

Käyttäjäkeskeinen suunnittelu Scrum-prosessimallissa

Linda Hellman

Helsinki 5. toukokuuta 2009

Pro gradu -tutkielma

HELSINGIN YLIOPISTO

Tietojenkäsittelytieteen laitos

Tiedekunta/Osasto – Fakultet/Sektion – Faculty/Section Matemaattis-luonnontieteellinen tiedekunta		Laitos – Institution – Department Tietojenkäsittelytieteen laitos	
Tekijä – Författare – Author Linda Marianne Hellman			
Työn nimi – Arbetets titel – Title Käyttäjakeskeinen suunnittelu Scrum-prosessimallissa			
Oppiaine – Läroämne – Subject Tietojenkäsittelytiede			
Työn laji – Arbetets art – Level Pro Gradu -tutkielma	Aika – Datum – Month and year 5.5.2009	Sivumäärä – Sidoantal – Number of pages 68	
Tiivistelmä – Referat – Abstract <p>Tutkielmassa kehitetään Scrum-prosessimallia siten, että siihen lisätään käyttäjakeskeisen suunnittelu -prosessin käytäntöjä ja menetelmiä. Tällöin käytettävyys huomioidaan johdonmukaisesti koko projektin ajan ja valmiin tuotteen käytettävyys ja hyödyllisyys voidaan varmistaa projektin alusta asti.</p> <p>Scrum on ketterä prosessimalli, joka keskittyy projektinhallintaan, ja joka tuottaa nopeasti laadukkaan tuotteen markkinoille. Käytettävyys on laatuattribuutti, jonka avulla voidaan mitata ohjelmiston toimivuutta käyttäjän näkökulmasta. Käyttäjakeskeisessä suunnittelu -prosessissa pyritään ymmärtämään käyttäjää, tämän tarpeita ja työympäristöä ja niiden pohjalta suunnittelemaan ohjelmisto, joka on käytettävä ja hyödyllinen. Scrumin ja käyttäjakeskeisen suunnittelun yhdistäminen tuo hyötyjä projektiin.</p> <p>Scrumissa ja käyttäjakeskeisissä suunnitteluprosesseissa samankaltaisuuksia ovat prosessien iteratiivinen luonne, asiakkaan tai käyttäjän tärkeä rooli koko kehitysprojektin ajan sekä tiivis tiimityöskentely ja kommunikointi. Scrumin ja käyttäjakeskeisten suunnitteluprosessien eroavaisuuksissa täytyy tehdä kompromisseja, jotta saadaan joustava ja mukautuva prosessimalli, jossa yhdistyy sekä Scrumin että käyttäjakeskeisten suunnitteluprosessien hyvät puolet.</p> <p>Tutkielman tuloksena käyttäjälähtöiseen Scrumiin otetaan käyttöliittymäsuunnittelija Scrum-tiimin jäseneksi. Käyttöliittymäsuunnittelija käyttää arviointimenetelmiä käyttöliittymän käytettävyyden varmistamisessa, ja jakaa tietoa suunnittelun vaiheista ja etenemisestä Scrum-tiimille. Käyttäjälähtöisessä Scrumissa käyttöliittymäsuunnittelu tehdään omassa iteraatiossa ennen kehitysiteraatiota.</p> <p>ACM Computing Classification System (CCS 1998): D.2.2. Design Tools and Techniques D.2.9. Management K.6.3. Project Management</p>			
Avainsanat – Nyckelord – Keywords Ketterät menetelmät, Scrum, käytettävyys, käyttäjakeskeinen suunnittelu			
Säilytyspaikka – Förvaringställe – Where deposited Kumpulan tiedekirjasto, sarjanumero C-			
Muita tietoja – Övriga uppgifter – Additional information			

Sisältö

1	Johdanto.....	1
2	Käytettävyys ja käytettävyyden arviointi.....	4
2.1	Käytettävyyden määritelmä	4
2.1.1	Nielsenin käytettävyys	4
2.1.2	ISO 9241-11	6
2.2	Käytettävyyden arviointi.....	8
2.2.1	Asiantuntijamenetelmät.....	9
2.2.2	Testimenetelmät	11
2.3	Käytettävyys laadun mittarina	12
3	Käyttäjakeskeinen suunnitteluprosessi	14
3.1	Käyttöliitymäsuunnittelijan määritelmä.....	14
3.2	Käyttäjakeskeinen suunnitteluprosessi	16
3.3	Käyttäjakeskeisen suunnitteluprosessin käytäntöjä	19
3.4	Käyttäjakeskeinen suunnittelu osana ohjelmistokehitystä.....	21
4	Scrum – ketterä ohjelmistokehitysprosessi	23
4.1	Ketterät menetelmät ja Scrum.....	23
4.1.1	Ketterät menetelmät	23
4.1.2	Ketterien menetelmien yleistyminen.....	25
4.2	Scrum-prosessimalli.....	26
4.2.1	Scrumin historia	26
4.2.2	Scrum-tiimin roolit.....	27
4.2.3	Prosessin kuvaus	28
4.3	XP:n ideoita yhdistettynä Scrumiin	33
4.4	Scrum ja tarve käyttäjälähtöiselle suunnittelulle	35

