Opinnäytetyö (AMK)

Elektroniikka

Elektroniikkatuotanto

2014

Teemu Saari ja Harriet Lenets

HUOLTOTIETOKANNAN SUUNNITTELU MS ACCESS-OHJELMISTOLLA

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Elektroniikan koulutusohjelma | Elektroniikkatuotanto

2014 | 47 sivua

Ohjaaja: Yngvar Wikström

Teemu Saari ja Harriet Lenets

HUOLTOTIETOKANNAN SUUNNITTELU MS ACCESS-OHJELMISTOLLA

Tässä työssä suunniteltiin ja toteutettiin sähköinen huoltotietokanta Turun Huoltoexperteille. Tietokannan tehtävänä oli tallentaa asiakas- ja laitetiedot sähköisesti huoltotietokantaan. Sähköisen järjestelmän oli tarkoitus korvata aikaisemmin käytössä ollut järjestelmä.

Teoriaosuudessa käsiteltiin ohjelmistokehitysmalleja ja ohjelmiston testaamista. Ohjelmistokehitysmalleihin perehtyessä ja niitä tutkittaessa päädyttiin käyttämään vesiputousmallia, koska se sopi tämän projektin tavoitteisiin parhaiten. Ohjelmiston testaus osuudessa tutustuttiin tarkemmin testauksen v-malliin. Teoriaosuudessa käsiteltiin myös suppeasti relaatiotietokantojen toimintaperiaatteita.

Työ aloitettiin haastattelemalla toimeksiantajaa ja tutustumalla asiakasyrityksen huoltoprosessiin. Tietokanta suunniteltiin haastattelujen ja omien ideoiden pohjalta ja se toteutettiin MS Access -tietokantasovelluksella. Huoltotietokanta avulla voidaan tallentaa asiakkaita ja laitteita sähköiseen muotoon. Sovelluksen avulla pystytään etsimään aiemmin huollossa olleiden laitteiden tietoja ja seuraamaan töiden etenemistä.

Lopputuloksena syntyi toimiva ja asiakkaan vaatimukset täyttävä huoltotietokanta. Järjestelmä otettiin käyttöön yrityksessä aluksi vanhan järjestelmän rinnalle, mutta myöhemmin se tulee korvaamaan täysin vanhan järjestelmän.

ASIASANAT:

MS Access, relaatiotietokanta, ohjelmistosuunnittelu, ohjelmistotestaus

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Electronics| Production

2014 | 47 pages

Instructor: Yngvar Wikström

Teemu Saari and Harriet Lenets

DESIGNING SERVICE DATABASE WITH MS ACCESS

In this Bachelor's Thesis an electronic service database was designed and programmed for a company called Turun Huoltoexpertit. The main purpose of the database was to electronically store customer and device records. The electronic database was needed, because in the old system, the customer and device records were written on the paper and that was no longer practical.

Turun Huoltoexpertit is a company that services electronic devices like TVs, computers and audio and HiFi systems. Turun Huoltoexpertit is a small company located in Turku, Finland.

This thesis focuses on describing different phases and progress of software engineering. It mainly focuses on different software engineering models and how to implement those methods in our project. Information and material for this project were collected by carrying out interviews and visits to our commissioner. The software was created by using MS Access. MS Access was selected because the customer was already using MS Office software.

The software produced was functional and accepted by the commissioner who was very satisfied with the results of this project. Our project was completed with the database transfer at the customer enterprise.

KEYWORDS:

MS Access, Relational database, Software engineering, Software testing

SISÄLTÖ

LYHENTEET

ER-kaavio	Ohjelmistosuunnitteluun käytettävä työkalu
Iteraatio	Iteroinnissa toistettavaa työvaihetta kutsutaan iteraatioksi
Iterointi	Nimitys menetelmille, joissa samoja työvaiheita toistetaan kunnes haluttu tulos on saavutettu
MS	Microsoft
MS Access	Tietokantahallintasovellus
Runtime	Tietokoneohjelman suorittamiseen tarkoitettu ohjelma
SQL	Tietotekniikassa käytetty kyselykieli (Structured Query Language)
VBA	Ohjelmointikieli (Visual Basic for Applications)

1 JOHDANTO

Tässä opinnäytetyössä suunniteltiin ja toteutettiin sähköinen huoltotietokanta elektroniikkahuoltoyhtiö Turun Huoltoexperteille. Asiakkaan tarve kyseiselle ohjelmistolle oli suuri, sillä asiakasyrityksellä ei aiemmin ollut ollenkaan käytössä sähköistä järjestelmää, jolla voidaan hallinnoida asiakastietoja sekä huollossa olevia laitteita.

Tässä työssä perehdyttiin ohjelmistokehitykseen ja -tuotantoon. Ohjelmistokehitysmalleja on monia, mutta työ suunniteltiin vesiputousmallin mukaisesti, koska se vastasi hyvin projektin tarpeita. Vesiputousmallissa (waterfall model) ohjelmiston kehitys etenee vaiheissa. Jokainen vaihe tehdään kerralla valmiiksi niin, että aikaisempiin vaiheisiin ei palata enää myöhemmin.

Tietokannan vaatimukset selvitettiin asiakkaan toimeksiannon, haastatteluiden sekä omien ideoiden pohjalta. Tavoitteeksi muodostui luoda helppokäyttöinen ja selkeä, mutta kuitenkin kattava järjestelmä. Tietokannan piti myös olla helposti jaettavissa kotiverkon kautta monelle työpisteelle. Ohjelmiston suunnittelu ja määrittely tehtiin huolellisesti, jotta työstä tuli kerralla toimiva ja tarkoituksen mukainen. Asiakasyrityksen edustaja oli tiiviisti mukana määrittely ja suunnittelu vaiheissa, jotta saatiin selkeät määritelmät siitä, mitä asiakas haluaa ohjelmistolta.

Tietokanta toteutettiin MS Access 2010 -ohjelmalla. Kyseinen ohjelmisto valittiin asiakkaan toivomuksesta. Asiakkaalla oli käytössä jo aikaisemmin MS Office -ohjelmisto ja näin ollen ohjelmisto oli yritykselle entuudestaan tuttu.

Aikaisemmin aiheesta on tehty useita opinnäytetöitä, mutta ei varsinaista elektroniikkahuoltoon tarkoitettua tietokantaa. Aikaisemmista opinnäytetöistä tutustuttiin seuraaviin Juntunen, Jari; Kotakorva, Marko: Asiakas- ja huoltotietokanta, ja Ojakorpi, Pia: Tietokantasovelluksen toteutus Access 2007 -ohjelmalla. Opinnäytetöissä oli käsitelty tietokannan tekemistä ja Accesin käyttöä.

Projektin toteutus tehtiin parityönä, jossa kumpikin osallistui tietokannan suunnitteluun ja tekoon yhtä paljon. Ohjelmisto tuotettiin siten, että suunnittelu tehtiin yhdessä. Varsinainen tietokannan luominen jakautui tasaisesti, sillä kaikki päätökset toteutus tavoista ja ongelmien ratkaisuista tehtiin yhdessä. Projekti toteutettiin siis tiivistä yhteistyötä tehden.

2 OHJELMISTOTUOTANTO

Ohjelmistotuotanto tarkoittaa tietokoneohjelmistojen valmistuksessa yleisesti käytettyjä tekniikoita, työkaluja, menettelytapoja ja periaatteita. Ohjelmistotuotanto on ollut olemassa käsitteenä jo tietokoneiden yleistymisestä asti, mutta sen alkuna voidaan pitää NATO 1968 – konferenssia, joka oli ensimmäinen ohjelmistokehitys-konferenssi. [1]

Ohjelmistotuotannon kattavuudesta ja olemassaolosta omana tieteenalanaan on epäilyksiä. Epäilyksiä herättää se, että ohjelmistotuotanto koostuu testatuista ja hyväksi havaituista toimintatavoista ja periaatteesta eikä matemaattisista menetelmistä, kuten yleisesti insinööritieteissä. Käytännössä ohjelmistotuotannon merkitykseksi voidaan ajatella, että tavoitteena on tuottaa asiakkaan odotuksia vastaava tietokoneohjelma siten, että kustannukset ja aikataulu pystytään määrittelemään vaadittavalla tarkkuudella. [1]

Ohjelmistotuotantoprosessiin kuuluvat kaikki ohjelmistotyön vaiheet, joita ovat määrittely, suunnittelu, valmistus, testaus ja ylläpito. Ohjelmistotuotanto sisältää myös itse tuotantoprosessin, johon kuuluvat laadun tarkkailu, projektinhallinta ja raportointi. Ohjelmistotuotanto sisältää siis kaikki osa-alueet ensimmäisestä ideasta valmiiseen ohjelmistoon. [1]

2.1 Suunnittelumalli

Suunnittelumallia kutsutaan myös ns. elinkaarimalliksi. Ohjelmistokehitys pohjautuu ajatukseen, jossa projekti etenee ns. elinkaaren läpi. Elinkaari alkaa ideasta tai määrittelystä ja päättyy valmiiseen ohjelmistoon. Elinkaari on järjestelmällinen malli siitä, mitä pitäisi tapahtua idean ja viimeisen version välillä. Ohjelmistojen valmistusta kyetään käsittelemään järjestelmällisesti, kun valmistusprosessia mallinnetaan elinkaarimallin mukaisesti. Elinkaarimallin avulla prosessin aikataulu pyritään näkemään mahdollisimman laajana kokonaisuutena, jossa tekninen ohjelmointi on vain pieni, mutta erittäin tärkeä osa. [2]

Elinkaarimalli sisältää monia osa-alueita, mutta ne kaikki sisältävät ainakin määrittelyn, toteutuksen, hyväksyttämisen ja ohjelmiston kehittämisen. Määrittelyssä asiakas ja suunnittelija päättävät, mitä ohjelmiston tulisi sisältää. Toteutuksessa ohjelmisto suun-

nitellaan ja ohjelmoidaan. Hyväksyttämisessä tarkistetaan, että ohjelmisto vastaa asiakkaan vaatimuksia. Ohjelmiston kehittämisessä ohjelmiston pitää kehittyä asiakkaan muuttuviin tarpeisiin tai markkinamuutoksiin. [3]

2.2 Suunnittelumallin yleisimmät vaiheet

Jokaisen elinkaarimallin aluksi kartoitetaan ohjelman tarve esitutkimuksen avulla. Esitutkimuksen tehtävänä on asettaa järjestelmätason vaatimukset, jotka ovat usein asiakasvaatimuksia. Asiakasvaatimukset määrittelevät ne tarpeet, jotka asiakas järjestelmältä vaatii. Esitutkimuksen ideana onkin määrittää miksi tai miksi ei järjestelmää tai ohjelmistoa tulisi tehdä. Joissakin malleissa esitutkimus on tärkein vaihe, koska siinä määritellään ohjelmiston vaatimukset eikä vaiheeseen enää myöhemmin palata. Tärkeintä on siis selvittää asiakkaan todelliset tarpeet ja niiden kunnollinen ymmärtäminen. [4]

Esitutkimusta seuraa usein määrittelyvaihe, joka voidaan myös mieltää esitutkimuksen kanssa yhdeksi vaiheeksi, koska asiakasvaatimusten tarkentaminen ja analysointi jatkuvat myös määrittelyvaiheessa. Määrittelyvaiheessa asiakkaan antamia vaatimuksia analysoidaan ja niiden avulla selvitetään vaatimukset ja ominaisuudet ohjelmistolle. Määrittelyvaiheen ideana on siis asiakasvaatimusten konkretisointi ohjelmistovaatimuksiksi. [4]

Määrittelyvaiheen jälkeen päästään itse suunnitteluun. Suunnitteluvaiheessa määritetään järjestelmän tekniset vaatimukset. Teknisten vaatimusten tehtävänä on selventää, miten ohjelmistovaatimukset sisältävä järjestelmä toteutetaan. Suunnitteluvaihe voidaan jakaa vähintään kahteen tasoon, jotka ovat moduulisuunnittelu ja arkkitehtuurisuunnittelu. [4]

Aluksi on hyvä jakaa ohjelma mahdollisimman moneen erilliseen moduuliin. Moduuleihin jakoa kutsutaan yleisimmin arkkitehtuurisuunnitteluksi. Tämän vaiheen dokumentointia kutsutaan tekniseksi määrittelyksi. Arkkitehtuurisuunnittelun jälkeen tulee moduulisuunnitteluvaihe, jossa moduuleitten sisäinen rakenne suunnitellaan. Suunnitteluvaiheen ideana on siis vastata kysymykseen, miten ohjelma tehtävänsä suorittaa. [4]

Ohjelmointivaihe sisältää ohjelman varsinaisen kirjoitusosuuden. Tässä vaiheessa edetään siis suunnittelusta toteutukseen. Ohjelmointivaihe ei välttämättä ole projektin aikaa vievin osuus, jos pohjatyö on tehty järjestelmällisesti ja dokumentoitu huolellisesti. [4]

Ohjelman valmistuttua ohjelma tulee testata. Testausvaiheen tarkoituksena onkin etsiä ohjelmiston virheitä. Yleisesti käytetty testausmalli on v-malli. V-malli on monitasoinen testausjärjestelmä, jossa testaus etenee samoissa vaiheissa suunnittelumallin kanssa. Testaus suunnitellaan niin pitkälle kuin mahdollista jo määrittelyvaiheessa. Testausvaihe on erittäin tärkeä osa ohjelmistotuotantoa, erityisesti tuotekehitysprojekteissa. Testaus ja virheiden jäljittäminen voi viedä merkittävän osan järjestelmän kustannuksista. Testausta käsitellään tarkemmin luvussa 4. [4]

Ohjelmistotuotanto ei pääty vielä siihen, kun ohjelmisto on toimitettu asiakkaalle, vaan ohjelmistoa on ylläpidettävä mahdollisen sopimuksen mukaisesti. Ylläpidon tärkeimpiä tehtäviä ovat virheiden korjaus, ongelmien selvitys, ohjelman päivitys uusiutuviin tarpeisiin sekä lisäominaisuuksien lisääminen. [4]

Ylläpito voidaan suurpiirteisesti jaotella kolmeen osaan. Nämä osat ovat korjaava, adaptiivinen ja täydentävä ylläpito. Korjaavassa ylläpidossa korjataan ohjelman mahdollisia virheitä, adaptiivisessa kehitetään ohjelmaa, koska asiakkaan tai markkinoiden vaatimukset ovat muuttuneet ja täydentävässä ylläpidossa parannellaan ohjelman ominaisuuksia tai lisätään uusia ominaisuuksia. [4]

Elinkaarimallit voidaan jakaa suunnittelupohjaisiin ja ketteräpohjaisiin. Suunnittelupohjainen malli on sellainen, jossa kaikki vaiheet suunnitellaan etukäteen ja edistymistä verrataan näihin vaiheisiin. Ketterissä malleissa suunnittelu on inkrementaalista ja ohjelma on helpompi muokata asiakkaan muuttuvien tarpeiden mukaiseksi. Nämä elinkaarimallit eivät ole toisiaan poissulkevia, vaan niitä voidaan käyttää myös yhdessä. [3]

Yksi elinkaarimalli ei sovi kaikkiin projekteihin, joten jokaista projektia varten on valittava sopiva malli. Ohjelmille, joilta vaaditaan korkeaa luotettavuutta, olisi parasta valita hyvin perusteellinen suunnittelumalli, vaikka tällöin tuotantoaika pitenee. Henkilökohtaiseen käyttöön ja pienyrityksille on parempi valita joustavampi malli, jolloin voidaan nopeasti puuttua muuttuviin tarpeisiin. Isommille projekteille voi olla järkevä yhdistää esimerkiksi vesiputousmallin ja inkrementaalisen mallin parhaita puolia. [3]

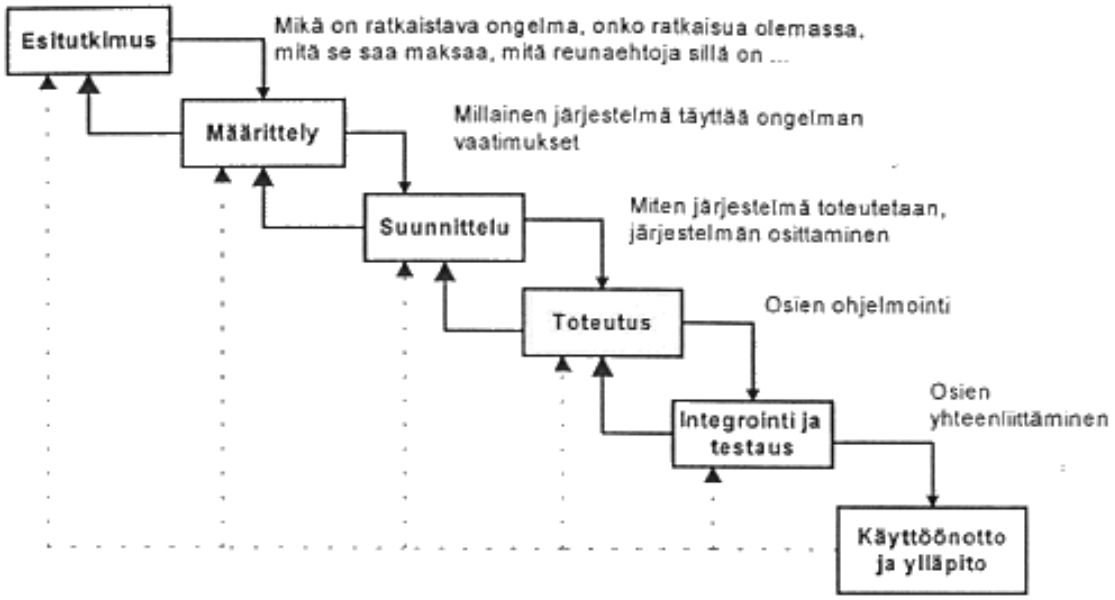
3 YLEISIMPIÄ SUUNNITTELUMALLEJA

Tässä luvussa esitellään yleisimpiä suunnittelumalleja, joihin perehdyttiin suunnittelumallia valittaessa.

