Joona Marjasalo

KÄYTTÄJÄKESKEINEN JÄRJESTELMÄKEHITYS JA SEN TUOMA HYÖTY OHJELMISTON KÄYTTÖÖNOTTOKUSTANNUKSISSA

JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2012

TIIVISTELMÄ

Marjasalo, Joona
Käyttäjäkeskeinen järjestelmäkehitys ja sen tuoma hyöty ohjelmiston käyttöönottokustannuksissa
Jyväskylä: Jyväskylän yliopisto, 2012, 28 s.
Tietojärjestelmätiede, Kandidaatin tutkielma
Ohjaaja: Makkonen, Pekka

Ketterän kehityksen ohjelmistoprojekti on nykyaikainen ohjelmistotuotannon väline, jolla saavutetaan tehokkaasti järjestelmävaatimukset kattava ohjelmisto. Siitä kuitenkin usein uupuu todellinen käytettävyyssuunnittelu ja järjestelmävaatimuksissakin on vielä nykyään vain maininta, että ohjelmiston tulee olla "käytettävä", mikä ei miltään osin ole riittävä kriteeri. Käytettävyys ei ole yksiselitteinen adjektiivi vaan kuuluu aina tietyn järjestelmän ja tietyn kontekstin piiriin ja siksi ohjelmistotuotteet ovat usein käytettävyydeltään heikkoja, koska käytettävyysasiantuntijat ovat vielä verrattain harvinainen näky ohjelmistoprojekteissa.

Käytettävyysmetodeita ei haluta käyttää siksi, että ne nähdään perinteisesti hitaina tapoina kehittää järjestelmiä ja siksi sopimattomina ketterän kehityksen projekteihin. Lopuksi vielä suhteellisen kallis hinta rajoittaa entisestään yleistä halukkuutta pestata käytettävyysasiantuntijoita ja siksi ohjelmistoprojektit pyrkivät selviämään pelkästään ohjelmistoarkkitehtien ammattitaidon avulla.

Oikeanlaisella lähestymistavalla käyttäjäkeskeinen ohjelmistokehitys on kuitenkin mahdollista sisällyttää ketterän ohjelmistokehityksen malliin ja kalleutensa sijasta se tuo projektiin tehokkuutta ja kustannussäästöjä. Oikeanlaiseksi suunniteltu lopputuote tuottaa asiakkaalle myös arvoa ja vähentää varjokustannuksia, jotka voivat olla huomattavia summia ja käytännössä tulevat ilmi vain asiakkaan tuottojen laskuna.

Avainsanat: Käyttäjäkeskeinen ohjelmistokehitys, käyttöönottokustannus, ylläpitokustannus, ketterä ohjelmistokehitys, käytettävyyssuunnittelu, projektimalli

ABSTRACT

Marjasalo, Joona
User Centered System Design and the profits that it makes
Jyväskylä: University of Jyväskylä, 2012, 28 p.
Information System Science, Bachelor's Thesis
Supervisor(s): Makkonen, Pekka

Agile software development is a modern tool to design and produce a system that meets the requirements that is given to it in the early phase of the software development project. Usually the problem is that the agile software development lacks the real usability design and it is usual that customers write system requirements like the product must be "usable" or "have a good usability", things that does not really mean anything. Usability is not an adjective that can be defined easily; it consists of a number of features that makes a whole system in the particular context. The reason, why developed systems usually lack proper usability is that usability engineers and designers are comparatively rear sight at the system development site.

Software firms do not want to use usability methods in their system design projects because they are traditionally seen quite slow to develop systems and that is why they do not fit in the agile development project. Quite high price is the final reason why usability engineers are not seen in the system development projects and software firms are relying on the knowledge of system architects.

With a right way, it is possible to integrate a user centered system design to an agile development project model and by being expensive; it will bring actually cost benefits. The product that has been designed well to fit the needs of the end users will bring value to the customer and it will lower shadow costs that can only be seen in the reduce of profits of the company. Shadow costs can also be potentially very high but they can still be avoided by proper system design.

Keywords: User centered system design, Cost of implementation, Maintance cost, Agile software development, Usability design, Project design

KUVIOT

SISÄLLYS

1 Johdanto

Todennäköisesti tunnetuimman määritelmän käytettävyydelle antoi Nielsen (1993), käytettävyys on opittavuutta, tehokkuutta, muistettavuutta, erheitä ja mielihyvää. Käytettävyyden tieteellinen tutkimus aloitettiin käytännössä 1980-luvun alussa ja alan kivijalkateoksia syntyi vuosikymmenen puolivälissä. Sen tutkimista ei ole missään vaiheessa lopetettu, mutta nykyaikaisen ketterän ohjelmistotuotannon vaatimat tehokkaat toimintatavat ovat syrjäyttäneet perinteisesti aikaa vievänä ja kalliina pidetyn kattavan käytettävyyssuunnittelun ohjelmistoprojekteista. Lukemalla tämän tutkielman, lukijalle selviää, kuinka on mahdollista löytää tapa, jolla käyttäjäkeskeinen järjestelmäsuunnittelun käytettävyydeltään laadukas lopputuote voidaan tuottaa käyttämällä ketterän kehityksen tehokkaita ohjelmakoodin tuotantometodeita ja löytää arvopohjaiset perustelut, miksi näin tulisi tehdä.

1.1 Tutkimuksen taustat

Käyttäjäystävällisyys on saanut sanontana suurta nimellistä nostetta tämän päivän ketterän kehityksen projekteissa. Vaikka projektimalli olisi minkälainen, sen sanotaan tukevat käyttäjäystävällisyyttä jo pelkästään siksi, että itse malli kuuluu ketterän kehityksen kategoriaan.

Ongelmakohtia kuitenkin löytyy, miksei lopputuote vastaa käytettävyydeltään sitä tasoa, jonka asiakas voisi yksinkertaisesti vaatia. Yksi syy on, että nykyäänkin asialleen omistautunut ja kouluttautunut käytettävyysinsinööri tai -suunnittelija on vielä harvinainen näky itse ohjelmistoprojekteissa. Asia ei tule muuttumaan, enne kuin organisaatioiden johtoportaat ottaa asian sydämelleen ja haluaa tehdä muutoksen. (Hirasawa, Yamada-Kawai & Kasai, 2010). Kuitenkin ohjelmistoprojektin onnistumisen kannalta on kriittistä, että ohjelmistoinsinöörien ja käytettävyyssuunnittelijoiden yhteistyö alkaisi jo projektin alkutaipaleella. Ohjelmiston todellista kokoa on vaikea arvioida pelkästään toiminnallisuuden näkökulmasta, jos itse käyttöliittymää ei pystytä arvioimaan kokonai-

suudessaan. (Nunes, Constantine & Kazman, 2011) Käytännössä aina ohjelmistoprojektin ensimmäinen vaihe tulisi olla oikean käytettävyysmetodin valinta. Kun maalina on tuottaa tuote mahdollisimman nopeasti ja mahdollisimman pienillä resursseilla, (Bevan, 2009) ei käytettävyyden tutkiminen ja korjaaminen jälkikäteen ole enää kustannustehokasta. Muutamat tutkimukset osoittavat myös, että suurin osa ylläpitokustannuksista, eli noin 80 % tulevat järjestelmän käyttöön liittyvistä ongelmista. (Nunes yms., 2011)

Tutkimushypoteesinani onkin, että käytettävyysasiantuntijoiden myöhäinen mukaan tulo ohjelmistoprojektiin tai puuttuminen kokonaan tuottaa asiakkaan kannalta valtavia lisäkustannuksia. Käytän tutkielmassa lopputuotteen käyttöönotossa syntyvistä kustannuksista nimitystä varjokustannukset, joita syntyy asiakkaalle asioista, joita täytyy tehdä lopputuotteen käyttöön saamiseksi. Tutkielman myöhemmässä vaiheessa näihin perehdytään tarkemmin.

1.2 Tutkielman sisältö

Kandidaatin tutkielmani on tyyliltään kirjallisuuskatsaus, jonka tarkoituksen on paneutua käyttäjäystävällisyyden käsitteeseen, miten se vaikuttaa hallintatasolla projektin elinkaareen ja selvittää, miten käyttäjäkeskeisiä järjestelmäkehityksen projekteja on tutkittu ja millaisia tutkimuksien lopputulokset ovat. Luvuissa kaksi ja kolme tutkin ketterää ja käyttäjäkeskeistä järjestelmäkehitystä ja koetan löytää niistä hyviä ja huonoja puolia, mitkä voisivat vaikuttaa lopputuotteen laatuun.

Luvussa neljä perehdyn käyttäjäystävällisien järjestelmäkehityksen kustannuksiin, koska on sanomattakin selvää, että todellisten käytettävyysasiantuntijoiden mukaantulo kasvattaa paperilla projektin kokonaiskustannuksia. Tarkoituksena onkin löytää perusteluja käyttäjäkeskeisen järjestelmäkehityksen projektimalleille ohjelmiston käyttöönotto- ja ylläpitokustannuksista. Potentiaalisesti lopputuotteen käyttäjät tarvitsevat koulutusta ja aikaa samaistua ohjelmiston kanssa ja muutosresistanssi kasvattaa entisestään aikaa, jonka loppukäyttäjät tarvitsevat ohjelmiston käyttöönottoon ja hyväksymiseen. Nämä tuottavat varjokustannuksia ja vievät rahaa ja sen mukana syövät tulosta asiakkaalta.