5 Käyttäjakeskeinen suunnitteluprosessi ja Scrum.....	37
5.1 Haasteet käyttäjakeskeisessä suunnittelussa ja Scrumissa.....	37
5.1.1 Oikea aika ja paikka käyttäjakeskeiselle suunnittelulle Scrumissa	38
5.1.2 Vaatimusten kerääminen	39
5.1.3 Suunnittelun määrä.....	40
5.1.4 Käyttöliittymäsuunnitteluun käytettävä aika.....	41
5.1.5 Käyttöliittymän testaus ennen toteutusta.....	41
5.1.6 Asiakas ja käyttäjä.....	42
5.1.7 Roolien vastuut.....	43
5.1.8 Tiimin jäsenten sijainti ja yhteistyö.....	44
5.2 Yhteiset käytännöt prosessien yhdistämisessä.....	44
5.2.1 Iteratiivinen luonne	44
5.2.2 Kommunikaatio sidosryhmien välillä	45
5.2.3 Käytettävyydestä kehittäjille.....	47
5.2.4 Aikataulutus	47
5.2.5 Visio tuotteesta.....	48
5.2.6 Ohjelmiston käytettävyyden testaus.....	48
5.2.7 Johdon tuki	48
6 Käyttäjälähtöinen Scrum -prosessi	50
6.1 Prosessinäkökulma.....	50
6.2 Projektinäkökulma prosessiin	51
6.3 Julkaisunäkökulma prosessiin.....	52
6.4 Iteraationäkökulma prosessiin	53
6.5 Käyttäjälähtöisen Scrum-prosessimallin erot Scrumiin.....	56
6.6 Muokatun prosessimallin rajoitukset	57
6.7 Jatkokehitystä käyttäjälähtöiseen Scrum -prosessimalliin.....	58
7 Yhteenveto	60
Lähteet	62

1 Johdanto

Käyttäjälähtöinen näkökulma on tärkeä sekä ketterissä menetelmissä että käyttäjakeskeisessä suunnittelussa. Molemmat menetelmät pyrkivät takaamaan lopputuloksen onnistumisen ottamalla käyttäjät ja asiakkaat prosessiin mukaan mahdollisimman aikaisessa vaiheessa ja antamalla käyttäjän mielipiteille arvoa. Ketterissä menetelmissä asiakas antaa vaatimukset kehitystiimille. Käyttäjakeskeisessä suunnittelussa käyttöliittymäsuunnittelija tutkii käyttäjän käyttäytymistä ja tarpeita, ja hahmottaa sen perusteella vaatimuksia ohjelmiston käytölle.

Ketterät menetelmät painottavat erityisesti ihmisten välistä kommunikointia ja asiakasyhteistyötä eivätkä niinkään laajojen ja tarkkojen dokumenttien tekoa [Bec01]. Näitä dynaamisempia ja kevyempiä ohjelmistokehitysprosesseja alettiin kehittää 1990-luvun puolella välissä vastalauseeksi aikaisempien prosessimallien raskaudelle ja jäykkyydelle. Ketterät menetelmät, kuten Scrum ja XP, on kehitetty vastaamaan nopeasti muuttuvan ympäristön vaatimuksiin ja tuottamaan ohjelmistoja, jotka antavat nykyistä enemmän lisäarvoa käyttäjälle ja asiakkaalle. Ketterien menetelmien prosessimalli on kevyt, ja vaatimusmäärittelyä sekä suunnittelua tehdään paloissa projektin edetessä. Kaikki vaatimukset huomioon ottavaa suunnittelua vältetään, koska jokaista vaatimusta ei voida määrittää tarkasti ennen projektin alkua. Ajan käyttäminen kaikkien vaatimusten kartoittamiseen projektin alussa lisää kustannuksia, koska osa vaatimuksista ehtii muuttua projektin aikana ja osasta määrittelyä tulee tarpeettomia toimintaympäristön muuttuessa. Yhteistä ketterille menetelmille on se, että toteutusta tehdään kahdesta viikosta kahteen kuukauteen kestävässä iteraatioissa ja ohjelma valmistuu vaiheittain osissa. Jokaisen iteraation jälkeen kehitystiimi esittää sidosryhmille toimivan osan ohjelmistoa.

