Teijo Kelander

KÄYTETTÄVYYSSUUNNITTELUN ASIANTUNTIJA- JA TESTAUSMENETELMIEN SISÄLLYTTÄMINEN KETTERÄÄN OHJELMISTOKEHITYKSEEN

JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2012

TIIVISTELMÄ

Kelander, Teijo
Käytettävyyssuunnittelun asiantuntija- ja testausmenetelmien sisällyttäminen ketterään ohjelmistokehitykseen
Jyväskylä: Jyväskylän yliopisto, 2012, 39 s.
Tietojärjestelmätiede, kandidaatintutkielma
Ohjaaja(t): Kuparinen, Liisa

Tutkielmassa käsitellään käytettävyyssuunnittelun asiantuntija- ja testausmenetelmien sisällyttämistä ketterään ohjelmistokehitykseen. Aihetta on toistaiseksi tutkittu varsin vähän, mutta se on todettu tärkeäksi kehityssuunnaksi IT-alalla. Käytettävyyssuunnittelun ja ketterän ohjelmistokehityksen yhteneväiset tavoitteet on todettu yhdeksi argumentiksi integroida menetelmät keskenään.

Tutkielma käsittelee käytettävyyssuunnittelun yleisimmät asiantuntija- ja testausmenetelmät, ketterän ohjelmistokehityksen filosofian sekä kaksi käytetyintä menetelmää, jotka ovat Scrum ja eXtreme Programming. Näiden ilmiöiden pohjalta lähestytään käytettävyyssuunnittelun testaus- ja arviointimenetelmien sisällyttämistä ketterään ohjelmistokehitykseen.

Haasteita ovat menetelmäperheiden erilaiset tavoitteet, jotka ovat kumminkin toisiaan täydentäviä. Käyttäjäkeskeisessä suunnittelussa toteutetaan laajaa etukäteissuunnittelua, kun taas ketterässä ohjelmistokehityksessä esisuunnittelua tehdään tuskin ollenkaan. Erityinen ongelma integroimisen toteuttamiseksi on kahden menetelmäperheen erot resurssien allokoinnissa.

Kirjallisuuskatsauksen pohjalta todetaan, että kevyemmät asiantuntija- ja testausmenetelmät ovat avainasemassa kustannustehokkaan käytettävyyssuunnittelun toteuttamiseen ketterässä ohjelmistokehityksessä, mutta asiantuntija- ja testausmenetelmiä voidaan käyttää täysimittaisina myös ketterässä ohjelmistoprojektissa. Käytettävyysasiantuntijoiden liittäminen kehitystiimiin todetaan tärkeäksi sekä asiantuntija- ja testausmenetelmien joustavuus projekteissa. Muita ratkaisuja esitetään taulukoituna.

Katsauksen myötä ongelmaksi nousevat pilottiprojektien vähäisyys ja varsinainen ehdotettujen menetelmien luokittelun puute. Ratkaisuna näille tulisikin toteuttaa kattava luokittelu menetelmistä ja toteuttaa laajempia käytettävyyssuunnittelun menetelmiä hyödyntäviä ketteriä projekteja.

Asiasanat: Ketterä ohjelmistokehitys, käyttäjäkeskeinen suunnittelu, käytettävyyssuunnittelu, käytettävyys, Scrum, eXtreme programming

ABSTRACT

Kelander, Teijo
Inclusion of Usability engineering's inspection and testing methods to Agile Software Development
Jyväskylä: University of Jyväskylä, 2012, 39 p.
Information Systems, Bachelor's Thesis
Supervisor(s): Kuparinen, Liisa

This thesis is about how to integrate Usability engineering inspection and testing methods to Agile software development projects. Issue has so far been studied very little, but it is found to be important in developing the direction of the IT industry. Usability engineering and Agile software development have consistent objectives, which have been identified one of the causes integrating methods with each other.

The thesis discusses the most common usability engineering inspection and testing methods, agile software development philosophy, and the two most used methods that are Scrum and eXtreme Programming. On the basis of these phenomena usability evaluation and testing methods are approached for the inclusion of them in agile software development.

Method families are challenged by different objectives, which still are complementary. User-centered design carries out extensive pre-planning, while in the agile software development pre-planning exists hardly at all. A particular problem in inclusion is the way how two method family differences in resource allocation.

It is found on the basis of the literature review that the more agile inspection and testing methods is the key to cost-effective implementation of usability engineering to agile software development, but the inspection and testing methods may be used full-scale in Agile software project. In particular, making the usability specialists as members of Agile software development team is recognized very important as well as inspection and test methods flexibility in Agile projects. Other solutions are presented in tabular form.

Literature review shows that real problems regarding the subject are lack of pilot projecting and a lack of the classification of the proposed methods. The solution to these is to do a comprehensive classification of methods and carry out more extensive Agile projects, which utilize usability engineering methods.

Keywords: Agile software development, user-centered design, usability engineering, usability, Scrum, eXtreme Programming

KUVIOT

Kuvio 1 Ihmiskeskeisen ohjelmistosuunnittelun prosessimalli (Jokela ym., 2003) 11
Kuvio 2 Scrumin prosessimalli (Sutherland & Schwaber, 2007) 18
Kuvio 3 XP-projektin elinkaarimalli (Abrahamsson ym., 2002) 20

TAULUKOT

Taulukko 1 Tutkimuskysymykset..... 7
Taulukko 2 Käytettävyyttä arvioivien menetelmien piirteet..... 13
Taulukko 3 Kolmetoista parhainta käytäntöä eXtreme Programming:ssa (Leffingwell, 2007) 20
Taulukko 4 Ratkaisuja käytettävyyssuunnittelun asiantuntija- ja testausmenetelmien soveltamisesta ketterään ohjelmistokehitykseen..... 32
Taulukko 5 Tiivistetyt vastaukset tutkimuskysymyksiin..... 35

SISÄLLYS

TIIVISTELMÄ 2

ABSTRACT 3

KUVIOT 4

TAULUKOT 4

SISÄLLYS 5

1 JOHDANTO 6

2 KÄYTETTÄVYYSSUUNNITTELU JA SEN MENETELMÄT 8
2.1 Mitä on käytettävyys? 8
2.2 Käyttäjäkeskeinen suunnittelu 9
2.3 Käytettävyyssuunnittelun menetelmät 11
2.4 Käyttäjäkeskeisen suunnittelun hyödyt 14

3 KETTERÄ OHJELMISTOKEHITYS 15
3.1 Mihin ketterät menetelmät perustuvat? 15
3.2 Scrum 16
3.3 eXtreme Programming (XP) 18

4 KÄYTETTÄVYYSSUUNNITTELUN TESTAUS- JA ASIANTUNTIJAMENETELMIEN SISÄLLYTTÄMINEN KETTERIIN OHJELMISTOPROJEKTEIHIN 22
4.1 Tilastot 22
4.2 Haasteet ja ongelmat 23
4.3 Vastauksia haasteisiin ja ongelmiin 25
4.3.1 Käytettävyyden asettuminen kehitysprosessiin 25
4.3.2 Resurssit ja testauksen määrä 26
4.3.3 Tiimit ja käytettävyyden asiantuntijat 26
4.3.4 Käytettävyystestausmenetelmät ketterästi 27
4.3.5 Prototyyppitestaus 28
4.3.6 Testauksen ja arvioinnin ajoitus ketterästi 29
4.3.7 Muita erikoishuomioita 30
4.4 Piirteiden koostaminen 31

5 POHDINTA JA YHTEENVETO 34

LÄHTEET 36

1 Johdanto

Käytettävyys on yhä jokapäiväinen ongelma ohjelmistoissa, vaikka käytettävyyteen on kiinnitetty vuosien ajan huomiota ohjelmistokehityksessä. Kankeat ja epäjohdonmukaiset käyttöliittymät hidastavat ja vaikeuttavat esimerkiksi työtehtävien suorittamista vaadituissa aikarajoissa. Käyttäjäkeskeisyyteen tuotteiden kehityksessä keskittyvät ketterät menetelmät ovat jatkuvassa kasvussa IT-alalla. Huolimatta käyttäjälähtöisestä näkökulmasta, monet ketterillä menetelmillä kehitetyistä tuotteista eivät vastaa käytettävyyteen liittyviin ongelmiin (Hussain, Slany & Holzinger, 2009a).

Ketterien ohjelmistoprojektien lopputuotteiden käytettävyysongelmiin ovat ottaneet kantaa useat asiantuntijaorganisaatiot ja tutkijat. Ratkaisuiksi on esitetty useita uusia ketterien menetelmien sovellutuksia, joihin on sisällytetty käytettävyyssuunnittelun menetelmiä. Lähtökohtaisesti käytettävyyssuunnittelu on osittain samankaltainen prosessiltaan ja tavoitteiltaan kuin ketterät menetelmät. Ne molemmat perustuvat käyttäjäkeskeiseen näkökulmaan. Toinen merkittävä näkökulma sisällyttämisen edellytyksiin on periaate arvon tuottamisesta loppukäyttäjille. Kolmas yhteinen tekijä on jatkuvan testauksen periaate ja neljäs tekijä on iteratiivinen kehitysmalli (Hussain ym., 2009a; Hussain, Slany & Holzinger, 2009b; Fox, Sillito & Maurer, 2008; Silva, Martin, Maurer & Silveira, 2011).

Sohaibin ja Khanin (2010) mukaan ketterissä menetelmissä on huomioitu käytettävyyttä, mutta toisaalta käytettävyyden ja käyttäjäkeskeisen suunnittelun näkökulmasta ketterät menetelmät eivät luonnostaan tarjoa tarvittavaa apua kehitysprojekteihin. Lee, McCrickard ja Stevens (2009) toteavat sekä ketterän ohjelmistokehityksen että käytettävyyden painottavan sidosryhmien hyödyntämistä osana kehitystä. Erona voidaan nähdä se, että ketterät menetelmät painottavat asiakkaan kanssa tapahtuvaa yhteistyötä, muttei loppukäyttäjän merkitystä yhteistyössä.

Tutkijat näkevät, että ketterillä ja käyttäjäkeskeisillä lähestymistavoilla on potentiaalia toimia hyvin keskenään. Käytettävyyssuunnittelun asiantuntija- ja testausmenetelmien sisällyttämisellä ketterään ohjelmistokehitykseen voidaan saavuttaa lisäarvoa ja parempia tuloksia ohjelmistokehityksen tuotteen laadus-

sa ja käytettävyydessä (Sohaib & Khan, 2010; Hussain ym. 2009a). Käyttäjäkeskeisen suunnittelun menetelmistä on sisällytetty ketterään ohjelmistokehitykseen myös muita kuin asiantuntija- ja testausmenetelmiä. Esimerkiksi vaatimusmäärittelyn ja käyttäjätutkimuksen menetelmät muodostavat olennaisen osan menetelmien yhteensovittamisessa. Ketterät ohjelmistokehittäjät ovat itsekin heränneet huomaamaan käytettävyyden ja sen testaamisen sekä arvioinnin merkityksen ohjelmistojen kehityksessä (Hussain ym., 2009b).

Tutkielma perehtyy tarkemmin ilmiöihin käytettävyyssuunnittelun asiantuntija- ja testausmenetelmien sekä ketterien menetelmien taustalla. Lisäksi esitellään useita ehdotettuja tapoja sisällyttää käytettävyyden asiantuntija- ja testausmenetelmiä ketterään ohjelmistokehitykseen. Lopuksi esitellään yleispiirteet käytettävyyssuunnittelun asiantuntija- ja testausmenetelmien käytöstä ketterässä ohjelmistokehityksessä.

Tutkielmassa käydään läpi käyttäjäkeskeisen menetelmän määritelmiä, kehitysfilosofiaa, hyötyjä ja ongelmia. Tutkielmassa käytettävyyssuunnittelu rinnastetaan sekä käyttäjäkeskeiseen suunnitteluun, että ihmiskeskeiseen suunnitteluun. Käytettävyyssuunnittelun osalta nojataan asiantuntijoihin, tutkimusraportteihin sekä standardeihin. Lisäksi käsitellään käytettävyyssuunnittelun asiantuntija- ja testausmenetelmiä.

Ketterien menetelmien osuudessa käydään läpi niiden kehitysfilosofia, yleiset piirteet ja kaksi menetelmäperheen jäsentä yksityiskohtaisemmin. Menetelmät ovat valittu tilastollisesti (Hussain, Slany & Holzinger, 2009a) korkeimman käyttöasteen mukaan. Tässä tapauksessa ne ovat Scrum ja eXtreme Programming.

Varsinainen kirjallisuuskatsaus toteutetaan tarkastelemalla kehitettyjä ja ehdotettuja ratkaisuja, joissa on sovellettu käytettävyyden testaus- ja asiantuntijamenetelmiä ketteriin ohjelmistoprojekteihin. Katsauksen pohjalta vertaillaan ratkaisumalleja sekä niiden ongelmia ja mahdollisuuksia, joiden pohjalta koostetaan yhteiset piirteet käytettävyyssuunnittelun testaus- ja asiantuntijamenetelmien sisällyttämiseksi ketterään ohjelmistokehitykseen. Lyhyesti, kuinka käytettävyyssuunnittelun asiantuntija- ja testausmenetelmät tulisi sisällyttää osaksi ketterää ohjelmistokehitystä? Jatkotutkimusaiheet esitetään tutkielman yhteenvedossa. Taulukossa 1 on esiteltynä tutkimuskysymykset, jotka ovat jaoteltu ensisijaiseen ja toissijaisiin.

Taulukko 1 Tutkimuskysymykset

Kysymys	**Ensisijainen vai toissijainen**
Millaisia ratkaisuja käytettävyystestauksen sisällyttämiseksi ketterään ohjelmistokehitykseen on esitetty?	Ensisijainen
Mitä etuja käytettävyyssuunnittelun testaus- ja asiantuntijamenetelmien sisällyttämisestä ketteriin menetelmiin on?	Toissijainen
Mitä ongelmia liittyy käytettävyyssuunnittelun testaus- ja asiantuntijamenetelmien sisällyttämiseen ketterään ohjelmistokehitykseen?	Toissijainen

2 Käytettävyyssuunnittelu ja sen menetelmät

Tämä luku käsittelee yleisesti käytettävyyssuunnittelua. Aluksi käsitellään käytettävyyden määritelmiä ja sen jälkeen syvennytään tarkemmin käytettävyyssuunnitteluun, sen menetelmiin sekä standardoituun prosessimalliin. Näiden lisäksi käsitellään lyhyesti käytettävyyssuunnittelun osoitettuja hyötyjä.