3.1 Vesiputousmalli

Ensimmäisiä tietokoneohjelmia laadittaessa työprosessit noudattivat järjestystä: määrittele tarve, suunnittele ohjelma, testaa ja ota ohjelma käyttöön. Tähän systeemiin pohjautui ensimmäinen prosessimalli, Beningtonin artikkelissa 1956. 1970 Royce julkaisi artikkelin ”Managing the development of large software systems”, jonka keskeisenä osana oli malli, joka myöhemmin tuli tunnetuksi vesiputousmallina. [1]

Vesiputousmallista on erilaisia versioita, mutta kaikissa niissä on kuitenkin määrittely, suunnittelu ja toteutus. Noudattamalla vesiputousmallia edetään vaiheesta toiseen. Eli kun ohjelman määrittelyt on saatu valmiiksi, edetään seuraavaan vaiheeseen joka on suunnitteluvaihe (Kuva 1). Myöhemmissä vaiheissa ei määrittelyitä enää muokata. Kaikissa vaiheissa tarkastetaan laatua, esimerkiksi testaamalla. Virheet ja ongelmat pyritään poistamaan mahdollisimman aikaisin, koska mitä myöhemmin virhe havaitaan, sitä kalliimpi se on korjata. Jokaisen vaiheen jälkeen kartoitetaan tilanne, onko tavoitteisiin päästy ja ollaanko täysin valmiita siirtymään seuraavaan vaiheeseen. [4], [5]



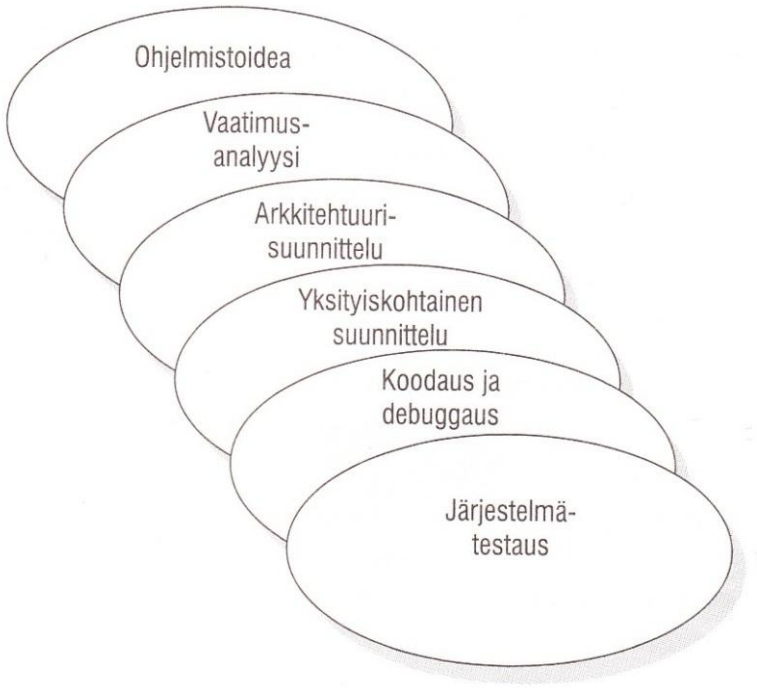
Kuva 1 Vesiputousmalli [4]

Vesiputousmallissa on monia ongelmia, mutta silti monet elinkaarimallit pohjautuvat siihen. Vaiheet eivät ole päällekkäisiä, vaan yksi vaihe toteutetaan loppuun asti ennen seuraavan vaiheen aloittamista. Vesiputousmalli toimii, jos on hyvä tuotemääritelmä ja tekniset menetelmät. Hyvien teknisten määrittelyiden jälkeen virheet löytyvät ajoissa, koska myöhäisemmässä vaiheessa voidaan tarkastella täyttyvätkö määrittelyt. Vesiputousmalli on hyvin säännönmukainen ja sen selkeän rakenteen vuoksi pystytään vähentämään turhaa työtä. Vesiputousmalli vaatii paljon dokumentaatioita, joiden tulisi olla sellaista, että seuraavan vaiheen voisi toteuttaa täysin uusi henkilö. [2]

Huonona puolena voidaan pitää sitä, että alussa ei ehkä pystytä määrittelemään ohjelman vaatimuksia kokonaan. Vesiputousmallin idea suorittaa vaiheet loppuun aiheuttaa ongelmia silloin, kun pitäisi palata vaiheissa taaksepäin. Vesiputousmalli voi olla aikaa vievä, koska se vaatii paljon dokumentaatiota, eikä siis välttämättä sovi nopeaan kehittämiseen. [2]

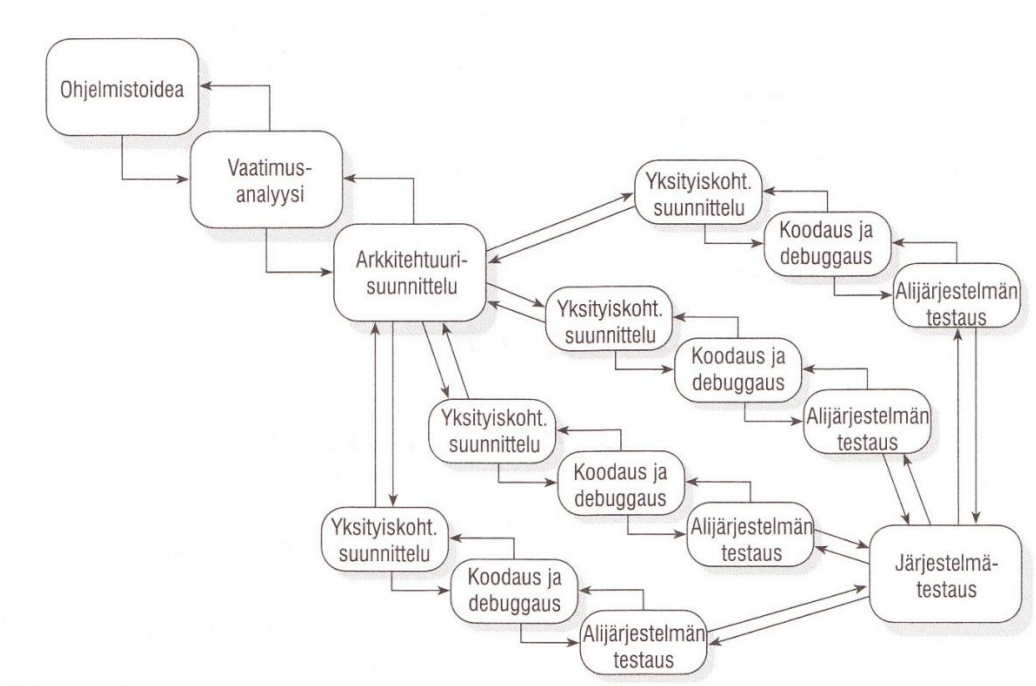
Yhtenä ongelmana voidaan myös pitää sitä, että valmista ohjelmaa ei ole ennen ohjelman käyttöönottovaihetta, ja näin asiakas näkee ensimmäisen kerran tuotoksen vasta kun se on valmis. Tämä vaikeuttaa palautteen saamista ohjelmistosta, mikä vaikeuttaa sitä, että ohjelmistoa ei pystytä muokkaamaan ennen lopullista versiota. [2]

Vesiputousmallin ongelmien vuoksi siitä on tehty muokattuja versioita yhdistelemällä sitä muihin suunnittelumalleihin. Yksi näistä malleista on sashimi eli vesiputousmalli limittäisin vaihein. Sashimimalli perustuu siihen, että vesiputousmallin vaiheet ovat osittain päällekkäin (Kuva 2). Mallissa voidaan siis jo olla pitkällä suunnitteluvaiheessa, ennen kuin vaatimusanalyysi on saatu täysin valmiiksi. Sashimimallissa etuihin lukeutuu myös se, että dokumentaatioita ei tarvita läheskään yhtä paljon kuin puhtaassa vesiputousmallissa. Huonona puolena sashimimallissa on se, että edistymisen seuraaminen ja aikataulun suunnittelu on hankalampaa. [2]



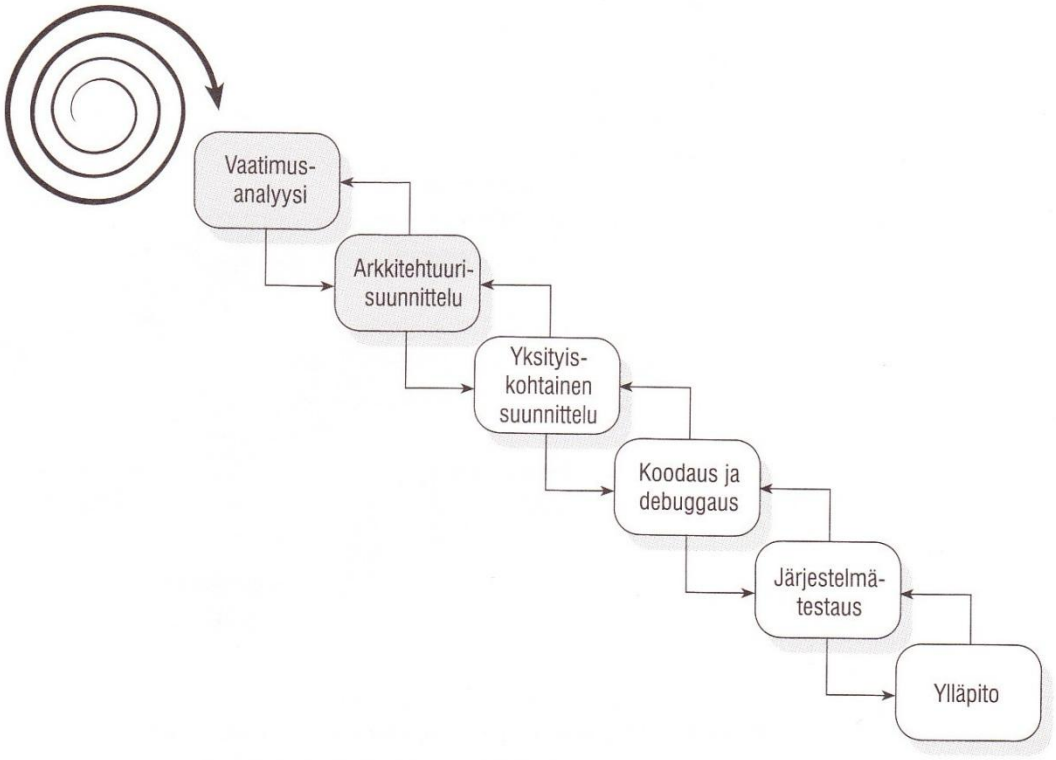
Kuva 2 Sashimimalli [2]

Toinen muunnelma vesiputousmallista on vesiputous aliprojekteilla. Muunnelma pyrkii korjaamaan sen ongelman, että yksinkertaisia osuuksia ei tarvitse suunnitella moneen kertaan. Mikäli on tehty hyvä arkkitehtuurisuunnittelu, voidaan toteutus jakaa moduuleihin, jossa helpommat osat voidaan toteuttaa erillään vaikeammista (Kuva 3). Riskinä tässä on moduulien yhdistämisestä syntyvät ongelmat, mutta tämän voi ratkaista viivyttämällä moduulien jakoa, kunnes on yksityiskohtainen suunnitelma toteutuksesta. [2]



Kuva 3 Vesiputousmalli aliprojekteilla [2]

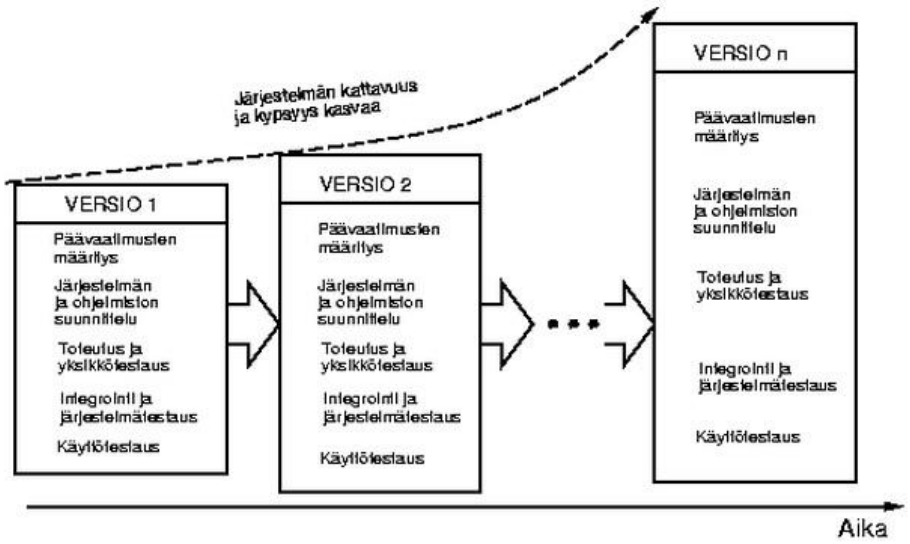
Kolmas muunnelma on vesiputous riskienvähennyksellä. Perinteisen vesiputousmallin hankaluus kulkea vaiheita taaksepäin saattaa aiheuttaa ongelmia esitutkimus- ja määrittelyvaiheissa. Riskejä vähentävä vesiputousmalli pyrkii korjaamaan tämän ongelman lisäämällä spiraalimallin ennen varsinaisen vesiputouksen alkamista (Kuva 4). Tämä malli auttaa vähentämään arkkitehtuuririskiä ja oikeastaan mitä tahansa riskiä projektissa. Tämä muunnelma sopii hyvin sellaisiin projekteihin, joissa tuote vaatii riskialttiin ytimen. [2]



Kuva 4 Vesiputous riskienvähennyksellä [2]

3.2 Evoluutiomallit

Evoluutiomallit toimivat kaikki saman kaavan mukaisesti. Evoluutiomalleissa vaatimusten määrittely, suunnittelu ja toteutus tapahtuvat joko samanaikaisesti tai osittain rinnakkain. Projektin edetessä ohjelmisto kehittyy joko prototyypistä toiseen tai alkuperäistä prototyyppiä muokkaamalla ja parantelemalla (Kuva 5). [6]



Ohjelmistoprosessin evolutionäärinen kehitys. Suunnittelu ja toteutus tapahtuvat samanaikaisesti. Ajan myötä järjestelmää kehitetään tekemällä uusia versioita.

