Tampereen yliopisto

Tuomas Laitinen

KIRJANPITO-OHJELMAN PROTOTYYPIN TOTEUTUS JA PSP

Informaatioteknologian ja viestinnän tiedekunta
Pro gradu -tutkielma
Toukokuu 2019

TIIVISTELMÄ

Tuomas Laitinen: Kirjanpito-ohjelman prototyypin toteutus ja PSP
Pro gradu -tutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Toukokuu 2019

Käyn tutkielmassani läpi eri ohjelmistotuotannon menetelmien sopivuuden kirjanpito-ohjelman prototyypin toteuttamiseen. Tutkitut menetelmät sisältävät alan tunnetuimmat ja yleisimmin käytössä olevat menetelmät. Menetelmien vertailun jälkeen asetetaan vielä henkilökohtaisen oppimisen tavoitteet prototyypin tuottamiselle.

Prototyypin toteutuksen on tarkoitus toimia pilottina tulevasta toteutuksesta ja osoittaa samalla tuottavan tahon asiantuntemus aihepiirin suhteen. Tutkielmassani puntaroidaan eri menetelmien sopivuutta myös tuottavan organisaation ominaisuuksien ja asiakkaan arvojen näkökulmasta. Keskeistä on huomata ettei yksi menetelmä ole yksiselitteisesti parempi kuin toinen. Eri menetelmien sopivuuden käyttötapaukseen ratkaisee enemmänkin tuottavan ja tilaavan osapuolen arvot ja ominaisuudet.

Toteutusmalliksi valittiin prototyyppimalli ja PSP:n tavoitteet valittiin PHP-ohjelmoinnista, arkkitehtuurista ja lain vaatimasta laskujen tietosisällöstä. Valmis prototyyppi sisältää mallin, jonka pohjalta tuleva ohjelmisto voidaan laatia.

Avainsanat: PSP, Ohjelmistotuotantomenetelmät, Prototyyppi

1 Johdanto

Kirjanpitopalvelut ovat yritykselle kuluerä ja varsinkin pienissä ja liiketoiminnaltaan vähäisissä yrityksissä kirjanpitopalvelun ostaminen voi olla liian kallista. Kirjanpitopalvelun kalleus voi olla jopa peruste yrityksen perustamatta jättämiselle. Automaattinen kirjanpito tarjoaa edullisen ja helpon ratkaisun kalliin tilitoimiston rinnalle. Se on enemmän kuin Excel tai pino kuitteja pöytälaatikossa, mutta vähemmän kuin tilitoimisto. Tällaisen ohjelman toteutuksessa tarvitaan vain muutamia toimintoja, mutta niiden avulla saadaan jo yrittäjälle tarvittavat työkalut toteutettua. Suomessa on runsaslukuinen joukko sivutoimisia pienyrittäjiä, joille tällainen ohjelma olisi tarpeen.

Tutkielmassani käsittelen tällaisen ohjelmiston prototyypin toteuttamista eri ohjelmistotuotantomenetelmien näkökulmista. Tutkielmassa tutkitaan myös Personal Software Process (PSP) näkökulmasta olemassaolevien ja uusien taitojen kehittämistä. Käsitellyt ohjelmistotuotantomenetelmät on valittu niiden tunnettavuuden perusteella.

Aiheen valintaan on vaikuttanut oma toimintani sivutoimisena yrittäjänä ja tämän toiminnan aikaansaama tarve kirjanpitotoimintojen ymmärrykselle. PSP:n näkökulma taas liittyy omaan haluuni kehittyä ohjelmistoalan ammattilaisena sekä tarkoituksesta etsiä toimivia tapoja kehittää omaa osaamistani. Tutkielmassa pyritään pitämään asiakkaan rooli keskeisessä asemassa. Ohjelman kehityksen pitää olla läpinäkyvä prosessi asiakkaalle, ja asiakkaalla pitää olla ymmärrys siitä mitenkä tuleva tietotekninen ratkaisu tulee vastaamaan hänen ongelmaansa. Asiakkaalle ei kuitenkaan voida olettaa kattavaa tietoteknistä taustaa ja tähän pitää kiinnittää huomiota prosessin kuvauksessa.

Tutkielmani tutkimusongelma on valita asiakkaan ja tuottajan tarpeet ja ominaisuudet huomioiden sopiva ohjelmistotuotantomenetelmä prototyypin toteutukseen. Menetelmää valittaessa kiinnitetään huomiota myös tulevaan mahdolliseen koko projektin toteutukseen ja käydään läpi millä muutoksin eri tuotantomallit voisivat myös sopia tuottamisen malleiksi. Toinen tutkimusongelmani on PSP-tavoitteiden asettaminen ja näiden tavoitteiden saavuttamisen toteaminen.

Luvussa 2 käyn läpi ohjelmistotuotannon menetelmiä ja menetelmien sopivuutta prototyypin toteutukseen. Menetelmistä tutkitaan millaisiin ratkaisuihin ne sopivat ja kenenkä käyttäminä. Luvussa 2 esitellään myös PSP ja asetetaan PSP:n tavoitteet. Luku 3 sisältää varsinaisen prototyypin toimeksiannon ja toimeksiannon osapuolet. Luvussa 3 valitaan myös sopiva ohjelmistotuotantomenetelmä. Luvussa 4 käydään läpi prototyypin toteutus ja PSP:n tulokset. Luku 5 sisältää tutkimuksen tulokset ja luku 6 pohdinnan.

2 Ohjelmistotuotannon menetelmät ja Personal Software Process

Ohjelmistotuotanto kattaa terminä menetelmät joiden avulla tuotetaan ohjelmistoja. Tuotettavat ohjelmistot vaihtelevat ominaisuuksiltaan ja työn määriltään suuresti. Esimerkiksi yksittäiselle kaupalle laadittavat nettisivut ovat ohjelmistotuotannon tuotos siinä missä on sairaanhoitopiirin tiedonhallintajärjestelmä, tai vaikkapa ajoneuvokeskuksen tietokanta. Projektin koon ja vaativuuden kasvaessa erilaiset tuotantoa mallintavat prosessit tulevat tärkeämmiksi.

Ohjelmistotuotannon mallit ovat eri toimintamalleja lähestyä käytännön ongelmia. On olemassa tarve, joka halutaan ratkaista. Ratkaisu voi olla esimerkiksi xml-parseri, joka käsittelee Finvoice-laskuja ja tutkii yrityksen maksuliikenteeseen liittyvää dataa. Oli ongelma mikä tahansa niin sen monimutkaistuessa struktuurinen ratkaisumalli auttaa tuotantoprosessia suuresti. On olemassa joku malli, mitä seuraamalla pääsemme valmiiseen lopputulokseen.

Käyn tässä luvussa läpi seuraavat suunnittelumallit ja niiden tunnusomaiset piirteet: vesiputousmalli, ketterät menetelmät ja prototyyppimalli. Käsittelen myös suunnittelumallien muokkaamista käytännön työelämässä.

2.1 Vesiputousmalli

Vesiputousmalli on ikoninen ”standardi” tehdä asioita ohjelmistokehityksessä. Toteutus alkaa suunnittelusta ja päättyy testauksen kautta käyttöönottoon. Malli sopii parhaiten tapaukseen, jossa kohdealue tunnetaan tarkoin jo entuudestaan ja suunnittelun aikana ei ole oletettavissa suuria yllätyksiä. Mitä pidemmälle vesiputousmalli etenee, niin sitä kalliimmaksi mahdolliset muutokset tulevat. Tästä johtuen jatkuva testaus ja asiakkaan mukanapito projektin edetessä on ensiarvoisen tärkeää. Kaikki ohjelmistotuotantomallit yhtenevät joiltain osin vesiputousmalliin, mutta ovat yleensä sitä joustavampia. Vesiputousmalli voidaan nähdä kaikkien ohjelmistotuotantomallien kuvitteellisena kantaäitinä jota ei sellaisenaan ole välttämättä edes olemassa. Vesiputousmallia ei ole mahdotonta soveltaa käytännössä, mutta se vaatii erittäin kattavan suunnittelun ja pohjatiedon sovellusalueesta. [Winston, 1970]

2.1.1 Vesiputousmallin kuvaus

Kaavamainen vesiputousmalli alkaa *tutkimuksesta/määrittelystä*. Tarkoituksena on saada aikaan kattava kuvaus tilaajan tarpeesta ja tehdä kattava kartoitus ohjelman suunnittelua varten. Tässä vaiheessa on tärkeää selvittää asiakkaan tarve ja muodostaa ymmärrys sovelluksen kuvaamasta ongelmasta. Jos määrittelyssä ja esitutkimuksessa tehdään virheellisiä arvioita, niin huonoimmassa tapauksessa valmis ohjelma ei pysty

vastaamaan alkuperäiseen asiakkaan tarpeeseen. Vaihe voidaan tiivistää kysymykseen: Mikä on ongelma? [Haikala & Märijärvi, 2006]

Alkuselvityksen jälkeen siirrytään *suunnitteluun*. Suunnittelussa laaditaan ohjelmiston rakenteen ja toiminnallisuuden kuvaus. Tämän suunnitelman täytyy pystyä kertomaan, miten asiakkaan ongelma ratkaistaan abstraktilla tasolla. Suunnitteluvaiheessa voidaan myös tarkastella edellisen vaiheen tutkimusta ja tarvittaessa palata muuttamaan sitä. Suunnitelma on hyvä käydä asiakkaan kanssa läpi ja selvittää asiakkaalle, miten ohjelma tulee käytännössä toimimaan. Tässä vaiheessa voi muodostua tarve lisätä, muuttaa tai poistaa toiminnallisuuksia. Suunnittelun tuloksena muodostetaan malli, joka vastaa kysymykseen: Miten ongelma ratkaistaan? [Haikala & Märijärvi, 2006]

Suunnitelman jälkeen siirrytään *toteutukseen*. Toteutuksessa abstrakti suunnitelma alkaa muuttua konkreettiseksi sovellukseksi. Toteutus saattaa paljastaa tarpeita muuttaa suunnitelmaa. Suunnitelmassa on saatettu jättää huomiotta toteutuksen vaativuus tai yksinkertaisesti hinta. Valmis toteutus voi myös olla liian hidas tai turhan monimutkainen toimiakseen vaaditulla tavalla. Toteutus saattaa olla myös liian vaikeasti muokattava. Näissä tapauksissa on hyvä palata suunnitteluun ja huomioida toteutuksessa ilmenneet ongelmat. Toteutuksen tukena on hyvä suorittaa tässä kohtaa testausta sitä mukaan, kun toiminnallisuutta saadaan luotua. Vaihe vastaa kysymykseen: Mitä/miten rakennetaan?

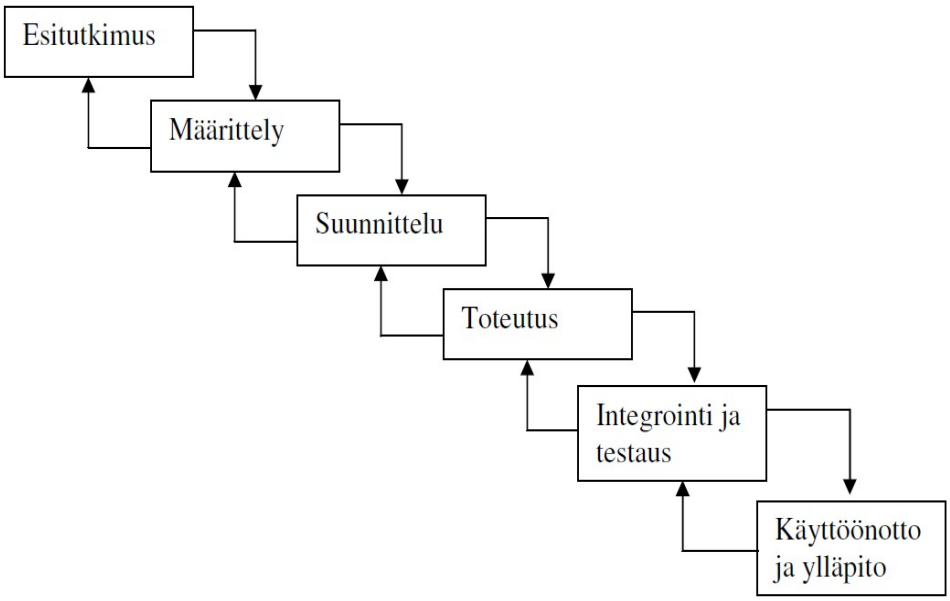
Testaus- ja integraatiovaiheessa tuotettu ohjelma tai sen osa integroidaan olemassaolevaan järjestelmään (esimerkiksi luodaan kirjanpito-ohjelmaan toiminnallisuus lähettää sähköpostitse pdf-muotoisia kuvia asiakkaalle). Tuotettu ohjelma voi olla myös itsenäinen toteutus, jolloin integraatiota ei tarvita. Testauksen toteutus on yksilöllisistä ja siinä voi olla suuriakin eroja. Lähtökohtaisesti testaus maksaa ja hyvä testaus maksaa vielä vähän enemmän. Testauksen laiminlyönti voi vaikuttaa houkuttelevalta taloudellisten säästöjen varjolla. Eri testaustavat käydään kattavasti läpi alakohdassa 2.2.1. Testausvaihe vastaa kysymykseen: Toimiiko se?

Käyttöönotossa ohjelma viedään käyttäjälle käytettäväksi. Sovellus voi toimia asiakkaan järjestelmässä tai asiakkaalla voi olla vaikkapa käyttöoikeus pilvessä olevaan kustomoituun palveluun. Käyttöönotossa asiakkaalle luovutetaan tarvittavat oikeudet/käyttäjätiedot ja mahdollisesti asennetaan ohjelma. Projektin onnistumisen kannalta asiakkaan koulutus sovelluksen käyttöön on myös ensiarvoisen tärkeää. Asiakkaalle on syytä avata kattavasti ohjelmiston toiminnallisuuksia, jotta tuotteesta saadaan mahdollisimman paljon irti. Näin voidaan varmistaa myös parantaa asiakastyytyväisyyttä.

Ylläpito voi olla toiminnallisuuksien/rakenteen muokkaamista tai sitten ihan vain käyttäjätietojen lisäystä järjestelmään. Toimittava taho voi toimia ylläpitäjän roolissa ja tarjota myös käyttötukea asiakkaalle. Ylläpidon laajuudesta päätetään jokaisen projektin osalta erikseen. Ylläpito voidaan myös suorittaa kolmannen osapuolen toimesta. Tässä

tapauksessa ohjelman hyvä dokumentointi koko tuotannon ajalta on ensiarvoisen tärkeää.

Kuvassa 1 on vesiputousmallin toiminta esitetty alaspäin virtaavana kaaviona, jossa joka vaiheesta voidaan aina palata edelliseen. Kehitys huipentuu käyttöönottoon ja ylläpitoon, jossa tuotteen ”avaimet” luovutetaan asiakkaalle ja ajo voi alkaa.

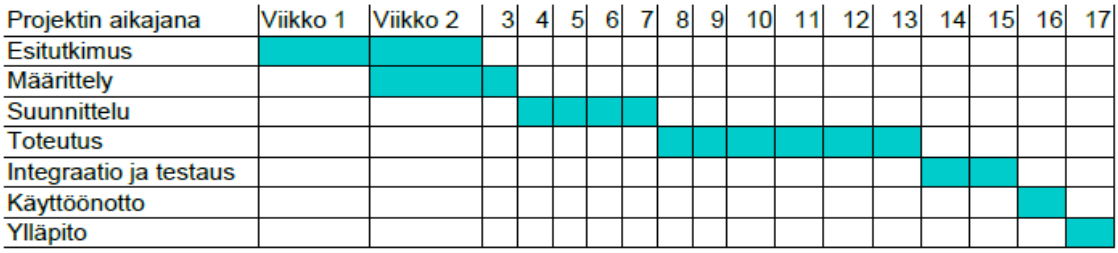


Kuva 1: Vesiputousmallin vaiheet [Haikala & Märijärvi, 2006].

2.1.2 Vesiputousmalli käytännössä

Alaspäin soljuva vesiputousmalli on ohjelmistotuotannon ihannetapaus. Sen onnistuminen vaatii kuitenkin asiakkaalta täydellisen ymmärryksen tarvittavasta tuotteesta ja toisaalta toteuttavalta osapuolelta täydellisen ymmärryksen kyseisen tuotteen suunnittelusta. Vesiputousmalli ei toimi, jos asiakas muuttaa vaatimuksiaan kesken projektin tai jos suunnittelu törmääkin toteutusongelmaan. Käyttäjän kannalta suurin huoli on ohjelman käytettävyys. Vesiputousmallissa käyttäjätestaukseen päästään mahdollisesti vasta käyttöönotossa ja vakavien käytettävyysongelmien korjaus siinä vaiheessa on erittäin ongelmallista ja kallista. Hienoinkaan tuote ei toimi käytännön maailmassa, jos asiakas ei osaa sitä käyttää.