2.1 Mitä on käytettävyys?

Eklundin ja Levingstonin (2008) mukaan käytettävyyden voi käsitteenä tulkita monella tapaa. Se on näkökulma, toimiala kuin myös menetelmä. Nielsen (1993) lähestyy ilmiötä laajemmin. Hän kategorisoi ohjelmiston käyttökelpoisuuden (engl. usefulness), käytettävyyden (engl. usability) ja hyödyllisyyden (engl. utility) kautta. Käyttökelpoisuus syntyy, kun ohjelmistolla voidaan saavuttaa haluttuja tavoitteita ja hyödyllisyys vastaa siitä, voidaanko ohjelmistolla tehdä sitä mitä on tarkoitus. Käytettävyyden Nielsen (1993) määrittelee viiden piirteen kautta, jotka ovat opittavuus, tehokkuus, muistettavuus, virheet ja miellyttävyys.

Jos käyttöliittymän opittavuus (engl. learnability) on kohdillaan, käyttäjän on tällöin helppo oppia ohjelman käyttö ja käyttöliittymä tuntuu intuitiiviselta. Tehokkuus (engl. efficiency) syntyy, kun käyttäjä on oppinut järjestelmän käytön ja pystyy olemaan varsin tehokas sen käytössä. Muistettavuus (engl. memorability) tarkoittaa järjestelmän helppoa muistamista niin, että järjestelmän käyttöön palaaminen on mahdollisimman vaivatonta eikä kaikkea tarvitse opetella uudestaan. Järjestelmässä tapahtuvien virheiden (engl. errors) määrä tulee olla mahdollisimman alhainen, jotta käyttäjät eivät tee virheitä tehtävien teon aikana ja jos tekevät, niistä palautuminen on oltava nopeaa. Järjestelmän miellyttävyys (engl. satisfaction) syntyy hyvästä käyttökokemuksesta. (Nielsen, 1993)

Käytettävyyttä ovat määrittäneet myös muut asiantuntijat. Bevanin, Kirakowskin ja Maisselan, määritelmän (1991) mukaan käytettävyys syntyy käyttäjän ja tuotteen vuorovaikutuksesta, mikä on mitattavissa käyttäjäsuoriutumisen,

tyytyväisyyden ja hyväksymisen kautta. Mikä tahansa muutos järjestelmässä, käyttäjässä, tehtävässä tai käyttöympäristössä voi muuttaa käytettävyyttä. Edellä esitetyt attribuutit eivät koske ainoastaan tuotteen ergonomiaa vaan käytettävyyteen kuuluu myös ohjelmistojen laatu, joka näkyy esimerkiksi helppokäyttöisyytenä.

ISO 9241-11 ja ISO 13407 ovat kaksi tärkeää käytettävyyden standardia. Ensimmäiseksi mainittu tarjoaa käytettävyyden määritelmän ja toinen ohjeita käytettävyyden suunnitteluksi (Jokela, Iivari, Matero & Karukka, 2003). Nielsenin (1993) määritelmä eroaa ISO 9241-11:sta. Standardi 9241-11 määrittelee käytettävyyden tilaksi, jossa tuotetta voidaan käyttää tiettyjen käyttäjien toimesta tiettyjen tavoitteiden saavuttamiseen niin tehokkuuden, hyötysuhteen kuin tyytyväisyyden osalta tietyssä käyttökontekstissa (Sohaib & Khan, 2010).

Tarkemmin tarkasteltuna ISO 9241-11 lähestyy käytettävyyttä kolmen kysymyksen kautta:

- Ketkä ovat käyttäjiä?
- Mitkä ovat käyttäjän tavoitteet?
- Mikä on tuotteen käyttöympäristö?

Käytettävyyden toteutumisen edellytykset ISO 9241-11:ssa ovat tuloksellisuus, tehokkuus ja käyttäjätyytyväisyys. Tuloksellisuus määrittyy käyttäjän tavoitteiden toteutumisen kautta. Tehokkuus vastaa kysymykseen siitä, kuinka paljon vaaditaan resursseja siihen, että käyttäjä pääsee tavoitteeseensa. Voidaankin sanoa, että mitä nopeammin käyttäjä pääsee tavoitteeseensa, sitä parempi on tehokkuus. Käyttäjätyytyväisyys toteutuu, kun käyttäjä kokee tuotteen miellyttävänä. (Jokela ym., 2003)

2.2 Käyttäjäkeskeinen suunnittelu

Ohjelmistojen ja järjestelmien käytettävyyden tehostamiseksi on ehdotettu käyttäjäkeskeisen kehityksen (engl. User-centred design, UCD) paradigmaa. Ihmiskeskeinen kehitys (engl. Human-centred design, HCD) ja käytettävyyssuunnittelu (engl. Usability engineering, UE) tarkoittavat käytännössä samaa asiaa (Jokela ym., 2003). Eroa selkeyttää se, että käytettävyyssuunnittelu tarkoittaa enemmän menetelmiä, joita käytetään käyttäjäkeskeisessä sekä ihmiskeskeisessä suunnittelussa (Silva ym., 2011).

Lyhyesti määriteltynä käyttäjäkeskeinen suunnittelu on vuorovaikutteisten järjestelmien suunnittelemisen ja kehittämisen lähestymistapa, joka keskittyy järjestelmien käytettävyyteen (Jokela ym., 2003). Läheisiä käsitteitä ovat myös käyttäjäkokemus (engl. User eXperience, UX) kuin myös vuorovaikutuksen suunnittelu (engl. Interaction Design). Käsitteitä käytetään ristiin ja rinnakkain erityisesti silloin kun puhutaan ketterästä ohjelmistokehityksestä ja käytettävyydestä (Silva ym., 2011).

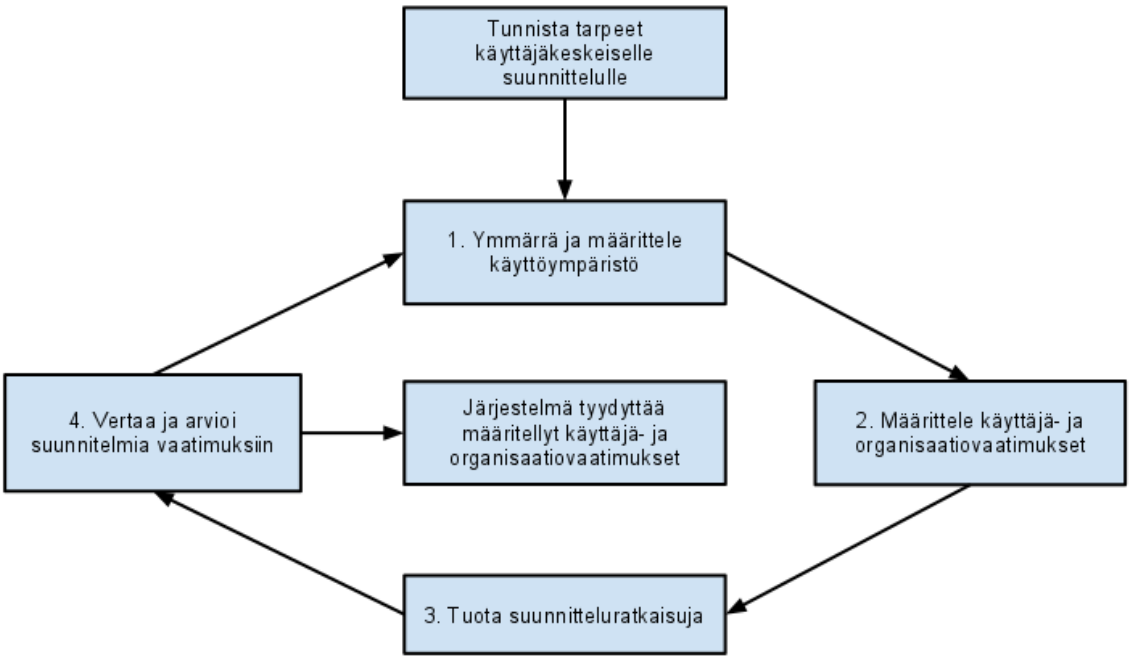
Näiden kaikkien käsitteiden juuret ovat ihmisen ja tietokoneen vuorovaikutuksessa (engl. Human-computer Interaction, HCI). HCI on käsitteiltään hyvin epäyhtenäistä ja jatkuvasti monet tutkimuksista käyttävät eri käsitteitä kuvaamaan samankaltaisia ilmiöitä varsinkin silloin kun on kyse ketterien menetelmien ja käytettävyyssuunnittelun integroimisesta (Silva ym., 2011). Edellä esitetyt käsitteet ovat tässä tutkielmassa tiivistetty käyttäjäkeskeisen suunnittelun alle, vaikkakin pääpaino on käytettävyyssuunnittelun menetelmissä.

Käyttäjäkeskeisen suunnittelun ovat määrittäneet myös Vredenburg, Mao, Smith ja Carey (2002). Heidän mukaansa käyttäjäkeskeinen suunnittelu on monialainen lähestymistapa, joka on iteratiivista suunnittelua ja arviointia, missä loppukäyttäjät osallistuvat aktiivisesti käyttäjä- ja tehtävävaatimuksien luomiseen. Sen sijaan standardi ISO 13407 eli ihmiskeskeiset suunnitteluprosessit vuorovaikutteisille järjestelmille tarjoaa opastusta käyttäjäkeskeiseen suunnitteluun. Standardi on tärkeä osa käyttäjäkeskeiseen suunnitteluun liittyvää kirjallisuutta. Abstraktiotasolta lähestyvä standardi on kansainvälisten asiantuntijoiden ja tutkijoiden konsensus, joka käyttää ISO 9241-11:n määritelmää käytettävyydestä (Jokela ym., 2003).

ISO 13407 kuvailee käyttäjäkeskeisen suunnittelun neljästä eri näkökulmasta, jotka ovat perustelut, suunnittelu, periaatteet ja toiminnot. Perusteluja ovat muun muassa koulutuksen tarpeen laskeminen, parantunut käyttäjätyytyväisyys ja työntuottavuuden kasvu. Ohjelmistokehityksen kaikki vaiheet voivat perustua näihin neljään periaatteeseen:

- Käyttäjä osallistuu aktiivisesti ja kehittäjillä on selkeä ymmärrys sekä käyttäjän, että toimintojen vaatimuksista.
- Tehtävät on jaettu sopivasti käyttäjien ja teknologian välillä.
- Suunnitteluratkaisut tuotetaan iteraatioissa.
- Suunnittelun tulee olla monialaista.

Suunnittelu tarjoaa ohjeistusta käyttäjäkeskeisen suunnittelun sovittamisesta osaksi järjestelmän kehitysprosessiin. Tällöin painotetaan projektisuunnitelmien resurssien merkitystä. Iteraatioiden, käyttäjäpalautteen resurssien ohella on myös painotettu tiimityöskentelyn merkitystä. Standardin keskeinen osa on käyttäjäkeskeisen suunnittelun toimintojen kuvaus (Jokela ym., 2003). Toiminnot ovat seuraavan kuvion mukaiset:



Kuvio 1 Ihmiskeskeisen ohjelmistosuunnittelun prosessimalli (Jokela ym., 2003)

1. Opitaan tuntemaan käyttäjä, käyttöympäristö ja tehtävät, jotka tuotteella tulisi ratkaista.

2. Luodaan kriteerit pohjautuen suoritettaviin tehtäviin, minkä perusteella voidaan arvioida kehitetyn tuotteen käytettävyyttä. Esimerkiksi mikä on keskivertoisen käyttäjän suoriutumisaika tuotteella annetusta tehtävästä. Määritetään suunnitteluohjeet ja -rajoitteet.

3. Sisällytetään ihmisen ja tietokoneen vuorovaikutuksen tietämystä, kuten visuaalista suunnittelua, vuorovaikutuksen suunnittelua tai käytettävyyttä suunnitteluratkaisuihin.

4. Suunnitteluratkaisujen käytettävyyttä arvioidaan luotujen kriteerien pohjalta esimerkiksi tehtävien ratkaisunopeuden pohjalta.

2.3 Käytettävyyssuunnittelun menetelmät

Holzinger (2005) jaottelee käytettävyyssuunnittelun ohjelmistotuotannossa käytettävät tekniikat kahteen kategoriaan: asiantuntija- (engl. Inspection methods) ja testausmenetelmiin (engl. Test methods). Asiantuntijamenetelmiksi luokitellaan heuristinen arviointi, kognitiivinen läpikäynti ja toimintoanalyysi. Testausmenetelmiä ovat sen sijaan ääneen ajattelu, käyttäjätarkkailut ja kyselyt.

Heuristinen arviointi (engl. Heuristic evaluation, HE) on järjestelmällinen käytettävyyden tarkastelu, joka perustuu asiantuntija-arviointiin (Nielsen, 1993). Heuristisen arvioinnin lähtökohta on yhden yksittäisen asiantuntijan toteuttama arviointi, jossa asiantuntija käy läpi käyttöliittymän eri elementit, dialogit tai muut ominaisuudet nojaten käytettävyyden vakiintuneisiin periaatteisiin.

Arvioinnille on nykyään olemassa useita erilaisia sovellutuksia, kuten pariarviointi. Etuja menetelmässä ovat intuitiivisuus, käytettävyysongelmien varhainen löytyminen, tehokas pienten ja suurten ongelmien tunnistaminen, nopeus ja käytettävyyden parantaminen prosessin läpi. Haittoja ovat loppukäyttäjien huomioimisen puute, epävarmuus kokonaissuunnittelun tarkastelun kattavuudesta ja liian yksipuoleiset arvioinnit (Holzinger, 2005).

Kognitiivinen läpikäynti (engl. Cognitive walkthrough, CW) on tehtäväorientoitunut menetelmä, jolla tutkitaan järjestelmien toimintoja. Kognitiivisessa läpikäynnissä käyttöliittymäsuunnittelija hahmottaa käyttäjän toimintoja vaiheittain tietyn tehtävän suorittamisessa (Nielsen, 1993). Asiantuntija on voinut ensiksi tutkia todellisia työnkulkuja tai on suunnitellut vaihtoehtoisia työnkulkuja ja rakentanut näiden pohjalta testitapaukset ja testaa sitten käyttöliittymää. Menetelmä korostaa kognitiivisia toimintoja, kuten opittavuutta. Menetelmästä on myös useita variaatioita, missä voidaan hyödyntää myös loppukäyttäjiä, ohjelmistokehittäjiä sekä käytettävyysasiantuntijoita. Menetelmän etuja ovat itsenäisyys loppukäyttäjistä, mahdollisuus jäljittää käyttäjän tavoitteet ja oletukset sekä käyttäjänäkökulman esiintuominen. Ongelmiksi voivat muodostua väärät tehtävävalinnat, käyttäjien osallistumattomuus sekä matalan tason yksityiskohtiin keskittyminen (Holzinger, 2005).