Kuva 5 Evoluutiomallien toimintaperiaate [6]

Evoluutiomallien hyvänä puolena pidetään sitä, että ne mahdollistavat muuttuvat asiakasvaatimukset ja määritelmät. Evoluutiomallit toimivatkin siis parhaiten silloin, kun projektin alussa ei haluta, tai osata antaa selkeää tavoitetta. Yleisimpiä evoluutiomalleja ovat spiraalimalli, inkrementaalinen malli ja prototyyppimalli. [7]

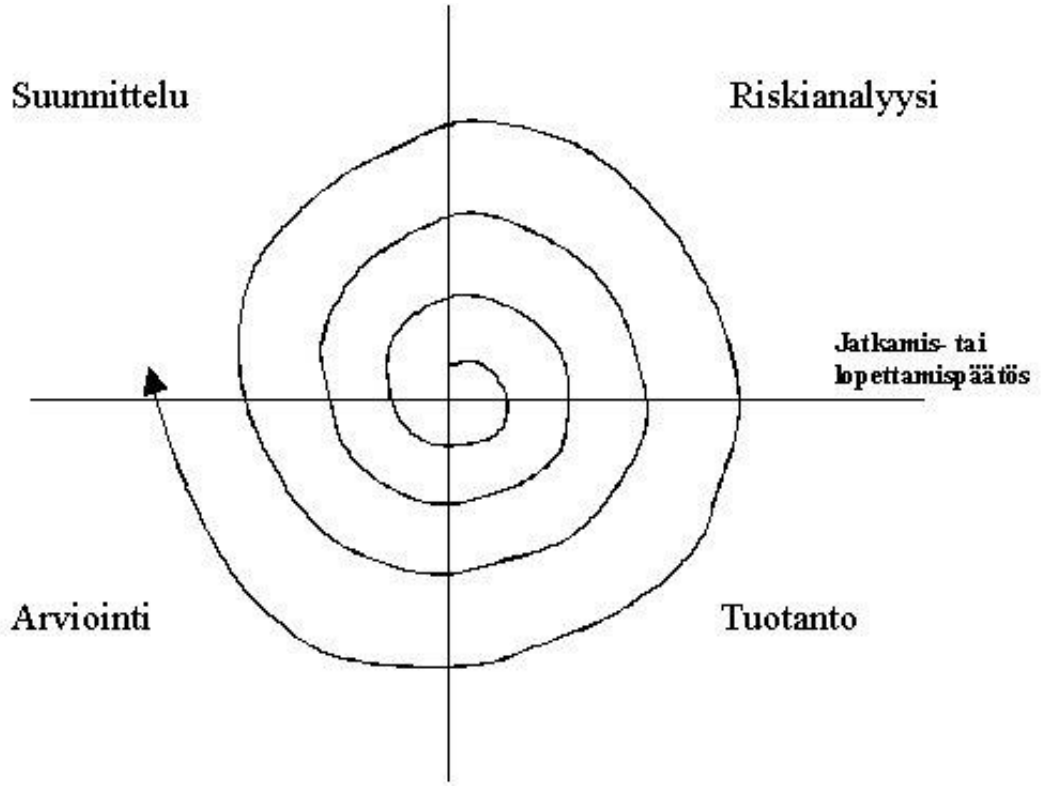
3.2.1 Spiraalimalli

Spiraalimallin on kehittänyt Barry Boehm vuonna 1988. Spiraalimalli pilkkoo projektin pienemmiksi projekteiksi. Jokainen pienempi projekti keskittyy yhteen tai useampaan riskiin. Riskeillä tässä tarkoitetaan huonosti ymmärrettyjä vaatimuksia, huonoa arkkitehtuuria, mahdollisia suorituskyky ongelmia ja perusteknologiaan liittyviä ongelmia. [2], [3]

Spiraalimallia voidaan yhdistää muihin suunnittelumalleihin, mutta yleensä projektin alkuun tulee seuraavat neljä vaihetta: tavoitteet, riskien analysointi, tuotanto ja suunnittelu (Kuva 6). Nämä neljä vaihetta toistuvat spiraalimallin edetessä. [2], [3]

Tavoitevaiheessa projektin tarkat tavoitteet määritellään, tuote identifioidaan ja yksityiskohtainen toimintasuunnitelma luodaan. Projektin riskit määritellään ja niiden pohjalta voidaan myös suunnitella vaihtoehtoisen strategiat riippuen riskeistä. Riskienanalysointivaiheessa jokaiselle riskille määritellään tarkka analyysi. Vaiheita muokataan riskien pienentämiseksi, esim. jos vaatimukset eivät ole selkeitä, voidaan loppu toteuttaa prototyyppimallilla. Tuotantovaiheessa valitaan sopiva tuotantomalli ja toteutetaan projekti loppuun. Projektin valmistuttua siirrytään suunnitteluvaiheeseen jossa tarkastellaan tuloksia ja tehdään päätös, jatketaanko spiraalissa eteenpäin. Suunnitteluvaiheessa suunnitellaan myös projektin seuraava vaihe, mikäli on päätetty jatkaa eteenpäin spiraalia. [2], [3]

Spiraalimallia ei kannata tulkita kirjaimellisesti, neljää kierrosta ei välttämättä tarvita, eikä kaikkia vaiheita välttämättä tarvitse toteuttaa järjestyksessä. Paras tapa käyttää spiraalimallia on soveltaa ja sovittaa sitä omaan projektiin sopivaksi. [2], [3]



Kuva 6 Spiraalimalli [8]

Hyvää spiraalimallissa on se, että isommissa projekteissa kustannusten noustessa myös riskit pienenevät. Mallista on myös se hyöty, että jos projektissa on sellainen riski, mitä on mahdotonta ylittää, se huomataan ajoissa. [2], [3]

Huonoina puolina voidaan pitää sitä, että se on kallis ja aikaa vievä. Spiraalimalli vaatii täsmällistä ammattitaitoa riskien tunnistamiseen ja prosessi on monimutkainen. Vaiheita läpikäydessä voi olla vaikea seurata koska ollaan valmiita seuraavaan kierrokseen. Projektien onnistuminen riippuu paljolti riskien tunnistamisvaiheesta, eikä se monimutkaisuutensa takia sovellu pienempiin projekteihin kovinkaan hyvin. Vaikkakin pienillä riskeillä spiraalimalli on helpompi toteuttaa. [2], [3]

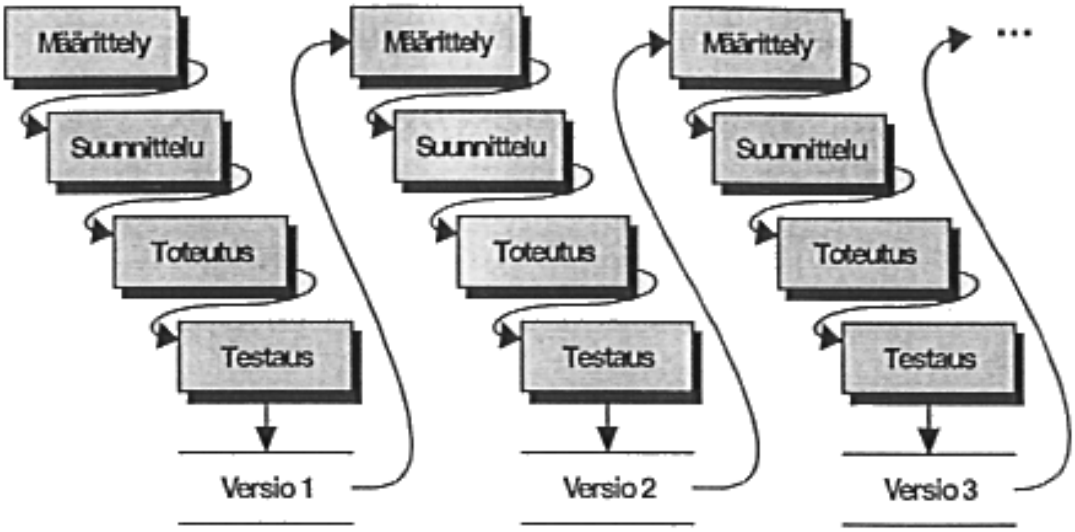
Spiraalimalli on hyvä sellaisissa projekteissa, joissa kustannukset ja riskien arviointi on tärkeässä osassa. Mallia on hyvä käyttää myös jos kyseessä on uusi tuote, jolloin merkittävät muutokset ovat odotettavissa. Spiraalimallia on hyvä käyttää myös silloin, jos asiakas on epävarma tarpeistaan tai vaatimukset ovat monimutkaisia. [9]

3.2.2 Inkrementaalinen malli

Inkerementaalinen malli on nykyään yksi yleisimpiä suunnittelumalleja. Inkrementaalinen kehitys muistuttaa ihmisen ongelmanratkaisutapoja. Harvoin ongelman edessä pystytään suunnittelemaan kaikki valmiiksi siten, että sen jälkeen edetään askeleittain kohti lopputulosta, vaan välillä pitää palata takaisin ja suunnitella uudelleen. Inkrementaalisen mallin pohjana ovat samantapaiset vaiheet, kuin vesiputousmallissakin eli määrittely, suunnittelu, toteutus ja testaus. [3]

Inkerementiaalinen kehitys pohjautuu ideaan kehittää ensin raakaversio, joka sisältää vain tärkeimmät toiminnalliset osuudet ja jättää muut osat kokonaan huomioitta. Inkrementaalisessa mallissa toteutus pyritään tuottamaan useissa peräkkäisissä prosesseissa. Aikaa yhden prosessin suorittamiseen kuluu yleensä noin yhdestä viikosta, muutamaan kuukauteen, riippuen projektin laajuudesta. [3], [10]

Prosessit toistetaan lisäten ominaisuuksia, kunnes ohjelma vastaa määrittelyitä jotka asiakas vaatii. Ohjelmaa ei kuitenkaan kehitetä joka versiossa alusta asti uudelleen, vaan aikaisempaa versiota käytetään pohjana, johon lisätään ominaisuuksia (Kuva 7). [3], [10]



Kuva 7 Inkrementaalinen malli [4]

Inkrementaalisesta rakenteesta on hyötyä ainakin siinä, että jonkinlainen versio tuotteesta saadaan asiakkaalle käyttöön jo hyvin aikaisessa vaiheessa projektia. Asiakas pääsee hyötymään sovelluksesta jo, vaikka se ei vielä täysin valmis olisikaan. Mitä aikaisemmin asiakas pääsee käyttämään ohjelmistoa, sitä helpommin saadaan palautetta ohjelmasta, mikä on etu ainakin verrattuna vesiputousmalliin. Inkrementaalinen malli onkin nopea reagoimaan asiakkaan muuttuviin tarpeisiin. Nopeuteen tehdä muutoksia vaikuttaa se, että inkrementaalisessa mallissa ei tarvitse tehdä uudelleen niin paljoa dokumentointia, kuin esim. vesiputousmallissa. [3], [9], [10]

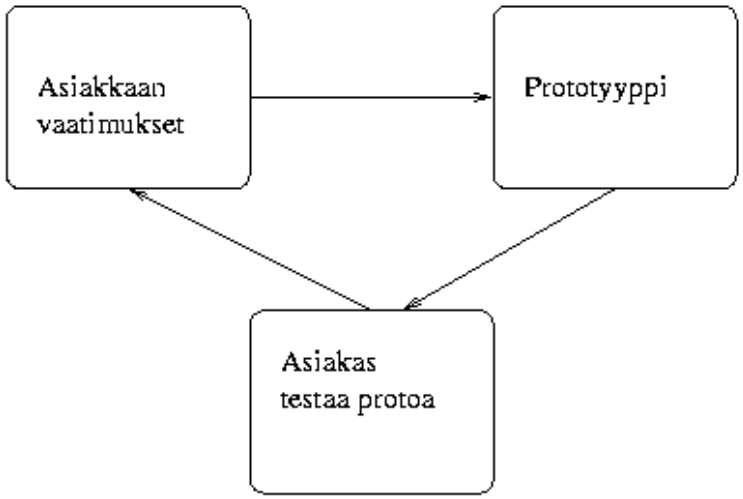
Suurimmat inkrementaalisen mallin vaikeudet tulevat suurissa, monimutkaisissa ja aikaa vievissä projekteissa, koska niissä eri tiimit voivat kehittää eri osia ohjelmaa. Ongelmana mallissa on myös se, että prosessi ei ole selkeästi näkyvissä. Projektijohtajan täytyy tehdä säännöllisiä tarkastuksia projektin etenemisestä. Nopealla aikataululla kehitettäessä kustannukset kasvavat, jos dokumentoidaan kaikki versiot. [3], [9], [10]

Inkrementaalisenmallin yhtenä ongelmana on ohjelman kehitys aikaisempien versioiden päälle, jolloin uusien ominaisuuksien lisääminen myöhäisemmässä vaiheessa voi olla vaikeaa ja aikaa vievää. Uusien versioiden luominen vanhojen versioiden päälle voi myös aiheuttaa ongelmia aikaisemman versio toiminnassa. Inkrementaalisessa kehityksessä suurien muutoksien tekeminen voi olla hankalaa, koska myöhäisemmät versiot pohjautuvat alkuperäiseen versioon. [3], [9], [10]

Inkrementaalinen malli sopii sellaisiin projekteihin joiden määrittely ja vaatimukset ovat selkeät ja ne on hyvin ymmärretty. Projektilla täytyy olla selkeät pääominaisuudet joiden pohjalta ensimmäistä versiota lähdetään kehittämään. Inkrementaalinen malli sopii erityisen hyvin sellaisiin projekteihin joissa tuote täytyy saada aikaisin markkinoille. Jos projektissa on muutamia riskikohtia, kannattaa käyttää inkrementaalista mallia, koska riskeihin pystytään reagoimaan riittävällä nopeudella. Inkrementaalinen malli sopii myös tuotekehitysprojekteihin jolloin uusia ominaisuuksia ja tarpeita ilmenee ohjelmistokehityksen edetessä. [9]

3.2.3 Prototyyppimalli

Prototyyppimalli on suunnittelumalli, jossa aluksi määritellään vain ohjelmiston näkyvimmät piirteet. Ohjelmistosta tehdään nopeasti prototyyppi, jonka asiakas saa arvioitavaksi jo alkuvaiheessa. Prototyypin kehittämistä jatketaan asiakkaalta saadun palautteen pohjalta (Kuva 8). Kun asiakas on prototyyppiin tyytyväinen, siihen lisätään mahdolliset puuttuvat osat ja se julkaistaan valmiina tuotteena. [2]



Kuva 8 Prototyyppimalli [11]

Protoilu on niin sanotusti itseään toistava prosessi. Ohjelmistokehityksen aikana testataan monenlaisia, epävarmojakin ratkaisuja. Huonot ratkaisut korjataan toimiviksi tai hylätään epäsopiviksi todettuina. [12]

Prototyyppimalli on erityisen toimiva silloin, kun asiakkaan vaatimukset ohjelmistolta muuttuvat nopeasti tai asiakas ei ole valmis sitoutumaan tiettyyn vaatimusmäärittelyyn. Prototyyppimallin eduksi voidaan lukea myös se, että siinä nähdään tasaisia ja näkyviä edistyksen merkkejä. Prototyyppimallin huonona puolena on se, että projektin alkaessa on hyvin vaikea ennustaa minkäänlaista aikataulua projektin valmistumiselle. Toisaalta asiakas näkee projektin edistyvän tasaisesti kokoajan, joka varmasti auttaa hyväksymään selkeän aikataulun puutteen. [2]

3.3 Ketterät menetelmät

Vuoteen 2000 mennessä oli noin kymmenen kevyttä iteratiivista ketterää ohjelmakehitysmallia. Aiemmin Yhdysvalloissa käytettiin paljon suunnitelmaohjattuja raskaita malleja, joiden lisäksi kehitettiin keveämpiä, ketteriä kehitysmalleja. Vuonna 2001 perustettiin Agile alliance -järjestö, jonka tarkoitus oli edistää ketteriä menetelmiä. Agile alliance -järjestön ns. perustuskirjana julkaistiin Agile manifesto, jonka perusideana oli, että prosessi, työkalut ja dokumentaatio ovat turhia. Kirjassa mainittiin myös, että vaikkakin nämä edellä mainitut ovat tavallaan turhia, ovat ne hyödyksi ohjelmistokehityksen kannalta. Tärkeintä on tyytyväinen asiakas ja toimiva ohjelmisto. Enää ei ketteriä malleja käytetä sellaisenaan, paitsi scrum-mallia, mutta niissä esitettyjä ideoita sovelletaan nykypäivän ohjelmistokehitykseen. Ketterissä menetelmissä käytetään lyhyitä iteraatioita. Iteraatioit saattavat olla jopa niin lyhyitä, että kehitys on lähes jatkuvaa. [1]