”Firman tavalla” muokattu vesiputousmalli on kuitenkin yleinen käytäntö tuottaa ohjelmia. Tuottajalle sovitettu malli tarjoaa selkeät jaksot tuotantoon ja sillä voidaan tehdä alustava aikataulutus projektille. Ohjelmistoprojektissa on usein mukana myös henkilöitä, jotka eivät ole tekemisissä ohjelmistotuotannon kanssa. Vesiputousmalli tarjoaa hyvin jäsennellyn rakennekaavion miten tullaan etenemään. Aikajanan toteutus on esitetty Kuvassa 2. Projekti kulkee esitutkimuksesta ylläpitoon luontevien välivaiheiden kautta.



Kuva 2: Vesiputousmallin aikataulutus.

Kuvassa ilmenee selvästi missä mennään ja liiketoimintapuolen ihmisten on helppo seurata tilannetta. Vesiputousmallin mukaista rakennetta käytetään myös monessa muussakin tuotantomallissa. Esimerkiksi talot rakennetaan vesiputousmallin mukaan. Ensin on asiakkaan tarve, sitten suunnittelu, toteutus, käyttäänotto ja ylläpito. Vesiputousmallin mukainen tapa toimia on arkielämän kannalta järkevä tapa lähestyä ongelman ratkaisua.

2.2 Ketterät menetelmät

Ketterät menetelmät ovat joukko menetelmiä, joissa toimiva tuote ja suora viestintä ovat hyvin dokumentoitua ja jaksotettua tuotantotapaa tärkeämpiä. Karkeasti voidaan yleistää että ketterät menetelmät ovat vastaisku jäykälle ja byrokraattiselle vesiputousmallille. Ketteriä menetelmiä on useita, mm XP, DSDM ja Scrum. [Lindberg, 2003; Schwaber & Beedle, 2002; Stapleton, 1997] Eri metodeja yhdistää Agile Manifeston seuraavat neljä arvoa: [Beck et al., 2001]

Yksilöitä ja kanssakäymistä enemmän kuin menetelmiä ja työkaluja
Toimivaa ohjelmistoa enemmän kuin kattavaa dokumentaatiota
Asiakasyhteistyötä enemmän kuin sopimusneuvotteluja
Vastaamista muutokseen enemmän kuin pitäytymistä suunnitelmassa

Ketterillä menetelmillä on sama tavoite kuin vesiputousmallilla, saada aikaiseksi toimiva ohjelmisto, jota asiakas tarvitsee. Menetelmät ovat kuitenkin joustavampia ja dynaamisempia.

Menetelmien pragmaattisuus tulee jatkuvasta testauksesta ja vuoropuhelusta asiakkaan kanssa. Ohjelmistoprojektit voivat olla pitkiä ja niiden toimituksessa voi esiintyä viivästyksiä ja niiden kustannukset voivat kasvaa huomattavasti alkuperäistä arviointia suuremmiksi. [Vänskä, 2017] Laajojen projektien kanssa on jopa mahdollista, että asiakkaan tarve ehtii hävitä ennen kuin sitä ratkaiseva ohjelma on saatu tuotettua. Ketterissä menetelmissä ei sitouduta tiukkaan alkumäärittelyyn vaan projekti saa elää ja asiakas voi muuttaa vaatimuksiaan ja määrittelyjään. Tämä voi tietysti johtaa

merkittävästi kasvaneisiin kuluihin. Pääroolissa on toimiva tuote, joka vastaa asiakkaan käytännön tarvetta.

Miksi ketteryyttä tarvitaan ohjelmistotuotannossa? Tietotekniikka on nopeasti etenevä ala. Kehityksen tempo saattaa tehdä jo 5 vuotta vanhasta ohjelmasta auttamattomasti vanhentuneen. Uusia tekniikoita esitetään jatkuvasti ja ratkaisut täytyy sovittaa yhteen monen muun järjestelmän kanssa. Ketteryydessä haetaan nopeasti käyttöönotettavaa ratkaisua, joka vastaa asiakkaan tarvetta. Turhan jäykkä ja byrokraattinen toimintamalli voi viivästyttää kehitysprosessia liiaksi, joten sitä halutaan välttää. Ketterissä menetelmissä tuottaminen ja keskustelu asiakkaan kanssa ovat pääroolissa. Ketterät menetelmät eroavat toisistaan ja niillä on eri painopisteitä. Ne myös vaativat erilaisia toimintaympäristöjä ja erilaista osaamistasoa asiakkaalta/tuottajalta. Luottamus toteuttavaa tiimiä ja sen autonomiaa kohtaan on myös oltava kunnossa niin oman talon sisällä kuin myös asiakkaan päässä.

2.2.1 Extreme Programming (XP)

Extreme Programming tähtää ohjelmiston hyvään laatuun ja asiakkaan tarpeiden muutoksiin. XP:n keskeisiä arvoja ovat: kommunikaatio, yksinkertaisuus, palaute ja rohkeus. [Beck, 1999] Näiden lisäksi menetelmän keskeisiä käsitteitä ovat: pariohjelmointi, testauksen eri menetelmät ja refaktorointi. [Lindberg, 2003]

Refaktorointi on prosessi, jossa koodin toiminta säilyy, mutta sen rakenne muuttuu. Koodi kirjoitetaan uusiksi, mutta sen funktio pysyy ennallaan. Refaktorointia voidaan käyttää parantamaan tai selkeyttämään olemassaolevaa koodia. Refaktorointi on hyödyllinen työkalu kun alkuperäistä koodia pitää laajentaa koskemaan tapauksia, joita ei aikaisemmin otettu huomioon. Refaktoroinnn jälkeinen testaus on äärimmäisen tärkeää ohjelman toimivuuden kannalta. [Weibgerber & Diehl, 2006]

Pariohjelmointi-menetelmässä kaksi ohjelmoijaa työskentelee yhdessä. Toinen ohjelmoija koodaa ja toinen arvioi ja seuraa. Pari vaihtelee rooleja säännöllisesti. Menetelmää voidaan käyttää opetuskäytössä. [Lindberg, 2003]

Testaus on merkittävässä asemassa XP:ssä. Testaus takaa asiakkaalle toimivan ja laadukkaan lopputuotteen. [Weibgerber & Diehl, 2006] Testaustatapoja on useita. Tuottava organisaatio vai valita sopivan tavan projektin ja omien tarpeidensa mukaan.

Black box -testauksessa ohjelmisto testataan ”mustana laatikkona”. Ohjelmiston testaaminen tehdään ilman tietoa ohjelmiston sisäisestä rakenteesta. Testatuksessa tarkastellaan ohjelman toimintaa käyttäjän näkökulmasta. [Pan, 1999]

White box -testauksessa ohjelmisto testataan ”läpinäkyvänä”. Testaajalla on näkyvyys ohjelmiston rakenteeseen kooditasolla. Testaajalla on myös ymmärrys ohjelmiston toiminnasta ja siitä miten sen pitäisi toimia. [Pan, 1999]

Regressiotestauksessa ohjelmiston toiminta testataan kokonaisuutena ja sen eri osien yhteensopivuuden kiinnitetään huomiota. Jos jotain ohjelman osaa muutetaan

vaikkapa refaktoroinnissa niin refaktoroidun osan yhteensopivuus muun ohjelman kanssa voidaan testata regressiotestauksella. [Duggal & Suri, 2008]

Integraatiotestauksessa testataan rajapintojen ja ohjelmiston eri osien yhteensopivuutta. Ohjelmistoon voidaan vaikkapa lisätä uusi ominaisuus (esimerkiksi uusi tiedonsiirtotapa), jonka toiminta muun ohjelmiston kanssa voidaan testata integraatiotestauksella. [Ould & Unwin, 1986]

Käytettävyyden testaaminen - Käytettävyys yhdistelee terminä jonkin tuotteen/apuvälineen/esineen toiminnan näkyvyyttä ja johdonmukaisuutta sen käyttäjälle. Käytettävyyttä voidaan mitata käytön onnistumisena ja käyttäjän tyytyväisyytenä. [Nielsen, 1994] Käytettävyyden testaamisessa testataan miten ohjelman rakenne ja toiminta avautuu heuristisesti käyttäjälle ja miten käyttäjän intentioiden mallinnus ohjelmassa on onnistunut.

XP kehitettiin 1990-luvulla vastaamaan muuttuviin määrittelyihin ja ohjelmistojen lyheneviin elinkaariin. XP:ssä keskitytään toimivaan ja testattuun koodiin, joka ratkaisee asiakkaan ongelman. Tuotettava ratkaisu pyritään ottamaan käyttöön mahdollisimman nopeasti. Koodia tuotetaan pariohjelmoinnin menetelmällä ja koodia arvioidaan ja testataan jatkuvasti. Tuotettava ohjelmisto pyritään pitämään mahdollisimman yksinkertaisena. Näin varmistetaan koodin laatu ja se, että tuotettu ohjelmisto vastaa sille annettua tarkoitusta.

XP koostuu kuudesta kehitysvaiheesta. Kehitysvaiheet ovat: tutkimusvaihe, suunnitteluvaihe, julkaisun iteraatiot, tuotteistusvaihe, ylläpitovaihe ja päätösvaihe. [Abrahamsson, 2002]

Tutkimusvaiheessa (Exploration phase*)* asiakas määrittelee tarinoita (stories), jotka ovat mukana ensimmäisessä julkaisussa. Tarinat ovat tulevan ohjelman ominaisuuksia ja niiden pohjalta rakennetaan ohjelman prototyyppi. Tarinoita kerätään runsaasti, jotta kaikki mahdolliset ominaisuudet saadaan selville.

Suunnitteluvaiheessa (Planning phase) tarinat (stories) priorisoidaan ja niistä tehdään resurssiarviot. Priorisoinnin myötä osa tarinoista ja ominaisuuksista voidaan pudottaa pois. Tulevan ohjelman tulee olla mahdollisimman yksinkertainen nopean tuottamisen ja kustannusten kannalta. Turhia ominaisuuksia karsimalla pystytään vaikuttamaan nopeaan käyttöönottoon.

Julkaisun iteraatiot -vaiheessa (Iterations to release phase) valitut tarinat muodostavat tulevan järjestelmän arkkitehtuurin. Toteutusta aletaan rakentamaan iteraatioissa parikoodauksen menetelmällä. Testaus ja kaikkien osapuolien palaute on jatkuvasti mukana toteutuksessa.

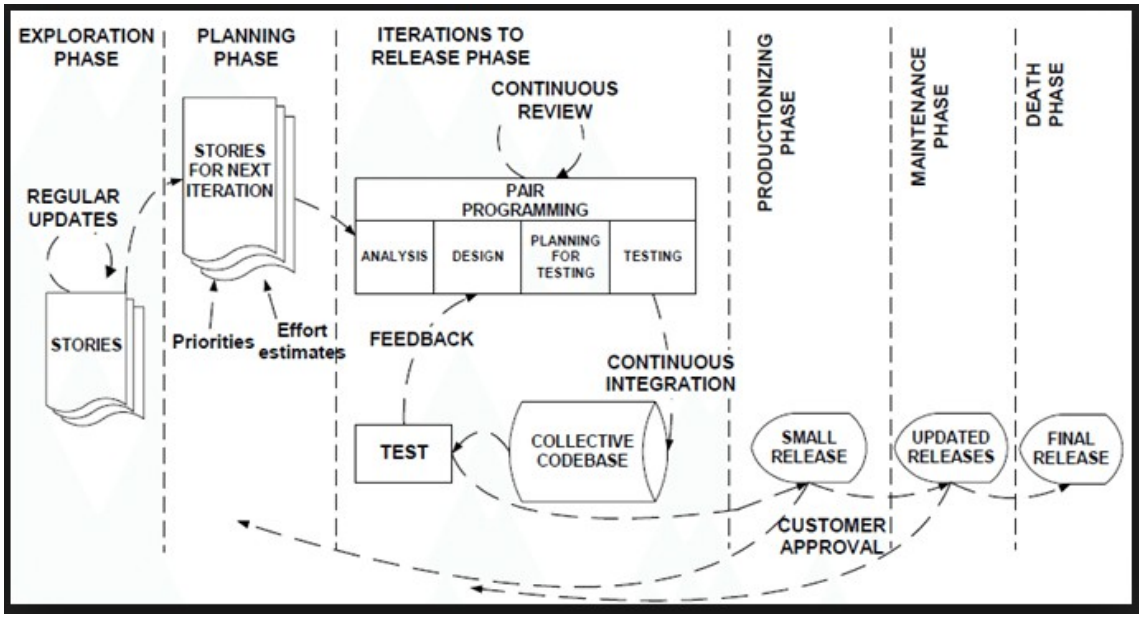
Tuotteistuksessa (Productionizing phase) ohjelmasta julkaistaan suppea versio asiakkaalle. Uusia toiminnallisuuksia voidaan lisätä yhteisellä päätöksellä asiakkaan kanssa. Vaiheen päätteeksi asiakas saa käyttöönsä toimivan version.

Ylläpidossa (Maintanence phase) ohjelmalle tarjotaan ylläpitoa ja tarvittaessa uusia ominaisuuksia voidaan lisätä toimivaan kokonaisuuteen. Toteuttavan ryhmän rakenne

voi muuttua (uusia projekteja toteuttavasta tuotantotiimistä siirrytään asiakastuen puolelle).

Päätöksessä (Death phase) ohjelma on tullut valmiiksi ja uusia ominaisuuksia ei ole enää tarve lisätä. Ohjelmasta laaditaan kattava dokumentaatio ja projekti voidaan sulkea. Tähän vaiheeseen päädytään myös jos jostain syystä projekti keskeytyy ennenaikaisesti.

Kuvassa 3 XP:n eri vaiheet on selitetty graafisesti. Kehitys kulkee lineaarisesti, mutta paluu aikaisempaan vaiheeseen on mahdollista. Tällainen tilanne voi olla jos tuotantovaiheessa tulee asiakkaalta hylkäävä päätös tai tarvitaan uusia toiminnallisuuksia. [Abrahamsson 2002]



Kuva 3: XP-menetelmän kulku [Abrahamsson et al., 2002].

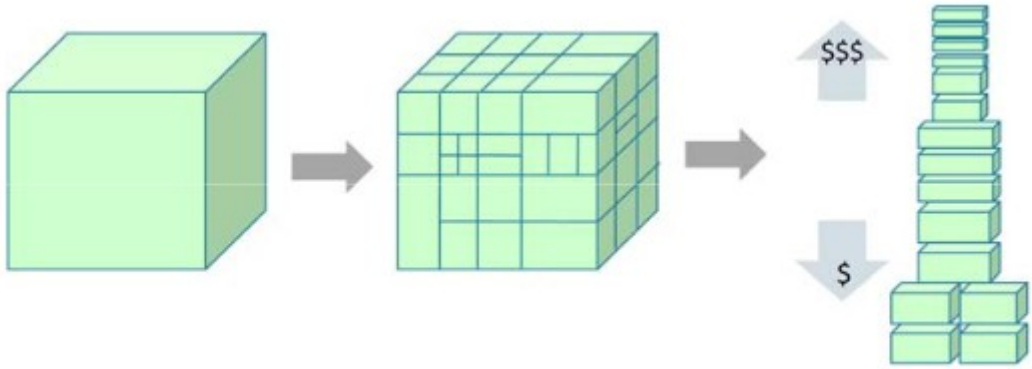
XP on kehitysmallina erittäin adaptiivinen ja dynaaminen. Se vaatii kuitenkin toteuttajiltaan ja tilaajiltaan hyvää ammattitaitoa ja osaamista. Erityisesti koodaajien on oltava senior-tason ammattilaisia. Yksittäisten tarinoiden ja niiden ominaisuuksien muutokset saattavat aiheuttaa laajempia muutoksia ja refaktorointia muussa toteutuksessa. Tämä pidentää aikatauluja ja voi lisätä projektiin käytettäviä työtunteja, joka taas näkyy asiakkaalle kasvaneena laskutuksena. XP vaatii asiakkaalta luottamusta ohjelmiston toimittajaa kohtaan, koska projektin kokonaiskustannusta on vaikea arvioida johtuen muuttuvasta työmäärästä.