Toimintoanalyysissa (engl. Activity analysis, AA) käytettävyyttä mitataan tehtävien kautta. Testauksessa mitataan kaikki painallukset ja tehtävän suoritukseen kulunut aika (Nielsen, 1993). Tämä selvittää kuinka kauan vie tietyn tehtävän suorittaminen ja paljonko se vaatii painalluksia. Menetelmä tuottaa tarkkoja ennustuksia tehtävien suorittamisen ajasta ja antaa syvemmän ymmärryksen loppukäyttäjien tavasta toimia. Ongelmia ovat resurssivaatimukset, sillä toimintoanalyysi vie paljon aikaa ja vaatii korkeaa ammattitaitoa (Holzinger, 2005).

Ääneen ajattelussa (engl. Think aloud, THA) käyttäjä suorittaa tehtäviä ohjelmistolla tai prototyypillä samalla kertoen ääneen, mitä on tekemässä tai ajattelee (Nielsen, 1993). Ääneen ajattelu on yksi tehokkaimmista käytettävyyssuunnittelun menetelmistä. Menetelmä mahdollistaa virhetilanteiden löytämisen ja osoittaa puutteelliset ohjeistukset. Asiantuntija seuraa ja kirjaa ylös testaajan kommentit, mutta ei puutu tehtävien suorittamiseen tai pyri vaikuttamaan testaajan toimintaan. Äänen ajattelu-menetelmästä on useita etuja, sillä se esimerkiksi selvittää syitä sille miksi käyttäjät tekevät jotain. Se antaa myös suhteellisen tarkkoja arvioita järjestelmän käyttötavoista, tarjoaa paljon tietoa joka voidaan kerätä suhteellisen vähällä määrällä käyttäjiä, tarjoaa usein täsmällisiä kommentteja käyttäjiltä eri toiminnoista ja ongelmista. Loppukäyttäjien ohjeistaminen testitilanteeseen vaatii paljon aikaa, mikä on yksi iso ongelma menetelmän käytössä sekä osallistuvien loppukäyttäjien saattaa olla vaikeaa tulkita joskus itseään (Holzinger, 2005).

Käyttäjätarkkailu (engl. Field observation, FO) on yksinkertaisempia käytettävyyden asiantuntijamenetelmiä. Tarkkailussa seurataan normaalia työtilannetta pyrkien olemaan häiritsemättä käyttäjän työtehtävien suorittamista (Nielsen, 1993). Muistiinpanojen tekeminen tai videointi on menetelmän toteut-

tamisen vaihtoehtoja. Tarkkailu voidaan tehdä myös sähköisesti, jolloin kerätään käyttödataa. Tilastointi voi sisältää esimerkiksi painallukset, virheilmoitukset tai käytetyt ominaisuudet. Menetelmä vaatii useita käyttäjiä sekä paljon aikaa datan analysointiin (Holzinger, 2005).

Kyselyt (engl. Questionnaires, Q) ovat monipuolinen menetelmä, sillä sitä voidaan käyttää useiden eri käytettävyyden piirteiden arviointiin. Menetelmällä voidaan selvittää kuinka loppukäyttäjät käyttävät järjestelmää ja mitkä asiat he kokevat hankaliksi käyttöliittymässä (Nielsen, 1993). Kyselyn etuja ovat niin tilastoinnin helppous kuin miellyttävyyden mittaus, mutta menetelmän haitoiksi voidaan lukea aikavaativuus, sillä kyselyitä pitää suunnitella tarkkaan etukäteen. Muita ongelmia voi olla matala todenmukaisuus vastauksissa ja matalampi ongelmakohtien esiintuominen kuin muissa menetelmissä. Erityinen ongelma menetelmän suhteen on se, että usein ihmiset käyttäytyvät toisin kuin mitä he väittävät (Holzinger, 2005).

Asiantuntijamenetelmät yksistään eivät ole tae hyvästä käytettävyydestä lopputuotteessa, vaan tukena kehityksessä tarvitaan myös testausmenetelmiä. Edellä esitettyjä menetelmiä tulisi käyttää ristiin niin, että toinen olisi asiantuntijamenetelmä ja toinen testausmenetelmä. Esimerkiksi käyttöliittymälle voidaan tehdä heuristinen arviointi, joka keskittyy muuhun kuin tehtäviin ja suoritettavia tehtäviä voitaisiin tehdä ja arvioida esimerkiksi toimintoanalyysissa. Hyvä käytettävyys saavutetaan oikealla ymmärtämisellä käyttäjästä, hänen kyvyistään, tavoistaan ja tehtävistään sekä hyödyntämällä käyttäjää aivan kehityksen alussa ja läpi kehityksen eri iteraatioissa (Holzinger, 2005).

Taulukko 2 kuvaa tiivistetysti käytettävyysarvioinnin menetelmien piirteitä (Holzinger, 2005).

Taulukko 2 Käytettävyyttä arvioivien menetelmien piirteet

	Asiantuntijamenetelmät			Testausmenetelmät		
	Heuristinen arviointi	Kognitiivinen läpikäynti	Toimintoanalyysi	Ääneen ajattelu	Käyttäjätarkkailu	Kyselyt
Testauksen vaihe	Kaikki	Kaikki	Suunnittelu	Suunnittelu	Lopputestaus	Kaikki
Kesto	Nopea	Kohtalainen	Pitkä	Pitkä	Kohtalainen	Nopea
Tarvittavat käyttäjät	0	0	0	3+	20+	30+
Vaadittavat asiantuntijat	3+	3+	1-2	1	1+	1
Vaaditut työvälineet	Vähän	Vähän	Vähän	Paljon	Kohtalainen	Vähän
Vaadittu asiantuntijuus	Kohtalainen	Korkea	Korkea	Kohtalainen	Korkea	Matala
Testaajan häiritsevyys	Ei	Ei	Ei	Kyllä	Kyllä	Ei

Nielsen (1993) on ehdottanut käytettävyyden arviointimenetelmiksi myös niin kutsuttuja kevennettyjä käytettävyysarviointimenetelmiä (engl. discount usability engineering methods). Kevyemmät käytettävyysarviointimenetelmät

ovat nimensä mukaisesti helpompia ja nopeampia toteuttaa ja näin ollen myös halvempia. Nämä kevyemmät menetelmät perustuvat usein käyttäjätarkkailuun, skenaarioihin, ääneen ajatteluun ja heuristiseen arviointiin (Nielsen, 1993). Kevyet käytettävyysarviointimenetelmät usein liitetään Nielsenin ehdottamiin menetelmiin, mutta monet muut asiantuntijat ovat esittäneet muitakin menetelmiä samassa hengessä. Kevennetyt menetelmät voivat olla epämuodollisempia, vähäisemmällä käyttäjämäärällä tehtyjä, vähäisempään suunnitteluun perustuvia, valmiita koeasetelmia ja -kysymyksiä, vähäisempiä dokumentoinniltaan tai pienempiä havainnointiaineistoltaan (Kane, 2003).

2.4 Käyttäjäkeskeisen suunnittelun hyödyt

Käyttäjäkeskeinen suunnittelu projektin toteutustapana tarjoaa useita etuja perinteisiin ohjelmistosuunnittelumenetelmiin verrattuna (Vredenburg ym., 2002). Monet eduista on huomioitu 1980-luvulta lähtien eri asiantuntijoiden toteamana. Käyttäjäkeskeinen suunnittelu tarjoaa parhaimmillaan niin kustannushyötyjä kuin myös sosiaalisia etuja (Bevan, 1999). Kustannushyödyt ovat useimmiten tärkein tekijä käyttäjäkeskeisen suunnittelun käyttöönotossa organisaatiossa (Vredenburg ym., 2002).

Määrällisesti suuri vuorovaikutus käyttäjien kanssa ei ole todellinen hyödyn lähde käyttäjäkeskeisessä suunnittelussa, vaan iteratiivinen ratkaisujen arviointi käyttäjien tekemänä mahdollistavat parhaimmat suunnitteluratkaisut projekteissa (Lai, Honda & Yang, 2010). Käyttäjäkeskeistä suunnittelua käyttäneet projektit ovat yleisimmin johtaneet parempaan käytettävyyteen lopputuotteessa, vaikka taustalla olisikin ollut eroja käytetyissä menetelmissä (Vredenburg ym., 2002).

2000-luvulla on kehittynyt monia käytettävyystestauksen menetelmiä, joita kutsutaan usein Usability 2.0 – nimellä tarkoittaen, että menetelmät pyrkivät olemaan helposti lähestyttäviä, käytännöllisiä, edullisia sekä tuovat todellista hyötyä yrityksen liiketoimintastrategioihin (Eklund & Levingston, 2008).

3 Ketterä ohjelmistokehitys

Tämä luku käsittelee yleisesti ketterää ohjelmistokehitystä lähtien liikkeelle ketterien menetelmien historiasta ja filosofiasta. Luvun loppupuolella käsitellään syvällisemmin kaksi ketterää menetelmää Scrum ja eXtreme programming.

3.1 Mihin ketterät menetelmät perustuvat?

Ketterä ohjelmistokehitys terminä syntyi vuonna 2001, kun ryhmä alan asiantuntijoita järjesti tapaamisen pohtiakseen ohjelmistokehityksen tulevia suuntauksia. He huomasivat käyttämiensä menetelmien sisältävän paljon yhteisiä piirteitä, joten he päättivät nimetä nämä prosessit ketteriksi (Cockburn, 2001).

Helmikuussa 2001 eri menetelmien edustajat päättivät muodostaa Agile Software Development Alliance - organisaation tuodakseen paremmin esiin kehitysfilosofiaansa. Järjestäytymisen seurauksena syntyi ketterän kehityksen manifesti (Agile Manifesto), johon sisältyy neljä arvoa, joiden pohjalta ketterät kehittäjät toimivat (Agile Alliance, 2001a):

> Yksilö on tärkeämpi kuin prosessit. Toimiva ohjelmisto on tärkeämpi kuin kaiken kattava dokumentaatio. Asiakasyhteistyö on tärkeämpää kuin sopimusneuvottelut. Reagointi muutokseen on tärkeämpää kuin suunnitelmaan juuttuminen.

Vaikka manifestin mukaisesti lauseissa viimeiseksi mainitut seikat ovat tärkeitä, pidetään ensiksi mainittuja arvoja eli yksilöä, toimivaa ohjelmistoa, asiakasyhteistyötä ja muutokseen reagointia tärkeämpinä. Prosesseja, dokumentaatiota, sopimusneuvotteluja ja suunnitelmia ei tule silti tyystin unohtaa, vaan nekin on huomioitava mahdollisuuksien mukaan.

Ketterän ohjelmistokehityksen manifestin taustalla vaikuttaa joukko ydinperiaatteita (Agile Alliance, 2001b), jotka ovat yleisiä kaikille ketterille menetelmille:

- Tärkein tavoitteemme on tyydyttää asiakas toimittamalla tämän tarpeet täyttäviä versioita ohjelmistosta aikaisessa vaiheessa ja säännöllisesti.
- Otamme vastaan muuttuvat vaatimukset myös kehityksen myöhäisessä vaiheessa. Ketterät menetelmät hyödyntävät muutosta asiakkaan kilpailukyvyn edistämiseksi.
- Toimitamme versioita toimivasta ohjelmistosta säännöllisesti, parin viikon tai kuukauden välein, ja suosimme lyhyempää aikaväliä.
- Liiketoiminnan edustajien ja ohjelmistokehittäjien tulee työskennellä yhdessä päivittäin koko projektin ajan.
- Rakennamme projektit motivoituneiden yksilöiden ympärille. Annamme heille puitteet ja tuen, jonka he tarvitsevat ja luotamme siihen, että he saavat työn tehtyä.
- Tehokkain ja toimivin tapa tiedon välittämiseksi kehitystiimille ja tiimin jäsenten kesken on kasvokkain käytävä keskustelu.
- Toimiva ohjelmisto on edistymisen ensisijainen mittari.
- Ketterät menetelmät kannustavat kestävään toimintatapaan. Hankkeen omistajien, kehittäjien ja ohjelmiston käyttäjien tulisi pystyä ylläpitämään työtahtinsa hamaan tulevaisuuteen.
- Teknisen laadun ja ohjelmiston hyvän rakenteen jatkuva huomiointi edesauttaa ketteryyttä.
- Yksinkertaisuus - tekemättä jätettävän työn maksimointi - on oleellista.
- Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itseohjautuvissa tiimeissä.
- Tiimi tarkastelee säännöllisesti, kuinka parantaa tehokkuuttaan, ja mukauttaa toimintaansa sen mukaisesti.

Ketterät menetelmät voidaan määritellä iteratiivisiksi, inkrementaalisiksi, itseohjautuviksi ja emergenteiksi (Larman, 2001). Ketterien menetelmien etuja ovat yksinkertaisuus, vähentyvät yleiskustannukset, lyhyet kehityssyklit ja vakaammat ohjelmistot (Constantine, 2002). Joka tapauksessa, edellä esitetyt arvot ja periaatteet ovat vain abstraktioita. Niiden hyödyntäminen sellaisenaan ei ole ketterien menetelmien käyttöä. Seuraava kappale esittelee tarkemmin Scrum-menetelmää, joka nojaa edellä esitettyihin arvoihin ja periaatteisiin.

3.2 Scrum

Ambler (2008) määrittelee Scrumin projektin ja vaatimuksenhallinnan työkaluksi, jota sovelletaan usein rinnakkain jonkin toisen ketterän menetelmän kanssa. Scrum on yksinkertainen työkalu, jota voidaan käyttää tiimien organisointiin tehokkaamman tuottavuuden saavuttamiseksi. Scrumin perusta on empiirisessä prosessinhallintateoriassa. Lähestymistapa ennustettavuuden optimoimiseen ja riskien kontrolloimiseen on iteratiivis-inkrementaalinen, mikä tarkoittaa projektin koostumista toistuvista vaiheista ja jokaisen vaiheen loppu-

tuloksen lisäävästä vaikutuksesta tuotteeseen (Schwaber, 2007). Iteratiivisia kehitysvaiheita Scrumissa kutsutaan sprinteiksi, jotka noudattavat aina samanlaista prosessia (Sutherland & Schwaber, 2007).

Scrumin painopiste on projektin johtamisessa ja muutosten hallinnassa, ei niinkään toteutuksessa, toisin kuin myöhemmin esiteltävällä XP:llä. Ketteränä menetelmänä Scrum ei yritä vastustaa muutosta, vaan yrittää tarjota tapoja tehokkaaseen toimintaan muuttuvassa ympäristössä. Scrumin iteratiivisen ja inkrementaalisen selkärangan muodostaa sen sisältämät kolme roolia: tuoteomistaja, scrummaster ja kehitystiimi. Kaikki hallinnolliset vastuut on jaettu näiden kolmen roolin kesken (Sutherland & Schwaber, 2007).

Tuoteomistaja on päätösvaltaisin henkilö ohjelmiston kehityksen kannalta. Omistaja voi olla niin asiakkaan edustaja kuin tuotepäällikkö. Tuoteympäristön erityinen tunteminen on tärkeä tuoteomistajan kannalta. Tärkein vastuu tuoteomistajalla on huolehtia vaatimuksista (Sutherland & Schwaber, 2007).

