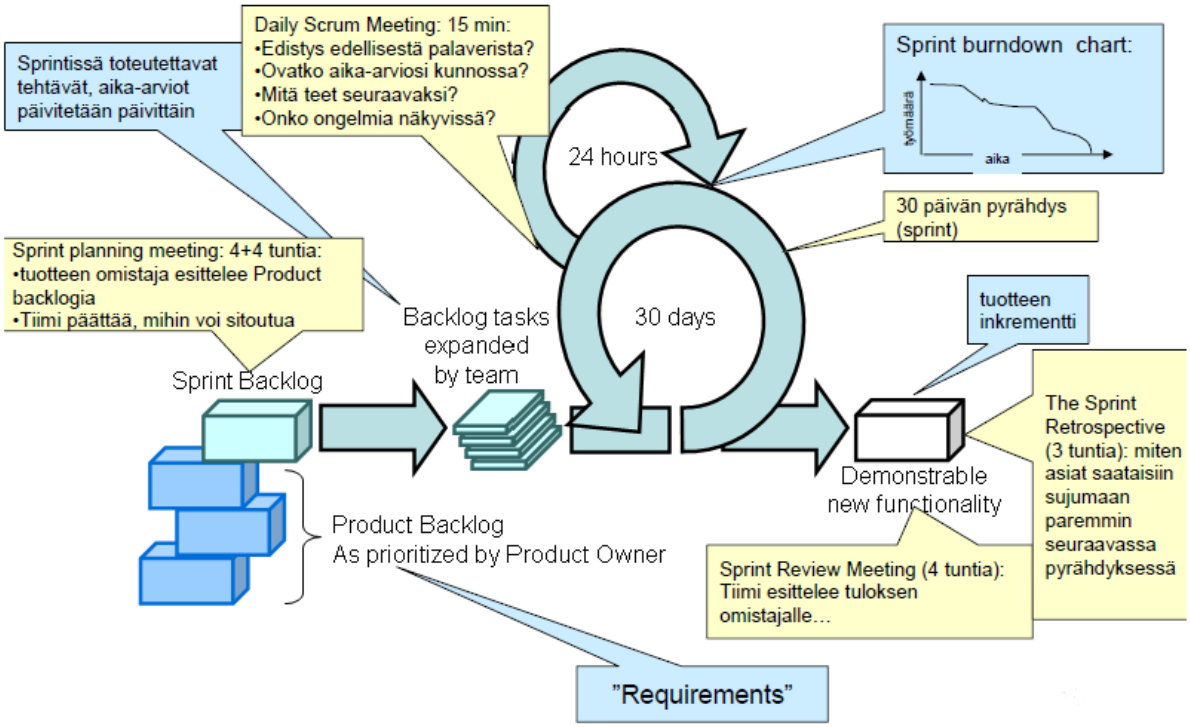
Ketterät menetelmät perustuvat iteratiivisiin ja inkrementaalisiin elinkaarimalleihin. Se yhdistää kummatkin näkemykset tarkoittaen inkrementilla pientä toiminnallisuuden palaa tai osaa, joka on tarvittaessa käyttöönotettavissa. Iteraatio taas tarkoittaa yhtä sykliä, jonka aikana edellä mainittu yksi inkrementti valmistetaan. On huomattava, ettei inkrementin valmistumisaika välttämättä ole samanmittainen kuin iteraatio. (Ketterät käytännöt)

Extreme Programming eli XP (ks. kuva 4.12) on yksi tunnetuimmista ketterän kehityksen elinkaarimalleista. Se on nimensä mukaisesti hyvin ohjelmointikeskeinen ja poikkeaa muista ketteristä menetelmistä tarjoamalla valikoidun joukon käytäntöjä, joita noudatetaan yhdessä, sillä ne on rakennettu tasapainottamaan toisiaan. XP ei ole pelkästään käytäntökokoelma vaan se sisältää myös joukon periaatteita ja omia arvoja. Pääpaino onkin toteutuskäytännöissä, mutta XP sisältää myös jonkin verran projektinhallintaan liittyviä käytäntöjä.
(Ketterät käytännöt)



Kuva 4.12 XP-menetelmä (Ilkko 2008)

Scrum on toinen tunnettu ketterän sovelluskehityksen elinkaarimalli, joka tarjoaa käytännöt, joiden mukaan projektia ohjataan. Se ei ota kantaa matalan tason insinöörikäytäntöihin, vaan keskittyy ennen kaikkea projektin vaiheistamiseen ja jatkuvaan kontrolliin projektin etenemisestä. Scrum-mallissa ohjelmistokehitys rakentuu erimittaisten syklien eli iteraatioiden ympärille. Tärkeimmät syklit Scrumissa ovat sprintti ja päivä. Sprintillä tarkoitetaan yhtä kehitysjaksoa, jonka jälkeen tuote on ainakin periaatteessa julkaisuvalmis. Tyypillisesti sprintin kesto on yksi kuukausi, mutta sen pituus voi vaihdella organisaation tarpeiden mukaan viikosta aina kahteen kuukauteen saakka. (Ketterät käytännöt)



Kuva 4.13 Scrum-menetelmä (Haikala 2009)

5 TYÖSSÄ KÄYTETYT MENETELMÄT

Tässä luvussa käydään läpi opinnäytetyössä käytetyt pääasialliset menetelmät. Ensimmäisenä on esitutkimus, joka aloittaa ohjelmiston kehittämisen ja seuraavaksi tarkastellaan toiminnallista määrittelyä, jossa tarkennetaan ja dokumentoidaan esitutkimuksessa saatuja vaatimuksia.

5.1 Esitutkimus

Ohjelmiston kehitys lähtee liikkeelle usein yhden ihmisen näkemyksestä tai ideasta. Näkemyksen on oltava selkeä ja vakaalla perustalla, jotta tuotteesta tulee oikeanlainen.

Ohjelmistoprojektin alkuvaiheessa näkemyksen omaavan henkilön on pidettävä kiinni näkemyksestään ja esiteltävä se selkeästi eri osapuolille. Ohjausryhmässä voi olla teknisiä asiantuntijoita ja ohjelmistoanalyytikkoja, joiden tehtävänä on auttaa näkemyksen omaavaa henkilöä ymmärtämään ideaa tarkemmin ja samalla tuoda esiin idean ydin asiat, rajoitteet ja tavoiteltava laatutaso. Jos näkemyksen omaava henkilö ei ymmärrä ideansa todellista luonnetta ja ydintä, voi hanke epäonnistua.

Esitutkimuksen tarkoituksena on saada selville projektin tavoitteet sekä kartoittaa onko idean toteuttamiselle riittävästi liiketoiminnallisia edellytyksiä. Esitutkimus tuottaa esitutkimusraportin, jonka sisällysluettelo on kuvassa 5.1.

SISÄLLYSLUETTELO

Kuva 5.1 Esitutkimusraportin sisältöluettelo (TTY, Ohjelmistotekniikanlaitos)

5.2 Toiminnallinen määrittely

Toiminnallisessa määrittelyssä lähdetään liikkeelle alustavien vaatimusten analysoinnista. Vaatimusten pohjalta erotellaan liiketoimintaprosessit ja kehitetään niitä kuvaava liiketoimintamalli. Liiketoimintamalli on kuvattava tarkasti ja yksityiskohtaisesti käyttäen siihen soveltuvia työkaluja.

Liiketoiminta-alueen käsitteiden täytyy olla selkeästi ohjelmistoidean keksineen henkilön tunnistettavissa. Niiden tunnistamiseen ei tarvita minkäänlaista tietotekniikan osaamista. Liiketoiminta-alueelta on tunnistettava kaikki ohjelmistotuotteeseen liittyvät käsitteet sekä niiden yhteydet ja suhteet toisiinsa sekä käsitteille ominaiset käyttäytymismallit.

Ohjelmistotuotteen vaatimukset kootaan yhteen dokumenttiin, jossa kuvataan kaikki järjestelmän toiminnot, tiedonkäsittelyn ja ulkoiset liittymät. Edellä mainitut asiat kuvataan aina loppukäyttäjän näkökulmasta. Tällöin lukija saa kuvan siitä, mitä ohjelmistotuotteella voidaan tehdä, miten se toimii ja miten se ei saa toimia. Dokumentissa ei oteta kantaa itse ohjelmistotuotteen toteutukseen.

Kuvassa 5.2 on toiminnallisen määrittelyn sisällysluettelo. Toiminnallinen määrittelydokumentti on keskeisin sopimusdokumentti ohjelmistonkehittäjien ja asiakkaan välillä. Molempien osapuolien on todettava dokumentti katselmointien ja tarkastuksien avulla virheettömäksi ja käyttökelpoiseksi. Toiminnallista määrittelydokumenttia päivitetään ja ylläpidetään koko ohjelmistoprojektin elinkaaren ajan.

SISÄLLYSLUETTELO

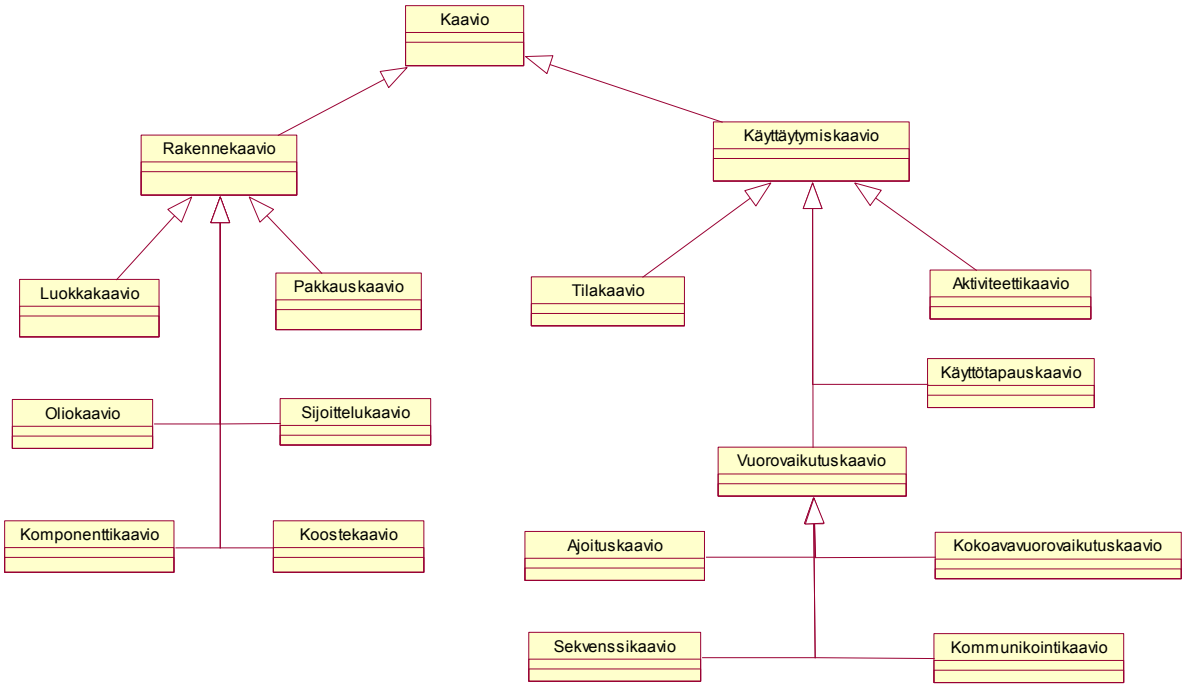
Kuva 5.2 Toiminnallisen määrittelyn sisällysluettelo (TTY, Ohjelmistotekniikan-laitos)

5.3 UML

UML (Unified Modeling Language) on graafinen ohjelmistojen analysointiin ja mallintamiseen tarkoitettu standardoitu kuvauskieli. UML-kielen ensimmäinen versio (1.0) standardoitiin vuonna 1997 ja se kehitettiin kolmesta tuolloin vallinneesta oliosuunnitteluun perustuneesta menetelmästä. Sen kehitykseen vaikuttivat Jacobsonin (OOSE), Boochin (BOOCH) ja Rumbaughin (OMT) kehittämät oliosuunnittelumenetelmät ja lisäksi myös OOA/D, Fusion ja David Harelin tilakartat. UML:n perustana olivat kuitenkin BOOCH-, OOSE- ja OMT-menetelmät. (Kylä-Nikkilä 2008)

UML:n kehittämisen aloittivat Grady Booch ja James Rumbauch Rational Softwarella vuonna 1994. Tavoitteena oli kehittää uusi menetelmä, joka yhdistäisi Boochin menetelmän ja OMT2:n. Myöhemmin vuonna 1995 Rational Softwaren ostaessa ruotsalaisen Objective Softwaren mukaan liittyi Ivar Jacobson. Jacobson tunnettiin OOSE:n ja Objectoryn kehittäjänä. Ensimmäiset versiot UML-kielestä julkaistiin 1990-luvun loppupuolella ja seuraavaksi kieltä ehdotettiin standardiksi. Versio 1.0 hyväksyttiin myöhemmin standardiksi loppuvuodesta 1997. Nykyisin UML-standardin kehittäminen ja ylläpito on OMG:n vastuulla. (Kylä-Nikkilä 2008 ; Eriksson & Penker 2001)

UML perustuu erilaisiin kaavioihin, jotka kuvaavat kyseessä olevan näkymän sisältöä erilaisten mallinnuselementtien avulla. Mallinnuselementit vastaavat yleisiä olio-ohjelmointiin pohjautuvia käsitteitä, kuten luokkia, olioita ja viestejä. Mallinnuselementtien väliset yhteydet kuvataan assosiaatioina, riippuvuuksina ja yleistyksinä. UML:ssä ei kuvata tarkasti mallinnuselementtien ulkoasua, mutta elementteinä käytetyt kuviot ja symbolit ovat vakiintuneet hyvin samankaltaisiksi. UML-kielen 2.2 versiossa on 13 erilaista kaaviotyyppiä ohjelmistokehityksen eri vaiheisiin (Kuva 5.3, Taulukko 5.1). (Kylä-Nikkilä 2008)



Kuva 5.3 UML-kaaviohierarkia (Kylä-Nikkilä 2008)

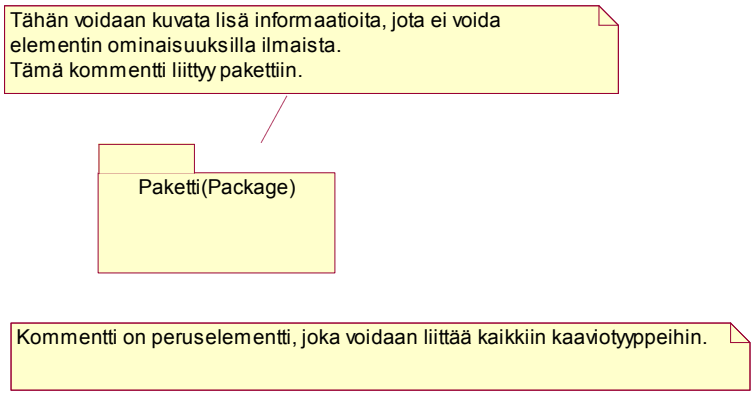
Taulukko 5.1 UML-kaavioiden päätyypit

Rakennekaaviot	**Käyttäytymiskaaviot**	**Vuorovaikutuskaaviot**
Luokkakaavio	Aktiviteettikaavio	Ajoituskaavio
Oliokaavio	Käyttötapauskaavio	Sekvenssikaavio
Komponenttikaavio	Tilakaavio	Kommunikointikaavio
Koostekaavio		Kokoava vuorovaikutuskaavio
Pakkauskaavio		
Sijoittelukaavio		

5.3.1 Peruselementit

UML-kieli sisältää vain muutaman peruselementin, joita voidaan käyttää kaikissa kaaviotyypeissä. Peruselementtien lisäksi kaavioissa käytettävät symbolit ovat kulloinkin käytettävälle kaaviotyypille spesifisiä. (Kylä-Nikkilä 2008)

Pakkaus (ks. kuva 5.2) on ryhmittelyssä apuna käytettävä elementti, jota harvoin toteutetaan varsinaiseen järjestelmään. Pakkaus voi sisältää samaan asiaan kuuluvia mallinnuselementtejä ja paketin sisällä oleva elementti voi kuulua vain yhteen pakkaukseen kerrallaan. (Kylä-Nikkilä 2008)



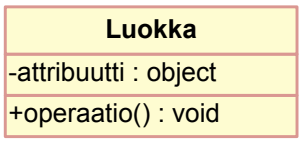
Kuva 5.4 Peruselementit

Kommentti (ks. kuva 5.2) on yleinen elementti, jota voidaan käyttää kaikissa kaaviotyypeissä. Se sisältää vapaamuotoista tekstiä, lisähuomioita tai infoa, jonka tekijä voi kohdistaa yhdiste viivan avulla tiettyyn elementtiin. Jättämällä yhdisteviiva pois kohdistuu kommentti tällöin kokokaavioon. Kommenteilla pyritään viestimään sellaista tietoa, jota kaaviotyyppiin liittyvillä elementeillä ei yksin pystytä esittämään.

5.3.2 Luokkakaavio

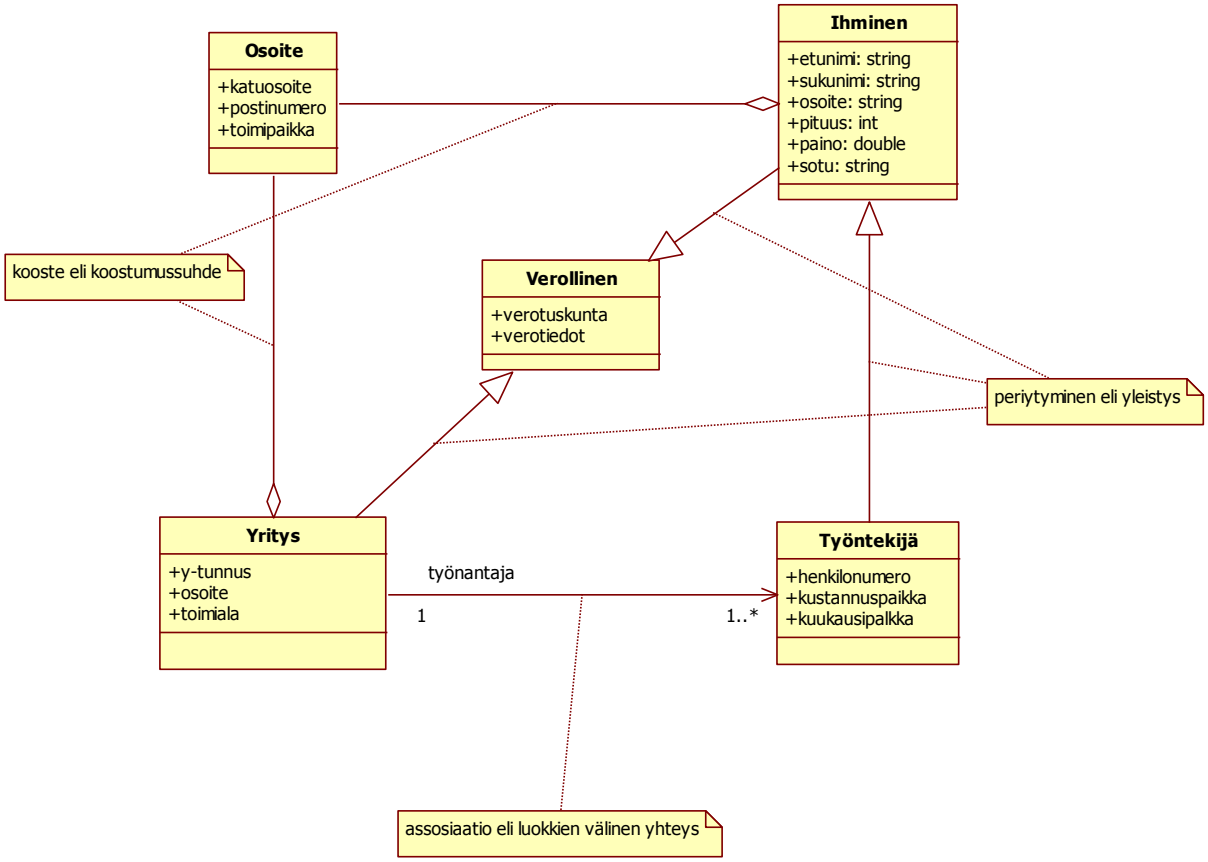
Luokkakaavio (class diagram) on staattinen mallinnustyyppi. Se kuvaa järjestelmän pysyvän rakenteen luokkien ja niiden välisten suhteiden avulla. Luokkakaavio muistuttaa tietomallia (data model), mutta se eroaa siitä merkittävästi siinä, että se näyttää tiedon rakenteen lisäksi myös käyttäytymisen.
(Eriksson & Penker 2001)

Luokkakaavion tehtävänä on määrittää perusta muille kaaviotyypeille, jotka kuvaavat järjestelmän muita ominaisuuksia. Luokkakaavio sisältää vain luokkia, jotka voidaan toteuttaa oliosuuntautuneella ohjelmointikielellä. Luokkakaaviosta on myös olemassa muunnos, oliokaavio, jossa kuvataan luokista luotuja ilmentymiä ja niiden toimintaa sekä vuorovaikutusta keskenään.
(Eriksson & Penker 2001)