Toinen 1990-luvulla alkanut prosessikehityksen suunta keskittyy ohjelmiston loppukäyttäjiin sekä heidän tarpeidensa selvittämiseen ja ymmärtämiseen. Käyttäjakeskeisissä suunnitteluprosesseissa huomioidaan käyttäjän tarpeet, osaaminen sekä rajoitteet, jotta niitä voidaan hyödyntää ohjelmistoa suunniteltaessa. Niissä otetaan huomioon käyttäjän käyttäytyminen ja työtehtävät, jotta ohjelma voidaan integroida osaksi käyttäjän työtehtäviä. Tarkoitus on, että käyttäjän ei tarvitse opetella uutta työskentelytapaa, vaan käyttäjän työnkulut ovat joustavia ja tehokkaita ilman tekniikan tuomia haasteita. Käyttäjakeskeiset suunnitteluprosessit painottavat kattavaa käyttöliittymäsuunnittelua käyttäjähaastatteluineen, prototyyppeineen ja testauksineen ennen toteutusvaiheen alkua. Näin voidaan olla varmoja siitä, että toteutettava käyttöliittymä on paras mahdollinen käyttäjälle. Koko tuotteen käyttöliittymät on testattava kokonaisuudessaan ennen toteutusta, jotta voidaan varmistaa ohjelmiston johdonmukaisuus ja yhtenäisyys.

Ohjelmistokehityksen ja käyttäjakeskeisen suunnitteluprosessin yhdistäminen on noussut tärkeäksi aiheeksi, koska asiakkaat osaavat jo vaatia tuotteelta käytettävyyttä. Ketterän kehityksen avulla tuotteet saadaan nopeasti markkinoille. Tuotteiden täytyy olla kuitenkin hyödyllisiä ja käytettäviä, jotta ne myisivät. Käyttäjakeskeisen suunnitteluprosessin yhdistäminen ketteriin menetelmiin varmistaa hyvän käytettävyyden ohjelmiston elinkaaren alusta lähtien.

Ketterien menetelmien ja käyttäjakeskeisen suunnittelun yhdistämisen haasteet löytyvät prosessien tavassa kerätä vaatimuksia ja ajoittaa suunnittelu. Toisaalta prosesseista on havaittavissa yhtäläisyyksiä, jotka helpottavat yhdistämistä. Yhtäläisyyksiä ovat esimerkiksi molempien prosessien iteratiivinen luonne ja asiakkaan tai käyttäjän mukaan ottaminen prosessin eri vaiheisiin. Myös molempien prosessien tavoitteet ovat lähellä toisiaan: molemmissa pyritään tuottamaan asiakkaalle lisäarvoa ja kehittämään ohjelmisto aitoon tarpeeseen. Näiden yhteisten tavoitteiden ansiosta käytettävyyden huomioiminen ohjelmistokehityksessä on mahdollista. Scrumin tavoitteena on tuottaa lisäarvoa asiakkaalle eli toteuttaa sellaiset ominaisuudet, joista on aidosti hyötyä. Käyttäjakeskeisen suunnittelun tavoitteena on tehdä käyttäjätasvällinen ohjelma, jossa on huomioidaan myös loppukäyttäjien tarpeet. Molemmat prosessit tavoittelevat lopputulokseen tuotetta, joka tuottaa lisäarvoa käyttäjälle ja nopeuttaa käyttäjän työskentelyä.

Tässä tutkielmassa kehitetään Scrum-prosessimallia siten, että siihen liitetään käyttäjakeskeisen suunnittelun vaiheita tukemaan kokonaisvaltaista ohjelmistokehitystä. Käytettävyys huomioidaan prosessin alusta lähtien, jotta jatkuvasti tiedetään ohjelmiston käytettävyyden taso. Kirjallisista lähteistä kerätään kokemuksia, ongelmakohtia ja ratkaisuehdotuksia tutkielman tavoitetta kehitettäessä. Kirjallisten lähteiden perusteella muodostetaan muokattu Scrum-prosesimalli, joka tukee käyttäjakeskeistä suunnittelua. Tutkielman lopputuloksena esitetään muokattu käyttäjälähtöinen Scrum-prosessimalli, joka huomioi käyttäjakeskeisen suunnittelun läpi koko projektin. Muokattua käyttäjälähtöistä Scrum-prosessimallia käyttämällä voi projektin alusta lähtien varmistua ohjelmiston käytettävyyden laadukkuudesta ja tehostaa Scrumin toteutusvaihetta, koska suunnittelussa huomioidaan tarkemmin käyttäjät ja heidän tarpeensa. Käyttäjälähtöinen Scrum myös lisää tietoutta käytettävyydestä projektin sidosryhmille.