Kehitystiimin vastuu on toteuttaa sovitut tehtävät ja toiminnallisuudet iteraation aikana. Tiimin jäsenet ovat moniosaajia tarkoittaen että kaikki voivat tehdä kaikkea aina toteutuksesta dokumentointiin. Kehitystiimi toimii itsenäisesti tehden omia päätöksiä (Sutherland & Schwaber, 2007).

Scrummaster on usein yksi tiimin jäsenistä, mutta myös joskus hän voi olla ulkopuolinen henkilö. Hänen ei tule olla tiimin jäsenten esimies, sillä hän varmistaa että projektissa noudatetaan Scrum-prosessimallin käytäntöjä ja ohjeita. Lisäksi Scrummasterin vastuulla on tiimin työrauha iteraation aikana. Yksinkertaistettuna Scrummaster on siis eräänlainen tiimin valmentaja (Sutherland & Schwaber, 2007).

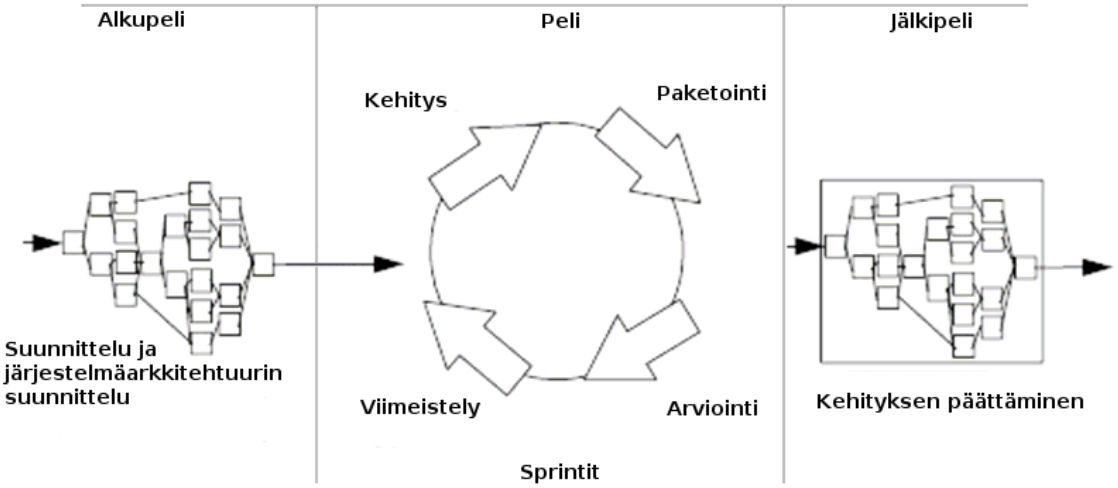
Roolien lisäksi yksi tärkeä elementti kehitysprojektissa on johdon edustajat. Tarvittaessa johdon edustajat toimivat Scrum-tiimin tukena ja apuna. Edustajat enemmänkin johdattelevat kohti ratkaisua kuin päättävät asioista itsenäisesti tiimin ylitse (Sutherland & Schwaber, 2007).

Scrumissa dokumentaatio koostuu pääasiallisesti tuotteen kehitysjonosta, sprintin tehtävälistasta, sprintin edistymiskäyrästä ja julkaisun edistymiskäyrästä. Tuotteen kehitysjono on lista kaikista tarpeellisista asioista, joita tuote voisi tarvita ja se on priorisoitu tarpeiden mukaan. Sprintin tehtävälista sisältää listan suoritettavista tehtävistä, jotta yhden sprintin aikana saadaan julkaisukelpoinen tuoteparannus. Julkaisun edistymiskäyrä mittaa tuotteen jäljellä olevaa kehitysjonoa suhteessa julkaisusuunnitelman aikaan. Sprintin edistymiskäyrä esittää jäljellä olevia tehtäviä suhteessa sprintin aikaan.

Edellä esitettyjen asioiden pohjalta Scrumin ominaiset piirteet voidaan tiivistää seuraavasti:

- Scrum on kaaoksenomainen, itseohjautuva, inkrementaalinen ja iteratiivinen prosessi.
- Scrum on muutokseen vastaava.
- Kehitystiimi on Scrummasterin valmentama.
- Scrumissa asiakas on läsnä.
- Scrumtiimi on itseohjautuva.

Scrum jakaa prosessin kolmeen vaiheeseen, jotka ovat alkupeli (engl. Pregame), peli (engl. Development) ja jälkipeli (engl. Post-game). Alkupeli koostuu suunnittelusta ja järjestelmäarkkitehtuurin suunnittelusta. Pelivaihe sisältää sprintit, joihin kuuluu kehitys, paketointi, arviointi ja viimeistely. Jälkipelivaihe sisältää kehityksen päättämisen (Sutherland & Schwaber 2007). Lisäksi Scrum käyttää aikarajoja säännöllisyyden luomiseen.



Kuvio 2 Scrumin prosessimalli (Sutherland & Schwaber, 2007)

Alkupelivaiheessa suunnittelu käynnistyy ensimmäisen julkaisun vaatimusten sijoittamisena tehtävälistaan. Tämän lisäksi laaditaan kustannus- ja aikatauluennuste. Uuden järjestelmän ollessa kyseessä selvitetään tarkemmin niin standardit, säännöt, kuin käytettävä teknologia. Lisäksi analysoidaan resurssien käyttöä ja arkkitehtuuria. Alkupelivaiheessa tulee myös miettiä projektitiimin kokoonpano ja tarpeet (Sutherland & Schwaber, 2007).

Pelivaihe on Scrumin iteratiivinen vaihe, jossa tapahtuu sprintit. Sprintit koostuvat neljästä osasta: Kehitysvaiheesta, paketointivaiheesta, arviointivaiheesta ja viimeistelyvaiheesta. Jokainen sprintti alkaa asiakkaan ja Scrum-tiimin palaverilla, jossa mietitään yhdessä alkavan sprintin tavoite. Sovitun tavoitteen pohjalta valitaan tuotteen tehtävälistasta tavoitteen täyttymisen mahdollistavat tehtävät. Sprintti on usein kestoltaan viikosta neljään viikkoon (Sutherland & Schwaber, 2007).

Jälkipelivaihe lopettaa kehityksen, silloin kuin kehitystiimi kokee, että on aika tehdä uusi julkaisu. Jälkipelivaiheessa suoritetaan integrointi, järjestelmän testaus, dokumentaatio, ennakkojulkaisu ja sen testaus sekä lopullinen julkaisu (Sutherland & Schwaber, 2007).

3.3 eXtreme Programming (XP)

Extreme Programming tai tuttavallisemmin XP on kevyt, tehokas, pieniriskinen, joustava, ennustettava, tieteellinen ja hauska tapa kehittää ohjelmistoja. Se pai-

nottaa lyhyitä kehityskiertoja sekä inkrementaalista suunnittelua (Beck & Anders, 2004). Menetelmä kehitettiin keskisuurille tiimeille epämääräisten ja muuttuvien tilanteiden sekä vaatimusten työkaluksi (Sohaib & Khan, 2011).

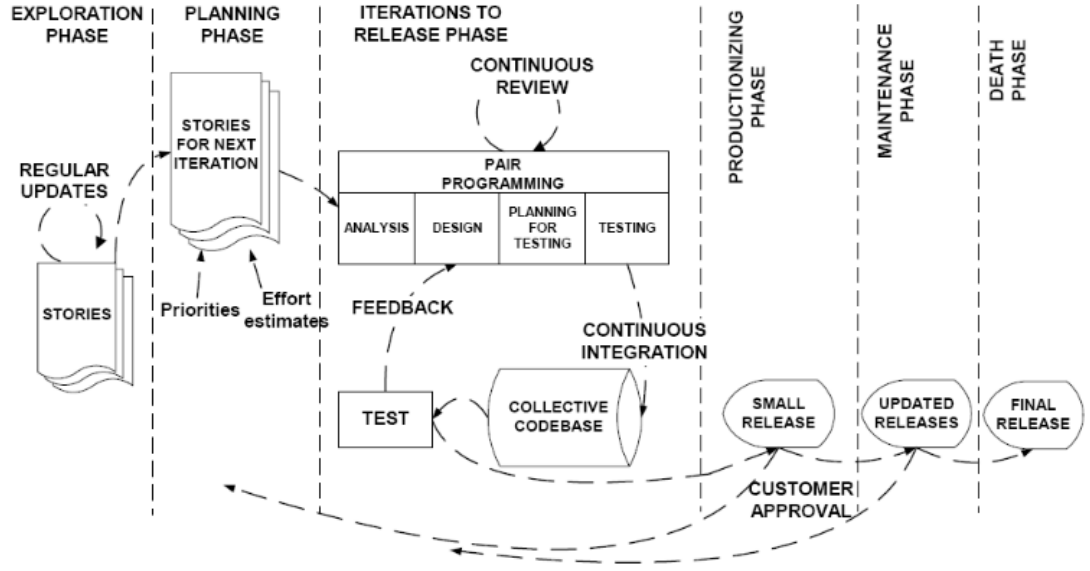
XP:n alkuperäiset neljä ydinarvoa ovat viestintä, yksinkertaisuus, palaute ja rohkeus. Näiden lisäksi listaan on myöhemmin lisätty kunnioitus (Leffingwell, 2007). Ydinarvoista voidaan johtaa XP:n kehittämisfilosofia, joka koostuu niin tehokkaasta kommunikaatiosta tiimin sisällä, vain välttämättömästä koodaamisesta, jatkuvasta sekä välittömästä palautteesta, rohkeudesta kirjoittaa testit ennen koodausta, yksinkertaisuuteen pyrkimisestä sekä kehitystiimin sisäisestä kunnioituksesta (Beck & Anders, 2004).

Viestintä (engl. communication) on ensisijainen arvo XP:ssä. Projektit kärsivät usein vajavaisesta viestinnästä esimerkiksi tilanteessa jossa ohjelmoija ei ilmoita eteenpäin jostain merkittävästä muutoksesta koodissa tai sen rakenteessa. XP tavoitteena on säilyttää viestinnänvirta projekteissa hyödyntämällä monia käytäntöjä (Beck & Anders, 2004).

Palautetta (engl. feedback) tarvitaan asiakkaalta, jotta ymmärretään asiakkaan tyytyväisyys tuotteesta. Palautteella ei ole XP-projekteissa viikkojen viivettä, vaan usein se tulee tunneissa tai päivissä. Yksinkertaisuutta (engl. simplicity) ei ole helppo saavuttaa vaan se on todellista taidetta. XP:ssä tulisikin painottaa yksinkertaisimman asian ohjelmointi ensimmäiseksi, sillä päälle rakennettavat tasot ja refaktorointi lopulta johtavat monimutkaisuuteen. Rohkeuden (engl. courage) tulee näkyä eXtreme Programming:ssa kaikkialla. Se on testien muodostamista ennen valmista koodia, yksinkertaisimman ratkaisun valitsemista, aikaista toimittamista ja välittömän palautteen vastaanottamista sekä rohkeutta refaktoroida koodia, joka ei toteuta yksinkertaisuustestiä. Kunnioitus (engl. respect) lähtee kehitystiimistä, joka rakentuu keskinäisen luottamuksen varaan. Kukaan tiimin jäsenistä ei ole toista arvokkaampi tai tärkeämpi. Sen tulee näkyä yhteistyössä ja viestinnässä (Leffingwell, 2007).

XP peruspiirteitä toiminnallisesta näkökulmasta tarkastellen ovat lyhyet iteraatiot pienillä julkaisuilla ja nopealla palautteella, asiakasläheisyys, jatkuva viestiminen, jatkuva integraatio ja testaaminen, koodin yhteinen omistajuus, pariohjelmointi sekä refaktorointi (Abrahamsson, Salo, Ronkainen & Warsta, 2002). Myös jatkuva asiakkaan läsnäolo, suunnittelupeli, 40 tuntiin rajoittuva työviikko, avoin työtila ja metaforat kuuluvat XP:n peruspiirteisiin (Leffingwell, 2007).

XP kehitysprosessi on jaettu iteraatioihin. Jokainen iteraatio kestää 1-4 viikkoa ja ne tähtäävät aina tuottamaan toimivan ohjelmiston uusien lisätoimintojen kanssa. XP:n prosessimalli koostuu viidestä iteratiivisesta osasta. Osia ovat tutkimus (engl. Exploration), suunnittelu (engl. Planning), julkaisuiteraatio (engl. Iterations to Release), tuotteistaminen (engl. Productionizing) ja loppujulkaisu, ylläpito sekä lopetus (engl. Maintaince and Death). Kaavassa 3 on esiteltynä prosessin eteneminen.



Kuvio 3 XP-projektin elinkaarimalli (Abrahamsson ym., 2002)

Vaikka arvot ja periaatteet luovat perustan XP:n filosofiasta, ne eivät kerro mitä tulisi tehdä tavoitteiden saavuttamiseksi. Peruskäytäntöjä XP:ssä on kaksitoista kappaletta (Beck & Anders, 2004). Sen sijaan Leffingwell (2007) on listannut kolmetoista mielestään parasta XP:n käytäntöä:

Taulukko 3 Kolmetoista parhainta käytäntöä eXtreme Programming:ssa (Leffingwell, 2007)

Nro	Käytäntö	Englanniksi	Kuvaus
1.	Yhdessä istuminen	Sit together	Yhdessä istuminen tarkoittaa työskentelyä samassa ja avoimessa tilassa.
2.	Koko tiimi	Whole team	Tarkoittaa sitoutuneisuutta kehitystiimissä projektiin.
3.	Informatiivinen työympäristö	Informative workspace	Työympäristön tulee sisältää mahdollisimman paljon kaikkien saatavilla olevaa informaatiota. Esimerkiksi tiimin jäsenet tietävät mitä muut tekevät ja tietävät tulevista iteraatioista ja niiden sisällöstä.
4.	Energiatehokas työskentely	Energized work	Yli 40 tuntia kestävä työviikko ei ole enää tehokasta, vaan kehittäjien tulee levätä saadakseen parhaimman hyödyn irti itsestään.
5.	Pariohjelmointi	Pair programming	Pariohjelmoinnissa kaksi tiimin jäsentä työskentelee keskenään keskustellen, ohjelmoiden ja testaten.
6.	Tarinat	Stories	Tarinat kuvaavat toiminnallisuutta XP:ssä vaatimusten sijaan, sillä vaatimuksiin liittyy huonoja kon-

			notaatioita. Tarinat ovat joustavampia.
7.	Viikkosykli	Weekly Cycle	Iteraation osat nojaavat viikkoihin ja tarinat on jaettu sen mukaan, että ne ehditään saada valmiiksi viikossa.
8.	Neljännessykli	Quarter Cycle	Neljännessykli on viikkosykliä karkeampi aikataulun suunnittelu isompien toimitusten kohdalla.
9.	Väljyys	Slack	Toimituspäivien ennakointi on haastavaa, joten on löydyttävä väljyyttä prosessista esimerkiksi matalampien prioriteetin tehtävien muodossa, joista voidaan joustaa.
10.	Kymmenen minuutin testit	Ten-Minute Build	Tiimin tulisi pystyä käynnistämään järjestelmä 10 minuutissa ja suorittaa samassa ajassa kaikki testit.
11.	Jatkuva integraatio	Continous Integration	Uusia ja vanhoja osia tuotteesta yhdistetään ja testataan useita kertoja päivässä.
12.	Testit ensin	Test-first programming	Testit tulisi kirjoittaa ennen varsinaista ohjelmointia.
13.	Inkrementaalinen kokonaissuunnittelu	Incremental Design	Jokapäiväinen ja vähäinen suunnittelu mahdollistaa inkrementaalisen järjestelmän kokonaissuunnittelun ja varmistaa yhdenmukaisen sekä tehokkaan arkkitehtuurin

4 Käytettävyyssuunnittelun testaus- ja asiantuntijamenetelmien sisällyttäminen ketteriin ohjelmistoprojekteihin

Tämä luku käsittelee käytettävyyssuunnittelun testaus- ja asiantuntijamenetelmien sisällyttämistä ketteriin menetelmiin. Aluksi luodaan lyhyt katsaus tilastoista ja siitä, mitä ongelmia ja haasteita liittyy käytettävyysmenetelmien sisällyttämiseen ketterään ohjelmistokehitykseen. Sen jälkeen esitetään, mitä ratkaisuja on ehdotettu käytettävyystestauksen ja asiantuntijamenetelmien sisällyttämiseksi ketterään ohjelmistokehitykseen. Lopuksi osoitetaan lähteistä sovellettu kokonaisuus asiantuntija- ja testausmenetelmien sisällyttämisestä ketterään ohjelmistokehitykseen.

