PROTOTYYPIT JA VAATIMUSMÄÄRITTELY SOVELLUSKEHITYKSESSÄ

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Ammattikorkeakoulun opinnäytetyö

Tietojenkäsittelyn koulutusohjelma

Visamäki, kevät 2015

Veli-Matti Mustonen

HAMK
HÄMEEN AMMATTIKORKEAKOULU

TIIVISTELMÄ

VISAMÄKI
Tietojenkäsittelyn koulutusohjelma
Systeemityö

Tekijä Veli-Matti Mustonen **Vuosi** 2015

Työn nimi Prototyypit ja vaatimusmäärittely sovelluskehityksessä

TIIVISTELMÄ

Ohjelmistojen määrittelyssä käytetään usein raskaita ja vaikeaselkoisia kirjallisia vaatimusmäärittelydokumentteja. Tämänkaltaiset dokumentit koetaan epäkäytännöllisiksi ja vaikeasti ymmärrettäviksi. Tässä työssä tähän ongelmaan etsittiin ratkaisua prototyyppien hyödyntämisestä vaatimusmäärittelyssä ja ohjelmointikehityksessä. Työn toimeksiantajana toimi vuonna 2010 perustettu ohjelmointiin erikoistunut Valo Interactive Finland Oy. Työn tarkoituksena oli etsiä toimeksiantajalle uusia näkökulmia vaatimusmäärittelyyn ja ohjelmointikehitykseen.

Tapaustutkimuksessa etsittiin yksityiskohtaista teoria- ja tutkimustietoa alan kirjallisuudesta sekä internetistä. Lisäksi toteutettiin muutamalle alan ammattilaiselle teemahaastattelu, jonka avulla selvitettiin heidän ajatuksiaan vaatimusmäärittelystä sekä prototyyppien hyödyntämisestä käytännössä. Aineistosta muodostettiin kirjallinen tiivistelmä, missä haastateltujen näkemykset yhdistettiin kirjallisuudesta löytyneisiin havaintoihin.

Opinnäytetyön tuloksena prototyypitys todettiin toimivaksi työvälineeksi kommunikoinnissa ja uusien ratkaisuiden innovoinnissa. Prototyypitys ei kuitenkaan yksinään riitä ratkaisemaan vaatimusmäärittelyn ja ohjelmistotuotannon ongelmia. Ohjelmistotuottajalla on vastuu omien työskentelytapojensa kehittämisestä, ja hänen tulee etsiä itselleen parhaiten sopivat ohjelmistotuotannon käytännöt. Asiakaslähtöisillä vaatimusmäärittelykäytännöillä on merkittävä rooli määrittelyjen keräämisprosessin onnistumisessa. Vaatimusmäärittelyn dokumentointiin olisi syytä käyttää raskaiden kirjallisten dokumenttien sijaan erilaisia helposti ymmärrettäviä ketteriä menetelmiä. Tarkka vaatimusten kirjallinen dokumentointi on syytä rajoittaa vain sitä välttämättä vaativiin toimintoihin. Asiakaslähtöisistä vaatimusten keräämiskäytännöistä ja niiden hyödyntämisestä käytännössä voisi tehdä oman opinnäytetyönsä.

Avainsanat Prototyypit, ohjelmistokehitys, vaatimustenhallinta

Sivut 35 s.

VISAMÄKI
Degree Programme in Business Information Technology
Programming and Software Engineering

Author Veli-Matti Mustonen **Year** 2015

Subject of Bachelor's thesis Prototypes and requirements analysis in software development

ABSTRACT

Software specification uses complicated requirement analysis documents that are often found impractical and difficult to understand. This Bachelor's thesis was commissioned by the company Valo Interactive Finland Oy which specializes in programming. The purpose of the thesis was to find a solution to the problem by using prototypes in software requirements analysis and to provide new perspectives on requirement analysis and software development for the company.

The thesis was carried out as a case study. Theoretical information and research data were searched for in the literature of the field and the Internet. In addition, a few professionals were interviewed to find out their views on using requirement specification and the use of prototypes in practice. The results of the interviews were compared with the material collected from the literature and the Internet.

The results of the thesis show that prototyping is a functional a tool for communicating and innovating new solutions. Prototyping alone is an insufficient solution to solve the problems in requirements specification and software development. The software providers have the responsibility of improving their own working practices and finding the best suitable practices for themselves. Customer-oriented requirements specification practices play a significant role in the success of the process. Specification documentation should use straightforward and agile methods. Accurate written documentation should only be used when necessarily required. Further studies could be conducted on the practices of collecting customer-oriented requirements and their use in practice.

Keywords Requirement analysis, programming, software development
Pages 35 p.

SISÄLLYS

1 JOHDANTO 1

2 VAATIMUSMÄÄRITTELYN VAIKEUS 2

2.1 Toimeksiantaja 2
2.2 Tutkimusmenetelmä 3
2.3 Tutkimuksen kulku ja aineiston analysointi 3
2.4 Haastattelujen luotettavuus 4

3 PROTOTYYPPI OHJELMISTOKEHITYKSESSÄ 5

3.1 Ohjelmiston elinkaari ja vaihejakomallit 5
3.2 Vesiputousmalli vs. ketterät menetelmät 6
3.3 Prototyyppi ja vaihejakomalli 7
3.4 Prototyypitysprosessi 8
3.4.1 Esitutkimus ja määrittelyvaihe 9
3.4.2 Suunnitteluvaihe 9
3.4.3 Ohjelmointivaihe 10
3.4.4 Evolutiivisen prototyypin testausvaihe ja palautteen keräys 10
3.4.5 Käyttöönotto 11
3.4.6 Ylläpito 11
3.4.7 Prototyypitysprosessin laadunvalvonta 11
3.5 Prototyyppi työvälineenä 14
3.5.1 Prototyyppi vuorovaikutuksen välineenä 14
3.5.2 Prototyyppi ja innovointi 15
3.6 Ohjelmistokehityksen prototyypit 15
3.6.1 Ohjelmointikehityksen prototyyppien ryhmittelystä 16
3.6.2 Prototyypitystavan valinta 16
3.6.3 Prototyyppi ja kustannukset 17
3.6.4 Paperiprototyypit 17
3.6.5 Esitysgrafiikkaprototyypit 20
3.6.6 Linkitetty PDF-prototyyppi 22
3.6.7 HTML-prototyypit 22

4 HAASTATTELUTUTKIMUKSEN TULOKSET 24

4.1 Haastattelun terminologian tunteminen 24
4.2 Kokemuksia vaatimusmäärittelystä 24
4.3 Käytännön esimerkkejä vaatimusmäärittelyn epäonnistumisista 26
4.4 Kokemuksia ilman vaatimusmäärittelyä ohjelmoinnista 27
4.5 Kokemuksia prototyypeistä sovelluskehityksessä 27

5 POHDINTA 29

6 YHTEENVETO 32

LÄHTEET 34

1 JOHDANTO

Julkisen hallinnon tietohallinnon neuvottelukunnan (JUHTA) suosituksissa vaatimusmäärittelystä todetaan seuraavaa: ”Riittämätön vaatimusten määrittely on yleisin yksittäinen syy ohjelmistoprojektien epäonnistumiseen.” Suosituksissa viitataan tutkimuksiin, joiden mukaan kaikista epäonnistuneista projekteista yli 75 prosentissa vaatimusmäärittely on ollut puutteellinen. (JUHTA 2009b, 9.)

Ohjelmistotuotteen vaatimusten määritteleminen on siis haasteellinen prosessi, ja sen epäonnistumiseen voidaan löytää monia syitä. Tyypillisesti tuotteen ominaisuudet pyritään kuvaamaan mahdollisimman kattavasti etukäteen käyttäen epäformaaleja sanallisia kuvauksia, jotka voidaan helposti tulkita väärin. Projektin alussa tuotteen todellisista vaatimuksista ei kuitenkaan ole välttämättä täysin selkeää käsitystä ja vaatimukset muuttuvat projektin edetessä. Usein vaatimuksia ei kirjata riittävän tarkasti tai jotain oleellista jää kirjaamatta. Ongelmia voi aiheuttaa myös se, että vaatimusten kerääjät ja käyttäjät eivät ymmärrä toisiaan. Tilaaja on useimmiten joku muu kuin varsinainen loppukäyttäjä, ja tilaajan käsitys järjestelmän vaatimuksista voi poiketa todellisten loppukäyttäjien vaatimuksista. Tuloksena on helposti puutteellinen, ristiriitainen tai helposti väärin ymmärrettävä määrittely ja sitä kautta ei-toivottu lopputulos.

Lisäksi sanallisten kuvausten päivittäminen on havaittu raskaaksi. Monet kokevat täten vaatimusmäärittelyn vaikeaksi tai turhaksi ja jättävätkin vaatimusmäärittelyn usein kokonaan tekemättä tai tekevät sen huolimattomasti. Toisaalta vaatimusmäärittely voidaan kokea tärkeäksi, mutta sen seurauksena järjestelmä määritellään liian tarkasti tai huomattavasti yli todellisten tarpeiden. Tämän seurauksena järjestelmään tulee toimintoja, joita ei tarvita ja näin kustannukset nousevat.

Menetelmät vaatimusten keräämiseen ja dokumentointiin voivat olla hyvinkin puutteellisia. Vaatimuksia ei monesti testata riittävästi tai lainkaan, vaan luotetaan niiden olevan kelvollisia testaamattakin. Lisäksi mikäli määrittelyä ei mielletä oleelliseksi osaksi projektia, myös siihen varatut resurssit saatetaan mitoittaa alakanttiin. Puutteellisten vaatimusmäärittelyiden syntyminen johtunee lähinnä puutteellisista työskentelytavoista ja käytännöistä. (JUHTA 2009b, 9; Ruohotie, haastattelu 31.3.2014.)

Työn toimeksiantajana toimii vuonna 2010 perustettu ohjelmointiin erikoistunut Valo Interactive Finland Oy. Opinnäytetyön tavoitteena on etsiä Valo Interactivelle uusia näkökulmia ohjelmointikehitykseen ja mahdolliseksi hyödyksi katsotaan omien toimintatapojen kehittäminen.

Opinnäytetyön tutkimuskysymykset:
1. Miten prototyyppejä hyödynnetään ohjelmointikehityksessä?
2. Voidaanko prototyyppejä hyödyntää vaatimusmäärittelyssä ja miten?
3. Miten prototyyppi helpottaa ohjelmoijan työtä jatkossa?

2 VAATIMUSMÄÄRITTELYN VAIKEUS

Opinnäytetyön idea syntyi aikanaan harjoittelujakson aikaisen työnantajan kanssa käytyjen ”kahvipöytäkeskustelujen” pohjalta. Keskusteluissa pohdittiin usein sitä, miksi ohjelmistojen määrittelyssä käytetään niin usein raskaita ja vaikeaselkoisia kirjallisia vaatimusmäärittelydokumentteja. Tämän kaltaiset vaatimusmäärittelyt koettiin hankaliksi ja vaikeasti ymmärrettäviksi. Tämän lisäksi havaittiin niiden olevan usein sisällöltään hyvin puutteellisia ja ristiriitaisia. Välillä jopa kyseenalaistettiin koko määrittelyjen tekemisen tarpeellisuuskin. Keskusteluissa päädyttiin kuitenkin lopulta siihen johtopäätökseen, että ohjelmointiprojekteissa on poikkeuksetta hyvä olla jonkinlainen määrittely. Vaatimusmäärittely ei siis itsessään voinutkaan olla ongelma, vaan ongelman täytyi olla tavassa tehdä vaatimusmäärittelyitä. Havaittiin, että ammattikorkeakoulussa saadussa opetuksessa oli käsitelty kattavasti ainoastaan kirjallisten vaatimusmäärittelyn tekemistä ja todettiin opetuksen olleen näin ollen puutteellista. Puutteellisen opetuksen katsottiin olleen osasyy negatiiviseen suhtautumiseen vaatimusmäärittelyitä kohtaan.

Keskusteluissa pohdittiin myös sitä, mitä voitaisiin tehdä toisin niin, että vaatimusmäärittely ja ohjelmointikehitys voisivat onnistua nykyistä paremmin. Heräsi kysymys, voisiko prototyypeistä olla apua ohjelmistojen määrittelyssä tai kehittämisessä. Hyvin nopeasti esimerkkitapausten kautta huomattiin, että prototyyppien hyödyntämisestä on havaittu olevan hyötyä sovelluskehityksessä tai määrittelyiden tekemisessä.

Keskusteluiden pohjalta heräsi aihetta kohtaan laajempi mielenkiinto, joka myöhemmin jalostui opinnäytetyön aiheeksi. Täten opinnäytetyön aiheeksi muodostui: prototyypit ja vaatimusmäärittely sovelluskehityksessä. Opinnäytetyössä pyritään selvittämään, miten prototyypitystä tehdään ja olisiko prototyypeistä apua myös vaatimusmäärittelyssä. Samalla tutkitaan tekstimuotoisten vaatimusmäärittelyiden ongelmia. Opinnäytetyön mahdolliseksi hyödyksi nähtiin uusien ideoiden ja näkökulmien löytyminen, joiden perusteella voitaisiin mahdollisesti kehittää yrityksen omaa ohjelmointikehitysprosessia.

2.1 Toimeksiantaja

Työn toimeksiantajana toimii vuonna 2010 perustettu ohjelmointiin erikoistunut Valo Interactive Finland Oy. Yritys kehittää asiakkailleen räätälöityjä järjestelmiä ja sovelluksia, sulautetuista järjestelmistä laajoihin verkkojärjestelmiin.(Ruohotie, haastattelu 4.2.2015.) Opinnäytetyön valvojana toimeksiantajan puolelta toimi yrityksen osa-omistaja ja johtohenkilö Juhani Ruohotie. Hänen kattavalla asiantuntemuksellaan ja näkemyksillään oli suuri vaikutus opinnäytetyön sisältöön.

2.2 Tutkimusmenetelmä

Tutkimuksen lähtökohtana oli olettamus siitä, että prototyypityksestä voisi olla hyötyä sovelluskehityksessä. Tämän olettamuksen todenperäisyyttä selvitettiin laadullisen tapaustutkimuksen näkökulmasta. Tapaustutkimukselle on luonteenomaista tuottaa valitusta kokonaisuudesta intensiivistä ja yksityiskohtaista tietoa (Saaranen-Kauppinen & Puusniekka 2006).

Tässä tapaustutkimuksessa etsittiin teoria- ja tutkimustietoa prototyypityksestä tietojenkäsittelyalan kirjallisuudesta sekä internetistä. Löydettyä teoriatietoa tutkimalla pyrittiin saamaan selville, miten prototyyppejä hyödynnetään sovelluskehityksessä nykypäivänä. Tämän lisäksi toteutettiin muutamalle alan ammattilaiselle teemahaastattelu. Teemahaastattelun avulla pyrittiin selvittämään jo alalla työskentelevien omia ajatuksia ja kokemuksia vaatimusmäärittelystä sekä prototyyppien hyödyntämisestä sovelluskehityksessä.