Kuva 5.5 Luokan rakenne

Luokkakaavion teossa pyritään tunnistamaan ja kuvaamaan järjestelmään toteutettavat luokat ja niiden väliset suhteet. Luokkien tunnistaminen ja etsintä on luova urakka, jonka tuloksena saadaan joukko elävää elämää kuvaavia kokonaisuuksia, joilla on tietyt tiedot, tila, toiminta ja vastuualue suunniteltavassa järjestelmässä. (Eriksson & Penker 2001)

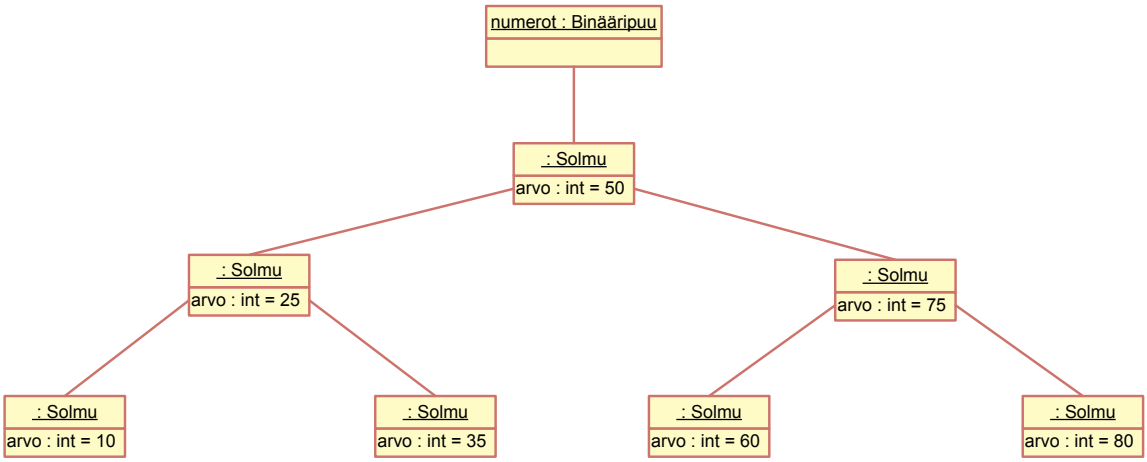


Kuva 5.6 Luokkakaavioesimerkki

5.3.3 Oliokaavio

Oliokaavio (object diagram) kuvaa järjestelmässä esiintyviä olioita tietyllä ajanhetkellä. Oliokaaviossa näkyy luokkien sijasta luokkien ilmentymiä eli olioita. Oliokaavio eroaa luokkakaaviosta siinä, että kaaviossa voi esiintyä yhdestä luokasta useita ilmentymiä, luokkakaaviossa luokka esiintyy vain kerran.
(Kylä-Nikkilä 2008)

Luokkakaavion ja oliokaavion merkinnät ovat muuten samat, mutta oliokaaviossa ilmentymän eli olion nimi ja luokan nimi kirjoitetaan kaksoispisteellä eroteltuna ja alleviivattuna. Olion ollessa nimetön jätetään nimi kirjoittamatta kaavioon. Oliokaaviossa voidaan myös tarvittaessa kuvata attribuuttien arvoja.
(Kylä-Nikkilä 2008)

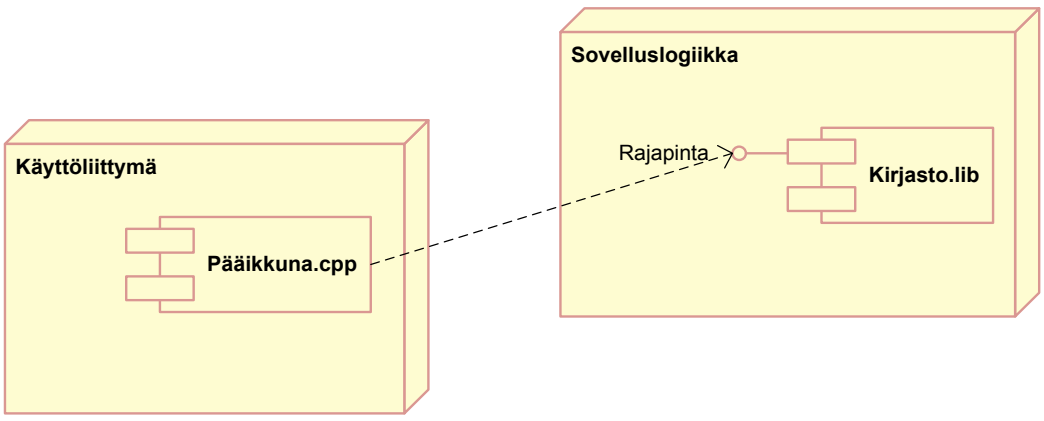


Kuva 5.7 Esimerkki tasapainotetusta binääripuusta (Kylä-Nikkilä 2008, muunneltu)

5.3.4 Komponenttikaavio

Komponenttikaavio (component diagram) kuvaa kaikki eri komponentit ja niiden väliset suhteet, joista järjestelmä rakentuu. Komponentti kuvaa järjestelmän itsenäistä osiota, joka voi olla suhteessa muihin järjestelmän komponentteihin. Komponentti on fyysisien toteutuksien joukko kaavioissa määritellyistä käsitteistä, kuten luokista. (Kylä-Nikkilä 2008)

Komponenttikaaviossa kuvataan lähdekoodin fyysinen rakenne ja sijoittelu. Sen avulla parannetaan muutosten hallintaa, sillä kaaviosta nähdään, mihin komponenttiin tehdyt muutokset vaikuttavat. Komponentteja voidaan myös tarvittaessa jaotella paketteihin organisoinnin helpottamiseksi. (Kylä-Nikkilä 2008)

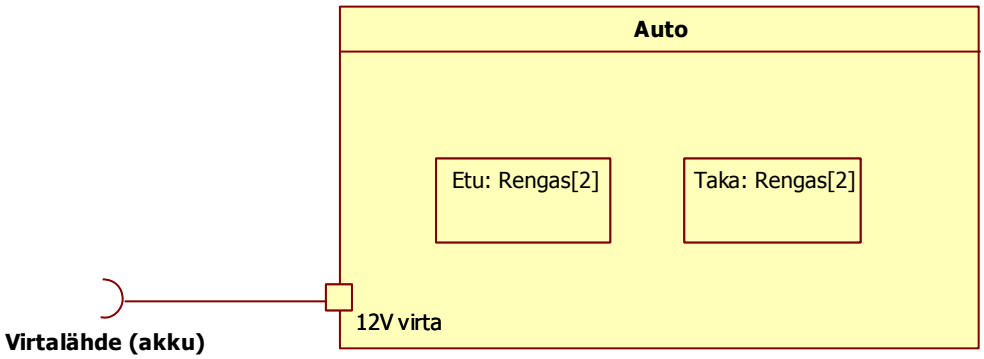


Kuva 5.8 Esimerkki komponenttikaaviosta (Kylä-Nikkilä 2008)

5.3.5 Koostekaavio

Koostekaavioita (composite structure diagram) käytetään monimutkaisten järjestelmien rakenteen kuvaamiseen. Sen avulla voidaan kuvata komponenttien ja luokkien sisäistä rakennetta, komponenttien välisiä riippuvuuksia ja kontekstiin sidottujen luokkien ja olioiden assosiaatioita silloin, kun niillä on merkitystä kokonaisuuden ja toiminnallisuuden kannalta. (Kylä-Nikkilä 2008)

Koostekaaviot täydentävät olio- ja luokkakaavioita sekä niiden avulla voidaan kuvata järjestelmän komponenttien suorittamia tehtäviä yhteistyössä toisten komponenttien kanssa. Koostekaavioita voidaan käyttää myös kuvaamaan järjestelmän suunnittelua ja arkkitehtuuria. Kuvassa 5.9 on esitetty autonrakennetta koostekaavion avulla. (Kylä-Nikkilä 2008)

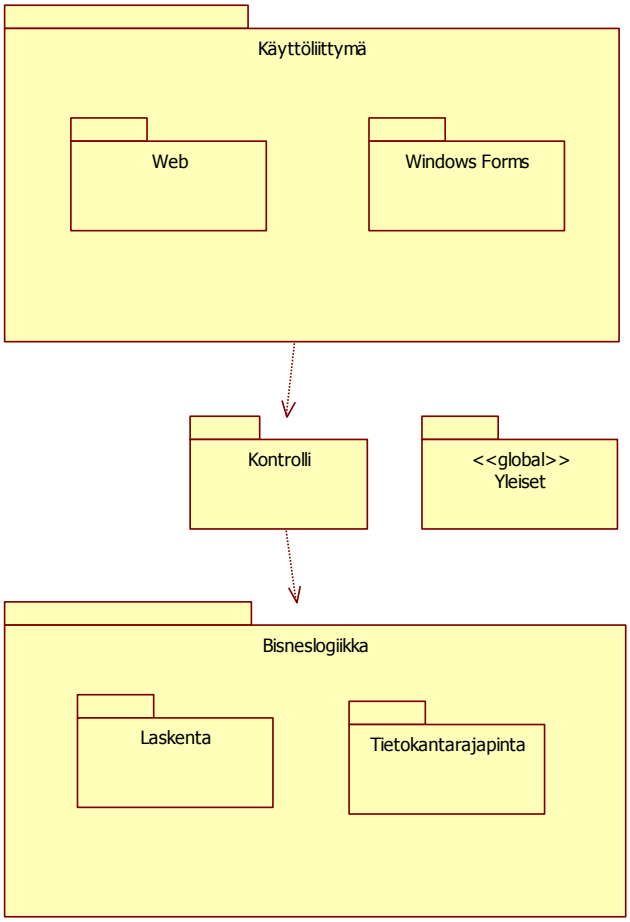


Kuva 5.9 Koostekaavio auton rakenteesta (Kylä-Nikkilä 2008)

5.3.6 Pakkauskaavio

Pakkauskaavio (package diagram) voi sisältää luokkia, pakkauksia tai mitä tahansa muita UML-elementtejä. Sen tarkoituksena on kuvata elementit ja niiden väliset riippuvuudet. Pakkauskaavion avulla voidaan seurata paremmin elementtien välisiä riippuvuuksia kuin suurista luokkakaavioista, joissa luokkien vaikutuksia toisiin luokkiin on vaikeaa hahmottaa nopeasti. Elementtien välillä on riippuvuus kun toinen elementti on riippuvainen toisesta elementistä ja ensimmäiseen tehdyt muutokset saattavat vaikuttaa siihen riippuvuussuhteessa oleviin elementteihin. Riippuvuuksien kuvaamisessa voidaan myös käyttää ste-

reotyyppejä tarkentamaan riippuvuussuhteen laatua. Esimerkiksi pakkaukset, joista kaikki muut pakkaukset ovat riippuvaisia, kuvataan pakkauskaaviossa <<global>>-stereotyypin avulla. Kuvassa 5.10 nähdään esimerkki pakkauskaaviosta. (Kylä-Nikkilä 2008)

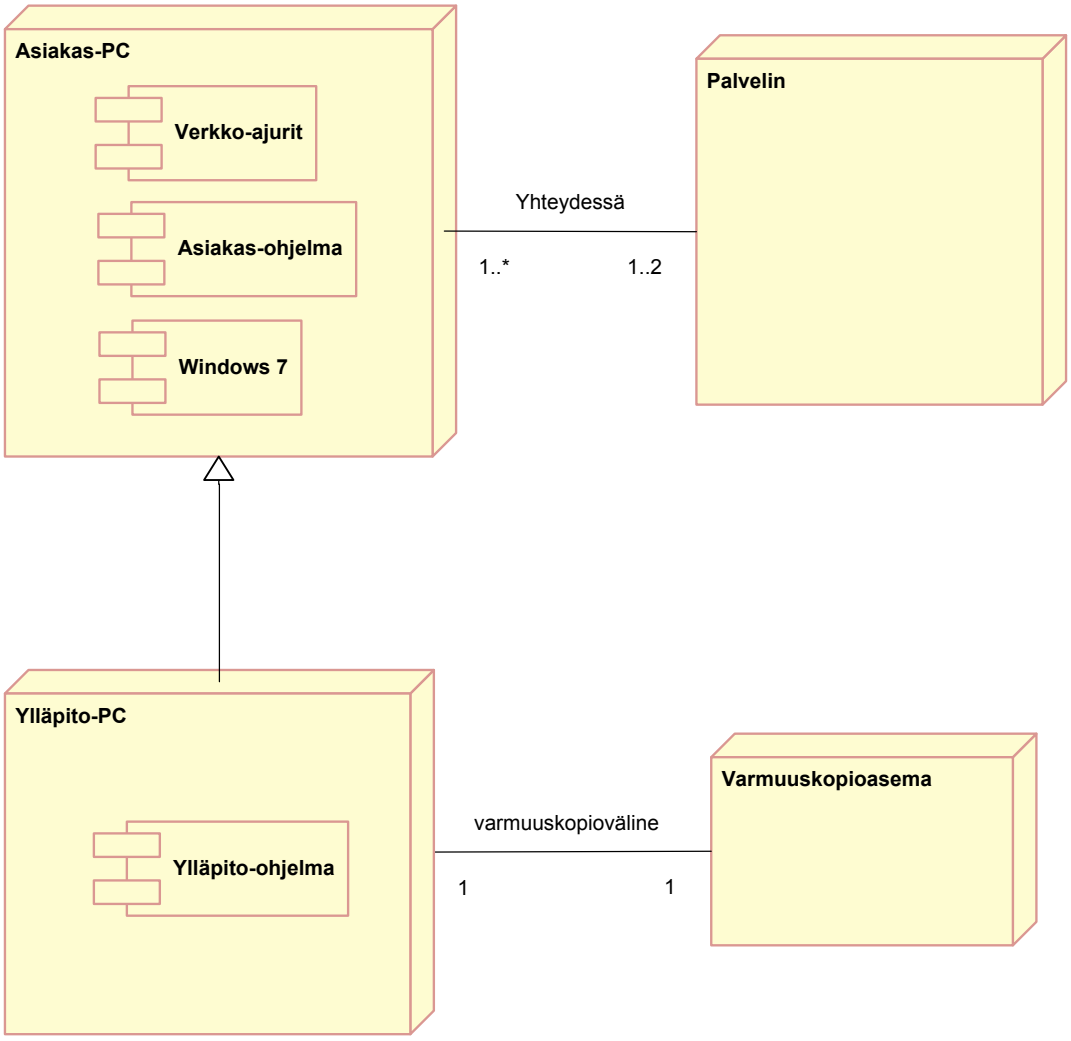


Kuva 5.10 Pakkauskaavio esimerkki (Kylä-Nikkilä 2008)

5.3.7 Sijoittelukaavio

Sijoittelukaavio (deployment diagram) kuvaa järjestelmän laitteiston ja ohjelmiston fyysisen arkkitehtuurin. Kaaviossa kuvattavat fyysiset laitteet näytetään solmuina. Solmut voivat myös sisältää komponentteja, joita kyseisen solmun laite käyttää. Myös komponenttien väliset riippuvuudet voidaan kuvata sijoittelukaaviossa. (Eriksson & Penker 2001)

Hyvin rakennetussa UML-mallissa on mahdollista siirtyä sijoittelukaaviossa olevasta solmusta sen sisällä oleviin komponentteihin ja sitä kautta aina komponenttien sisältämään luokkaan, luokan olioiden vuorovaikutuksiin ja lopulta käyttötapaukseen. Kuvassa 5.11 nähdään esimerkki sijoittelukaaviosta.
(Eriksson & Penker 2001)



Kuva 5.11 Sijoittelukaavio (Eriksson & Penker 2001, muunneltu)

5.3.8 Aktiviteettikaavio

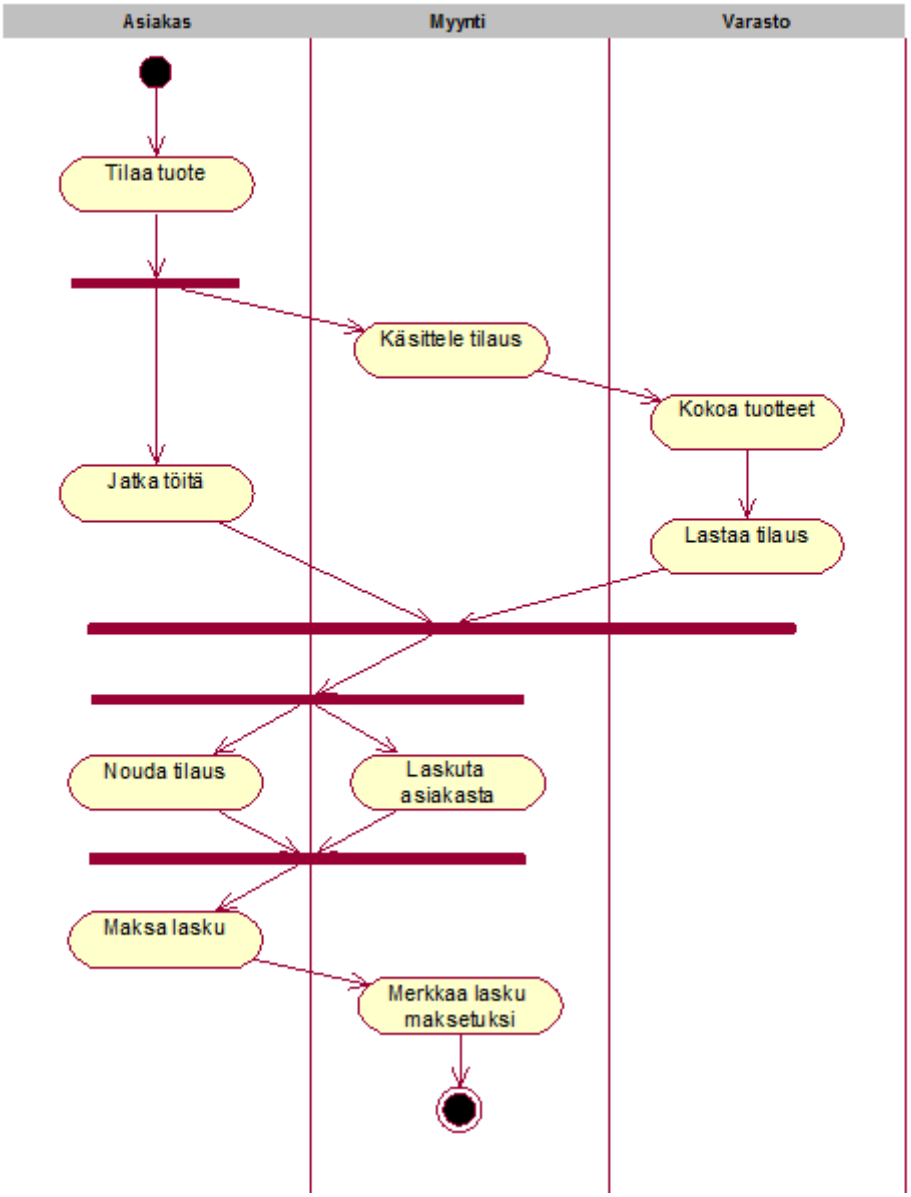
Aktiviteettikaavio (activity diagram) kuvaa tietyn tehtävän sisäisen logiikan. Tehtävä voi olla yksittäinen operaatio, käyttötapaus tai viestin välitys. Aktiviteettikaavio on muunnelma tilakaaviosta ja täyttää hieman erilaisen tarkoituksen eli toimintojen ja niiden tuottamien tulosten kuvaamisen. (Eriksson & Penker 2001)

Aktiviteettikaavio koostuu toiminnoista, jotka sisältävät suoritettavan tapahtuman määritelmän ja niitä yhdistävistä siirtymistä. Toiminnosta siirrytään pois, kun sen sisältämä toiminto on saatu suoritettua. (Eriksson & Penker 2001)

Aktiviteettikaavioissa kuvattavaan tehtävään tai toimintoon osallistuvat suorittajat erotellaan toisistaan sarakkeilla, joita kutsutaan uimaradoiksi. Uimaradat esitetään kaaviossa pystysuunnassa ja niille annetaan myös nimet.
(Eriksson & Penker 2001)

Aktiviteettikaavioita voidaan käyttää kahdella eri tavalla, joko työnkuvauksien tai operaatioiden mallintamiseen. Työnkuvauksien mallintamisessa kuvataan järjestelmää toimijoiden näkökulmasta ja liiketoimintaan liittyviä prosesseja voidaan määritellä, rakentaa tai dokumentoida aktiviteettikaavioihin. Operaatioiden mallintamisessa taas pyritään kuvaamaan järjestelmän toiminnan yksityiskohtia. (Eriksson & Penker 2001)

Kuvassa 5.12 on kuvattu yleinen tilauksen liiketoimintaprosessi aktiviteettikaaviossa.