4.1 Tilastot

On olemassa jonkin verran tilastollista tietoa käytettävyystestauksesta ketterissä ohjelmistoprojekteissa. Yleisimmät ketterän ohjelmistoprojektin käytettävyyden mittaamistekniikat ovat (Hussain ym., 2009a):

- käyttäjätarkkailut (56%),
- asiantuntijamenetelmät (51%) ja
- kenttätutkimukset (47%).

Nopeat iteratiivisemmat testaukset (40%) ovat myös kohtalaisen yleisiä, kuten myös laboratoriotestaukset (39%). Sen sijaan etätestauksia (engl. remote usability testing) käytetään vain noin joka neljännessä projektissa (26%). Automaattiset käytettävyysarvioinnit (engl. automated usability evaluations) ovat harvinaisempia (7,6%). Joka tapauksessa luvut ovat kasvussa (Hussain ym., 2009a). Tilastot tosin eivät kerro kuinka menetelmiä on sovellettu osana kehitystä, sillä ne voivat olla kevennettyjä tai täysmittaisia.

Käytettävyyssuunnittelun menetelmien käytöstä ketterissä projekteissa on asiantuntijoiden mukaan hyötyä niin lisäarvon kasvuna, parempana käytettävyytenä ja laatuna, kuin myös loppukäyttäjien tyytyväisyyden kasvuna. Lähes

kolmasosa (29%) asiantuntijoista on vahvasti sitä mieltä, että käytettävyysmenetelmien käyttö on antanut lisäarvoa ketterään ohjelmistokehitykseen ja lopputuotteisiin (Hussain ym., 2009a). Ohjelmistojen käytettävyys ja laatu ovat selkeästi parantuneet käytettävyyssuunnittelun menetelmien hyödyntämisen myötä, toteaa viidesosa asiantuntijoista (20%). Yli kaksi viidesosaa asiantuntijoista (43%) näkee asian samalla tavalla. Yli viidesosa asiantuntijoista (22%) toteaa loppukäyttäjien tyytyväisyyden kasvaneen, kun ohjelmistokehitysprojektissa on ketterien menetelmien rinnalla hyödynnetty käytettävyyssuunnittelun menetelmiä. Lisäksi noin kaksi viidesosaa (41%) asiantuntijoista on hieman samaa mieltä tyytyväisyyden kasvusta (Hussain ym., 2009a).

Huomattavasti pienempi osa asiantuntijoista on eri mieltä asiasta. Kaksi prosenttia (2%) asiantuntijoista on täysin eri mieltä lisäarvosta, käytettävyydestä, laadusta sekä loppukäyttäjän tyytyväisyydestä. Hieman eri mieltä olevien asiantuntijoiden osuus on noin kuuden-seitsemän prosentin luokkaa (Hussain ym., 2009a).

Merkittävä osa asiantuntijoista on käytettävyyssuunnittelun menetelmien hyödyntämisen puolella. He näkevät todellista hyötyä menetelmien käytöstä osana ketterää ohjelmistokehitystä. Hyödyntämisen esteenä ovat erilaiset haasteet ja ongelmat, joita käsitellään seuraavassa kappaleessa tarkemmin.

4.2 Haasteet ja ongelmat

Käytettävyyssuunnittelun sisällyttäminen ketterään ohjelmistokehitykseen on haaste – ei mahdottomuus (Eklund & Levingston, 2008). On olemassa useita haasteita ja ongelmia käyttäjäkeskeisen suunnittelun ja ketterän ohjelmistokehityksen välillä. Käyttäjäkeskeinen suunnittelu ja ketterä ohjelmistokehitys sisältävät tavoitteita, jotka ovat erilaisia, mutta toisiaan täydentäviä. Lisäksi ketterät menetelmät keskittyvät enemmän laajempaan kontekstiin, mikä tarkoittaa esimerkiksi ylläpitoa, pieniä julkaisuja ja projektinhallintaa osana menetelmää.

Ennen kehitysprosessin alkua käyttäjälähtöinen suunnittelu ja ketterä ohjelmistokehitys panostavat eri tavalla suunnitteluun. Käytettävyyssuunnittelussa toteutetaan laajaa etukäteissuunnittelua, mikä tarkoittaa, että käytettävyysasiantuntijat haluavat saada aikaiseksi kattavan kokonaiskuvan käyttäjästä ja käyttöliittymästä ennen varsinaista ohjelmiston täytäntöönpanoa. Ketterä ohjelmistokehitys tekee toisin, sillä useimmiten etukäteissuunnittelua tehdään tuskin lainkaan ja ensisijainen tavoite on saada mahdollisimman varhain toimiva ohjelmisto, joka ei sisällä vielä kaikkia toimintoja vaan osan niistä (Ambler, 2008).

Lee (2006) osoittaa käytettävyyssuunnittelun ja ketterän ohjelmistokehityksen tavoitteiden olevan erilaisia. Ohjelmistokehittäjät keskittyvät järjestelmän suunnitteluun, toteuttamiseen ja ylläpitoon. Toisaalta käytettävyyssuunnittelijat keskittyvät loppukäyttäjän kannalta tehokkaiden järjestelmien kehittämiseen. He eivät osallistu lähtökohtaisesti järjestelmän suunnitteluun tai to-

teuttamiseen, eikä liioin markkinoiden vaikutukseen toisin kuin ohjelmistokehittäjät. Joka tapauksessa nämä nähdään toisiaan täydentävinä tekijöinä.

Yksi merkittävä ongelma integroimisen toteuttamisen tiellä on kahden menetelmäperheen erot resurssien allokoinnissa. Ketterät menetelmät pyrkivät tuottamaan ohjelmiston pienissä erissä ja useissa iteraatioissa, kun taas käyttäjäkeskeisessä suunnittelussa käytetään paljon aikaa etukäteissuunnitteluun ja vaatimusmäärittelyyn ennen varsinaisen kehityksen aloittamista (Silva ym., 2011).

Ketterän ohjelmistokehityksen nojatessa iteratiivisuuteen, kehitysprosessi tarvitsisi käytettävyyssuunnittelun menetelmiä useassa eri iteraatiossa. Testauksen näkökulmasta suurimmaksi ongelmaksi muodostuu usein se, että perinteisesti käytettävyystestaukselle on hyvin rajattu budjetti ohjelmistokehitysprojektissa ja useimmiten tehdään yksi tai kaksi testausta koko projektin aikana (Eklund & Levingston, 2008).

Constantinen (2002) mukaan eXtreme Programming kohtaa erityisiä ongelmia, kun kyseessä käyttöliittymäkeskeinen projekti: Käyttöliittymätestaukset vaativat paljon aikaa ja muita resursseja, kuten työvoimaa. Täysimääräisen käytettävyystestauksen toteuttaminen vaatii toistettavia testejä ja useita loppukäyttäjien edustajia. Prototyyppien käyttö testauksessa toiminnallisten käyttöliittymien sijaan voi johtaa hyvinkin väärin johtopäätöksiin (Constantine, 2002). eXtreme Programmingissa on käytettävyyden testaukseen liittyviä toimintoja, kuten käytettävyyspalaute tai toteutuksen vertaaminen suunnitteluvaatimuksiin. Käytännössä ne liittyvät hyvin vähän käytettävyyteen, ja tarjoavat laihan perustan käytettävyyden onnistumiseksi lopputuotteessa (Jokela & Abrahamsson, 2004).

Chamberlain, Sharp ja Maiden (2006) listaavat käytettävyyssuunnittelun ja ketterien menetelmien yhteistoimivuuden ongelmakohdiksi:

- valtataistelut ohjelmistokehittäjien ja käytettävyyssuunnittelijoiden välillä,
- ajankäytön erot kehittämis- ja suunnitteluiteraatioissa,
- asiantuntijoiden välisen viestinnän,
- muiden osapuolten tarpeiden ymmärtämisen haluttomuuden,
- loppukäyttäjien osallistumattomuuden.

Ohjelmistokehittäjien ja käytettävyyssuunnittelijoiden valtataistelut syntyvät tilanteissa, joissa ei tiedetä selkeitä rooleja kehitysprosessissa. Esimerkiksi käytettävyyssuunnittelijoiden esittämiä muutoksia, jotka ovat syntyneet asiantuntija- tai testausmenetelmien käytön pohjalta, ei huomioida ohjelmistokehittäjien tuottamassa ohjelmistossa (Chamberlain ym., 2006). Käytettävyyden sivuuttaminen muutoksissa johtaisi tuotteen näennäiseen käytettävyyteen.

Toinen ongelmatilanne on kehityksen ja suunnittelun ajankäytön erot tuotannossa. Ohjelmistokehittäjien tavoitteena on tuottaa jokaisen iteraation lopussa jotain uutta, kun taas käytettävyyssuunnittelijoilla on pidempiä kehittämis-

jaksoja. Aikaa kuluu erityisesti arvioinnin tai testauksen aineiston analysoinnissa (Chamberlain ym., 2006).

Ohjelmistokehittäjien ja käytettävyyssuunnittelijoiden välinen viestintä ja vuorovaikutus voi muodostua ongelmaksi. Välttämättä ei ymmärretä toisen asiantuntijan työkulttuuria tai puhutaan samoista asioista eri käsittein. Erityinen viestinnällinen ongelma syntyy silloin, kun joku kehitystiimin jäsenistä ei osallistu täysiaikaisesti jokaiseen kehitysiteraatioon (Chamberlain ym., 2006). Esimerkiksi käytettävyyssuunnittelijat saattavat olla vain osan ajasta mukana ohjelmistoprojektissa ja viestintä jää vajavaiseksi.

Työkulttuurierot kehittäjien välillä voivat johtaa haluttomuuteen ymmärtää toisen näkökulmaa, jolloin syntyy edelläkin mainittua aliarviointia (Chamberlain ym., 2006). Ongelma voi olla lähtöisin myös käytettävyysasiantuntijoista, jotka eivät välttämättä haluaisi ymmärtää esimerkiksi alhaista budjetointia käytettävyyden suhteen projektissa.

Loppukäyttäjällä voi olla haluttomuutta tai esteitä osallistua ohjelmistokehitysprosessiin, jolloin on ongelmallista järjestää esimerkiksi käytettävyys- tai hyväksymistestaukset. Ongelman osapuolena voi olla myös ohjelmistokehittäjät ja käytettävyyssuunnittelijat, jotka saattavat väheksyä periaatteistaan huolimatta käyttäjän merkitystä kehityksessä (Chamberlain ym., 2006).

4.3 Vastauksia haasteisiin ja ongelmiin

Seuraavat kappaleet esittelevät mitä ratkaisuja on ehdotettu käytettävyystestaus- ja asiantuntijamenetelmien sisällyttämiseksi ketterään ohjelmistokehitykseen. Lisäksi tarkastellaan muita erityispiirteitä kuten käytettävyysasiantuntijoiden sijoittumista ketterään projektiin ja testauspaikkoja sekä tuoteomistajuutta.

4.3.1 Käytettävyyden asettuminen kehitysprosessiin

Käytettävyyssuunnittelun asiantuntija- ja testausmenetelmien käyttöä ketterässä ohjelmistokehityksessä voidaan lähestyä ohjelmistokehitysprosessin näkökulmasta. Eklundin ja Levingstonin (2008) mukaan käytettävyyden tulee olla lähellä ketterää ohjelmistokehitysprosessia, mutta sen tulee olla itsenäinen ja irrallinen kokonaisuus. Irtonaisuutta perustellaan parempien testiasetelmien ja -tulosten saavuttamisella, jolloin ohjelmistokehittäjien näkemykset eivät sekoitu testauksen suunnitteluun. Parsonsin, Lalin, Ryun ja Langen (2007) mukaan paras tapa hyödyntää käytettävyyssuunnittelun menetelmiä on käyttää niitä toimivan prototyypin testaukseen ja arviointiin, joiden pohjalta prototyyppiä voidaan kehittää yhdessä ohjelmistokehittäjien kanssa.

Humayoun, Dubinsky ja Catarci (2011) tarkastelevat menetelmien sisällyttämistä laajemmin. He ehdottavat niin sanottua kolmikantaintegraatiota, jossa käyttäjälähtöinen suunnittelu yhdistetään ketterään ohjelmistokehitykseen

kolmella tasolla: elinkaarimallin, iteraatioiden ja kehitysympäristön kautta. Tällöin koko rakenne olisi yhtenäinen. Myös Chamberlain ym. (2006) näkevät, että kehitysprosessin tulee olla yhteinen.

Yhteisen kehitysprosessin rakenne käytettävyyssuunnittelun osalta koetaan ongelmalliseksi, sillä esimerkiksi käytettävyyttä ei voida testata samaan aikaan ohjelmiston kehityksen kanssa (Najafi & Toyoshiba, 2008). Useat asiantuntijat ja tutkijat ehdottavat, että käytettävyyssuunnittelun asiantuntijoiden tai tiimien tulisi työskennellä ketterässä ohjelmistokehityksessä yhden iteraation (tai sprintin) edellä (Chamberlain ym., 2006; Williams & Ferguson, 2007; Najafi & Toyoshiba, 2008; Sy & Miller, 2008; Ungar & White, 2008). Projektin alkuvaiheen etumatka on nähty tärkeäksi (Chamberlain ym., 2006; Najafi & Toyoshiba, 2008; Sy & Miller, 2008). Myös kahden iteraation etumatkaa on ehdotettu (Budwig, Jeong, & Kelkar 2009).