Käyttäjäkeskeinen järjestelmäkehitys ottaa huomioon loppukäyttäjän vaatimustason ja tuottaa ohjelmistosta hänelle helposti samaistuttavan. Vaikka itse ohjelmistoprojektin kustannukset kasvaisivatkin, käyttäjäkeskeinen järjestelmäkehitys voi mahdollisesti tuottaa merkittäviä säästöjä asiakkaalle ohjelmiston käyttöönottovaiheessa, ja tuoda perustelut tällaisille projektimalleille.

2 Ketterä ja käyttäjäkeskeinen ohjelmistokehitys

Useimmissa ohjelmistoprojekteissa ketterän kehityksen malli keskittyy, kuinka päästään nopeasti järjestelmävaatimuksista lopulliseen ohjelmistoon. Painopiste on kuitenkin järjestelmien toiminnallisissa ja teknisissä vaatimuksissa ja käytettävyyteen liittyvät vaatimukset jäävät näyttelemään vain sivuosaa. Tämä johtunee faktasta, että ketterän kehityksen manifestin julkistivat insinöörit, jotka olivat järjestelmäarkkitehteja ja ryhmään ei kuulunut ainuttakaan käytettävyysasiantuntijaa. Tarkoituksena olikin auttaa järjestelmäsuunnittelijoita tuottamaan asiakkaalle nopeasti tarpeellisia ohjelmistoja. Tärkein käytettävyyden tuottamiseen liittyvä idea on asiakkaan läsnäolo itse ohjelmistoprojektissa, mutta todelliset käyttäjät jäävät kuitenkin suunnitteluprosessista pois. Asiakas on läsnä vain projektinhallinnallisella tasolla. Tämä tarkoittaa, että ketterän kehityksen projekteissa on korkea riski tuottaa ohjelmistoja, jotka eivät ole todellisten loppukäyttäjien mieleen. (Chamberlain, Sharp & Maiden, 2006) Kahden kehittämistavan erot voisi summata yhdellä virkkeellä: ketterät menetelmät ovat hyvin asiakassuuntautuneita, kun taas käyttäjäkeskeiset keskittyvät käyttäjään. (Kelander, 2012)

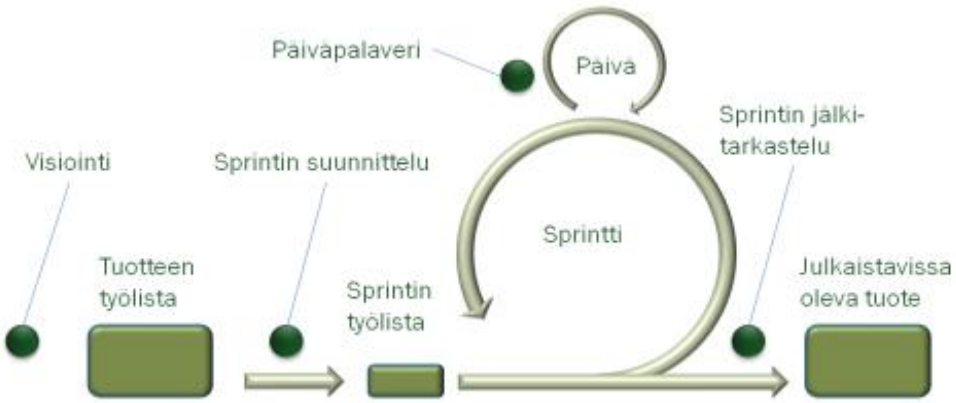
Tässä luvussa keskityn kertomaan ketterästä ja käyttäjäkeskeisestä järjestelmäkehityksestä projektien hallinnallisella tasolla. Lisäksi käyn läpi, kuinka ne yhtenevät ja eroavat tästä näkökulmasta katsottuna toisistaan. Tutkielman laajuuden ja aiheen takia eri järjestelmäkehityksien käyttämiin teknologioihin perehtyminen ei olisi kannattavaa.

2.1 Ketterä kehitys

Ennen ketterää ohjelmistokehitystä ohjelmistot kehitettiin vanhanaikaisilla ja jähmeillä kehitysmenetelmillä, kuten vesiputousmallilla. Se sopii, jos ohjelmiston vaatimukset ovat suhteellisen muuttumattomia, mutta on todettu, että todellisessa maailmassa vaatimukset elävät ja niitä korjataan ohjelmistoprojektin edetessä. Ketterä kehitys on vastaus tällaiseen tilanteeseen. (Nummiaho, 2006)

Ketterän kehityksen projektimallit, kuten esimerkiksi Scrum, alleviivaavat tuotteen omistajan tärkeää roolia ohjelmistoprojektin onnistumisen tai epäonnistumisen kannalta. Avaintekijöinä pidetään omistajan roolia ylläpitää ja priorisoida keräämänsä käyttäjätarinoiden informaatiota ja luoda kriteerit ominaisuuksien hyväksymiselle.

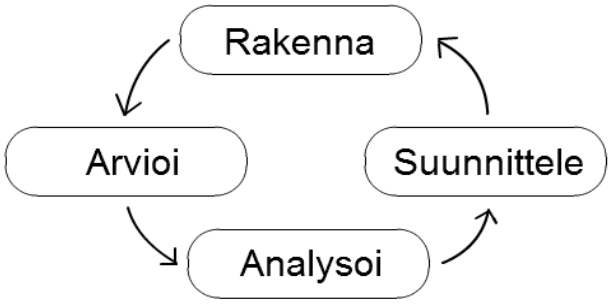
Aluksi ohjelmistotalo tuottaa ehdotuksen tuotteesta asiakkaalle, jolloin asiakkaan vastuulle jää tuottaa yksityiskohtaisempi analyysi tarvittavasta tuotteesta sisältäen tavallisesti tuotekonseptin tarvittavista ominaisuuksista asiakkaan perspektiivistä. Lisäksi käytettävyyden suunnittelu jää melko usein pelkästään ajatustasolle. Ketterän kehityksen pääasiallisena ajatuksena on, ettei käytettävyyden puolesta tehdä minkäänlaista etukäteissuunnittelua ennen ohjelmoinnin aloittamista, eikä se vaadi kehitystiimiä ottamaan huomioon käyttäjien kommentteja ohjelmointipyrähdyksen aikana. Myös prototyyppien teolle ja ohjelmiston vaatimuksien tarkistukselle on tiukan aikataulun takia hyvin vähän mahdollisuuksia ennen kehityksen aloittamista. (Singh, 2008)



Kuvio 1 Scrumin projektimalli (meteori.com)

2.2 Käyttäjäkeskeinen järjestelmäkehitys

Käytettävyydestä on tullut tekijä, joka vastaa tuotteen kaupallisesta menestyksestä. Käytettävyyden on todettu kasvattavan tuottavuutta, yrityksen moraalia ja vähentävän yleiskustannuksia. Käyttäjäkeskeinen järjestelmäkehitys nähdään perustuvan tuotteen evaluointiin ja vain sen avulla voidaan tarkastella käyttöliittymän käytettävyyden laatua. Analysoi, suunnittele, rakenna ja arvioi on erittäin käytetty yleispiirre käyttäjäkeskeisessä järjestelmäkehityksessä (Butler, 1996), jotta se voi päätyä tuotteeseen, joka käytettävyydeltään mieluinen ihmisille, jotka sitä käyttävät. Tärkeää on myös suunnitella käyttöliittymä riittävän kattavasti, ennen kuin se yhdistetään tuotteeseen. (Nummiaho, 2006)



Kuvio 2 Käyttäjäkeskeinen järjestelmäsuunnittelun metodi, muun muassa (Nummiaho, 2006)

Käyttäjäkeskeinen järjestelmäkehitys rakentuu käytännössä käytettävyystestauksen päälle. Järjestelmän ominaisuuksista ja käyttöliittymästä rakennetaan suunnitelmien pohjalta prototyyppejä, joita päivitetään uusien kehityskierroksen aikana. Testausten jälkeen voidaan kerätyn informaation pohjalta suunnitella uusi prototyyppi. Tätä käytännössä jatketaan, kunnes päädytään käytettävyydeltään riittävään loppuratkaisuun, joka ottaa niin käyttäjän tarpeet, vaatimukset, halut, kuin rajoitteetkin huomioon.(Nummiaho, 2006) ISO 13407 (1999) ottaa huomioon vielä tarkemmin organisaation ja se määrittelee neljä aktiviteettia, jota täytyy aloittaa projektin ensivaiheessa: käyttötarkoituksen tarkennus ja ymmärtäminen, käyttäjän ja organisaation vaatimuksien tarkennus, tuottaa suunnitelmia järjestelmäratkaisuista ja suunnitelmien arviointi vaatimuksiin nähden.

2.2.1 Mitä tarkoittaa käytettävyys?

Ihmiset ymmärtävät termin käytettävyys eri tavalla. Monille se tarkoittaa vain "helposti käytettävä" tai "käyttäjäystävällinen", eli termiä, joka esiteltiin käyttäjäkeskeisen järjestelmäkehityksen alkutaipaleella. Molempia edeltävistä termeistä näemme vielä usein projektin vaatimusmäärittelyissä yhdessä monien muiden ei-toiminnalliset vaatimuksien kanssa, vaikkakin termi sisältää itsessään kaikki, mitä niistä tiedetään koko tiedealalla.