Teemahaastattelu on puolistrukturoidun haastattelun malli, jossa haastattelijan etukäteen valitsemat teemat sitovat haastattelussa käydyn keskustelun tutkimusongelmaan. Puolistrukturoidun haastattelun kysymykset ovat kaikille samat, mutta haastattelun vastaajat voivat vastata kysymyksiin omin sanoin. Teemahaastattelun katsottiin olevan hyvä tapa kerätä tietoa, koska näin haastateltavien ”ääni” pääsee paremmin kuuluviin. (Hirsjärvi & Hurme 2011, 18, 35, 47–48.)

2.3 Tutkimuksen kulku ja aineiston analysointi

Haastatteluihin pyydettiin kahdeksaa alalla työskentelevää henkilöä, joista haastatteluun suostui kolme. Näistä kolmesta haastattelusta saatiin kuitenkin opinnäytetyön kannalta riittävästi erittäin täsmällistä ja asiantuntevaa tutkimusmateriaalia. Lisäksi alun perin tavoitteena oli haastatella muutama henkilö myös asiakkaan roolissa, jotta saataisiin käsitys prototyypityksen merkityksestä asiakkaalle. Haastatteluun löydettiin yksi asiakkaana toiminut henkilö. Asiakkaan näkemyksillä katsottiin olevan vähäpätöinen merkitys itse työn tuloksiin. Asiakkaan haastattelusta tehdyt havainnot tukivat täysin muita opinnäytetyössä esitettyjä näkemyksiä ja alan kirjallisuudesta löydettyjä tutkimustuloksia. Kirjallisuudesta löytyneiden lähteiden katsottiin olevan merkityksellisemmässä roolissa ja tämän vuoksi asiakkaan haastattelun aineisto jätettiin lähes kokonaan pois tästä opinnäytetyöstä.

Haastatteluissa syntynyttä materiaalia verrattiin kirjallisuudesta ja eri lähteistä löytyneisiin tutkimuksiin ja havaintoihin. Aineiston havaittiin olevan sopusoinnussa kirjallisuudesta ja internetistä löytyneiden lähteiden kanssa. Aineistosta muodostettiin kirjallinen tiivistelmä, missä haastateltujen näkemykset yhdistettiin kirjallisuudesta löytyneiden havaintojen kanssa. Haastattelututkimuksen tuloksia ei olisi voinut pitää luotettavana pienestä otannasta johtuen, mikäli alan kirjallisuudesta tehdyt havainnot eivät olisi olleet sopusoinnussa haastattelujen tuottaman materiaalin kanssa.

2.4 Haastattelujen luotettavuus

Opinnäytetyön haastatteluilla kerättiin tietoa, jonka painoarvo oli henkilöiden omien kokemusten ja näkemysten kirjaamisessa. Haastatteluissa kerätyn tiedon oikeellisuuden varmistamisen katsottiin olevan erittäin tärkeää. Haastattelussa pyrittiin tästä syystä varmistamaan, että haastattelija ja haastateltava olivat ymmärtäneet asiat samalla tavalla. Haastattelun alussa haastatelluilta varmistettiin lyhyen kyselyn avulla, että he tuntevat haastattelussa käytettävät termit.

Haastattelussa kerätyn aineiston purkamisen ja analysoinnin jälkeen haastateltavat tarkistivat vielä opinnäytetyön sisällön. Joitakin lausuntoja tarkennettiin ja korjattiin. Näillä toimenpiteillä pyrittiin varmistamaan, ettei purkamisen ja analysoinnin aikana mahdollisia haastattelijan tekemiä virheellisiä päätelmiä ja tulkintoja haastateltujen ajatuksista päätyisi valmiiseen opinnäytetyöhön.

3 PROTOTYYPPI OHJELMISTOKEHITYKSESSÄ

Prototyypillä tarkoitetaan yleisesti ensimmäistä versiota jostakin tuotteesta, mitä on tarkoitus testata. Testaamisen aikana tuotteessa havaitut puutteet korjataan ennen virallista tuotteen julkaisemista. (Effective prototyping 2007, 3.)

Ohjelmointikehityksessä prototyyppi on usein näennäisesti tai alkeellisesti toimiva versio tulossa olevasta ohjelmasta, järjestelmästä tai niiden osasta. Tyypillisesti prototyyppi tehdään järjestelmän demonstraatiota varten osana kehitysprosessia. Prototyypin avulla testataan tuotteen toiminnallisuutta, tarkoituksenmukaisuutta ja käytettävyyttä. Prototyyppiä testataan ja jatkokehitetään usein sykleissä, kunnes testauksessa ei enää löydy korjattavaa. Prototyypin avulla pyritään selvittämään, onko lopputuotteesta tulossa toimiva ja halutunkaltainen. Prototyypityksen lopputuloksena voidaan saada selville jopa se, ettei tuotetta kannata lainkaan toteuttaa, mikä säästää aikaa ja rahaa. (Sommerville 2001, 172; Tech Target 2005.)

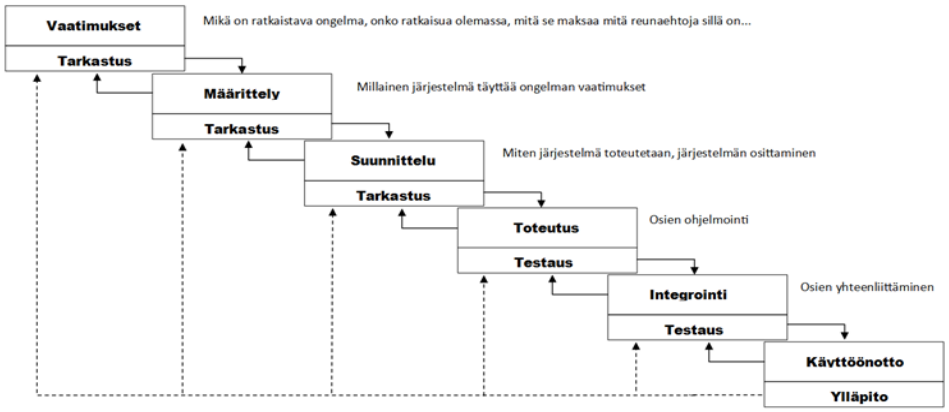
Prototyyppi mielletään ohjelmistotuotannossa usein koodatuksi sovelluksen tai järjestelmän prototyypiksi, vaikka todellisuudessa prototyypitystä voidaan tehdä ilman riviäkään oikeaa koodia. Tässä opinnäytetyössä prototyyppi mielletään Richard Munozin (1992) artikkelissa In search of the ideal prototype olleen Bill Verplankin määritelmän mukaisesti: “Prototyypitys on järjestelmän idean ilmaisemista ja konkreettiseksi tekemistä arviointia varten”. Määrittelyn katsotaan antavan prototyypille huomattavasti laajemman merkityksen kuin yleisesti ajatellaan. (Effective prototyping 2007,4.)

Prototyypit nähdään tässä opinnäytetyössä ilmaisutavoista riippumattomiksi konkreettisiksi pyrkimyksiksi kuvata ohjelmiston rakennetta ja toimintoja tavoilla, jotka mahdollistavat kehitystiimille ohjelmiston testaamisen ja tutkimisen. Prototyyppi voi tämän määrittelyn mukaisesti yksinkertaisimmillaan olla paperille tehty luonnos järjestelmän keskeisimmistä toiminnoista, listaus järjestelmän vaatimuksista tai monimutkaisimmillaan lähes käyttövalmis sovellus rikkaalla vuorovaikutuksella ja visuaalisella palautteella. (Effective prototyping 2007, 10.) Prototyypiksi luetaan siis mikä tahansa yritys esittää mitä tahansa osaa ohjelmistosta, lähes millä tahansa tavalla. Prototyyppi nähdään siis varsin laajana käsitteenä, jonka tarkoituksena on edesauttaa ohjelmointiprojektin onnistumista. (Haikala & Märijärvi 2004, 42.)

3.1 Ohjelmiston elinkaari ja vaihejakomallit

Ohjelmistotuotannossa puhutaan usein ohjelmiston elinkaaresta. Tällä tarkoitetaan aikaa, mikä kuluu ohjelmiston kehittämisestä siihen, kun se havaitaan tarpeettomaksi ja poistetaan käytöstä. Ohjelmiston elinkaari jaetaan ohjelmistotuotannossa usein pienempiin osiin. Vaihejakomalliksi kutsutaan tapaa jolla osittaminen tehdään.

Vaihejakomalleista tehdään useimmiten jonkinlainen kaavio, jonka avulla niitä on helpompi ymmärtää. Kaikkein käytetyin ja tunnetuin vaihejakomalli lienee vesiputousmalli (kuvio 1) (Haikala & Märijärvi 2004, 36).



Kuvio 1. Vesiputousmalli esimerkki vaihejakomalleista (Haikala & Märijärvi 2004 ,36).

Vaihejakomalleihin ohjelmistotuotannossa kuvataan yleensä työnkulun tärkeimmät osa-alueet. Se, miten nämä osa-alueet nähdään ja koetaan, vaihtelevat tapauskohtaisesti hyvinkin paljon. Vaihejakomalleihin kuvataan useimmiten seuraavat vaiheet: määrittely, suunnittelu, ohjelmointi, testaus, käyttöönotto ja ylläpito. Joissakin vaihejakomalleissa nämä osa-alueet on pilkottu vielä pienempiin ja yksityiskohtaisempiin osasiin.

3.2 Vesiputousmalli vs. ketterät menetelmät

Perinteisissä vaihejakomalleissa, kuten vesiputousmallissa olennaista on tarkka ja huolellisesti laadittu vaatimusmäärittely, jonka perusteella ohjelmisto tuotetaan. Ruohotien (haastattelu 27.4.2014) mukaan vesiputousmallissa pyritään määrittelemään sovellus tarkasti ja määrittelyn pohjalta projektille lasketaan aikataulu ja hinta. Määrittely lukitaan ennen toteutusta, ja tästä syystä määrittelyssä pyritään täydellisyyteen. Heikkoutena tässä on se, että ihminen on huono ennustamaan tulevaisuutta, eikä itsekään aina tiedä mitä haluaa. Tämän lisäksi harva osaa määritellä vaatimukset formaalilla tavalla niin, ettei niitä voi ymmärtää väärin.

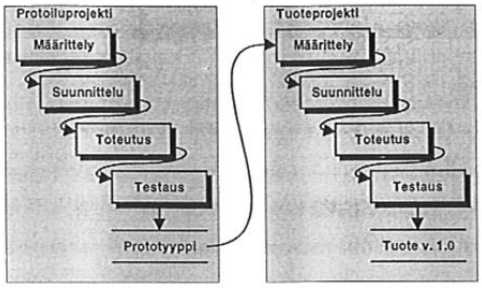
Perinteisen vesiputousmallin mukainen kehittäminen on hidasta täydellisyyteen pyrkivän vaatimusmäärittelyn takia. Käytännössä tämä tekee mallin käyttämisestä lähes mahdotonta. Ympäröivä maailmamme ja sen markkinat muuttuvat nopeasti ja prosessin edetessä hitaasti saatetaan havaita tuotteen olevan vanhentunut jo ennen kuin se valmistuu. Muuttuneeseen markkinatilanteeseen vastaaminen vesiputousmallissa vaatisi tällöin aina uuden määrittelykierroksen. Tämä voi johtaa pahimmillaan tilanteeseen, missä määrittelyä päivitetään jatkuvasti ja mitään ei saada koskaan valmiiksi. (Haikala & Märijärvi 2004, 41, 92; Ruohotie, haastattelu 31.3.2014.)

Ketterillä menetelmillä pyritään ratkaisemaan nämä vesiputousmallin ongelmat. Ketterille menetelmille on ominaista pyrkiä saamaan ohjelmiston toteutus liikkeelle nopeasti ja ohjelmiston on tarkoitus kehittyä matkan varrella. Ketterä ohjelmistokehitys on suunniteltu korvaamaan ja täydentämään perinteisiä jäykkiä ohjelmistotuotantomalleja ja -prosesseja. Toisin kuin perinteisessä vesiputousmallissa, ketterän kehityksen malleissa vaatimusmäärittelyn odotetaankin muuttuvan markkinoiden mukaan prosessin edetessä. (Sininen Meteoriitti 2011.)

Ketteryyden lisääntyessä on ilmeistä, että raskaat kirjalliset määrittelydokumentit eivät sovellu sellaisenaan ohjelmiston ominaisuuksien määrittelyyn. Tilalle tarvitaan kevyempiä ja huomattavasti varmempia määrittelymenetelmiä.

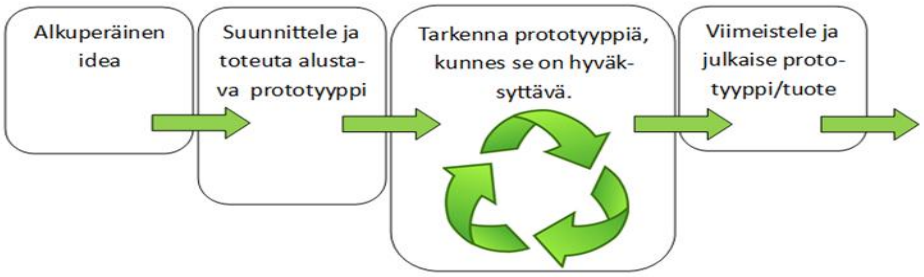
3.3 Prototyyppi ja vaihejakomalli

Vaihejakomalleja käytetään myös prototyypitysprosessien osittamiseen, tehtiinpä prototyypitystä perinteisen vesiputousmallin mukaisesti tai erilaisilla ketterillä menetelmillä. Ohjelmistojen ja prototyyppien kehittämiseen tähtäävät vaihejakomallit ovat joka tapauksessa useimmiten hyvin samankaltaisia. Tämä johtunee siitä että työn kulku ohjelmistotuotannossa on luonnollisesti hyvin samankaltaista, oli sitten kyse ohjelmistonprototyypistä tai varsinaisesta ohjelmistosta. (Haikala & Märijärvi 2004, 42.)



Kuvio 2. Esimerkki prototyypitys mallista (Haikala & Märijärvi 2004, 42).

Prototyypityksen vaihejakomallit jaetaan kahteen ryhmään: poisheitettäviin (kuvio 2) ja evolutiivisiin (kuvio 3). Evolutiivisissa malleissa prototyyppi kehitetään lopputuotteeksi asti, ja poisheitettävissä malleissa prototyyppi toimii vain mallina ja varsinainen tuote kehitetään prototyypin perusteella. (Haikala & Märijärvi 2004, 42; Sommerville 2001, 175–180.)



Kuvio 3. Esimerkki evolutiivisesta prototyyppivaihejakomallista (McConnell 2002, 434).