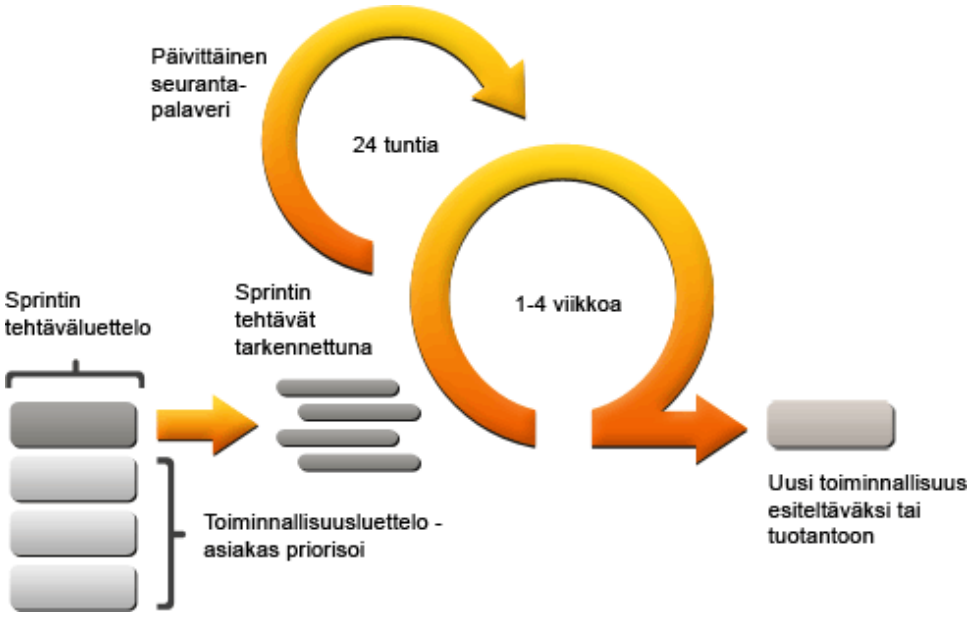
Scrum on viimevuosien yksi yleisimmin käytössä olleista ohjelmistokehitysmalleista. Scrum on alkujaan Japanista ja se esiteltiin ensimmäisen kerran Harward business review lehdessä vuonna 1986. Sen kehittivät japanilaiset Takeuchi ja Nonaka. Lehdessä esiteltiin scrum-mallin keskeiset periaatteet, jotka ovat edelleen käytössä. Ennen kuin ketterä kehitysmalli yleistyi, oli scrum kehitetty jo kymmenen vuotta aiemmin. [1]

Scrumin etuna on yksinkertaisuus, vaikka se ei sitä välttämättä ole, sillä scrum ei käsitä käytettyjä kehitystyökaluja eikä menetelmiä. Periaatteessa scrum ei ole projektinhallintamenetelmä, vaan ennemmin toteutusvaiheeseen tarkoitettu tapa organisoida projektin iteraatiot. Scrum ei siis yksinään riitä, vaan se on yhdistettävä myös muuhun projektinhallintaan. [1]

Scrumissa on kolme tärkeää roolia, tuotteen omistaja, scrum-mestari ja tiimi. Tuotteen omistaja vastaa tuotepäällikköä ja on vastuussa projektin taloudellisesta tuloksesta. Tuotteen omistaja kerää ohjelmavaatimukset työlistaksi ja ylläpitää sitä tärkeysjärjestyksessä. Työlista jaetaan alkioihin, joihin kuuluu alustava aika-arvio. Mitä tärkeämpi alkio, sitä tarkempi kuvaus sillä on. Tuotteen omistajalla on kaikki vastuu vaatimusten hallinnasta, joten hänellä on iso vastuu projektin onnistumisesta. [1]

Scrum-mestari vastaa normaalin projektin projektipäällikköä. Scrum-mestari on vastuussa siitä että prosessia noudatetaan ja hän toimii tiimin ja tuotteen omistajan ohjaajana. Scrum-mestarin vastuualueena on vastata pyrähdyksen tuloksesta ja että määritellyt ehdot toteutetaan. [1]

Tiimin optimaalinen koko on noin seitsemän henkilöä. Tiimi koostuu mielellään eri alojen spesialisteista, kuten käyttöliittymän suunnittelija, testaaja ja ohjelmoija. Tiimi on itseorganisoituva, eli sillä ei ole virallista projektipäällikköä, vaikkakin scrum-mestari usein mielletään projektipäälliköksi. Tiimi vastaa pyrähdyksen työlistan jakamisesta keskenään. [1]



Kuva 9 Scrum-malli [14]

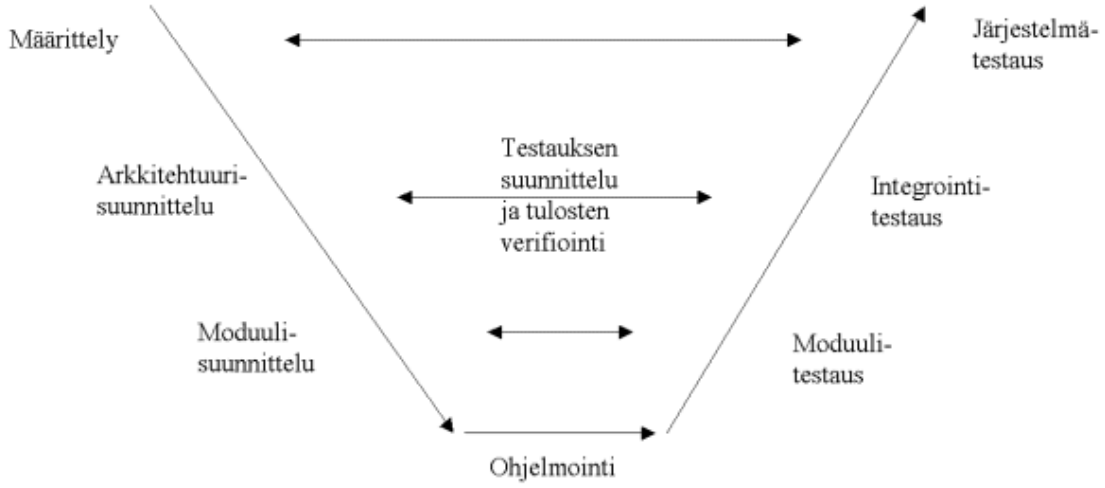
Projekti etenee pyrähdyksissä, joiden pituus on noin 1–4 viikkoa. Projekti aloitetaan suunnittelukokouksella. Scrum-malli toimii toistavasti ja lisäävästi, jotta riskejä pysty-

tään kontrolloimaan ja ennustamaan. Tuote ei tule kerralla valmiiksi, vaan täydentyy projektin edetessä täydellisemmäksi. Jokaisen pyrähdyksen lopuksi pidetään pyrähdyskatselmus, jossa esitellään projektin konkreettiset saavutukset (Kuva 9). [1], [13]

4 OHJELMISTOTESTAUS

Ohjelmiston testaus on tärkeä osa ohjelmistokehitystä, jotta viallisia tuotteita ei pääsisi asiakkaalle asti. Ohjelmiston kehitystyö ja testauksen suhdetta havainnollistetaan usein ns. v-mallilla. Tulokset voidaan todeta oikeiksi vertaamalla niitä vastaaviin dokumentteihin, jotka ovat syntyneet määrittelyn, arkkitehtuurisuunnittelun ja moduulisuunnittelun tuloksena. [4]

V-mallissa on kolme erillistä testaustasoa, jotka ovat yksikkötestaus, integrointitestaus ja järjestelmätestaus (Kuva 10). Järjestelmätestausta voi myös joskus seurata erillinen kenttä tai hyväksymistestaus, joka on asiakkaiden ja järjestelmän käyttäjien tekemä testi. Tämä testi varmistaa sen, että ohjelma on valmis otettavaksi käyttöön. V-mallin mukaisesti testauksen suunnittelu tapahtuu testaustasoa vastaavalla suunnittelutasolla. Järjestelmätestaus suunnitellaan määrittelyvaiheessa, integraalitestaus suunnitteluvaiheessa ja moduulitestaus moduulisuunnitteluvaiheessa. [4], [15]



Kuva 10 V-malli [4]

4.1 Moduulitestaus

Moduulitestausvaiheessa testataan yksittäiset moduulit. Moduuli on ohjelman erillinen looginen kokonaisuus, joka koostuu noin 100–1000 ohjelmarivistä. Moduulin toimintaa verrataan tekniseen määrittelydokumenttiin. Moduulitestauksen suorittaa usein sama henkilö, joka on moduulin toteuttanut. [4]

Testeissä pyritään selvittämään moduulin logiikan ja tietorakenteen ongelmia. Kun ohjelma on jaettu moduuleihin, on ohjelma helpompi testata, koska laajemmista kokonaisuuksista on vaikeampi löytää virheitä. Moduulitestauksessa pyritään havaitsemaan ristiriitoja moduulien toiminnan ja määrittelyjen välillä. Ohjelmistoa ei voi testata pelkästään moduulitestauksella, koska sen avulla ei saada käsitystä laajojen kokonaisuuksien toiminnasta. [15]

4.2 Integrointitestaus

Integrointitestauksessa yhdistellään moduuleita tai moduuliryhmiä. Testauksen painopiste on moduulien välisten rajapintojen tutkimisessa. Tuloksia verrataan tekniseen määrittelyyn. Integrointitestaus kulkee rinnan moduulitestauksen kanssa ja sitä onkin usein tarpeetonta tarkastella erikseen. [4]

Integrointitestaus suoritetaan yleensä kokoavasti alimman tason moduuleista ylöspäin tai jäsentävästi ylimmästä tasosta alaspäin. Integrointitestaus löytää virheet, jotka syntyvät moduuleita yhdistettäessä eli toiminta ja suoritustason virheet. [4]

Integrointitestauksessa on kolme yleismallia. Yksi niistä on big bang, jossa testaajalle on toimitettu ohjelmisto kokonaisuudessaan ja osien toimintaa testataan keskenään ristiin. Top down -mallissa ensin testataan päällystason moduulit ja sen jälkeen niiden toiminta alimoduuleiden kanssa. Kolmas malli on bottom up, jossa alimoduulit testataan ensimmäiseksi ja vasta sitten päällystason moduulit. [16]

4.3 Järjestelmätestaus

Järjestelmätestauksessa koko järjestelmää verrataan toiminnalliseen määrittelyyn ja asiakasdokumentaatioon. Järjestelmätestauksen suorittaja on yleensä joku ulkopuolinen, koska suunnittelijan on usein vaikea löytää järjestelmästä epäkohtia. Järjestelmätestauksessa testataan myös ei-toiminnalliset osuudet, joita ovat kuormitustestit, luotettavuustestit, asennustestit ja käytettävyystestit. [4]

Mitä korkeammalla tasolla testauksessa virhe havaitaan, sitä kalliimpi se on korjata, koska korjaus voi aiheuttaa uusia virheitä sekä korjauksen jälkeen pitää tehdä uudet testaukset myös alemmilla tasoilla. Tälläinen uudelleentestaus tunnetaan nimellä regressiotestaus. Manuaalisesti suoritettuna regressiotestaus voi olla erittäin kallista. [15]

Yksinkertaisimmillaan regressiotestaus on siis aikaisempien integraatiotestien uudelleen suorittamista. Joidenkin testien ajaminen saattaa viedä huomattavan paljon aikaa, eikä näin ole mahdollista tehdä montaa kertaa uudelleen. Joitakin osia saattaa olla turha testata uudelleen, koska nämä eivät ole muuttuneen koodin vaikutusalueella. Tästä syystä regressiotestausta voidaan toteuttaa kohdistettuna tiettyihin moduuleihin, jolloin pystytään vaikuttamaan suoritettavien testien määrään. [15]

4.4 Hyväksymistestaus

Hyväksymistestaus on yksi projektin haastavimmista testausvaiheista. Hyväksymistestauksen tehtävänä on varmistaa, että asiakkaalle toimitetaan toimiva ja tavoitteiden mukainen ohjelmisto. Hyväksymistestaus suoritetaan joko viimeisenä testausvaiheena, tai se voidaan suorittaa osana järjestelmätestausta. [15], [17]

Kun ohjelmistotestauksessa noudatetaan v-mallia, hyväksymistestaus on ensimmäinen vaihe testauksen suunnittelussa ja viimeinen vaihe toteutuksessa. Hyväksymistestaussuunnitelma tulee pitää ajan tasalla, mikäli asiakas muuttaa vaatimuksia kesken kaiken. Hyväksymistestauksen tekee yleensä tuotteen loppukäyttäjä ja testausympäristön tulisi olla mahdollisimman todenmukainen. [18]

Hyväksymistestauksen pääperiaate on siis osoittaa asiakkaalle, että tavoitteisiin on päästy. Tässä vaiheessa virheiden etsiminen ei enää ole ajankohtaista. Jos virheitä

vielä löydetään, niiden korjaaminen tulee kalliiksi. Poikkeuksena on kuitenkin tilanne, jossa asiakas näkee valmiin työn vasta hyväksymistestausvaiheessa ensi kerran, silloin virheitä valitettavasti usein löytyy. Loppuvaiheessa suurien muutosten tekemistä kannattaa harkita tarkasti ja miettiä onko muutosten tekeminen todella tärkeää. Jos muutoksia kuitenkin päätetään tehdä, tulee riskit kartoittaa uudelleen. [18]

5 JÄRJESTELMÄN LUOMINEN JA MS ACCESS

Tässä luvussa käsitellään relaatiotietokantojen suunnittelua ja MS Access -tietokantahallintasovellusta ja sen ominaisuuksia.

5.1 Relaatiotietokannat ja ER-kaaviot

Tietokanta on usein tietotekniikassa käytetty nimitys tietovarastolle. Tietokanta on varasto tietoja, joilla on jonkinlainen suhde toisiinsa. Tietokanta ei aina ole sähköinen, vaan se voi olla myös paperille kerättyjä tietoja. Tietokantaan kerätään tietoja rajatusta aiheesta, kuten yrityksen asiakkaista. [19]

Tietokantoja on paljon erikokoisia, ja ne voivat sisältää eri formaateissa olevaa tietoa, kuten mediatiedostoja ja tekstiä. Tietotekniikan alalla tietokannat kuvataan usein taulukoina. Yhdessä taulussa esitetään tietoa yhdestä aiheesta. Esimerkiksi asiakastaulu voi sisältää tietoja, kuten asiakkaan nimi, osoite, puhelinnumero ja sukupuoli. Taulun jokaisessa sarakkeessa on aina tietynlainen tietotyyppi, joka määrittää, minkälaista tieto siihen sarakkeeseen tullaan tallentamaan. Sarakkeeseen voidaan myös määrittää, voiko kyseisessä sarakkeessa olla tyhjiä soluja. [19]

Relaatiotietokannat on keksinyt E. F. Codd IBM:llä vuonna 1970. Relaatiotietokannan etuna on tuoda paljon tietoa kokoon nopeasti. Relaatiotietokannan ideana on, että tietoja ei tallenneta kahteen kertaan tietokantaan. Tiedot jaetaan mahdollisimman pieniin osiin, esimerkiksi nimi voidaan jakaa etunimi, toinen nimi ja sukunimi. Kaikilla tiedoilla on pääavain, jonka arvo voi esiintyä vain kerran taulukossa ja identifioi tiedot. [19]

ER-kaavioita käytetään apuna tietokantaa mallintaessa. ER tulee sanoista Entity Relationship joka, kuvastaa hyvin menetelmää. ER-mallinnus on oliopohjainen käytäntö, jolla voidaan kuvata käsitteitä ja niiden suhteita. Mallinnukseen on useita eri tyylejä, vaikka periaatteeltaan ne ovat samanlaisia. Eri ER-malleja ovat esimerkiksi Chen, Bachman, Barker, EXPRESS, IDEF1X ja UML. ER-malli on käsitekaava, joka mallintaa tietokannan sisällön ja käsitteellisen rakenteen sekä määrittelee tietoihin liittyviä sääntöjä. ER-mallin tarkeimpiä käsitteitä ovat kohde, attribuutti ja suhde. [20] [21]

Kohde kuvataan ER-mallinnuksessa neliöllä. Kohde kuvastaa eroteltavaa ja ymmärrettävää asiaa tai tapahtumaa, kuten ajoneuvoa, rakennusta tai henkilöä. Kohdejoukko on joukko, joka sisältää kaikki samaan kategoriaan liittyvät asiat, kuten kaikki henkilöt ja kaikki rakennukset. Samanlaisuudet kohteissa ilmenevät attribuuttien ja suhteiden samankaltaisuudessa. Kaikkiin kohteisiin liittyy siis samoja tietoja. [20]

Attribuutit kuvataan ER-mallinnuksessa soikiolla, joka yhdistetään yleensä kohteeseen. Attribuutti kuvastaa jonkin kohteen yksittäistä piirrettä, josta kohteessa ollaan kiinnostuneita. Attribuutti voi olla, muun muassa nimi. Attribuutit saavat arvoja ja esimerkiksi nimi attribuutin arvo voi olla Pentti. Jokaisella kohteella pitää olla jokin avainattribuutti, joka yksilöi tiedon. Avainattribuutti tulee siis olla sellainen, että se ei voi esiintyä kuin vain kerran. Henkilökohteelle hyvä avainattribuutti on esimerkiksi henkilötunnus, koska se on yksilöllinen. [20]