Ihminen-tietokone vuorovaikutuksen tehokkuudelle, toisin sanoen käytettävyydelle on annettu monia määritteitä. IEEE Standardeista löytyy esimerkiksi seuraavanlaiset selitykset: "Ohjelmiston kapasiteetti, jolla sitä pystytään ymmärtämään, oppimaan, käyttämään ja houkuttelemaan käyttäjää tiettyjen olosuhteiden alaisena." ja toinen, jonka itse näen parempana, "Määre, joka kuvastaa, kuinka tyydyttävästi ja tehokkaasti tietyt käyttäjät saavuttavat ohjelmistolla heille määritetyt tehtävät." (Seffah, Gulliksen & Desmarais, 2005) Jacob Nielsen kuvaa kirjassaan Usability Engineering (1993) käytettävyyden attribuutteja:

- Opittavuus: kuinka nopeasti ensikäyttäjä oppii riittävän taidon käyttää uutta ohjelmistoa

- Tehokkuus: opittuaan järjestelmän, kuinka nopeasti käyttäjä voi suorittaa sillä tehtäviä
- Muistettavuus: kuinka hyvin käyttäjä muistaa, miten järjestelmää käytetään, jos on ollut käyttämättä sitä tietyn ajanjakson
- Virheet: Kuinka paljon käyttäjä tekee virheitä, kuinka haitallisia ne ovat ja kuinka käyttäjä toipuu virheistään
- Tyytyväisyys: kuinka mieluisaa järjestelmän käyttö on käyttäjälle

2.3 Käytettävyyden integrointi

Käyttäjäkeskeisen järjestelmäkehitysprosessien kehittyessä 1990-luvulla, huomattiin, että sekä käyttäjäkeskeinen, että ketterä järjestelmäkehitys jakavat samanlaiset päämäärät ohjelmistotuotteen lopputuloksesta. Oli pelkästään järkevää lähteä etsimään keinoa, kuinka nämä kaksi kehitysvaihtoehtoa voitaisiin yhdistää. (Nummiaho, 2006)

Tieto siitä, keitä käyttäjät ovat, mitkä heidän prioriteettinsa ja heidän aktiivisuutensa järjestelmän vaatimusten määrittämiseen usein tiedostetaan ketterässä järjestelmäkehityksessä. Kuitenkin heidän roolinsa ja kuinka heidän tulisi liittää osaksi ohjelmistoprojektia on aiheuttanut tutkimuskentällä riitoja. Käyttäjäkeskeinen suunnittelu (User Centered Design) on lähestymistapa, joka tähtää liittämään käyttäjän oikealla ja tuottavalla tavalla ohjelmiston kehittämisprojektiin sen koko elinkaaren ajaksi. (Chamberlain, yms., 2006)

John Gould ja Clayton Lewis (1985) esittelivät ensimmäiset kolme periaatetta käyttäjäkeskeiselle suunnittelulleen:

1. Aikainen fokus käyttäjiin ja heidän suorittamiin tehtäviin

Ensimmäisenä järjestelmäarkkitehtien täytyy ymmärtää, keitä loppukäyttäjät tulevat olemaan. Tämä saavutetaan osaksi tutkimalla suoraan heidän kognitiivisia, käyttäytymis-, antropometrisia ja asenteellisia ominaisuuksia ja toiseksi, tutkimalla heidän suorittaman työn luonnetta.

2. Empiirinen mittaus

Aikaisessa ohjelmiston kehitysvaiheessa, loppukäyttäjien täytyy osallistua järjestelmän käyttöliittymän testaukseen, jossa he suorittaisivat prototyypeillä oikeita tehtäviä. Heitä tulee tarkkailla ja suoritukset ja reaktiot tallentaa ja analysoida.

3. Iteratiivinen suunnittelu

Jos järjestelmästä löytyy pullonkauloja, tulee ne korjata mahdollisimman nopeasti iteratiivisella suunnittelulla. Tämä tarkoittaa, että itse suunnittelu ta-

pahtuu sykleissä ja testaus, mittaus, uudelleen suunnittelu toistetaan niin useasti kun tarve vaatii.

Nämä kolme periaatetta löytyy käytännössä kaikista uusimmista tutkimuksista ja malleista, joissa kuvataan käyttäjäkeskeistä ohjelmistokehitystä. Näiden jälkeen on kehitetty tarkentavia lisäperiaatteita varsinkin, jotka liittävät käyttäjän onnistuneesti ketterään ohjelmistoprojektiin. Käyttäjäkeskeisen ohjelmistosuunnittelun integroiminen ketterän ohjelmistokehityksen projektimalliin on potentiaalia auttaa järjestelmäarkkitehteja ymmärtämään paremmin käyttäjän tarpeen ja lisäämään heidät jo alkuvaiheessa ohjelmiston suunnitteluun.

2.4 Samankaltaisuudet ja erot käyttäjäkeskeisellä ja ketterällä ohjelmistokehityksellä

Käyttäjäkeskeisen, että ketterän ohjelmistokehityksen sisältävä projekti törmää haasteisiin vaikkakin molemmilta tavoilta löytyy yhteneväisyyksiä, ne sisältävät myös selviä eroja.

Yksi ketterän ja käyttäjäkeskeisen ohjelmistotuotannon yhdistämien ongelmista on lähtökohtaisesti resurssien suuntaaminen. Ketterät mallit pyrkivät tuottamaan asiakkaalle mahdollisimman nopealla aikataululla pieniä ohjelmisto-osia projektin alusta pitäen, kun taas käyttäjäkeskeiset mallit vaativat tavallisesti suhteellisen suuren panostuksen alkututkimukseen ja -analysointiin ennen, kuin itse ohjelmiston rakentaminen aloitetaan. Molemmat mallit ovat kuitenkin lähtökohtaisesti asiakassuuntautuneita. Käyttäjäkeskeisessä tavassa ohjelmisto rakennetaan loppukäyttäjän tarpeiden mukaan ja ketterässä kehityksessä asiakkaalla on projektissa mukana edustaja. Pyrkimyksenä on lyhentää palautteen saamiseen kuluvaa aikaa, jotta pystytään mahdollisimman nopeasti suunnittelemaan ja toteuttamaan projektin seuraava kierros.(Silva, Martin, Maurer & Silveira, 2011)

Alla on lueteltuna molempien projektimallien pääyhteneväisyydet:

1. Molemmat rakentuvat iteratiivisesta kehittämismetodista, jossa aikaisemman kierroksen lopuksi kerätyn empiirisen informaation avulla suunnitellaan seuraava, joko korjaava tai uutta tuottava kierros.

2. Ketterät menetelmät painottavat käyttäjän mukanaoloa ja vaikutusta koko prosessin elinkaaren ajan. Esimerkiksi Scrumissa tuotteelle tehdään ihannetapauksessa kuukausittain käyttäjäarviointi ja "tuotteen omistaja" on vastuussa vaatimuksista ja ohjelmiston ominaisuuksien priorisoinnista. Käyttäjäkeskeisen ohjelmistokehityksen toinen periaate vaatii käyttäjän aikaista ja jatkuvaa läsnäoloa projektissa.

3. Molemmat tyylit painottavat tiimityöskentelyn toimivuuden tärkeyttä. Beck (2000) esittääkin, että yksi pelin suunnittelun tärkeydestä on yhdistää joukkue. Lisäksi käyttäjäkeskeinen kehitys vaatii, että koko projektitiimillä on käyttäjä jatkuvasti mielessä, kun he kehittävät tuotetta.

Kaksi pääasiallista eroa ovat:

1. Käyttäjäkeskeinen järjestelmäkehityksen puolestapuhujat väittävät, että tietyt suunnitteluapuohjelmat ovat välttämättömiä ylläpitämään kommunikaatiota kehittäjien kanssa, kun taas ketterät menetelmät painottavat minimaallista dokumentointia.

2. Käyttäjäkeskeinen järjestelmäkehitys vaatii tiimiltä täydellistä ymmärrystä tuotteen loppukäyttäjistä ennen kuin riviäkään ohjelmistokoodia kirjoitetaan, kun taas ketterät menetelmät vastustavat ohjelmistokoodin kirjoittamisen kustannuksella tapahtuvaa esitutkimusta. (Chamberlain, ym. 2006)

2.5 Ongelmakohtia

Tutkimuksissa on huomattu, että käyttäjäkeskeinen järjestelmäkehitys törmää kahteen mittavaan ongelmaan jo sen integroimisvaiheessa. Jos haluamme vähentää projektin kokonaiskustannuksia, ensinnäkin johtoportaalle on perusteltava selkeästi numeroilla, miksi käyttäjäkeskeinen järjestelmäkehitys tulee integroida ohjelmistoprojektiin.

Johtoportaan hyväksyessä uusi tapa, törmätään seuraavaksi ongelmaan, jossa itse järjestelmäkehittäjät eivät halunneet hyväksyä uutta tekniikkaa. Vaikka heitä painostettiin jopa johdon taholta, he joko sivuuttivat aiheen kokonaan tai vastasivat siihen formaalisti ja tulokset jäivät laihoiksi. Tämä tarkoittaa lyhyesti sanottuna, että käyttäjäkeskeinen ohjelmistokehitys ei paranna lopputuotteen laatua tai tuo lisäarvoa ohjelmistoprojektille, jos koko organisaatio ei sitoudu siihen. (Hirasawa ym., 2010) Luvussa 3.1 pilottiprojektissa huomataankin käytännön tasolla, että jopa yksittäiset henkilöt voivat päätöksillään saada koko projektin epäonnistumaan ja eliminoimaan lopputuloksen vähintään käytettävyyden näkökulmasta.