Kuva 5.12 Tilauksen aktiviteettikaavio (Eriksson & Penker 2001)

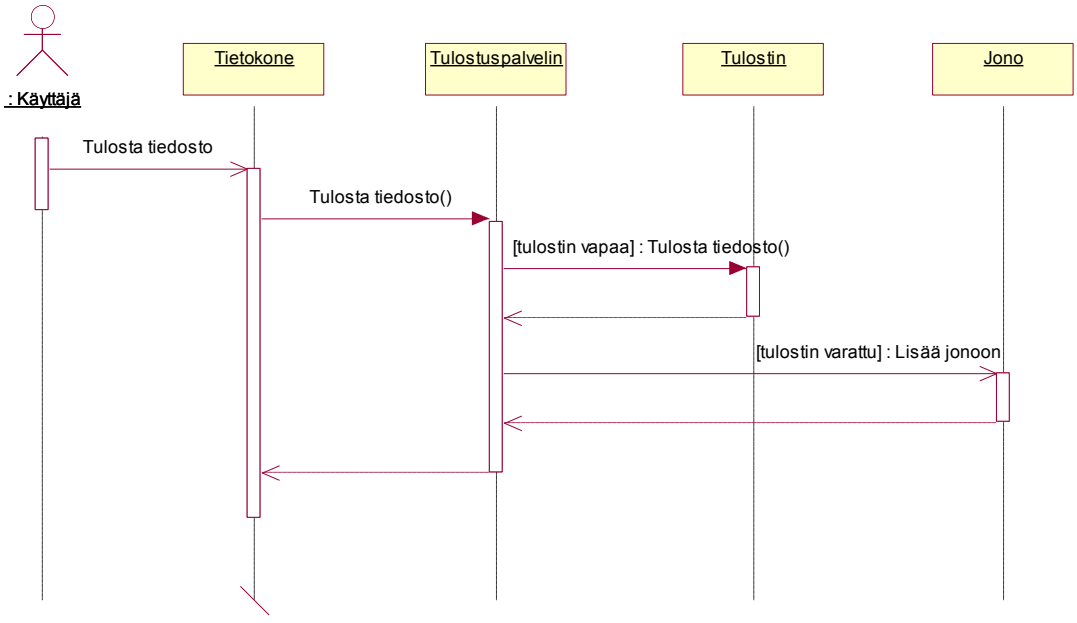
5.3.9 Sekvenssikaavio

Sekvenssikaavio (sequence diagram) kuvaa oliojoukkoa ja olioiden välisiä vuorovaikutuksia. Kaavio jakautuu kahteen akseliin, vaaka-akselilla kaaviossa kuvataan oliojoukon oliot ja pystyakselilla määritellään aika, joka kulkee kaaviossa ylhäältä alaspäin. Sekvenssikaaviolla pyritään kuvaamaan järjestelmän toimintoja, yhteen kaavioon kuvataan vain yksi toiminto. (Eriksson & Penker 2001)

Pystysuora katkoviiva sekvenssikaaviossa kuvaa siihen liittyvän olion elämänviivaa. Olioiden välinen viestintä kuvataan vaakasuorilla viivoilla elämänviivojen välissä. Nuolella osoitetaan, mihin suuntaan olioiden välinen yhteys kulkee. Olion elämänviiva paksunee kaaviossa, kun olio aktivoituu. Kun olio palaa inaktiiviseksi, palautuu elämänviiva takaisin katkoviivamuotoon. (Kylä-Nikkilä 2008)

Sekvenssikaavioita on kahdentyyppisiä: yleismuotoisia ja ilmentymämuotoisia. Ilmentymämuotoisessa sekvenssikaaviossa kuvataan tietyn toiminnon tilannetta, joka voi tapahtua. Ilmentymämuotoinen kaavio kuvaa tilaan liittyviä olioita ja niiden välisiä viestejä hyvin yksityiskohtaisesti, eikä se sisällä ehtoja, haaraumia tai toistoja. Jos kokonainen toiminto halutaan kuvata ilmentymämuotoisilla sekvenssikaavioilla, täytyy kaavioita piirtää useita kappaleita riippuen toiminnon mahdollisista tilanteista. Yleismuotoisilla kaavioilla kuvataan kaikki toiminnon mahdolliset tilanteet ja vaihtoehdot, joten se sisältää ehto-, haarauma- ja toistorakenteita. (Eriksson & Penker 2001)

Kuvassa 5.13 kuvataan käyttäjän suorittaman tulostus toiminnon viestiyhteyksiä sekvenssikaaviossa.



Kuva 5.13 Tulostus-toiminnon sekvenssikaavio (Eriksson & Penker 2001)

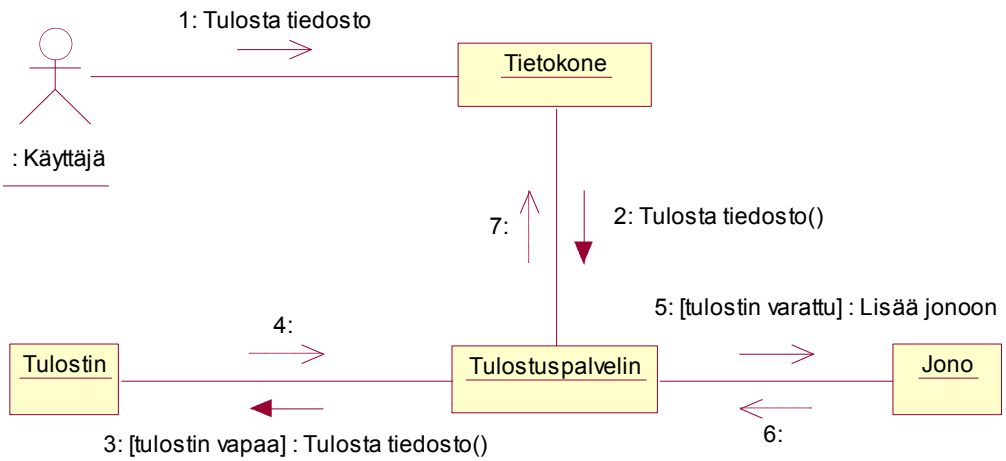
Kuvassa 5.13 tietokone saa viestin tiedoston tulostamisesta, seuraavaksi tietokone välittää viestin tulostuspalvelimelle. Tulostuspalvelin tarkastaa, onko tulostin vapaana. Jos se on vapaana, tulostetaan tiedosto, mutta jos tulostin on varattu, tallennetaan tiedosto jonoon odottamaan tulostimen vapautumista.

5.3.10 Kommunikointikaavio

Kommunikointikaavio (collaboration diagram) on hyvin samankaltainen kuin sekvenssikaavio, koska kummatkin kaaviot kuvaavat oliojoukon olioiden keskinäistä vuorovaikutusta. Kommunikointikaavio eroaa sekvenssikaaviosta siinä, että se kuvaa olioiden vuorovaikutusta vain tilaan sidottuna. Se sisältää lähes samat komponentit ja käsitteet kuin sekvenssikaavio, vain viestiyhteydet kuvataan eri tavalla. (Kylä-Nikkilä 2008)

Kommunikointikaaviossa kuvataan oliojoukon oliot, olioiden väliset suhteet ja viestit. Viestien keskinäinen suhde esitetään hieman eri tavalla kuin sekvenssikaaviossa kronologisen järjestyksen vuoksi. Viestien järjestyksen esittämiseksi, yhteyden tai viestin nimen eteen sijoitetaan juokseva numero, joka määrittelee viestien järjestyksen. Monimutkaisemmissa kaavioissa useita samanaikaisesti lähetettäviä viestejä kuvattaessa voidaan lisätä kirjain tai jakaa numerointi useampaan osaan. (Kylä-Nikkilä 2008)

Kuvassa 5.14 nähdään Tulostus-toiminnon kommunikointikaavio, joka kuvaa samaa toimintoa kuin kuvan 5.13 sekvenssikaavio.

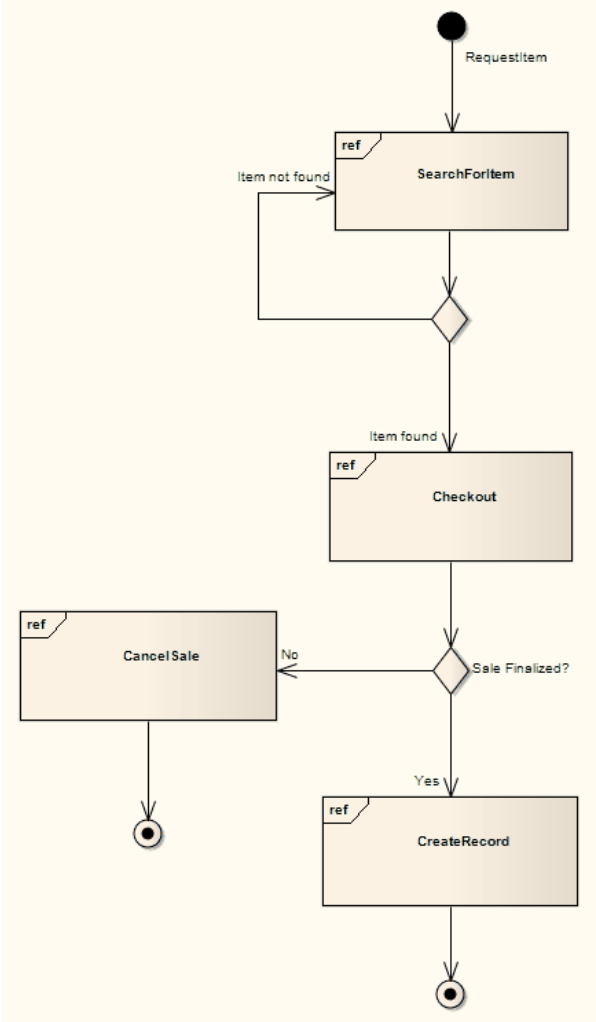


Kuva 5.14 Tulostus-toiminnon kommunikointikaavio (Eriksson & Penker 2001)

5.3.11 Kokoava vuorovaikutuskaavio

Kokoavaa vuorovaikutuskaaviota (interaction overview diagram) käytetään mallintamaan järjestelmän ohjelmalogiikkaa tai muita suuria käyttötapauksia, joissa on yhdisteltävä useita eri vuorovaikutuskaavioita yhteen näkymään. Kokoava vuorovaikutus kaavio antaa lukijalle hyvän kokonaiskuvan. Se piirretään karkeasti ottaen aktiviteettikaaviona, jossa aktiviteetteinä käytetään vuorovaikutuskaavioita. Vuorovaikutuskaaviot, joita kokoavissa vuorovaikutuskaavioissa voidaan käyttää, ovat ajoituskaavio, kommunikointikaavio ja sekvenssikaavio. (Kylä-Nikkilä 2008)

Vuorovaikutus merkitään kokoavassa vuorovaikutuskaaviossa sd-lyhenteellä ja viittaukset muihin kaavioihin ref-lyhenteellä. Lyhenteet kirjoitetaan itse vuorovaikutuksen ympyröivään kehykseen. (Kylä-Nikkilä 2008)

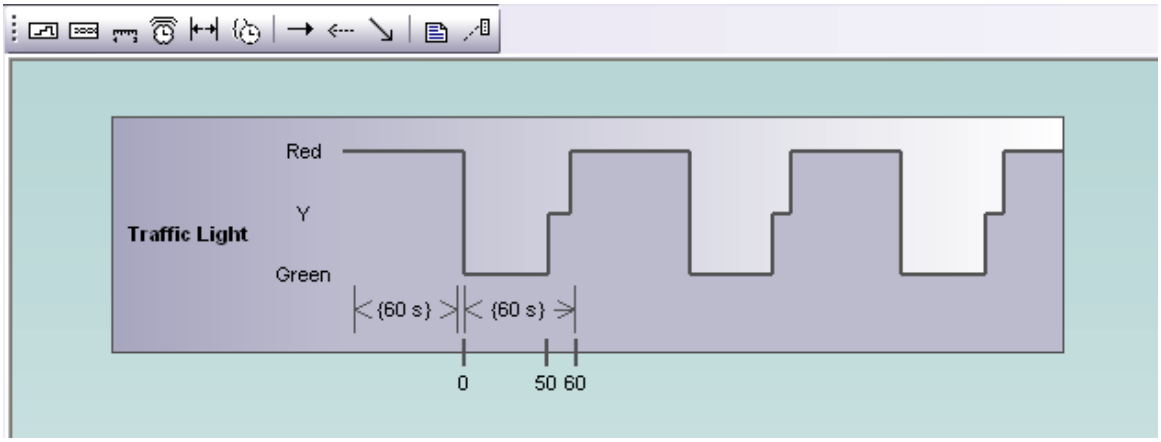


Kuva 5.15 Kokoava vuorovaikutuskaavio (Sparx Systems 2010)

5.3.12 Ajoituskaavio

Ajoituskaaviolla (timing diagram) pyritään kuvaamaan järjestelmän aktiivisia olioita ja niiden tiloja. Niitä käytetään lähinnä sulautetuissa järjestelmissä ja tosiaikajärjestelmissä, joissa vuorovaikutuksiin kuluva aika halutaan selvittää. Kaaviossa kuvataan aikaa x-akselilla, ja y-akselilla on kuvattuna oliot ja niiden mahdolliset tilat. (Kylä-Nikkilä 2008)

Kuvassa 5.16 on esitetty liikennevalojen toiminta ajoituskaavion avulla.



Kuva 5.16 Ajoituskaavio, liikennevalon toiminta (Altova 2010)

5.3.13 Käyttötapauskaavio

Käyttötapauskaavio (use case diagram) liittyy käyttötapausmallinnustekniikkaan, jonka avulla kuvataan, mitä uuden järjestelmän tulisi tehdä tai mitä jo olemassa oleva järjestelmä tekee. Käyttötapauskaavioita rakennetaan iteratiivisesti ja niitä käytetään asiakkaan ja kehittäjien välisissä neuvotteluissa apuvälineenä, jotta päästään lopulta sellaiseen vaatimusmäärittelyyn, josta kaikki ovat samaa mieltä. (Eriksson & Penker 2001)

Käyttötapauskaavion tärkeimmät elementit ovat itse käyttötapaukset, toimijat ja mallinnettava järjestelmä. Mallinnettavan järjestelmän rajaus tehdään määriteltyjen käyttötapauksien eli toimintojen mukaisesti. Käyttötapaukset kuvataan aina ulkopuolisten toimijoiden näkökulmasta. Käyttötapauskaaviossa järjestelmä kuvataan niin sanotusti "mustana laatikkona", joka tarjoaa kuvatut toiminnot eli

käyttötapaukset toimijoille. Kaaviossa ei oteta kantaa siihen kuinka järjestelmä toteuttaa kuvatut toiminnot. Käyttötapauskaaviot ja käyttötapausmallinnus tehdään ohjelmistoprojektin alkuvaiheessa, jolloin kehittäjillä ei ole mitään käsitystä käyttötapausten eli toimintojen toteuttamisesta. Käyttötapausten päätarkoitukset ovat (Eriksson & Penker, 2001):

- Järjestelmän toiminnallisten vaatimusten päättäminen ja kuvaaminen, jotta asiakkaan ja ohjelmistokehittäjien välille syntyy yhteisymmärrys.
- Järjestelmän toimintojen selkeä ja yhtenäinen kuvaus, jonka perusteella kehittäjät suunnittelevat toteutuksen, joka täyttää vaatimuksissa esitetyn toiminnallisuuden.
- Järjestelmän testauksen perusta, jonka avulla varmistetaan järjestelmän toiminnan oikeellisuus. Toteuttaako järjestelmä toiminnot, joita alunperin haluttiin.
- Mahdollistaa siirtyminen toiminnallisista vaatimuksista luokkiin ja operaatioihin, jotka toteuttavat ne.
- Yksinkertaistaa järjestelmän muutos- ja laajennustyötä, kun muutokset tehdään ensin käyttötapausmalliin ja jatkamalla niistä järjestelmän suunnitelman ja toteutuksen vastaaviin osiin.

Käyttötapauskaavio on yksi UML:n käytetyimmistä kaavioista luokkakaavion ohella. Se on rakenteeltaan yksinkertainen ja koostuu jo aiemmin mainituista toimijoista, käyttötapauksista ja niiden välisistä suhteista. (Kylä-Nikkilä 2008)

Toimija kuvaa järjestelmän ulkopuolisia olioita, jotka toimivat vuorovaikutuksessa järjestelmän kanssa. Toimijat ovat siis järjestelmän käyttäjiä, jotka suorittavat kaaviossa kuvattuja käyttötapauksia. Toimija voi olla esimerkiksi henkilö tai jokin muu ulkopuolinen järjestelmä. (Kylä-Nikkilä 2008)

Käyttötapaus kuvaa jotakin toimijan havaitsemaa täydellistä toimintoa. Käyttötapauksen suorituksen seurauksena on tietyn toimijan havaitsema hyödyllinen tulos. Näihin voi kuulua useiden toimijoiden välistä viestintää, järjestelmän suorittamaa sisäistä laskentaa ja työtä. Käyttötapaukset nimetään UML:ssä usein

niiden suorittamien toimintojen mukaan. Esimerkiksi "Raportin tulostus", "Henkilo tietojen hakeminen". Nimi on usein pidempi kuin yksi sana.
(Eriksson & Penker 2001)

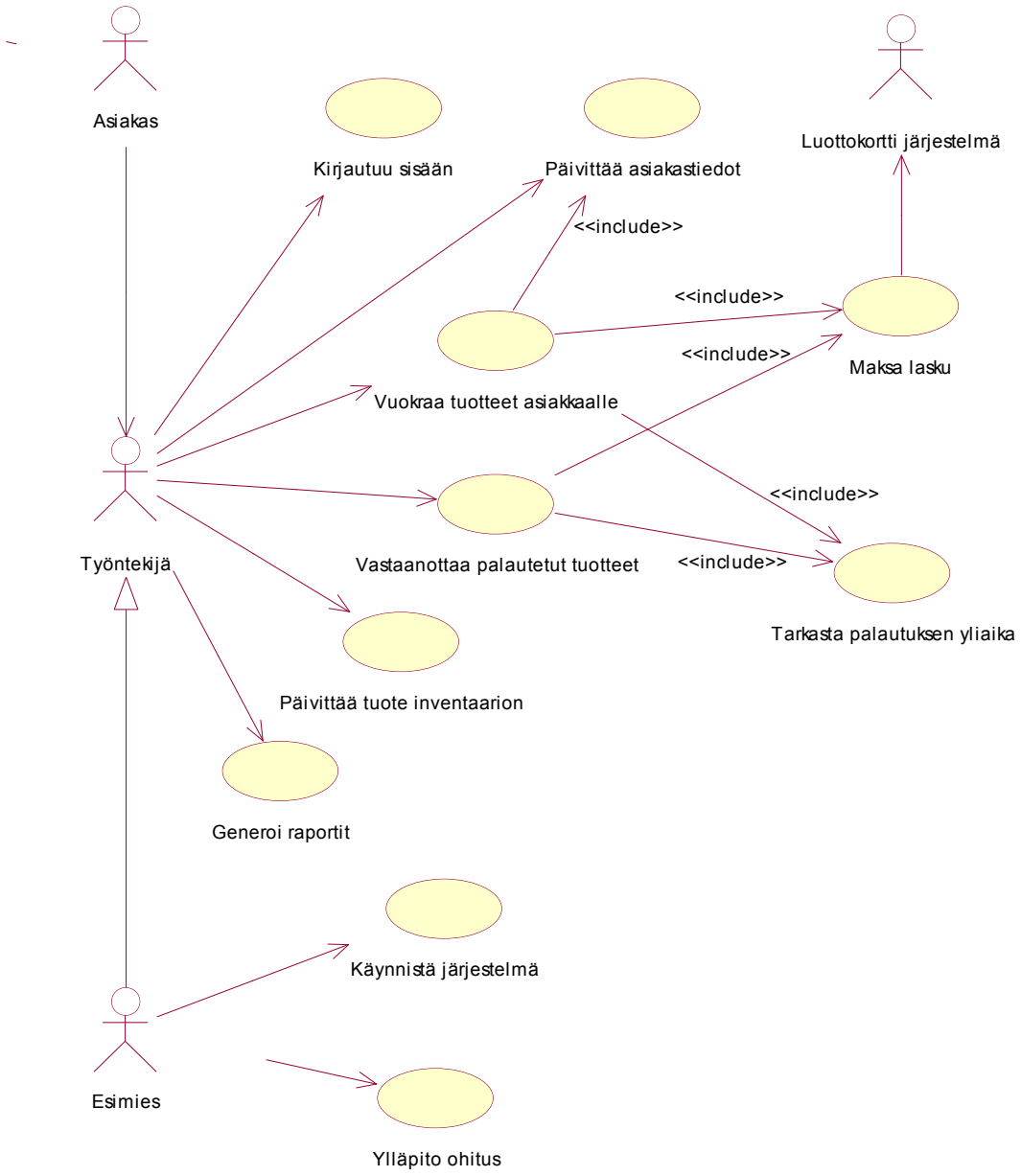
Käyttötapaukset ja toimijat kytketään toisiinsa yhteyksillä, joita joskus kutsutaan viestiassosiaatioiksi. Nämä assosiaatiot näyttävät toimijat, joiden kanssa käyttötapaukset kommunikoivat. Toimijoilla voi olla erilaisia rooleja, joita voidaan yleistää tarvittaessa. Yleistys kuvataan luokkakaavion tapaan nuolena, joka osoittaa yleiseen toimijan rooliin. Myös käyttötapausten välillä voi olla yhteyksiä, näitä kutsutaan suhdetyypeiksi, joita voivat olla periytymis-, laajennus- tai sisällytyssuhde. (Kylä-Nikkilä 2008)

Periytymissuhteessa käyttötapaus perii toisen käyttötapauksen ja lisää siihen omaa toiminnallisuutta tai korvaa vanhaa (Kylä-Nikkilä 2008).