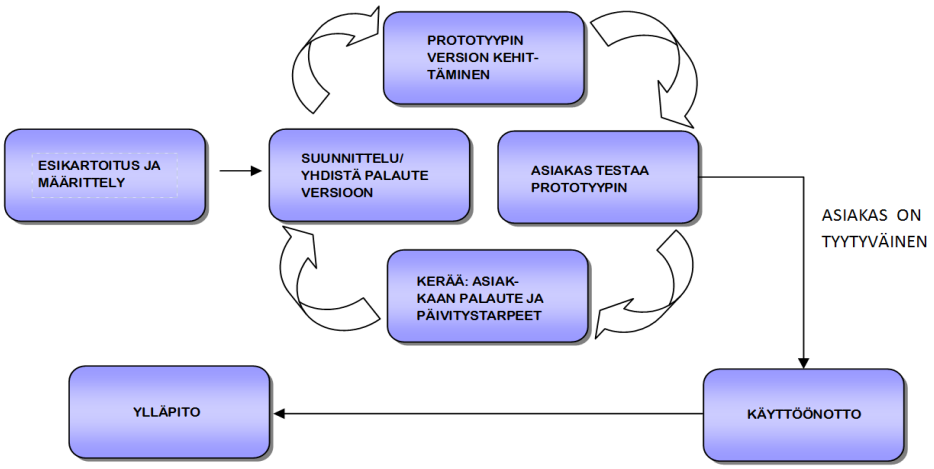
3.4 Prototyypitysprosessi

Kuten aikaisemmin totesin, prototyypitysprosessi ei eroa juuri millään tavalla ohjelmointiprosessista. Keskeisin ero prototyypitysprosessin ja ohjelmointiprosessin välillä lienee siinä, että prototyypitysprosessin tarkkuutta ja käyttötarkoitusta voidaan vapaasti muuttaa ja skaalata omien tarpeiden mukaan. Prototyypityksen avulla voidaan pyrkiä esimerkiksi vain selvittelemään asiakkaan todellisia vaatimuksia järjestelmälle tai testata tulossa olevaa järjestelmää tai sen osaa jollakin tavalla ennen sen todellista kehittämistä (Haikala & Märijärvi 2004, 42).

Prototyyppiprosessi voidaan siis skaalata koskettamaan ohjelmistoprojektin pientä osa-aluetta, jota on tarpeen testata ennen varsinaisen ohjelmistoprojektin aloittamista. Mitä pienemmästä osa-alueesta on kyse, sitä vähemmän prosessi muistuttaa mitään ohjelmistotuotannon prosessia. Yksinkertaisimmillaan prototyypitysprosessi onkin vain ideoiden luonnostelua paperille, jonka pohjalta aloitetaan myöhemmin oikea ohjelmointiprosessi. (Effective prototyping 2007, 21.)

Prototyypitysprosessin on tarkoitus olla ennen kaikkea oppimisen ja ideoinnin väline, jonka lopputuloksena on tarkoitus syntyä jotakin käyttökelpoista. Jokaisessa prototyypitysprosessin syklissä tuotetta on tarkoitus kehittää eteenpäin. Tässä prosessissa ilman asianmukaista laadunvalvontaa voidaan päätyä tilanteeseen, missä prototyyppiin päätyy jatkuvasti lisää turhia toimintoja. Tämän prototyypitykselle luonteenomaisen ongelman ratkaisemiseksi on hyvä varmistaa, että prototyyppiin päätyvät ideat ovat syntyneet todellisista vaatimuksista ja tarpeista. On hyvin tärkeää että turhat ominaisuudet karsiutuvat pois vain oleellisten jäädessä jäljelle. Tätä prototyypitysprosessin ilmeistä riskiä, voidaan vähentää laadunvalvonnalla. (Effective prototyping 2007, 10, 31.).

Tässä opinnäytetyössä sivutaan prototyypitysprosessin vaiheita ketteriin menetelmiin kuuluvan evolutiivisen prototyypitysprosessin näkökulmasta (kuvio 4), minkä lopputuloksena syntyy valmis käytössä oleva ja ylläpidettävä järjestelmä.



Kuvio 4. Opinnäytetyössä käsiteltävä evolutiivisen prototyypityksen vaihejakomalli. Kuvio on tehty soveltaen McConnellin (2002, 427) evolutiivisen toimituksen elinkaarimallia.

3.4.1 Esitutkimus ja määrittelyvaihe

Esitutkimusvaihe tai tarvekartoitukseksi kutsuttu vaihe edeltää joissakin vaihejakomalleissa määrittelyvaihetta. Tämä vaihe on useimmissa vaihejakomalleissa kuitenkin yhdistetty määrittelyvaiheeseen. Tämä johtunee siitä, että esitutkimuksen sisältö on usein sellaista, joka kehittyy vielä eteenpäin määrittelyvaiheessa. (Haikala & Märijärvi 2004, 37.)

Esitutkimuksen tarkoituksena on lähinnä kartoittaa asiakkaan yleiset perustason vaatimukset järjestelmälle asiakkaan yritystoiminnan näkökulmasta niin, että pyritään selvittämään mitä järjestelmällä on tarkoitus tehdä. Esimerkkinä voitaisiin käyttää kuvitteellista verkkokaupantuotehallinnan kehittämistä niin, että asiakas löytää verkkokaupasta etsimänsä, kun vanhassa versiossa tuotteiden etsiminen on todettu hankalaksi tai jopa mahdottomaksi. Tämänkaltaisessa tilanteessa järjestelmän kehittämisen vaatimukseksi nousisi verkkokaupanhakujen tehostaminen, jonka pohjalta järjestelmää alettaisiin kehittää eteenpäin. Esitutkimus pyrkii siis ensisijaisesti vastaamaan kysymykseen miksi järjestelmä tulisi tehdä tai jättää tekemättä. (Haikala & Märijärvi 2004, 37). Tämän vaiheen suurin vaikeus on asiakkaan todellisten tarpeiden ja vaatimusten esille saaminen, joka tuntuu olevan monien ohjelmistoprojektien suurin ongelma.

Esitutkimusvaiheesta siirrytään varsinaiseen määrittelyvaiheeseen. Määrittelyvaiheessa esitutkimuksessa selvitettyjen tarpeiden pohjalta määritellään järjestelmän toiminnallisia vaatimuksia, jotka kirjataan vaatimusmäärittelyksi tai toiminnalliseksi määrittelyksi kutsuttuun dokumenttiin. Tähän dokumenttiin kuvataan järjestelmän rajoitukset sekä ei-toiminnalliset ja toiminnalliset vaatimukset mahdollisimman tarkasti. Rajoituksia ovat mm. ohjelmointikieli, alusta mille järjestelmä tehdään sekä sen asettamat tekniset rajoitukset esim. muisti ja suorittimenteho. Ei-toiminnallisia vaatimuksia ovat mm. käytettävyys, vaste-aika ja suoritusteho. Toiminnalliset vaatimukset ovat taas asioita, mitä järjestelmä tekee, tai mitä käyttäjä voi järjestelmässä tehdä. Esimerkkinä käyttötapaus haku verkkokaupasta: ”Käyttäjä etsii hakusanoilla tuotteen verkkokaupasta hakutoiminnolla, järjestelmä suorittaa haun ja palauttaa tuloksen x”. Määrittelyn tarkoituksena on siis muuntaa alkuperäiset asiakasvaatimukset täsmällisiksi ohjelmistovaatimuksiksi, jotka kuvataan mahdollisimman tarkasti dokumentaatioon (Haikala & Märijärvi 2004, 39).

3.4.2 Suunnitteluvaihe

Suunnitteluvaiheessa siirrytään määrittelyvaiheen mitä järjestelmä tai sen käyttäjä tekee -kuvauksista siihen, että pyritään vastaamaan kysymykseen miten järjestelmä tulee toteuttamaan nämä määrittelyn asiakasvaatimukset (Haikala & Märijärvi 2004, 40).

Suunnitteluvaiheessa pyritään mm. selvittämään ja suunnittelemaan, miten järjestelmän käyttöliittymässä tullaan navigoimaan ja miten eri työnkulut etenevät sekä millainen arkkitehtuuri järjestelmällä tulee olemaan. Viimeistään suunnitteluvaiheessa pyritään lyömään lukkoon, miten järjestelmän tekniset ratkaisut toteutetaan. (Effective prototyping 2007, 22.) Mikä-

li myöhemmin havaitaan jotkut tekniset ratkaisut huonoiksi, on evolutiivisessa mallissa mahdollista muuttaa näitä ratkaisuja myöhemmin. Ongelmana ratkaisuiden muuttamisessa on se, että projektin hinta saattaa nousta.

Monesti suunnitteluvaihe jaetaan vähintään kahteen tai kolmeen vaiheeseen. Yleensä ensin suunnitellaan, miten järjestelmän tekninen toteutus eli arkkitehtuuri toteutetaan. Esimerkiksi päätetään, tuleeko järjestelmästä moduuleihin jaettu vai ei. Mikäli järjestelmästä tulee moduuleihin jaettu, järjestelmä pilkotaan mahdollisimman pieniin osasiin eli moduuleihin, jotka voidaan itsenäisesti toteuttaa toisista osista riippumatta. Tämä mahdollistaa myöhemmin näiden moduulien uudelleen käytön toisissa ohjelmointiprojekteissa. (Haikala & Märijärvi 2004, 40, 313; Sommerville 2001, 310–311.) Tämänkaltaista itsenäistä ja hyvin suunniteltua moduulia voikin ihannetapauksessa käyttää uudelleen ilman minkäänlaista päivitystä. Esimerkiksi voidaan ottaa vaikka kuvitteellisen järjestelmän sisäänkirjautuminen. Tästä voidaan luoda itsenäinen moduuli, jota voidaan sitten uudelleen käyttää sellaisenaan johonkin samankaltaiseen projektiin. Moduulien vahvuudeksi voidaan myös lukea, että niiden päivittäminen on myös usein yksinkertaisempaa, kuin laajojen ja monimutkaisten kokonaisuuksien. Moduuleihin jakaminen on järkevin tapa toteuttaa tietojärjestelmiä.

Arkkitehtuurin määrittelemisen jälkeen järjestelmän toteutuksen suunnitelma jaetaan itsenäisesti toteutettaviksi moduuleiksi. Jakamisen jälkeen moduulien sisäinen rakenne ja toteutus suunnitellaan. (Haikala & Märijärvi 2004, 40.)

3.4.3 Ohjelmointivaihe

Ohjelmointivaiheessa suunnitteluvaiheessa määritellyt asiat toteutetaan ja testataan moduuli kerrallaan. Mikäli suunnitelmissa havaitaan puutteita, tehdään järjestelmään tarvittavat muutokset. Kun moduulit on saatu valmiiksi, ne yhdistetään järjestelmäksi asiakkaan ensimmäistä testikierrosta varten. (Haikala & Märijärvi 2004, 40.)

3.4.4 Evolutiivisen prototyypin testausvaihe ja palautteen keräys

Evolutiivisessa prototyypitysprosessissa asiakas testaa ensimmäisellä kierroksella todennäköisesti hyvin alkeellista versiota järjestelmästä. Mikäli järjestelmästä löytyy virheitä, asiakas kirjaa havaitut kehittämiskohteet ja virheet mahdollisimman tarkasti ylös. Kuvaruutukaappaukset ja tarkka kuvaus virheitä edeltäneistä toimenpiteistä auttavat asiakasta toistamaan virheet katselmoinneissa. Tämä testausvaihe toistuu järjestelmän kehittyessä eteenpäin, kunnes asiakas on tyytyväinen valmiiseen järjestelmään.

Testausvaiheessa löydetyt virheet ja parannusehdotukset käsitellään esimerkiksi läpikäyntitilaisuuksissa (ks. luku 3.4.8). Ohjelmoijien tehtävä näissä tilaisuuksissa on yleensä tehdä havainnoista tarkentavia kysymyksiä ja muistiinpanoja.

Tämän jälkeen evolutiivisessa prosessissa palataan suunnitteluvaiheeseen, missä suunnitellaan esiin tulleiden virheiden korjaus. Tämän jälkeen siirrytään uudelleen ohjelmointivaiheeseen, missä esiin tulleet virheet ja puutteet korjataan, jonka jälkeen alkaa taas uusi testauskierros. Tätä prosessia toistetaan niin monessa syklissä, että kehitettävää tai korjattavaa ei enää löydy.

3.4.5 Käyttöönotto

Kun asiakas on hyväksynyt järjestelmän, suoritetaan sen käyttöönotto. Käyttöönottovaiheessa järjestelmä rakennetaan tai integroidaan olemassa oleviin järjestelmiin. Tämän jälkeen suoritetaan hyväksymistestaus, missä varmistetaan vielä järjestelmän toiminta. Käyttöönottoon kuuluu usein myös henkilökunnan kouluttaminen uuden järjestelmän käyttämiseen ja ohjeiden luominen.

3.4.6 Ylläpito

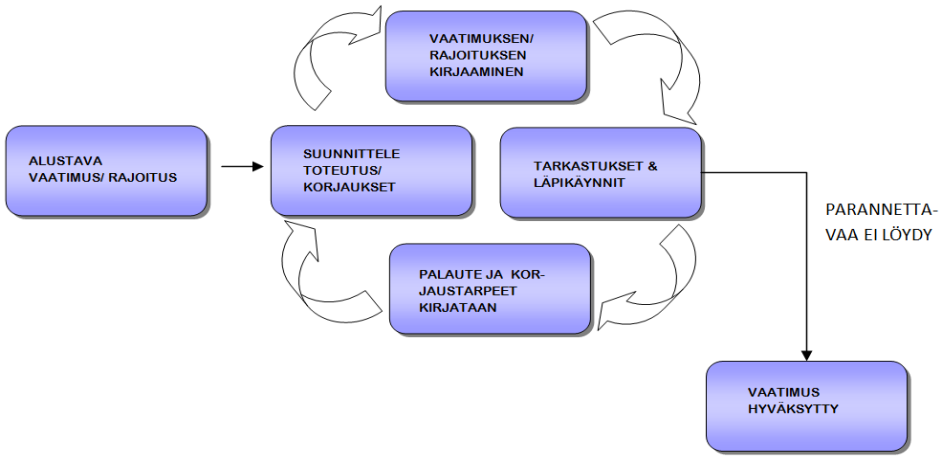
Ylläpitovaiheessa järjestelmää joudutaan yleensä vielä päivittämään, korjaamaan tai muuttamaan. Testausvaiheessa eivät kaikki järjestelmän viat ole vielä välttämättä tulleet esiin ja näiden korjaaminen tapahtuu yleensä ylläpitovaiheessa. Asiakkaan yrityksen tai ympäristön muutokset saattavat myös aiheuttaa järjestelmälle muutospaineen, jonka seurauksena järjestelmä päivitetään uudistunutta tilannetta vastaavaksi (Haikala & Märijärvi 2004, 41). Yrityksen graafinen ilmekin saattaa kokea muutoksen ja se pitää myös päivittää yrityksen tietojärjestelmiin.

3.4.7 Prototyypitysprosessin laadunvalvonta

Laadunvalvonta on oleellinen osa ketterien menetelmien järjestelmätestausta ja yksi prototyypitysprosessin tärkeimmistä asioista. Prototyypitysprosessin laadunvalvontaan on syytä kiinnittää erityistä huomiota jo projektin alusta alkaen ja aina sen loppuun asti. Hyvin suunniteltu ja valvottu prototyypitysprosessi on tie vähintään kohtalaiseen lopputulokseen. Mikäli laadunvalvontaa laiminlyödään projektin aikana, on varmaa että se tietää jatkossa vaikeuksia ja lisä kustannuksia. Laadunvalvontaa voidaan suorittaa esimerkiksi testaamalla ja tarkistuttamalla jokainen vaatimusmäärittelyn rajoitus, vaatimus sekä moduulin suunnitelma ja toteutus jollakin hyväksi havaitulla tavalla ennen niiden lopullista hyväksymistä.