3 Projektimalli ketterälle käyttäjäkeskeiselle ohjelmistokehitykselle

Tässä luvussa käsittelen, miten ketterää käyttäjäkeskeistä järjestelmäkehitystä on jo kokeiltu käytännössä ja mitä malleja sille on olemassa. Tarkastelen myös, mitä täytyy vielä saada aikaan, jotta saamme kehitettyä tavan integroida käyttäjäkeskeinen järjestelmäkehitys perinteiseen ketterän kehityksen projektiin. Rajaan alueen projektien hallinnalliselle tasolle, enkä erittele ja etsi teknologioita ja työkaluja, joilla projektien osa-alueita toteutettaisiin käytännöntasolla.

3.1 Pilottiprojekti

Seffah, Gulliksen ja Desmarais (2005) suorittivat pilottiprojektin, jossa Ruotsissa tehdyssä kehitysprojektissa otettiin käyttöön käyttäjäkeskeinen lähestymistapa ja koko projekti rakentui vahvasti Gouldin ja Lewisin esittelemien periaatteiden päälle. Lisäksi he ottivat mukaan järjestelmän prototyyppisuunnittelun mahdollisimman aikaisessa vaiheessa.

Lupaavan projektin lopputulos jäi kuitenkin käytettävyyden näkökulmasta heikoksi. Käytännössä saatiin vain selvä siitä, ettei käyttäjäkeskeiselle järjestelmäkehitykselle tiedetty oikeanlaista tapaa yhdistää sitä perinteiseen ketterän kehityksen projektiin. Projektissa havaittiin seuraavanlaisia seikkoja. Vaikka mukana oli kokeneita käytettävyysasiantuntijoita, heitä ei käytännössä enää kuunneltu projektin loppupuolella. Käytettävyysasiantuntijat saivat kerättyä huomattavan määrän informaatiota esimerkiksi käyttötapauksista, mutta kaaviot, joilla niitä kuvattiin kasvoivat epäedullisen suuriksi, eikä niitä enää pystytty käyttämään tehokkaasti suunnitteluprosessissa. Ketterän kehityksen projektin tapaan, se dokumentoitiin UML:llä ja kun käyttäjät kutsuttiin arvioimaan sitä, heillä oli huomattavia vaikeuksia ymmärtää edes osaa suunnitelmasta. Niinpä käyttäjän perspektiivi jäi käytännössä kokonaan pois projektin suunnitteluvaiheessa. Käyttäjät pääsivät jyvälle vasta, kun heille esiteltiin ensimmäiset prototyypit järjestelmästä

Projektin asiakasorganisaatio halusi vaihtaa järjestelmän kehitysympäristön web-pohjaiseksi. Projektin sisällä oli kuitenkin huomattava taidonpuute kehittää järjestelmiä uudella, vaaditulla tavalla ja käytettävyysasiantuntijat esittivätkin, että lopputuotteen laatu ei tule vastaamaan sille asetettuja vaatimuksia. Tästä huolimatta projektin kehitys muutettiin.

Projektin loppuevaluoinnissa havaittiin, että yksittäiset henkilöt ovat avainasemassa, mitä tulee lopputuotteen käytettävyyden laatuun. Huomion arvoista oli, että ongelmia syntyi yksilötasolla liittyen ristiriitojen ratkaisuun henkilökohtaisten ja liiketoiminnallisien maalien välillä.

Tämä pilottiprojekti toi selvästi esiin seuraavanlaisia seikkoja, joiden puute vaarantaa lopputuotteen käytettävyyden laatua radikaalisti:

- käytettävyysasiantuntijoilla ei ole riittävän vahvaa päätäntäasemaa projektissa
- tehoton suunnitelma, joka määrittelee, mitä informaatiota käyttäjästä kerätään ja miten kerätty informaatio otetaan mukaan järjestelmän suunnitteluun projektissa tehokkaasti
- järjestelmädokumentaation laatu sellaista, joka ei ole projektin kaikkien osapuolien ymmärrettävissä
- yksittäiset ihmiset projektin avainasemissa, jotka eivät ymmärrä tai asenteidensa vuoksi välttämättä edes halua ymmärtää käytettävyyden tarvetta ja merkitystä lopputuotteen kannalta

3.2 Projektimallin rakentaminen

Kuten edellisessä luvussa esitelty pilottiprojekti osoittaa, käytettävyyden integroiminen ketterän kehityksen malliin ei ole miltään osin yksiselitteistä. Kuitenkin elementit, joista käyttäjäkeskeinen ohjelmistokehityksen malli luodaan, on olemassa. Osama Sohaib ja Khalid Khan (2010) esittelevät artikkelissaan neljä perusvaihetta ohjelmiston suunnitteluprosessille:

1. Tunnista tarpeet ja määrittele vaatimukset
2. Suunnittele mahdolliset ratkaisut (iterointi mahdollista, jopa suotavaa)
3. Valitse vaihtoehtojen väliltä
4. Rakenna tuote

Käyttäjäkeskeisen järjestelmäkehityksen tarkoituksena on siis kerätä informaatiota todellisilta loppukäyttäjiltä järjestelmän rakentamisvaiheessa, jotta vältytään lopputuotteen epäonnistumiselta. Missä vaiheessa järjestelmän kehittämisprojektia tuo informaatio kerätään, jakaa tutkimuskenttää tällä hetkellä. Joidenkin tutkimuksien mukaan käyttöliittymä toteutetaan tuottamalla hyvin vähän työaikaa vaativia, esimerkiksi paperisia tai Power Point- tyylisiä käyttöliittymiä, joita voidaan uudelleen suunnitella ja toteuttaa käyttäjätestauksesta saa-

dun informaation perusteella. Tämä tapahtuu kokonaisuudessaan ennen, kuin riviäkään itse ohjelmakoodia on kirjoitettu. (Chamberlain yms., 2006) (Ferreira, Noble & Biddle, 2007) Osa taas, kuten Miller(2005) kannattavat suunnittelua ohjelmointipyrähdyksien aikana ja välissä keräämällä loppukäyttäjiltä palautetta tuotetuista ominaisuuksista.

Silva, Martin, Maurer ja Silveira(2011) kuvasivat projektin sisällön systemaattisessa kirjallisuuskatsauksessaan. He kannattavat käyttöliittymän suunnittelemista ennen ohjelmakoodin kirjoittamista. Käymällä läpi satoja tutkimuksia he tunnistivat selvät toimenpiteet, joilla käyttäjäkeskeinen ohjelmistokehitys voitaisiin integroida ketterään kehitykseen.

Pieni etukäteissuunnittelu (Little design up front) on vaihtoehto massiiviselle etukäteistutkimukselle, joka liittyy perinteiseen käyttäjäkeskeiseen ohjelmistokehitykseen. Useimpien artikkeleiden mukaan ideana on luoda suhteellisen pienellä panoksella käyttäjien haastattelujen avulla käyttäjätarinoita, joiden avulla luodaan ongelmatilanteita. Näiden pohjalta suunnitellaan malli tulevasta käyttöliittymästä, joka ratkaisee käyttäjien kyseiset ongelmat. Tärkein idea on, ettei ohjelmiston rakentamista aloiteta tyhjästä, vaan projektin osapuolilla on selkeä kuva lopputuotteesta ennen ensimmäistäkään riviä koodia.

Prototyyppien rakentaminen (prototyping) nähdään hyödyntävän kahta seikkaa ohjelmistoprojektissa. Ne yksinkertaistavat kehittäjien ja käytettävyyssuunnittelijoiden välistä kommunikaatiota ja niiden avulla on huomattavasti helpompaa kerätä palautetta loppukäyttäjiltä. Prototyypit tehdään sellaisiksi, että loppukäyttäjä pystyy ymmärtämään ne täysin ja hänelle täytyy tulla niiden perusteella oikeanlainen kuva lopputuotteesta. Näin heiltä saadaan huomattavasti tehokkaammin arvokasta palautetta. Ketterässä kehityksessä ongelma on usein käyttäjien heikko ymmärrys heille esitetyistä kaavioista tulevasta järjestelmästä, joten he eivät voi antaa todellista ja käytettävää palautetta tuotteesta. Prototyypit voidaan suunnitella aiemmin tuotettujen käyttäjäkertomusten pohjalta ja kirjoittajat ovat sitä mieltä, että käytettävyysasiantuntijat toimivat projektissa yhden pyrähdyksen tai kierroksen edellä järjestelmäkehittäjiä.

Käyttäjätestaus (user testing) on tärkeä osa käyttöliittymäsuunnittelua. Se voidaan käytännössä toteuttaa monella eri tapaa ja metodeita voidaan myös yhdistää. Prototyyppi voi olla paperiversio tai valmista ohjelmistoa ja käyttäjän tulisi jatkuvasti puhua ääneen, mitä hänen mielessään liikkuu prototyypistä. Kuvankaappaus ja hiiren liikkeiden ja painallusten määrä voidaan ottaa ylös jos kyseessä on riittävän pitkälle toteutettu käyttöliittymäprototyyppi.