4.3.2 Resurssit ja testauksen määrä

Ketterän ohjelmistoprojektin johdon tulisi ottaa kantaa ajankäytön eroihin esimerkiksi suunnittelemalla rinnakkaisia tai lomittaisia toimintoja, jotta kehitys jatkuisi ilman pysähdyksiä. Vapautta ja itsenäisyyttä ketterän manifestin mukaisesti tulisi jättää myös käytettävyysasiantuntijoille (Chamberlain ym., 2006). Resurssien allokoinnin kohdalla kompromissit ovat myös tärkeitä. Käytettävyysmenetelmiin panostaminen varmistaa soveltamisen hyödyn (Fox ym., 2008).

Budjetti asettaa omat rajansa käytettävyystestien määrälle. Kaikkein parhainta ketterän ohjelmistokehityksen kannalta olisi useampi kuin kaksi käytettävyystestausta, mutta suoritettuna pienemmällä määrällä loppukäyttäjiä (Eklund & Levingston, 2008). Kevyemmät ja epämuodolliset käytettävyystestaukset mahdollistavat useamman testin toteuttamisen projektissa (Memmel, Gundelsweiler & Reitererin, 2007).

4.3.3 Tiimit ja käytettävyyden asiantuntijat

Ketterässä ohjelmistoprojektissa tiimi on kehittämisen ydinyksikkö. Käytettävyyden ja sen testaamisen tuominen lähemmäksi kehitysprosessia on asiantuntijoiden mukaan mahdollista, jos kehitystiimiin liitetään käytettävyysasiantuntija (Chamberlain ym., 2006; Parsons ym., 2007; Memmel ym., 2007; Eklund & Levingston, 2008; Ambler, 2008).

Tiimiin liittyvä käytettävyysasiantuntija voi olla joko organisaation sisäinen tai ulkoinen henkilö (Ambler, 2008). Eklund ja Levingston (2008) painottavat, että organisaation ulkopuolelta hankittu käytettävyystestaaja tuo erilaista näkemystä kehitykseen ja tämän ei välttämättä tarvitse tällöin olla täysiaikainen. Ambler (2008) painottaa täysiaikaista projektiin sitoutumista, jotta projektin kaikilla osapuolilla olisi jatkuva ymmärrys ohjelmiston kehityksestä, muutoksista ja suunnasta.

Käytettävyysasiantuntija voi toteuttaa niin käytettävyystestaukset kuin asiantuntija-arvioinnit ohjelmistolle (Chamberlain ym., 2006; Memmel ym., 2007; Parsons ym., 2007), mutta käytettävyysasiantuntijan osallisuus kehitystiimiin ei ole välttämättömyys (Eklund & Levingston, 2008).

Käytettävyysasiantuntijoilla voi olla myös oma suunnittelutiiminsä (engl. UCD team), joka toimii osittain osana ketterää tiimiä. Suunnittelutiimin tarpeellisuus on todellinen, sillä yksittäisen käytettävyysasiantuntijan kohdalla työmäärä saattaa joskus olla aivan liian suuri. Suunnittelutiimin jäsenillä voi olla myös omat roolinsa. Esimerkkinä käytettävyystutkija (engl. UCD researcher) vastaa muun muassa käytettävyystestauksesta ja prototyypittäjä (engl. UCD prototyper) hoitaa prototyyppien kehittämisen ja käyttöliittymäspesifikaatioiden kirjoittamisen ohjelmistokehittäjille (Williams & Ferguson, 2007).

4.3.4 Käytettävyystestausmenetelmät ketterästi

Käytettävyystestaamiselle ketterässä ohjelmistokehityksessä on pääpiirteittäin kaksi lähestymistapaa. Käytettävyyden testausmenetelmiä voidaan käyttää sellaisenaan kuin ne on esitetty kirjallisuudessa tai niitä voidaan soveltaa ja käyttää niin sanotusti kevyempänä versiona tavoitellen ketterän manifestin mukaista kehittämistä (Silva ym., 2011). Esimerkiksi Benigni ym. (2009) sekä Kane (2003) ovat ehdottaneet heuristisen arvioinnin käyttöä osana ketterää ohjelmistokehitystä.

Chamberlain ym. (2006) ovat ehdottaneet osallistavan käytettävyystestaamisen hyödyntämistä ketterään ohjelmistokehitykseen. Yksittäisten osallistavien käytettävyystutkimusten teko ei riitä, sillä niiden on todettu osoittavana vain vähän vakavia käytettävyysongelmia (Kane, 2003). Parhaimmat tulokset saavutetaan, kun toteutetaan kaksi erilaista osallistavaa käytettävyystestiä ohjelmistolle (Humayoun ym., 2011).

Asiantuntijat ovat painottaneet käytettävyyden testauksen olevan laajamittaisena aivan liian kallista ja aikaa vievää, joten he ovat ehdottaneet kevyempiä tapoja käytettävyyden testaukseksi ketterässä ohjelmistoprojektissa (Kane, 2003; Eklund & Levingston, 2008; Memmel ym., 2007; Lee ym., 2009; Illmensee & Muff, 2009; Sohaib & Khan, 2011). Kevyitä käytettävyysarviointimenetelmiä pidetään kustannustehokkaina ja niitä voivat soveltaa myös ohjelmistokehittäjät pienellä koulutuksella (Kane, 2003). Myös Eklund ja Levingston (2008) pitävät erittäin tärkeänä kevyiden testausmenetelmien käyttöä.

Valmiiden kysymyssarjojen käyttäminen osana testaamista ei ole poissuljettua, jos ne soveltuvat tarpeeseen. Kyselyt voivat karsia kuluja, olla nopeita toteuttaa ja analysoida sekä ne ovat helposti vähäisemmällä asiantuntijuudella toteutettavissa (Memmel ym., 2007). Lisäksi on ehdotettu käyttäjälähtöisten käytettävyystestauksien sijaan käytettäväsi jonkin sortin automaatiotestausta tai etukäteen määriteltyä käytettävyystestausta (Hellmann, Khosseini-Khayat & Maurer 2010; Humayoun ym., 2011).

Testaaminen voi olla rakenteeltaan joustavampaa, jolloin saadaan enemmän tietoa ongelmista. Kysymykset voivat olla periaatteiden vastaisesti johdat-

televia, mutta se voi olla välttämätöntä, jotta saadaan tarpeeksi tietoa irti käyttäjästä. (Eklund & Levingston, 2008). Epämuodollisten testauksien tekeminen on oiva ratkaisu, kun on mahdotonta budjetin puitteissa tehdä laajamittaista käytettävyystestausta (Lee & McCrickard, 2007). Esimerkiksi Belchev ja Baker (2009) toteuttivat kevyen käytettävyystestauksen, joka koostui ruudunkaappauksesta, hiirenliikkeen mittaamisesta, ääneen ajattelu-menetelmästä sekä käyttäjän puheen nauhoittamisesta.

Eklundin ja Levingstonin (2008) mukaan käytettävyystestaus käyttäjillä tulisi olla hyvin erilaista ketterässä projektissa. Testauksen tulisi olla tutkivampaa ja vapaampaa, jolloin käyttäjät enemmänkin nostavat esiin ongelmat ja analysoivat testaajan apuna käytettävyyttä. Myöhemmässä vaiheessa testejä voidaan muotoilla täsmällisemmiksi ja rajatuimmiksi.

4.3.5 Prototyyppitestaus

Usein ketterissä projekteissa käytetään käyttöliittymävariaatioita, joista osa syntyy prototyyppien rakentamisessa (Chamberlain ym., 2006). Lähtökohta tulisi olla mahdollisimman monen erilaisen prototyypin rakentamisessa ja testaamisessa (Lee ym., 2009). Mahdollisuuksien mukaan tulisi toteuttaa vertailevaa testausta, mistä voi olla monipuolisiakin hyötyjä (Eklund & Levingston, 2008).

Prototyyppitestausta ilman sen tarkempaa määrittelyä ovat ehdottaneet useat asiantuntijat ja tutkimukset (Holzinger ym., 2005; Meszaros & Aston, 2006; Lee & McCrickard, 2007; Parsons ym., 2007; Obendorf & Finck, 2008). Ainoastaan Holzinger ym. (2005) ovat ehdottaneet prototyyppitestauksessa käytettäväksi ääneen ajattelu-menetelmää. Testaukset voidaan toteuttaa joko matalan tai korkean tason prototyyppeihin (Miller, 2005; Hussain ym., 2009). Lisäksi Singh (2008) ehdottaa prototyyppitestauksen osalta yksinkertaisten prototyyppien ja rautalankamallien testaamista sekä toimintoanalyysien tekoa. Humayoun ym., (2011) näkevät, että prototyyppitestausta tulisi tehdä mahdollisimman aikaisin projektissa.

Vuorovaikutteisten prototyyppien käytettävyystestaus on yksi mahdollinen tapa testata. Testauksen tavoitteena on luoda pohja paremmalle ja hiotummalle käyttöliittymäprototyypille (Williams & Ferguson, 2007; Federoff ym., 2008). Beyer, Holtzblatt ja Baker (2004) ehdottavat paperiprototyypin käyttöä ja testaamista, todeten sen olevan helpompi ratkaisu. Suurimman osan käyttäjätarinoiden sisällöstä ja järjestelmän toiminnallisuudesta voidaan sisällyttää yksinkertaiseen paperiprototyyppiin. Testauksen voi toteuttaa haastatteluna. Mikäli kehityksessä on aikaa ja resursseja enemmän, voidaan käyttää yksityiskohtaisempaa paperiprototyyppiä testaukseen (Beyer ym. 2004). Paperiprototyypillä testaamista ja sen iteraatiokehitystä ovat ehdottaneet myös monet muut (Miller, 2005; Fox ym., 2005; Williams & Ferguson, 2007; Hussain ym., 2008; Ungar & White, 2008).

Hellmann ym. (2010) ehdottavat prototyyppitestauksen jatkamista niin kauan, että lopputulos on selkeä, ja prototyypin voi käyttää suoraan käyttöliittymän rakentamiseen. Obendorf ja Finck (2008) ovat ehdottaneet paperiproto-

tyyppien skenaarioihin pohjautuvaa testaamista. Federoff ym. (2008) ovat maininneet asiantuntijamenetelmien käytön mahdollisuuden paperiprototyyppien arviointiin.

4.3.6 Testauksen ja arvioinnin ajoitus ketterästi

Yksi haastava näkökulma käytettävyyssuunnittelun testaus- ja asiantuntijamenetelmien käytöstä ketterään projektiin on menetelmien käytön ajoittaminen. Käytettävyyssuunnittelijat toimivat vähintään yhden iteraation ohjelmistokehittäjiä edellä, mutta kuinka käytettävyystestaus tai -arviointi ajoitettaisiin?

Ferreira ym. (2007) ehdottavat käytettävyystestauksien toteuttamista aina kehitysiteraation päätteeksi. Myös moni muu on ehdottanut samaa (Lee & McCrickard, 2007; Parsons ym., 2007; Düchting, Zimmermann, & Nebe, 2007). Humyoun ym. (2011) ehdottavat käytettävyystestauksen tekemistä iteraation loppupuolella sen tarkemmin määrittelemättä ajoitusta.

Yhtenä vaihtoehtona käytettävyystestausta on ehdotettu liitettäväksi hyväksymistestaukseen (Kane, 2003; Jokela & Abrahamsson, 2004; Lee & McCrickard, 2007; Benigni ym., 2010; Sohaib & Khan, 2011). Jokela ja Abrahamsson (2004) ehdottavat järjestelmällisten käytettävyystestien tekoa osana hyväksymistestausta. Järjestelmällistä käytettävyystestiä ei tarkemmin määritetty. Sen sijaan Sohaib ja Khan (2011) ehdottavat eXtreme Programming-pohjaisissa projekteissa heurististen arviointien käyttöä hyväksymistestausten yhteydessä ja ääneen ajattelu-menetelmän käyttöä pienten julkaisuiden tuotteistamisvaiheessa. Myös Lee ja McCrickard (2007) ehdottavat kevyiden käytettävyystestauksien tekemistä yhdessä hyväksymistestauksien kanssa.

Düchting, Zimmermann ja Nebe (2007) ehdottavat käytettävyystestauksen tekoa Scrum-projekteissa sprintin katselmoinnin ohessa. Detweiler (2007) ehdottaa etätestauksen kohdalla ajoittamista julkaisun loppuun, sillä iteraation aikaisen koodin testaaminen voi olla niin epävakaata, ettei testausta ole kannattavaa suorittaa.

Memmelin ym. (2007) mukaan jokaisessa kehitysvaiheessa voidaan hyödyntää monialaista tiimiä. Tiimi koostuisi esimerkiksi ohjelmistokehittäjistä, käytettävyysasiantuntijoista ja graafikoista. Tällöin sovellettaisiin käytettävyyssuunnittelun asiantuntijamenetelmiä, joilla arvioidaan ohjelmiston käytettävyyttä. He eivät tarkentaneet tai antaneet asiasta esimerkkiä.

Ketterän ohjelmistokehityksen yhtenä periaatteena on toimittaa asiakkaalle toimiva ohjelmisto aikaisessa vaiheessa kehitystä. Lisäksi toimivaa ohjelmistoa pidetään edistymisen mittarina (Agile Alliance, 2001), ja siksi useat asiantuntijat painottavat käytettävyyden testausta toimivalle ohjelmistolle (Lee & McCrickard, 2007; Sohaib & Khan, 2010; Ferreira, Noble, Biddle, 2007; Düchting, Zimmermann & Nebe, 2007; Detweiler, 2007). Toteutetun käyttöliittymän vahvistamiseen tähtäävää testaamista on myös moneen otteeseen ehdotettu (Detweiler, 2007; Wolkerstorfer ym., 2008; Sy & Miller, 2008; Najafi & Toyoshiba, 2008; Benigni ym. 2008). Käytettävyystestien ajoittamiseen voidaan ottaa

aivan toisenlainenkin kanta. Testauksia ja arviointeja tulisi toteuttaa aina, kun se on mahdollista (Najafi & Toyoshiba, 2008; Wolkerstorfer ym., 2008).

4.3.7 Muita erikoishuomioita

Lähtökohta käytettävyystestien tuloksien hyödyntämiseen tulisi vastata ketterän ohjelmistokehityksen periaatetta nopeasta muutokseen reagoimisesta (Lee, McCrickard & Stevens, 2009). Ferreira ym. (2007) toteavat, ettei testaamisella ole väliä, mikäli tuloksia ei hyödynnetä välittömästi kehityksessä.