Testaamisen ja tarkastamisen tarve muodostuu yksinkertaisesti siitä, että se on taloudellisesti kannattavaa ja hyvin perusteltua. Esimerkiksi Jonesin (1986b) ja Boehmin (1987a) tutkimusten mukaan 40–50 % ohjelmistoprojektin kuluista koostuu vaatimusmäärittelyn, suunnittelun ja ohjelmoinnin uusimisesta ja erilaisten virheiden korjaamisesta. Gilbin (1988) mukaan yli 60 % ohjelmistoihin päätyneistä vioista on ollut olemassa jo suunnitteluvaiheessa. Laadunvalvonnan tarkoituksena on löytää virheet jo ennen kuin ne edes pääsevät järjestelmään. Jonesin (1994) mukaan jokainen tunti, mitä erilaisiin laadunvalvontatoimenpiteisiin käytetään, vähentää virheiden korjaamiseen käytettyä aikaa 3–10 tuntia. (McConnell 2002, 71–73.) Kaikki virheet, jotka korjataan jo ennen kuin ne ovat järjestelmässä, ovat säästöjä työajassa.

Laadunvalvontaan voidaan käyttää monenlaisia testaamisen työvälineitä. Esittelen esimerkkinä kaksi katselmustapaa, joita voi molempia hyödyntää aivan prosessin alusta alkaen: läpikäyntipalaverit ja tarkastukset (kuvio 5).



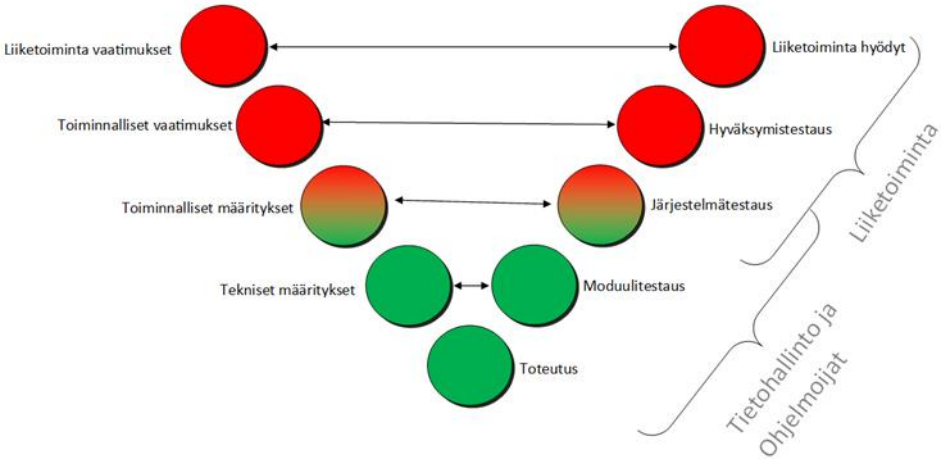
Kuvio 5. Läpikäyntien ja tarkastusten prosessi.

Läpikäyntipalavereilla tarkoitetaan, mitä tahansa tapaamista, missä keskustellaan kahden tai useamman kehittäjän voimin toteutuksen sisällöstä. Keskustelussa esiin tulleet virheet kirjataan ja korjataan myöhemmin. Asiakkaan osallistuminen läpikäynteihin on erittäin suotavaa, koska se edesauttaa virheiden löytämistä. Asiakas saa samalla tilaisuuden haastaa oman ajattelunsa ja se edes auttaa häntä havaitsemaan omia virheitään ja puutteitaan vaatimuksissa. Läpikäynneissä voi löytyä Meyersin (1979), Boehmin (1987b) sekä Yourdonin (1989b) tutkimusten mukaan jopa 30–70 % ohjelman virheistä (McConnell 2002, 73).

Tarkastukset ovat kulultaan ennalta määriteltyjä katselmustilanteita, joiden kulku on hyvin muodollinen. Niiden on havaittu muodollisuudestaan huolimatta olevan erittäin tehokas keino vikojen havaitsemisessa. IBM:llä työskennelleen Faganin (1976) kehittämät tarkastukset ovat säilyttäneet vuodesta toiseen asemansa parhaana tunnettuna laadunvarmistuskeinona. (Haikala & Märijärvi 2004, 269.) Tarkastuksiin annetaan erityistä koulutusta ja osallistujille jaetaan tarkastuksia varten erityiset roolit. Tarkastaja jakaa tarkastettavan materiaalin ennen tarkastustilaisuutta katsastajille. Katsastajat tutkivat jaetun materiaalin ennen tilaisuutta. Materiaalin tekijä

kuvailee tekeleen omin sanoin, josta katsastajat antavat palautetta. Kirjuri kirjaa esille nousseet ongelmat, joista tarkastaja laatii raportin, jonka perusteella vikoja korjataan. Tarkastusten vaikutukset ohjelmointiprojekteihin ovat olleet hyvin myönteisiä. Aikataulut lyhenevät 10–30 % ja tarkastusten avulla ohjelman vioista 60−90 % tulee esille, mikä on huomattavasti paremmin kuin läpikäyntipalavereilla. (McConnell 2002, 74.)

Testaaminen on hyvä aloittaa jo projektin alussa ja sen olisi hyvä jatkua koko projektin loppuun saakka. Testausta tehdään yleensä monella tasolla v-mallin (kuvio 6) mukaisesti, missä testausta tehdään aina sitä vastaavalla tasolla. Esimerkiksi esimäärittelyvaiheen liiketoimintavaatimukset testataan liiketoiminnan hyötyjen näkökulmasta. Testaamisen tehokkuus riippuu myös paljolti, siitä kuka testaamisen suorittaa. Esimerkiksi valmiin järjestelmän testaamiseen suorittamiseen on hyvä käyttää henkilöitä, jotka eivät ole olleet mukana järjestelmän kehittämisessä millään tavalla. (Haikala & Märijärvi 2004, 40, 290.) Tämä johtunee siitä, että kehittämisessä mukana olleet ovat usein sokeutuneet järjestelmän ilmeisille virheille, joita he ovat virheellisesti oppineet sietämään järjestelmän ominaisuuksina.



Kuvio 6. V-mallin mukainen testauskaavio (ICT Standard Forum 2010).

Testauksen ohjenuorana voidaan pitää sitä, ettei ohjelmoija voi tehdä liiketoimintatestausta ilman sen täysin tuntevaa asiakasta ja asiakas ei voi taas tehdä koodin testaamista ilman ohjelmoijaa. Usein virheellisesti ajatellaan, ettei asiakkaan tarvitse olla mukana koodikatselmuksissa tai ohjelmoijaa ei tarvita liiketoiminnan asettamien vaatimusten testauksessa. Kuitenkin on havaittu, että molempien osapuolien edustajan olisi hyvä olla paikalla molemmissa testaustilanteissa. Näin toimimalla molempien ymmärrys projektista lisääntyy. (McConnell 2002, 236–237.)

3.5 Prototyyppi työvälineenä

Juhani Ruohotien (haastattelu 31.3.2014) mukaan prototyyppi on työkalu, jonka avulla saadaan muutostarpeet tai ominaisuudet konkreettisesti näkyviin ja testattavaksi järkevällä työmäärällä. Prototyypitystä tulee kuten muidenkin työkalujen käyttöä harjoitella, ennen kuin päästään hyviin tuloksiin. On hyvin epätodennäköistä, että ensimmäinen prototyypitysprojekti sujuu ongelmitta.

Prototyyppejä ohjelmistotuotannossa hyödyntäville prototyypityksen katsotaan olevan olennaisen tärkeä työväline, joka oikein toteutettuna varmistaa sovelluskehityksen ja käyttäjäkokemuksien onnistumisen. Prototyypitys johtaa hyvin toteutettuna useimmiten onnistuneeseen lopputulokseen. Prototyypittäminen ei kuitenkaan ole aivan riskitöntä. On todettu, että yhtä helposti kuin hyvin toteutettu prototyypitys auttaa ohjelmointiprojektissa, huolimattomasti ja huonosti toteutettu aiheuttaa vain lisää ongelmia niiden ratkaisemisen sijaan. (Effective prototyping 2007, 3–4.)

3.5.1 Prototyyppi vuorovaikutuksen välineenä

Prototyyppiä voidaan hyödyntää tehokkaasti vuorovaikutusvälineenä ohjelmistokehittäjien ja loppukäyttäjien välillä. Juhani Ruohotien (haastattelu 31.3.2014) mukaan vuorovaikutuksen ongelma syntyy siitä, että ohjelmistokehittäjällä on jo koulutuksensa kautta erilainen tapa ajatella ongelmia ja kokonaisuuksia kuin asiakkaalla. Ohjelmistokehittäjä pyrkii ratkaisemaan ongelman tämän usein pinttyneen oman ajattelutapansa mukaisesti ja ei osaa hahmottaa ongelmaa asiakkaan näkökulmasta. Tämä sama pätee tietenkin myös toisinpäin.

Ongelmia kommunikoinnissa Ruohotien mukaan aiheuttaa erityisesti se, kun aletaan puhua alakohtaisista erikoisuuksista. Ei voida olettaa, että ohjelmoija tuntee matkatoimiston matkavarauksen tarpeet tai ydinvoimalaitoksen toimintaperiaatteet. Asiakas/tilaaja on kuitenkin oman alansa ammattilainen ja omaa paljon tietoa, jota ohjelmiston toteuttajalla ei ole. Ohjelmoija ja asiakas eivät siis useinkaan ymmärrä toisiaan riittävän hyvin. Ongelmia aiheuttaa muun muassa se, että ohjelmiston tarpeita ja sen tulevaa käyttötarkoitusta on usein vaikea ohjelmoijana ymmärtää asiakkaan verbaalisesta selityksestä. (Ruohotie, haastattelu 31.3.2014.)

Asiakkaat eivät useinkaan ymmärrä ohjelmiston toteuttamisesta juuri mitään tai eivät suunnittelupalavereissa muista jotain hyvin oleellista toimintoa, ja näin ohjelmoija ei saa selkeää kuvaa siitä mitä ohjelman pitäisi oikeasti tehdä. Tämän vuoksi usein onkin käynyt niin, ettei tiukkaan esimääritelty lopputuote vastaa ollenkaan asiakkaan todellisia tarpeita. Voidaan siis tiivistäen sanoa, etteivät asiakas ja ohjelmoija aina puhu samaa kieltä, ja tämä aiheuttaa ongelmia ohjelmistoa kehitettäessä. (Ruohotie, haastattelu 31.3.2014.)

Ihminen tarvitsee monta kertaa visuaalista palautetta ymmärtääkseen paremmin sen, mistä on oikeasti kysymys (Effective prototyping 2007, 3). Tätä asiaa voi avata hieman yksinkertaisella ajatusleikillä. Kuvittele mielessäsi miten joutuisit selittämään sanallisesti jollekin henkilölle, joka ei ole koskaan nähnyt Eiffel-tornia edes valokuvissa, miltä se näyttää tai minkä kokoinen se on, niin että hän todella ymmärtää, millainen se on. Asian selittäminen tyhjentävästi vaatisi todella paljon aikaa ja vaivaa. Entäpä jos käytettävissäsi olisi kynä ja paperi? Mahdottomasta tehtävästä tulikin mahdollinen.

Prototyypitys toimii hyvin vuorovaikutuksen välineenä, kun se mahdollistaa selkeän ja ymmärrettävän käyttötapausten kuvaamisen. Prototyyppiä voidaan käyttää visualisoimaan vaatimusmäärittelyn kirjalliset käyttötapaukset, jotka ovat monesti kirjallisessa muodossa hyvin vaikeaselkoisia. Prototyypityksen avulla voidaan siis vaatimusmäärittely ja sen sisältö testata, ja näin varmistaa että vaatimukset on ymmärretty oikein. Usein testauksen aikana huomataan, että vaikka alkuperäiset vaatimukset olisikin ymmärretty oikein, ne eivät kuitenkaan vastaa asiakkaan todellisia tarpeita. Hyvin toteutetun prototyypityksen avulla löydetäänkin usein optimaalisia ratkaisuja asiakkaalle pelkkien käyttötapausten toteuttamisen sijaan. (Effective prototyping 2007, 3; Sommerville 2001, 172.)

3.5.2 Prototyyppi ja innovointi

Prototyyppi on hyödyllinen työkalu myös innovoinnissa. Prototyypin avulla voidaan epämääräisenä mielissä ja papereilla pyörivä konsepti saada konkretisoitumaan visiosta todelliseksi tuotteeksi (FixUi 2013).

Juhani Ruohotien (haastattelu 31.3.2014) mukaan ohjelmistotuotannon asiakkaalla ei useinkaan ole selkeää käsitystä siitä, mitä hän kehitettävältä ohjelmistolta todellisuudessa odottaa. Prototyypityksen avulla näitä epäselviä asiakasvaatimuksia voidaan tehokkaasti paikallistaa. Prototyyppien katsotaan soveltuvan erityisen hyvin uusien ratkaisuiden etsimiseen, kokeilemiseen ja testaamiseen (Haikala & Märijärvi 2004, 42).

3.6 Ohjelmistokehityksen prototyypit

Ohjelmistotuotannossa prototyyppien tekemiseen on olemassa lukemattomia erilaisia tapoja ja metodeja. Prototyyppejä voidaan yksinkertaisimmillaan tehdä luonnostelemalla järjestelmä paperin ja kynän tai tietokoneen piirto-ohjelman avulla.

Prototyyppien tekemiseen tietokoneella on myös varta vasten tehty erilaisia ohjelmistoja ja ohjelmistotyökaluja, näitä on olemassa kymmeniä ja niitä tulee markkinoille jatkuvasti lisää. Yleisimmin tunnettuina esimerkkeinä näistä voidaan mainita Microsoftin Visio ja Adoben Fireworks.

Prototyyppejä tehdään myös ohjelmoimalla ja näin toteutetut prototyypit keskittyvät yleensä toiminnallisuuden, tehokkuuden ja erilaisten alustaratkaisuiden rajoitusten testaamiseen. Graafista ilmettä näissä ohjelmoidussa

prototyypeissä pyritään mallintamaan harvoin, koska siihen voidaan keskittyä myöhemmin ja tämä veisi huomiota pois teknisistä ominaisuuksista.

Ohjelmistoprototyypeiksi luetaan myös erilaiset käyttöliittymäprototyypit, joiden pääpaino on yleensä graafisessa ulkoasussa ja käytettävyydessä. Graafisten käyttöliittymien yleistymisen myötä niiden prototyypityksestä on tullut hyvin merkittävä osa käyttöliittymien suunnittelua. Tämä johtunee siitä, että loppukäyttäjän käyttökokemuksen tutkiminen on ainoa järkevä tapa selvittää graafisen käyttöliittymän käytettävyyttä (Sommerville 2001, 188).