2.2.2 Scrum

Scrum on ketterä ohjelmistotuotannon malli, jossa osaamiseltaan monipuolinen ryhmä tuottaa valmiin ohjelmiston alusta loppuun. Scrum ei sitoudu käyttämään yksittäisiä toteutustekniikoita, vaan tekniikat valitaan projektien ja osallistuvien henkilöiden

taitojen mukaan. Scrum on tiivistä yhteistyötä tekevä itseohjautuva joukkue, jonka jäsenet edustavat eri osaamisalueita. [Kniberg & Skarin, 2010]

Scrum-prosessin päätavoite (esimerkiksi valmis ohjelmisto) pilkotaan osatavoitteisiin (esimerkiksi tietoyhteydet, tietokanta, käyttöliittymä, toiminnallisuudet). Kokonainen projekti pilkotaan pienempiin tavoitteisiin, jotka suorittamalla saadaan aikaiseksi valmis kokonaisuus. Tavoitteita suoritetaan tärkeysjärjestyksessä ja kulloinkin suoritettavissa tavoitteessa huomioidaan tavoitteen kustannus vs hyöty. Kuvassa 4 esitetään, miten suurempi kokonaisuus jaetaan pieniin osiin ja osat arvotetaan kustannusten perusteella. Oikealla on koko projekti, joka sitten jaetaan osiin ja osia voidaan priorisoida niistä aiheutuvien kustannusten perusteella.

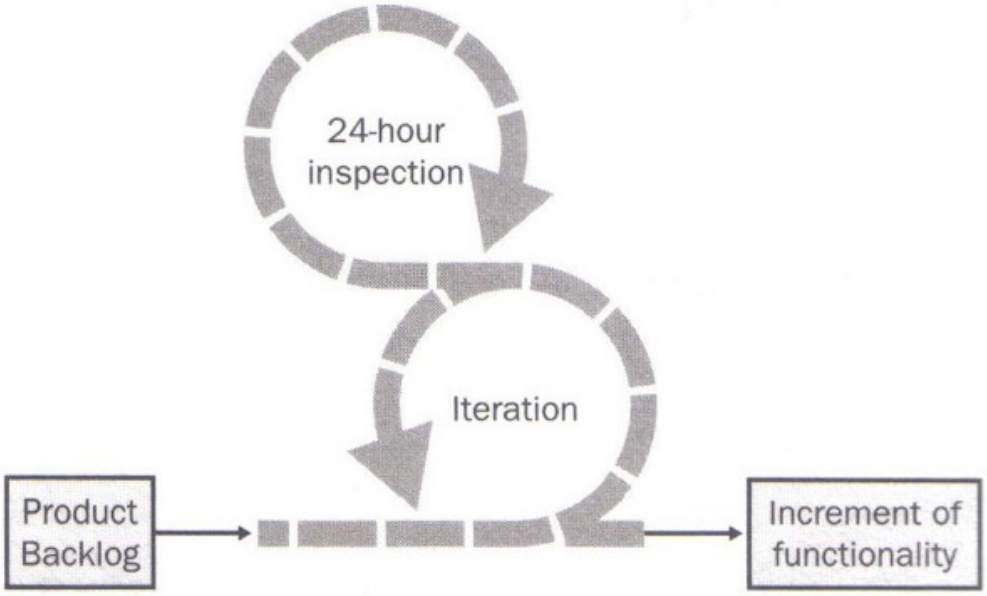


Kuva 4: Scrum tavoitteiden pilkkominen ja priorisointi [Kniberg & Skarin, 2010].

Scrum-prosessia voi soveltaa ohjelmistotuotannon ulkopuolella ja sitä voi hahmottaa esimerkiksi talon rakentamisen kautta. Valmis talo on kuvan 4 vasemmalla puolella oleva laatikko. Talo pitää sisällään erinäisiä elementtejä kuten seinät, katon, perustukset, tapetit, keittiön kalusteet, saunan ja listat. Elementit on kuvattu keskellä olevaan laatikkoon. Kaikki nämä elementit yhdessä muodostavat kokonaisen käsitteen talo. Elementit voidaan jakaa kustannus vs hyöty -suhteessa ja tätä katsotaan rakentamisvaiheessa. Oikean puoleisessa kuvan osassa palaset on jaoteltu hyödyn ja kustannuksen mukaan. Isot, tärkeät ja edulliset palaset ovat alhaalla. Pienet, kalliit ja ydinmerkityksen kannalta vähäpätöisemmät ovat ylhäällä. Talon tapauksessa isoja alhaalla olevia palikoita voisivat olla talon perustukset, seinät, katto ja vaikkapa viemäröinti. Pienempiä ja vähäpätöisempiä asioita voisi olla ylhäällä esimerkiksi tapetit, listat, sisämaalit yms. Kaikki palaset muodostavat kokonaisen talon ja taloa ei sellaisenaan olisi ilman jokaista yksittäistä elementtiä. Talon olemassaolon kannalta kuitenkin perustus lienee oleellisempi asia kuin makuuhuoneen kattolistat. Scrum-toteutuksessa pyritään saamaan aikaan toimiva käyttökelpoinen tuote. Tässä viitekehyksessä on tärkeää tehdä ensisijaisesti kantava runko ja vasta sen jälkeen lisätä hienot yksityiskohdat.

Scrum-prosessilla voidaan rakentaa ydinvoimala tai voileipä. Tärkeää on pystyä jakamaan isompi käsite pienempiin osiin ja nähdä se polku, mikä määrää osatavoitteiden toteuttamisen järjestyksen. Talon rakentamisen tapauksessa tietty järjestys asioiden toteuttamisessa on ilmeistä. Ei ole kovin mielekästä tapetoida ensin kipsilevyjä ja asentaa levyjä vasta sen jälkeen seiniin. Tai rakentaa kattoa ennen perustuksia. Huonolla rakennusjärjestyksellä saadaan aikaan vain Olkiluoto 3 -ydinvoimalan tapaisia rakennushankkeita.

Scrum-prosessi alkaa visiointivaiheessa. Visiointivaiheessa synnytetään tuotteen työlista (Product Backlog). Tuotteet työlista voi olla vapaamuotoinen tai formaali. Työlista on jäsennelty kuvaus asiakkaan toiveesta, joka on vielä toistaiseksi abstrakti. Työlistalla voisi olla seuraavia vaatimuksia: rakennettavan sivuston tulee pystyä palvelemaan kerralla 10 000 käyttäjää tai että sivuston käyttöliittymän pitää olla helppokäyttöinen. Tuotteen työlistaa toteutetaan sprintin tehtävälistojen kautta (Sprint Backlog) sprinteissä. Kuvassa 5 esitetään scrum-prosessi ja sprinttien rooli siinä.

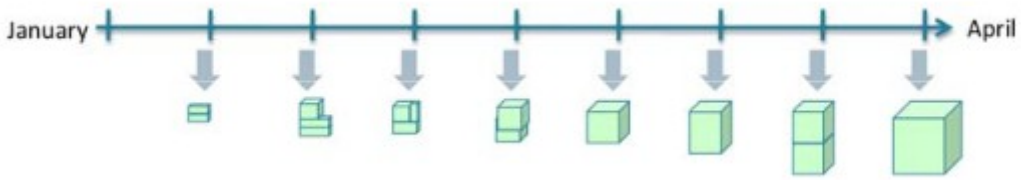


Kuva 5: Scrum-prosessi [Schwaber, 2004].

Sprintti

Scrumissa kehitystyö on jaettu ajallisesti sprintteihin. Scrum-prosessin tavoitteet suoritetaan yksittäisissä sprinteissä. Sprinttien tehtävälistalle valitaan ajankohtaisia toteuttavia ominaisuuksia tuotteen työlistalta. Sprintit ovat 1-4 viikon mittaisia jaksoja, joista jokainen tähtää tuoteversion (yksittäinen elementti) julkaisuun. Sprintti koostuu alkupalaverista, toteutuksesta ja arvioinnista. Sprinttien tavoitteet sovitaan palaverissa sprintin alussa. Sprintin kuluessa sovittu tavoite pyritään luonnollisesti saavuttamaan. Edellisen kappaleen talo-analogiaa käyttäen yksittäinen sprintti voisi olla vaikkapa

perustusten valaminen. Sprinteissä toteutetaan yksittäisiä elementtejä kasaten toiminnallisuutta toiminnallisuuden päälle. Kustannus vs. hyöty -ajattelun mukaisesti ensin tehdään tärkeimpiä palasia ja loppua kohden keskitytään enemmän pieniin ja kalliisiin tavoitteisiin. Projektin kuluessa tuote tulee aina valmiimmaksi ja koska valmistettavat elementit pyrkivät olemaan käyttövalmiita niin rakennettava tuote on koko prosessin ajan jossain määrin käyttökelpoinen. Sprintin lopuksi käydään katselmus, jossa tavoitteet ja niiden toteutuminen arvioidaan. Talon tapauksessa arvioinnissa voitaisiin testattaisiin anturan kovettuminen ja valetun betonin kovuus. Kuvassa 6 esitetään miten projektin edetessä yksittäisistä palikoista alkaa muodostua suurempi valmis kokonaisuus.



Kuva 6: Ohjelmiston laajeneminen Scrumissa [Kniberg & Skarin, 2010].

Scrum ohjelmistokehityksessä projektiin osallistuu yksi tai useampi scrum-tiimi. Tiimit toteuttavat sprintissa prosessin tavoitteita ja rakentavat elementeistä valmista taloa pala kerrallaan. Tiimit koostuvat tuotteen omistajasta (Product Owner), kehitystiimistä (Development Team) ja scrummasterista (Scrum Master). Itseohjautuvat tiimit koostuvat monitaitoisista jäsenistä ja ne saavat päättää omista työskentelytavoistaan. Jäsenten määrää ei ole vakioitua ja tiimien koko voi vaihdella. Tiimien itseohjautuvuus voi kuitenkin kärsiä jäsenmäärän kasvaessa liian suureksi.

Product Owner on yksittäinen henkilö. Hän ymmärtää tuotteen liiketoimintaa ja edustaa asiakasta. Product Owner on kuitenkin projektia tuottavan tahon jäsen. Yleisesti ohjelmistokehityksessä hänen roolinsa on lähellä projektipäällikön roolia. Hän on jatkuvasti yhteydessä asiakkaaseen ja varmistaa, että asiakkaan toiveet välittyvät muulle toteuttavalle tiimille. Toisaalta hän on myös vastuussa siitä, että asiakas ymmärtää toteuttavan tiimin toiveet ja tavoitteet. Product owner valvoo sekä tuotteen tilaajan ja toteuttajan vaatimuksia ja toimii keskustelevana tahona näiden välillä. Tuotteen omistaja vastaa tuotteen kehitysjonosta.

Scrum Master on yksittäinen henkilö. Hän on tiimin esimies, mutta kuitenkin palvelija. Scrum Master ei ole työnjohtaja, hänen tehtävään on varmistaa, että toteuttava tiimi saa tehdä työnsä rauhassa. Master varmistaa, että tiimillä on käytössään tarvittavat resurssit ja että tiimi noudattaa scrumin sääntöjä. Hän myös ohjaa ja valmentaa tiimiä.

Kehitystiimi (Development Team) on ammattilaisista koostuva tiimi. Tiimin jäsenten osaamisalueet voivat edustaa eri osa-alueita ja tiimin jäsenten tekninen

osaaminen voi vaihdella. Samassa tiimissä voi olla senior tason osaajia, kuin myös junior- tai trainee -statuksella olevia jäseniä. Kehitystiimi suorittaa varsinaisen konkreettisen tekemisen, ohjelmistotuotannon tapauksessa ohjelmoinnin. Kehitystiimi työskentelee joukkueena ja jäsenet tukevat toistensa työtä. Ihanteellista olisi, että jäsenet oppivat työskentelyssä toisiltaan ja tiimiin kuuluminen kehittäisi jäsenten ammattitaitoa. Scrum-tiimiin kuulumista voidaan käyttää ohjelmistotalon sisäisenä koulutusmetodina, jossa traineet ja seniorit työskentelevät saman projektin parissa. Työn ohessa senior-jäsenet voivat opettaa muita suoraan sekä hiljaisen tiedon välittämisen kautta.

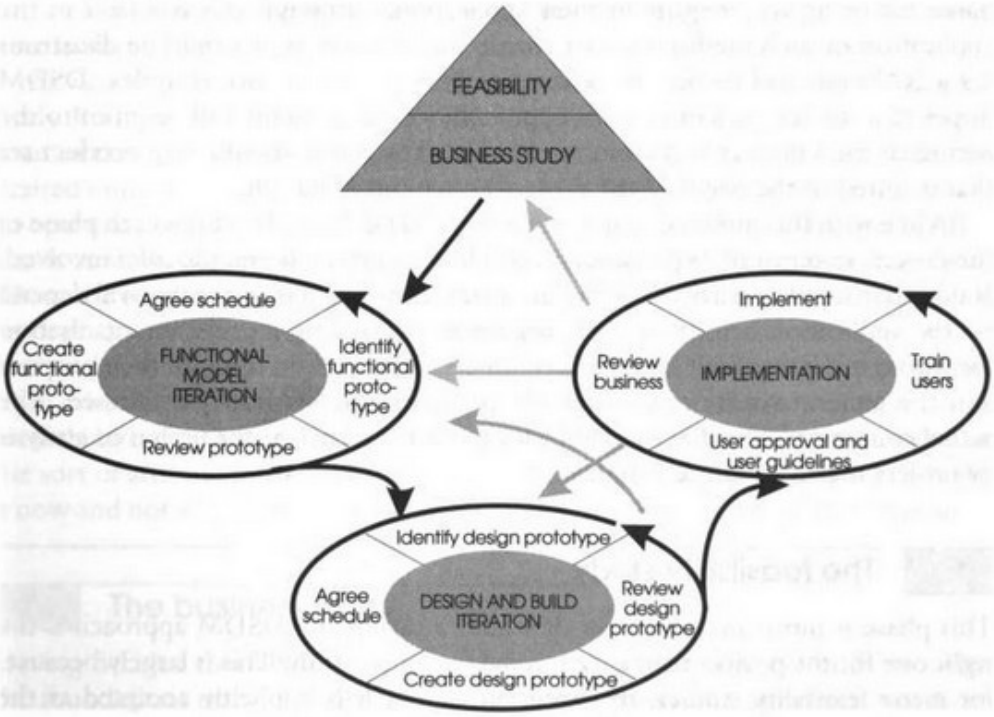
Scrum on intuitiivinen tuotantomalli, joka soveltuu käytettäväksi useille eri aloille. Scrumissa voi työskennellä kokemukseltaan monia eritasoisia henkilöitä (toisin kuin vaikkapa XP:n toteuttavassa tiimissä). Scrum vaatii toimiakseen luottamusta scrum-tiimejä kohtaan ja tuotantotiimien työrauhaa. Menetelmän etuna on sen joustavuus mutta riskinä on tuotannon hyvä järjestys. Oikeita asioita pitää tehdä oikeassa kohdassa ja tuotettavien palasten pitää sopia yhteen. Jos sprinttien tuottamien palasten integroinnissa ilmenee vaikeuksia, niin koodia saatetaan joutua refaktoroimaan laajasti, mikä on omiaan lisäämään kustannuksia ja venyttämään aikataulua. Ehdottomana etuna on kuitenkin dialogi tuottajan ja tilaajan välillä. Samoin tuotteen jatkuva esittely. Tilaaja saa näin nähdä jatkuvasti mitä ollaan tekemässä, ja jos valmistuva tuote ei vastaakaan odotuksia, niin prosessia voidaan muuttaa suotuisampaan suuntaan. Ohjelmia ei tehdä tyhjiössä vaan ohjelmat ovat suuresti riippuvaisia toisista ohjelmistoista. Scrum-prosessi mukautuu joustavana myös muuttuvaan ympäristöön. Pitempien, jopa vuosia kestävien projektien kanssa, voi ilmetä tarvetta muuttaa alkuperäisen vision tavoitteita. Scrum-prosessilla on mahdollista valmistaa ajankohtainen tuote, vaikka alkuperäinen suunnitelma ei enää olisikaan ajankohtainen.

2.2.3 Dynamic Systems Development Method (DSDM)

Dynamic Systems Development Method (DSDM) on ketterä ohjelmistotuotantomenetelmä, jossa huomio suunnataan liiketoiminnan vaatimusten palvelemiseen. DSDM-menetelmää voidaan esimerkiksi käyttää, kun on tarve luoda ripeästi liiketoimintaa palveleva ratkaisu uuden markkinatarpeen ilmetessä. DSDM-menetelmän yhdeksän periaatetta ovat: loppukäyttäjien osallisuus, työryhmän autonomia, näkyvät tulokset, toimitusten tarkoituksenmukaisuus, iteratiivisuus ja käytännön ratkaisut, palautuspisteet, kokonaiskuva, testaus toteutusvaiheessa ja yhteistyö. Periaatteista tärkein on loppukäyttäjien aktiivinen osallistuminen projektiin. Liiketoiminnan vaatimusten ja tietämyksen on siirryttävä esteettömästi ja nopeasti kehittäjille. Projekti tehdään aina loppukäyttäjiä varten, ja jos projekti ei kuvaa käyttäjien tarvetta, niin ratkaisu on epäonnistunut.