Ferreiran, Noblen ja Biddlen mukaan (2007) läpi iteraatioiden jatkuva vuorovaikutus käyttöliittymäsuunnittelijoiden ja ketterien kehittäjien välillä on elinehto käytettävyyden syntymiselle lopputuotteessa. Vuorovaikutuksen olisi hyvä olla päivittäistä (Chamberlain ym., 2006). Parhaimmillaan vuorovaikutus voi opettaa paljon uutta sekä ohjelmistokehittäjille, että käytettävyysasiantuntijoille (Hussain ym., 2009b). Vuorovaikutuksen ei tulisi rajoittua pelkästään kehittäjien välille vaan ohjelmistokehitysprosessi tarvitsee toistuvasti käyttäjäpalautetta (Lee & McCrickard, 2007). Eklundin ja Levingstonin (2008) mukaan käyttäjien tulee osallistua jokaiseen vaiheeseen kehityksessä.

Mieluiten muutosten ja korjausten tekeminen käyttöliittymään tulisi tapahtua saman iteraation aikana, kun testi on tehty. Muutokset tulisi tehdä viimeistään seuraavassa iteraatiossa, mikäli ne ovat liian isoja välittömään reagoimiseen (Hussain ym., 2009b; Humayoun ym., 2011; Chamberlain ym., 2006).

Muutosten teko tarkoittaa myös nopeaa ongelmien käsittelyä. Eklundin ja Levingstonin (2007) mukaan testausraporttien tulisi syntyä lyhyessä ajassa testaamisen jälkeen. Tavoite testausraportin valmistumisesta on päivä tai kaksi käytettävyystestistä. Tällöin aikataulu pysyy yllä ja muutokset on helpompi ja nopeampi tehdä.

Käytettävyystestauksen ja arvioinnin ei tarvitse olla vain käytettävyysasiantuntijoiden ja loppukäyttäjien tekemää ketterässä ohjelmistoprojektissa. Memmel ym. (2007) toteavat, että todellisen hyödyn saavuttaa käyttämällä monialaista asiantuntijuutta käytettävyyden arvioimiseksi. Käytettävyyden arvioinnin voi toteuttaa esimerkiksi ohjelmistokehittäjät sekä graafikot, mikäli heidät on siihen koulutettu.

Ohjelmistokehittäjien osallistumisen käyttöliittymän arviointiin on todettu olevan hyväksi ohjelmiston käytettävyydelle, sillä aiemmin ohjelmistokehittäjien näkemys on keskittynyt nimenomaisesti loistavan ja tehokkaan koodin mukaiseen kehittämiseen (Albisetti, 2011). Yhteisarvioinnin ja -testauksen väitetään monialaisena olevan nopeampaa ja tehokkaampaa nostaen paremmin esiin käytettävyysongelmat (Memmel ym., 2007).

Käytettävyystestauksen toteuttamispaikan valintaan tulisi kiinnittää huomiota. Eklundin ja Levingstonin (2008) mukaan erityinen etu saavutetaan, jos käytettävyystestaus voidaan toteuttaa asiakkaan luonnollisessa työympäristössä. Erityisesti laboratoriossa toteutettavat käytettävyystestaukset nähdään jäykkänä vaihtoehtona, kun kyseessä on ketterä ohjelmistoprojekti (Illmensee & Muff, 2009).

Tuoteomistajuus on Scrumin erityispiirre. Hyvän käytettävyyden saavuttamiseksi Scrum-pohjaisessa ohjelmistoprojektissa on ehdotettu kahden tuoteomistajan käyttöä yhden sijaan. Tuoteomistajista toinen keskittyy käytettävyyteen sekä käyttökokemukseen ja toinen tavanomaisempiin Scrumin toimintoihin. Käytettävyyden tuoteomistaja vastaisi käytettävyyden toteutumisesta, testaamisesta ja arvioimisesta ohjelmistoprojektissa. Lisäksi tuoteomistajat toimisivat työparina ja vertaisina läpi ohjelmistoprojektin, jotta mahdollisimman hyvä vuorovaikutus saavutetaan (Singh, 2008).

Chon (2009) mukaan käytettävyys on osa liiketoimintastrategiaa ja käytettävyystestaus tulisi aina toteuttaa liiketoiminnan kannalta. Ratkaisuksi ehdotetaan käytettävyyssuunnittelijoiden parempaa perehtymistä liiketoimintatavoitteisiin. Tavoitteena on saavuttaa ohjelmiston kehityksessä optimaalinen ja kustannustehokas toiminta, jolloin myös käytettävyystestauksen ja –arvioinnin tulisi olla sellaista.

Useita luvussa esiteltyjä ratkaisuja yhdistää yksi yhteinen ongelma. Monet ratkaisuista pohjautuvat vähäiseen pilotointiin tai subjektiiviseen asiantuntijuuteen (Chamberlain ym., 2006; Ambler, 2008; Eklund & Levingston, 2008; Humayoun ym., 2011;). Sen sijaan osa ratkaisuista oli muotoutunut pilotoinnin kautta (Parsons ym., 2007; Memmel ym., 2007; Fox ym., 2008). Osassa kirjallisuudesta oli jätetty pilotoinnin puute tai toteuttaminen mainitsematta.

4.4 Piirteiden koostaminen

Kirjallisuudesta nousevat ratkaisut on sijoitettu taulukkoon 4. Taulukko tiivistää myös käytettävyyssuunnittelun ja ketterän ohjelmistokehityksen periaatteita vertailun saavuttamiseksi. Ratkaisuissa voidaan esittää useampikin vaihtoehto. Taulukon 4 jälkeen on lyhyt yhteenveto sen sisällöstä sekä pohdintaa.

Taulukko 4 Ratkaisuja käytettävyyssuunnittelun asiantuntija- ja testausmenetelmien soveltamisesta ketterään ohjelmistokehitykseen

Käytettävyyssuunnittelu	**Ketterä ohjelmistokehitys**	**Ratkaisu**
Laaja etukäteissuunnittelu	Vähäinen etukäteissuunnittelu (Scrum & XP)	Yhtenäinen, tiivis esisuunnittelu
Itsenäistä suunnittelua ja ratkaisujen esittelyä	Vahva vuorovaikutus tiimissä koko ajan (XP)	Läpi iteraatioiden jatkuvaa vuorovaikutusta käytettävyysasiantuntijoiden ja kehittäjien välillä
Laajaa suunnittelua käytettävyystestauksesta	Nopeaa ja pientä suunnittelua sekä testien etukäteiskirjoittamista (XP)	Suunniteltua tai vapaampaa käytettävyystestaamista
Yksi tai kaksi testiä kehityksen aikana	Iteraatioiden (sprinttien) lopulla tapahtuvat testaamiset, kuten yksikkö- sekä hyväksymistestit (Scrum & XP)	Käytettävyystestaukset helppo sijoittaa iteraatioiden loppuun tai kevyttä ja jatkuvaa käytettävyystestausta muiden ohjelmistotestien ohella
Käytettävyystestauksen palaute	Jatkuvaa palautetta kehityksen aikana (XP)	Mahdollisimman jatkuvaa palautetta loppukäyttäjältä koko kehityksen ajan
Projektipäällikkö	Tuoteomistaja (Scrum)	Tuoteomistaja ja käytettävyyden tuoteomistaja (Scrum)
Laboratoriotestaus, kenttätestaus	Läsnä oleva asiakas (Scrum & XP)	Testausta asiakkaan tiloissa tai läsnä oleva asiakas, vältettävä laboratoriotestausta
Käytettävyysasiantuntija(t)	Kehitystiimi (Scrum & XP)	Käytettävyysasiantuntija kehitystiimin jäseneksi tai käytettävyysasiantuntijoiden tiimi
Jäykät toimintamallit	Kevyet toimintamallit (Scrum & XP)	Käytettävyyssuunnittelun menetelmien laajamittainen käyttö ketterässä ohjelmistokehityksessä tai kevyt, mukautuva lähestymistapa
Käytettävyysasiantuntijat ja käyttäjät kehityksessä	Monialainen osaaminen ja käyttäjät kehityksessä (Scrum & XP)	Monialaista asiantuntija- ja testausmenetelmien käyttöä

Käytettävyyssuunnittelun testaus- ja asiantuntijamenetelmien toimimiseksi ketterässä ohjelmistokehityksessä on löydettävä tasapaino kahden menetelmän välillä ohjelmistoprojektien etukäteissuunnitteluun. Ketterä ohjelmistokehitys on päämenetelmä, joten käytettävyyssuunnittelumenetelmien tulisi mukautua kevyempään etukäteissuunnitteluun. Suunnitelman tulee olla tiivis ja sisältää joustovaraa.

Käytettävyysasiantuntijoiden toimiminen itsenäisenä ryhmänä voisi rikkoa ketterän ohjelmistokehityksen periaatetta jatkuvasta vuorovaikutuksesta. Näin ollen olisikin tehokkaampaa, että tiimin jäsenenä toimii jatkuvasti yksi käytettävyysasiantuntija. Vaihtoehtoinen tapa on käyttää kahta tiimiä, käytettävyys- ja ketterätiimi, joiden välillä on jatkuvaa vuorovaikutusta läpi ohjelmistoprojektin. Näin toimittuna selkeytyisi myös jokaisen osapuolen rooli projektissa.

Käytettävyyssuunnittelu vaatii selkeän ja laajan suunnitelman testauksesta. Ketterissä menetelmissä on periaatteena kirjoittaa ohjelmistotestit etukäteen.

Periaatteeseen nojaten käytettävyystestit voidaan muotoilla etukäteen, mutta lisäksi voidaan noudattaa ketterää periaatetta mukautumisesta. Tällöin käytettävyystestit muotoutuisivat projektin aikana. Lisäksi testien sisältö voi olla vapaampaa, epämuodollisempaa ja jopa johdattelevaa.

Ohjelmistokehityksen rajallinen budjetti mahdollistaa usein vain yhden tai kahden käytettävyystestauksen suorittamisen projektissa. Ratkaisuna tälle ongelmalle on kaksi tapaa: yksinkertaisesti suppeammat testit tai kevennettyjen käytettävyystestausmenetelmien käyttö. Testien ajoitus on usein optimaalisin iteraatioiden lopussa, jolloin seuraava iteraatio voi hyödyntää käytettävyystestauksen raportin tuloksia.

Prototyyppitestaus on tehokas tapa saada riskit esiin liittyen käyttöliittymän suunnitteluratkaisuihin, sillä korjauskulut myöhemmässä vaiheessa toimivasta ohjelmistosta maksaisi huomattavasti enemmän. Pahimmillaan voidaan joutua uudelleen mallintamaan tai ohjelmoimaan koko järjestelmä. Prototyypit toimivat myös erittäin hyvänä pohjana lopullisen käyttöliittymän rakentamiseksi.

Useimmiten asiakas on myös tilatun tuotteen loppukäyttäjä. Ketterän ohjelmistokehityksen mukaan olisi hyvä, että asiakas on osana tuotteen kehittämistä. Tämä periaate mahdollistaa myös sen, että käytettävyyttä voidaan testata nimenomaan läsnä olevalla asiakkaalla. Vaihtoehtoisesti testaus voi tapahtua myös asiakkaan tiloissa luonnollisen työympäristön saavuttamiseksi.

Ketterissä menetelmissä moniosaajat ovat merkittävässä asemassa. Käytettävyyden kannalta monialaisuus on kaiken onnistumisen perusta. Kokonaisvaltaisen ymmärryksen saavuttamiseksi on ohjelmistokehittäjienkin mahdollista osallistua käytettävyystestaukseen tai sen suunnitteluun.

Menetelmät on taulukossa eriteltynä, vaikka useimmiten ohjelmistoprojekteissa hyödynnetään Scrumia projektinhallinnassa ja sovelletaan eXtreme Programming-menetelmiä ja -käytäntöjä kehitysvaiheissa. Näin ollen suurin osa taulukossa 4 esitetyistä periaatteista soveltuu lähes kaikkiin ketteriin ohjelmistoprojekteihin, joissa painotetaan käytettävyyden merkitystä.

5 Pohdinta ja yhteenveto

Ketterä tapa kehittää ohjelmistoja yleistyy vuosi vuodelta, mutta käytettävyysnäkökulmat ovat ohjelmistoprojekteissa jääneet taka-alalle. Asiantuntijat ovat huomanneet, että lopputuotteen laatu on parempi, kun käytettävyyssuunnittelun testaus- ja asiantuntijamenetelmiä on hyödynnetty. Jotta käytettävyys olisi hyvä ketterällä tavalla kehitetyissä ohjelmistoissa, asiantuntijat ovat ehdottaneet useita tapoja sisällyttää käytettävyyssuunnittelun menetelmiä osaksi ketterää ohjelmistokehitystä.

Tutkielmassa tarkasteltiin käytettävyyssuunnittelun asiantuntija- ja testausmenetelmien sisällyttämistä ketterään ohjelmistokehitykseen. Aihepiiriä lähestyttiin aluksi käytettävyyssuunnittelun näkökulmasta esitellen sen määritelmiä, yleisiä piirteitä sekä menetelmiä. Seuraavaksi käsiteltiin ketterän ohjelmistokehityksen filosofiaa ja sen menetelmiä. Tutkielmassa esiteltiin myös tilastoja käytettävyysmenetelmien hyödyntämisestä ketterässä ohjelmistokehityksessä sekä kuvattiin haasteita ja ongelmia aihepiiriin liittyen.

Käytettävyyssuunnittelun menetelmien soveltaminen ketterissä menetelmissä jakautuu kahteen pääleiriin. Ensimmäinen ja yleisempi lähestymistapa käytettävyyden arviointiin on kevyet, epämuodolliset ja kustannustehokkaat käytettävyystestaukset käyttäjillä sekä kevyet asiantuntijamenetelmät osana ketterää ohjelmistoprojektia. Toinen ja harvinaisempi tapa on toteuttaa täysimittaisia käytettävyystestauksia käyttäjillä.

Erityisesti kevyemmät ja kustannustehokkaammat asiantuntija- ja testausmenetelmät ovat hyvä ratkaisu projekteille, joissa on tiukka aikataulu. Tällöin myös testausraportit syntyvät nopeasti. Käytettävyystestaukset ovat helpointa toteuttaa iteraation lopussa hyväksymistestien ohessa. Käytettävyystestauksen tulokset tulisi hyödyntää heti tai seuraavassa alkavassa iteraatiossa.

Kehitystiimiin suositellaan liitettäväksi käytettävyysasiantuntija, joka toimisi täysivaltaisena tiimiläisenä. Asiantuntija toteuttaisi tai vastaisi joko testaukset tai asiantuntija-arvioinnit. Lisäksi erillinen käytettävyysasiantuntijatiimi on mahdollinen ratkaisu.