Käyttäjätarinoiden (user stories) tarkoitus on selventää järjestelmävaatimuksia. Niihin tuodaan selkokielellä yksittäisiä tapauksia, miten käyttäjä haluaa suorittaa tietyn yksittäisen tehtävän, tai miten hän haluaa järjestelmän auttavan tietyssä tilanteessa. Toisin sanoen käyttäjätarinat tuotetaan skenaarioista, jotka taas johdetaan järjestelmävaatimuksista, tehtäväanalyyseistä ja käyttäjätutkimuksista. Käyttäjätarinoiden pohjalta aloitetaan prototyyppien, ensi vaiheessa paperiversioiden, kehittäminen. Lisäksi käyttäjätarinoita tulee päivittää aina, jos tietyn kierroksen käyttäjätestauksessa löydetään niitä selventäviä ja parantavia ominaisuuksia, jotka tulee liittää lopputuotteeseen. Lisäksi artikke-

lissa ehdotettiin, että käytettävyysasiantuntijoiden tulee olla koulutettuja kirjoittamaan käyttäjätarinat riittävän teknisestä näkökulmasta, että niiden pohjalta voidaan aloittaa suoraan kyseessä olevan osion ohjelmointi. Lisäksi, omasta mielestäni käyttäjätarinat ja käyttöskenaariot ovat yksi ja sama asia, vaikka osasta artikkeleista käyttäjätarinat on esitetty skenaarioista vielä seuraavana vaiheena.

Käytettävyystarkistus (usability inspection) tapahtuu ketterän kehityksen kehityskierroksen viimeisessä vaiheessa, jopa mahdollisesti loppuarvioinnin yhteydessä. Lopputuloksena määritetään käyttöliittymän korjattavat ominaisuudet seuraavaa kierrosta varten. Käytettävyystarkistus tapahtuu niin pitkään, kun käyttöliittymästä tulee riittävän vakaa, jotta se voidaan liittää ohjelmiston päälle. M. Albisetti (2010) raportoi, kun järjestelmäkehittäjille annettiin tehtäväksi tuottaa käyttöliittymäarviointeja, se muutti heidän näkemystään käytettävyysasiantuntijoiden työstä täysin. Nähdessään, millaisiin ongelmiin käyttäjät törmäsivät käyttöliittymän kanssa, vaikka ohjelmakoodi oli kehittäjien mielestä hienoa, sai heissä aikaan ennen näkemättömän vaikutuksen.

Omasta mielestäni käytettävyystarkistus on yksi tärkeimmistä käytettävyysasiantuntijoiden toimenkuvasta projektissa. Sen avulla löydetään yksittäiset, huonosti toteutetut ominaisuudet tuotteesta välittömästi ja ne voidaan korjata heti seuraavan kierroksen aikana. Jos käytettävyystarkistus toteutetaan vasta lopputuotteelle, voi huonosti suunniteltuja ominaisuuksia olla jo niin paljon, ettei kaikkea pystytä löytämään ja vielä vähemmän korjaamaan varsinkin, jos halutaan pysyä budjetin raameissa. Myös halu korjauksiin sekä ohjelmiston tuottajan, että asiakkaan puolella tässä vaiheessa on heikko.

Ensimmäisenä täytyy siis tunnistaa, millaisia loppukäyttäjiä tuotettavalla ohjelmistolla on. Tämä tapahtuu työskentelyä loppukäyttäjien parissa ja kerätä heistä mahdollisimman paljon informaatiota, joka helpottaa ohjelmiston räätälöintiä juuri heille. Käyttäjistä kerätyn informaation avulla kartoitetaan ohjelmiston vaatimukset. Näiden pohjalta aloitetaan itse käyttöliittymän suunnittelu, jonka aikana luodaan useita malleja, jotka testataan loppukäyttäjillä heille sopivimman vaihtoehdon löytämiseksi. Tässä vaiheessa on myös suotavaa iteroida, eli loppukäyttäjien palautteen pohjalta suunnitella uudelleen ja korjata malleja, poistaa ja lisätä yksiköitä, jotta kokonaisuudesta saadaan mahdollisimman toimiva. Tällainen toimintatapa tuo käyttäjän todellisesti mukaan itse käyttöliittymän suunnitteluun, heiltä saadun palautteen pohjalta ohjelmistoa suunnitellaan uudelleen ja iterointi voidaan toistaa, ja sitä tulisi toistaa niin pitkään, kunnes huonoa palautetta ei enää tule.

3.3 Olemassa oleva mallipohja

Seuraavaksi tarkoituksena on esitellä omasta mielestäni toimivia ja jo olemassa olevia malleja käyttäjäkeskeiselle järjestelmäkehitykselle. Näissä kaikissa on lähtökohtana ketterän ja käyttäjäkeskeisen järjestelmäkehityksen integrointi.

3.3.1 Skenaariopohjainen järjestelmäkehitys

Skenaariopohjaisen järjestelmäkehityksen mallissa on paneuduttu lähtökohtaisesti vanhaan ongelmaan; ohjelmistoarkkitehtien ja käytettävyysinsinöörien, yhteisymmärryksen puuttuminen on useimmiten nähty suurimpana ongelmana käyttäjäkeskeisien ja ketterän kehityksen mallien yhdistämisessä.

Lee ja McCrickard (2007) esittelivät tutkimuksessaan skenaariopohjaisen käyttäjäkeskeisen ohjelmistokehitysmallin, jonka pääasiallisena tarkoituksena oli ratkaista ongelmat liittyen ohjelmistokehitystapojen kahteen päätarkoitukseen; ketterän kehityksen vauhdikas tahti ja käyttäjäkeskeisen kehityksen seikkaperäinen loppukäyttäjien analysointi ja testaus tuli saada yhdistettyä. He huomasivat, että oletus käytettävyyden synnystä tuotteeseen pitämällä aktiivinen asiakas projektissa mukana, ei pitänyt paikkaansa. Mutta antamalla järjestelmäkehittäjille, heitä kiinnostavalla tavalla, huomattavasti syvemmän ymmärryksen käyttäjistä, heidän työstään ja työtavastaan ja kuinka he pystyvät saavuttamaan maalit kaikkein tehokkaimmin, tuotti kiinnostusta itse lopputuotteen käytettävyyteen ketterässä järjestelmäkehitysyhteisössä.

Skenaariopohjainen suunnittelu perustuu nimensä mukaisesti skenaarioihin ja lisäksi väittämiin, jotka kuvaavat loppukäyttäjien aitoja järjestelmän käyttötilanteita ja alleviivaa käyttöliittymän tiettyjen ominaisuuksien kompromisseja. Ketterässä järjestelmäkehityksessä käytetään samankaltaisia tarinoita käyttäjistä, mutta nämä skenaariot voivat sisältää paljon enemmän järjestelmän ominaisuuksia ja kuinka yksi tai useampi ihminen osallistuu tiettyyn toimintaan.

Esimerkkiskenaario:

> Pascal, opiskelija, työstää tutkimusta ja lukee siihen liittyvää artikkelia. Lukiessaan artikkelia, hän toivoo saavansa laboratoriossaan tiedon, jos joku jakaa hänen tutkimukseensa liittyvää informaatiota. Hän käyttää ilmoitustauluohjelmaa, joka on auki hänen toisella monitorillaan, jotta hän olisi jatkuvasti tietoinen tällaisen informaation jaosta. Hän voi silloin tällöin vilkaista ilmoitustaulua nähdäkseen jaetut uutiset tai artikkelit. Kun Pascal katsoo ilmoitustaulua, hän käy silmillään melko satunnaisesti läpi uusimmat ja päällimmäisenä olevat kohteet. Katsoessaan otsakkeita, hän saa ymmärryksen, mitä jaettu informaatio sisältää, mutta ei tiedä onko informaatio miten uutta. Tietäen, että hänen täytyy etsiä myöhemmin, milloin kyseinen artikkeli jaettiin, hän palaa tutkimuksensa pariin.

Skenaariosta johdetaan väitteitä ja yllä olevasta mainitaan seuraavanlaiset:

Ilmoitustaulu on hyvin samankaltainen, kuin vanha lappuja sisältävä todellisen elämän vastaavansa. Siinä julkaistaan sattumanvaraisesti informaatiolähteitä vain kätevämmällä tavalla ja ne ovat samanaikaisesti kaikkien nähtävillä.

+Antaa käyttäjille ymmärryksen kohteen iästä ja sovellettavuudesta vaikka uusia kohteita olisi julkaistu useampi tämän jälkeen.

+ Informaatiota ei kategorioida, joten ilmoitustaululla voidaan jakaa monenlaatuisia lähteitä.

- Organisoimattomuuden takia liiallinen materiaalin julkaisumäärä voi vaikeuttaa tietyn informaation löytämistä ja tunnistamista

- Vaikka kohteet tulevat julkaisujärjestyksessä näytölle, niiden todellinen ikä (milloin mikäkin artikkeli on julkaistu) voi olla suhteellisesti täysin eri.