DSDM-menetelmän prosessikaavio etenee kuvan 7 mukaisesti. Vahvat mustat nuolet ovat prosessin kulun pääsuunta. Pienemmät harmaat ovat mahdollisuuksia palata

aiempaan vaiheeseen. Yksinkertaisessa tapauksessa projekti alkaa alkumäärittelystä ja päättyy implementoinnin kautta käyttöönottoon suoraviivaisesti. Prosessi ”kiertää” niin kauan kunnes asiakkaalle on toimitettu liiketoiminnan tarvetta hyväksyttävästi vastaava tuote.



Kuva 7: DSDM-menetelmän prosessi [Stapleton, 1997].

Projektin lähtökohta on kuvassa 7 kolmiossa liiketoiminnan määrittely (Business Study) ja toteutettavuus (Feasability). Liiketoiminta on kaiken lähtökohta. Se on tarve ja ”ongelma” mitä pyritään ratkaisemaan. Ongelman tekninen ratkaisu laaditaan liiketoiminnan tukemisen ehdoilla. Loppukäyttäjät ja sovellusalue ovat näin päärooolissa jo heti ensimmäisessä suunnitteluvaiheessa. Alkuvaiheessa projektille määritellään prosessin ja tietosisällön kuvaus. Tämän jälkeen siirrytään seuraavaan ympyrään.

Funktional Model Iteration -vaiheessa ympyrässä määritellään edellisen vaiheen kuvaukset tarkemmin tietotekniseltä kannalta. Tämän jälkeen toteutetaan toimiva prototyyppi kuvausten pohjalta. Prototyyppi sisältää tulevan järjestelmän testatut päätoiminnallisuudet. Prototyypistä kerätään käyttäjäkommentteja ja loppukäyttäjiä pidetään mukana kehitystyössä. Vaiheen lopussa kehittäjätiimi on tuottanut asiakkaan hyväksymän prototyypin tulevasta ratkaisusta ja kehitys siirtyy seuraavaan ympyrään.

Design and Build Iteration -vaiheessa prototyyppi kehitetään loppukäytön vaatimalle tasolle. Lopputuloksena on testattu ja valmis itsenäisesti toimiva tuote.

Implementation-vaiheessa rakennettava tuote toimitetaan asiakkaalle ja saatetaan käyttöön. Implementoinnin oleellisena osana on käyttäjien koulutus. Asiakas suorittaa tuotteen testausta vielä omassa käyttöympäristössään. Implementointivaiheessa voidaan

todeta, että asiakkaan liiketoiminnan mallin kuvauksessa on ollut puutteita tai uusia tarpeita on ilmennyt kehityksen myötä. Tässä tapauksessa prosessi voi palata mihin tahansa edelliseen vaiheeseen mukaanlukien alkumäärittelyt.

Roolit projektissa ovat seuraavat:

Senior User	-	Varmistaa kaikkien käyttäjäprofiilien mukanaolon toteutusta suunniteltaessa.
Ambassador User	-	Liiketoiminnan ymmärrys projektissa.
Technical Co-ordinator	-	Suunnittelee arkkitehtuurin ja varmistaa järjestelmän yhteensopivuuden muun ympäristön kanssa.
Tester	-	Testaaja, suorittaa jatkuvaa teknistä -ja käyttäjätestausta prosessin kaikissa vaiheissa.
Developer	-	Toteutusta tuottava rooli. Esimerkiksi ohjelmoija tai analyytikko.

DSDM-menetelmä keskittyy liiketoimintakeskeisen ratkaisun luomiseen. Ratkaisussa laadukas testaus ja loppukäyttäjien mukanaolo prosessin kaikissa vaiheissa on välttämätöntä. Tarkoituksena on varmistaa että ratkaisu toimii niin teknisesti kuin käytön kannalta oikein. Tilaava asiakas ja loppukäyttäjät näkevät prototyyppejä ja pääsevät heti toteutamaan niiden toiminnan. Prosessi voi palata edelliseen vaiheeseen jos ratkaisu ei toimi käytännössä. Tavoite on, että seuraavaan vaiheeseen siirrytään vain jos nykyinen on hyväksytysti saatettu loppuun. Prosessissa ei saa välttää refaktorointia tai paluuta edelliseen vaiheeseen jos niille ilmenee tarvetta. Virheet tai huono käytettävyys kumuloituvat prosessin edetessä ja korjauksesta tulee aina kalliimpaa ja hankalampaa. Huonoimmassa mahdollisessa tapauksessa toimitettu ohjelmisto ei vastaa asiakkaan käyttötarvetta. Eri roolien läsnäolo ja kunnioitus kehitystyössä on prosessin toiminnan edellytys.

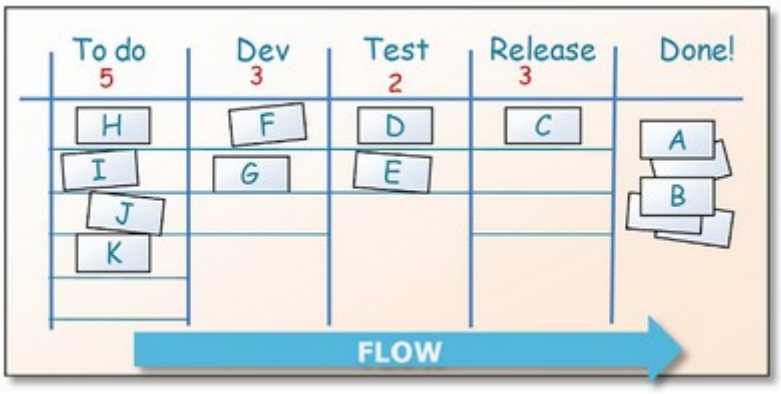
2.3 Muita ohjelmistotuotantomalleja

Tutkielmassa käydään läpi vain muutamia tunnetuimpia ohjelmistotuotantomalleja. Valittujen mallien ulkopuolelle jää suuri osa erilaisia tuotantomalleja. Seuraavaksi tarkastelen muutamia pienempiä malleja/toimintatapoja, joita voidaan käyttää pienissä toteutuksissa tuotantomalleina tai osana jotain toista mallia.

2.3.1 Kanban board

Kanban on Toyotalla kehitetty toiminnanohjausmenetelmä, joka on käytössä laajasti eri teollisuuden aloilla. Kanban on osa Lean-ohjelmistotuotantomenetelmää.

Kanban on taulu, jossa on pystyrivillä prosessin vaihe ja vaakasarake muodostuu tekijöistä (projektin potentiaaliset työntekijät). Tehtävät (taskit) ovat korteilla. Kortteja liikutellaan taulussa projektin edetessä. Kulkusuunta on vasemmalta oikealle. Kuvassa 8 on kanban taulu ja taskien eri vaiheet.



Kuva 8: Kanban-taulu [Kniberg & Skarin, 2010].

Kanbania voi soveltaa tekniseen asiakastukeen, mutta se sopii ihan yhtä hyvin vaikkapa perheen sisäisten kotitehtävien suorittamiseen. Tauluja voi olla käytössä useita. Esimerkiksi perheellä voisi olla päivittäinen taulu, jossa on tekijöinä perheenjäsenet ja tehtävinä päivittäiset askareet: koiran lenkitykset, roskien vienti, tiskien tiskaus ja ruoanlaitto.

Kanban-taululle sijoitetaan tehtäviä ensimmäiseen sarakkeeseen, jossa tehtävälle ei ole vielä määritetty tekijää. Tätä voidaan kutsua tehtäväpooliksi (task pool). Tehtäväpooliin kerätään tehtäviä sitä mukaan kun tehtäviä havaitaan. Esimerkiksi tehtävä voisi olla seuraava: Tekninen asiakastuki saa pyynnön muuttaa Finvoice-laskun sisältöä yritykselle. Kuvitteellinen tehtävä on muuttaa viestin enkoodaus UTF-8 → UTF-16. Tehtävän tultua teknisen tuen tietoon se sijoitetaan tehtäväpooliin. Teknisen tuen työntekijät poimivat tehtäviä itselleen sitä mukaa kun heillä on aikaa työstää uusia tehtäviä. Seuraavassa vaiheessa työntekijä ottaa tehtävän itselleen ja sijoittaa sen kanban-taululla seuraavaan sarakkeeseen nimensä alla. Kuvan taulussa tämä on Dev (development) sarake. Sarakkeiden määrä ja niiden merkitykset vaihtelevat eri ympäristöissä ja tuotantoprosesseissa. Kuvan sarakkeet ovat kehitys, testaus, julkaisu ja valmis. Kehityksen jälkeen tehtävä siirtyy siis testaukseen ja julkaisuun. Näiden jälkeen on vuorossa valmis-sarake, jonka jälkeen taski poistetaan taululta. Valmiit tehtävät voidaan arkistoida dokumentointia varten.

Kanban-taulu on visuaalinen ja intuitiivinen tapa hahmottaa prosessin eri vaiheita. Se rakentuu myös luottamukselle ja läpinäkyvyydelle. Taulun ollessa käytössä muut näkevät mitä henkilöllä on työn alla ja miten tehtävät etenevät. Kanbanin itseohjautuva käyttö vaatii käyttäjiltään oma-aloitteisuutta. Uudet tehtävät täytyy saada työn alle

järkevässä ajassa ja niiden pitää edetä järkevässä aikataulussa. Taulu on hyödytön jos tehtäväpooli vain kasvaa ja allokoidut tehtävät eivät etene valmiiksi asti.

Kanban-taulun muoto, sisältö ja rakenne ovat vapaasti muokattavissa. Mukautuvaisuutensa puolesta se sopii niin henkilökohtaisen elämän hallintaan kuin myös suuriin teollisiin projekteihin. Perusidea on kerätä tehtäviä tehtäväpooliin ja siirtää niitä eri järjestysten kautta valmiiksi.

2.3.2 Prototyyppimalli

Prototyyppimallissa tuotetaan nimensä mukaisesti ensimmäiseksi prototyyppi tulevasta sovelluksesta. Prototyyppi tarjoaa asiakkaalle mahdollisuuden tutustua ratkaisuun ja tehdä sen pohjalta ratkaisuja. Prototyyppi voidaan tuottaa millä tuotantomenetelmällä hyvänsä. Asiakas esittää tarpeen, jonka pohjalta tuotetaan ongelmaa ratkaiseva prototyyppi. Protyyppiä voidaan parannella iteraatiomaisissa vaiheissa niin kauan kunnes toiminnallisuus on hyväksytty asiakkaan toimesta. Toimintamalli pitää sisällään samankaltaisuuksia ketterän DSDM-menetelmän ja XP-menetelmän kanssa. [Haikala & Märijärvi, 2006]

Varsinaista toteutusta aletaan rakentamaan kun asiakas on antanut prototyypille hyväksynnän. Prototyyppi voisi olla vaikka verkkokaupan nettisivut. Tilaavalle asiakkaalle esitetään suunnitelmia ulkoasusta ja sivuston toiminnallisuutta. Esimerkiksi: Milloin tuote siirtyy ostoskoriin ja miten ostoskorista pääsee pois. Asiakkaan täytyy kuitenkin ymmärtää että prototyyppin on malli, eikä lopullinen tuote. Prototyyppiä ei voi hioa liian pitkälle vaan sen on lähinnä tarkoitus esittää toteuttavan ratkaisun idea yleisellä tasolla. Prototyypin tuotantotapaan ei myöskään sovi kiinnittää liikaa huomiota. Verkkokauppaa voidaan demota paperilapuilla ja piirtotaululla. Se ei silti tarkoita, että lopullinen tuote olisi teknisesti mitään sanomaton. [Haikala & Märijärvi, 2006]

Prototyyppimalli on hyvä tilanteessa, jossa tuottava taho toteuttaa prototyypin nopeasti ja prototyypin pohjalta pystytään antamaan alustava hinta-arvio ratkaisusta asiakkaalle. Prototyyppimallia voidaan käyttää esimerkiksi osana sopimusneuvottelua tai projektin ensivaiheena vaatimusten ja asiakkaan käyttötarpeen määrittelyssä.

Prototyyppimallin avulla voidaan myös toteuttaa kokonaisia ohjelmistoja iteraatiokierroksissa. Jokaisessa iteraatiokierroksessa käydään läpi valittua ohjelman osa-aluetta (käyttöliittymä, tietokanta, haku, tietoyhteys, arkkitehtuuri). Iteraation päätteeksi tuotos esitetään asiakkaalle ja asiakas hyväksyy tai hylkää prototyypin. Jos prototyyppi (esimerkiksi käyttöliittymä) hyväksytään, niin se siirretään kehitysversioon ja siirrytään seuraavan osa-alueen kimppuun. [Huttunen, 2006]

2.3.3 Lean

Lean-ohjelmistokehitys on tuotannon yksinkertaisuuteen ja tehokkuuteen pyrkivä filosofia. Kaikki tuottamaton toiminta pyritään poistamaan prosessista ja tuottavaa toimintaa pyritään alati tehostamaan. Toiminta on ratkaisukeskeistä ja käytännön läheistä. [Poppendieck & Poppendieck, 2006]

Lean-filosofia voidaan nähdä vastakohtana massatuotannolle. Perinteiseen massatuotantoon verrattuna Lean käyttää vain puolet työvoimasta, puolet investonneista, puolet tuotantotiloista, puolet kehityskustannuksista ja puolet ajasta. [Womack et al., 1990]. Edelliset lupaukset tuotannon kulujen karsimisesta on suunnattu perinteiselle teollisuudelle (etupäässä autoteollisuudelle), mutta samat periaatteet sopivat toki ohjelmistotuotannollekin. Siinä missä massatuotanto pyrkii valmistamaan 10 000 kuorma-autoa jotka valmistaja haluaa asiakkaille tehdä, niin Lean-filosofia pyrkisi valmistamaan niin monta kuorma-autoa kun asiakkaat haluavat ostaa (ja kuorma-autot ovat toki sellasia joita asiakkaat haluavat ostaa). Leanissa tarve tulee asiakkaalta samoin kuin valmiin tuotteen hinta. Ohjelmistoalalla eroa voisi verrata ison ja byrokraattisen firman sanelupolitiikan ja pienen asiakasta kuuntelevan toimivan tuottajan välillä.

Leanissa tehdään oikeita asioita oikeaan aikaan ja minimoidaan kaikki turha. [Abrahamsson et al., 2002] Lean-periaatteet ovat abstrakteja ja yleismaallisia, joten niiden tarkkaa mittaamista on hankala suorittaa. Koko filosofian voisi tiivistää siihen että tehdään asioita järkevästi.

2.3.4 Mukautetut tuotantomallit

Eri tuotantomalleilla on ”puhtaat ideaalit” toteutusmuotonsa. Se miten mallia sovelletaan käytännössä on täysin soveltavasta tahosta kiinni. Eri tuotantomallit kantavat myös tiettyä kuvaa työyhteisöstä. Harva ohjelmistofirma markkinoi käyttävänsä vesiputousmallia ja toisaalta taas moni yritys markkinoi käyttävänsä ohjelmistoprojekteissa ketteriä menetelmiä. Ketterät menetelmät ovat joustava ja moderni tapa tehdä asioita.

Ketteryys on käsite, jonka yritykset mielellään liittävät itseensä. On sitten toinen asia mitä ohjelmistotuotantotapaa yrityksessä sovelletaan ja kuinka ketterää toiminta oikeasti on. Ketterä ohjelmistotuotanto voi parhaimmillaan esiintyä osana vesiputousmallia. Sama yritys voi myös soveltaa pienien projektien kohdalla ketteriä menetelmiä (teknisen tuen kanban-taulu) ja suurempien projektien kohdalla vesiputousmallia. Projektien pitää olla hahmotettavissa myös yritysten johdolle ja liiketoimintapuolen ihmisille. Mitä suurempi projekti on kyseessä ja mitä enemmän se maksaa, niin sitä selkeämpi projektin kuvauksen ja rakenteen tulee olla. Kaikista selkein

kuvaus projektista on mahdollista tehdä vesiputousmallilla. Projektia seuraavilla tahoilla on kiinnostuksia myös ajankäyttöön ja kustannuksiin.

Ketterän ohjelmistotuotannon heikkoutena on sen vaiheiden hahmotus projektin ulkopuolisille jäsenille ja kustannusten seuraaminen. Toteutus vaatii myös tiimiltä itseohjautuvuutta, ammattitaitoa ja kokemusta sovelluskentästä. Ketterästi tuotettu projekti voi näyttäytyä pahimmillaan avoimena shekkinä jonka laskutus ei lopu koskaan. Toisaalta taas vesiputousmallilla saadaan kerralla laadittua alustava aikataulu ja työmäärä projektille. Vesiputousmalli vastaa erinomaisesti kysymyksiin: milloin on valmista ja mitä tämä tulee maksamaan?