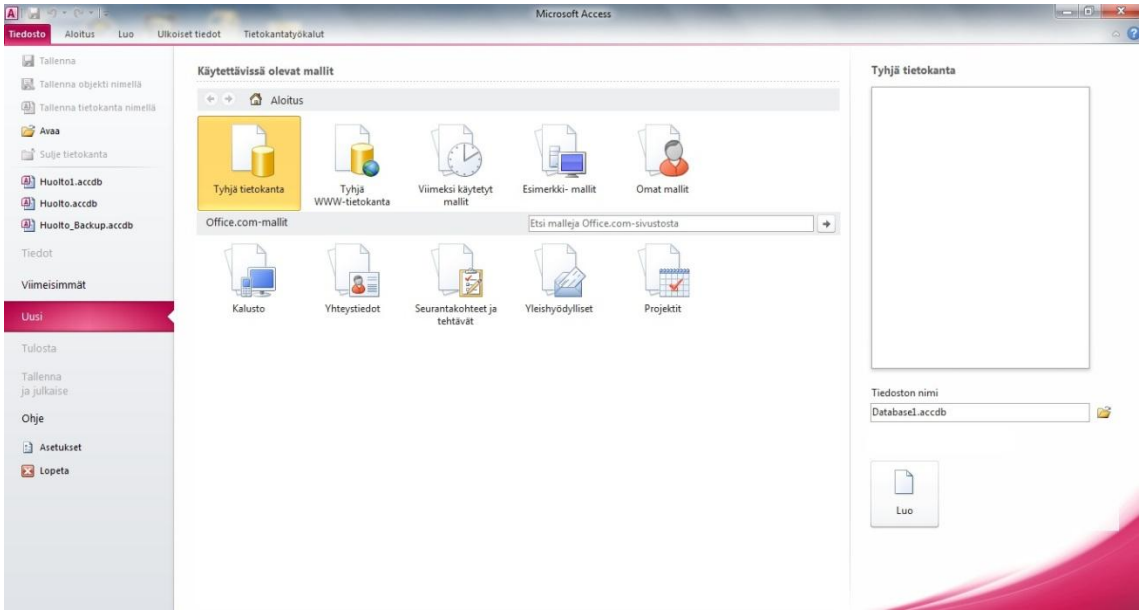
Esimerkissä mainittu henkilötunnus on ns. luonnollinen avain, koska sen arvo on olemassa ja se täyttää yksilöllisyyskriteerit. Vaihtoehtona on liittää entiteettiin geneerinen avain, joka tyypillisesti on inkrementoiva luku. Sen avulla voidaan saavuttaa nopeusetuja, mutta tällöin yksilöllisyyskriteeri ei täyty ja sama tietorivi voi esiintyä rajattomasti, ellei tietokannalle määrätä hylätä duplikaatit valituissa attribuuteissa. [22]

ER-mallinnuksessa suhdetta kuvataan salmiakkikuviolla. Suhde tulee olla yhdistettynä vähintään kahteen kohteeseen, joiden välistä suhdetta kuvataan. Esimerkiksi Pentti omistaa television, jossa Pentti ja televisio ovat kohteet, joiden välistä suhdetta kuvataan. Suhdejoukot muodostuvat samalla tavalla kuin kohdejoukot. Suhdejoukot muodostuvat samanlaisista suhteista, joiden mallintamiseen nelikulmiota on käytetty. Suhdeaste kuvaa yhden suhteen sitomien kohteiden lukumäärän. Suhteilla on myös kardinaalisuus, joka kuvaa, kuinka monta kohdetta voi suhteeseen samanaikaisesti liittyä. Kardinaalisuus määrää, kuinka moneen suhteeseen kohde voi samanaikaisesti osallistua. Esimerkiksi henkilö voi omistaa monia laitteita, mutta laitteen voi omistaa vain yksi henkilö. [20]

5.2 MS Access

Microsoft Access on tietokantojen hallintaohjelmisto, jolla voidaan luoda ja hallita relaatiotietokantoja. Access tulee Office Enterprise ja Professional -pakettien mukana tai

sen voi ostaa erikseen. MS Access muistuttaa ulkoasultaan muita Officen ohjelmistoja ja voi siksi olla tutumpi käyttää myös tietokantojen suunnitteluun (Kuva 11). Access on helppokäyttöinen tietokantaohjelmisto ja sopii siksi vähemmän tietokantojen luomiseen perehtyneelle henkilölle. [23]



Kuva 11 MS Access aloitusvalikko

MS Accessissa on seitsemän pääosa-aluetta, jotka ovat taulut, taulujen väliset suhteet, kyselyt, lomakkeet, raportit, makrot ja moduulit. Taulut ovat tietokannan selkäranka ja tietovarasto johon tietokannan tiedot tallennetaan. Huono taulujen rakenne ja suunnittelu voi aiheuttaa ongelmia, kuten tietokannan hitaus tai väärät tulokset kyselyissä. Kyselyt ja lomakkeet perustuvat taulujen tietoihin. Taulujen ulkonäkö Accessissa muistuttaa paljon esimerkiksi Excelin taulukoita. Taulujen väliset suhteet luodaan siten, että ne yhdistävät taulut joissa on yhteisiä tietoja. [23]

Kyselyt ovat tärkeässä osassa tiettyjen tietojen näyttämisessä näytöllä. Access käyttää kyselyissä SQL-kieltä. Kyselyillä voidaan järjestellä, kerätä, suodattaa ja suorittaa laskutoimituksia taulujen tiedoista. SQL-koodia ei tarvitse välttämättä kirjoittaa itse, vaan Access tekee sen puolestasi, kunhan ohjelmoija ensin määrittelee tiedot, jotka hän haluaa näkyviin. [23]

Lomakkeissa tiedot näytetään käyttäjälle ja niissä voidaan syöttää tietoja tietokantaan. Lomakkeiden avulla voidaan myös muokata ja poistaa tietoja. Lomakkeet tulee luoda

niin, että niiden avulla voidaan hallinnoida tietokantaa ja se sisältää kaikki tarvittavat tiedot ja ominaisuudet. [23]

Raportit näyttävät halutut tiedot näytöllä, toisinkuin lomakkeissa raporteissa tietoja ei voida muokata. Raportit on tarkoitettu esimerkiksi tulostusta varten. Raporteilla voidaan luoda esimerkiksi laskut tai meidän tapauksessa huoltoraportit. [23]

Makroilla voidaan automaattisesti luoda joitakin haluttuja toimintoja tietokantaan. Valmiita makroja Accessissa on paljon ja niitä voi halutessaan lisätä itse. Makrot toteutetaan siinä järjestyksessä kuin ne on syötetty. Moduulinäytöllä voidaan kirjoittaa itse VBA-koodia. VBA-koodilla voidaan siis myös itse kirjoittaa toimintoja, jos valmiita makroja ei ole tai halutaan ennemmin kirjoittaa oma koodi makrojen sijaan. [23]

6 PROJEKTIN TOTEUTUS

Projekti toteutettiin asiakastyönä Turun Huoltoexpertit Oy:lle. Turun Huoltoexpertit on vuonna 1985 perustettu turkulainen yritys. Yritys on kooltaan suhteellisen pieni ja se työllistää muutaman työntekijän. Yrityksessä huolletaan elektroniikkalaitteita tietokoneista pölynimureihin. Huoltoexpertit tekevät takuuhuoltoja ainakin Pioneerin, Yamahan, Audiopron ja JVC:n laitteisiin. [24]

Asiakasyritys halusi sähköisen huoltojärjestelmän vanhan järjestelmän tilalle. Aikaisemmin työt kirjattiin sisään paperisille lomakkeille, joissa oli kaikki tarvittavat laite- ja asiakastiedot. Lomake kulki laitteen mukana läpi koko huoltoprosessin. Tämä ns. vanhanaikainen systeemi alettiin kokea kuitenkin hankalaksi ja töiden etenemistä oli vaikea seurata. Varsinkin tapauksissa joissa asiakas tiedusteli laitteensa edistymistä, oli hankala nopeasti selvittää laitteen huoltotila. Tästä seurasi tarve sähköiselle järjestelmälle, jossa kaikki tarvittavat ja oleelliset tiedot olisivat helposti ja nopeasti kaikkien työntekijöiden saatavilla. Sähköistä järjestelmää oli pohdittu jo kauan, mutta vaikka olemassa olevia ohjelmistoja oli tutkittu, ei sopivaa ohjelmistoa ollut löydetty, koska useat olivat liian raskaita ja monimutkaisia käyttää.

6.1 Esitutkimus

Toimeksiannon jälkeen aloitettiin nopeasti työn esitutkimusvaihe. Työn esitutkimusvaihe aloitettiin keskustelemalla toimeksiantajan kanssa heidän toiveistaan huoltojärjestelmälle. Päällimmäisenä ajatuksena oli selkeä ja helppokäyttöinen järjestelmä, koska aikaisempaa käyttökokemusta vastaavasta järjestelmästä ei ollut. Toisaalta järjestelmän tulisi kattaa oleellisimmat tiedot joita huoltoprosessissa tarvitaan.

Toimeksiantaja selvensi huoltoprosessin vaiheita, koska suunnitteleminen olisi helpompaa, kun on hyvä käsitys huollon toiminnasta. Tämä antoi myös valmiuksia ideoida itse asioita, joita asiakas ei ollut tullut ehkä ajatelleeksi. Esitutkimukseen käytettiin paljon aikaa ja siinä haastateltiin asiakasta, jotta asiakkaan todelliset tarpeet tulivat esille. Asiakkaan kanssa käytiin läpi joitakin valmisohjelmistoja, niiden hyviä ja huonoja puolia. Tämä antoi hyvän käsityksen siitä, millaisen järjestelmän asiakas halusi.

Esitutkimusvaiheessa dokumentoitiin asiakkaan haastattelut sekä kirjattiin projektin tavoitteita. Dokumentaatiosta syntyi ajatuskartta, jossa haluttuja ominaisuuksia oli listattu sen tarkemmin vielä puuttumatta niiden totutukseen.

Sähköisen järjestelmän oli tarkoitus aluksi tulla vanhan järjestelmän rinnalle, ja myöhemmin siirryttäisiin kokonaan sähköiseen järjestelmään. Esitutkimusvaiheessa valittiin sopiva suunnittelumalli kyseisen projektin toteutukseen. Suunnittelumallissa päädyttiin hieman muokattuun vesiputousmalliin, joka sopi hyvin tämän tyyliseen projektiin. Malli sopii myös siitä syystä tähän projektiin, että malli on yksinkertainen ja nopeasti ymmärrettävissä. Vesiputousmalliin päädyttiin myös osaksi siksi, että asiakasyrityksen ei tarvinnut osallistua itse projektiin.

6.2 Määrittely

Määrittelyvaiheessa tehtiin tarkemmat vaatimukset järjestelmän ominaisuuksien toiminnasta. Ohjelma tulisi toimia siten, että monelta tietokoneelta pystytään syöttämään samanaikaisesti tietoja tietokantaan. Käyttöliittymä tulisi olla selkeä ja mahdollisimman yksinkertainen, koska asiakasyrityksen henkilökunnalla ei aiempaa kokemusta vastaavasta ohjelmistosta ole. Sovellus tulisi toteuttaa siten, että ylimääräisiä kustannuksia ei ohjelmiston käytöstä syntyisi. Sovellus tulisi toteuttaa siis joko jollain ilmaisella tietokantasovelluksella tai MS Accessilla, koska yrityksellä on käytössä entuudestaan Office Enterprise. Ohjelman tulisi sisältää seuraavat ominaisuudet:

- Sisäänkirjautumissivu
- Asiakastietojen lisäys ja muokkaus
- Laitteen lisäys
- Huoltotila
- Vikahaku
- Työlista
- Varasto
- Työntekijöiden hallinta
- Työhintojen hallinta
- Työtilojen hallinta.

Sisäänkirjautumissivuun tulisi sisältyä työntekijän nimi ja salasana, jolla järjestelmän saa auki. Kirjautumissivulla on hyvä olla myös sellainen toiminto, että jos salasanan arvaa liian monesti väärin, ohjelmisto sulkeutuu. Tietokanta tulee sisältämään luottamuksellista tietoa, ja siitä syystä sen tulee olla salasanasuojattu. Ohjelma tulisi pystyä myös sulkemaan sisäänkirjautumissivulta.

Vastaanottolomakkeeseen tulisi voida helposti kirjata asiakkaan ja laitteen tiedot, kun laite vastaanotetaan huoltoon. Vastaanottolomakkeeseen tulisi pystyä kirjaamaan ainakin seuraavat laitteen tiedot: merkki, malli, sarjanumero, vastaanottopäivämäärä, sisäänkirjaaja, vikakuvaus ja mukana tulleet lisälaitteet. Lomakkeessa tulisi olla myös paikka asiakkaan tiedoille, joihin lukeutuu nimi, osoite, puhelinnumero, sähköposti ja asiakasnumero. Asiakkaan tiedot tulisi pystyä tallentamaan järjestelmään pysyvästi niiden mahdollista myöhempää käyttöä varten.

Huoltolomakkeessa tulisi näkyä laitteen tärkeimmät tiedot, asiakkaan tiedot yhteydenottoa varten ja kustannusarvio. Myös laitteen huoltohistoria tulisi olla nähtävissä helposti. Lomakkeessa tulisi olla myös paikka, johon kirjoittaa huollossa tehdyt toimenpiteet.

Vikahaku tulisi olla sellainen, jossa pystytään laitteen mallilla, merkillä tai vialla hakemaan aikaisemmin huollossa olleita laitteita. Vikahaussa tulisi näkyä laitteessa ollut vika, ja miten vika on korjattu.

Työlistan tärkeimpinä ominaisuuksina on nähdä jokaisen työntekijän työnalla ja valmiina olevat laitteet. Työlistan avulla on helppo seurata missä vaiheessa laitteen korjaus etenee. Korjauksen kestoa tietokanta ei kellota, koska työhinnasto ei ole tuntipohjainen.

Varastojärjestelmä, jossa pystytään seuraamaan osien varastosaldoja ja hyllypaikkoja, jotta komponenttien löytäminen tarvittaessa olisi helpompaa. Varastojärjestelmässä tulisi myös olla kirjattuna varaosien hinnat, jotta kustannusarvioiden laskeminen olisi helpompaa.

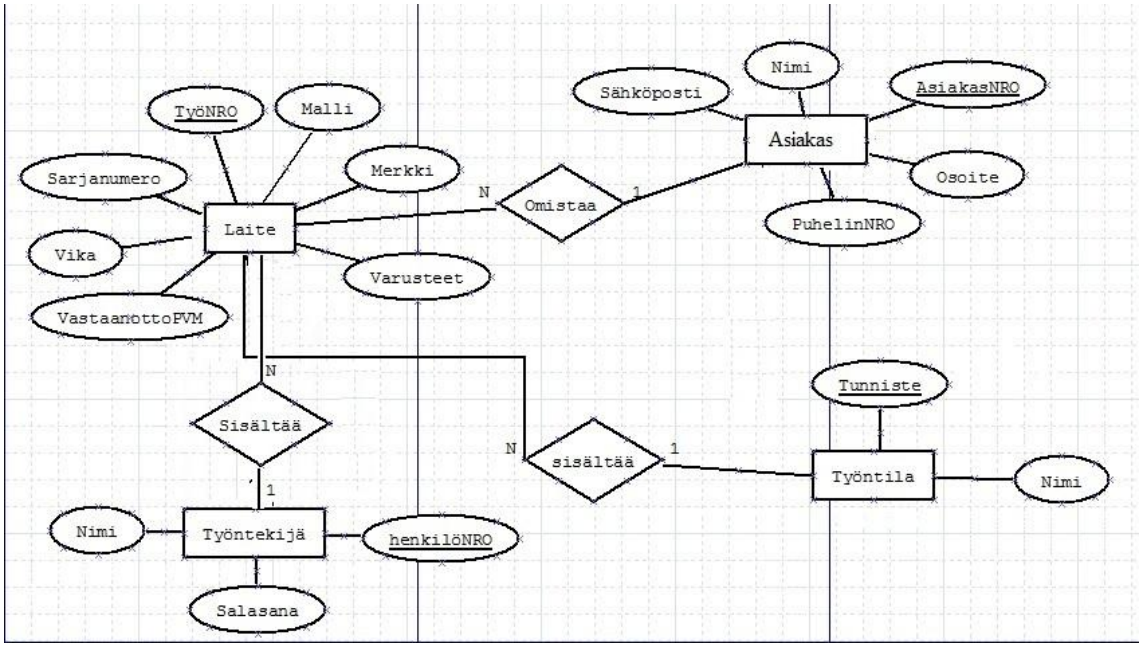
Järjestelmässä tulisi pystyä lisäämään, muokkaamaan ja poistamaan työntekijöitä ja heidän henkilökohtaisia salasanojaan. Asiakasyrityksen työhinnasto sekä työkuvaukset tulisi olla muokattavissa, jotta mahdolliset muutokset olisi helppo päivittää myös tieto-

kantaan. Työntilat, joita ovat esimerkiksi vastaanotettu, työnalla ja valmis, tulisivat olla muokattavissa, mikäli uusia vaiheita huoltoprosessiin tulevaisuudessa ilmenee.