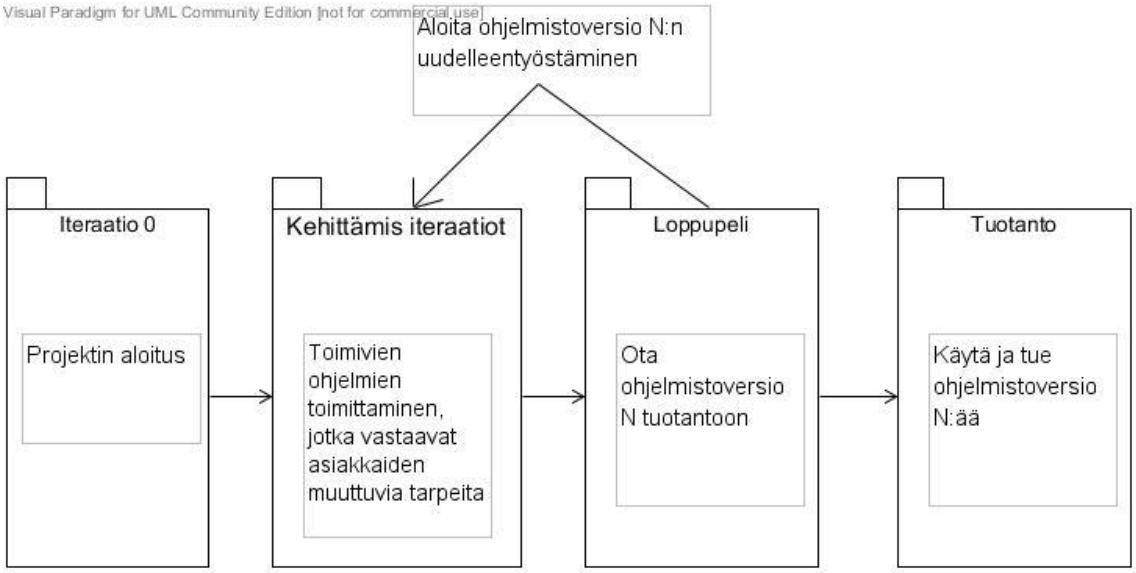
Näiden pohjalta voidaan lähteä suunnittelemaan, minkälainen kyseisen ilmoitustaulun tulee olla, jotta se on käyttäjälle sellainen, minkä hän tarvitsee ja jolla on hänelle todellista hyötyä. Omasta mielestä olisi suositeltavaa, että järjestelmän jokaisesta ominaisuudesta tehtäisiin vastaavanlainen selvitys. Näiden pohjalta on huomattavasti helpompaa suunnitella työkalu, josta on todellista hyötyä loppukäyttäjälle. Vähintäänkin järjestelmän eniten käytetyistä työkaluista pitäisi olla kaikki käyttäjän tarvitsemat ominaisuudet selvillä.

Skenaariopohjainen suunnittelu koostuu neljästä vaiheesta. Vaatimusanalyysissa suunnittelijat keräävät informaatiota nykyisistä käytännöistä haastattelemalla, tutkivat yhteisön etnografiaa erilaisilla informaation keräysmetodeilla. Kerättyä informaatiolla rakennetaan juurikonseptidokumentti, joka määrittelee järjestelmästä kokonaisnäkemyksen ja sidosryhmien kuvaukset. Suunnittelijat voivat tämän jälkeen työstää ongelmaskenaariot ja -väitteet kuvaamaan, miten käyttäjät tällä hetkellä tekevät työtään ja mitä avainongelmia on olemassa.

Toimintasuunnittelu on toinen vaihe ja siinä suunnittelijat kehittävät skenaarioita ja väitteitä kuvaamaan aktiviteetteja ja tehtäviä, joita uusi järjestelmä tulee tukemaan. Apuna käytetään edellisessä vaiheessa tuotettuja ongelmaskenaarioita ja -väitteitä. Informaatio- ja vuorovaikutussuunnitteluvaiheessa suunnittelijat päättävät, miten käyttöliittymän antama informaatio tukee käyttäjän aktiviteetteja ja heidän välistä vuorovaikutusta. Käyttöliittymä kannattaa pitää tässä vaiheessa erossa ohjelmakoodista, jolloin käytettävyysasiantuntija voi muutella prototyyppiä käyttäjätestausten mukaan. Tarpeen vaatiessa koko käyttöliittymän tyyli voidaan muuttaa vielä tässä vaiheessa ilman vakavia projektin aikataulullisia seuraamuksia. (Ferreira, Noble & Biddle, 2007)
Neljä vaihetta kuvataan jaksoittain, mutta todennäköisesti ne sekoittuvat keskenään ohjelmistoprojektin edetessä iteratiivisesti. Esimerkiksi informaatio- ja vuorovaikutussuunnittelut voivat tapahtua yhdellä iteroinnilla, jonka jälkeen voi seurata käytettävyyden arviointi, joka voi vaatia suunnittelijoita uudelleen arvioimaan kokonaisaktiviteettisuunnittelua. (Lee yms., 2007)

3.3.2 Amblerin malli

Amblerin malli koostuu neljästä vaiheesta, jotka ovat iteraatio 0, kehittäminen, loppupeli ja tuotanto.



Kuvio 3 Järjestelmäkehityksen elinkaarimalli, mukaillen (Ambler, 2008)

Ketterän kehityksen projektit käyttävät yleisesti ketterän kehityksen ohjelmistotkehityksen elinkaari-mallia (the Agile Software Development Lifesycle, SDLC) projekteissaan. Nopeasti tarkasteltuna Amblerin malli näyttä hyvin paljon samankaltaiselta, mutta tarkemmin katsottuna ei sitä ole. Tässä mallissa korostuu asiakkaan kanssa kiinteästi työskentelevien kehittäjien rooli. Heidän vastuullaan on ohjelmistoversioiden testaaminen, hyväksyminen ja palauttamisesta uuteen kehittämisiteraatioon. Tämä tapahtuu jatkuvalla yhteistyöllä asiakkaan kanssa, jolloin pystytään vastaamaan nopeasti muuttuviin järjestelmävaatimuksiin ja testaamaan tuotettuja ominaisuuksia oikeilla käyttäjillä. (Ambler, 2008)

Projektin ensimmäistä viikkoa kutsutaan tavallisesti iteraatio 0:ksi. Tarkoituksena on käynnistää projekti, eli löytää rahoitus, työstää asiakasosapuolen kanssa yleisellä tasolla projektimalli, jota tullaan käyttämään ohjelmiston työstämisessä. Lisäksi kootaan tiimi ja pystytetään työskentely-ympäristö.

Kehitysiteraatioissa kehittäjät tuottavat korkean tason toimivaa ohjelmakoodia, joka vastaa asiakkaiden muuttuvia tarpeita. Käytettävyysasiantuntijat tekevät yhteistyötä läheisesti sekä asiakasorganisaation, että kehittäjien kanssa; analysoivat tarpeita ja vaatimuksia ja suunnittelevat malleja järjestelmästä. He tuovat myös tärkeänä osana malliin mukaan testien ajaman kehityksen, jossa ohjelmistoversiot palaavat työpöydälle, jos eivät pääse läpi, niin käyttäjä-, kuin järjestelmätestauksia. Malli siis puoltaa ajatusta, jossa mallit suunnitellaan samanaikaisesti ohjelmakoodin kanssa.

Loppupelin vaiheessa kehitysversio työstetään tuotantoon kelpaavaksi. Tämäkin vaihe sisältää käyttäjä-, että järjestelmätestauksen, jonka pohjalta muokkauksia voidaan vielä tehdä. Vaiheeseen sisältyy myös dokumentointien viimeistelyn ja loppukäyttäjien ja asiakkaan teknisen tuen kouluttamisen järjestelmän käyttöönottoa varten. Viimeiseksi järjestelmä otetaan käyttöön.

Amblerin mallin viimeinen vaihe on tuotanto. Se sisältää tarpeen vaatiessa teknisen tuen antamista asiakkaalle ja tuotteen ylläpitämistä toimivana. Vaiheen pohjimmainen tarkoitus on pitää järjestelmä toimintakykyisenä ja tuottavana. (Ambler, 2008)

4 Käyttäjäkeskeisen ohjelmistokehityksen tuoma arvo

Käyttäjäystävällinen lopputuote tuo lisätehokkuutta itse ohjelmistoprojektiin, sekä tuotteen käyttöönottoprosessiin. (Constantine & Lockwood, 2003) Näillä on potentiaalisesti erittäin suuri vaikutus asiakastaholle ohjelmistoprojektin kokonaiskustannuksiin lisääntyneen tehokkuuden ansiosta projektin aikana ja elinkaaren loppuvaiheessa vähentämällä käyttöönotto- ja varjokustannuksia. En ole löytänyt suomenkielistä vastinetta englannin termille "cost benefits", eli kustannushyödyille, joten käytän termiä varjokustannukset, joka mielestäni kuvaa hyvin yrityksille koituvia näkymättömiä kustannuksia. Nämä syntyvät heikosta käytettävyydestä, joka johtaa muun muassa lisääntyneeseen koulutuksen tarpeeseen, pidempiin tehtävien suoritusaikoihin ja vaikeasti omaksuttavan tuotteen muutosresistanssista. (Bevan, 2005)

Tutkimuksien avulla on todistettu, että käyttäjäystävällisellä järjestelmäkehityksellä on potentiaalia tuoda lisäarvoa tuotteelle. Useissa lähteissä mainitaan seuraavanlaisia faktoja:

1. Kehityskustannuksia voidaan alentaa
2. Internetissä käytävän kaupankäynnin tuottoja voidaan kasvattaa käyttäjiä miellyttävällä nettisivulla
3. Yksittäisen tuotteen myyntiä voidaan parantaa tuotteen esteettisyydellä ja käyttömukavuudella (tästä loistavana esimerkkinä Applen tuotteet).
4. Alihankkijoiden ja työntekijöiden aiheuttamia kustannuksia voidaan pienentää hyvin suunnitellulla käyttöliittymällä vähentäen yrityksen/laitoksen ylläpitokustannuksia, toisin sanoen työntekijöiden tehokkuus kasvaa, kun heiltä kuluu vähemmän aikaa suorittaa tehtäviä ohjelmistolla.