Vesiputousmalli voi toimia projektin taustalla aikataulutuksessa ja eri vaiheiden kuvauksessa. Itse tuotanto voi sitten olla ainakin osittain ketterästi tuotettu.
Ketterät menetelmät ovat yksilöitä ja brändättyjä. Esimerkiksi tuotantomallissa XP pariohjelmointi on tärkeässä roolissa. Voisi hyvin kuvitella, että joku taho haluaa hyödyntää XP-tuotantomallia, mutta ilman pariohjelmointia. Samaten projektin kaaviossa voisi olla, että tuotantovaiheessa suoritetaan lopullinen julkaisu, jonka jälkeen siirrytään suoraan ylläpitovaiheeseen. Käytetty ohjelmistotuotantomenetelmä ei olisi XP, mutta se olisi ketterä menetelmä. Sitä voisi kutsua vain agileksi.

Jokaisella yrityksellä on omat toimintapansa ja tuotantomallinsa. Ne pohjautuvat johonkin tuotantomalliin, mutta harvemmin toteuttavat sen täysin. Saman yrityksen sisällä voi olla myös useita tuotantotapoja ja toimintamalleja. Ketterä tuotantotapa olisi hyvä nähdä enemmänkin suhtautumisena tuottamiseen kuin tiukkana määrittelynä.

Vaikka tuotantotapa olisi pitkälti vesiputousmallin mukaista niin seuraamalla edes osittain ylläolevia periaatteita siitä saadaan agilea (tai ainakin agilemaista). Niin hyvässä kuin pahassakin, jos ei muuten niin mainokseksi yrityksen työpaikkailmoitukseen.

2.4 Personal Software Process (PSP)

Personal Software Process (PSP) -mallin idea on selvittää ohjelmoijalle itselleen hänen käyttämänsä metodit ja työtavat. Selvittämisen jälkeen tapoja voidaan alkaa kehittämään ja niistä voidaan muokata tehokkaampia järjestelmällisyyden ja mallien avulla. Prosessissa voidaan käyttää hyväksi esimerkiksi ohjelmoinnissa käytettävää aikaa ja koodissa ilmenneitä vikoja. Valittujen muuttujien perusteella muodostetaan data, joka toimii kehityksen pohjana. PSP:n vaiheet ovat: PSP0, PSP0.1, PSP1, PSP1.1, PSP2 ja PSP2.1. [Pomeroy-Huff et al., 2009]

PSP prosessissa lähdetään liikkeelle tasolta PSP0. Tämä on prosessin käyttäjän tämänhetkinen taso. PSP0 ja PSP0.1 -vaiheissa käyttäjä laatii itselleen Personal Process Improvement Plan:n (PIP). Henkilökohtaisessa kehityssuunnitelmassa otetaan käyttöön koodausstandardi ja määritetään kehittymisen tavoite ja tavoitteen mittaustapa. Tässä vaiheessa kerätään kehittymisen seurannassa käytettävä data.

PSP1 ja PSP1.1 -vaiheissa keskitytään aikataulujen laatimiseen ja niiden pitävyyden testaamiseen. Verrokkina käytetään edellisiä vastaavia projekteja/toteutuksia. Tarkoituksena on kehittää ajankäytön arviointia ja tehostaa nykyistä ajankäyttöä.

PSP2 ja PSP2.1 -vaiheissa lisätään rakenteen ja koodin arviointi. Tutkitaan havaittujen virheiden ja heikkouksien laatua ja määrää. Puutteiden korjaamiseen käytettyä aikaa seurataan myös.

Prosessin tarkoituksena on kehittää ja verrata yksilöiden/ryhmien työskentelytapoja ja malleja. Mittaus perustuu kerättävään dataan ja sitä hyödyntäviin mittareihin, joten datan oikeellisuus ja laatu ovat onnistumisen ehdottomat edellytykset. Mallissa on tärkeässä osassa luottamus siitä, että tuloksia ja kerättyä dataa käytetään oikein ja niissä mahdollisesti ilmenevät negatiiviset puolet eivät vaikuta yksilöiden asemaan työyhteisössä. Jos osallistuvat yksilöt pelkäävät kerätä itsestään totuudenmukaista dataa, niin prosessi ei toimi. Yksittäiselle kehittäjälle voisi tulla kiusausta esimerkiksi kaunistella ohjelmassa olevien virheiden määrää ja niiden korjaamiseen käytettyä aikaa tarkoituksenaan saada oma datansa näyttämään paremmalta. [Pomeroy-Huff et al., 2009]

3 Tapaustutkimus pienyrityksen ohjelmistoprojektista

Tapaustutkimuksessa tutkitaan sopivaa ohjelmistotuotantomenetelmää jolla voidaan tuottaa nopeasti ja asiakkaalle selkeällä tavalla prototyyppi tulevasta ohjelmistosta. Prototyyppi toimii myös näyttönä tilaajayritykselle siitä, miten ohjelman tuottava taho toimii ja mikä on tämän ammatillinen pätevyys. Sopivan menetelmän valinnan jälkeen laaditaan prototyyppi joka vastaa tilaajan tarvetta.

Rooleja tutkielmassa on kaksi. Molemmat roolit suoritetaan kuitenkin kirjoittajan toimesta. Tutkielmaa tehdessä pyrin mukautumaan ja tarkastelemaan asioita tilaajan ja tuottajan näkökulmasta. Tärkein lähtökohta on, se että yksikään ratkaisu ei ole pelkästään hyvä tai huono. Ratkaisuilla on ominaisuuksia joiden sopivuus ratkaisuun riippuu kunkin roolin päämääristä ja odotuksista. Tuottajan päämääränä on valmistaa prototyyppi, joka vakuuttaa tilaajan tuottajan ammattitaidosta. Prototyypissä esitettävät lasku ja kuitti ovat erittäin tärkeitä. Molempien pitää olla selkeitä käytön kannalta ja niiden täytyy täyttää lain antamat vaatimukset. Tilaajan päämääränä on varmistaa tuottajan osaaminen ja vakuuttua ratkaisun kattavuudesta heidän tarpeensa huomioiden. Valmiin prototyypin pohjalta molemmat osapuolet ymmärtävät paremmin millainen lopullinen ratkaisu tulee olemaan ja kauanko sen tuottamisessa tulee kestämään. Samalla voidaan arvioida syntyviä kustannuksia.

Tutkimuksessa kehitetään ratkaisu, joka toimii pohjana varsinaiselle toteutukselle. Toteutuksessa huomioidaan modulaarisuus ja yksinkertaisuus tulevan laajenemisen kannalta. Prototyypin laadinta lähtee toimeksiannosta, jossa määritellään tarpeet joihin prototyyppi tulee vastaamaan. Ohjelmistotuotantomalli valitaan tilaajan, tuottajan ja toimeksiannon pohjalta. Valitun mallin mukaisesti laaditaan arkkitehtuuri (iteraatio 1), laskun ja kuitin kuva (iteraatio 2) sekä asiakas- ja maksutietojen rakenne, käyttö ja säilytys (iteraatio 3).

3.1 Tilaajayritys

Tilaajayritys on pieni lihaleikkaamo, joka työllistää kaksi työntekijää. Leikkaamo leikkaa/pakkaa ruhoja asiakkaiden toimeksiantojen mukaan ja valmistaa ruhoista lihavalmisteita. Leikkaamolla on asiakkainaan maatiloja, jotka leikkauttavat ja teettävät itse kasvattamistaan eläimistä valmisteita. Leikkaamo ei itse myy tuotteita eteenpäin. Asiakkaat noutavat valmiit tuotteet tai tilaavat niille kuljetukset. Asiakkaat hoitavat tuotteiden myynnin itse.

3.1.1 Tilaajayrityksen tämän hetkinen tietojärjestelmä

Yrityksen toimintaa pyöritetään paperilla ja kynällä. Työn laskutus tapahtuu syöttämällä tietoja erilliseen laskutusohjelmaan, johon kootaan paperilta ylös toteutetut työt. Valmiit laskut syötetään käsin Exceliin (laskun numero, summa, alv erottelu). Exceliin kootaan kuukausittain aineisto joka sitten aina kuun lopussa lähetetään tilitoimistolle kirjanpitoa varten. Tilauksia leikkaamo ottaa vastaan sähköpostitse ja puhelimitse. Asiakkaina toimivat maatilat ovat leikkaamolle tuttuja, joten protokolla toiminnan suhteen on vapaamuotoista. Tärkeintä on, että ruhot tulevat teurastamosta suurinpiirtein ajallaan ja että laskut maksetaan suurinpiirtein ajallaan.

3.1.2 Tilaajayrityksen suunnitelma toiminnan kehittämiseen

Leikkaamo haluaa jatkossa alkaa myymään lihaa itse eteenpäin lähialueen kaupoille ja ravintoloille. Tässä tapauksessa leikkaamo ostaisi ruhoja tuottavilta maatiloilta ja myisi ne eteenpäin valmisteina. Leikkaamo alkaisi maksamaan asiakkailleen ruhoista. Kirjanpidon kannalta tämä on uusi toiminto. Leikkaamo haluaa myös alkaa myymään tuotteita suoraan toimitilastaan käsin. Aikaisemmin kaikki myynti on tapahtunut laskutuksen kautta ja asiakkaat ovat olleet tuttuja. Suoramyynnissä asiakkaat ovat käteisasiakkaita ja yrityksen täytyy alkaa käsittelemään käteismyyntiä. Samalla kirjanpidollinen aineisto kasvaa merkittävästi koska yksittäiset myyntitapahtamut pienenevät rahallisesti ja niiden lukumäärä kasvaa.

3.1.3 Tilaajayrityksen toimeksianto

Tilaaja haluaa yrityksensä pyörittämisen tueksi helpon ja mobiilin tavan hoitaa laskutusta ja kuittien kirjoittamista. Laskut ja kuitit halutaan lähettää paperittomasti ja niistä tarvitaan selkeä kopio myöhempiä tarkistuksia varten. Aineiston on tarkoitus toimia kirjanpidon tukena, joten sen pitää olla selkeä ja helposti monistettava. Tulevaisuudessa asiakkaalla saattaa olla tarve lisätä ratkaisuun toiminnallisuuksia (kuukausikohtainen myynti, tietyn asiakkaan laskutuksen kokonaissumma, koko vuoden myynnin summa yms). Toiminnallisuuksien lisäyksiä ei toistaiseksi tarvitse työstää, mutta toteutusta tehdessä muokattavuus ja avoimuus on pidettävä korkealla prioriteetilla.

Tilaajalla on tulevaisuudessa kaksi peruskäyttötarvetta ohjelmalle:

1) Tilaaja myy elintarvikkeita ravintoloille ja jälleenmyyjille. Lasku sisältää useita tuotteita ja siitä ilmenee laskussa vaadittavat pakolliset tiedot. Yritysmyynnissä

käytetään vain laskua. Laskua ei tavallisesti lähetetä yksityisasiakkaille, mutta jos kyseessä on luotettava asiakas ja tilaus on suuri, niin laskun käyttö on mahdollista.

2) Tilaaja myy tuotteitaan suoraan asiakkaalle toimitiloistaan. Asiakas maksaa tuotteet käteisellä ja saa ostosta kuitin. Kuitti lähetetään ensisijaisesti sähköpostiin, mutta halutessa kuitti on mahdollista tulostaa paperiversiona.

3.2 Tuottaja

Tuottaja on pienehkö IT-startup, joka on erikoistunut tekemään kevyitä ratkaisuja nopealla aikataululla. Ratkaisujen teossa keskitytään valmiin tuotteen tekemiseen kevyesti ja joustavasti yhteistyössä asiakkaan kanssa. Aloittavalle yritykselle liikevaihdon kartuttaminen on tärkeää, joten asiakkaita kalastetaan tiheäsilmäisillä verkoilla. Tuottajan puolelta projektiin osallistuu projektipäällikkö ja tarvittava määrä koodaajia. Projektipäälliköllä on myös koodaajan rooli, joten on mahdollista että hän hoitaa yksin koko asiakastyön toteutuksen.

3.3 Ratkaisuehdotus yleisesti

Ratkaisuehdotukseksi tarjotaan selainpohjaista käyttöliittymää. Selainpohjainen ohjelmisto toimii mobiililaitteilla ja tietokoneilla, jos käytössä vain on verkkoyhteys. Itse ohjelmisto pyörii ulkopuolisen palveluntarjoajan palvelimella. Tuottajayritys vastaa yhteydenpidosta palvelun tarjoajaan. Tuotettavan ohjelmiston työnimenä toimii kirjanpitosofta.

Kirjanpitosoftan pitää olla helppokäyttöinen. Ohjelmaan voi kirjautua sisään omilla tunnuksilla ja luoda laskuja tai kuitteja. Käyttäjiä on useita ja kaikilla on yhtäläiset mahdollisuudet tehdä laskuja/kuitteja. Ohjelmalla on myös pääkäyttäjä joka voi luoda uusia tunnuksia muille käyttäjille. Pääkäyttäjä voi muokata ja katsoa kaikkea aineistoa. Muilla käyttäjillä oikeudet ovat rajatummat. Luodut laskut ja kuitit lähetetään sähköpostilla eteenpäin. Laskun/kuitin voi myös halutessaan tulostaa. Toteutuksessa kuitti periytyy laskusta. Ulkoasullisesti kuitti ja lasku ovat muuten samanlaisia, mutta kuitista puuttuu laskun maksamiseen liittyvät tiedot. Kuitissa ostajana voi myös olla varmentamon käteisasiakas. Laskussa ostaja on aina varmennettu.

Käyttäjän tarvitsevat perustoiminnat ovat:

1) laskun teko ja lähetys,
2) kuitin teko ja lähetys,
3) asiakastilien hallinta,
4) käyttäjätilien hallinta,

5) aineiston selaus ja
6) kirjanpidolliset toiminnot.

Käyttöliittymässä siirrytään sivulta toiselle valikoita käyttäen. Käyttö aloitetaan kirjautumalla sisään käyttäjän omilla tunnuksilla. Etusivun valikko sisältää useimmin käytetyt toiminnot (laskun luonti, kuitin luonti). Pääkäyttäjällä on pääsy kaikkiin ohjelman toiminnallisuuksiin, muilla käyttäjillä pääsy on rajattu käyttötarpeen mukaan. Kirjanpitäjällä, myyjällä ja pääkäyttäjällä on erilaiset tarpeet ohjelman käyttöön. Tarpeet huomioidaan näkyvyydessä.

3.4 Ratkaisuehdotuksen toteutus

Ratkaisusta tarjotaan asiakkaalle kolmiosainen toteutus. Toteutukseen kuuluu:

- prototyyppi toteutuksesta,
- suunnitelma ohjelmiston arkkitehtuurista ja
- esimerkkiaineistoa (lasku ja kuitti).

Prototyypissä demotaan joitain toiminnallisuuksia, mutta pääpaino on käyttöliittymän suunnitelmalla. Arkkitehtuuri sisältää suunnitelman ohjelmiston rakenteesta ja siinä huomioidaan myös web-pohjaisen sivuston toiminnallisuutta osana rakennetta. Esimerkkiaineistosta laaditaan visuaalinen lasku ja kuitti, joita asiakkaille voitaisiin ohjelman kautta tuottaa.

Tilaaja tekee lopullisen ostopäätöksen tai tilaa mahdollisesti toisen prototyypin ensimmäisen version valmistumisen jälkeen. Koska tilausta ei ole tarkasti määritelty ja tilaaja vasta itsekin käy läpi tarvettaan, niin ensimmäisen prototyypin ja suunnitelman esittelyn yhteydessä voi ilmetä muutosehdotuksia. Varsinaisen tilauksen sopimus laaditaan vasta prototyypin hyväksymisen jälkeen.

3.5 Ohjelmistotuotantomenetelmän valinta

Tilaaja haluaa nähdä ennen kehityksen aloitusta prototyypin ohjelmasta. Prototyypistä pitää käydä ilmi tuotettavan sovelluksen keskeisiä toimintoja. Tilaaja käyttää prototyypin tuottamista myös toimittavan tahon osaamisen, työtapojen ja luotettavuuden testaamiseen. Tilaaja tarkkailee erityisesti onnistutaanko prototyyppi toimittamaan ajallaan ja onko se tarpeeksi laadukas. Lisäksi huomiota kiinnitetään viestintään ja sen läpinäkyvyyteen. Prototyypin pohjalta pystytään laskemaan tarkempi hinta-arvio toteutukselle ja pohtimaan, onko tilaajan käyttötarkoitus ymmärretty ja kuvattu tarpeeksi selvästi. Tilaaja jättää myös itselleen mahdollisuuden määritellä tarvittavia ominaisuuksia prototyypin näkemisen jälkeen, jos uusia liiketoiminnallisia tarpeita

ilmenee. Tilaaja saattaa tilata uuden prototyypin tai nykyisen prototyypin korjaustyön, jos ensimmäisessä ilmenee puutteita tai tilaajan oma tarve selkiytyy toteutuksen aikana.

3.5.1 Menetelmien läpikäynti

Toteutuksen tulee olla joustava ja dynaaminen. Asiakas ei ole täysin varma tulevasta tarpeestaan ja vaatimusmäärittelyä on tarkoitus tehdä projektin edetessä. Työskentelevä tiimi on kooltaan pieni ja projektissa on tiivis keskusteluyhteys asiakkaan kanssa.