6.3 Suunnittelu

Määrittelyiden pohjalta tehtiin yksityiskohtainen suunnitelma järjestelmän toteutuksesta. Suunnitteluvaiheessa valittiin tietokantasovellukseksi MS Access, koska se on asiakasyritykselle ja projektille sopivin tietokantasovellus. Tietokanta suunniteltiin jaettavaksi kotiverkon välityksellä, jotta tietokantaan pystytään liittymään monelta eri tietokoneelta samanaikaisesti.

Tietokannan suunnittelu aloitettiin tekemällä ER-kaavio, jossa kaikki määrittelyvaiheessa ohjelmalle valitut ominaisuudet luotiin kaavioksi, josta ilmenee niiden väliset yhteydet (Kuva 12). ER-kaaviot luotiin ilmaisella Dia Portable -ohjelmalla.



Kuva 12 ER-kaavio

Tietokannan suunnittelu jaettiin kahteen osaan, arkkitehtuurisuunnitteluun ja moduulisuunnitteluun. Arkkitehtuurisuunnitteluvaiheessa suunniteltiin ohjelmiston laajemmat kokonaisuudet, kuten ohjelman ulkoasu. Arkkitehtuurisuunnitteluvaiheessa valittiin oh-

jelmistoon tulevat ominaisuudet ja tarkemmat yksityiskohdat jätettiin vielä myöhäisempään suunnitteluvaiheeseen.

Moduulisuunnitteluvaiheessa suunniteltiin miten jokainen ominaisuus saataisiin toteutettua yksittäisenä osana järjestelmää. Moduulisuunnitteluvaiheeseen kuului taulujen, kyselyiden ja lomakkeiden suunnittelu. Taulujen suunnittelussa pyrittiin valitsemaan mitä kenttiä taulujen tulisi sisältää ja niiden oikeat tietotyypit. Lomakkeet suunniteltiin siten, että ne kattavat asiakkaan ohjelmistolle määräämät vaatimukset.

6.4 Toteutus

Työ toteutettiin MS Access -ohjelmistolla, koska se koettiin parhaimmaksi vaihtoehdoksi asiakasyrityksen tarpeille. Työtä lähdettiin toteuttamaan luomalla tarvittavat taulut ja yhteydet niiden välille. Tauluiksi luotiin sellaiset osa-alueet, jotka oli määrittelyvaiheessa määritelty.

Taulujen rakenne muodostuu kentistä ja jokainen kenttä edustaa yhtä tietoa. Kentille valitaan tietotyyppi sen mukaan, millaista tietoa siihen aiotaan syöttää. Esimerkiksi vastaanottopäivämääräkentän tietotyyppi on päivämäärä/kellonaika (Kuva 13). Pääavainkentän tietotyyppi on yleensä laskuri, jotta lisättävä tieto on aina uniikki. Kenttiin joihin syötetään paljon tekstiä, kuten vikakuvaus kentässä, on hyvä valita memo-tietotyyppi. Memo-kenttään voidaan syöttää jopa 65 000 merkkiä, kun taas tyypin Teksti-kentän enimmäismerkkimäärä on 255 merkkiä. Teksti-tietotyyppiä kannattaa kuitenkin käyttää vain sellaisissa kentissä joihin ei ole odotettavissa pitkää tekstiä, koska tiedoston kokoon vaikuttaa myös valittu tietotyypppi, eikä pelkästään kirjoitetun tekstin määrä. Ei siis kannata ajatella, ”laitanpas varmuudeksi isomman tietotyypin”, koska tämä kasvattaa tiedoston kokoa ja hidastaa järjestelmän toimintaa Luku-tietotyyppisessä kentässä ei voi lisätä muita kuin lukuja, joten sitä ei voi käyttää muissa kuin sellaisissa kentissä, joihin tiedetään tulevan pelkkiä lukuja. Tyypillisesti näillä on jokin matemaattinen merkitys, kuten inkrementointi. Tietotyppi laskuri, kuten pääavain kuvassa 13 on myös luku. [22]

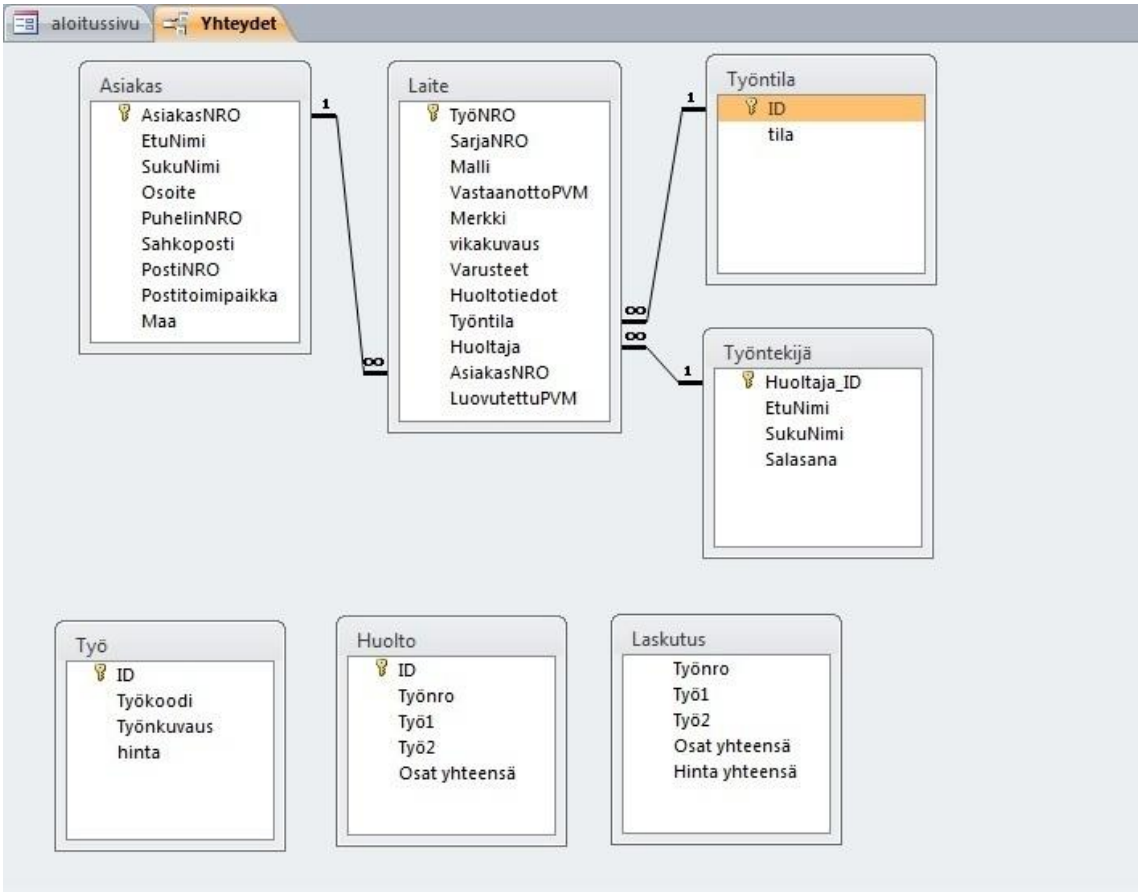
Laite

Kentän nimi	Tietotyyppi	Kuvaus
TyöNRO	Laskuri	
SarjaNRO	Teksti	
Malli	Teksti	
VastaanottoPVM	Pvm./klo	
Merkki	Teksti	
vikakuvaus	Memo	
Varusteet	Memo	
Huoltotiedot	Memo	
Työntila	Luku	
Huoltaja	Luku	
AsiakasNRO	Luku	
LuovutettuPVM	Pvm./klo	

Kuva 13 Laitetaulun rakenne

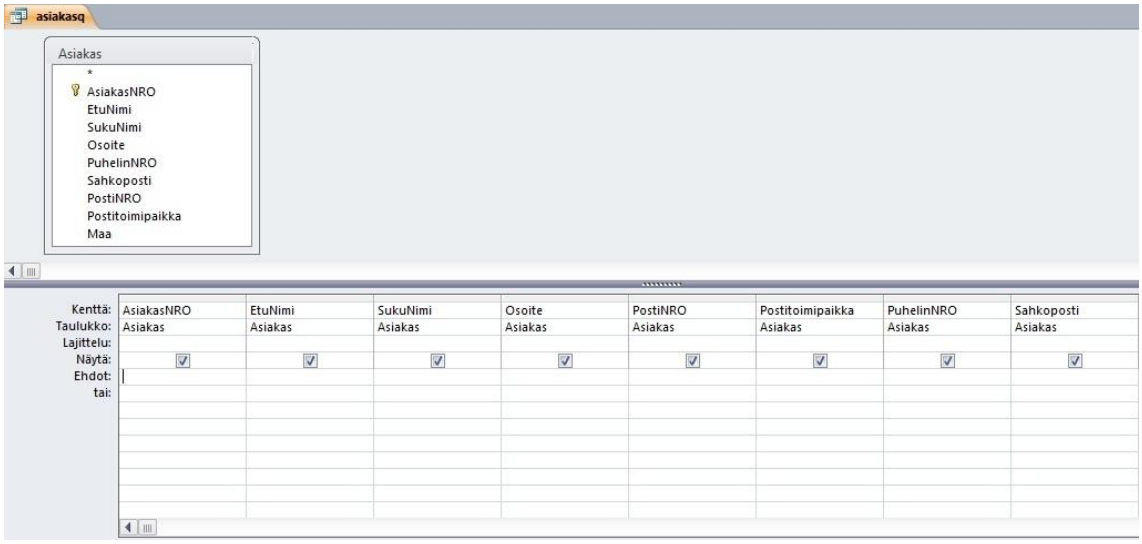
Taulut toteutettiin osa-alueista, joita ovat asiakastiedot, laitetiedot, työntekijät, työntila, työt, huolto ja laskutus (Kuva 14). Taulujen pääavaimet näkyvät kuvassa 14 jokaisen taulun kohdalla pieninä avainsymboleina. Viivat taulujen välillä näyttävät taulujen välisen yhteyden. Asiakas- ja laitetaulujen välillä oleva yhteys on yksi moneen tyyppinen, eli yhdellä asiakkaalla voi olla monta laitetta, mutta yhdellä laitteella on vain yksi asiakas (Kuva 14). Jokaisessa taulussa tulee olla perusavain, jonka avulla taulussa olevat tiedot yksilöidään. Perusavaimen sisältävän kentän tulee olla yksilöllinen. Esimerkiksi asiakastaulun avaimeksi on valittu asiakasnumero, joka ei voi olla sama kahdella eri asiakkaalla (kuva 14). Avaimeksi olisi voinut valita myös asiakkaan henkilötunnuksen, joka on jokaisella yksilöllinen, mutta päädyttiin käyttämään asiakasnumeroa, koska asiakasnumero on selkeämpi ja se on ollut asiakasyrityksellä aiemminkin käytössä.

Kaikki taulut eivät ole yhteydessä toisiinsa, vaan ne voivat olla myös itsenäisiä. Tällöin tauluilla ei ole suhdetta muihin tauluihin. Tauluja joilla ei ole yhteyttä muihin ovat työ, huolto ja laskutus. Taulut ovat itsenäisiä ja niihin on tallennettu yksittäisiä tietoja, joiden avulla on pystytty luomaan tietyt lomakkeet. Näiden taulujen tiedot eivät myöskään saa muuttua, vaikka muiden taulujen tietoja muutettaisiin.



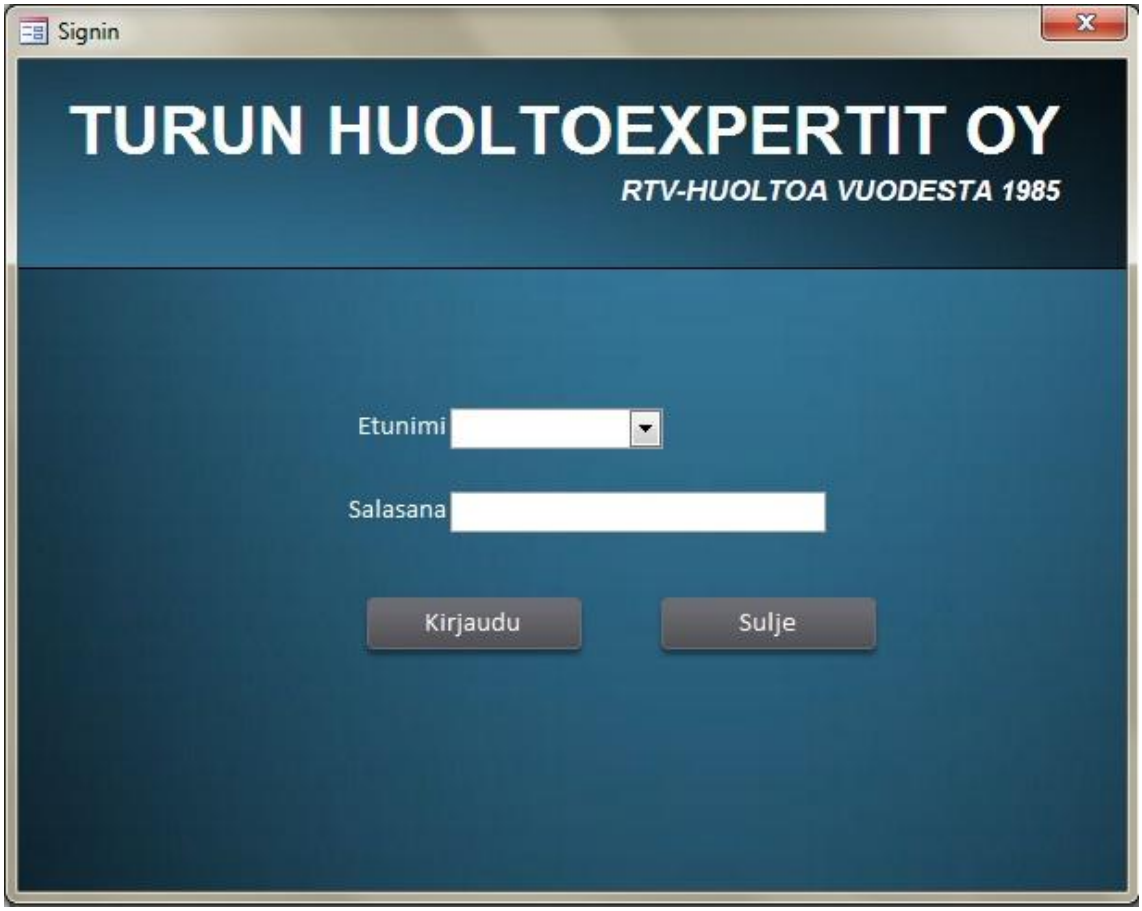
Kuva 14 Taulut ja niiden väliset yhteydet

Kyselyt-ikkunassa voi taulusta valita halutut tiedot näkyviin. Kyselyille voidaan määrittää ehdot, joiden perusteella halutut tiedot tulevat näkyviin. Esimerkiksi asiakaskyselyssä on valittu lähes kaikki kentät, jotta tätä kyselyä voidaan käyttää asiakaslomakkeen pohjana (Kuva 15). Ehtona voidaan käyttää esimerkiksi asiakkaan etunimeä, jolloin näytetään kaikki sen nimiset asiakkaat, jotka tietokannasta löytyvät. Kyselyiden avulla voidaan myös laskea yhteen eri kenttien arvoja, jota on tässä työssä käytetty laskutuksen toteutukseen. Kyselyt käyttävät SQL-kieltä, mutta varsinaista koodia ei tarvitse välttämättä kirjoittaa itse, vaan MS Access tekee sen automaattisesti puolestasi. Joissain tapauksissa joutuu kuitenkin itse kirjoittamaan osan koodista.