4.1 Ohjelmistoprojektin tehokkuus

Kun käyttäjäystävällinen ja ketterä ohjelmistokehitys yhdistetään, saadaan aikaan säästöjä myös ohjelmistoprojektin aikana. Käytettävyysasiantuntijoiden mukaantulo projektiin saattaa paperilla lisätä kustannuksia, mutta pitkällä aikavälillä tehokkuus korvaa aiheutuneet lisäkustannukset.

Tehokkuutta lisää käytettävyysasiantuntijoiden tekemät käyttäjätarinat, joiden pohjalta voidaan tuottaa tuote, joka sisältää loppukäyttäjille tarpeellisia ominaisuuksia. Lopputuotteeseen ei näin tule ohjelmakoodia jonka lopullinen funktio ei ole asiakkaan ja käyttäjien puolesta olennaista. Lisäsäästöä tuo myös testauksen tarpeen ja projektin ohjelmointipyrähdyksien vähentyminen. (Bevan, 2005) Perinteisessä ohjelmistossa ja vielä enemmän Internetin kaupankäynnissä, käyttäjät kuluttavat 95% aikaa käyttäessään 5%:a järjestelmän ominaisuuksista. Vielä huikeampi tilasto on, että 70% käyttöliittymän suunniteltuja ominaisuuksia ei käytetä koskaan, tai hyvin harvoin. (Mauro, 2002)

Tehokkuutta lisää myös ohjelmiston ongelmien havaitseminen mahdollisimman aikaisessa vaiheessa. (Bevan, 2005) Muutoksen tekeminen voi kustantaa 1.5 yksikköä projektin resursseista suunnitteluvaiheessa, 6 aikaisessa ohjelmointivaiheessa, 60 lopputestauksen aikana ja 100 jälkiylläpidollisessa vaiheessa.(Pressman, 1992)

Hyvin suunniteltu ohjelmisto vähentää kustannuksia myös siinä vaiheessa, kun järjestelmää lähdetään päivittämään tulevaisuudessa. (Bevan, 2005) MauroNewMedian tuottaman Internetin kaupankäyntiä koskevassa suuressa tutkimuksessa kuluttajille tarkoitetun Internet-sivun hakukone tuotti puutteellisen hakutuloksen 57% kaikista hauista. Lopputuloksena 46% kuluttajista lähti sivustolta löytämättä haluamaansa tuotetta, vaikka se olisi sivustolta löytynyt. Ongelma olisi löytynyt pienellä käytettävyystestauksella, mutta lopulta koko systeemiarkkitehtuuri täytyi uusia uuden version myötä ja kokonaiskustannukset olivat yli miljoona dollaria. Käytettävyystutkimuksen teettäminen edellisen version kehitysvaiheessa olisi maksanut noin 25 000 dollaria. (Mauro, 2002)

Myös helppokäyttöinen lopputuote tuottaa asiakkaalle lisäarvoa. Käytettävyyden eteen tehty työ lopetti eräältä yritykseltä tarpeen tulostaa uudelleen ja jakaa käyttöoppaita, mikä tuotti säästöjä kyseiselle yritykselle 40 000 dollaria yhdessä vuodessa. (Bias & Mayhew, 1994)

Kuitenkin käyttäjäystävällisellä ohjelmistokehityksellä saavutettava tärkein tavoite mielestäni on Bevanin(2005) mainitsema lopputuotteen epäonnistumisen riskin pieneneminen. Eli asiakas saa sellaisen tuotteen, jota hänen työntekijänsä, asiakkaansa tai alihankkijansa oikeasti tarvitsevat. Miller(2005) mainitsi oikeanlaisen tavan kerätä informaatiota loppukäyttäjiltä ja tuoda informaatio ohjelmistoprojektiin ennen, kuin tuote julkaistaan. Tämä on ehdottoman tärkeää varsinkin Internetissä tapahtuvalle kaupankäynnille, joka kärsii huomattavasti lopputuotteen heikosta käytettävyydestä. Tällaisissa projekteissa käytettävyyssuunnittelu on omiaan lisäämään ohjelmistoyrityksen asiakkaalle toimittavaa arvoa. The Standish Groupin(2002) teettämän tutkimuksen mukaan vain

34% ohjelmistoprojekteista valmistui ajallansa, ei ylittänyt budjettiaan ja toimitti vaadittavat toiminnallisuudet sisältävän ohjelmiston; 51% ohjelmistoprojekteista oli "vaikeuksissa" tarkoittaen, että ne eivät olleet aikataulussa, budjettiraameissaan tai eivät tuottaneet tarpeellisia toiminnallisuuksia omaavaa ohjelmistoa; 15% ohjelmistoprojekteista epäonnistui tai hylättiin. Yksi tärkeimmistä syistä oli riittämättömät käytettävyysvaatimukset. (Bevan, 2005)

4.2 Hyöty käyttöönotto- ja ylläpitokustannuksissa

Hyvin suunniteltu käyttöliittymä auttaa käyttäjää alun pitäen keskittymään itse tehtävään, eikä sen suorittamiseen vaadittavaan työkaluun. Kun käyttöliittymä on suunniteltu oikeiden tehtävien oikeanlaiseen suunnitteluun, se vapauttaa käyttäjän suuntaamaan tehokkuutensa tehtävän suorittamiseen, eikä tämän tarvitse kuluttaa voimavarojaan painimiseen huonosti suunnitellun käyttöliittymän kanssa. (Bevan, 2005) Informaatioteknologiaa paljon käyttävät organisaatiot voivat säästää huomattavia määriä rahaa minimoimalla inhimilliset hidasteet ohjelmistosta. Esimerkiksi IBM käytti 20 700 dollaria käytettävyystyöhön parantaessaan sisään kirjautumisen. Lopputuloksena yritys säästi 41 700 dollaria uuden kirjautumisjärjestelmän ensimmäisenä käyttöönottopäivänä ja vuoden aikana 6 800 000 dollaria, mikä tarkoittaa säästösuhdetta 1:100.(Bias & Mayhew, 1994) (Karat, 1993).

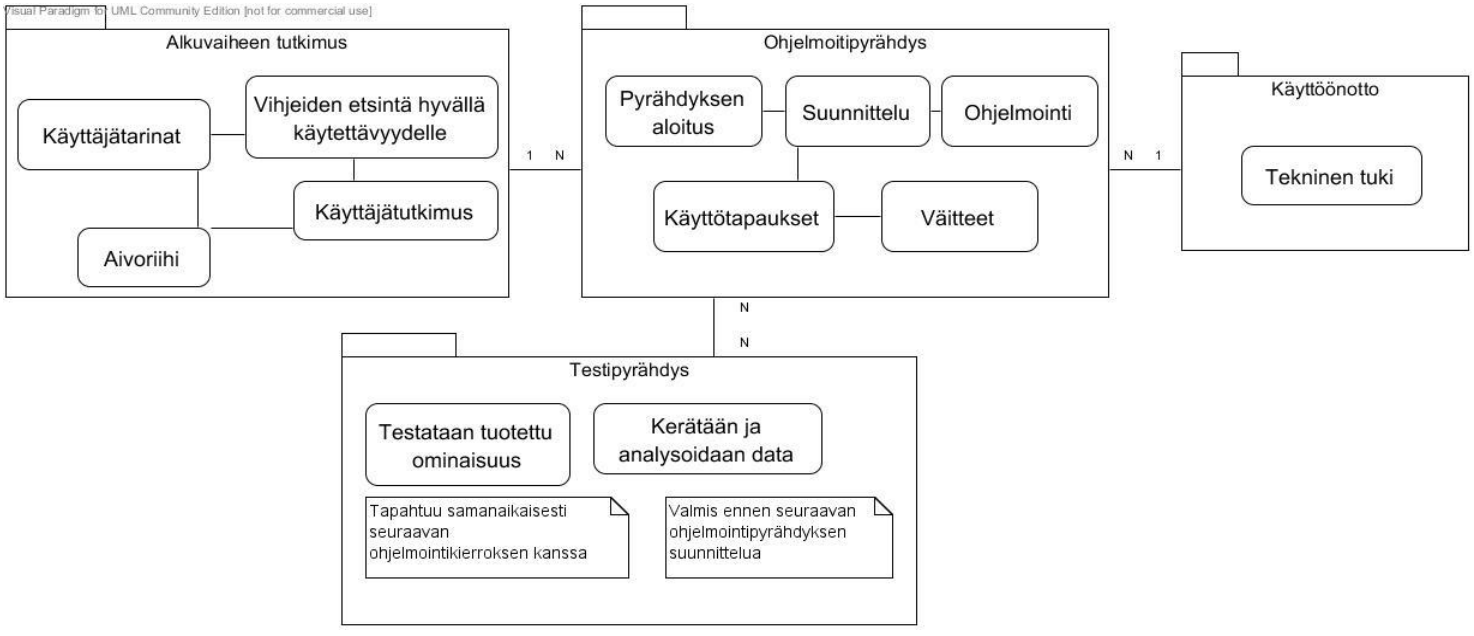
Käyttäjäkeskeisellä järjestelmäsuunnittelulla voidaan myös vähentää merkittävästi käyttäjien tekemiä virheitä, joita joudutaan korjaamaan myöhemmin. Tällä on vaikutusta tuotantoon ja palveluiden tasoon, mikä kasvattaa yrityksen arvoa.