Vesiputousmalli olisi näin pieneen projektiin aivan liian raskas. Vesiputousmallia ei myöskään voida soveltaa, koska tilaajan vaatimukset eivät ole vielä kokonaan selvillä. Lopullisten vaatimusten selvittyä tilanne on kuitenkin toinen. Kyseisessä asiakascasessa kattavien alkumäärittelyjen tekeminen ja vesiputousmalliin sitoutuminen voisi kuitenkin olla anniltaan epävarma investointi. Prototyypin laatimisen jälkeen asiakkaalla on jo selkeämpi malli ja tarkempi määrittely vaatimuksista. Vesiputousmallin soveltaminen kokonaisen ohjelman tekemiseen ei ole poissuljettu vaihtoehto.

Ketteristä menetelmistä DSDM-menetelmä ja XP sopisivat tuotantomalleiksi prototyypin osalta. Malleissa valmistetaan alussa prototyyppi asiakkaan vaatimusten pohjalta. Projektin koko on kuitenkin niin pieni, ettei kaikkia rooleja saada täytettyä vaadittavalla tavalla. Lisäksi turhan kaavamainen ja byrokraattinen tuotantomalli lisäisi työmäärää, mutta ei välttämättä tarjoaisi vastavuoroisesti etuja prototyypin suunnittelussa. XP ja DSDM-menetelmä ovat myös raskaita raportoinnin ja testauksen suhteen. Asiakkaalle tuotetaan Lean-filosofian mukaisesti prototyyppi, joten alkuvaiheessa turhaa byrokratiaa/testaamista/raportointia voi karsia. Tulevaan varsinaiseen toteutukseen DSDM-menetelmä tulisi sopimaan erinomaisesti.

Scrum olisi myös varteenotettava malli kokonaisen ohjelman toteuttamiseen, kunhan alkumäärittelyt on saatu hoidettua. Prototyypin tekemisessä Scrum tarjoaa samoja rajoitteita kuin DSDM-menetelmä ja XP. Scrum-tuotantomallin etuna DSDM-menetelmään ja XP-malleihin verrattuna tässä asiakascasessa on scrum-tiimin pienempi mahdollinen koko.

Toimeksiannon toteuttamisen työkalu valitaan muista ohjelmistotuotantomenetelmistä. Menetelmäksi valitaan prototyyppimalli. Prosessiin otetaan vaikutteita lean- ja scrum-menetelmistä. Vaikutteet ilmenevät siten, että asiakkaaseen tullaan pitämään tiivistä yhteyttä ja asiakas on mukana prototyypin rakentamisessa.

Prototyyppimallissa ryhmä laatii prototyypin, joka sitten esitellään asiakkaalle. Tässä tapauksessa ryhmä laatii kolmessa iteraatiossa kolme eri prototyyppiä, jotka sitten lopuksi esitetään kerralla asiakkaalle. Kolme erillistä osatoteutusta olivat:

- prototyyppi toteutuksesta,
- suunnitelma ohjelmiston arkkitehtuurista ja

- esimerkkiaineistoa (lasku ja kuitti).

Luvussa 5 käsitellään edellämainitut osat omina prototyyppeinään.

3.6 PSP:n tavoitteet

Prototyypin koodaukseen kieleksi olen valinnut PHP:n. PHP on minulle entuudestaan tuntematon ohjelmointikieli. Tavoitteenani on opiskella PHP ja omaksua siitä sellainen tietämyksen taso, että pystyn prototyypin kirjoittamaan. Samalla myös tutustua PHP:n rakenteeseen arkkitehtuurin kannalta. Pyrin tekemään ohjelman suunnitelmasta sellaisen, että se on mahdollisimman helposti toteuttavissa PHP:lla.

Arkkitehtuurin osalta tavoitteenani on luoda selkeä malli kokonaisen ohjelman arkkitehtuurista. Mallin luominen on ensimmäinen itsenäisesti suorittamani arkkitehtuurin suunnitelma. Tavoitteenani on tutustua eri arkkitehtuurimalleihin. Toinen ja varsinainen päätavoitteeni arkkitehtuurin saralta on luoda arkkitehtuurikuvaus tulevasta toteutuksesta.

Kolmas tavoite on muodostaa esimerkkiaineisto laskusta ja kuitista jotka täyttävät Finvoice-standardin vähimmäisvaatimukset. [Finanssivalvonta, 2019] Esimerkkiaineiston laadinnassa pyrin myös kiinnittämään huomiota aineiston luettavuuteen ja ymmärrettävyyteen ja siihen kuinka käytettäviä laskut ovat. [Nielsen 1994]

Prototyypin tekemisessä valitut PSP tavoitteet:

1. PHP ohjelmoinnin perusteet, prototyypin ohjelmointi
2. Arkkitehtuurimallien perusteet, arkkitehtuurimallin luominen
3. Finvoice-standardiin tutustuminen ja lain mukaisen aineiston luonti

4 Toteutus ja PSP:n tavoitteiden toteutuminen

Toteutus koostuu kolmesta iteraatiosta. Jokainen iteraatio on oma kokonaisuutensa ja ne käsitellään yksitellen. Ensimmäisessä iteraatiossa käydään läpi ohjelman tuleva arkkitehtuuri yleisellä tasolla. Toisessa iteraatiossa muodostetaan laskun ja kuitin kuva. Kolmannessa iteraatiossa käsitellään maksujen ja asiakastietojen käyttöä ja säilytystä. Iteraatioihin liittyvät PSP:n tavoitteiden toteutuminen käydään läpi luvun lopussa.

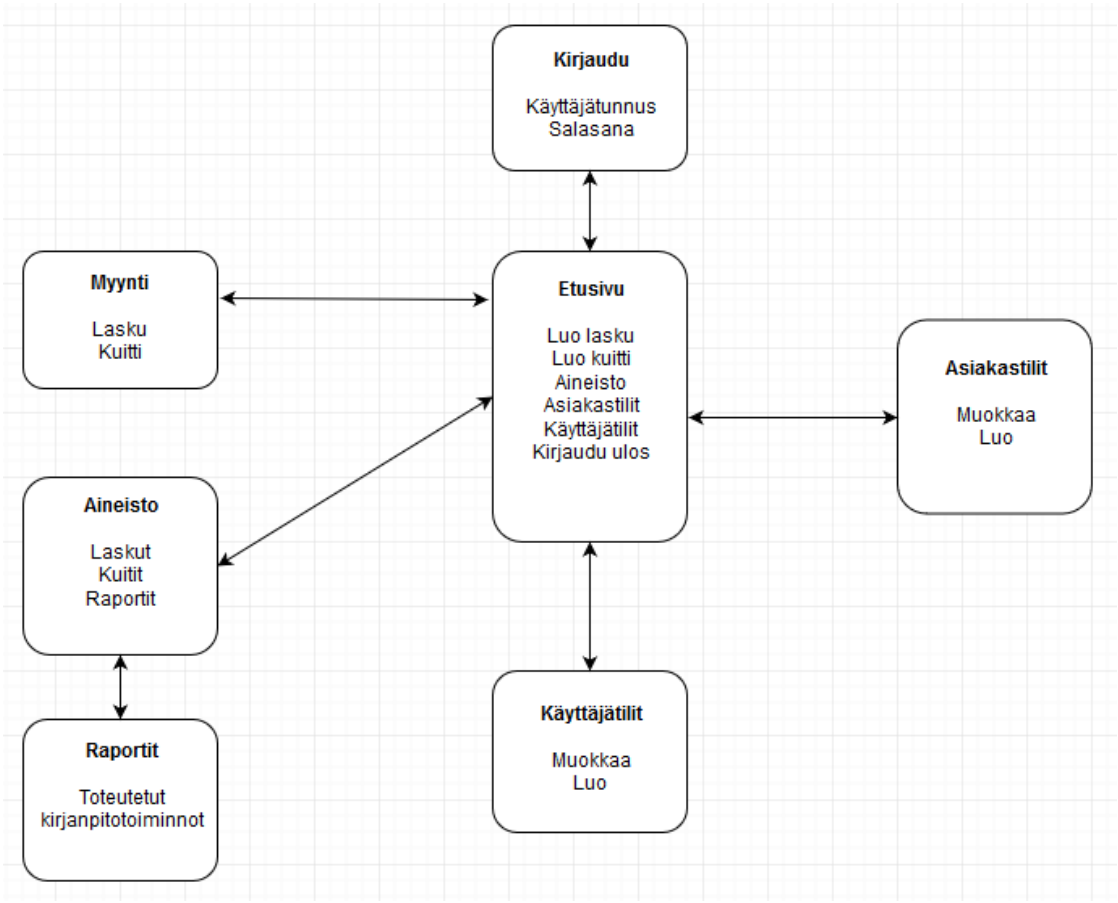
4.1 Iteraatio 1: Arkkitehtuuri

Arkkitehtuuri on tulevan ohjelman rakenteen selkäranka. Onnistuneella arkkitehtuurilla voidaan vaikuttaa toteutuksen selkeyteen, toimivuuteen ja kustannukseen. Arkkitehtuurisuunnitelmassa tullaan kiinnittämään erityistä huomiota muokattavuuteen ja toteutuksen edullisuuteen.

4.1.1 Prototyypin rakenne

Toteutus tullaan tekemään PHP:lla ja sovellus on selaimella käytettävissä oleva web-sovellus. Sovellukseen kirjaudutaan ensin sisään, jonka jälkeen tunnistettu käyttäjä voi navigoida ohjelman sisällä. Peruskäyttönä on laskun/kuitin luominen ja lähettäminen asiakkaalle tai tulostaminen. Uusien asiakastilien perustaminen on myös toimenpide joka on yleinen. Kirjanpidolliset toiminnot ja materiaalin tarkastelu on harvinaisempaa käyttöä, johon kaikilla käyttäjillä ei tule olemaan oikeutta.

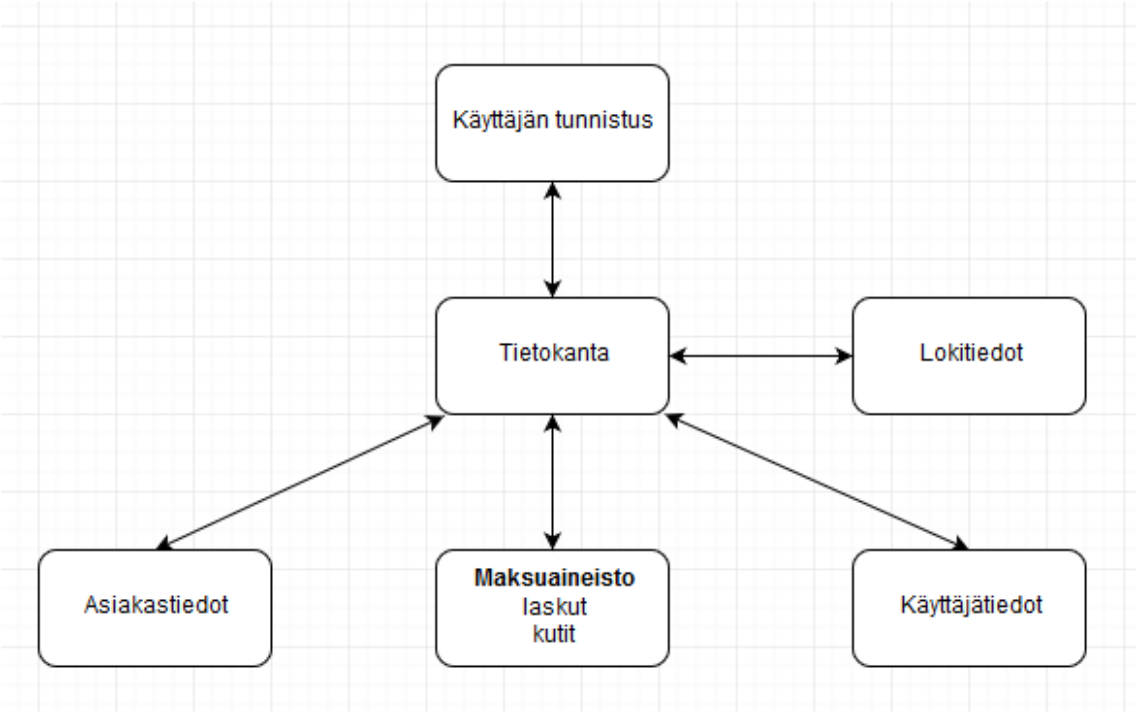
Ohjelman käyttö keskittyy etusivun toimintojen ympärille. Myyntikäytössä ohjelman käyttäjä valitsee joko laskun tai kuitin riippuen maksutapahtumasta. Laskulle myyminen vaatii aina käyttäjätilin perustamisen. Toiminnon käytön jälkeen ohjelmassa on paluu etusivulle (esimerkiksi uuden käyttäjätilin luominen). Kuvassa 9 esitellään ohjelman keskeisimmät toiminnot ja rakenne.



Kuva 9: Ohjelman keskeisimmät toiminnot ja rakenne.

4.1.2 Tietokanta

Tietokanta sijaitsee palvelimella ja käyttäjällä on siihen pääsy ohjelman välityksellä. Käyttäjän oikeuksista riippuu, mitkä tietokannan ominaisuudet ovat käytettävissä. Käyttäjän tunnistus tapahtuu ohjelmaan kirjautumisen yhteydessä. Tämä on syytä huomioida ohjelmaa käytettäessä siten, että käyttäjät kirjautuvat ulos jos lopettavat avoimella päätteellä ohjelman käytön. Näin muut eivät tule vahingossa käyttäneeksi toisten tunnuksia. Jos erillinen kirjautuminen tuottaa ohjelman käytössä tarpeettomasti lisätyötä, niin on mahdollista luoda kaikille avoin käteismyyntitili jota käytetään vain myyntityössä. Laskujen muokkaus onnistuu vain pääkäyttäjältä, mutta myynnin yhteydessä asiakastietojen haku laskulle on kaikille käyttäjille jaettu ominaisuus. Lokitietoihin tallentuu kuka käyttäjä on laatinut laskun/kuitin tai asiakas/käyttäjätiedon. Lokitietoja ei ole tarkoitus seurata aktiivisesti vaan niitä on tarkoitus tarkistella vain erikoistapauksissa. Lokitietoihin tallentuu myös jos yksittäinen lasku tai maksu poistetaan järjestelmästä tai jos sitä muokataan jälkikäteen. Kuvassa 10 esitellään tietokanta vapaamuotoisena vuokaaviona.



Kuva 10: Tietokanta.

4.1.3 Yhteys palvelimelle

Ohjelmaa käytetään selaimella, joten käyttöön tarvitaan toimiva internetyhteys. Ohjelma sijaitsee kolmannen osapuolen palvelimella ja kolmas osapuoli vastaa palvelimen ylläpidosta. Nämä kaksi seikkaa muodostavat kaksi mahdollista virhetilannetta jotka estävät ohjelman käytön:

1) Internetyhteys ei toimi.
2) Palvelin on alhaalla.

Ensimmäisen virhetilanteen voi aiheuttaa vaikka sähkökatkos. Katvealueella tapahtuva käyttö voi myös kärsiä huonosta signaalista tai käyttöpiikki verkon alueella voi aiheuttaa tiedonsiirrossa ruuhkaa. Näihin voi varautua oikean tyyppisellä liittymällä ja vaikkapa lisäantenneilla ja tukiasemilla. [Korhonen, 2014]

Kolmannen osapuolen palvelimen virheeseen on vaikea varautua. Kumppania valittaessa tulee kiinnittää huomiota tarjottavan palvelun laatuun ja sopimuksen sisältöön. Mahdolliset virhetilanteet ja niiden aiheuttamat toimenpiteet tulee käydä selvästi ilmi sopimuksesta.

Molemmat virhetilanteet voidaan ratkaista siten, että palvelin sijoitetaan asiakkaan tiloihin ja käyttö tapahtuu kaapelin välityksellä. Tässä tapauksessa käytön voisi kuitenkin estää sähkökatkos tai palvelimen virhe. Palvelimeen virhetilan ratkaiseminen olisi lisäksi tilaajan vastuulla ja se tuo omat vastuunsa toimintaan.

Riskeihin voi myös varautua manuaalisella laskujen/kuittien kirjoittamisella. Virhetilanteen varalta voidaan tilata esitäytettyjä kuitti/laskupohjia joihin kirjoitetaan käsin tuotteet ja hinnat. Käsin laaditut kuitit/laskut voidaan sitten siirtää kirjanpito-

ohjelmaan yhteyden palattua. Manuaalinen kirjoitusmahdollisuus on edullinen tapa varautua mahdollisiin yhteysvirheisiin.