3.6.1 Ohjelmointikehityksen prototyyppien ryhmittelystä

Ohjelmistokehityksessä prototyyppejä ryhmitellään monilla eri tavoilla. Prototyypit jaetaan usein kahteen pääryhmään, koodattuihin ja koodittomiin. Nämä taas jakautuvat edelleen pienempiin ryhmiin. Esimerkiksi ne voidaan jakaa edelleen kahteen ryhmään: evolutiivisiin ja poisheitettäviin prototyyppeihin.

Prototyyppejä ryhmiteltäessä usein puhutaan myös niiden tarkkuudesta. Englanninkielisessä tietoteknisessä kirjallisuudessa törmää usein ryhmittelyyn mikä perustuu prototyypin tarkkuuteen: high fidelity prototypes ja low fidelity prototypes. Suomenkielessä tämä high fidelity eli korkean tarkkuuden prototyyppi tarkoittaa sitä, että prototyyppi voi olla lähes valmis sovellus runsaalla vuorovaikutuksella. Matalan tarkkuuden eli low fidelity -prototyypillä tarkoitetaan esimerkiksi yksinkertaista paperille piirrettyä luonnosta järjestelmästä. (Effective prototyping 2007, 10, 22.)

Tässä opinnäytetyössä esitellyt prototyypit jaotellaan kahteen pääryhmään: toiminnallisiin ja ei-toiminnallisiin. Toiminnallisille prototyypeille on ominaista, että käyttäjä voi jollakin tavalla testata järjestelmää itse. Ei-toiminnallista prototyyppiä kuvaa parhaiten sen esityksenomaisuus ja staattisessa piirroksessa piilevää toiminnallisuutta tarvitseekin jonkun olla esittelemässä.

3.6.2 Prototyypitystavan valinta

Prototyypitys on tapauskohtaista ja prototyypin tarkkuus tulee määritellä tarpeen mukaan. Aluksi suoritetaan arviointi siitä, millainen prototyyppi kannattaa tehdä ja kuinka se rajataan. Ohjenuorana voitaneen pitää sitä, että aina kannattaa pyrkiä keskittymään vain olennaisimpiin asioihin. Esimerkiksi kysymys siitä, tarvitaanko käyttäjien palautetta järjestelmän käyttämisestä, ratkaisee sen, tehdäänkö prototyypistä toiminnallinen vai esityksen omainen. Optimaalisen tuloksen saavuttaminen riippuu usein siitä, miten hyvin prototyypitysvälineen valinta ja tarkkuuden määritteleminen on onnistunut. (Effective prototyping 2007, 22–23.)

Prototyypityksen tehottomuuden katsotaan usein johtuvan siitä, että prototyypityksen tarkkuuden määritteleminen on epäonnistunut kohdeyleisöön nähden. Tästä johtuen prototyypistä ei välttämättä saada halutunkaltaista palautetta. Liian korkealla tarkkuudella toteutettu prototyyppi vie testaajien ja yleisön huomion oleellisista asioista liiaksi epäoleellisiin. Esimerkiksi graafisiin yksityiskohtiin, kun tarkoitus olisi ollut selvittää teknisen toteutuksen asianmukaisuutta. Liian matalalla tarkkuudella toteutettu taas ei anna riittävän selkeää kuvaa yleisölle tulevasta järjestelmästä. (Effective prototyping 2007, 23–24.) Prototyypitystavan valinta olisi hyvä mieltää prosessina, missä alustavien vaatimusten kautta selvitetään tarkkuus ja välineet, jolla prototyypitys kannattaa tehdä.

3.6.3 Prototyyppi ja kustannukset

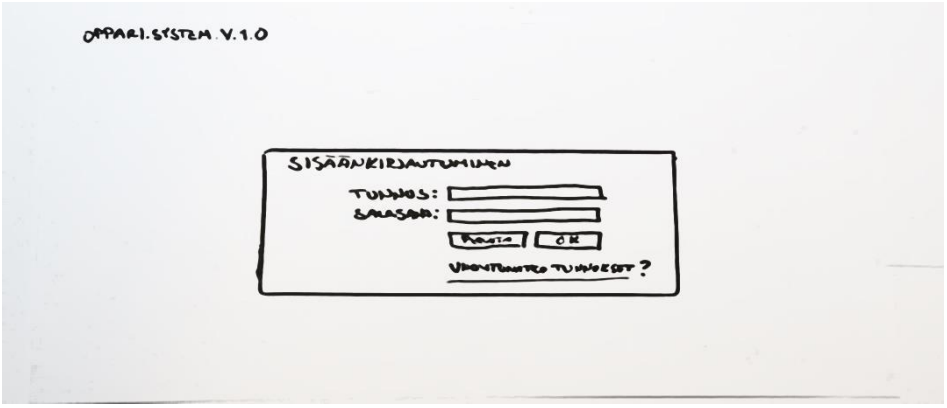
Gordonin ja Biemanin (1995) tutkimuksessa, joka käsitti 39 erilaisia prototyyppejä hyödyntänyttä projektia, saatiin selville, ettei prototyypitysprosessin käyttäminen lisääkään kustannuksia ohjelmointiprojekteissa niin kuin oli yleisesti oletettu. Useimmiten Prototyypitysprosessi kyllä lisää kustannuksia projektien alkuvaiheessa, mutta myöhemmässä vaiheessa prototyypitys aiheuttaa vastaavasti säästöjä. Tutkimuksessa todettiin mm. prototyyppien hyödyntämisen lisäävän järjestelmien käytettävyyttä, ylläpidettävyyttä, parantavan suunnittelun laatua, vähentävän ohjelmistokehityksen määrää ja lopputuotteen pääsevän lähemmäksi asiakkaan toivomuksia. Merkittäväksi ongelmaksi prototyyppien hyödyntämisessä havaittiin joissakin tapauksissa järjestelmien suorituskyvyn aleneminen, mikäli prototyypin esimerkinomaisia alkeellisia koodausratkaisuita uusiokäytettiin uudelleen oikeaa järjestelmää tehtäessä. (Sommerville 2001, 173.)

3.6.4 Paperiprototyypit

Paperiprototyyppejä on tehty jo vuosikymmenien ajan. Tämä on kaikkein käytetyin tapa toteuttaa prototyyppejä ohjelmistoista ja käyttöliittymistä. Erään kyselytutkimuksen mukaan paperiprototyyppien osuus kaikista prototyypeistä on yli 70 prosenttia. Tämä johtunee siitä, että paperisen prototyypin tekemiseen ei välttämättä tarvita kuin hyviä ideoita, aikaa, kynä ja paperia. (Warfel 2009, 110–111; A List Apart 2007.)

Paperiprototyypitys on yksi harvoista metodeista, jotka soveltuvat yhtä hyvin ohjelmistojen kuin fyysisten tuotteiden suunnitteluun. Tämän metodin vahvuuksia on sen nopeus, edullisuus, helppous ja muokattavuus sekä se, että sitä voidaan tehdä missä ja milloin vain. (Warfel 2009, 111–113.)

Käytännössä paperiprototyypin tekeminen tapahtuu niin, että piirretään kaikki järjestelmään suunnitellut elementit paperille. Elementit pyritään asettelemaan paperille ”käyttäjien” eteen niin kuin ne näkyisivät oikeallakin tietokoneen ruudulla, tosin niiden ei tarvitse välttämättä olla visuaalisesti lähelläkään lopputulosta (kuva 1).



Kuva 1. Sisäänkirjautuminen esitettynä paperiprototyypissä.

Toiminnallisuutta paperiprototyypeissä esitellään usein esimerkiksi siten, että ensin piirretään kaikki ohjelmassa aukeavat ikkunat, ilmoitukset ja tapahtuvat toiminnot erikseen samassa mittakaavassa erillisille papereille. Tämän jälkeen toiminnallisuutta paperiprototyypeissä testataan ja esitellään usein vain napsauttamalla nappulaa kuvitteellisesti sormella, jonka jälkeen seuraavaa toimintoa esittävä kuva laitetaan leikekirjamaisesti edellisen kuvan päälle (kuva 2). (speckyboy 2010.) Tässä tapauksessa salasanaa eikä tunnusta ole annettu, joten kirjautuminen epäonnistuisi.

Kuva 2. kirjautuminen epäonnistui tunnusten puuttuessa.

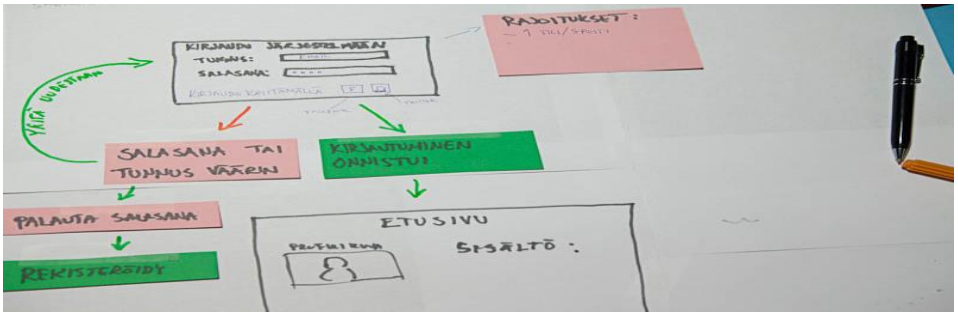
Suuremmissa ohjelmisto projekteissa ongelmaksi tällaisessa esittelyssä voi kuitenkin muodostua eri toimintojen ja niihin liittyvien paperileikkeiden lisääntyminen. Pienissä sovelluksissa tästä tuskin koituu suurempia ongelmia, mutta ongelma esiintyy sitä todennäköisemmin, mitä laajemmasta sovelluksesta on kyse. (speckyboy 2010.)

Tämän paperileikkeiden hallinnoinnin ongelman ratkaisemiseksi voidaan ottaa käyttöön teknisiä apuvälineitä, esimerkiksi joko kamera tai videokamera. Ongelma pyritään näiden avulla ratkaisemaan siten, että kuvataan jokaiseen toimintoon liittyvät yksittäiset askeleet askel kerrallaan. Yksi toiminto voi olla vaikka jo aiemmin esimerkkinä ollut sisäänkirjautumisen onnistuminen tai epäonnistuminen. Näistä toiminnoista muodostetaan en-

sin yksi toiminto kerrallaan dia- tai videoesityksiä, jotka voidaan katsoa myöhemmin tietokoneelta. (speckyboy 2010.)

Toiminnot kuvataan mielellään yksi kerrallaan sen tähden, että ne olisivat paremmin hallinnassa ja niitä voitaisiin mahdollisesti hyötykäyttää myös osana projektinhallintaa. Nämä toimintojen kuvauksiin liittyvät yksittäiset esitykset voidaan sitten liittää projektinhallintaohjelmiston ”backlogiin” tai tehtävälistoihin tehtäväkohtaisesti. Koko ohjelmiston esittelyä varten näistä osiosta voidaan kuitenkin koostaa melko vaivattomasti koko ohjelman kattava esitys.

Tämän paperileikkeiden ongelman voi ratkaista myös käyttämällä Adaptive Pathin kehittämää Sketchboarding-menetelmää (kuva 3), missä järjestelmän määrittäminen aloitetaan noin ilmoitustaulun kokoisesta paperista. Tähän paperille aletaan leikekirjamaisesti järjestellä alustavia ideoita, rajoituksia, vaatimuksia ja erilaisia ongelmakohtia. Tätä leikekirjamaista paperiprototyyppiä laajennetaan kunnes siihen on kirjattuna koko järjestelmä ja sen työnkulut. Sketchboardin valmistuttua se voidaan laittaa näkyvälle paikalle toimiston seinälle. Sketchboarding-menetelmän ehdoton vahvuus on siinä, että järjestelmä on kokonaisuudessaan kuvattuna yhdessä paikassa, missä sitä on helppo tutkia. Tämä menetelmä sisältää jo itsessään kevyen vaatimusmäärittelyn, jonka perusteella järjestelmä voidaan toteuttaa. Määrittelyn ollessa näkyvillä projektin jäsenet voivat löytää siitä jatkuvasti parannus-, korjaus- ja kehityskohteita. (Adaptive Path 2007.)



Kuva 3. Alkeellinen esimerkki Sketchboarding-menetelmästä.

Näistä paperisista prototyypeistä muodostuu todella arvokas apuväline ohjelmoijille, jotka toteuttavat järjestelmän toiminnallisuudet. Näiden toiminnallisuuksien kuvausten avulla on selkeästi nähtävissä myös visuaalisesti se miten ohjelman tulisi toimia. Ohjelmoijien työksi jää lähinnä vain toiminnallisuuksien tekninen toteuttaminen.

Paperiprototyyppien ehdottomasti paras puoli on siinä, että niitä voi rajattomasti luonnostella uudestaan ilman minkäänlaista sitoutumista koodiriveihin tai teknisiin ratkaisuihin. Tämänkaltaisen prototyypin tekeminen on tehokasta ja edullista. Hyvin toteutettuna paperiprototyypit auttavat kaikkia ohjelmointiprojektiin osallistuvia heidän omassa työssään. (Warfel 2009, 111–113.)

Konsultti Carolyn Snyderilta on kysytty usein paperiprototyyppien tarkkuudesta ja luotettavuudesta. Hän kirjoittaa nettisivuillaan, että hän tuntee vain yhden luotettavan tutkimuksen aiheesta. Tämän tutkimuksen mukaan yksinkertaisten paperiprototyyppien avulla löytyi suurin osa samoista ongelmista kuin parempien oikeasti toimivien sovellusprototyyppien avulla. Sovellusprototyyppien avulla erilaisia ongelmia löytyi kyllä enemmän, mutta näiden monimutkaisempien prototyyppien tekemiseen menee myös huomattavasti enemmän aikaa. Carolyn Snyder toteaa, että hänen mielestään paperiset prototyypit ovat huomattavasti kannattavampia suhteessa niihin hyötyihin mitä nämä paremmat prototyypit tarjoavat suhteessa kuluneeseen aikaan. (Snyder Consulting 2004.)

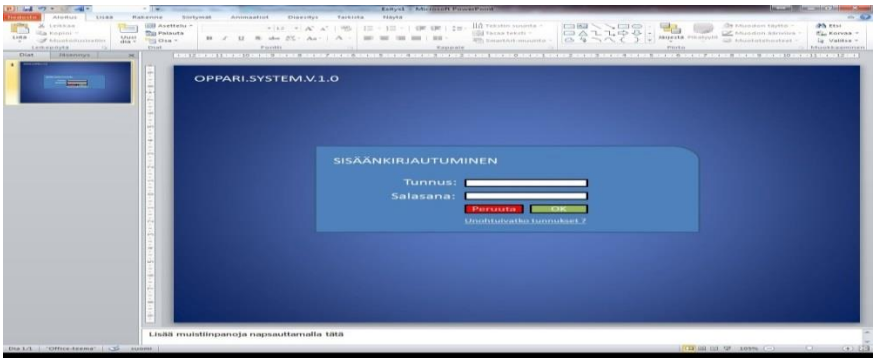
Paperiset prototyypit ovat parhaimmillaan silloin, kun ollaan tekemässä jotain aivan uutta. Paperiset prototyypit on helppo hyllyttää, jos tuotteesta ei tule mielekästä tai sitä ei kannata tehdä, ja näin voidaan siirtyä nopeasti seuraavaan projektiin. (Snyder Consulting 2004.)