Laajennussuhde on yleistyssuhde, jossa käyttötapaus laajentaa toista käyttötapausta. Siinä ei tarvitse välttämättä ottaa kaikkea toiminnallisuutta mukaan laajennettavaan käyttötapaukseen. Sitä käytetään yleensä kuvaamaan poikkeuksia tai erikoistapauksia. Laajennussuhde kuvataan kaaviossa <<extend>> -stereotyypin avulla käyttötapausten välisessä yhteydessä. (Kylä-Nikkilä 2008)

Sisällytyssuhteessa käyttötapaus sisältyy toiseen käyttötapaukseen. Käyttötapauksilla voi olla sellaisia yleisiä toimintoja, joita muut käyttötapaukset käyttävät ja tällöin on hyödyllistä tehdä kyseisistä toiminnoista erillinen käyttötapaus, johon muut sitä käyttävät käyttötapaukset viittaavat. Tällaista käyttötapausta eivät myöskään kutsu tai käytä järjestelmän toimijat. Sisällytyssuhde kuvataan kaaviossa <<include>> -stereotyypin avulla. (Kylä-Nikkilä 2008)

Kuvassa 5.17 on esimerkki yksinkertaisesta vuokrausjärjestelmän käyttötapauskaaviosta.

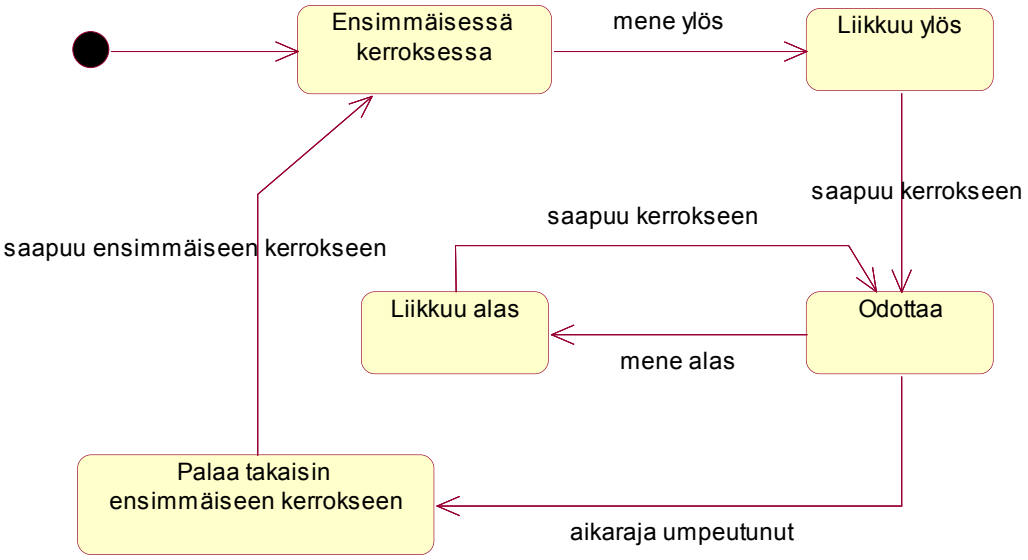


Kuva 5.17 Yksinkertaisen vuokrausjärjestelmän käyttötapauskaavio

5.3.14 Tilakaavio

Tilakaavioilla (state diagram) voidaan täydentää luokkien kuvausta. Kaaviolla kuvataan kaikki kohdeluokan mahdolliset tilat, joihin kohdeluokan ilmentymät voivat joutua ja tapahtumat, jotka aiheuttavat ilmentymille tilasiirtymiä. Tapahtuma voi olla jokin toinen olio, joka lähettää viestin esimerkiksi siitä, että tietty aika on kulunut tai että jokin ehto on toteutunut ja kohdeolion tila vaihtuu tapahtuman seurauksena. Siirtymään voidaan myös liittää toiminto, joka määrittelee, mitä ohjelman tulee tehdä siirtymän aikana. (Kylä-Nikkilä 2008)

Tilakaavioita piirretään vain sellaisille luokille joista on tunnistettu useita selkeästi eroteltavia tiloja. Toisaalta tilakaavioita voidaan piirtää myös koko järjestelmälle. (Kylä-Nikkilä 2008)



Kuva 5.18 Hissin tilakaavio

6 TYÖSSÄ KÄYTETYT TEKNIIKAT

Seuraavaksi käydään läpi opinnäytetyössä käytetyt tekniikat ja työkalut. Laajan nykyaikaisen tietojärjestelmän kehittäminen on monimutkaista. Kehitystyön avuksi kannattaa opetella käyttämään oikeita tekniikoita ja työkaluja, joilla työntekoa voidaan helpottaa, nopeuttaa ja hallita sekä parantaa lopputuotteen laatua.

6.1 HTTP-protokolla

HTTP-protokollaa (Hypertext Transfer Protocol) käytetään tiedonsiirtoon esimerkiksi WWW-selaimen ja WWW-palvelimen välillä. HTTP-protokolla sijaitsee OSI-mallin sovelluskerroksella. HTTP on geneerinen tilaton protokolla, jota voidaan käyttää moniin muihinkin käyttötarkoituksiin kuin hypertextin jakeluun, kuten nimipalvelimiin ja hajautettujen olioiden hallintajärjestelmiin. Kyseisiä toimintoja voidaan toteuttaa HTTP:n tarjoamien metodien, vikakoodien ja otsikkotietojen avulla. HTTP:n tehtävänä on tyypittää ja neuvotella tiedon esitystapa sekä mahdollistaa päälle rakennettavien itsenäisten järjestelmien tiedonsiirto ja toiminta. HTTP on ollut laajalti käytössä 1990-luvulta lähtien, ja sen uusin standardi HTTP/1.1 on peräisin vuodelta 1999.

6.2 Apache

Apache on maailman käytetyin WWW-palvelinohjelmisto, joka tarjoaa tarvittavan palvelurajapinnan WWW-sovellusten kehittämiseen ja isännöintiin internetissä. Apache on ilmainen vapaan lähdekoodin ohjelmisto, joka on kirjoitettu C-kielellä. Apache-palvelinohjelmiston ensimmäinen versio (0.6.2) julkistettiin huhtikuussa 1995 Apache Groupin toimesta. (Lipitsäinen & Marttila 2003)

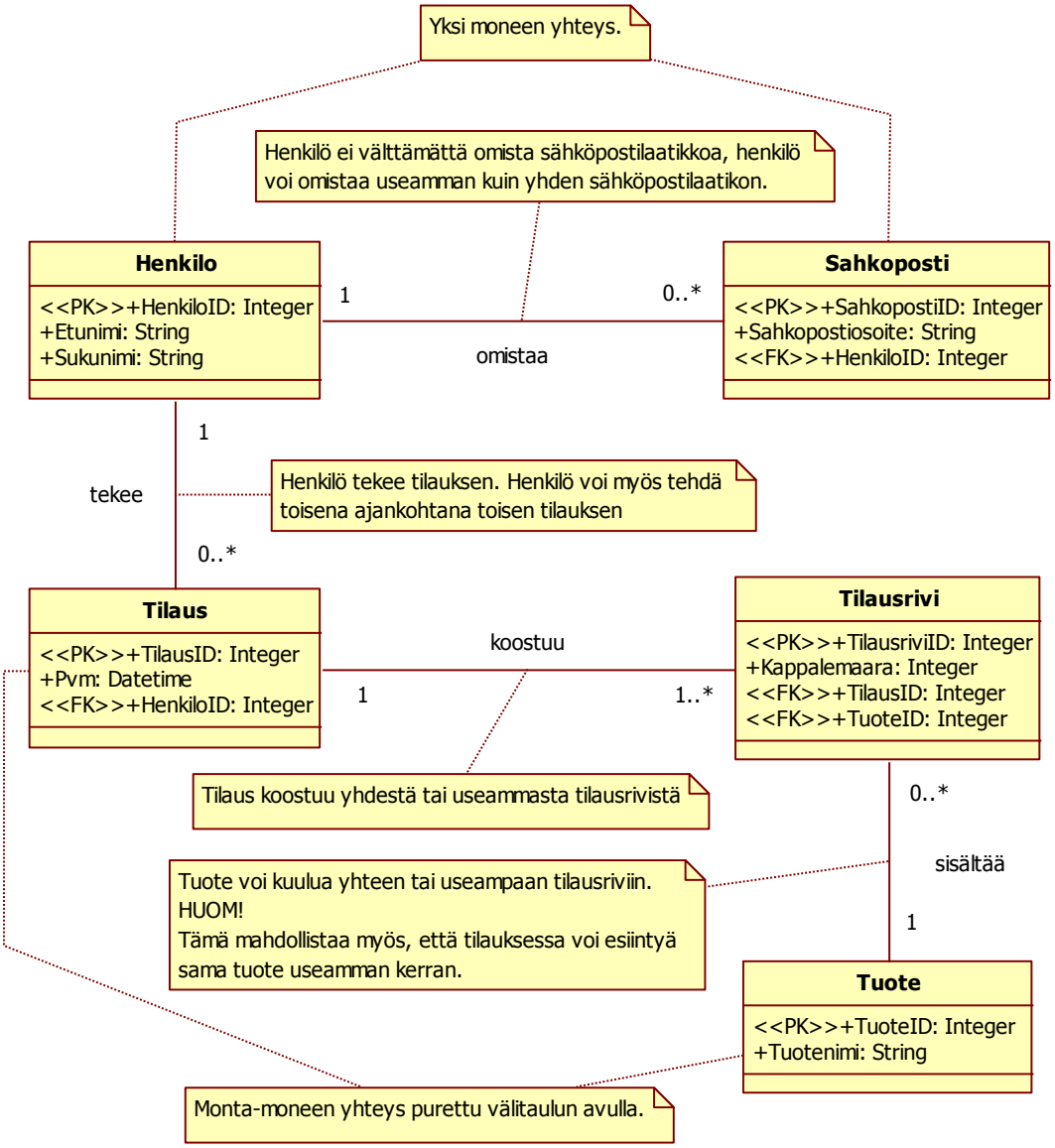
Apache Group -ryhmä oli voittoa tavoittelematon yhdistys ja se operoi kokonaan Internetin kautta. Yhdistyksen toiminta laajeni ja kesäkuussa 1999 sen toimintaa jatkamaan perustettiin Apache Software Foundation -yhdistys (ASF). ASF-yhdistyksen toimintaa valvoo yhdistyksen hallitus, jonka valitsee ASF-jäsenistö vuosittain yhtymän johtosäännön mukaisesti. Hallituksen tehtävänä on nimittää

virkailijat hoitamaan yhdistyksen jokapäiväisiä asioita ja valvomaan ASF:n projekteja. Projekteja johtaa itsevalintainen teknisistä asiantuntijoista koostuva tiimi, jonka jäsenet ovat aktiivisia projektin tukijoita ja avustajia.
(Lipitsäinen & Marttila 2003)

6.3 Relaatiotietokanta

1960-luvun lopulla IBM:n tutkija, tohtori E.F. Codd käytti matematiikan oppeja ja rakenteita hyväkseen ratkaistaakseen senaikaisia tietokantamalleja ja tuotteita vaivanneita ongelmia kuten tietojen heikko eheys, riippuvuus fyysisestä toteutuksesta ja ylimääräisen datan määrä. Hän julkaisi myöhemmin vuoden 1970 kesäkuussa työnsä "A Relational Model of Data for Large Shared Databanks". Teoksessa esiteltiin relaatiotietokantamalli, joka perustui kahteen matematiikan haaraan, joukkoteoriaan ja ensimmäisen kertaluvun predikaattilogiikkaan. Yleisen harhakäsitys on, että relaatiotietokantamallin nimitys tulee siitä, että tietokannan tauluilla voi olla niin sanottuja suhteita eli relaatioita toisiinsa.
(Hernandez 2000)

Relaatiotietokantamallissa tiedot tallennetaan relaatioina, jotka käyttäjä näkee tauluina. Jokainen relaatio muodostuu monikoista eli tietueista ja attribuuteista eli kentistä. Taulun tietueiden tai kenttien fyysinen järjestys on täysin yhdentekevää. Jokainen taulun tietue tunnistetaan kentän avulla, joka sisältää muista poikkeavan arvon eli perusavaimen. Edellä mainitut ominaisuudet mahdollistavat tiedon olemassa olon ja talletuksen tietokoneeseen riippumatta tavasta, jolla ne on fyysisesti kirjoitettu keskusmuistiin tai kiintolevylle. Tästä seuraa, ettei käyttäjän tarvitse tietää tietueen fyysistä sijaintia muistissa tai levyllä voidakseen hakea sen tiedot. (Hernandez 2000)



Kuva 6.1 Esimerkki relaatiotietokannasta

Relaatiotietokantamallissa taulujen väliset yhteydet luokitellaan seuraavasti: yhden suhde yhteen, yhden suhde moneen ja monen suhde moneen. Yhteydet taulujen välille luodaan epäsuorasti asettamalla kahden jaetun kentän arvot täsmääviksi. Tämä mahdollistaa, että käyttäjä voi hakea tietoja millä tahansa tavalla, jos hän vain tietää tietokannan taulujen väliset yhteydet. Käyttäjä voi hakea tietoja tauluista, joiden välillä on suora yhteys, tai tauluista, jotka liittyvät toisiinsa epäsuorasti. (Hernandez 2000)

6.4 SQL

SQL (Structured Query Language) eli rakenteinen kyselykieli kehitettiin IBM:n laboratoriossa San Josessa Kaliforniassa 1970-luvun lopulla. Se kehitettiin alunperin IBM:n DB2:ta varten. Tarkemmin katsottuna SQL ei ole rakenteellinen kieli, vaan ns. joukko-orientoitunut, joka tarkoittaa, että SQL käsittelee tietojoukkoja ryhmissä. (Stephens, Plew, Morgan & Perkins 2001)

SQL-standardeja tuotetaan samanaikaisesti kahden standardiorganisaation toimista. Kyseiset organisaatiot ovat ANSI (American National Standards Institute) ja ISO (International Standards Organization).
(Stephens, Plew, Morgan & Perkins 2001)

SQL on standardi, kun käsitellään, haetaan tai hallinnoidaan tietoja relaatiotietokannoissa. SQL:n avulla ohjelma tai tietokannan hoitaja voi hakea, lisätä, muokata, päivittää ja poistaa tietokannasta sisältöä sekä lisätä käyttäjiä ja käyttöoikeuksia eri käyttäjille eri kohde tietokannoille ja tauluille. Tietokannan hoitaja voi myös muokata tarvittaessa tietokannan turvallisuusasetuksia. SQL:n avulla käyttäjä voi luoda tietokantaan tarvittavat taulut ja yhteydet sekä tallettaa tietokantaan valmiita kyselyjä, ajastimia tai valmiita proseduureja.
(Stephens, Plew, Morgan & Perkins 2001)

```
SELECT HenkiloID, Etunimi, Sukunimi
FROM henkilo
WHERE Sukunimi LIKE 'Meikäläinen'
ORDER BY Sukunimi ASC
```

Kuva 6.2 Esimerkki-MySQL-kysely

Kuvassa 6.2 MySQL-lause suorittaa kyselyn, joka hakee henkilo-taulusta kaikki ne henkilöt, joiden Sukunimi kenttä on kuin 'Meikäläinen'. Kysely haluaa, että tuloksista palautetaan kentät HenkiloID, Etunimi ja Sukunimi ja että tulostietueet järjestetään Sukunimi-kentän mukaan kasvavaan järjestykseen eli aakkosjärjestykseen.

6.5 MySQL

Helsinkiläinen Michael "Monty" Widenius ja ruotsalainen David Axmark kehittivät MySQL:n alunperin vuonna 1995. Ensimmäinen versio MySQL:stä julkaistiin vuonna 1996 ja sitä kehittää ruotsalainen MySQL AB yritys, jonka Sun Microsystems osti 16.1.2008. (OpenSource Ratol)

MySQL on suosituin avoimeen lähdekoodiin perustuva SQL-tietokannan hallintajärjestelmä, joka on kooltaan suhteellisen kompakti ja rakenteeltaan yksinkertainen sekä vaatii vähän ylläpitoa. MySQL:n tärkeimmät vahvuudet ovat sen nopeus, siirrettävyys, hinta ja yhteensopivuus eri ohjelmointikielien kanssa. MySQL:n suosio pohjautuu hintaan, yhteensopivuuteen ja käytön helppouteen, joiden takia se usein valjastetaan WWW-sivustojen käyttöön. (Patosuo 2009)

MySQL noudattaa SQL-standardia kyselykielessään, mutta se sisältää myös omia laajennuksia, jotka eivät ole yhteneväisiä SQL-standardin kanssa. MySQL perustuu avoimen lähdekoodin GNU GPL -lisensointiin, joka mahdollistaa kenen tahansa asentaa, käyttää ja muokata MySQL:ää ilmaiseksi. MySQL-tuotteelle löytyy myös kaupallinen lisenssi. (OpenSource Ratol)

6.6 MyISAM

MyISAM on MySQL:n käyttämä oletustietokantamoottori versioon 5.5.5 saakka. Se on tehokas ja vähän levytilaa käyttävä ratkaisu, jossa jokainen taulu tallennetaan fyysiselle levylle kolmeen eri tiedostoon. Ensimmäinen tiedosto (.frm) sisältää taulun rakenteen, toinen tiedosto taulun datan (.myd) ja kolmas (.myi) taulun indeksit. MyISAM heikkoutena on transaktioiden ja viiteavaimien puute. Transaktiot huolehtivat tietokantataulujen tiedon eheydestä ilman ulkopuolisen ohjelman tukea. MyISAM soveltuu hyvin pienille tietosisällöille ja sellaisiin käyttötarkoituksiin, joissa taulujen välisiä viite-eheyskytköksiä on vähän tai tilanteisiin, joissa niitä ei tarvita. Virhetilanteissa palautumisaika on sidoksissa taulun datan määrään. (Oracle 2010)

6.7 InnoDB

InnoDB on suomalaisen Innobase Oy:n kehittämä tietokantamoottori suosittuun avoimen lähdekoodin MySQL-tietokantaan. InnoDB on tarkoitettu käytettäväksi sellaisissa tietokannoissa, joissa tiedon määrä on suuri ja taulujen väliset yhteydet ja viite-eheydet ovat tärkeässä asemassa. InnoDB takaa siis tallennetun tiedon eheyden ja luotettavuuden, mutta se vie enemmän levytilaa ja tietojen käsittely on hidasta. Näitä ongelmia on pyritty InnoDB:ssä lievittämään Oraclen mukaan mallinnetuilla ACID-transaktio (Atomicity, Consistency, Integrity, Durability) menetelmillä ja aggressiivisella välimuistin käytöllä. (Oracle 2010)

InnoDB soveltuu hyvin suurien tietomäärien käsittelyyn ACID-transaktioiden ja virheistä palautumisen johdosta. Verrattuna MyISAM-moottoriin InnoDB:n virheistä palautumiseen kuluva aika skaalautuu verrannollisesti transaktiolokien kokoon, kun taas MyISAM:n palautumisen aika skaalautuu verrannollisesti sisällytetyn datan määrään. Tietokannan käytön suhteen InnoDB on paremmin skaalautuva samanaikaisen käytön lisääntyessä johtuen ACID-transaktioiden tietuekohtaisesta lukinnasta MyISAM:n käyttäessä taulukohtaista lukitusmenetelmää.

6.8 HTML

HTML (HyperText Markup Language) on dokumentin muodon määrittelykieli. HTML on yksinkertainen ja helposti opittava strukturoidun eli rakenteisen tekstin merkkauskieli. Dokumentin kirjoittaja voi määritellä tekstin korostukset, luetteloinnit, taulukot, linkit eli siirtymät muihin dokumentteihin tai dokumentin osiin. HTML-kieli mahdollistaa viittaukset muihin verkon resursseihin kuten uutisryhmiin ja FTP-palvelimiin. Dokumentteihin voidaan lisätä kuvia ja ääntä. Huomioitavaa HTML-kielessä on se, ettei se ole varsinainen ohjelmointikieli.
(Perustietoa HTML:stä.)

6.9 CSS

CSS (Cascading Style Sheets) on web-dokumentin esitysasun määrittely. Kun HTML(XHTML)-dokumenttiin lisätään CSS -tyylisäännöstö, niin kaikki tyylisäännöstössä kuvatut HTML-elementit muutetaan ulkoasultaan vastaamaan tyylitiedoston määrittelemiä muotoiluja. Säännöt eivät ole ehdottomia vaan ne voidaan tarvittaessa kiertää. Tarkemmin katsottuna CSS on kieliperhe, johon kuuluvat nykyisin CSS1 ja sitä paljon laajempi CCS2, lisäksi CSS3 on valmisteilla. CSS-kielien määritelmiä ylläpidetään W3C:n toimesta. (Värikkyyttä: Tyylisäännöstöt.)