Yhteysvirheen mahdollisuus on olemassa, mutta oikein valitun internetyhteyden ja kolmannen osapuolen kanssa riskin todennäköisyys voidaan laskea pieneksi. Mahdolliseen yhteysvirheeseen kannattaa kuitenkin varautua jollain edullisella ja helposti käyttöön otettavalla varajärjestelmällä, joka voidaan myöhemmin siirtää järjestelmään. Yksi hyvä vaihtoehto on edellä esitetty käsin kirjoitettava osittain valmiiksi täytetty tallentava kuitti- tai laskupohja.

4.1.4 Web-pohjaisen toteutuksen edut

Selaimen kautta suoritettava käyttö on ennen kaikkea joustavaa. Ohjelmaa voidaan käyttää millä tahansa laitteella, jossa vain on toimiva internetyhteys. Ohjelmaa voidaan toteuttaa niin, että sitä voi käyttää helposti sekä puhelimella, että tietokoneella. Suunnittelussa otetaan huomioon millä laitteella käyttö tulee pääasiassa tapahtumaan ja ohjelma voidaan optimoida sitä laitetta varten. Alustavasti on sovittu, että käyttö tulee tapahtumaan pääasiassa kannettavalla tietokoneella.

Ohjelmaa voidaan myös käyttää käyttäjän sijainnista riippumatta, kunhan vain on olemassa toimiva internetyhteys. Samoin ohjelmaan on helppo tarjota pääsy toisille osapuolille, esimerkiksi tilitoimistolle.

4.2 Iteraatio 2: Laskun ja kuitin kuva

Lasku ja kuitti ovat välttämättömiä tositteita yrityksen kaupankäynnin yhteydessä. Laskua ja kuittia käytetään kirjanpidon pohjana ja niissä on yhteisiä ominaisuuksia. Kuitti voidaan nähdä maksettuna laskuna, jossa ei tarvitse olla maksajan tietoja. Yhteisiä ominaisuuksia ovat mm: päiväys, tunniste, tuotteiden erittely, verojen erittely ja maksun saajan tiedot. [Finlex, 2019; Lasku, 2019]

4.2.1 Kuitti

Kuitti voidaan nähdä ostotapahtuman yhteydessä maksettuna laskuna. Kuitti on muutamaa poikkeusta lukuunottamatta pakollinen tosite kaupankäyntitilanteissa, jossa myyvänä osapuolena on elinkeinon harjoittaja. Kuitin perimmäinen tarkoitus on valtion kannalta verotuksellinen. Kuitin pakollisia tietoja ovat [Finlex, 2019]:

1) elinkeinonharjoittajan nimi, yhteystiedot ja y-tunnus,
2) kuitin antamispäivä,

3) kuitin tunnistenumero tai muu yksilöivä tieto,
4) myytyjen tavaroiden määrä ja laji sekä palvelujen laji ja
5) tavaroista tai palveluista suoritettu maksu ja suoritettavan arvonlisäveron määrä verokannoittain taikka arvonlisäveron peruste verokannoittain.

Kuittiin voi kirjata muitakin tietoja, mutta edellä listatut ovat pakollisia. Suunnittelun kannalta kuitin esimerkkikuvassa on kiinnitetty huomiota selkeyteen. Selkeyttä korostetaan taulukkomaisella rakenteella ja informaation minimoinnilla. Kuvassa 11 esitellään kuitti, josta käyvät ilmi kaikki lain vaatimat pakolliset tiedot.

Kuitti
Kuittinumero 180607
Päivämäärä 01.02.19

Myyjä
Nimi Myyjä
Y-tunnus Myyjän Y-tunnus
Postiosoite Myyjän postiosoite

Erittely:

Rivi	Tuote	Lisätiedot	Määrä (kg)	Hinta per määrä	Hinta sis alv	Alv %	Alv €	Hinta alv 0
1	Villasika	potka	1,525	16	24,40 €	14	3,00 €	21,40 €
2	Villasika	kulmapaisti	1,2	16	19,20 €	14	2,36 €	16,84 €
3	Villasika	ulkofile	1,39	16	22,24 €	14	2,73 €	19,51 €
	yhteensä		4,115		65,84 €		8,09 €	57,75 €

Kuitattu maksetuksi **Pvm** 01.02.19

Kuva 11: Esimerkki kuitista.

4.2.2 Lasku

Siinä missä kuitti on tosite maksetusta kauppatapahtumasta, niin lasku on tosite kauppatapahtumasta, joka jää avoimeksi. Kauppatapahtuma sulkeutuu vasta kun maksu on suoritettu ja palveluntarjoajan ostajalle antama luotto on kuitattu maksetuksi. Kuittiin verrattuna lasku sisältää asiakkaan tiedot ja tiedot maksuehdoista. Laskussa pitää siis olla tiedot siitä, että kuka maksaa ja milloin/miten maksu tulee tapahtumaan. Laskussa on myös huomautusaika, jonka puitteissa palveluntarjoaja voi katsoa maksuehdot rikotuiksi ja lasku voi siirtyä jopa perintään. [Lasku, 2016]

Lasku on aina myös riski palveluntarjoajalle. Ostajalle annettu luotto voi jäädä maksamatta ja palveluntarjoaja ilman hänelle kuuluvaa yhteisesti sovittua korvausta. Tätä varten laskussa on asiakkaan tiedot. Prototyypissä lasku tarjotaan vain yritysasiakkaalle tai luotetulle henkilöasiakkaalle. Asiakkaan tunnisteena myyjän järjestelmässä on yrityksen tapauksessa y-tunnus ja henkilön tapauksessa henkilötunnus.

Laskua varten asiakkaan pitää perustaa tili palveluntarjoajan järjestelmään, jonka kautta asiakastiedot haetaan laskulle laskun luomisen yhteydessä. Tilin perustamisen yhteydessä myyjä tarkistaa asiakkaan mahdolliset maksuhäiriömerkinnät ja tilin

avauksen yhteydessä on hyvä arvioida asiakkaan maksukyky myös yleisellä tasolla. Esimerkiksi juuri perustettu tuntematon yritys, joka haluaa ostaa isoja eriä tuotteita laskulla voi herättää myyjän puolella huolta asiakkaan maksukyvystä.

Laskun ulkoasu muistuttaa kuitin ulkoasua. Erotuksena ovat kuittiin verrattuna ylimääräiset tiedot asiakkaasta ja maksun ehdoista. Suunnittelussa on kiinnitetty huomiota selkeyteen ja tietojen jäsentelyyn. Kuvassa 12 esimerkkilasku.

Lasku

Laskunumero	190107
Lähetys toimitettu	08.01.19

Lähettäjän tiedot:

Nimi	Myyjän nimi
Y-tunnus	Myyjän y-tunnus
Postiosoite	Myyjän osoite

Vastaanottajan tiedot:

Nimi	Ravintola Nili
Y-tunnus	8765432-1
Postiosoite	Nilintie 1 95355 Rovaniemi

Erittely:

Rivi	Tuote	Lisätiedot	Määrä (kg)	Hinta per määrä	Hinta sis alv	Alv %	Alv €	Hinta alv 0
1	Villasika	potka	1,525	16	24,40 €	14	3,00 €	21,40 €
2	Villasika	T-bone	0,41	16	6,56 €	14	0,81 €	5,75 €
3	Villasika	niska	0,53	16	8,48 €	14	1,04 €	7,44 €
4	Villasika	kulmapaisti	1,2	16	19,20 €	14	2,36 €	16,84 €
5	Villasika	ulkofile	1,39	16	22,24 €	14	2,73 €	19,51 €
	yhteensä		5,055		80,88 €		9,93 €	70,95 €

Saajan pankkiyhteys:

Pankki, nimi	OP Pohjola
Pankki, BIC	OKOYFIHH
Tilinumero, IBAN	myyjän tilinumero
Viite	190107
Summa	80,88 €
Eräpäivä	1.7.2018
Huomautusaika	7 päivää

Kuva 12: Esimerkki laskusta.

4.2.3 Visuaalinen yhteenveto laskusta ja kuitista

Suunnittelun lähtökohtana on ollut dokumentin selkeys. Selkeyteen on pyritty ensisijaisesti taulukkomaisella rakenteella ja yhteenliittyvien tietojen sijoittamisella omiin osioihinsa. Yhteenliittyviä tietoja ovat esimerkiksi maksajan tiedot, maksun saajan tiedot, maksutiedot, dokumentin tunnisteet ja tuotteiden listaus. Dokumenteissa on myös pyritty käyttämään tietosisällön kannalta lain minimiä selkeyttä lisäävänä tekijänä. Kuitissa ja laskussa voisi olla paljon muitakin tietoja, mutta nyt visuaaliseen esitykseen on valittu vain lain kannalta pakollisia tietoja.

4.3 Iteraatio 3: Asiakas- ja maksutietojen rakenne, käyttö ja säilytys

Tietojen rakenteeksi valitaan taulukkopohjainen selkeä ja kevyt malli. Säilytyksessä ja käytössä tullaan kiinnittämään huomiota lain vaatimaan turvallisuuteen.

4.3.1 Asiakkaiden tietojen tallennus

Asiakkaan tiedot kerätään erilliseen tekstitiedostoon. Samassa tiedostossa on kaikkien asiakkaiden tiedot riveittäin. Kaikki tiedot eivät ole pakollisia ja tyhjiä kenttiä voi olla. Kentät on eroteltu toisistaan puolipisteillä. Ohjelman on tarkoitus lähettää asiakkaalle lasku sähköpostiin, joten sähköpostiosoite on pakollinnen. Etunimi ja sukunimi ovat myös pakollisia tietoja asiakkaiden erottelun kannalta. Järjestelmä tuottaa asiakasid:n automaattisesti tilin luomisen yhteydessä, joten se on pakollinen mutta sitä ei anneta erikseen. Luomisen yhteydessä asiakkaan sähköposti ja y-tunnus tarkistetaan muiden tietokannassa olevien tietojen suhteen. Tarkoituksena on varmistaa, ettei asiakasta ole jo luotu ja ennaltaehkäistä mahdollinen duplikaatti. Kellään olemassaolevalla asiakkaalla ei saa olla samoja tietoja (y-tunnus voi olla kuitenkin tyhjä jos kyseessä on käteisasiakas). Jos sähköposti tai y-tunnus on jo käytössä, niin ohjelma ei anna luoda uutta asiakasta.

Esimerkkisyöte:
2;Sami;Peltomaa;Peltotie3;11110;Salmi;045-12345678;*s.peltomaa@gmail.com*;
peltotalli; Peltotie 3;11110;Salmi;1234567-8;;;

4.3.2 Maksun rakenne ja atk

Yksittäinen maksu tallennetaan omaan tekstitiedostoon maksun luomisen jälkeen. Maksut tallennetaan asiakaskohtaisiin kansioihin. Tiedoston nimi on samalla maksun yksilöivä tunniste. Maksulla on otsikkorivi, tuotekohtaiset rivit ja summarivi. Rivien sisällä tietokentät erotellaan toisistaan puolipisteellä. Jos kenttä on tyhjä niin sen merkkinä on silloin vain puolipiste.

Otsikkorivillä on maksun tyyppi, päiväys ja tunniste. Otsikkorivin tiedot yksilöivät maksun ja jakaa aineiston laskuihin ja kuitteihin. Otsikkorivin perusteella maksuja voidaan käsitellä kirjanpidollisin toimin. Päiväyksen perusteella voidaan hakea tietyn ajanjakson aineisto.

Tuotekohtaisilla riveillä on eritelty maksun sisältämät tuotteet, niiden määrät ja hinnat. Hintojen lisäksi tuoterivi sisältää rivikohtaisen alv-erottelun.

Summarivillä on laskun yhteenlaskettu arvo ja maksukohtainen alv-erottelu. Summarivin tiedoista voidaan poimia suoraan koko maksun summa asiakkaalle ja kirjanpitoa varten.

Maksussa ei ole tietoja lähettäjästä/myyjästä (vakiona vain yksi), eikä myöskään vastaanottajasta eikä saajan pankkiyhteydestä. Lähettäjä ja pankkiyhteys ovat aina sama ja ne haetaan laskulle vasta laskun/kuitin kuvan luomisen yhteydessä. Tietoja ei siis säilytetä maksukohtaisesti tietokannassa. Asiakkaan tietoja ei myöskään pidetä maksussa mukana muuta kuin asiakasid:n osalta. Asiakasid:n avulla tiedot haetaan vasta laskun/kuitin kuvan luomisen yhteydessä. Tarkoituksena on minimoida tietokannassa olevan maksun rakenne.

Esimerkkimaksu (tyypiltään lasku):
L_190107;A14;010819;;;
1;potka;1,525;16;24,4;14,3;21,4;
2;kulmapaisti;1,4;14;16;72,58;17,42;
S;130;104,84;25,16;€;

Maksujen rakenne perustuu niiden kirjanpidolliseen käyttöön. Maksuja voidaan käsitellä esimerkiksi ajallisesti tai asiakkaan perusteella. Maksujen käytölle on laadittu seuraavia skenaarioita:

- Tietyn asiakkaan yksittäisen maksun hakeminen.
- Koko myynnin rahamäärä tietyltä ajanjaksolta.
- Myyntitapahtumien määrä tietyllä ajanjaksolla.
- Arvolisäveron kokonaismäärä tietyllä ajanjaksolla.

Skenaarioista esitellään tarkemmin arvolisäveron kokonaismäärän laskeminen. Kirjanpidon kannalta on tärkeää pystyä ilmoittamaan verottajalle koko myynnin arvolisäveron määrä tietyllä ajanjaksolla. Elinkeinonharjoittajan keräämästä arvonlisäverosta vähennetään kuluihin käytetty arvonlisäveron määrä ja saadun arvon perusteella suoritetaan arvonlisäveron maksu. Maksu voi olla myös negatiivinen jos ostoja on ollut enemmän kuin myyntiä. Arvonlisänveron määrään vaikuttaa myös arvonlisäveron alaraja ja huojennus, mutta niihin ei kiinnitetä toistaiseksi huomiota.
Tietojenkäsittelyn kannalta arvonlisänveron kokonaismäärän laskeminen toteutetaan seuraavalla tavalla:

1. Etsitään aineistoista kaikki maksut halutulta ajanjaksolta. Ajallinen kohdentaminen tapahtuu otsikkorivin päiväyksen perusteella. Päiväys on jaettu kolmeen eri kenttään (vuosi, kuukausi, päivä). Haku kohdentuu niihin maksuihin, joissa on haluttu päiväys (esimerkiksi 2018;10;* palauttaisi aineiston lokakuulta 2018).

2. Kerätään ja summataan laskuriin maksuista yhteenlaskettu_veron maara.
3. Aineiston läpikäynnin jälkeen palautetaan laskurin summa.

4.3.3 PDF-kuva

Maksun luomisen yhteydessä maksusta tehdään pdf-kuva, joka lähetetään vastaanottajan sähköpostiosoitteeseen. Pdf-kuva tallennetaan myös tietokantaan tekstitiedostomuotoisen laskun yhteyteen.

4.3.4 Tietojen käyttö

Asiakastietoja on tarkoitus muokata ja tarkastella erillisellä käyttöliittymän asiakastilit-sivulla. Sivulla voidaan hakea, päivittää, poistaa ja lisätä asiakastilejä.

Maksuja ei voi selata käyttöliittymän kautta, vaan ne pitää avata niiden tiedostosijainnista. Maksuja ei voi enää niiden luomisen jälkeen muuttaa. Jos maksua olisi tarkoitus muokata jälkikäteen, niin se tulee tehdä erillisellä toteutuksella esimerkiksi muutoslaskuna. Ohjelman tässä vaiheessa muutoslaskua ei vielä toteuteta.

4.3.5 Asiakkaalle esitetty prototyyppi

Asiakkaalle on esitetty lomakemuotoinen prototyyppi, jossa asiakastietoja syötetään lomakkeeseen ja lomake tallentaa tiedot tekstitiedostoon. Asiakkaalle on myös esitetty maksun luominen syöttämällä tiedot lomakkeeseen ja lomakkeen tietojen tallennus. Valmis maksun kuva on tyypistä riippuen joko lasku tai kuitti, joista kuvat on esitetty aiemmin kohdassa 5.2.

4.4 PSP:n tavoitteiden toteutuminen

PSP:n tavoitteet valittiin itselleni tärkeiden kehityskohteiden pohjalta. Tarkoituksena oli myös selvittää, miten mallia voidaan soveltaa uusien asioiden oppimisessa. Nyt saavutetut tavoitteet ovat 1. tason tuloksia ja tarvittaessa tavoitteita voidaan laatia lisää seuraaville tasoille.