Kuva 15 Kyselynäkymä

Lomake on käyttäjälle näkyvä ikkuna, eli siis lomakkeet muodostavat ohjelman käyttöliittymän. Lomakkeet voidaan luoda joko automaattisesti kyselyn tai taulun pohjalta tai lomake voidaan luoda manuaalisesti valitsemalla halutut tiedot. Lomakkeeseen valitaan ne kentät, jotka halutaan käyttäjälle näyttää. Esimerkiksi kirjautumislomakkeeseen haluttiin vain kentät huoltajan etunimeä ja salasanaa varten (Kuva 16). Lomakkeeseen luotiin myös painikkeet kirjautumista ja sovelluksen sulkemista varten (Kuva 16).



Kuva 16 Valmis kirjautumislomake

Lomakkeeseen voidaan lisätä painikkeita ja muita hallintaobjekteja. Painikkeiden avulla voidaan esimerkiksi avata toisia lomakkeita. Tässä työssä tätä on hyödynnetty siten, että huoltotilassa saadaan laitteen omistajan tiedot napin painalluksella (Kuva 17). Lomakesuunnittelussa voidaan muokata myös ohjelman visuaalista ulkonäköä, esimerkiksi taustakuvilla ja -väreillä. Lomakkeisiin voidaan sisällyttää myös alilomakkeita, jotka näkyvät taulukkomuodossa valmiissa lomakkeessa.

Lomakkeen suunnittelutilassa voidaan jokaiselle objektille määrittää tapahtuma, esimerkiksi tallenna-painikkeen toiminta on ohjelmoitu siten, että hiirellä painiketta painettaessa tietokantaan tallentuu avoinna olevan lomakkeen tiedot. Jokaisen objektin toiminta voidaan määrittää ominaisuudet-ikkunassa (kuva 17). Toiminnot voidaan toteuttaa makroilla, jotka ovat valmiita toimintoja MS Accessissa tai kirjoittamalla VBA -koodi itse. Makroista löytyy yleisimmät perustoiminnot, kuten lomakkeiden ja taulujen avaus sekä sivun päivitys. VBA -koodilla voidaan kirjoittaa vaativammat ohjelmoin-

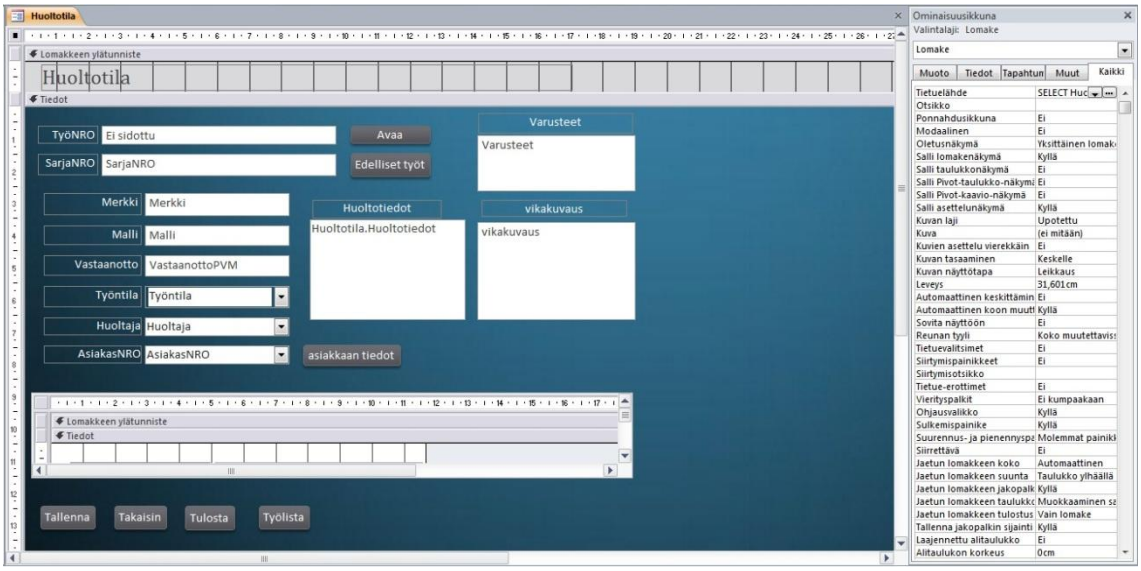
tiosuudet, kuten esimerkiksi kirjautumislomakkeessa kirjoitettiin koodi, jolla ohjelma tarkistaa että käyttäjä nimi ja salasana täsmäävät.

Painikkeisiin käytettiin usein valmiita makroja, mutta haastavammat osuudet tehtiin koodaamalla. Jos halutaan koodata tietty osa lomakkeesta, on yleensä myös hyvä kirjoittaa muukin koodi itse. Suurin ohjelmointiosuus, mitä VBA-koodilla tehtiin, oli sisäänkirjautuminen. Koodi luotiin VBA-koodi ikkunassa. Sisäänkirjaumiseen piti ohjelmoida se, että järjestelmä tarkistaa kirjaudu-painiketta painettaessa seuraavat asiat:

- Onko käyttäjä valittu?
- Onko salasana syötetty?
- Vastaako salasana käyttäjän salasanaa?

Näille kaikille piti määrittää tapahtumat, jotka käsittävät mitä tapahtuu jos ne ovat oikein tai väärin. Lisäksi piti myös määrittää mikä ikkuna aukeaa ja suljetaanko sisäänkirjaussivu kirjautumisen jälkeen. Lomake ohjelmoitiin sellaiseksi, että se ilmoittaa käyttäjälle ponnahdusikkunalla, mikäli käyttäjä tai salasana puuuttuvat sekä ilmoittaa jos salasana on väärä. Kun ollaan syötetty oikea käyttäjä ja salasana kirjaudu-painikkeella päästään siirtymään aloitussivulle.

Itse koodi luotiin ehtolauseilla, jossa ensin tarkistetaan onko käyttäjä kenttä tyhjä, jos ei ole tarkistetaan sama myös salasana kentästä. Jos salasana kenttään on syötetty jokin arvo, verrataan sitä taulussa olevaan arvoon. Jos salasana vastaa taulussa olevaa arvoa avataan aloitussivu ja jos se on väärä, järjestelmä ilmoittaa salasanan vääräksi.



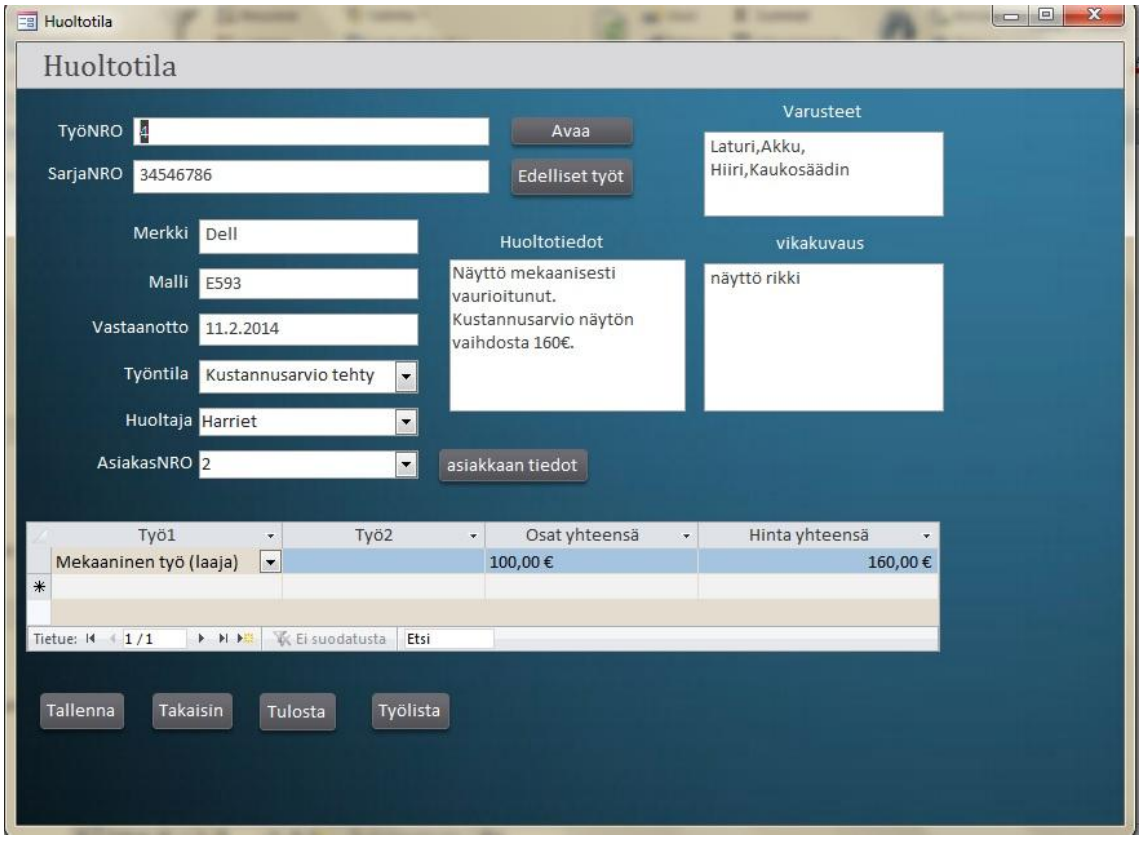
Kuva 17 Lomakkeen suunnittelutila

Työhön suunniteltiin aloitussivu, josta voidaan edetä haluttuun toimintoon, esimerkiksi aloitussivulta pääsee laitteenlisäyslomakkeeseen, jossa laite voidaan kirjata sisään huoltoon. Muut lomakkeet toteutettiin siten, että niistä päästään aina palaamaan aloitussivulle. Lomakkeista löytyy myös tarvittavat painikkeet eri lomakkeiden välillä siirtymistä varten.

Käyttöliittymä tehtiin yksinkertaiseksi, koska se oli yksi toimeksiantajan tärkeimmistä vaatimuksista. Aloitussivulla on suuret ja selkeät painikkeet, joista pääsee nopeasti tarvittaviin lomakkeisiin (Kuva 18). Selkeiden painikkeiden avulla ohjelman käyttö on helpompaa, joka vastaa asiakasyrityksen vaatimuksiin.

Ohjelmiston yksi tärkeimpiä ominaisuuksia on huoltotila, joka on huoltajalla auki työtä tehdessään (Kuva 19). Huoltotilaan voi kirjoittaa toimenpiteet, jotka laitteelle on tehty, ja siinä voidaan myös laskea huollolle hinta, joka on asiakas palvelijan helposti nähtävissä. Huoltoon voi lisätä kaksi työtä, koska työhön yksi, voidaan esimerkiksi lisätä sähköinen työ ja työhön kaksi, voidaan lisätä pientarvike tai jokin muu työ.

Kuva 18 Ohjelman käyttöliittymän aloituslomake



Kuva 19 Huoltotila käytössä

6.5 Testaus

Ohjelma testattiin v-mallin mukaisesti ja testaus suoritettiin ohjelmistotuotannon aikana. Testaus aloitettiin moduulitestauksella, jonka jälkeen siirryttiin integrointitestauksen kautta järjestelmätestaukseen. Lopuksi ohjelmistolle tehtiin hyväksymistestaus, joka suoritettiin asiakasyrityksen toimesta.

Moduulitestaus aloitettiin siinä vaiheessa, kun ensimmäiset kyselyt oli luotu. Kyselyiden toiminta testattiin ennen lomakkeiden luomista. Kyselyiden testaus suoritettiin siten, että ehtoihin syötettiin jokin haluttu arvo, jonka perusteella halutut tiedot saatiin näkyviin. Kun kyselyt oli todettu toimiviksi, luotiin niiden pohjalta lomakkeet. Jokaisen lomakkeen toiminta testattiin osana moduulitestausta. Testivaiheessa pyrittiin käymään läpi kaikki mahdolliset tilanteet, jotka voivat lomaketta käyttäessä syntyä.

Kun jokainen lomake oli todettu toimivaksi irrallisena osana, jatkettiin testausta integrointivaiheeseen. Integrointitestausvaiheessa testattiin lomakkeiden välisiä yhteyksiä. Testaus suoritettiin aina muutaman moduulin välillä, jotta saatiin selkeä kuva moduulien välisten yhteyksien toimivuudesta. Integrointitestausta laajennettiin siten, että yhä useampia moduuleita otettiin mukaan testaukseen.

Kun moduulien väliset yhteydet oli integrointitestauksessa todettu toimiviksi, voitiin suorittaa järjestelmätestaus. Järjestelmätestaukseen kuului tietokannan toimivuuden testaaminen MS Access Runtime -version kanssa. Järjestelmätestauksessa myös testattiin järjestelmän toimivuus usean käyttäjän käyttäessä tietokantaa yhtäaikaisesti. Koska asiakasyrityksessä tietokanta jaetaan kotiverkon välityksellä, testattiin myös järjestelmän toimivuus kotiverkossa.

Kun järjestelmä oli todettu järjestelmätestauksessa toimivaksi, otettiin järjestelmä käyttöön asiakasyrityksessä. Asiakasyritys suoritti ohjelmalle hyväksymistestauksen realistisissa olosuhteissa, yrityksen tiloissa. Ohjelma oli käytössä vanhan järjestelmän rinnalla, jotta vältyttäisiin tärkeiden tietojen häviämiseltä, mikäli ohjelmistossa ilmenisi odottamattomia ongelmia. Asiakasyritys oli tyytyväinen ohjelmiston sisältöön ja toimivuuteen, eli asiakkaan antamat määritykset ohjelmistolle täyttyivät.

6.6 Käyttöönotto

Käyttöönottovaihe suoritettiin kokonaisuudessaan asiakasyrityksen tiloissa. Käyttöönottovaiheen tavoitteena oli saada järjestelmä asennetuksi ja testatuksi sen lopullisessa ympäristössään. Käyttöönottovaihe pyrittiin toteuttamaan kokonaisuudessaan yhden työpäivän aikana, koska käyttöönottovaiheesta voi aiheutua häiriöitä huoltajille. Ohjelmisto oli toteutettu kotona mahdollisimman pitkälle, jotta käyttöönottovaiheeseen ei jäisi paljon muuta tehtävää kuin sellaiset asiat, joita ei voi muualla kuin yrityksessä toteuttaa.

Käyttöönottovaiheessa järjestelmä asennettiin yrityksen tietokoneisiin. Itse tietokanta asennettiin yhdelle tietokoneelle, ja muilla koneilla yhteys tietokantaan otettiin kotiverkon kautta. Microsoft Accessin sivuilta on ilmaiseksi ladattavissa Accessin Runtime -versio, jolla tietokantaa voidaan käyttää, vaikka Accessin täyttä versiota ei olisikaan tietokoneella asennettuna. Runtime -versio on suppea versio Accessista, eikä siinä voi esimerkiksi tehdä muutoksia tietokannan toimintoihin. Kaikilla tietokoneilla ei ollut aikaisempaa Accessin versiota, joten näihin koneisiin asennettiin Runtime -versio, jotta tietokantaan pystytään liittymään. Jokaisella tietokoneella testattiin, että tietokantaa pystyttiin käyttämään ja yhteydet todettiin toimiviksi.

Käyttöönottovaiheessa suoritettiin myös ohjelmiston hyväksymistestaus asiakasyrityksen edustajan toimesta. Asiakasyrityksen henkilökuntaa perehdytettiin ohjelmiston toimintaan huolellisesti, koska myöhemmässä vaiheessa opastusta ei voida enää järjestää muuten kuin puhelimen välityksellä.

Käyttöönottovaiheessa muokattiin työnumerointi alkamaan samasta numerosta josta vanhassa järjestelmässä edettiin, jotta ohjelmistoa voidaan käyttää rinnakkain vanhan järjestelmän kanssa. Työnumeroiden synkronointi on tärkeää, jotta samasta työstä ei ole kahta työnumeroa. Järjestelmään syötettiin myös kaikki huollossa sillä hetkellä olleet laitteet, jotta järjestelmää pystyttiin testaamaan myös käytännössä ja toimintojen opastus oli helpompaa konkreettisilla esimerkeillä. Järjestelmään syötettiin kaikki työntekijät, ja he saivat määrittää omat salasanansa kirjautumista varten. Ohjelmistoon muutettiin vielä yrityksen käyttämät työhinnastot, minkä jälkeen järjestelmä oli valmis käyttöä varten.