Ihan kaikkeen paperiset prototyypit eivät kuitenkaan taivu. Paperisten prototyyppien avulla ei voida esimerkiksi testata teknisten ominaisuuksien toiminnallisuutta, kuten sivujen vierittämistä, tietokantahakujen nopeutta tai sivujen latautumisaikaa. Värejä ja fontteja on myös hyvin vaikea mallintaa tarkasti paperilla. (Snyder Consulting 2004.)

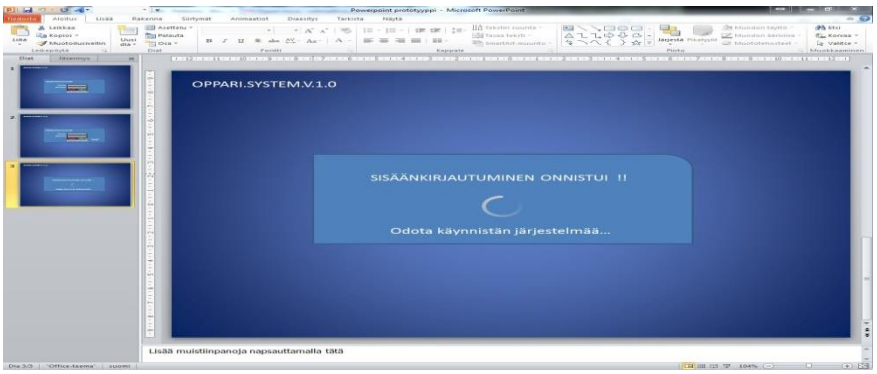
3.6.5 Esitysgrafiikkaprototyypit

Esitysgrafiikkaohjelmistot taipuvat luovasti käytettynä loisteliaiden diaesitysten luomisen lisäksi myös prototyyppien tekemiseen. Esimerkiksi Microsoftin PowerPoint ja Applen Keynote ovat tähän hyvin soveltuvia ohjelmia. Esitysgrafiikkaohjelmien erityiseksi vahvuudeksi voidaan katsoa se, että lähes jokaisella tietokoneella maailmassa on joku esitysgrafiikkaa tukeva ohjelmisto. Tämä mahdollistaa esitysgrafiikkaprototyyppien testaamisen missä ja milloin vain, kunhan käytössä on tietokone. (Warfel 2009, 140.)

Esitysgrafiikkaprototyyppejä tehdään yleensä toiminnallisina tai luentomaisina diaesityksinä. Näiden kahden merkittävin ero lienee siinä, että toiminnallista esitysgrafiikkaprototyyppiä voi kuka tahansa itsenäisesti kokeilla, kun taas esitykselliseen prototyyppiin tarvitaan yleensä järjestelmän toiminnan tuntevaa esittelijää. Tämä sen tähden, että ilman esittelijää diaesityksen muodossa olevasta prototyypistä voi olla vaikea ymmärtää, miten sovellus toimii tai kuinka sen olisi tarkoitus toimia. Toiminnallisessa prototyypissä sovelluksen toiminnasta voi saada ihan hyvän käsityksen itse kokeilemalla. (Warfel 2009, 140.) Esitykseen perustuvassa prototyypissä on esittelijän selostettava jokaisen ikkunan merkitys projektiin osallistuville tahoille (kuva 4 ja 5).



Kuva 4. Esitysgrafiikkaprototyypin sisäänkirjautumisikkuna.



Kuva 5. Esitysgrafiikkaprototyypin sisäänkirjautumisilmoitus.

Esitysgrafiikkaprototyypit muistuttavat monimuotoisuudessaan paljolti paperiprototyyppejä, mutta niiden tekemiseen menee usein hieman enemmän aikaa. Varsinkin silloin, jos elementit piirretään huolellisesti ja pyritään pääsemään graafisesti mahdollisimman lähelle toivottua lopputulosta.

Esitysgrafiikkaohjelman avulla voidaan myös paperiprototyyppi muuntaa alkeellisesti toimivaksi toiminnalliseksi prototyypiksi. Tämä tapahtuu seuraavasti: Skannataan tai valokuvataan paperiprototyypin kaikki osaset ja tallennetaan ne tietokoneelle kansioon. Näistä kuvista sitten muodostetaan esitysgrafiikkaohjelmistossa prototyyppi, jonka nappeja painamalla niihin linkitetyt toiminnallisuudet tapahtuvat. Tämä toteutetaan nappuloihin linkitetyillä uusien diojen avaamisilla tai kuvien ilmestymisillä diaan. Esitysgrafiikalla prototyyppi toteutetaan samoilla periaatteilla, mutta käyttämällä esitysgrafiikkaohjelmiston omia piirtotyökaluja. Nämä eivät kuitenkaan ole kovin edistyneitä ja huomattavasti laadukkaampaan lopputulokseen päästään, jos käytetään apuna jotain piirtämiseen tarkoitettua ohjelmaa kuten Adoben PhotoShop. (Warfel 2009, 142–143, 148–157.)

Esitysgrafiikka taipuu myös hyvin nopeaan prototyypittämiseen, mikäli apuna käytetään valmiiksi luotuja prototyyppikirjastoja. Tällaisen kirjaston voi luoda itse tai sellaisen voi ladata internetistä. Erilaisia prototyyppikirjastoja esitysgrafiikkaohjelmille löytyy internetistä satoja. Yksi hienoimmista oli Windows 8 -käyttöjärjestelmän visuaalinen pohja, jonka avulla pystyy luomaan omien sovelluksien prototyyppejä suhteellisen kivuttomasti kyseiselle alustalle. Valitettavasti kyseisen pohjan lataaminen olisi ollut maksullista, joten tätä pohjaa ei käytännössä lähdetty testaa-

maan. Tällaisen kirjaston voi myös osaava PhotoShop ohjelman käyttäjä luoda itselleen muutamassa päivässä. (Warfel 2009, 141, 147-149.)

3.6.6 Linkitetty PDF-prototyyppi

Linkitetty PDF-prototyyppi ei eroa esitysgrafiikkaprototyypeistä juuri millään tavalla, koska ne ovat toteuttavissa täysin samankaltaisesti kuin esitysgrafiikkaprototyypit. Toiminnallisen esitysgrafiikkaprototyypin voi tallentaa PDF-muodossa, jolloin tämä toimii linkitettynä PDF-prototyyppinä. Näiden kahden välillä on toki olemassa joitakin pieniä eroavaisuuksia. Esimerkiksi linkitettyyn PDF-prototyyppiin voidaan Adoben Acrobat-ohjelmistossa luoda lomakkeita, joihin voi kirjoittaa tekstiä prototyyppiä testatessa. (Boxes and Arrows 2007.)

Linkitettyjä PDF-prototyyppejä voi tehdä monilla eri ohjelmistoilla, mikäli ne vain tukevat PDF-muotoon tallentamista. PDF-prototyypin etuna on sen pieni koko, ja suurin hyöty tuleekin siitä, että PDF-prototyyppi samoin kuten esitysgrafiikkaprototyyppi on helposti lähetettävissä edelleen sähköpostilla. Kaikki, joille PDF-prototyyppi lähetetään, voivat tutkia sitten tätä prototyyppiä itsenäisesti, jos heillä vain on asennettuna tietokoneelleen joku PDF-tiedostoja avaava ohjelma. (Boxes and Arrows 2007.)

3.6.7 HTML-prototyypit

HTML-prototyyppi, kuten jo nimestä voi arvata, tehdään HTML-merkintäkielellä ja muotoillaan CSS:n avulla. HTML-prototyyppi luodaan lähes poikkeuksetta verkkoselaimessa toimivaksi. HTML-prototyyppeihin tulee lähes aina toiminnallisuutta, tosin on mahdollista tuottaa myös vain visuaalista ilmettä mallintava käyttöliittymäprototyyppi.

HTML-prototyypit voidaan tehdä yksinkertaisimmillaan yksinkertaisten tekstieditorien avulla. HTML-prototyyppejä tehdään usein kuitenkin erilaisilla ohjelmistoilla, kuten Adobe Dreamwiever, Tiggr tai Adobe Fireworks. Näitä ohjelmia käytetään, koska niiden katsotaan helpottavan työskentelyä. Näiden ohjelmistojen lisäksi on olemassa myös lukuisia ilmaisia frameworkejä, joiden avulla prototyyppien tekeminen helpottuu huomattavasti. Esimerkkinä mainittakoon grid960 system, joka perustuu valmiiksi kirjoitettuun CSS-tiedostoon, jonka avulla HTML-prototyypitys helpottuu huomattavasti. (Warfel 2009, 251.) Tämänkaltaisen frameworkin voi myös luoda itse omien tarpeidensa mukaan.

Warwelin mukaan HTML-prototyyppien tekeminen voi olla alkuun hidasta ja aikaa vievää, mutta taitojen kehittyessä tällä tavoin prototyypittämisestä voi tulla jopa nopeampaa kuin muilla tavoilla (Warfel 2009, 253). HTML-prototyyppiä käytetään yleensä selainsovelluksien mallintamiseen, mikä on luonnollista käyttöympäristöstä johtuen. HTML-prototyypin eduiksi voidaan katsoa se, että tämä prototyyppi on aina jatkokehitettävissä valmiiksi sovellukseksi.

HTML:n käyttäminen on joissakin piireissä saanut aikaa vievän ja hankalan toteuttamistavan maineen, vaikka todellisuudessa se voi olla nopea ja palkitseva kokemus loistavilla eduilla, toteaa Dreamvieweriä käyttävä Julie Stanford artikkelissaan. Hän mainitsee myös että suurin syy siihen ettei HTML-prototyyppejä tehdä, johtunee siitä, että tekijät ovat tottuneet käyttämään muita tapoja tehdessään prototyyppejä. (Boxes and Arrows 2003.)

4 HAASTATTELUTUTKIMUKSEN TULOKSET

Haastatteluun osallistuneet ammattilaiset työskentelivät haastattelua tehtäessä erilaisissa tietojenkäsittelyalan tehtävissä. Yksi vastanneista toimi ohjelmisto-alan yksityisyrittäjänä, toinen toimi ohjelmistokehittäjänä Majoituspalvelu Forenom Oy:n sisäisessä ohjelmistokehitystiimissä ja kolmas työskenteli W3 Group Finland Oy:ssä ohjelmistokehittäjänä. Haastateltavien koulutustaso oli korkea: ohjelmisto- ja diplomi-insinööri, ohjelmistotekniikan insinööri ja tietojenkäsittelyn tradenomi.

4.1 Haastattelun terminologian tunteminen

Teemahaastattelun aluksi käytiin lyhyesti läpi haastattelun keskeisimmät termit ja niiden tunteminen. Tämä oli välttämätöntä, jotta saatiin selville, mitä ajatuksia termit haastateltavissa herättivät ja päästiin yhteisymmärrykseen siitä, mistä ollaan keskustelemassa. Näitä termejä olivat vaatimusmäärittely, prototyyppi ja ohjelmistoprototyyppi

Vaatimusmäärittely oli terminä tuttu jokaiselle haastatteluun osallistuneelle. Vaatimusmäärittelyn merkitys ymmärrettiin kuitenkin kahdella eri tavalla. Yksi haastatelluista mielsi vaatimusmäärittelyn ainoastaan paperiseksi vaatimusdokumentiksi, kun toiset taas näkivät sen huomattavasti laajempana kokonaisuutena, mihin dokumentoinnin lisäksi kuului koko vaatimusmäärittelyn tekeminen ja siihen liittyvät prosessit. Haastateltaville tarkennettiin, että tässä haastattelussa on tarkoitus keskittyä nimenomaan paperiseen vaatimusdokumenttiin ja siihen liittyviin erilaisiin ongelmiin.

Prototyyppi ja ohjelmistoprototyyppi olivat termeinä tuttuja ja ne osattiin erottaa toisistaan. Haastatteluun osallistuneet henkilöt olivat kaikkiaan hyvin perillä haastatteluun liittyneistä termeistä.

4.2 Kokemuksia vaatimusmäärittelystä

Haastateltavista jokaisella oli kokemusta ohjelmointiprojektin tekemisestä pelkän tekstimuotoisen vaatimusmäärittelyn tai käyttötapausten pohjalta. Vaatimusmäärittely koettiin oleelliseksi osaksi ohjelmistokehitystä, mutta kirjallisen oppikirjamaisen vaatimusmäärittelyn katsottiin olevan riittämätön tapa tehdä ohjelmistokehitystä.

Haastateltujen mielestä asiakkaan näkemys eroaa usein toteuttajan vastaavasta, koska kirjoitettua tekstiä voidaan tulkita liian vapaasti. Lähes poikkeuksetta kirjallisten käyttötapausten joukosta löytyy vaatimuksia, jotka on kirjoitettu tavalla, jotka johtavat väärinymmärryksiin. Vaatimukset saattavat myös olla asiakkaan oman erikoisosaamisen alueella niin, ettei toteuttaja voi ymmärtää vaatimusta ilman sen asianmukaista tulkintaa.

Haastateltujen mukaan asiakkaalla on useimmiten selkeä näkemys siitä, mitä hän tarvitsee mutta vääristynyt mielikuva siitä, että hän ymmärtää todelliset tarpeensa. Käytäntö on kuitenkin osoittanut, että asiakas ei ymmärrä tarpeitaan riittävän kattavasti ja vaatimusmäärittelyt eivät vastaa-

kaan asiakkaan todellisia tarpeita. Tätä näkemystä tukee myös Steve McConnellin (2002, 239) toteamus siitä, että asiakkaan todelliset vaatimukset ovat usein ristiriidassa kirjattujen vaatimusten kanssa

Asiakkailla on haastateltujen mukaan myös usein tapana lisätä toiminnallisuuksia ohjelmistoon vain varmuuden vuoksi tai projektista innostumisen seurauksena. Näillä ominaisuuksilla ei kuitenkaan todellisuudessa tehdä mitään ja nämä aiheuttavat asiakkaalle vain turhia kustannuksia.

Tiukka vaatimusmäärittely on haastateltujen mukaan bisnesmaailman mieleen, kun sen avulla voidaan tehdä tarkka kustannusarvio. Käytännössä tämä ei toimi, koska tuotteen kehitysvaiheessa määrittelyä joudutaan usein tarkentamaan tai muuttamaan. Esimerkiksi nopeasti muuttuvat markkinat saattavat aiheuttaa tarpeen muuttaa järjestelmän käyttötarkoitusta kesken järjestelmän kehityskaarta. Usein myös valmiissa tuotteessa havaitaan puutteita ja siltä osin määrittelyä joudutaan laajentamaan. Haastateltujen näkemyksiä tukee myös Haikalan ja Märijärven (2004, 92) havainnot samoista ilmiöstä ja toteamus siitä että ohjelmistoprojekteissa on hyväksyttävä se, että osa vaatimuksista tulee muuttumaan projektin aikana.

Ongelmaksi ohjelmointiprojektissa voi muodostua myös se, että määrittelyn tekee johtoasemassa oleva henkilö tai joku muu tavalla, missä ohjelmiston todelliset käyttäjät eivät saa ääntään kuuluviin. Näin määrittely ei vastaa loppukäyttäjän tarpeita lainkaan ja ohjelmistoa joudutaan korjaamaan tai se tehdään uudestaan.