4.4.1 PHP-ohjelmoinnin perusteet, prototyypin ohjelmointi

PHP-ohjelmointi tuli minulle uutena ohjelmointikielenä. Entuudestaan vieraaseen ohjelmointikieleen oli toisaalta helppo päästä sisään JAVA-osaamiseni kautta, mutta toisaalta hankalaa, koska erot olivat pieniä mutta ilmeisiä. Tuotin ensiksi koodia pseudokoodina, jonka kokosin Top-Down-paradigman [Kornee, 2017] mukaisesti yleisestä toiminnasta kohti yksityiskohtaista pala palalta toteutusta. Periaate on sama mitä käytän Java:n kanssa, mutta nyt vain kieli oli eri. Koodin tuottamisen ajattelussa ei tapahtunut muutosta, mutta ajatuksen PHP-ohjelmointikielen syntaksin mukainen ilmaisu tapahtui uudella ohjelmointikielellä. PHP:stä opin kielen peruskäytön, jolla prototyyppi oli mahdollista muodostaa. Omaksuttavaa jäi vielä runsaasti, mutta aihealue on nyt avattu ja siihen on tutustuttu peruskäytön ominaisuudessa. Alustava tavoite tuli saavutettuksi, mutta avasi uusia tavoitteita, joihin kuuluu ohjelmointikielen parempi ja syvempi hallinta. Myös koodin tuottamisen nopeuttaminen olisi seuraava tavoite. Jos prototyyppiä aletaan tuottamaan valmiiksi tuotteeksi, niin asetan näistä jatkotavoitteita itselleni PSP:n osalta.

4.4.2 Arkkitehtuurimallien perusteet ja arkkitehtuurimallin luominen

Arkkitehtuurimalleista osaamiseni perustui käytyyn arkkitehtuurit-kurssiin. Kurssin käymisestä on kulunut jo hetki. Siinä käsiteltiin eri arkkitehtuurimalleja ja kurssin lopuksi suoritettiin suunnitelma toimivasta ohjelmistosta. Kurssin jälkeen suunnittelin työssäni ohjelmistointegraatioiden parissa pieniä kokonaisuuksia, jotka usein toimivat osana suurempia olemassaolevia järjestelmiä. Opinnäytetyötä aloittaessa suuremman kokonaisuuden suunnittelusta oli kulunut jo hetki ja tavoitteena oli suunnitella uusi ja selkeä kokonaisuus, joka olisi helposti muokattavissa ja kevyt toteuttaa. Tulos täytti tavoitteensa sen osalta, että palautin aiemman osaamiseni ja päivitin sitä myös hieman suunnittelua tutkiessani. Tarkoitus olikin palauttaa aiempi osaaminen ja luoda pohja tulevalle kehittämiselle. Mahdollisesti seuraavassa kokonaisen järjestelmän luonnissa uusia tavoitteita voisi olla osaamisen päivittäminen entuudestaan ja olemassaolevan suunnitelman testaus käytännössä. Käytännön toteutuksissa suunnitelman toteuttamisen edullisuus ja helppous on taloudellisesti merkittävä hyvä. Katson, että tämän hyveen toteuttaminen on tärkeää ja haluan kehittyä siinä jatkossakin.

4.4.3 Finvoice-standardiin tutustuminen ja lain mukaisen aineiston luonti

Minulla oli entuudestaan käsitys Finvoice-standardista ja lain vaatimista pakollisista tiedoista aineistossa. [Lasku, 2016; Finlex, 2019; Finvoice soveltamisohje, Finassivalvonta] Aineistoon tutustuessa huomasin kuitenkin, että alkuperäinen

käsitykseni ei pitänyt täysin paikkaansa ja jouduin päivittämään tietokenttiä. Yllätykseksi paljustui niin ikään se, ettei torimyynnissä ole kuittipakkoa. Jos yritys myisi tuotteitaan pelkästään toritapahtumissa, niin sen ei olisi pakko kirjoittaa asiakkailleen kuitteja. Tämä on tieto, joka voisi vaikuttaa ohjelman rakenteeseen.

Tavoitteiden mukaisesti saavutin tietotason, joka mahdollistaa lain ja standardin mukaisen aineiston luomisen. Finvoicen osalta en aseta itselleni jatkotavoitteita.

5 Tulokset

Tutkimusongelma oli oikean ohjelmistotuotantomallin valinta toimeksiannon tekemiseen. Tutkimusongelman lisäksi tutkielmassa tarkasteltiin PSP-prosessin edistymistä valitun tuotantomallin soveltamisessa. Toimeksianto oli geneerinen ja jätti toteuttajan ratkaistavaksi monia kysymyksiä. Toimeksiannon tarkoituksena oli enemmänkin osoittaa toimittajan taholta kyky suoriutua tehtävästä ja auttaa asiakasta laatimaan tarkempi toimeksianto. Toisen tutkimusongelman muodosti myös mahdollisesti tuleva seuraava toimeksianto, joka pitäisi sisällään kokonaisen ohjelmiston rakentamisen.

Tutkimus keskittyi tunnettujen ohjelmistotuotantomenetelmien tutkimiseen ja niiden vertailuun. Tutkimuksen perusteella muodostettiin kuva menetelmien soveltuvuudesta eri käyttötarkoituksiin. Valitut menetelmät eivät edusta kaikkia menetelmiä. Tutkittavien menetelmien määrää jouduttiin rajaamaan siitä syystä, ettei tutkielmasta olisi tullut liian laajaa.

Esittelymäisen prototyypin toteutukseen valittiin prototyyppimalli. Pienessä toteutuksessa omina iteraatioinaan toteutut prototyypit ovat kevyt ja nopea tapa saada valmis lopputulos. Eri malleja vertaillessa kiinnitettiin huomiota myös koko toteutuksen tekemiseen. Kokonaisen toteutuksen osalta todettiin vastauksena tutkimuskysymykseen, että vesiputousmalli tai scrum olisivat pontentiaalisia toteutusmalleja. Vaikka vesiputousmalli on vanhahtava ja sillä on omat puutteensa, niin mallilla on myös hyviä puolia selkeästi määritellystä projektissa. Vesiputousmalli on selkeä ja jäsennelty, se on myös helppo avata ohjelmistotuotannon ulkopuolisille henkilöille.

Tutkimuksen tuloksissa myös huomattiin, että oikea ohjelmistotuotantomenetelmä riippuu myös toteuttavasta tahosta ja asiakkaasta. Onnistuneen menetelmän valintaan liittyy läheisesti se, kuinka paljon asiakas on valmis projektiin sitoutumaan ja mikä on asiakkaan ohjelmistotuotannon osaamisen taso. Toteuttavaa tahoa taas voi rajoittaa toteuttavan tiimin koko ja resurssit. Sama toteuttaja voi käyttää eri toteutusmalleja eri projekteissa.

6 Pohdinta

Opinnäytetyön tekeminen alkoi ajatuksen tasolla kesän 2017 aikana. Ennen varsinaista aloitusta aloin jo suunnitella tutkielmaa ja mitä se tulee sisältämään. Tärkeänä arvona suoritetun tutkielman lisäksi oli syventää omaa osaamistani erityisesti ohjelmoinnin parissa ja saada sitä kautta hyvä ponnahduslauta työelämään. Varsinainen toteutus käynnistyi keväällä 2018 ja sen yhteydessä tuleva aihepiiri tarkentui. Toteutus oli lyhyellä tauolla kesän 2018 aikana ja jatkui sen jälkeen päätoimisena. Aihepiirinä kirjanpito ja pienen yrityksen taloushallinta on ollut mukana omassa elämässäni yritystoiminnan kautta ja tarjonnut näkökulmia aiheeseen.

Tutkielma muuttui hiukan toteutuksen aikana, joka osaltaa viivästytti sen valmistumista. Jouduin myös rajaamaan aihetta, ettei se muodostunut liian laajaksi. Henkilökohtaisena haasteena koin myös tutkielman opinnäytetyön tieteellisen luonteen. Työn tekeminen on kuitenkin perehdyttänyt minua entisestään alan tieteelliseen kirjallisuuteen ja lähdeviitekäytäntöihin.

Aiheena laskutuksen ja maksuliikenteen automatisointi kiinnostaa minua ennen kaikkea pienyrittäjän näkökulmasta. Pelkästään manuaalisen käsinkirjoituksen varassa olevaa aineistoa on jähmeää käyttää ja pitää sisällään riskin tositteiden hukkaamisesta. Sähköisessä muodossa oleva aineisto on helpompi pitää tallessa ja sitä voidaan käsitellä automaattisesti esimerkiksi kirjanpidon tukena. Tavoitteena oli suunnitella ja osittain toteuttaa ohjelmisto, jolla pienyrittäjä voisi korvata ulkopuolisen kirjanpitopalvelun.

Personal Software Process tarjosi myös mielenkiintoisen tavan mitata ja kehittää omaa osaamistani. Uskon, että jatkossa tulen pitämään menetelmän mielessäni ja soveltamaan sitä omaan osaamiseeni. Soveltaminen ei välttämättä ole ”oppikirjan mukaista”, mutta perusajatus tullee toteutumaan. Oman osaamisen kartoittaminen ja uusien tavoitteiden asettaminen itselleen on hyödyllistä ihan jo yleisenä ohjesääntönä.

Opinnäytetyötä tehdessä kiinnostuin jatkon kannalta erityisesti talon sisäisistä tuotantomalleista. Eri tuotantomallit on kuvattu tarkasti tieteellisessä kirjallisuudessa, mutta niiden soveltaminen käytännössä on toinen asia. Haluaisin myös selvittää, että onko asiakkaalla merkitystä tuotantomallia valittaessa. Ja että korreloiko asiakkaan luottamus toimittajaa kohtaan ketterien mallien käytössä. Asiakashan voi olla myös yritys itse jos kehitys on sisäistä. Koen myös kiinnostavaksi sen, että miten yritys itse tiedostaa käyttämänsä mallit ja keskustellaanko niistä yrityksen sisäisesti. Aihepiiri voisi tarjota hyvän tarttumapinnan esimerkiksi väitöskirjalle.

Opinnäytetyön tekeminen kehitti omaa osaamistani ohjelmoinnin suhteen ja tarjosi paljon uutta teoriaa tuttujen asioiden takana. Erityisesti syvempi tutustuminen eri tuotantomalleihin tulee varmasti olemaan hyödyllistä tulevaisuudessa.

7 Lähteet

[Abrahamsson et al., 2002] Pekka Abrahamsson, Jussi Ronkainen, Outi Salo, Agile Software development methods. Tutkimukset ja selvitykset, 2002. VTT publications Otamedia Oy, Espoo, 2002.

[Beck, 1999] Kent Beck, Extreme Programming Explained – Embrace Chance. Addison-Wesley, 1999.

[Beck et al., 2001] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cocburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas. Ketterien menetelmien manifesti. *https://agilemanifesto.org/* Haettu 1.2.2019.

[Duggal & Suri, 2008] Gaurav Duggal, Bharti Suri. Understanding Regression Testing Techniques. *http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.460.5875&rep=rep1&type=pdf* Haettu 26.5.2019.

[Finanssivalvonta, 2019] Finvoicen soveltamisohje. *http://www.finanssiala.fi/finvoice/dokumentit/Finvoice_3_0_soveltamisohje.pdf* Haettu 1.2.2019.

[Finlex, 2013] Laki kuitintarjoamisvelvollisuudesta. *https://www.finlex.fi/fi/laki/alkup/2013/20130658* Haettu 15.1.2019.

[Haikala & Märijärvi, 2006] Ilkka Haikala, Jukka Märijärvi, Ohjelmistotuotanto, Gummerus Kirjapaino Oy, Jyväskylä, 2006.

[Huttunen, 2006] Janne Huttunen. Ketterän ohjelmistokehityksen määrittely, vertailu ja käyttäjäkysely. Diplomityö, Tekninen korkeakoulu, 2006.

[Pan, 1999] Jiantao Pan, Software Testing, Carnegie Mellon University, 1999.

[Korhonen, 2014] Langattoman verkon signaalin parantaminen. *https://www.tivi.fi/Vinkit/2014-10-02/N%C3%A4in-parannat-kotona-langattoman-verkon-signaalia-3149133.html* Haettu 2.3.2019.

[Kornee, 2017] Top-Down -ohjelmointi *https://dzone.com/articles/how-does-top-down-programming-work* Haettu 1.2.2019.

[Kniberg & Skarin, 2010] Henrik Kniberg, Mattias Skarin, Kanban and Scrum – Making the Most of Both, C4Media Inc, USA, 2010.

[Lasku, 2016] Laskutusvaatimukset arvonlisäverotuksessa. *https://www.vero.fi/syventavat-vero-ohjeet/ohje-hakusivu/48090/laskutusvaatimukset_arvonlisaverotukses3/* Haettu 22.2.2019.

[Lindberg, 2003] Harri Lindberg. Extreme Programming, Tampereen yliopisto, Tietojenkäsittelytieteiden laitos, Tietojenkäsittelyoppi, Pro gradu -tutkielma, 2003.

[Nielsen, 1994] Jacob Nielsen. Ten Usability Heuristics. *http://www.useit.com/papers/heuristic/heuristic_list.html* Haettu 1.3.2019.

[Ould & Unwin, 1986] Martyn A Ould, Charles Unwin. Testing in Software Development. University of Cambridge. Press Syndicate, 1986.

[Pomeroy-Huff et al., 2009] Marsha Pomeroy-Huff, Robert Cannon, Timothy A. Chick, Julia Mullaney, William Nichols. The Personal Software Process (PSP) Body of Knowledge, Version 2.0. Pittsburgh, Pennsylvania: Software Engineering Institute, Carnegie Mellon University, 2009.

[Poppendieck & Poppendieck, 2006] Mari Poppendieck, Tom Poppendieck, Implementing Lean Software Devolopment: From Concept to Cash. Addison-Wesley, 2006.

[Schwaber, 2004] Schwaber Ken. Agile project management with Scrum. Redmond Washington. Microsoft Press, 2004.

[Schwaber & Beedle, 2002] Ken Schwaver, Mike Beedle, Agile Software Development with Scrum. Prentice Hall, 2002.

[Stapleton, 1997] Jennifer Stapleton, DSDM: Dynamic Systems Development Method: The Method in Practice, Addison-Wesley Professional, 1997.

[Vänskä, 2017] Epäonnistuneet IT-projektit. *https://www.tivi.fi/Kaikki_uutiset/paholaisen-tusina-13-epaonnista-it-projektia-naihin-poltettiin-suomessa-miljoonia-6679372* Haettu 1.3.2019.

[Weibgerber & Diehl, 2006] Peter Weibgerber, Stephan Diel, Identifying Refactoring from Source-Code Changes, University of Trier, Germany, 2006.

[Winston, 1970] Royce Winston. Managing the Development of Large Software Systems. *Proceedings of IEEE WESCON 26 (August): 1-9, s.328-s.338,* 1970.

[Womack et al., 1990] James P. Womack, Daniel T. Jones, Daniel Roos. The Machine That Changed the World: The Story of Lean Production. Rawson Associates, New York, USA, 1990.

8 Liitteet

8.1 Asiakastietokuvaus

Asikasid, asiakkaan yksilöivä id
Etunimi, asiakkaan/yhteyshenkilön etunimi
Sukunimi, asiakkaan/yhteyshenkilön etunimi
Osoite, asiakkaan/yhteyshenkilön osoite
Postinumero, asiakkaan/yhteyshenkilön postinumero
Postitoimipaikka, asiakkaan/yhteyshenkilön postitoimipaikka
Puhelinnumero, asiakkaan/yhteyshenkilön puhelinnumero
Sposti**,** asiakkaan/yhteyshenkilön sähköposti
yrityksen_nimi, yrityksen nimi
yrityksen_osoite, yrityksen katuosoite
yrityksen_postinumero, yrityksen postinumero
yrityksen_postitoimipaikka, yrityksen postitoimipaikka
y-tunnus, y-tunnus
verkkolaskutunnus, verkkoalaskutunnus
operaattori, verkkolaskuoperaattori

8.2 Maksun tietokuvaus

Otsikko:

Maksuid, maksun yksilöivä id. Kirjain L tarkoittaa laskua ja K kuittia. Erittelevän kirjaimen jälkeen _järjestysnumero.
Asiakasid, asiakkaan yksilöivä id (haetaan tietokannasta)
Paivays, dd.mm.yyyy
Erapaiva, vakiona 2 viikkoa eteenpäin päiväyksestä
Huomautusaika, vakiona 7 päivää

Rivi:

Rivinumero, juokseva numerointi
Tuote, sanallinen kuvaus tuotteesta
Lisatiedot, sanallinen kuvaus
Maara, tuotteiden lukumäärä
Yksikkohinta, hinta per määrä
Verollinen_hinta, hinta sis alv
Verokanta, arvonlisävero prosentti
Veron_maara, arvonlisävero euroissa
Veroton_hinta,

Summarivi:

Summarivi, tunniste S

Yhteenlaskettu_maara, kaikkien rivien yhteenlaskettu maara

Yhteenlaskettu_verollinen_hinta, kaikkien rivien yhteenlaskettu verollinen hinta

Yhteenlaskettu_veron maara, kaikkien rivien yhteenlaskettu veron määrä

Yhteenlaskettu_veroton_hinta, kaikkien rivien yhteenlaskettu veroton hinta