Käyttäjää miellyttävä käyttöliittymä kasvattaa usein työntekijöiden työtyytyväisyyttä. Gartnerin(1992) tekemän tutkimuksen mukaan käytettävyysmetodeiden käyttö järjestelmäsuunnittelussa paransi käyttäjätyytyväisyyttä 40%(Bias ym., 1994). Lisäksi työtyytyväisyyden lisääminen vähentää työntekijöiden vaihtuvuutta työpaikalla, joka vähentää henkilöstökustannuksia.(Karat, 1993) Lisäksi voidaan mainita, että hyvin suunnitellun järjestelmän myötävaikutuksella voidaan vähentää kontaktien määrää yritysten tekniseen tukeen niin sähköpostilla, kuin puhelinsoitoilla, mikä vähentää yrityksen ylläpitokustannuksia (Bevan, 2005).

5 Loppupäätelmät

Kandidaatin tutkielmassani olen löytänyt todellisia syitä, miksi käyttäjäkeskeinen järjestelmäkehitys tulisi ottaa mukaan ketterän kehityksen projektimalleihin. Vaikka kehityskustannukset näyttävät paperilla nousevan, voidaan sen avulla päästä säästöihin kasvaneen tehokkuuden ansioista projektin elinkaaren eri vaiheissa. (Bevan, 2005) Lisäksi käytettävyydeltään asiakkaan kannalta oikeanlainen tuote on arvo, josta luulisi ohjelmistotalojen kilpailevan. Suurin syy käyttäjäkeskeiseen ohjelmistokehitykseen on kuitenkin asiakkaalle koituvat säästöt lopputuotteen käyttöönotto- ja ylläpitokustannuksista. (Bias ym., 1994) Ne aiheuttavat näkymättömiä kustannuksia, jotka pelkästään heittävät varjonsa yrityksen tulokseen.

Käyttäjäkeskeisen ja ketterän ohjelmistokehityksen yhdistämisestä on reilusti tutkimustuloksia, mutta itse toimivat käytännönsovellukset näyttävät vielä puuttuvan. Syynä on yleensä organisaatiorakenne, johon on vaikea saada aikaan muutoksia ja, että pelkästään yksittäiset ihmiset voivat toiminnallaan saada projektin epäonnistumaan. (Seffah ym., 2005) Yhdistelemällä tutkimustuloksia (Osama ym.,2010)(Silva ym., 2011) sain aikaan mallin, joka voisi olla pohjakäyttäjäkeskeisen kehityksen lisäämisestä ketterän kehityksen malliin:



Kuvio 4 Projektimalli käyttäjäkeskeiselle ketterälle kehitykselle

Ensimmäisessä vaiheessa täytyy saada tietoon järjestelmän vaatimukset. Jos näitä ei tutkita tarkemmin asiakkaan päädyssä, on suuri vaara saada vain ylimalkaiset vaatimukset, joissa ei selviä, miten loppukäyttäjä haluaisi työtään tehdä ohjelmistolla. "Tunne käyttäjä" -vaatimuksen täyttämiseksi täytyy asiakkaan päädyssä tehdä tutkimusta, jolla voidaan löytää vinkkejä itse loppukäyttäjiltä, miten heistä olisi järkevää ohjelmistoa käyttää. (Silva ym., 2011) Esimerkiksi, missä kohdassa tietyn informaatiotaulun olisi kaikkein järkevin olla ja mikä työkalu x olisi järkevää asettaa työkalun y läheisyyteen. Toisin sanoen kaivaa heiltä informaatiota, jota tavallisesti asiakkaan edustajalla, joka on ketterän kehityksen projektissa mukana, ei ole. Kerätyn informaation avulla voidaan johtaa käyttäjätarinat, jotka auttavat projektin seuraavassa vaiheessa. (Miller, 2005)

Järjestelmän suunnitteluvaiheessa tuotetaan skenaariot auttamaan ohjelmistoarkkitehteja saamaan paremman kuvan loppukäyttäjistä. Tutkimuksien pohjalta tämä tuo motivaatiota ohjelmoijille tuottaa käyttöliittymästä oikeanlaisen käytettävyyden kannalta. (Albisetti, 2010)

Kun itse ohjelmointi alkaa, täytyy jokaisen pyrähdyksen jälkeen suorittaa erillinen testauspyrähdys, jonka tarkoituksena on tarkistaa tuotetun ohjelmistoosan laatu. Tämä voi tapahtua samanaikaisesti, kun ohjelmointitiimi on suunnittelemassa ja toteuttamassa jo projektin seuraavaa pyrähdystä. Jos testauksessa ilmenee puutteita, voidaan sitä seuraava ohjelmointipyrähdys suunnitella korjaamaan löydetyt viat. (Ambler, 2008)

Seuraavaksi jää jäljelle enää tuotteen julkaisu tai luovutus asiakkaalle sisältäen teknisen avun käyttöönotossa. Jos käytättäjäkeskeinen ohjelmistotuotanto on onnistunut, tämä vaihe tuottaa asiakkaalle huomattavia säästöjä varjokustannuksien vähentymisenä. Löytämiini varjokustannuksiin kuuluu:

- työntekijöiden tehokkuuden väheneminen pidentyneiden tehtäväaikojen ja kanssatyöntekijöiden auttamisen takia
- koulutuksen tarve
- teknisen tuen tarve
- manuaalien tarve
- käyttäjien tekemät virheet ja niiden korjaus
- työtyytyväisyyden lasku ja siitä johtuva työntekijöiden vaihtuvuus

5.1 Jatkotutkimusaiheita

Aihetta olisi erittäin mielenkiintoista päästä tutkimaan tarkemmin jatkossa. Olisi mahdollista tutkia käytännön tasolla, miten käyttäjäkeskeinen järjestelmäkehitys auttaa lisäämään ohjelmistoyrityksen tuottamaa arvoa asiakkaille. Tämä voisi olla mahdollista erilaisilla kyselyillä, jotka toteutettaisiin ketterän kehityksen projektin lopputuotteen käyttäjille ja käyttäjäkeskeisen ketterän kehityksen lopputuotteen käyttäjille. Lisäksi heidän suoriutumistaan järjestelmän kanssa

voitaisiin tutkia ja mahdollisesti myös kustannuksia, mitä molemmat tuotteet tuovat asiakasyritykselle esimerkiksi tarvittavan koulutuksen ja käyttäjien tarvitsevan teknisen avun takia. Kun ohjelmistojen tuottajana olisi sama yritys, olisi todennäköisesti ohjelmakoodin laatu samanlaista ja lopputuotteet eroaisivat pelkästään ominaisuuksien suunnittelultaan. Saatu informaatio olisi arvokasta aiheen kannalta.

LÄHTEET

Ajzen,I. (1991). The Theory of Planned Behavior. Academic Press, Inc.

Ambler S. (Ed.). (2008). *Tailoring usability into agile software development projects*. Lontoo: Springer-Verlag.

Beck K. (Ed.). (2000). *Extreme programming explained* Addison Wesley.

Bias R. G, Mayhew D. J. (Ed.). (1994). *Cost-justifying usability* Academic Press.

Butler, A. (1996). Usability engineering turns 10. 1-3.

Chamberlain, S., Maiden. (2006). Towards a framework for integrating agile development and user-centered design., 9-10.

Constantine L., L. L. (2003). Usage-centered software engineering: An agile approach to integrating users, user interfaceses, and usability into software engineering practice., 746-747.

Ferreira J., Noble J., Biddle R. (2007). Agile development iterations and UI design.

Gould J., Lewis C., (1985). Designing for usability: Key principles and what designers think. http://delivery.acm.org/10.1145/10000/3170/p300gould.pdf?ip=130.234.244.162&acc=ACTIVE%20SERVICE&CFID=106258887&CFTOKEN=64594548&__acm__=1338470832_dcbcd37b2a47b7cda136ed7a5f13f172

ISO. (1999). *Human-centered design processes for interactive systems* (Technical report, ISO 13407

Kelander T. (2012) A Framework for the Integration of Usercentered and Agile Development. University of Jyväskylä

Karat. (1993). Usability in dollars and cents, IEEE software. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=210610

Mauro, C. L. (2002). *Professional usability testing and return on investment as it applies to user interface design for web-based products and sevices.* http://www.ergonomia.ca/doc/UsabTesting&ROI.pdf

Miller L. (2005). Case study of customer input for a successful product. pp. 225-235.

Naotake Hirasawa, Kiko Yamada-Kawai, Hideaki Kasai. (2010). Effectifeness of introducing human-centered design process.

Nielsen, J. (1993). Usability engineering. *Academic Press. Inc*

Nielsen, L. L. (2011). Usability requirements in agile development process. Technical University of Denmark.

Nigel Bevan. (2005). Cost benefits evidence and case studies. http://www.nigelbevan.com/papers/Cost_benefits_evidence.pdf

Nummiaho, A. (2006). User-centered design and extreme programming. Software Engineering Seminar. pp. 1-5.

Osama Sohaib, K. K. (2010). Integrating usability engineering and agile software development.

Pressman, R. S. (Ed.). (1992). *Software engineering: A practitioner's approach* McGraw Hill.

Seffah A., Gulliksen J., Desmarais M (Ed.). (2005). *Human-centered software engineering: Integrating usability in the software development lifecycle* Springer.

Singh, M. (2008). U-scrum: An agile methodology fot promoting usability. Agile 2008 Conference. pp. 1-4.

The Standish Group. (2002). *Chaos.* http://www4.informatik.tu-muenchen.de/lehre/vorlesungen/vse/WS2004/1995_Standish_Chaos.pdf

Tiago Silva da Silva, Angeka Martin, Frank Mauer, Milene Silveira. (2011). User centered design and agile methods: A systematic review.