Erityisesti prototyyppitestaus ketterässä ohjelmistokehityksessä nähdään tärkeänä. Käyttöliittymän rakentaminen tulee halvemmaksi paremmalla proto-

tyyppitestauksella ja iteratiivisella kehittämisellä, jolloin vältytään uudelleenohjelmoinnilta ja -mallintamiselta.

Eräs suurimmista ongelmista on se, ettei esitettyjä ratkaisuja ole pilotoitu laajamittaisesti. Osalle esitellyistä ratkaisuissa on toteutettu yksittäisiä ja suppeita tapaustutkimuksia. Seuraava askel olisi toteuttaa parempi pilotointi menetelmille.

Edellisen luvun taulukko 4 on varsinainen vastaus tutkimuskysymykseen. Taulukko 4 tarjoaa tiivistelmän periaatteista, joihin ketterä ohjelmistokehitystiimi voi nojata harkitessa käytettävyyssuunnittelun menetelmien käyttöä.

Taulukossa 5 on tiivistettynä vastaukset tutkimuskysymyksiin.

Taulukko 5 Tiivistetyt vastaukset tutkimuskysymyksiin

Kysymys	**Tiivistetty vastaus**
Millaisia ratkaisuja käytettävyystestauksen sisällyttämiseksi ketterään ohjelmistokehitykseen on esitetty?	Testaus- ja asiantuntijamenetelmiä voi käyttää sellaisenaan ketterässä ohjelmistokehityksessä tai joustavasti, vapaammin ja näin ollen kustannustehokkaasti.
Mitä etuja käytettävyyssuunnittelun testaus- ja arviointimenetelmien sisällyttämisestä ketteriin menetelmiin on?	Saavutetaan laadukkaampi käytettävyys ja lisäarvoa loppukäyttäjälle. Mahdollisia hyötyjä ovat myös kustannustehokkuus ja vähäisempi korjauksen tarve.
Mitä ongelmia liittyy käytettävyyssuunnittelun testaus- ja arviointimenetelmien sisällyttämiseen ketterään ohjelmistokehitykseen?	Resurssien allokoinnin ja suunnittelun tasapainottaminen kahden menetelmäperheen kesken on ongelmallista. Ohjelmistokehittäjien puutteelliset tiedot käytettävyydestä luo haasteita ja ongelmia.

Tutkielman erityisiä löytöjä olivat vajaa pilotointi ehdotetuissa ratkaisuissa sekä kaksi pääleiriä käytettävyyssuunnittelun asiantuntija- ja testausmenetelmien käytöstä osana ketterää ohjelmistokehitystä.

Lyhyiden ja koostavien kirjallisuuskatsausten ohella varsinaista syvällisempää analyysia käytettävyyssuunnittelun menetelmien sisällyttämiseksi ketterään ohjelmistokehitykseen ei ole toteutettu ehdotetuista ratkaisuista.

Vaikka aihepiirin kirjallisuudelle on toteutettu luokitteluanalyysi, se on toteutettu vain yleisellä tasolla. Menetelmäkohtainen luokittelu olisi tarpeellinen. Jatkotutkimuskysymys olisikin kuinka Scrum-projekteihin voidaan sisällyttää käytettävyyssuunnittelun menetelmiä?

Toinen aihepiiristä nouseva ja hyvin ajankohtainen jatkokysymys on se, kuinka laajasti tutkielmassa esiteltyjä periaatteita hyödynnetään IT-alalla ketterissä projekteissa? Myös muita kysymyksiä on noussut. Esimerkiksi kuinka käytettävyyssuunnittelun asiantuntijamenetelmiä hyödynnetään ketterissä projekteissa tai millainen merkitys prototyyppitestauksella on käytettävyyden onnistumiseksi ketterässä ohjelmistoprojektissa?

Lähteet

Abrahamsson, P., Salo, O., Ronkainen, J. & Warsta, J. (2002). Agile software development methods. Review and analysis. *VTT Publications*, nro 478, s. 9-17.

Albisetti, M. 2010. Launchpad's Quest for a Better and Agile User Interface. *Agile Processes in Software Engineering and Extreme Programming*, (s. 244-250). Springer.

Ambler, SW. (2008). Tailoring usability into Agile software development projects. Teoksessa *Maturing Usability* (s. 99-120). Springer London.

Agile Alliance, (2001a). Ketterän ohjelmistokehityksen julistus. Agile Alliance. Haettu 10.10.2011 osoitteesta http://agilemanifesto.org/iso/fi/

Agile Alliance, (2001b). Ketterän ohjelmistokehityksen 12 periaatetta. Agile Alliance. Haettu 10.2.2012 osoitteesta http://agilemanifesto.org/iso/fi/principles.html

Beck, K. & Anders, C. (2004). *Extreme Programming Explained: Embrace Change*. Addison-Wesley.

Belchev, B. & Baker, P. (2009). Improving Obama Campaign Software: Learning from Users. *Proceedings of the AGILE 2009*, Chigaco, 2009, (s. 395-399). IEEE Computer Society.

Benigni, G., Gervasi, O., Passeri, F. L., & Kim, T.-H. (2010). USABAGILE_Web: A Web Agile Usability Approach for Web Site Design. *Proceedings of Computational Science and Its Applications (ICCSA 2010),* (s. 422-431). Springer.

Bevan, N. (1999). Design for usability. *Human-computer interaction: proceedings of HCI International'99 (the 8th International Conference on Human-Computer Interaction)*, (s. 762-766). CiteseerX.

Bevan, N., Kirakowski, J. & Maissela, J. (1991). What is Usability? *Proceedings of the 4th International Conference on HCI.* CiteseerX.

Beyer, H., Holtzblatt, H., & Baker, L. (2004). An Agile Customer-Centered Method: Rapid Contextual Design. *Proceedings of Extreme Programming and Agile Methods - XP/Agile Universe 2004*, Calgary, 2004, (s. 527-554). Springer-Verlag.

Budwig, M., Jeong, S., & Kelkar, K. (2009). When User Experience Met Agile: A Case Study. *Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*, Boston, (s. 3075-3084). ACM.

Chamberlain, S., Sharp, H. & Maiden, N. (2006). Towards a Framework for Integrating Agile Development and User-Centred Design. *Extreme Programming and Agile Processes in Software Engineering,* Oulu, 2006, (s. 143-153). Springer-Verlag Berlin Heidelberg.

Cho, L. (2009). Adopting an Agile Culture: A User Experience Team's Journey. *Proceedings of the AGILE 2009*, Chigaco, 2009, (s. 416-421). IEEE Computer Society.

Cockburn, A. (2001). *Agile Software Development*. Addison Wesley.

Constantine, L. (2002). Process Agility and Software Usability: Toward Lightweight Usage-Centered Design. *Information Age.*

Detweiler, M. (2007). Managing UCD Within Agile Projects. *Interactions, 14,* (s. 40-42). ACM.

Düchting, M., Zimmermann, D., & Nebe, K. (2007). Incorporating User Centered Requirement Engineering into Agile Software Development. *Proceedings of the 12th international conference on Human-computer interaction* (HCI'07), (s. 58-67*).* ACM.

Eklund, J. & Levingston, C. 2008. Usability in Agile development. UX Research.

Federoff, M., Villamor, C., Miller, L., Patton, J., Rosenstein, A., Baxter, K., & Kelkar, K. (2008). Extreme usability: adapting research approaches for agile development. *CHI '08 extended abstracts on Human factors in computing systems* (CHI EA '08), (s. 2269-2272). ACM.

Ferreira, J., Noble, J. & Biddle, R., (2007a). Agile Development Iterations and UI Design. *AGILE 2007 , (*s.50-58, 13-17*)*. IEEE.

Ferreira, J., Noble, J. & Biddle, R., (2007b). Up-Front Interaction Design in Agile Development. *Agile Processes in Software Engineering and Extreme Programming 2007*, (s. 9-16). Springer.

Ferreira, J., Sharp, H. & Robinson, H. (2011). User experience design and agile development: managing cooperation through articulation work. *Software Practice and Experience 2011, (s.* 963-974). John Wiley & Sons.

Fox, D., Sillito, J. & Maurer, F. (2008). Agile methods and User-Centered design: How these two methodologies are being successfully integrated in industry. *AGILE 2008 Conference*, (s. 63-72). ACM.

Hellmann, TD., Khosseini-Khayat, A., & Maurer, F. (2010). Supporting Test-Driven Development of graphical user interface Using Agile Interaction

Design. *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference*, (s. 444-447). IEEE.

Holzinger, A. (2005). Usability Engineering Methods for Software Developers. *Communications* 48, 1, (s. 71-74). ACM.

Humayoun, S., Dubinsky, Y., & Catarci, T. (2011). A Three-Fold Integration Framework to Incorporate User-Centered Design into Agile Software Development. *Lecture Notes in Computer Science, Human Centered Design*, (s. 55-64). Springer Berlin, Heidelberg.

Hussain, Z., Slany, W. & Holzinger, A. (2009a). *Current State of Agile User-Centered Design: A Survey*. Springer-Verlag Berlin Heidelberg.

Hussain, Z., Slany, W. & Holzinger, A. (2009b). *Investigating Agile User-Centered Design in Practice: A Grounded Theory Perspective*. Springer-Verlag Berlin Heidelberg.

Illmensee, T. & Muff, A. (2009). 5 Users Every Friday: A Case Study in Applied Research. *Proceedings of the AGILE 2009*, Chigaco, 2009, (s. 404-409). IEEE Computer Society.

Jokela, T. & Abrahamsson, P. (2004). Usability Assessment of an Extreme Programming Project: Close Co-operation with the Customer Does not Equal to Good Usability. *Lecture Notes in Computer Science, Product Focused Software Process Improvement*, (s. 393-407). Spinger-Verlag Berlin, Heidelberg.

Jokela, T., Iivari, N., Matero, J. & Karukka, M. (2003). The standard of user-centered design and the standard definition of usability: analyzing ISO 13407 against ISO 9241-11. *Proceedings of the Latin American conference on Human-computer interaction (CLIHC '03), (*s. 53-60). ACM.

Kane, D. (2003). Finding a Place for Discount Usability Engineering in Agile Development: Throwing Down the Gauntlet. *Proceedings of the Agile Development Conference 2003.* (s. 40-46). IEEE.

Lai, J., Honda, T. & Yang, M. (2010). A study of the role of user-centered design methods in design team projects. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 24, (s.* 303-316). ACM.

Larman, C. (2003). *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley.

Lee, J.C. (2006). Embracing Agile Development of Usable Software systems. *CHI '06 Extended abstracts on Human factors in computing systems (CHI EA '06),* (s. 1767-1770). ACM.

Lee, J.C. & McCrickard, D.S. (2007). Towards Extreme(ly) Usable Software: Exploring Tensions Between Usability and Agile Software Development. *Proceedings of the AGILE 2007,* (s. 59-71). Computer Society. IEEE.

Lee, J.C., McCrickard, D. & Stevens, K.T., (2009). Examining the Foundations of Agile Usability with eXtreme Scenario-Based Design. *Proceedings of the AGILE 2009,* (s. 3-10). IEEE.

Leffingwell, D. (2007). *Scaling software agility: Best practices for large enterprises*. Upper Saddle River, New Jersey, USA: Addison-Wesley Professional.

McInerney, P., Maurer, F. (2005). *UCD in Agile Projects: Dream Team or Odd Couple?*. Interactions 12, 6, (s. 19-23) ACM.

Memmel, T., Gundelsweiler, F. & Reiterer, H. (2007). Agile Human-Centered Software Engineering. *BCS-HCI '07 Proceedings of the 21st British HCI Group Annual Conference on People and Computers,* (s. 167-175). ACM.

Meszaros, G. & Aston, J. (2006). Adding Usability Testing to an Agile Project. *Proceedings of the conference on AGILE 2006*, Minneapolis, 2006, (s. 289-294). IEEE Computer Society.

Miller, L. (2005). Case Study of Customer Input For a Successful Product. *Proceedings of the Agile Development Conference (ADC '05),* Denver, 2005, (s. 225-234). IEEE Computer Society.

Najafi, M., & Toyoshiba, L. 2008. Two Case Studies of User Experience Design and Agile Development. *Proceedings of the AGILE '08*, (s. 531-536). IEEE.

Nielsen, J. (1993). *Usability Engineering*. Morgan-Kaufmann.

Obendorf, H., & Finck, M. (2008). Scenario-Based Usability Engineering Techniques in Agile Development Processes. *Proceedings of CHI '08* extended *abstracts on Human factors in computing systems*, Florence, 2008, (s. 2159-2166). ACM.

Parsons, D., Lal, R., Ryu, H. & Lange, M. (2007). Software Development Methodologies, Agile Development and Usability Engineering. *ACIS 2007 Proceedings*, (s. 172-178). CiteSeer.

Silva da Silva, T., Martin, A., Maurer, F. & Silveira, M. (2011). User-Centered Design and Agile Methods: A Systematic Review. AGILE Conference 2011, (s. 77-86). IEEE.

Singh, M. (2008). U-SCRUM: An Agile Methodology for Promoting Usability. *AGILE '08. Conference*, (s. 555-560). IEEE.

Sohaib, O. & Khan, K. (2010). Integrating Usability Engineering and Agile Software Development: A Literature Review. *2010 International Conference On Computer Design And Appliations (ICCDA 2010)*, (s. 32-38). IEEE.

Sohaib, O. & Khan, K. (2011). Incorporating Discount Usability in Extreme Programming. *International Journal of Software Engineering and Its Applications,* 5, 1, (s. 51-62). SERSC.

Sutherland, J., & Schwaber, K. (2007). The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process. Origins, 202. The Scrum Training Institute. CiteSeerX.

Sy, D., & Miller, L. 2008. Optimizing Agile User-Centred Design. Proceedings of CHI '08 extended abstracts on Human factors in computing systems, Florence, 2008, (s. 3987-3900). ACM.

Ungar, J. & White, J. (2008). Agile User Centered Design: Enter the Design Studio - A Case Study. *Proceedings of CHI '08 extended abstracts on Human factors in computing systems*, Florence, 2008, (s. 2167-2178). ACM.

Williams, H. & Ferguson, A. (2007). The UCD Perspective: Before and After Agile. *Proceedings of the AGILE 2007,* Washington D.C., 2007, (s. 285-290). ACM.

Wolkerstorfer, P., Tscheligi, M., Sefelin, R., Milchrahm, H., Hussain, Z., Lechner, M., & Shahzad S. (2008). Probing an agile usability process. *Proceedings of CHI '08* extended *abstracts on Human factors in computing systems* (CHI EA '08), (s. 2151-2158). ACM.

Vredenburg, K., Mao, J.Y., Smith, P. & Carey, T. (2002). A Survey of User-Centered Design Practice. *Proceedings of the SIGCHI conference on Human factors in computing systems: Changing our world, changing ourselves*, (s. 471-478). ACM.