CSS:llä voidaan kuvata monipuolisesti sekä nähtävää että kuultavaa esitystapaa. Äänisyntetisaattoreita varten on tehty äänen korkeutta, painotusta ja väriä säätteleviä ominaisuuksia. Visuaalisessa esitystavassa käytetään niin sanottua laatikkomallia, jossa jokainen dokumentin elementti mallinnetaan laatikkona. Laatikot sijoitellaan ympäröivien elementtien sisälle tai saman tason elementtien kanssa vierekkäin. Jokainen elementti koostuu neljästä sisäkkäisestä laatikosta. Uloin laatikko kuvaa elementin marginaalia, seuraava laatikko elementin reunusta, joka rajoittuu kolmanteen elementin sisältöä ympäröivään täytelaatikkoon. Viimeinen ja sisin laatikko sisältää itse kuvattavan elementin sisällön. (Värikkyyttä: Tyylisäännöstöt)

```
/* Esimerkki yksinkertaisesta CSS-tyylitiedostosta */
body
{
    background-color:#d0e4fe;
}

h1
{
    /* Box Model */
    margin:0px;                     /* Marginaali   */
    border:5px solid gray;          /* Rajaus       */
    padding:10px;                   /* Täyte        */
    font-family:"Times New Roman";  /* Sisältö      */
    font-size:20px;                 /* Sisältö      */
}

p
{
    font-family:"Times New Roman";
    font-size:20px;
}
```

Kuva 6.3 Esimerkki yksinkertaisesta CSS-tyylitiedostosta kommentteineen

6.10 PHP

PHP (PHP: Hypertext Preprosessor) on yleiskäyttöinen vapaan lähdekoodin skriptikieli, joka soveltuu parhaiten dynaamisten web-sovellusten kehitykseen. PHP-kielen syntaksi muistuttaa C-kieltä, Javaa ja Perliä ja sitä voidaan upottaa suoraan HTML-dokumentin sisälle. PHP-koodi kapseloidaan kielelle ominaisen "<?php ... ?>"- merkkauksen sisälle. HTML-dokumenttia, joka sisältää PHP-koodia, kutsutaan myös XHTML-dokumentiksi, ja se nimetään yleisesti .php-päätteiseksi tiedostoksi. XHTML-dokumentit voivat sisältää useita <?php ?> -elementtejä. (What is PHP; Johdatus PHP kieleen)

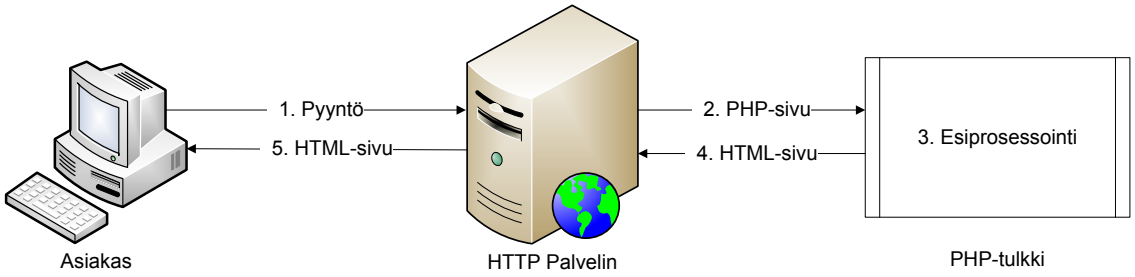
```
<?php echo '<?xml version="1.0" encoding="ISO-8859-1"?>'?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fi">
<head>
  <title>Sivun otsikko</title>
</head>
<body>
  <h1>Tulostetaan:</h1>
  <p>
    <?php echo "PHP-koodia" ?>
  </p>
</body>
</html>
```

Kuva 6.4 Esimerkki HTML-dokumenttiin upotetusta PHP-koodista

PHP-kieli on lisenssivapaa avoimen lähdekoodin tuote, jota voidaan käyttää maksutta myös kaupallisten sovellusten toteutukseen. PHP-tuki on mahdollista saada kaikkiin yleisimpiin WWW-palveluohjelmistoihin ja käyttöjärjestelmiin. Ohjelmointikielenä PHP on helppo oppia ja se soveltuu hyvin aloittelijoille sekä ammattimaiseen käyttöön. PHP-kieli sisältää tuen myös kaikille yleisimmille tietokannoille ja uusimmat PHP:n versiot tukevat oliosuuntautunutta ohjelmointia. (What is PHP; Johdatus PHP-kieleen.)

Käytännössä PHP-koodi toimii vain palvelinpuolella, jossa se tulkataan ja suoritetaan. Tavallisessa tapauksessa asiakkaan pyytäessä palvelimelta sivua tunnistaa HTTP-palvelin tiedoston päätteen perusteella, että kysymyksessä on

PHP-sivu. Seuraavaksi palvelin välittää PHP-tulkille pyynnön esiprosessoida pyydetty sivu. PHP-tulkki kääntää ja suorittaa tiedoston sisältämän PHP-koodin. Lopputuloksena PHP-tulkki palauttaa HTTP-palvelimelle pelkkää HTML-koodia. Lopuksi HTTP-palvelin palauttaa asiakkaalle HTML-koodin. PHP-koodi on siis olemassa ainoastaan palvelimella. Asiakas ei näe sitä missään vaiheessa. (What is PHP; Johdatus PHP-kieleen.)



Kuva 6.5 HTTP:n ja PHP:n toiminta

6.11 JavaScript

JavaScript on alun perin Netscape Corporationin kehittämä oliopohjainen ohjelmointikieli. JavaScript yhdistetään usein Javaan, mutta Java on kuitenkin täysin erilainen ohjelmointikieli, jonka rakenne on paljon monimutkaisempi. Java on kehitetty Sun Microsystemsillä. JavaScript-kielen kehitti Brendan Eich Netscapella ja se otettiin käyttöön vuonna 1996 Netscapen ja Microsoftin Internet-selaimissa. JavaScriptin virallinen nimi on kuitekin ECMAScript, joka tulee sitä kehittävän ja ylläpitävän ECMA-organisaation nimestä. ECMA-262-standardin virallinen kehitys on nykyisin edennyt jo versionumero 5:een asti, joka julkaistiin joulukuussa 2009. (ECMA-262 2009)

JavaScript on nimensä mukaisesti komentosarjakieli, joka tulkitaan ja suoritetaan asiakasympäristössä (Internet-selaimessa). JavaScript on yksi monista asiakasympäristöissä tulkittavista ja suoritettavista kielistä, mutta sen vahvuudet ovat alustariippumattomuus laajan Internet-selaintuen vuoksi, lähdekoodin tulkinnan ja suorituksen keveys sekä tehokkuus. Syntaksiltaan JavaScript muistuttaa löyhästi C-kieltä, joten se on myös helposti opittava kieli.
(Korpela 2009; Saarikumpu 2010)

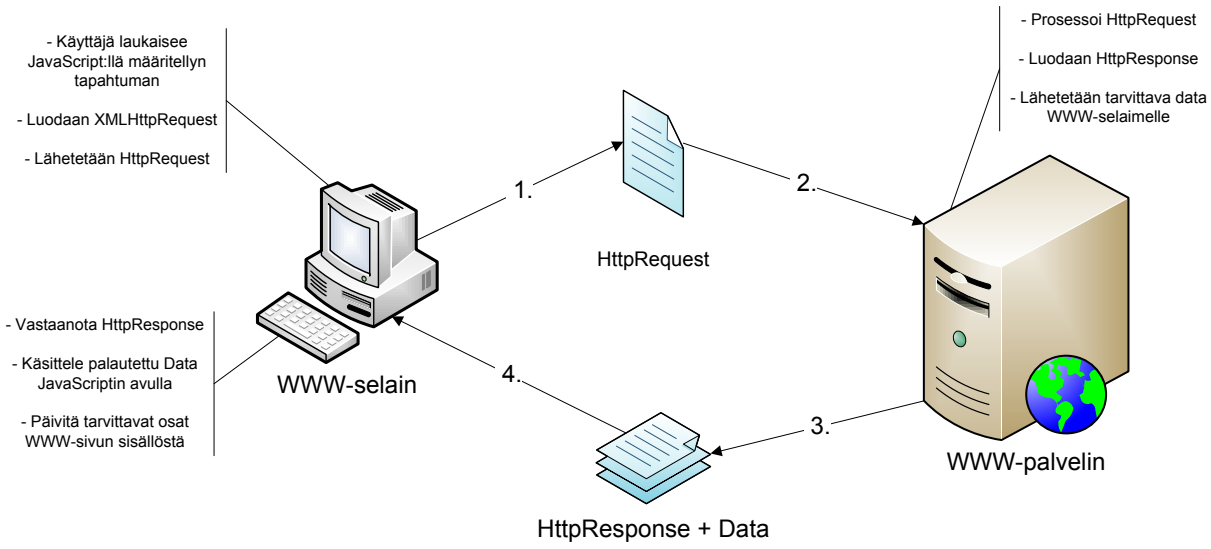
Yleensä JavaScriptiä käytetään lisäämään WWW-sivujen dynaamista toiminnallisuutta ja vähentämään ylimääräistä kuormitusta sivustoilla, joilla on paljon käyttäjiä, esimerkiksi uutissivustot. JavaScript-koodia voidaan upottaa normaaliin HTML-dokumenttiin (kuva 6.6) tai se voidaan erottaa omaksi ulkoiseksi JavaScript-tiedostoksi kuten CSS-tyylitiedostot. JavaScript-koodin tuottamiseen soveltuu mikä tahansa tekstieditori.

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fi">
<body>
    <script type="text/javascript">
        document.write("Hello World!");
    </script>
</body>
</html>
```

Kuva 6.6 Esimerkki HTML-dokumentista ja upotetusta JavaScript-koodista

6.12 Ajax

Ajax (Asynchronous JavaScript and XML) ei ole ohjelmointikieli vaan kokoelma hyväksi todettuja käytäntöjä olemassa olevista standardeista. Ajax mahdollistaa tiedon siirron palvelimen ja tietyn WWW-sivulla päivitettävän alueen välillä ilman, että Internet-selaimen tarvitsee ladata koko sivua uudelleen. Palvelin lähettää selaimelle vain tietyn osan sivusta, joka tarvitsee päivittämistä.
(Ajax, Refsnes Data 2010.)



Kuva 6.7 Ajaxin toimintaperiaate

Ajaxin käyttö vähentää Internet-selaimen ja WWW-palvelimen välistä turhan datan siirtämistä sekä nopeuttaa ja lisää dynaamisuutta Internet-sivuihin. Ajax pohjautuu täysin vallitseviin Internet-standardeihin ja se käyttää XMLHttpRequest -oliota datan välittämiseen selaimen ja palvelimen välillä asynkroonisesti, JavaScriptiä tai DOMia tiedon näyttämiseen ja interaktioon sekä XML:ää tiedonvälittämisen formaattina.

Taulukko 6.1 Yleisimmät Ajax-tekniikat

Tekniikka	**Rooli**
XMLHttpRequest	Käytetään tiedonsiirtoon WWW-selaimen ja -palvelimen välillä asynkroonisesti.
JavaScript / DOM	Käytetään toteuttamaan tarvittavat ohjelmalliset toiminnot ja niihin liitettävien tapahtumien käsittelyt.
XML	Käytetään siirrettävän datan formaatin kuvaamiseen.
XHTML	Käytetään sivun rakenteen ja sisällön esitystavan määrittelyyn.
CSS	Käytetään kuvaamaan siirrettävän sisällön muotoilut.

6.13 XML

XML (eXtensible Markup Language) on W3C:n määrittelemä looginen kuvauskieli eli merkkauskieli. XML-kieli on suunniteltu pelkästään tiedonsiirtoon ja jäsentämiseen, toisin kuin HTML-kieli, joka kuvaa Internet-sivujen rakennetta ja ulkoasua. XML-kielen tarkoituksena ei ole yksinään saada aikaan minkäänlaista toiminnallisuutta vaan XML-kielellä tehtyjen kuvausten ja omien rakenteiden tulkintaan ja käyttöön tarvitaan ohjelmallinen XML-jäsennin. Ulkoasultaan XML-kieli vastaa sitä lähellä olevaa HTML-kieltä. (XML, Refsnes Data 2010.)

Merkkauskielissä kuvataan yleisesti dokumentin muoto eli kuinka sisältöä lukeva jäsennin sitä tulkitsee. XML:ssä voidaan luoda omia elementtejä, joilla tietoa jäsennetään. Näin XML-kieli eroaakin HTML-kielestä, jossa kaikki käytettävissä olevat elementit on etukäteen määritelty eikä niitä ole tarpeeksi kaikkiin käyttötarkoituksiin. XML onkin itse asiassa metamerkkauskieli, jolla voidaan luoda omia räätälöityjä merkkauskieliä. (Holzner)

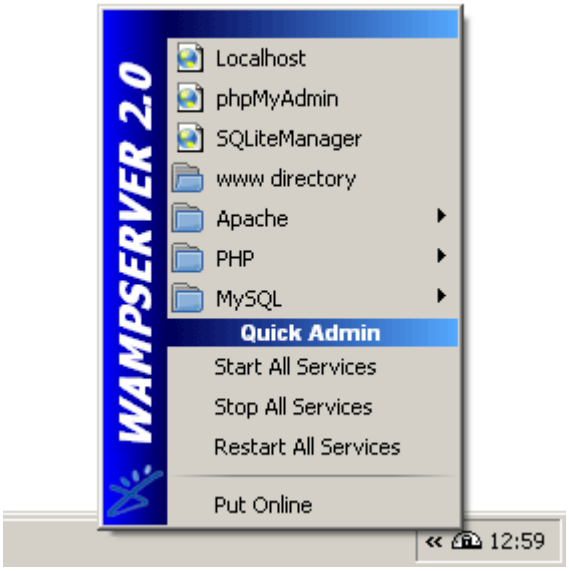
```
<!-- XML:n prosessointikäsky -->
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Juurielementin aloitus tunniste -->
<DOKUMENTTI>
<!-- Dokumentin sisältö kuvataan Juurielementin sisälle -->

    <!-- Sisältö elementin aloitustunniste -->
    <TERVEHDYS>
        Terveisiä XML:stä
    <!-- Sisältä elementin lopetus tunniste -->
    </TERVEHDYS>
    <VIESTI>
        Tervetuloa XML:n villiin ja vallattomaan maailmaan.
    </VIESTI>

<!-- Juurielementin lopetus tunniste -->
</DOKUMENTTI>
```

Kuva 6.8 Esimerkki yksinkertaisesta XML-dokumentista ja sen rakenteesta

Yksinkertainen XML-tiedosto alkaa XML:n prosessointikäskyllä, joka sisältää tarvittavat tiedot käytettävästä XML-versiosta ja dokumentissa käytetystä merkkikoodauksesta. Seuraavaksi määritellään XML-dokumentin juurielementti. XML:ssä jokaiseen elementtiin liittyy aloitus- ja lopetustunnisteet, jotka alkavat aina "<"-merkillä ja päättyvät ">"-merkkiin. Seuraavaksi sisältöä kuvaavat elementit kuvataan samalla tavoin juurielementin sisälle. Tällä tavoin olemme kuvanneet uudentyyppisen XML-dokumentin (kuva 6.8).

6.14 WampServer 2.0i

WampServer (**W**indows **A**pache **M**ySQL **P**hp) on nimensä mukaisesti Windows-alustalle tarkoitettu yleisimmät Internet-palvelut sisältävä ohjelmisto kokonaisuus. Se perustuu avoimen lähdekoodin ohjelmistoihin ja tarjoaa hyvän alustan PHP- ja MySQL -sovelluskehitykselle. WampServer 2.0 sisältää automaattisen asennusohjelman ja helppokäyttöisen graafisen käyttöliittymän sen sisältämien ohjelmistojen hallintaan.

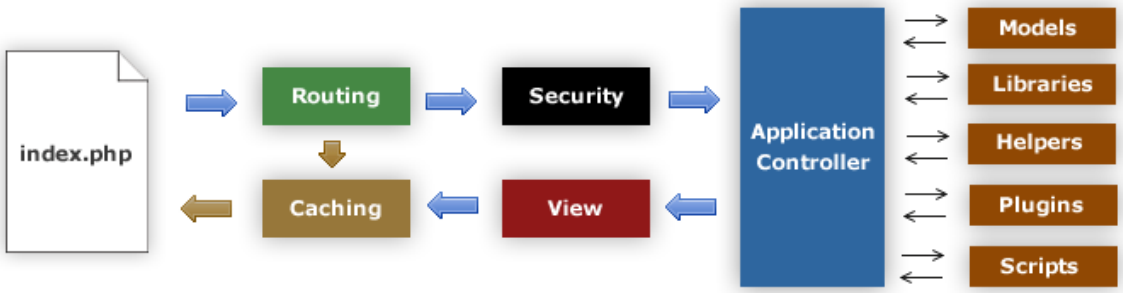


Kuva 6.9 WampServer 2.0i:n graafinen käyttöliittymä

WampServerin graafisen käyttöliittymän avulla voidaan helposti hallita eri osa-ohjelmistoja ja luoda tarvittaessa esimerkiksi symbolisia linkkejä Apachen omiin kehityshakemistoihin. Käyttöliittymä sisältää myös pikalinkit palveluiden käynnistämiseen, pysäyttämiseen ja uudelleen käynnistykseen sekä palvelun tilan vaihtamiseen. Palvelu voi olla joko online-tilassa, jolloin se näkyy myös paikalliseen verkkoon, tai offine-tilassa, joka rajoittaa palveluiden näkyvyyden isäntäkoneelle (localhost). WampServerin vahvuutena on helppo käytettävyys graafisen käyttöliittymän kautta, jolloin käyttäjän ei välttämättä tarvitse koskea ollenkaan ohjelmistojen asetustiedostoihin.

6.15 CodeIgniter

CodeIgniter on avoimeen lähdekoodiin perustuva PHP-sovelluskehys. CodeIgniter on pienikokoinen, yksinkertainen ja elegantti työkalukokonaisuus, jolla voidaan kehittää täysipainoisia Internet-sovelluksia. CodeIgniter on ominaisuuksiltaan tehokas, nopea ja hyvin dokumentoitu kokonaisuus, joka ei tarvitse monimutkaista konfigurointia ennen käyttöönottoa ja sovelluskehittämisen aloittamista ja oppimiskynnys sovelluskehyksen käyttöönotossa on matala. CodeIgniter noudattaa MVC-arkkitehtuurimallia toteutuksessaan, mutta antaa myös sovelluskehittäjälle mahdollisuuden poiketa siitä. CodeIgniter sisältää myös erittäin kattavan dokumentaation ja tuen kaikille yleisille tietokantojen hallintajärjestelmille (CodeIgniter).

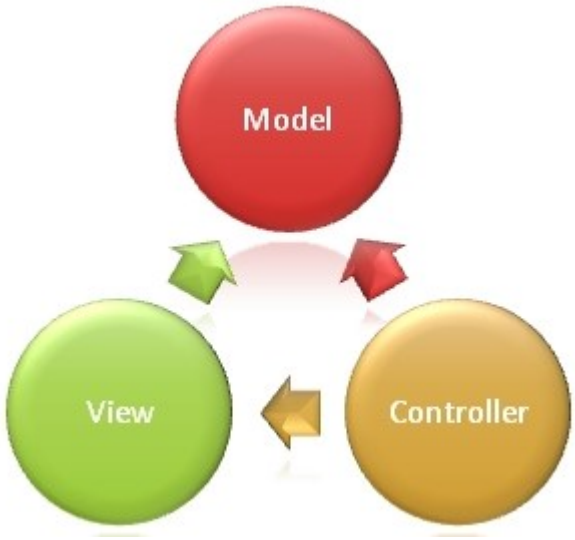


Kuva 6.10 CodeIgniter, MVC tietovuokaavio (CodeIgniter Application FlowChart)

Kuvassa 6.10 index.php toimii ns. pääkontrollina, joka ohjaa sovelluskehykselle tulevat pyynnöt eteenpäin varaten tarvittavat perusresurssit. Jos pyydettävä sivu löytyy välimuistista, lähettää reititys pyydetyn sivun asiakkaalle suoraan välimuistista. Normaalitapauksessa pyyntö viedään eteenpäin ohjelmakontrollille turvallisuustarkastuksien kautta. Turvallisuustarkastukset pitävät huolen perusturvallisuudesta suodattaen epäilyttävän käyttäjätiedon pyynnöistä. Seuraavaksi ohjelmakontrolli varaa tarvittavat tehtäväkohtaiset resurssit pyynnön suorittamiseen, suorittaa pyynnön vaatimien tietojen kokoamisen ja toteuttaa ohjelmalogiikan. Pyyntöön vastaamiseksi renderöidään sivu(näkymä), joka lähetetään pyynnön tehneelle asiakkaalle. Jos välimuisti on käytössä, talletetaan valmis näkymä välimuistiin seuraavaa identtistä pyyntöä varten. Näin säästetään

resursseja ja pystytään vastaamaan asiakaspyyntöihin nopeammin. Lopuksi vastaus lähetetään pyynnön tehneelle asiakkaalle.
(CodeIgniter Application FlowChart)

Model-View-Controller eli MVC-arkkitehtuurimalli on yksi ajansaatossa hyväksi havaittu ohjelmistotekninen suunnittelumalli, joka jakaa ohjelman kolmeen pääkomponenttiin (ks. kuva 6.11): malliin (model), näkymään (view) ja ohjaimeen (controller). Trygve Reenskaug kehitteli MVC-arkkitehtuurimallin 1980-luvun alussa työskennellessään Smalltalk-kielen parissa Xerox PARC:lla. MVC-arkkitehtuurin tekemällä komponentteihin jaolla on tarkoitus erotella erilaiset ohjelmiston näkökulmat (input logic, business logic, user interface logic) ja ongelmat toisistaan helposti hallittaviin komponentteihin, jotka ovat vaihdettavissa ja uudelleen käytettävissä vähäisen kytkennän ansiosta.
(MVC Overview; MVC Wikipedia)



Kuva 6.11 MVC-arkkitehtuurimalli (MVC Overview)

MVC-malli määrittelee, missä mikäkin tyyppinen logiikka sijaitsee. Käyttöliittymän logiikka eli esitystapa sijaitsee näkymässä, syötteiden käsittely ohjaimessa ja bisneslogiikka mallissa. Tämän tyyppinen jaottelu auttaa ohjelmoijaa hallitsemaan kehitettävän ohjelmiston monimutkaisuutta, koska se mahdollistaa keskittymisen yhden osa-alueen komponentin toteuttamiseen kerrallaan.

MVC-mallin rakenteen vähäinen kytkentä mahdollistaa myös ohjelman samanaikaisen kehittämisen, esimerkiksi yksi kehittäjä voi työstää mallia, toinen ohjainta ja kolmas näkymää samanaikaisesti.

CodeIgniter perustuu oliopohjaiseen ohjelmointiin. Se tarjoaa kaikki tarvittavat perusedellytykset täysipainoisien WWW-sovellusten kehittämiseen. Sovelluskehittäjä voi laajentaa ja kirjoittaa tarvittaessa uusia uudelleen käytettäviä kirjastoja ja avustajia sekä laajentaa sovelluskehyksen ydinluokkien toiminnallisuutta. Kaikissa sovelluskehyksellä tehdyissä töissä ja sovelluksissa on oltava viittaus käyttölisenssiin ja niin, että lisenssin käyttöehdot täyttyvät. Kaikki uudet kirjastot ja avustajat kannattaa dokumentoida hyvin ennen niiden käyttöönottoa.

Kuvassa 6.2 on yksinkertainen esimerkki CodeIgniterilla tehdystä ohjain-luokasta, joka käyttää henkilö-mallia tiedon hakuun ja esimerkki-näkymää tiedon esittämiseen.