Vaatimusmäärittelyt kannattaisi erään haastatellun mukaan tehdä tiiviissä yhteistyössä asiakkaan kanssa tavalla, missä asiakas ei kirjaa paperille yhtäkään vaatimusta. Järjestelmän todelliset vaatimukset tulisi pyrkiä saamaan selville haastattelemalla asiakasta. Vaatimukset tulisi testata järjestelmällisesti yhdessä asiakkaan kanssa ja pyrkiä näin varmistamaan niiden oikeellisuus. Tätä näkemystä tukee hyvin vahvasti myös Steve McConnellin viittaus Vosburghin (Voshburgh et. al. 1984) tutkimukseen. Tutkimuksessa oli havaittu, että asiakkaan osallistuminen vaatimusmäärittelyn tekemiseen vaikuttaa myönteisesti tuottavuuteen. Positiivinen vaikutus oli Voshburghin mukaan sitä suurempi, mitä enemmän asiakkaat osallistuivat vaatimusmäärittelyyn. Suurimmillaan päästiin jopa 50 % tuottavuuden kasvuun suhteessa yleiseen keskiarvoon. Samassa tutkimuksessa havaittiin, että asiakkaiden itsensä kirjoittamat vaatimukset vaikuttivatkin negatiivisesti tuottavuuteen ja niistä piti jopa yli puolet kirjoittaa uudestaan. (McConnell 2002, 240–241.)

Vaatimukset ja järjestelmät tulisi samaisen haastatellun mukaan luoda lyhyissä iteraatioissa, milloin määriteltävää ei olisi liikaa. Ohjenuorana voitaisiin tällöin pitää 20/80 -mallia, missä ohjelmisto pilkotaan osiin ja ensimmäiseen versioon tehdään vain tärkeimmät ja oleellisimmat toiminnot, jotka käsittävät noin 20 % kaikista toiminnoista. Tämän toimintatavan hyödyksi haastateltu katsoi, että tuotetta päästään testaamaan nopeasti ja testauksen aikana havaitaan useimmiten suuren osan lopuista toiminnoista olevan turhia ja ettei niitä tarvitse lainkaan toteuttaa.

4.3 Käytännön esimerkkejä vaatimusmäärittelyn epäonnistumisista

Haastatelluilta tiedusteltiin myös konkreettisia kokemuksia pieleen menneistä ohjelmistomäärittelyistä ja niihin johtaneista syistä. Eräs haastatelluista kertoi olleensa mukana toteuttajan roolissa eräässä mittavassa projektissa, jonka määrittely oli kaatunut väärinymmärrysten seurauksena.

Asiakas oli käyttänyt 50 henkilön ryhmällä yli neljä kuukautta aikaa määrittelyn tekemiseen. Määrittelyprojektia vetämään oli palkattu kolme ulkopuolista konsulttia tunnetusta konsulttitalosta. Määrittelyn lopputuloksena oli yli 800 sivua käsittävä sekavin ja vajavaisin määrittely, mitä hän oli koko uransa aikana nähnyt. Määrittelyssä oli ollut valtava määrä ristiriitoja ja erilaisia ongelmia. Lisäksi haastatellulle toimitettiin asiakkaan toimesta suuren mainostoimiston tekemät käyttöliittymäsuunnitelmat, jotka esittivät täysin eri sovellusta kuin mitä määrittelyssä oli kuvattu. Käyttöliittymästä puuttui suurin osa määrittelyn toiminnoista ja siellä oli toimintoja joita määrittelystä ei löytynyt lainkaan. Osapuolet syyttivät ongelmista väärinymmärryksiä.

Haastateltu oli todennut hyvin nopeasti, ettei kyseistä määrittelyä voi lähteä toteuttamaan lainkaan. Hän sai asiakkaan taivuteltua tilaamaan sovelluksen toteuttajavetoisesti ketterillä menetelmillä ja kevyillä määrittelyillä. Sovellus valmistui määrittelyineen kolmessa kuukaudessa neljän hengen voimin. Sovellus tuli lopulta mittavasta määrittelystä johtuen asiakkaalle melko kalliiksi.

Haastatellulta tiedusteltiin, mitkä asiat hänen mielestään olivat johtaneet kyseiseen tilanteeseen. Haastatellun mielestä määrittelyä tekemään oli valittu vääriä henkilöitä ja heitä oli liikaa. Mainostoimistolla oli käyttäjälähtöinen näkökulma, kun taas asiakas mietti sovellusta vain omien prosessiensa kautta. Toteuttajaa ei haluttu kuunnella määrittelyvaiheessa lainkaan. Tätä tapausta voidaan tarkastella teoreettisesti Vosburghin (Voshburgh et. al. 1984) tutkimuksen valossa, johon jo aikaisemmin viitattiin. Tutkimuksen tulosten perusteella 50 % tuosta asiakkaan tekemästä vaatimusmäärittelystä olisi todennäköisesti pitänyt kirjoittaa uudestaan.

Toinen haastatelluista nosti esiin tapauksen, missä osapuolilla ei ollut selkeää kuvaa siitä, mitä oltiin oltu tekemässä. Kehittäjät olettivat ymmärtävänsä, mitä tilaaja halusi alkukeskustelujen perusteella. Kehittäjät toteuttivat sovelluksen tämän alustavan oletuksiin perustuvan vaatimusmäärittelyn perusteella ja noudattivat tätä tarkasti. Tästä seurasi se, että järjestelmän toteutus lähti jo projektin alusta alkaen peruuttamattomasti väärään suuntaan. Kehittäjät olivat muiden väärien olettamustensa lisäksi valinneet sovellukselle väärän alustan. Projekti epäonnistui, koska toteutettu järjestelmä oli täynnä virheitä ja sitä ei voitu edes muuttaa oikealle alustalle.

Haastatellun mukaan väärinymmärrykset ja olettamukset aiheuttivat projektin epäonnistumisen. Ongelmat olisi vältetty haastatellun mukaan esimerkiksi tarkastuksilla ja prototyypittämisellä. Nämä esimerkkitapaukset tukevat tässä tutkimuksessa esiin tulleita havaintoja kommunikaation, testaamisen ja tarkistusten erityisestä merkityksestä vaatimusmäärittelyprosessissa.

4.4 Kokemuksia ilman vaatimusmäärittelyä ohjelmoinnista

Haastatelluilta tiedusteltiin, olivatko he työssään joutuneet työskentelemään ilman vaatimusmäärittelyä. Tarkoituksena oli saada jonkinlainen käsitys siitä, ovatko vaatimusmäärittelyt aina välttämättömiä ja mitä siitä seuraa, jos määrittelyä ei ole.

Kaikki haastatellut olivat työskennelleet ilman ohjelmiston määrittelyä, suullisen tiedon ja muistilappujen varassa. Haastatellut olivat yhtä mieltä siitä, että tämä voi toimia ainoastaan pienissä ohjelmointiprojekteissa.

Haastateltujen mukaan vaatimusmäärittelyttä ohjelmointi toimii hyvin silloin, kun toteuttajana on osaava henkilö, joka tuntee tilaajan toimintamallit ja tarpeet entuudestaan. Mikäli käytetyt menetelmät ja prosessit eivät ole hallussa, voi muutosten aiheuttama kuorma kasvaa nopeasti liian suureksi. Sovitut asiat myös unohtuvat helposti ilman dokumentointia ja tiedon jakaminen voi olla ongelmallista jos kyseessä on tiimi yksilön sijaan.

Tämänkaltainen projekti vaatii useimmiten onnistuakseen hyvin tiivistä kommunikointia asiakkaan kanssa. Mikäli asiakas ei ole aina saatavilla, on riskinä töiden seisahtuminen. Seisahtuminen aiheuttaa sen, että koodaajat joutuvat tekemään montaa projektia yhtä aikaa. Erään haastatellun mukaan projektien välillä surffaaminen lisää stressiä ja tekee itse työn tekemisestä hankalaa. Projektista toiseen hyppiminen aiheuttaa projektiin liittyvien ajatusten katkeamisen ja niistä voi olla vaikea saada uudestaan kiinni.

Haastatellut näkivät vaatimusmäärittelyttä työskentelyssä lukuisia ongelmia. Esimerkiksi kaikki tieto projektista on usein yhden henkilön varassa. Tämän sairastuessa voi työn haltuun ottaminen olla muille todella haasteellista. Eräs haastatelluista kertoi joutuneensa jatkuvasti tämänkaltaisiin tilanteisiin, mitkä hän koki erittäin stressaaviksi. Tästä seurasi jatkuvaa ylitöiden tekemistä, tiukkojen aikataulujen ja myöhästymisestä aiheutuvien sakkojen pelossa.

4.5 Kokemuksia prototyypeistä sovelluskehityksessä

Haastatelluista jokainen oli hyödyntänyt prototyypitystä jollakin tavalla työurallaan. Selkeästi haastateltujen vastauksissa oli kuitenkin havaittavissa se, että prototyypitystä käytetään harvemmin.
Haastatelluista vain yksi ilmaisi käyttävänsä prototyypittämistä jatkuvasti omassa työssään. Haastateltujen mielipiteissä suhteessa prototyypitykseen oli havaittavissa pieniä eroavaisuuksia, sen mukaan miten paljon prototyypittämisestä oli kokemusta.

Haastatellut suhtautuivat matalan tarkkuuden prototyypittämiseen erittäin myönteisesti. Varsinkin paperiprototyypittäminen ja käyttöliittymien yksinkertaiset prototyypit koettiin erittäin käytännöllisiksi. Haastateltujen mukaan asiakkaalle prototyypitys näyttäytyy helposti vain lisäkuluna, vaikka todellisuudessa se voi tuoda säästöjä.

Prototyypityksen avulla on mahdollista saada asiakaskin helpommin ja paremmin sitoutettua projektiin, kun hän näkee visuaalisesti, mitä ollaan tekemässä. Näin on mahdollista saada asiakaskin innostumaan projektista. Asiakkaan innostumisella on myös riskinsä: helpossa prototyypittämisessä asiakas saattaa innostua ja lisäillä toimintoja mitä hän ei koskaan tule tarvitsemaan. Eräs haastatelluista suosittelikin prototyypitykseen inkrementaalista mallia, missä ominaisuuksia lisätään ainoastaan tarpeen mukaan. Jokaiselle lisätylle toiminnolle tulee asiakkaan antaa perustelu, jotta toteuttaja todella ymmärtäisi miksi toimintoa tarvitaan. Samalla asiakas joutuu hyväksyttämään ajatuksen itsellään. Toteuttajalle tarjoutuu samalla mahdollisuus haastaa asiakkaan vaatimus ja tarvittaessa pyytää siihen täsmennystä.

Haastatelluilta tiedusteltiin, onko heidän mielestään olemassa riski, että asiakas luulee prototyypin olevan joissakin tapauksissa lähes valmis sovellus. Haastatelluista ne kaksi, jotka olivat vähemmän olleet tekemisissä prototyypittämisen kanssa, olivat yhtä mieltä siitä, että asiakas saattaa kuvitella prototyyppiä lähes valmiiksi sovellukseksi, vaikka käytännössä suurin osa töistä olisi vielä tekemättä. Tätä näkemystä tukee myös Haikalan ja Märijärven ohjeistus prototyypityksestä, jonka mukaan liian viimeistellyn näköistä prototyyppiä kannattaa välttää. Tämä sen tähden, että asiakaskin ymmärtää suurimman osan työstä olevan vielä tekemättä. (Haikala & Märijärvi 2004, 43.) Enemmän prototyypitystä tehnyt henkilö oli asiasta kuitenkin osittain eri mieltä. Hänen kokemuksensa mukaan asiakas on tyytyväinen nähdessään asioiden etenevän ja saadessaan jotain konkreettista hypisteltävää, jonka avulla hän voi haastaa oman ajattelunsa tulevasta sovelluksesta. Riski prototyypin luulemisesta valmiiksi sovellukseksi voidaan välttää riittävällä ja oikeanlaisella läpinäkyvyydellä.

Korkean tarkkuuden prototyypittämistä, kuten esimerkiksi pois heitettävää prototyyppiä yksi haastatelluista piti täysin käsittämättömänä työskentelytapana. Hän ei voinut käsittää, mitä hyötyä on työn tekemisessä kahteen kertaan. Muiden haastateltujen näkemykset asiasta olivat hyvin samansuuntaisia. Haastatelluista jokainen näki korkean tarkkuuden prototyypittämisen olevan järkevää ainoastaan silloin, kun on kyse evolutiivisesta tai inkrementaalisesta mallista, minkä lopputuloksena on tarkoitus syntyä valmis sovellus.

Haastatellulta kysyttiin lopuksi, olisiko heidän mielestään hyödyllistä aloittaa jokainen ohjelmointiprojekti vähintään paperisella prototyypillä. Kaikki olivat yhtä mieltä siitä, että se olisi hyödyllistä. Yksi haastatelluista kuitenkin tarkensi, että eihän se ole aina järkevää. Varsinkaan silloin kun kyseessä on pieni projekti tai projekti mihin kyseinen toimintamalli ei sovi.

5 POHDINTA

Helsingin yliopiston tietojenkäsittelytieteen laitoksen luentomateriaalissa sanotaan ohjelmistojen olevan monimutkaisimpia ihmisen aikaansaannoksista. Luentomateriaalissa todetaan kuitenkin myös ohjelmistotekniikan olleen kriisissä jo 1960-luvulta alkaen. Kriisillä kuitenkin tarkoitetaan lyhytaikaista vakavaa tilaa, ja siksi ohjelmistotekniikan kohdalla ei voida puhua oikeastaan edes kriisistä, vaan pikemminkin on kyse pitkäaikaista hoitoa vaativasta kroonisesta sairaudesta. Ohjelmistotuotannon katsotaan olevan sairaassa tilassa, jossa kehityksen nopeasta tahdista johtuen vastaan tulee jatkuvasti uusia hoitoa vaativia oireita. Tästä syystä lääkitystä ei ehditä koskaan saamaan ajan tasalle. (Lokki 2008.)

Monesti näitä oireita aiheuttavat taudinkuvat pysyvät samoina ohjelmistotuotannon projektista toiseen, vaikka niihin olisi olemassa lääkityskin, jolla taudinkuvaa voitaisiin helpottaa. Oireet pysyvät samoina, kun olemassa olevaa lääkitystä ei jostain syystä oteta käyttöön. Näitä yleisimmin ilmeneviä tauteja ovat mm. aikataulupaineet, yletön innovointi, vaatimusten luisuminen, tavoitteiden muuttuminen, olennaisten asioiden unohtaminen, tieteellisten menetelmien puute, epäeettinen käyttäytyminen, teknisten määritysten sekä projektisuunnitelman kirjallisen dokumentoinnin laiminlyönti. Yleisimmäksi ja vakavimmaksi vaivaksi ohjelmistotuotannossa nostetaan kuitenkin epärealistisen kiireiset aikataulut, joista johtuu suurin osa ohjelmistoprojektien epäonnistumisista. (Lokki 2008.) Ohjelmistotuotannon kroonistuneeseen tilaan tuskin koskaan löydetään pysyvää ratkaisumallia, joka parantaisi kaikki siihen liittyvät sairaudet.