6.7 Ylläpito

Valmiin ohjelmiston ylläpitovastuu sovittiin asiakasyritykselle, koska aikaa ei ylläpidolle projektissa jäänyt. Asiakasyritystä opastettiin asentamaan järjestelmän asennuksessa, jotta yritys voi jatkossa ottaa ohjelman käyttöön myös yrityksen uusille tietokoneille. Yritystä opastettiin myös hallinnoimaan ja muokkaamaan ohjelmistontietoja. Ongelmatapauksia varten sovittiin, että yritys voi puhelimitse olla yhteydessä meihin konsultointia varten. Ohjelmistoon ei tarkoitus ollut enää käyttöönoton jälkeen lisätä ominaisuuksia, joten ominaisuuksia lisäävään ylläpitoon ei ollut tarvetta. Ohjelmiston ylläpito rajoittuu siis vain järjestelmäongelmien korjaamiseen.

6.8 Jatkokehitysideat

Kehitettäviä osa-alueita järjestelmään jäi muutamia, vaikka lopulliseen versioon oltiin tyytyväisiä niin ohjelmiston kehittäjien kuin asiakasyrityksenkin puolesta. Lopullisesta versiosta poistettiin muutamia ideoita, joiden pohjalta sovellusta voidaan tulevaisuudessa mahdollisesti kehittää.

Aluksi suunnitelmissa oli toteuttaa tietokantaan myös varastointijärjestelmä, mutta tästä ideasta luovittiin ennen toteutusvaihetta. Varastointijärjestelmä ei olisi ollut välttämättä kannattava, koska asiakasyritys huoltaa monia erilaisia laitteita ja niihin joudutaan usein tilaamaan erikoisia osia, joita ei voida muissa laitteissa käyttää. Yksittäisten varaosien syöttäminen järjestelmään tuottaa vaivaa eikä tällaisenaan ole kannattava osa järjestelmää.

Lopullisesta versiosta jäivät pois viivakoodit, joilla piti toteuttaa työnumerointi ja sarjanumerot. Viivakoodinlukulaitteella olisi voitu lukea huoltoraportista työnumero, jolloin numeroa ei olisi tarvinnut manuaalisesti syöttää järjestelmän huoltotilassa työn avausta varten. Viivakoodinlukulaitteet olisivat kuitenkin tulleet liian kalliiksi ostaa jokaiselle työpisteelle. Ekologista syistä myös turhaa paperin käyttöä pyrittiin vähentämään, jotta ei tulisi niin paljoa jätettä. Kaikki mahdollinen toteutettiin sähköisesti, paitsi huoltoraportti voidaan tulostaa, mikäli asiakas sellaisen haluaa.

Asiakasyritys ehdotti esitutkimusvaiheessa myös sitä, että huoltoon tuoduista laitteista otettaisiin kuvat, jotta laitteiden tunnistaminen olisi helpompaa. Kuvista luovuttiin kui-

tenkin melko nopeasti, koska kuvat olisivat kasvattaneet merkittävästi tietokannan kokoa. Kuvaaminen olisi pitkittänyt myös sisäänkirjausvaihetta, koska kuvat olisi pitänyt siirtää ensin kamerasta tietokoneelle, josta ne olisi voitu lisätä tietokantaan. Myös yleisen kuvakirjaston luominen laitteista olisi ollut hankalaa, koska huoltoon tulee paljon erilaisia laitteita.

Ominaisuuksia jätettiin pois yhteisymmärryksessä asiakasyrityksen kanssa. Mitkään poisjääneistä ominaisuuksista eivät olleet järjestelmän toimivuuden kannalta olennaisia. Ominaisuudet kuitenkin suunniteltiin ja niiden myöhäisempi käyttöönotto on mahdollista, mikäli asiakasyritys tulevaisuudessa näin haluaa.

6.9 Omat huomiot ja päätelmät

Projektia aloitettaessa käytettiin paljon aikaa ohjelmistosuunnitteluun tutustumiseen ja tietokantojen suunnitteluteoriaan perehtymiseen, jotka ovat suurimmaksi osin niin sanottua näkymätöntä työtä projektissa. Aikaa käytettiin siis jo paljon ennen projektin varsinaista aloittamista. Myös erilaiset suunnittelumallit ja testauksen v-malli olivat entuudestaan lähes tuntemattomia.

Aiheisiin tutustuminen ja niiden perinpohjainen tutkiminen tuottivat tulosta, ja projektia voitiin näin lähteä tekemään tukevalta tietopohjalta. Tutkimusten jälkeen aiheesta opittiin paljon uusia näkökulmia, jotka edesauttoivat projektin toteutuksessa.

Suunnittelumalleja tutkittua päädyttiin käyttämään tässä projektissa vesiputousmallia, koska sen selkeä ja systemaattinen eteneminen koettiin tälle kyseiselle projektille parhaaksi. Vesiputousmalli toimii siten, että kun yksi vaihe on tehty kokonaan ja dokumentoitu huolella, seuraavan vaiheen pystyy tekemään täysin eri henkilö.

Vaikka työ toteutettiinkin tiiviissä yhteistyössä, oli projekti helpompi toteuttaa ja pysyä aikataulussa hyvän dokumentoinnin pohjalta. Hyvä dokumentointi helpotti myös kirjallisen osuuden tekemistä. Projektissa huomattiin myös, että tarkempi dokumentointi aiemmin tehtyihin projekteihin verrattuna auttoi pitämään projektin paremmin hallinnassa.

Vesiputousmallin etuja tässä projektissa oli esimerkiksi se, että vesiputousmalli toimi hyvin projektissa, koska uusia määrittelyjä ei enää esitutkimuksen jälkeen tullut. Vesi-

putousmallin toiminta käytännössä tuli tutuksi, mutta varmasti ei voi sanoa, ettei jokin toinen malli olisi voinut toimia paremmin tai vähintäänkin yhtä hyvin.

Vesiputousmallin eräänä huonona puolena huomattiin, että palautetta ohjelmasta ei saatu ennen lopullista versiota. Se ei kuitenkaan haitannut, koska asiakasyritys oli tyytyväinen lopulliseen versioon, johon varmasti vaikutti se, että esitutkimusvaihe oli toteutettu tarkasti ja ajatuksella.

Näin pienessä projektissa päätettiin hieman muuttaa vesiputousmallia ketterämpään suuntaan ja lopulta työ toteutettiin enemmän sashimimallin kuin puhtaan vesiputousmallin mukaisesti. Sashimimallissa työvaiheita suoritetaan hieman limittäin, eikä siis aiemman vaiheen tarvitse olla täysin valmis ennen seuraavaan vaiheeseen siirtymistä. Sashimimallissa pystyttiin tekemään myös hieman muutoksia aiempiin vaiheisiin, toisin kuin puhtaassa vesiputousmallissa. Systemaattinen toimintamalli projektin toteutukseen helpotti aikataulussa pysymisessä ja laajan projektin toteuttamisessa.

Työn tuloksena opittiin paljon lisää tietokannan tekemisestä, projektinhallinnasta, suunnittelumalleista ja aikataulujen noudattamisesta pitkässä projektissa. Projekti oli iso ja aikaa vievä työ kahdellekin henkilölle, varmasti osittain siksi, että aiempaa kokemusta vastaavanlaisesta projektista ei ollut.

Projektin lähtökohta ja tavoite oli suunnitella sekä toteuttaa selkeä ja helppokäyttöinen huoltotietokanta asiakasyritykselle. Asiakasyrityksessä tietokantaan tallennetaan asiakas- ja laitetietoja. Tavoitteeseen päästiin mielestämme erittäin hyvin. Tietokannan luominen olisi varmasti voinut tapahtua nopeamminkin, mutta vähäinen kokemus tietokantojen suunnittelusta ja toteutuksesta aiheutti odottamattomia ongelmia. Ongelmat saatiin kuitenkin ajan kanssa ja kirjallisuutta tutkimalla ratkaistua, ja lopputulos oli onnistunut.

Työn valmistuttua asiakasyrityksessä huomattiin, että tietokannan avulla laitteiden huoltoa pystyttiin seuraamaan tarkemmin aiempaan verrattuna. Kun ohjelmisto oli ollut hetken käytössä, huomattiin, että ohjelmiston avulla huoltoprosessista suoriuduttiin tehokkaammin.

Varsinkin vikavinkit-tila nopeuttaa huoltoprosessia, koska vikaa voidaan lähteä etsimään sellaisista kohdista joissa aiemminkin on ollut vikoja. Vikavinkit kehittyvät ja tulevat sitä paremmiksi, mitä kauemmin ohjelmisto on yrityksellä käytössä, koska tietokan-

ta täydentyy paremmaksi päivä päivältä. Vikavinkit helpottavat jokaisen huoltajan työtaakkaa, koska sieltä voidaan helposti etsiä vikakuvausta vastaavia korjauksia.

7 YHTEENVETO

Projektin tavoitteena oli luoda selkeä ja helppokäyttöinen huoltotietokanta toimeksiantajalle. Tietokannan tulisi sisältää asiakkaiden ja laitteiden tiedot. Tarve tietokannalle syntyi, kun asiakasyritys halusi siirtyä nykyaikaisempaan, sähköiseen laite- ja asiakastietojen hallinnointiin.

Työ aloitettiin haastattelemalla asiakasyrityksen edustajia. Haastattelujen avulla saatiin selkeä ja kattava käsitys siitä, mitä ohjelmiston tulisi sisältää. Haastattelujen jälkeen tehtiin karkea suunnitelma aikataulusta ja projektin vaiheista. Tämän jälkeen perehdyttiin erilaisiin ohjelmistosuunnittelumalleihin. Kun oli saatu riittävän hyvä käsitys eri malleista, päädyttiin käyttämään vesiputousmallia työn toteutukseen. Mallin pohjalta pystyttiin luomaan tarkempi suunnitelma työn toteutuksesta. Työ toteutettiin aikataulun mukaisesti ja noudattamalla suunnittelumallin vaiheita.

Tutkimusosuutta tehtäessä tutustuttiin laajalti alan kirjallisuuteen ja aihetta käsitteleviin www-dokumentteihin. Myös aiemmin aiheesta tehtyihin lopputöihin tutustuttiin ja yritettiin löytää uusia näkökulmia vanhoihin aiheisiin. Lähteinä käytettiin siis alan kirjallisuutta, www-dokumentteja ja useita opinnäytetöitä.

Tietokanta toteutettiin MS Access tietokantahallintasovelluksella. MS Access valittiin pitkälti sen pohjalta, että se oli tuttu asiakasyritykselle osana MS Office -pakettia. Sovellus oli monipuolinen työkalu, ja sen avulla tietokannan luominen onnistui ilman suurempia ongelmia. Sovelluksen valintaan vaikutti osaltaan myös se, että Microsoftin ohjelmistot olivat tuttuja aikaisemmista projekteista.

Lopputuloksena saatiin asiakasyrityksen vaatimukset täyttävä, toimiva ja riittävän yksinkertainen huoltotietokanta. Asiakasyrityksen palaute ohjelmistosta oli positiivista. Erityisesti kiitosta sai ohjelman ulkoasun selkeys ja se, että ylimääräisiä kustannuksia ei tietokannan käytöstä synny. Asiakasyritys oli tyytyväinen siihen, että tietokantaan oli saatu toteutettua heidän toiveidensa mukaiset ominaisuudet. Saadun palautteen mukaan tietokanta on helpottanut yritystä seuraamaan laitteen huoltoprosessia, joten siltäkin osalta tavoitteeseen on päästy. Tässä projektissa opittiin paljon uusia asioita sekä syvennettiin aikaisemmin oppimia tietoja tietokannoista sekä ohjelmistosuunnittelusta.

LÄHTEET

[1] Haikala, I. ja Mikkonen, T., *Ohjelmistotuotannon käytännöt*. Helsinki: Talentum, 2011

[2] McConnell, S., *Ohjelmistotuotannon hallinta*. Helsinki: IT-Press, 2002

[3] Sommerville, I., *Software engineering.* Harlow Englanti: Addison-Wesley, 2010

[4] Haikala, I. ja Märijärvi, J., *Ohjelmistotuotanto*. Helsinki: Talentum, 2006

[5] Mäkelä, T-P., Laskujenseurantasovelluksen suunnittelu ja toteutus, opinnäytetyö, Satakunnan ammattikorkeakoulu, 2011

[6] Jyväskylän yliopisto, ” Ohjelmistotuotannon prosessimallit verkkokurssin tuottamisessa”, [www-dokumentti]. Saatavilla: http://www.mit.jyu.fi/ope/kurssit/TIE358/sivusto/johdanto/ohjelmistoprosessi.html (Luettu: 11.4.2014)

[7] Kasila, E., Ketterien menetelmien hyödyntäminen sulautettujen järjestelmien kehityksessä, diplomityö, Lappeenrannan teknillinen yliopisto, 2013

[8] ”Kehittämistyön vaiheet ja elinkaarimallit”, [www-dokumentti]. Saatavilla: http://www.okol.org/verkkokurssit/datanomi/tietojarjestelmien_kaytto_ja_kehittaminen/johdatus_tietojarjestelmiin/kehittamistyon_vaiheet_ja_elikaarimallit/kehittamistyon_vaiheet_ja_elinkaarimallit_asia.htm (Luettu: 11.4.2014)

[9] Istqb exam certification, ”What is spiral model- advantages, disadvantages and when to use it”, [www-dokumentti]. Saatavilla: http://istqbexamcertification.com/what-is-spiral-model-advantages-disadvantages-and-when-to-use-it/ (Luettu: 12.4.2014)

[10] ”What Is Incremental Model In Software Engineering? It’s Advantages & Disadvantages”, [www-dokumentti]. Saatavilla: http://www.technotrice.com/incremental-model-in-software-engineering/ (Luettu: 12.4.2014)

[11] Taina Juha, ”Ohjelmistotuotannon prosessimallit”, [www-dokumentti]. Saatavilla: http://www.cs.helsinki.fi/u/taina/ohtu/s-2000/luennot/prosessi/kaikki.html (Luettu: 12.4.20014)

[12] Oulun kauppaoppilaitos, ”Johdatus tietojärjestelmiin”, [www-dokumentti]. Saatavilla: http://www.okol.org/verkkokurssit/datanomi/tietojarjestelmien_kaytto_ja_kehittaminen/johdatus_tietojarjestelmiin/kehittamistyon_vaiheet_ja_elikaarimallit/kehittamistyon_vaiheet_ja_elikaarimallit.htm (Luettu: 13.4.2014)

[13] ”Scrum”, [www-dokumentti]. Saatavilla: http://fi.wikipedia.org/wiki/Scrum (Luettu: 13.4.2014)

[14] ”Scrum”, [www-dokumentti]. Saatavilla: http://psalmel.wikispaces.com/Scrum (Luettu: 13.4.2014)

[15] Oulun ammattikorkeakoulu, ”Software Business Competence”, [www-dokumentti]. Saatavilla: http://www.oamk.fi/sbc/testaus/testaustasot.htm (Luettu: 23.4.2014)

[16] ”What is integrated testing in software testing”, [www-dokumentti]. Saatavilla: http://software-testing-tutorials-automation.blogspot.fi/2011/06/what-is-integrated-testing-in-software.html (Luettu: 25.4.2014) [blogikirjoitus]

[17] HiQ, ”Hyväksymistestaus”, [www-dokumentti]. Saatavilla: http://www.unthink.fi/Laadunvarmistus/Hyvaksymistestaus/ (Luettu: 24.4.2014)

[18] Katara Mika, ”Ohjelmistojen testaus”, [www-dokumentti]. Saatavilla: http://www.cs.tut.fi/~testaus/s2011/luennot/OHJ-3060_2011_110-170.pdf (Luettu: 24.4.2014)

[19] ”Tietokanta”, [www-dokumentti]. Saatavilla: http://fi.wikipedia.org/wiki/Tietokanta (Luettu: 29.4.2014)

[20] Kilpeläinen Ari, ”ER-mallinnus”, [www-dokumentti]. Saatavilla: http://homes.jamk.fi/~kivni/http0140/material/ermallinnus.html (Luettu: 29.4.2014)

[21] ”Entity–relationship model”, [www-dokumentti]. Saatavilla: http://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model (Luettu 29.4.2014)
[22] Wikström Yngvar, ”Tietokannat”, [www-dokumentti]. Saatavilla: https://elektroniikka.turkuamk.fi/dalton/ (Luettu: 2.5.2014)

[23] ”What is Microsoft Access”, [www-dokumentti]. Saatavilla: http://www.simply-access.com/What-Is-Microsoft-Access.html (Luettu: 26.4.2014)

[24] Turun Huoltoexpertit, [www-dokumentti]. Saatavilla: http://www.turunhuoltoexpertit.com/ (Luettu: 26.4.2014)