```php
<?php
class Esimerkki extends Controller
{
    /**
     *  Muodostimet
     **/
    function Esimerkki()
    {
        // Suoritetaan yliluokan muodostin
        parent::Controller();
    }

    /**
     *  Kontrolli-luokan metodit
     **/
    // Automaattisesti ajettava Index-metodi
    function index()
    {
        // Ladataan tarvittavat kirjastot
        $this->load->library(array('database'));     // Tarvitaan tietokantayhteys
        // Ladataan tarvittavat avustajat
        $this->load->helper(array('url'));           // Tarvitaan url avustajaa
        // Ladataan tarvittavat mallit
        $this->load->model(array('henkilo_model')); // Tarvitaan henkilö mallia
        // Määritellään hakuehdot
        $options['sort_order'] = 'asc';
        $options['sort_by'] = 'Sukunimi';
        // Haetaan henkilöt tietokannasta
        $data['henkilolista'] = $this->henkilo_model->get($options);
        // Asetetaan otsikko kentät
        $data['otsikko'] = array('ID', 'Sukunimi', 'Etunimi');
        // Luodaan näkymä
        $this->load->view('esimerkki_view', $options);
    }
}
/* End of file esimerkki.php */
/* Location: ./system/application/controllers/esimerkki.php */
```

Kuva 6.12 Esimerkki CodeIgniterin ohjain-luokasta

Kuva 6.13 sisältää ohjaimen käyttämän yksinkertaisen mallin esimerkin, joka hakee tietokannasta henkilöitä ja järjestelee ne valmiiksi määriteltyjen ehtojen mukaisesti.

```php
<?php
class Henkilo_model extends Model
{
    /**
     *  Muodostimet
     **/
    function Henkilo_model()
    {
        parent::Model();    // Suoritetaan yliluokan muodostin
    }

    /**
     *  Malli-luokan metodit
     **/
    // Get-metodi hakee tietokannasta henkilö-taulusta kaikki tietueet
    function get($options = array())
    {
        // Tarkastetaan järjestys ehdot
        if (isset($options['sort_order']) && isset($options['sort_by']))
        {
            // Järjestys ehto
            $options['sort_order'] =
                ($options['sort_order'] == 'desc') ? 'desc' : 'asc';
            // Järjestyksen kentät
            $sort_columns = array('HenkiloID', 'Etunimi', 'Sukunimi');
            // Järjestyksen kenttä ehto
            $options['sort_by'] =
                (in_array($options['sort_by'], $sort_columns))
                    ? $options['sort_by'] : 'Sukunimi';
        }
        else
        {
            $options['sort_order'] = 'asc';
            $options['sort_by'] = 'Sukunimi';
        }
        // Kootaan SQL-lause
        $this->db->select('HenkiloID, Etunimi, Sukunimi')
                 ->from('henkilo')
                 ->order_by($options['sort_by'], $options['sort_order']);
        // Haetaan tietueet tietokannasta
        $data['rows'] = $this->db->get()->result();
        // Lasketaan tietueiden lukumäärä
        $data['num_rows'] = $data['rows']->ArrayObject.count;
        // Palautetaan kerätyt tiedot kontrolli-luokalle
        return $data;
    }
}
/* End of file henkilo_model.php */
/* Location: ./system/application/controllers/henkilo_model.php */
```

Kuva 6.13 Esimerkki CodeIgniterin malli-luokasta

Kuvassa 6.14 on käyttöliittymälogiikka eli tapa, jolla toiminnon tuottama sisältö esitetään käyttäjälle.