Paras mahdollinen lääkitys ohjelmistotuotannon ongelmiin syntyy siitä, että ohjelmistotuottaja esittää itselleen kysymyksen, miten pääsen projekteissani parhaaseen lopputulokseen. Hänen tulee etsiä ratkaisua ongelmaan kokeillen erilaisia olemassa olevia ohjelmistotuotannon käytäntöjä, kunnes sopivien ja toimivien käytäntöjen paletti on valmis. Paletin sisältöä joudutaan varmasti muuttamaan ajan kuluessa ja erilaisten asiakkaiden tarpeiden mukaan. Ohjelmistotuottajan tulee siis tulla tietoiseksi omista vahvuuksistaan toimijana etsien ja opiskellen itselleen parhaiten sopivat käytännöt. Tämän kaltainen oppiminen ei tapahdu ilman virheiden tekemistä. Virheistä oppiminen on arvokasta ja niiden hallintaan kannattaa ohjelmistotuottajan paneutua hyvin.

Parhaiden käytäntöjen etsimiseen voi ohjelmistotuottaja soveltaa esimerkiksi Steve McConnellin kirjaa Ohjelmistotuotannonhallinta. Siinä pureudutaan kattavasti ohjelmistotuotannon parhaisiin käytäntöihin ja esitellään niiden hyötyjä ja haittoja. Käytäntöjen esittelyllä McConnell pyrkii vastaamaan ohjelmistotuotannon erilaisiin haasteisiin parhaalla mahdollisella tavalla ja mainitseekin kirjansa keskeisen tavoitteen olevan alhaisen tuottavuuden kanssa painivien ohjelmointiyritysten tuottavuuden parantamisen (McConnell 2002, xi).

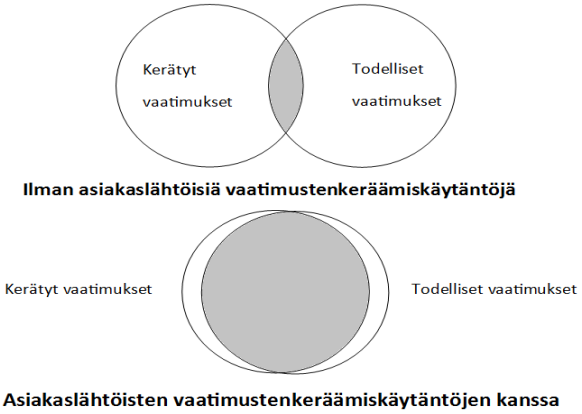
Opinnäytetyön teoriaosassa paneudutaan kattavasti siihen, kuinka prototyypitystä tehdään käytännössä. Opinnäytetyöprosessin aikana prototyypityksen havaittiin olevan hyödyllinen työväline sovelluskehityksessä, jonka avulla voidaan ratkoa myös vaatimusmäärittelyihin liittyviä ongelmia. Prototyypityksen avulla on mahdollista saavuttaa tarkempi ja asiakaslähtöisempi vaatimusmäärittely kuin perinteisillä keinoilla. Prototyyppiä voidaan hyödyntää myös osana vaatimusmäärittelyn dokumentaatiota ja tämä auttaa ohjelmoijaa koko projektin ajan.

Matalan tarkkuuden prototyypityksen havaittiin olevan erinomainen työväline kommunikoinnissa sekä uusien ratkaisuiden innovoinnissa. Prototyypin tarjoaman visuaalisen palautteen tutkiminen yhdessä asiakkaan ja ohjelmoijan välillä on erittäin hyvä tapa varmistaa, että asiakas on tullut ymmärretyksi ja toisinpäin. Prototyyppi auttaakin ohjelmoijaa pääsemään lähemmäs asiakkaan todellisia tarpeita sovelluksen kehittämisessä.

Haastattelututkimuksessa havaittiin vanhanaikaisten vesiputousmallin mukaisten vaatimusmäärittelykäytäntöjen olevan vallitsevaan tilanteeseen nähden liian hitaita ja jäykkiä. Tämänkaltaisten vaatimusmäärittelydokumenttien ymmärtäminen ja käsittely voi olla myös asiakkaalle liian haasteellista ilman koulutusta. Monet asiakkaat kokevatkin tällaiset vaatimusmäärittelyt raskaiksi. Tämä tekee niistä sellaisenaan lähes käyttökelvottomia. Toki joistakin järjestelmän ominaisuuksista voi olla tarpeen tehdä tarkka kirjallinen määrittely, mutta kokonaisen järjestelmän määritteleminen kirjallisesti on useimmiten rajallisten voimavarojen haaskaamista. Vaatimusmäärittelyn ylläpitämisestäkin voi laajoissa projekteissa vaatimusten muuttuessa tulla mahdotonta. Vaatimusmäärittelyyn onkin nykyisin syytä käyttää erilaisia nopeasti opittavia ketteriä menetelmiä ja kirjata vaatimukset tarkasti vain sitä välttämättä vaativien toimintojen osalta. Esimerkkinä hyvästä ja asiakasystävällisestä määrittelyyn käytetystä ketterästä menetelmästä voidaan mainita Sketchboarding -menetelmä (ks. luku 3.6.4).

Asiakkaan osallistumisella vaatimuksien määrittelyyn havaittiin olevan todella suuri merkitys vaatimusmäärittelyjen onnistumisen kannalta. Tärkeää on myös se, ettei asiakas kirjaa vaatimuksia itse, vaan ohjelmistotuottaja kirjaa asiakkaan kanssa vuorovaikutuksessa löydetyt vaatimukset. Asiakas ei useinkaan osaa kirjata vaatimuksia ohjelmistotuottajalle ymmärrettävään muotoon ilman koulutusta. Vaatimuksien etsintä tulee siis suorittaa erilaisilla asiakaslähtöisillä vaatimusten keräyskäytännöillä, joihin myös prototyypitys kuuluu.

Asiakaslähtöiset vaatimusmäärittelykäytännöt nousevat tämän opinnäytetyön keskeisimmäksi havainnoksi. Näillä keräysmenetelmillä on merkittävä rooli vaatimusmäärittelyprosessissa (kuvio 7). Oikeat henkilövalinnat sekä eri osapuolien aktiivisuus vaatimusmäärittelyprosessissa saattavat myös osaltaan ratkaista sen, onnistuuko projekti vai ei. Asianmukaiset tarkistukset, testaaminen ja vaatimusten kyseenalaistaminen edesauttavat laadukkaan määrittelyn syntymistä.



Kuvio 7. Asiakaslähtöisyyden merkitys vaatimusmäärittelyssä (McConnell 2002, 240).

6 YHTEENVETO

Opinnäytetyö lähti liikkeelle olettamuksesta, että prototyypityksestä olisi hyötyä sovelluskehityksessä sekä vaatimustenmäärittelyssä. Työn edetessä tämä olettamus osoittautui oikeaksi. Opinnäytetyön alkuperäiset tutkimuskysymykset tarkentuivat työn edetessä vastaamaan paremmin oleellisimmiksi katsottuja havaintoja. Lisäksi havaittiin, ettei prototyypitys yksinään riitä ratkaisemaan vaatimusmäärittelyn ja ohjelmistotuotantoon liittyviä ongelmia. Ratkaisu ongelmiin syntyy pikemminkin ohjelmistotuottajien itsensä vastuusta kehittää omia toimintatapojaan vastaamaan omia ja asiakkaidensa tarpeita paremmin. Erityisen hankalaksi tämän tekee se, ettei aikaa ja motivaatiota toimintatapojen kehittämiseen ole yleensä riittävästi. Ohjelmistotuottajan itsensä kehittäminen muodostuu osin parhaiden käytäntöjen etsimisestä ja niiden hyödyntämisestä omassa työssä.

Tutkimusmenetelmänä käytetty tapaustutkimus ja sen tueksi tehty teemahaastattelu osoittautui erinomaiseksi tavaksi kerätä aineistoa opinnäytetyötä varten. Teemahaastattelun avulla saatiin selvitettyä erittäin hyvin alalla työskentelevien ammattilaisten näkemyksiä suhteessa tutkimuskysymyksiin. Valittujen metodien avulla saatiin kerättyä aineisto, jonka avulla tutkimuskysymyksiin voitiin vastata kattavasti.

Kirjoittaja sai työtä tehdessään selkeän kuvan vaatimusmäärittelyiden tarpeellisuudesta, ja tutustui itselleen ennalta tuntemattomiin vaatimusmäärittely ja prototyypityskäytäntöihin. Opinnäytetyö toi kirjoittajalle teoreettisen käsityksen prototyypityksestä ja vaatimusmäärittelystä, sekä niiden mahdollisista hyödyistä ja haitoista. Työssä opittuja asioita hän tulee hyödyntämään työskennellessään omassa ammatissaan tulevaisuudessa.

Kirjoittajalle opinnäytetyö osoittautui todella haastavaksi projektiksi, jonka lopputuloksena kirjoittajan oma suhtautuminen elämään ja opiskeluun koki huomattavan muutoksen. Opinnäytetyössä käsitellyt aiheet auttoivat kirjoittajaa hahmottamaan uudella tavalla omaan elämänhallintaan ja opiskeluun liittyviä ongelmia. Esimerkkinä voitaisiin mainita työn pilkkomisen merkitys suhteessa edistymiseen. Mikäli laajaa kokonaisuutta ei pilkota osiin, etenemisestä voi tulla vaikeaa työtaakan muodostuessa liian suureksi. Tämän opinnäytetyön aiheuttaman itsetutkiskelun lopputuloksena kirjoittaja koki kasvaneensa ihmisenä, oppijana ja ammattilaisena..

Opinnäytetyössä todettiin, että vaatimusten määrittely koetaan usein hyvin hankalana ja vaativana prosessina. Tämä aiheuttaa vaatimusmäärittelyprosessille jatkuvan muutospaineen, kunnes näihin ongelmiin löydetään optimaalinen ratkaisu. Tästä syystä ohjelmistotuotanto tarvitsee työkaluja, joiden avulla saadaan selvitettyä asiakkaan todelliset vaatimukset tai tehostettua vaatimusmäärittelyprosesseja entisestään. Tässä opinnäytetyössä havaittujen asiakaslähtöisten vaatimustenkeräyskäytäntöjen hyödyntämisestä käytännössä voitaisiin tehdä tulevaisuudessa oma opinnäytetyönsä, jolla osaltaan pyrittäisiin ratkaisemaan olemassa olevia ongelmia.

Monet ohjelmistotuotannon ongelmat on jo ratkaistu, mutta jostain syystä silti ohjelmistoprojektit kärsivät näistä samoista ongelmista projektista toiseen. Ohjelmistotuottajien olisi hyvä herätä kehittämään omaa toimintaansa jollakin tavalla. Ohjelmistotuotannon parhaiden käytäntöjen etsimisprosessi voisi auttaa ohjelmistotuottajaa pääsemään optimaaliseen tuottavuuteen. Parhaiden käytäntöjen etsimisestä saisi myös tulevaisuudessa laajan aiheen opinnäytetyölle.

LÄHTEET

Adaptive Path 2007. Brandon Schauer. Sketchboards: Discover Better & Faster UX Solutions. Viitattu 6.2.2015 http://www.adaptivepath.com/ideas/sketchboards-discover-better-faster-ux-solutions/

A List Apart 2007. Shawn Medero. Paper Prototyping. Viitattu 13.11.2014 http://alistapart.com/article/paperprototyping

Arent Michael, Arnowitz Jonathan & Berger Nevin.2007. Effective prototyping for software makers Published by Elsevier, Inc

Boxes and Arrows 2003. Julie Stanford HTML Wireframes and Prototypes: All Gain and No Pain. Viitattu 20.1.2015 http://boxesandarrows.com/html-wireframes-and-prototypes-all-gain-and-no-pain/

Boxes and Arrows 2007. Kyle Soucy PDF Prototypes: Mistakenly Disregarded and Underutilized. Viitattu 20.1.2015 http://boxesandarrows.com/pdf-prototypes-mistakenly-disregarded-and-underutilized/

FixUI. 2013. Prototyypin toteutus. FIXUI OY Viitattu 27.6.2014. http://www.fixui.fi/palvelut/prototyypin_toteutus

ICT Standard Forum. Valmistelu ja business case. 2010 Viitattu 17.2.2015 https://www.tietohallintomalli.fi/bookpage/2010-09-21/valmistelu-ja-business-case

Haikala Ilkka & Märijärvi Jukka. 2004. Ohjelmistotuotanto. Talemtum media oy Helsinki.

Hirsjärvi Sirkka & Hurme Helena . 2011. Tutkimushaastattelu: Teemahaastattelun teoria ja käytäntö. Gauedamus Helsinki University Press. Oy Yliopistokustannus, HYY yhtymä.

Julkisen hallinnon tietohallinnon neuvottelukunta (JUHTA). 2009b. JHS 173 ICT-palvelujen kehittäminen: Vaatimusmäärittely. Viitattu 4.2.2014. http://docs.jhs-suositukset.fi/jhs-suositukset/JHS173/JHS173.pdf

Lokki, Heikki. Johdatus tietojenkäsittelytieteeseen. 2008. Viitattu 10.2.2015 http://www.cs.helsinki.fi/u/lokki/JTKT/sy2008_11_26-12_3_HL.pdf

McConnell Steve, 2002. Ohjelmistotuotannon hallinta, Edita Publishing Oy, IY Press

Puusniekka Anna & Saaranen-Kauppinen Anita. 2006. KvaliMOTV - Menetelmäopetuksen tietovaranto. Tampere: Yhteiskuntatieteellinen tietoarkisto. Viitattu 1.12.2014.
http://www.fsd.uta.fi/menetelmaopetus/kvali/L5_5.html

Sininen Meteoriitti 2011. Tolvanen Perttu & Toivola Sami. Ketteryys haltuun viitattu 29.1.2015
http://www.meteoriitti.com/Artikkelisarjat/Ketteryys-haltuun/

Snyder Consulting 2004. Snyder Carolyn. Paper Prototyping. Viitattu 13.11.2014 http://www.snyderconsulting.net/us-paper.pdf

Sommerville Ian. 2001. Software engineering. Pearson Education Limited.

Speckyboy Design Magazine 2010. Andrew Paul. 10 Effective video examples of paper prototyping. Viitattu 13.11.2014
http://speckyboy.com/2010/06/24/10-effective-video-examples-of-paper-prototyping/

Tech Target. 2005. Rouse Margaret: Prototype.
Viitattu 3.9.2014
http://searchmanufacturingerp.techtarget.com/definition/prototype

Zaki Warfel Todd, 2009. Prototyping. Rosenfeld Media.

HAASTATTELUT

Luuri, Olli 2014. Ohjelmistokehittäjä. Forenom majoituspalvelu. Haastattelu 5.4.2014

Ruohotie, Juhani 2014. Yksityisyrittäjä. Valo Interactive. Haastattelut 31.3.2014 ja 4.2.2015.

Veijonen Petri. Ohjelmistokehittäjä. W3 Group Finland Oy. Haastattelu 5.4.2014.