```
<?php
    /* Ladataan header elementti */
    // Sisältää HTML-koodin header-tagin lopetukseen saakka.
    $this->load->view('element/header');
    /* Ladataan menu elementti */
    // Sisältää HTML-koodin sivun sisällön div-tagiin saakka.
    $this->load->view('element/menu');
?>
<div id="content">
<h1>Henkilöt</h1>
<p>
<?php
    // Tulostetaan taulun aloitus
    echo "<b>Tietokannasta löytyi" , $henkilolista['num_rows'] , "henkilöä.</b><br />";
    echo '<table><thead>';
    // Tulostetaan otsikkosarakkeet
    foreach ($kentat as $header)
    {
        echo '<th><b>' , $header , '</b></th>';
    }
    // Tulostetaan välitagit
    echo '</thead><tbody>';
    // Tulostetaan sisältörivit
    foreach ($henkilolista['rows'] as $rows)
    {
        echo '<tr><td>' , $rows->HenkiloID , '</td>';
        echo '<td>' , $rows->Sukunimi , '</td>';
        echo '<td>' , $rows->Etunimi, '</td></tr>';
    }
?>
</p>
</div>
<?php
    /* Ladataan footer elementti */
    // Sisältää HTML-koodin alatunnisteesta ja lopetus tagit.
    $this->load->view('element/footer');
?>
```

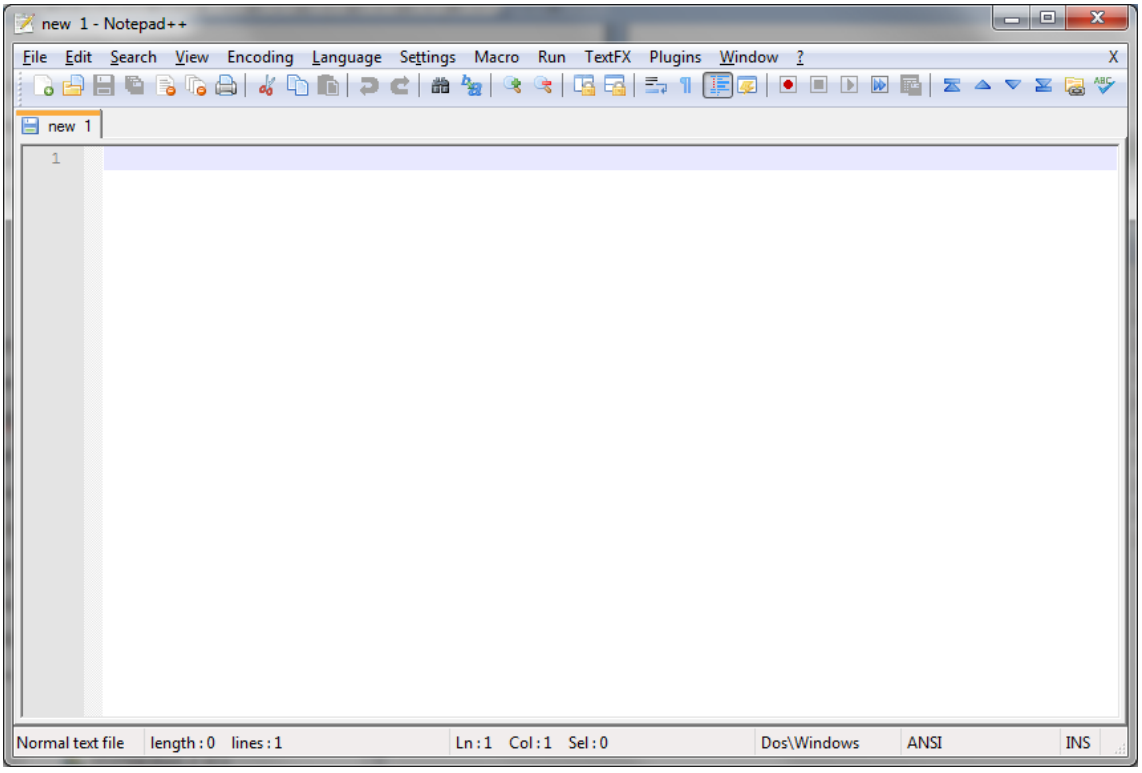
Kuva 6.14 Esimerkki CodeIgniterin näkymästä

Koska CodeIgniterissa lähdekoodi jaotellaan osiin MVC-arkkitehtuurimallin mukaisesti (ks. kuvat 6.12, 6.13 ja 6.14), on tärkeää, että tekijä dokumentoi järjestelmällisesti lähdekooditiedostoihin riittävän kuvauksen: huomiot ja kommentit koodista sekä tarvittaessa tiedostonimen ja polun. Tällöin toiminnallisuuden ja lähdekoodin jaottelu eri tiedostoihin on hallittavissa. MVC-arkkitehtuurissa on hyvä käytäntö tehdä ohjainluokkien toteutuksista yksinkertaisia ja vain tarvittavan ohjelmalogiikan ja käyttäjäsyötteiden tarkastuksen sisältäviä. Malliluokista on tarkoitus tehdä suuria ja paljon toiminnallisuutta, tietokannan käsittelyä ja liiketoimintalogiikkaa sisältäviä. Tämä menettely helpottaa kehitettyjen toiminto-

jen uudelleenkäytettävyyttä ja hallintaa jatkossa sekä vähentää ohjain ja malliluokkien välistä kytkentää. Näkymät voidaan räätälöidä täysin käyttötarpeisiin perustuviksi tekemällä kontrollin ja näkymän välille oma perusrajapinta, kuinka tieto näiden kahden välillä kulkee.

6.16 Notepad++

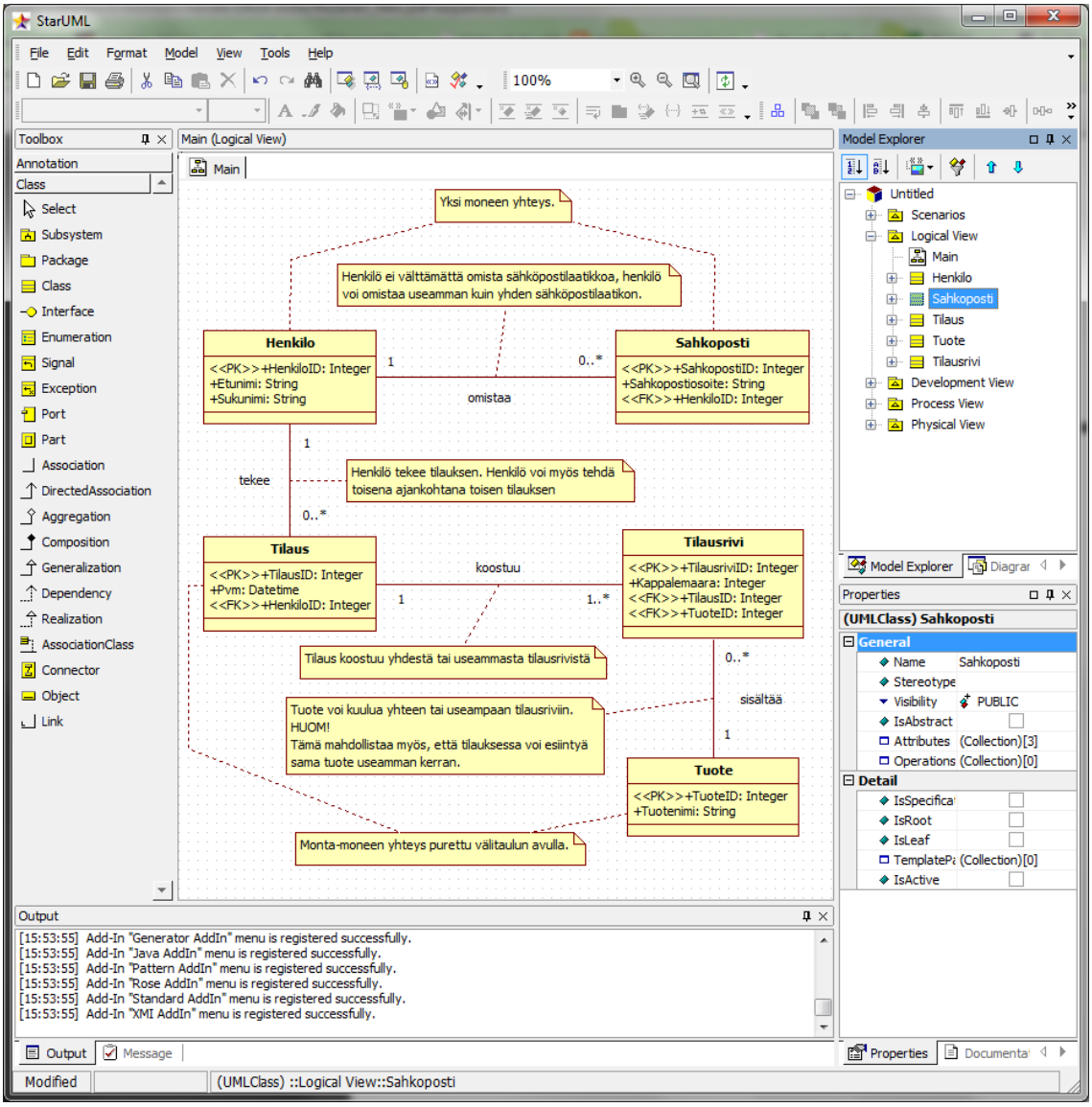
Notepad++ on ilmainen lähdekoodin editointityökalu. Se korvaa muistion Windows-ympäristöissä ja tukee useita ohjelmointikielien syntakseja. Se pohjautuu erittäin tehokkaaseen Scintilla-komponenttiin, Win32 API:n ja STL-kirjastoihin, jotka takaavat nopeamman suoritustehon ja pienemmän ohjelmatiedostojen koon. Tällä hetkellä uusin versio ohjelmasta on 5.8.2, joka saattaa vielä sisältää joitakin virheitä. Ohjelmaa kehitetään kuitenkin aktiivisesti ja päivitykset ohjelman erikomponentteihin pystytään lataamaan automaattisesti ohjelman sisältämän päivitystyökalun avulla. Notepad++ on siis erittäin kevyt ja hyvä ohjelma aloittaa uuden ohjelmointikielen opiskelu. Notepad++ ohjelman käyttöä säädellään GPL-lisenssin avulla. (Notepad++ project, GNU GPL-license)



Kuva 6.15 Notepad++ -käyttöliittymä

6.17 StarUML

StarUML on vapaan lähdekoodin vastine kaupallisille UML-mallinnusohjelmistoille. Sen tarkoituksena on myös kilpailla kaupallisten ohjelmistojen kanssa. Ohjelman nimeä kantavan projektin tavoitteena on ollut luoda ja kehittää nopea, joustava ja laajennettava UML-työkalu Windows-alustalle. (Kylä-Nikkilä 2008)

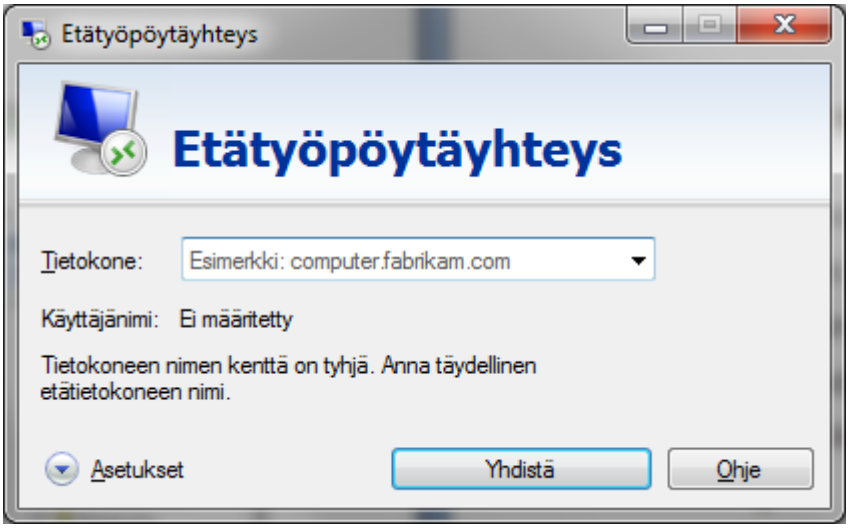


Kuva 6.16 StarUML:n käyttöliittymä ja luokkakaavio esimerkki

StarUML tukee UML 2.0 standardin lisäksi myös OMG:n kehittämää ja ylläpitämää MDA:ta. (Kylä-Nikkilä 2008)

6.18 Etätyöpöytäyhteys

Etätyöpöytäyhteys (Remote Desktop Connection) perustuu Windows-ympäristössä Microsoftin RDP-protokollaan, joka tarjoaa käyttäjälle mahdollisuuden työaseman etäkäyttöön Terminal Service -palvelun avulla. Terminal Service -palvelu on ollut osa Microsoftin palvelinkäyttöjärjestelmiä, jo Windows NT:n ajoilta 1980-luvulta asti. Etätyöpöytäyhteys on käytettävissä osassa Microsoftin käyttöjärjestelmiä. Windows XP Professional-, Windows Vista- ja Windows 7-käyttöjärjestelmäversioissa etätyöpöytäyhteys tukee vain yhden käyttäjän kirjautumista tietokoneelle kerrallaan. Microsoftin palvelin käyttöjärjestelmissä samaa tietokonetta voi käyttää useampi samanaikainen käyttäjä.
(Kinnunen 2010)



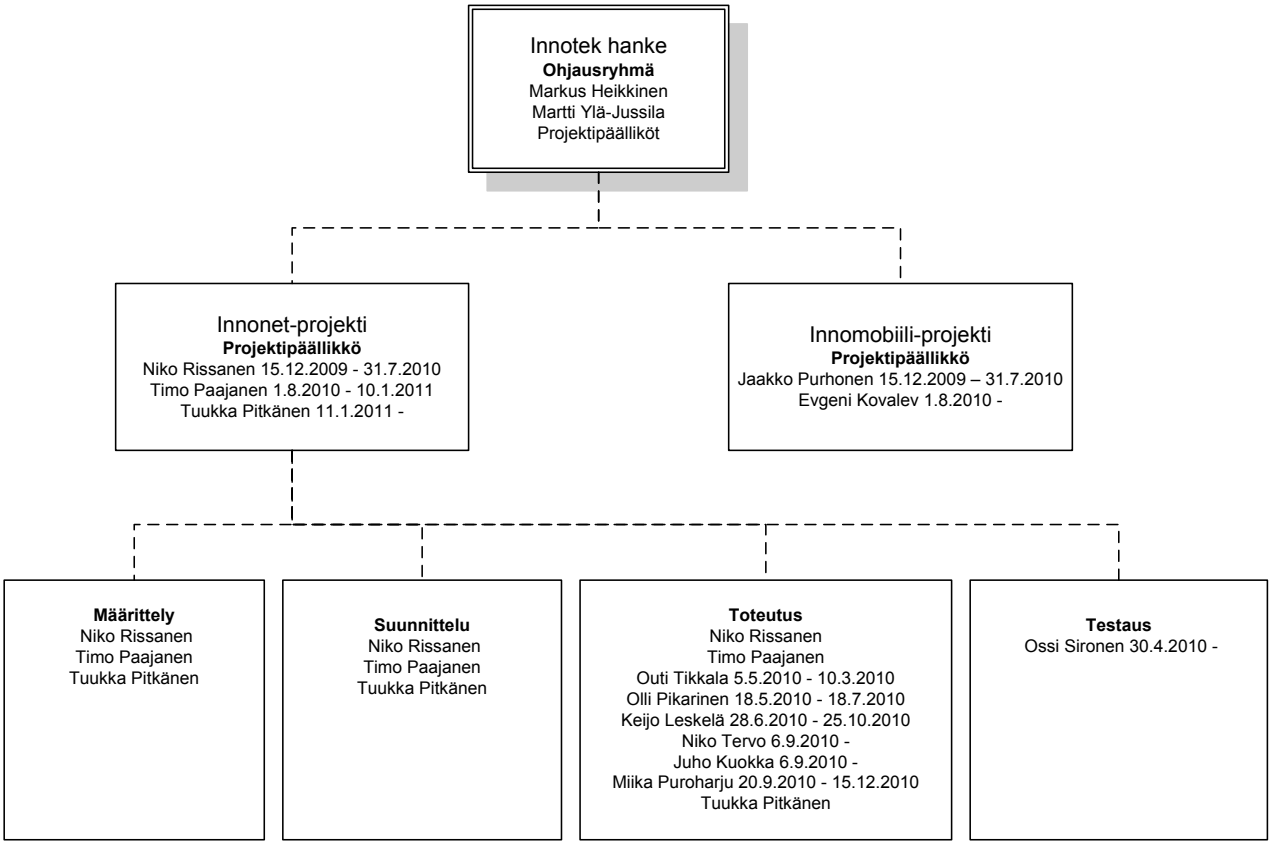
Kuva 6.17 Etätyöpöytäyhteyssovellus Windows 7:ssä

7 PROJEKTIN KULKU

Tässä luvussa on kuvattu Innonet-hankkeen ensimmäisten vaiheiden projektiorganisaatio ja eri vaiheiden työnkulut.

7.1 Projektin organisaatio

Kuvassa 7.1 on esitelty Innotek-hankkeen projektiorganisaatio ja henkilöstö.



Kuva 7.1 Innotek-hankkeen organisaatiokaavio

7.2 Projektin ohjaus

Innonet-projektissa projektin ohjaus perustui esitutkimusvaiheen jälkeen laadittuun projektisuunnitelmaan. Projektin ohjaus toteutettiin pitämällä kuukausittain ja tarvittaessa asiakaspalavereita sekä projektiryhmän viikkopalavereja. Asiakaspalavereissa projektiryhmä keskusteli asiakkaan kanssa projektiin liittyvistä

asioista, hakien vastauksia syntyneisiin kysymyksiin ja ratkoen esiin tulleita ongelmia.

Projektiryhmän kesken pidettiin viikkopalavereita, joiden tehtävänä oli antaa projektipäällikölle kuva projektin tilanteesta. Tekijät valmistautuivat viikkopalavereihin päivittämällä projektin työmääräarvion omalta osaltaan ajan tasalle. Viikkopalavereissa aihealueina olivat edellisen jakson työt, havaitut ongelmat, riskit ja seuraavan jakson työt. Viikkopalavereissa keskusteltiin myös muista projektiin liittyvistä asioista.

Viikkopalaverien ja asiakaspalaverien päällimmäisenä tarkoituksen oli seurata projektin edistymistä, hallita riskejä ja ratkoa ongelmia yhdessä projektiryhmän kesken. Kaikista palavereista kirjoitettiin palaverimuistio, joka mahdollisti tarvittaessa asioihin palaamisen myöhemmin. Jokaiseen palaveriin valittiin kirjuri, joka vastasi muistiinpanojen tekemisestä ja dokumentoinnista palaverimuistioksi. Muistio lähetettiin kaikille projektiin osallistuville henkilöille sähköpostilla. Tämä mahdollisti tiedon kulun myös palaverista poissa olleille henkilöille.

Projektin alkuvaiheessa palavereita pidettiin samassa tilassa, jossa kaikki osallistuvat pystyivät keskustelemaan vapaasti kasvotusten. Myöhemmin syksyllä otettiin käyttöön Adobe Connect Pro -ohjelmisto, jonka avulla palavereihin pystyi osallistumaan Internet-yhteyden avulla.

7.3 Esiselvitys

Prosessi lähti liikkeelle helmikuun puolivälissä 2010. Liityin aiemmin meneillään olleeseen projektiin, johon tarvittiin lisätyövoimaa alustavan projektisuunnitelman työmääräarvion perusteella. Projektia oli tuolloin opinnäytetyönä tekemässä samalla luokalla ollut opiskelija, Niko Rissanen, joka oli asetettu kyseisen opinnäytetyöprojektin päälliköksi. Liityin projektiin suunnittelijan rooliin. Tässä vaiheessa projektissa oli tehty esitutkimus ja alustava projektisuunnitelma. Asiakkaan puolelta projektin tehtävänantona oli vanhan ja keskeneräisen Innonet-järjestelmän jatkokehittäminen.

Itse projektiin liittyvät työt aloitettiin tutustumalla olemassa olevaan järjestelmään ja siihen liittyvään dokumentaatioon sekä projektissa syntyneisiin dokumentteihin. Tässä vaiheessa opinnäytetyön tavoitteena oli kehittää nykyisen Innonet-järjestelmän käyttöliittymää helppokäyttöisemmäksi ja yksinkertaisemmaksi asiakkaan toiveiden mukaisesti. Heti alkuvaiheessa huomattiin, että vanha järjestelmä on todella hankala hahmottaa ja käyttää. Ongelmaa vaikeutti entisestään se, ettei vanhaa järjestelmää ja sille asetettuja vaatimuksia eikä liiketoiminnallisia prosesseja ollut dokumentoitu. Vain muutamat järjestelmän rakennetta ja yleisiä vaatimuksia kuvaavat dokumentit ja lähdekoodit olivat saatavilla. Tämän huomattuamme tarkastelimme tietokantaa pinnallisesti ja huomasimme, ettei se vastannut asiakkaan järjestelmälle asettamiin alustaviin vaatimuksiin, lisäksi se oli toteutettu monessa suhteessa epäjohdonmukaisesti.

Järjestelmän yksityiskohtainen tutkimus ja kartoitus osoittivat, ettei vanhaa järjestelmää ollut mielekästä ruveta korjaamaan ja dokumentoimaan epäjohdonmukaisuuksien ja virheellisten liiketoimintamallien vuoksi. Edellä mainittujen syiden pohjalta päädyimme ohjausryhmässä siihen johtopäätökseen, ettei järjestelmän jatkokehittäminen ole järkevää. Totesimme, että ennen kuin järjestelmää alettaisiin suunnitella ja toteuttaa, olisi saatava järjestelmälle asetettavat vaatimukset ja liiketoiminnalliset prosessit dokumentoitua, tarkennettua ja tarkastettua asiakkaan kanssa.

7.4 Määrittely

Ensimmäisessä asiakaspalaverissa asiakasta pyydettiin kertomaan yksityiskohtaisesti esimerkkitapauksin selventäen järjestelmälle asetettavat vaatimukset ja niiden takana oleva liiketoiminta. Työ eteni näiden palaverin pohjalta saatujen tietojen ja vaatimusten dokumentoinnilla sekä toiminnallisen määrittelydokumentin laatimisella. Kun keskeiset osa-alueet vaatimuksista oli jaoteltu hallittaviin osioihin, jatkettiin järjestelmän määrittelyä tietokannan ja käsitteiden kuvaamisella. Samalla laadittiin alustava suunnitelma tietokannasta. Seuraavissa asiakaspalavereissa käytiin läpi dokumentoituja asioita ja tarkennettiin alustavia suunnitelmia sekä rajattiin järjestelmän kokonaisuutta.

Palavereissa todettiin, että asiakkaan järjestelmälle kohdistamat vaatimukset kattavat huomattavasti suuremman osan yrityksen liiketoiminnallisista prosesseista kuin alunperin ymmärrettiin. Myös kehitettävään Innonet-järjestelmään liittyvä mobiililaitteessa toimiva Innomobiili-sovellus oli vahvasti kytköksissä Innonetiin ja yhteiseen tietokantaan. Tämän vuoksi päätimme tehdä vaatimusten mukaisen tietokannan suunnittelun yhdessä Innomobiili-projektia tehneen Jaakko Purhosen kanssa. Töiden ja vaatimusten dokumentoinnin edetessä ilmeni useita kysymyksiä ja ongelmia, joihin haettiin ratkaisuja niin asiakaspalavereista kuin asiakkaalle lähetettyjen kysymyslistojen avulla. Tietyt kysymykset olivat tärkeitä ja saatujen vastauksien perusteella jo aiemmin dokumentoituja vaatimuksia jouduttiin pohtimaan uudelleen ja tarkentamaan. Suurinta osaa näistä asioista ei ollut otettu lainkaan huomioon vanhassa järjestelmässä.

Määrittelyn edetessä huomattiin, ettei järjestelmää ole mahdollista toteuttaa valmiiksi kahteen opinnäytetyöhön varatun ajan puitteissa ja projektiin tarvittaisiin lisää työvoimaa.

7.5 Suunnittelu ja toteutus

Alkukesästä 2010 projektiin liittyivät Outi Tikkala ja Olli Pikarinen sekä myöhemmin kesällä Keijo Leskelä. Heidän tehtävänään projektissa oli suunnitella ja aloittaa järjestelmän toteutus. Toteutuksen ohjelmistokehykseksi oli jo aiemmassa vaiheessa valittu kevyt, nopea ja yksinkertainen PHP-sovelluskehys CodeIgniter. Sen päällimmäisenä tarkoituksena oli tarjota toteutukselle perusarkkitehtuuri ja hyväksi havaittu toteutusmalli, joka jaottelee toteutuksen tietokannankäsittelyn, liiketoimintalogiikan ja tiedon esitystavan omiin osioihinsa. Tästä oli se hyöty, että toteutus voitiin osittaa ja jakaa rajattuihin osa-alueisiin. Lisäksi sovelluskehys oli hyvin dokumentoitu ja nopea oppia yksinkertaisen rakenteensa ansiosta. Toteutuksen suunnittelussa ja ohjelmoinnissa tarvittiin WWW-palvelinta. Tarvittavat WWW-palvelut pystytettiin, jotta toteutuksen suunnittelu ja sovelluskehykseen tutustuminen saatiin liikkeelle. Jatkossa palvelinta käytettiin toteutuksessa ja testauksessa keskitettynä varastona, johon tekijät pääsivät tekemään ja toteuttamaan määrittelyn mukaista järjestelmää.

Kesällä 2010 toiminnallista määrittelyä ja toteutuksen suunnittelua jatkettiin samanaikaisesti. Uusien tekijöiden liityttyä projektiin syntyi tarve päivittää aiempi projektikohtainen työmääräarvio, jotta projektin edistymistä ja projektiin käytettävää työmäärää voitaisiin seurata täsmällisesti projektin aikana. Projektiin liittyneiden uusien tekijöiden opastamiseen jouduttiin käyttämään paljon aikaa. Tehtäviä jaettiin viikoittain ja niiden suorittamista seurattiin projektiryhmän viikkopalavereissa. Nämä aikaa ja resursseja kuluttavat tehtävät pyrittiin ottamaan mahdollisimman hyvin huomioon, kun työmääräarviota ja projektisuunnitelmaa päivitettiin.

Kesäkuun 2010 lopulla opinnäytetöihin varattu aika täyttyi, mutta lupauduin jatkamaan projektissa projektipäällikön roolissa joulukuun loppuun asti, koska edellisen syksyn poissaolon johdosta en ollut suorittanut neljännen vuosikurssin opintoja ja tulisin suorittamaan ne tulevan syksyn 2010 syyslukukauden ja kevään 2011 lukukauden aikana. Elokuussa 2010 olin projektissa ainoana tekijänä ja päätin keskittyä syyslukukauden opintoihin, projektissa tein vain projektipäällikön viikoittaiset tehtävät.

Projektiin liittyivät syyskuun 2010 alussa Niko Tervo, Juho Kuokka ja Miika Puroharju, joiden tehtäväksi annettiin aluksi tutustua kunnolla projektissa syntyneeseen dokumentaatioon ja projektin käytäntöihin. Syyslukukaudella 2010 ongelmaksi projektin etenemisen kannalta muodostui tekijöiden samanaikaisesti suorittamat syyslukukauden opinnot, jotka jakoivat keskittymisen opintojen ja projektin kesken. Projektiryhmä käytti projektiin liittyvän työajan järjestelmän dokumentaation, projektikäytäntöjen ja toteutettujen osien opiskeluun.

Projektiin etsittiin jatkuvasti uusia henkilöitä, jotka voisivat olla mukana projektissa useita kuukausia ja ottaisivat joulun jälkeen projektipäällikön tehtävät vastuulleen. Projektiin liittyi 1.11.2010 Tuukka Pitkänen, joka suoritti projektissa ensin työharjoittelua ja jatkoi 1.12.2010 alkaen opinnäytetyötä. Tuukalle määriteltiin aluksi tehtäväksi tutustua dokumentaatioon ja järjestelmään sekä valmistautua joulun jälkeen ottamaan projektipäällikön tehtävät ja johtamaan projektia.

11.1.2011 lähtien projektipäällikkönä on projektissa toiminut Tuukka Pitkänen ja hänen lisäksi projektiryhmässä ovat toimineet suunnittelijoina Alex Grönholm, Juho Kuokka, Niko Tervo ja Outi Tikkala. Itse olen osallistunut projektiin vain satunnaisesti konsulttina.

Kuukausia kestäneen opiskeluvaiheen jälkeen projektin toteutus on nyt vauhdittunut, mutta projektissa on vielä töitä useiden miestyökuukausien verran.

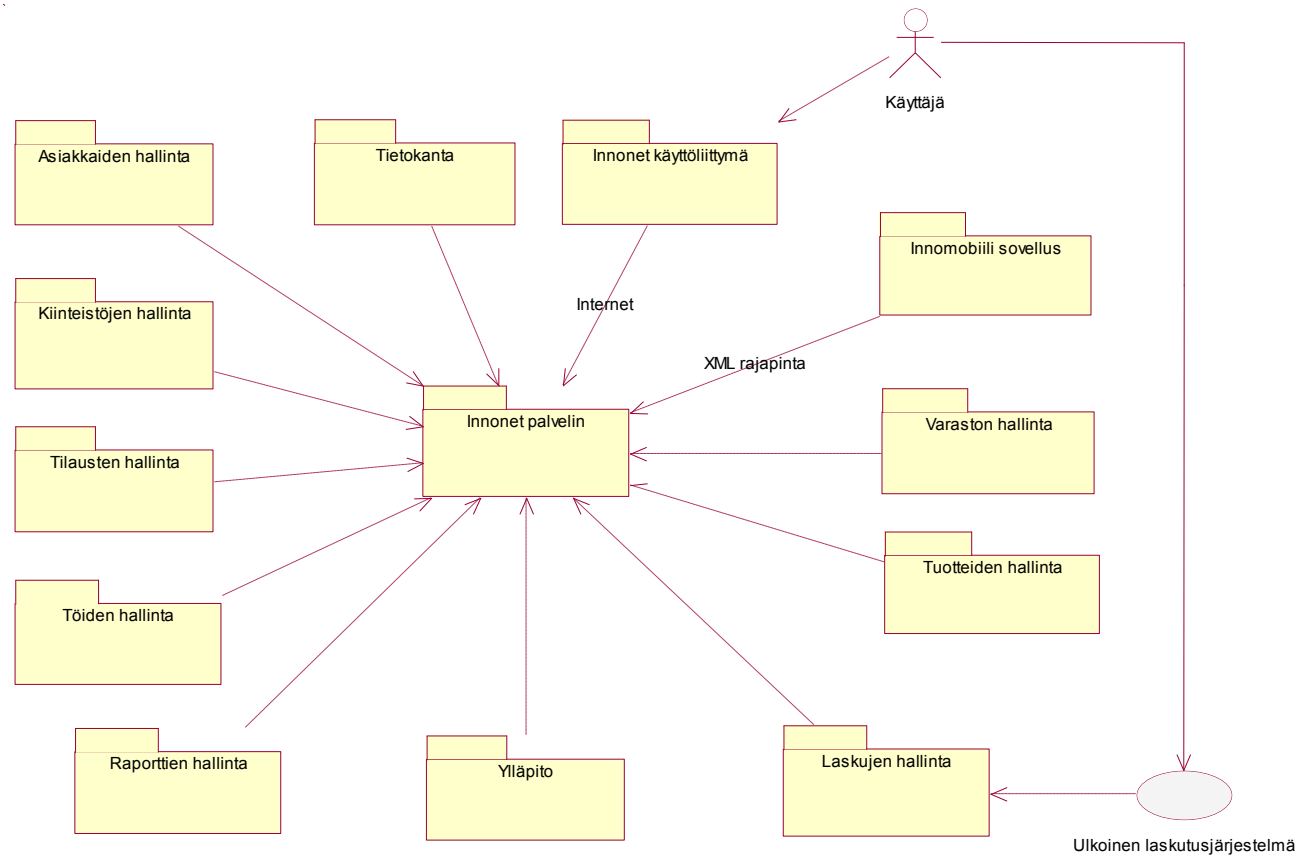
8 INNONET-JÄRJESTELMÄN ESITTELY

Uuden Innonet -järjestelmän on tarkoitus helpottaa, tehostaa ja ohjata Innotek Oy:n työntekijöiden päivittäisiä tehtäviä. Järjestelmällä hallitaan asiakkaita, kiinteistöjä, tilauksia, työnantoja, raportteja, tuotteita, varastoja ja laskuja sekä järjestelmän ylläpitotoimintoja. Järjestelmän toimintaan liittyy vahvasti Innomobiili-sovellus, joka on Innotek Oy:n asentajien käyttöön tarkoitettu mobiili tiedonkeruu- ja töidenohjaussovellus.

Innomobiili-sovelluksen avulla asentajat saavat tilauksista nyt työnannot. Asentaja suorittaa työnannossa määritellyt suoritustiedot ja samalla sovellus tallettaa tarvittavat tiedot työnantoon liittyvistä töistä ja tehtävistä. Innomobiili-sovellus kommunikoi Innonet-järjestelmän kanssa yhteisen rajapinnan avulla. Rajapinnan kautta haetaan ja talletetaan kaikki työnantoihin liittyvät tiedot yhteiseen tietokantaan. Innonet-järjestelmästä voidaan generoida työnantojen pohjalta erityyppisiä raportteja asiakas tilaukseen liittyvistä töistä, jotka ovat osa Innotek Oy:n Energo-säästöohjelmaa.

Innonet on WWW-pohjainen toiminnanohjausjärjestelmä. Se toteutetaan PHP-ohjelmointikielellä käyttäen pohjana CodeIgniter PHP -sovelluskehystä. Järjestelmän käsittelemä tieto talletetaan MySQL-tietokantaan käyttäen InnoDB-tietokantamoottoria, joka tukee transaktioita ja sisältää kunnolliset viiteeheyksien tarkistukset.

Seuraavassa on esitelty lyhyesti Innonet-järjestelmän yleisarkkitehtuuri ja sen osat.



Kuva 8.1 Innonet-järjestelmän looginen rakenne

8.1 Asiakkaiden hallinta

Asiakkaiden hallinnan tarkoituksena on käsitellä ja säilyttää Innotek Oy:n asiakastietoja. Asiakkaita voidaan lisätä, muokata, hakea ja poistaa sekä tarvittaessa asiakastietoja voidaan tarkastella tarkemmin.

8.2 Kiinteistöjen hallinta

Kiinteistöjen hallinnassa pidetään listaa järjestelmään lisätyistä asiakkaiden kiinteistöistä. Kiinteistöjä voidaan hakea, tarkastella, muokata ja poistaa. Kiinteistöihin liittyy myös Innomobiili-sovelluksessa kerätyt tarkemmat tiedot kiinteistön rakennuksista, tiloista ja huoneista, joissa työnantoihin liittyvät tehtävät suoritetaan.

8.3 Tilausten hallinta

Tilausten hallinnassa lähdetään liikkeelle tarjouksesta. Myyjä laatii asiakkaalle tarjouksen, ja jos asiakas hyväksyy tarjouksen, voidaan tarjouksesta generoida tilaus. Tilaukseen liittyvät työnannot, jotka myyjä laatii asentajien aikataulujen mukaisesti. Tässä vaiheessa tilauksen työnanto on ladattavissa Innomobiili-sovellukseen suoritettavaksi. Tilausten hallinta on jaoteltu kahteen osioon, tarjouksiin ja tilauksiin. Tarjouksissa käyttäjä voi laatia uuden tarjouksen, tarkastella tarjouksen tietoja, muokata tarjousta ja hakea tarjouksia sekä generoida tarjouksesta tilauksen. Tarjoukset listataan aloitusnäkymässä niin, että oleellisimmat ja ajankohtaisimmat tarjoukset näytetään ensimmäisenä. Tilauksissa käyttäjälle tuodaan esiin ajankohtaiset tilaukset. Käyttäjä voi luoda uusia tilauksia, muokata tiettyä tilausta ja laatia tilauksesta työnantoja sekä hakea tiettyä tilausta.

8.4 Töiden hallinta

Töiden hallintaosion avulla käyttäjä hallitsee omaa työaikatauluaan ja näkee meneillään olevat työnannot. Töiden hallinnassa on myös mahdollisuus suorittaa työnantoja, jos Innomobiili-sovellus tai Pocket PC -laite on epäkunnossa. Työnannot esitetään käyttäjälle kalenterinäkymässä käyttäen värikoodeja työnantojen päällekkäisyyksien ja tilojen kuvaamiseen.

8.5 Raporttien hallinta

Raporttien hallinnan tavoite on vähentää paperityön osuutta ja automatisoida tilauksista tehtävien Energo-säästöohjelmaan liittyvien raporttien tekoa. Raporttien hallinnassa voidaan hakea, selata ja tarkastella raportoituja tilauksia. Raportoidut tilaukset ovat tilauksia, joista on suoritettu työnantoja, tilaus voi sisältää yhden tai useampia työnantoja. Työnannot muodostavat yhdessä tiedot, joista järjestelmä generoi käyttäjälle erityyppisiä raportteja. Raporttien tyyppi määräytyy tilaukseen liittyvän palveluntyypin mukaisesti.

8.6 Laskujen hallinta

Laskujen hallinnassa tuotetaan laskutettavista tilauksista vientitiedostoja ulkoiselle laskutusjärjestelmälle. Vientitiedostot generoidaan ulkoisen laskutusjärjestelmän määrittämän rajapinnan mukaisesti. Vientitiedostot jatkokäsitellään ulkoisessa laskutusjärjestelmässä, eikä Innonet-järjestelmä käsittele laskutukseen liittyvää tietoa tämän enempää.

8.7 Tuotteiden hallinta

Tuotteiden hallinnassa käsitellään ja hallitaan kaikkia järjestelmässä esiintyviä tuotteita. Tuotteet voivat olla fyysisiä tuotteita tai laskutettavaa työtä. Tuotteiden hallinnassa voidaan lisätä uusia tuotteita järjestelmään, muokata järjestelmässä olevien tuotteiden tietoja ja poistaa vanhoja tuotteita käytöstä. Tuotteita voidaan jaotella tuoteperheisiin, tuoteryhmiin ja kategorioihin. Tuoteperhe on Innotekin oma käsite, jonka avulla jaotellaan työnannoissa asennettavat tuotteet työn kohteen mukaan, esimerkiksi keittiöhanan huoltoon voi liittyä tietyt ennalta määritellyt tuotteet ja työ.

8.8 Varastojen hallinta

Varastojen hallinnassa voidaan luoda uusia varastoja, muokata varastotietoja, poistaa vanhoja varastoja, hallita varastoissa olevia tuotteita ja niiden saldoja. Varastoissa tuotteita käsitellään tuote-erinä. Jokaiseen tuote-erään liittyy kappalemäärä, tuote ja sisäänostohinta. Lisäksi tuotteita on käsiteltävä tapahtumapohjaisesti, jotta tuotemääriä ja inventaariota voidaan seurata.

9 YHTEENVETO JA POHDINTA

Tässä opinnäytetyönä aloitetussa Innotek-hankkeessa asiakkaalla on selkeä tavoite kehittää liiketoimintaprosessia ja palvelutuotetta sekä määritellä, suunnitella ja kehittää ohjelmisto, joka vastaa uuden liiketoimintamallin tarpeita. Ennen opinnäytetyön aloitusta ohjelmistoideaa oli pohdittu ja osittain toteutettu sekä testailtu. Toteutetut osiot ja alkupalavereissa saadut tiedot toimivat pohjana alustavassa tutkimustyössä ja vaatimusten kartoittamisessa. Projektin aikana kuitenkin monet toimintokokonaisuudet ja vaatimukset tarkentuivat, kun määrittelydokumentin laadinta ja vaatimusten tarkentaminen etenivät. Tulevan ohjelmiston hyödyt olivat mielestäni selkeästi ymmärrettävissä ja kuvailtavissa. Tavoitteena järjestelmällä on tehostaa asentajien toiminnan ohjausta ja vähentää yrityksen palveluiden aiheuttamaa käsin tehtävää paperityötä sekä automatisoida tiettyjä tarkoin mietittyjä liiketoiminnallisia prosesseja. Samalla järjestelmä tehostaa yritystoimintaa ja parantaa palveluiden laatua. Jatkossa kaikki palveluihin liittyvä tieto ja niistä generoitava dokumentaatio ovat keskitetyssä yrityksen yhteisessä järjestelmässä.

Kuten mille tahansa projektille, myös tälle projektille asetettiin alussa selkeät päätavoitteet, joita lähdettiin tavoittelemaan. Tavoitteena oli kerätä vaatimukset sekä tarkentaa ja dokumentoida ne mahdollisimman hyvin sekä toteuttaa ja käyttöönottaa määrittelyn mukainen järjestelmä. Kaikkia näitä tavoitteita ei opinnäytetyöhön varatun ajan puitteissa saavutettu, mutta projektia jatketaan ja samat tavoitteet ovat edelleen voimassa. Projekti jatkuu uudella kokoonpanolla, joka lähtee tavoittelemaan projektille asetettuja tavoitteita.

Aina kun ollaan suorittamassa ainutlaatuista tehtävää, siihen liittyy tietyn tyyppisiä riskejä, jotka tulee hallita, ja ongelmia, jotka tulee ratkaista, jotta projektilla on edellytyksiä edetä kohti tavoitteita. Seuraavaksi pohdin muutamia havaitsemiani ongelmia, riskejä ja ratkaisuja.

Alkuvaiheessa työtä vaikeutti se, että aiemmasta järjestelmästä ei ollut kunnollista dokumentaatiota. Tämä osoittautui ongelmaksi vanhan järjestelmän kartoituksessa ja sen pohjalta tehtyjen alustavien vaatimusten keräämisessä. Ongel-

ma pyrittiin ratkaisemaan projektiryhmän kesken tehdyllä päätöksellä tehdä tarvittava määrittelydokumentti. Tätä dokumenttia pyrittäisiin myös ylläpitämään koko projektin ajan. Samalla sitä käytettäisiin projektiryhmän ja asiakkaan välisenä sopimuksena sovituista asioista, joita kehitettävässä järjestelmässä otetaan huomioon. Dokumenttiin kerättiin vanhasta järjestelmästä kartoitetut asiat ja myös asiakaspalavereissa esiin tulleet vaatimukset. Asioita tarkennettiin kysymyslistojen avulla. Kaikki epäselvät asiat kirjattiin kysymyslistoihin, jotka käytiin läpi asiakkaan kanssa.

Aikaisemmasta ohjelmistoprojektissa tehdystä työstä toiminnanohjausjärjestelmän parissa oli monessa suhteessa hyötyä opinnäytetyöprojektin kannalta ja useimmat projektin hallintaan liittyvät asiat olivat tuttuja. Nämä helpottivat alussa tehtyjen töiden tekemistä vanhan järjestelmän tutkimisessa ja kartoittamisessa. Projektia haittasi alkuvaiheessa se, ettei asiakkaalla ei ollut riittävästi aikaa yritystoiminnan pyörittämisen ohella keskittyä projektin ohjaamiseen ja palautteen antamiseen. Mielestäni on ensiarvoisen tärkeää opinnäytetyötä tekevälle opiskelijalle, että asiakas on jatkuvasti mukana liiketoiminnan mallintamisessa, antaa palautetta sekä vastaa kysymyksiin. Jos kunnollista keskustelua ei synny dokumentoiduista vaatimuksista, altistutaan tekemään virheellisiä päätelmiä, joita saatetaan joutua korjaamaan myöhemmässä vaiheessa. Usein myöhemmin korjattavat asiat vievät selkeästi enemmän aikaa ja resursseja kuin alkuvaiheessa. Tässä projektissa kyseistä riskiä lisäsivät käytettävissä olevan ajan lisäksi suuret maantieteelliset etäisyydet, joihin pyrittiin varautumaan pitämällä puhelinpalavereja ja valmistelemaan erilaisia kysymyslistoja, joiden avulla tärkeitä vaatimuksiin ja ohjelmistoideaan liittyviä asioita saatiin selvitettyä. Osa kysymyslistojen kohdista käytiin läpi sähköpostin välityksellä tapahtuneella kirjeenvaihdolla. Suurin osa kysymyksistä selvitettiin asiakkaan aikataulun ehdoilla järjestetyissä asiakaspalavereissa. Mielestäni tähän oltaisiin voitu etsiä parempaa ratkaisua.

Eräs merkittävä ongelma projektissa oli se, ettei kukaan tekijöistä ollut aiemmin ohjelmoinut tai toteuttanut selainpohjaista ohjelmistoa HTML- ja PHP-kielillä. Tämän takia toteutusvaiheen alussa käytettiin paljon aikaa edellä mainittujen kielten opetteluun, hyväksi havaittujen tekniikoiden tutkimiseen ja järjestelmän

suunnitteluun. CodeIgniter-framework valittiin yhdeksi ratkaisuksi, koska se antaa ajansaatossa hyväksi havaitun perusrakenteen HTML- ja PHP-sovellusten rakentamiseen ja sisältää entuudestaan tutun MVC-arkkitehtuurin sekä useita valmiiksi toteutettuja komponentteja ja työkaluja. Lisäksi se oli hyvin dokumentoitu ja nopeasti opittavissa. Tästä on se hyöty, että seuraavat tekijät pystyvät nopeasti omaksumaan käytetyn toteutustavan.

Projektissa tekijöillä on oltava tietyt perusedellytykset, jotta annetut tehtävät ja työhön kuluva aika saadaan käytettyä mahdollisimman tehokkaasti projektin edistämiseksi. Nykypäivänä ohjelmistotekniikan projekteissa joudutaan käyttämään useita erilaisia työkaluja ja välineitä, joiden avulla projektia ja sen aikana syntyneitä tuotoksia voidaan hallita. Tämä tuottaa ongelmia projekteissa, joissa uusia tekijöitä liittyy mukaan ja vanhat tekijät lähtevät projektista työhön varatun ajan loppuessa. Jokainen uusi tekijä on koulutettava projektin käytäntöihin. Tämä sitoo vanhan ja uuden tekijän työpanosta opastukseen ja opettelutyöhön. Projektissa käytettävät työkalut ja niiden osaaminen on tärkeässä asemassa ja niitä on opeteltava käyttämään heti alkuvaiheessa. Alkusyksystä 2010 projektiin liittyi useita uusia tekijöitä, jotka jatkoivat edellisten tekijöiden työtä. Ongelmaksi muodostui aikaa vievä opettelu- ja opastusprosessi jokaisen uuden tekijän kohdalla. Tähän pyrittiin vastaamaan pitämällä projektiryhmän kesken tilaisuuksia, joissa työkalun käyttämiseen liittyviä asioita käytiin läpi ja joissa tekijöitä opastettiin. Tilaisuuksissa uusilla tekijöillä oli myös mahdollisuus esittää kysymyksiä ja saada niihin vastauksia pidempään projektissa olleilta tekijöiltä. Samalla tavalla projektiryhmän sisäisissä tilaisuuksissa katselmoitiin jo tehtyjä töitä ja järjestelmän määrittelyssä. Tällä tavalla saatiin tehokkaasti tuettua jokaisen uuden tekijän itse tekemää tutkimustyötä projektissa kehitettävästä järjestelmästä. Tilaisuuksissa käydyt asiat ja huomiot dokumentoitiin myöhempää tarkastelua varten, mikä osoittautui hyödylliseksi myöhemmässä vaiheessa.

Uuden tekijän liittäminen projektiin on mielestäni tehtävä tarkasti arvioimalla tekijän työpanoksen tuoman hyödyn mukaan. Isossa projektissa uuden tekijän kouluttaminen projektin käytäntöihin, käytettyihin apuvälineisiin ja projektissa syntyneisiin tuotoksiin vie paljon aikaa varsinkin pidemmälle edenneissä projekteissa. Tällöin on suoritettava ensin tarkka arviointi tekijän osaamistasosta, jon-

ka jälkeen on etsittävä projektista sellainen työpaketti, joka soveltuu parhaiten uudelle tekijän osaamistasoon. Työpaketti on rajattava tarkasti ja tarvittavat edellytykset tehtävässä onnistumiselle on arvioitava, jonka jälkeen voidaan päättää onko tekijän ottaminen projektiin kannattavaa sekä projektin että uuden tekijän näkökulmasta. Mielestäni Innonet-projektissa ei ole kummastakaan näkökulmasta kannattavaa liittää uutta tekijää projektiin, jos opiskeleminen vie enemmän kuin 50 % tekijän projektiin tuomasta työpanoksesta. Useissa ohjelmistotuotantoa käsittelevissä kirjoissa on pohdittu uusien tekijöiden ja lisätyövoiman tuomista keskeneräiseen ohjelmistoprojektiin. Edellä mainittua tapausta on kuvattu erääksi klassiseksi virheeksi, jossa uusien tekijöiden ottaminen projektiin voi viedä vanhoilta tekijöiltä enemmän työpanosta kuin mitä uudet tekijät voivat antaa. Tämä kuvaa mielestäni hyvin uuden tekijän mukaan ottamisen projektin näkökulmasta.

Opinnäytetyön aikana projektityöskentely sujui alkuperäisellä kokoonpanolla mielestäni mainiosti, koska olimme työskennelleet aiemmassa projektissa ja tiesimme toistemme toimintatavat. Myöhemmässä vaiheessa syksyllä uusien tekijöiden liittyessä projektiin huomasin, ettei projektin tekeminen ja syyslukukauden opintoihin keskittyminen ollut hyvä asia, koska tekijöillä ollut aikaa keskittyä opintojen ohella projektiin. Ongelmaksi muodostui myös tekijöiden opastaminen projektin käytäntöihin. Jos projektiin liittyy uutta työvoimaa ulkopuolelta, vie liittyviltä tekijöitä reilusti aikaa opiskella ja perehtyä kunnolla projektin käytäntöihin, järjestelmän vaatimuksiin ja työkaluihin. Pidimme kuitenkin viikoittaisia palavereja ja kävimme läpi, mitä kukin oli tehnyt projektiin liittyvistä asioista ja samalla kävimme myös läpi, jos jollakin oli kysyttävää tai epäselviä asioita. Keskittyminen useampaan tehtävään on ongelmallista ja syksyn aikainen projektiin liittyvä työ epäonnistui, koska minkäänlaista selkeää edistymistä ei syksyn aikana projektissa tapahtunut.

KUVAT

TAULUKOT

LÄHTEET

Ajax, Refsnes Data 2010. Introduction to Ajax.
http://www.w3schools.com/Ajax/Default.Asp
(Luettu 25.9.2010)

Altova, 2010. UML Timing Diagrams
http://www.altova.com/umodel/timing-diagrams.html
(Luettu 15.2.2011)

ECMAScript Language Specification, ECMA-262 2009
http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf
(Luettu 15.9.2010)

Ellislab, Inc. CodeIgniter
http://codeigniter.com/
(Luettu 18.10.2010)

Ellislab, Inc. CodeIgniter Application FlowChart
http://codeigniter.com/user_guide/overview/appflow.html
(Luettu 18.10.2010)

GNU GPL-license
http://www.gnu.org/copyleft/gpl.html
(Luettu 18.10.2010)

Haikala, I. 2009. Ohjelmistotuotannon perusteet.
Tampereen teknillinen yliopisto.
http://www.cs.tut.fi/~otupk/luennot/kalvot/alku.pdf
(Luettu 21.2.2011)

Hans-Erik Erikson & Magnus Penker, 2000. UML
Helsinki, IT Press.
(Luettu 21.10.2010)

Hernandez, M. 2000. Tietokannat: Suunnittelu ja toteutus käytännössä.
Helsinki. IT Press.
(Luettu 21.9.2010)

Holzner, S. Inside XML
Helsinki. IT Press.
(Luettu 18.9.2010)

Ilkko, L. 2008. Ohjelmistotekniikka.
Oulun ammattikorkeakoulu.
http://www.students.oamk.fi/~s6matu00/sekalaista/ot/teema2.ppt
(Luettu 21.2.2011)

Innotek Oy. 2000a. Energo-ohjelma esityskalvot.
http://www.innotek.fi/datapankki/esityskalvot1428kt.pdf
(Luettu 21.2.2011)

Innotek Oy. 2000b. Lehdistötiedote.
http://www.innotek.fi/datapankki/lehdistotiedote.pdf
(Luettu 21.2.2011)

Jacobson, I. Applying UML in The Unified Process
Rational Software
http://www.jeckle.de/files/uniproc.pdf
(Luettu 21.2.2011)

Johdatus PHP-kieleen
http://users.jyu.fi/~kolli/ITK215_05/php/
(Luettu 13.9.2010)

Kampman K. 2001. UML ohjelmistoprosessien tukena
Pro Gradu -tutkielma, Joensuun yliopisto
ftp://www.cs.joensuu.fi/pub/Theses/2001_MSc_Kampman_Kimmo.pdf
(Luettu 21.2.2011)

Ketterät käytännöt
http://www.ketteratkaytannot.fi/fi-FI/Ketteryys/Periaatteet/
(Luettu 21.2.2011)

Kettunen, J. & Simons, M. 2001. Toiminnanohjausjärjestelmän käyttöönotto pk-yrityksessä.
Valtion teknillinen tutkimuskeskus.
http://www.vtt.fi/inf/pdf/julkaisut/2001/J854.pdf
(Luettu 21.10.2010)

Kinnunen, S. 2010. Etätyöympäristön asennus ja konfigurointi.
Opinnäytetyö, Turun ammattikorkeakoulu
https://publications.theseus.fi/bitstream/handle/10024/7938/Kinnunen_Sami.pdf?sequence=1
(Luettu 18.10.2010)

Korpela J. 2009. JavaScript
http://www.cs.tut.fi/~jkorpela/webjulk/3.2.html#js-mita
(Luettu 15.9.2010)

Kylä-Nikkilä, J. 2008. UML-kaaviot.
Pro Gradu-tutkielma, Tampereen yliopisto.
http://www.cs.uta.fi/research/thesis/masters/Kyla-Nikkila_Jouni.pdf
(Luettu 18.10.2010)

Lipitsäinen, A. & Marttila, S. 2003. Apache-palvelin
http://myy.helia.fi/~atk90d/palvelin/apache.html
(Luettu 20.9.2010)

McConnell, S. 2002. Ohjelmistotuotannon hallinta
Helsinki, IT Press.
(Luettu 26.01.2011)

MVC Overview
Microsoft ASP.NET MVC Overview
http://www.asp.net/mvc/tutorials/asp-net-mvc-overview-cs
(Luettu 7.3.2011)

MVC Wikipedia
Wikipedia, Model-View-Controller
http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
(Luettu 7.3.2011)

Mäkipää, M. 2002
Pro Gradu -tutkielma, Tampereen yliopisto
http://www.cs.uta.fi/research/theses/masters/Makipaa_Marko.pdf
(Luettu 21.10.2010)

Notepad++ project
http://notepad-plus-plus.org/
(Luettu 18.10.2010)

Object Management Group, About OMG
http://www.omg.org/gettingstarted/gettingstartedindex.htm
(Luettu 18.10.2010)

OpenSource Ratol. MySQL-opetusmateriaali
http://www.ratol.fi/opensource/mysql/index.htm
(Luettu 25.9.2010)

Oracle. 2010. MySQL 5.5 Reference Manual.
http://dev.mysql.com/doc/refman/5.5/en/index.html
(Luettu 25.9.2010)

Patosuo, M. 2009. MySQL-tietokantavarastointimoottori pienyrityksen tietovarastointiin.
Opinnäytetyö Lahden ammattikorkeakoulu.
http://urn.fi/URN:NBN:fi:amk-200905072546
(Luettu 25.9.2010)

Perustietoa HTML:stä
http://cc.joensuu.fi/~marttila/html/yleista.html#yleista
(Luettu 13.9.2010)

Rational Software, 2001.
Rational Unified Process Best Practices for Software Development Teams
http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf
(Luettu 21.2.2011)

Saarikumpu O. 2010. JavaScript kielen alkeet
http://weppipakki.com/js/opas/alkeet1.htm#mojs
(Luettu 15.9.2010)

Sparx Systems, 2010. Interaction Overview Diagram
http://www.sparxsystems.com/enterprise_architect_user_guide/8.0/modeling_languages/interactionoverviewdiagram.html
(Luettu 15.2.2011)

Stephens, Plew, Morgan & Perkins 2001. SQL Tietokantaohjelmointi
Helsinki. IT Press.
(Luettu 24.9.2010)

The Menlo Institute LLC, 2002. The Rational Unified Process (RUP)
http://www.menloinnovations.com/freestuff/whitepapers/Rational%20Unified%20Process.pdf
(Luettu 21.2.2011)

The PHP Group. 2010. What is PHP?
http://fi.php.net/manual/en/intro-whatis.php
(Luettu 13.9.2010)

TTY, Ohjelmistotekniikanlaitos.
http://www.cs.tut.fi/ohj/dokumenttipohjat/
(Luettu 22.2.2011)

W3C
http://www.w3c.tut.fi/reports/2003/0113aboutw3c/index.html
(Luettu 15.9.2010)

XML, Refsnes Data. 2010. Introduction to XML.
http://www.w3schools.com/xml/xml_whatis.asp
(Luettu 25.9.2010)