Tutkielma on jaettu lukuihin seuraavasti. Luvussa 2 tutustutaan käytettävyyden määritelmiin ja sen erilaisiin arviointimenetelmiin, joilla voidaan mitata ohjelmiston käytettävyyttä. Luku 3 esittelee käyttöliittymäsuunnittelijan työnkuvaan, käyttäjakeskeiseen suunnitteluprosessiin ja prosessin käytäntöjä. Luvussa 4 esitellään ketterät menetelmät yleisesti, perehdytään Scrum-prosessimallin ideologiaan ja periaatteisiin, ja käydään läpi XP-prosessimalliin liittyviä käytäntöjä, joilla voidaan tehostaa kehitystä Scrumissa. Luvussa 5 lähestytään kirjallisuuden kautta haasteita ja onnistumisia Scrumin ja käyttäjakeskeisen suunnittelun yhdistämisessä. Luvussa 6

käydään läpi Scrumin ja käyttäjakeskeisyyden yhdistävää prosessimallia ja parannuskeinoja prosessiin sekä rajoitteita, jotka asettavat vaatimuksia prosessimallin käytölle. Luku 7 on yhteenvedo.

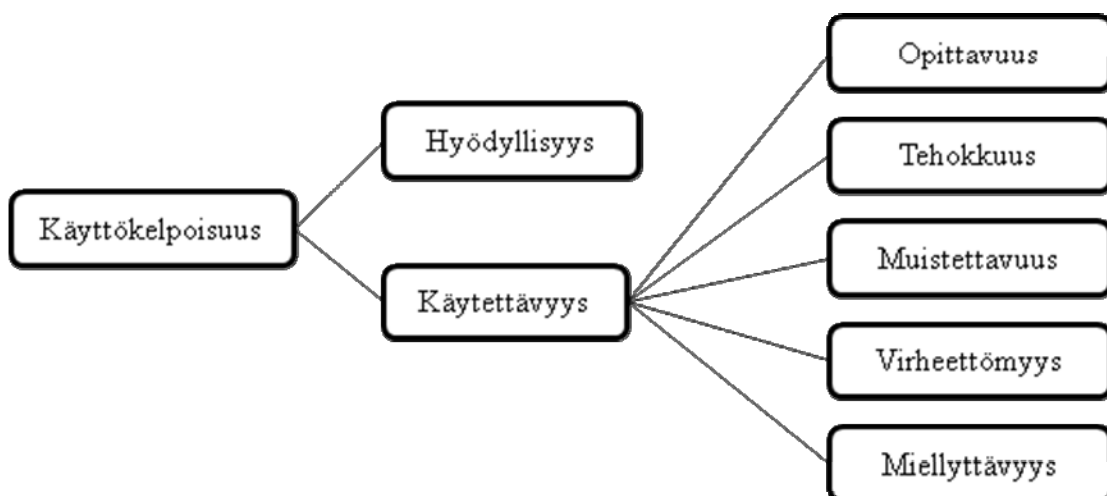
2 Käytettävyys ja käytettävyyden arviointi

Käytettävyys on eräs laadullinen mittari ohjelmistokehityksessä. Ohjelmiston erilaiset sidosryhmät vaativat erilaisia asioita käytettävyydeltä. Ohjelmiston käyttäjän kannalta käytettävyys mittaa tyytyväisyyttä ja tuottavuutta [SeM04]. Käytettävyys on onnistunut, jos käyttäjä pystyy täyttämään tarpeensa ohjelmiston avulla [Kuj98]. Itse ohjelmiston kannalta käytettävyys varmistaa sen, että tehtävien suoritus on tehokasta. Ohjelmiston ostaja käyttää käytettävyyttä yhtenä oston kriteerinä [SeM04]. Hyvä käytettävyys pienentää asiakkaan kustannuksia, koska työntekijät pystyvät työskentelemään tehokkaammin [Kan03] ja ovat tyytyväisempiä työhönsä. Käytettävyyttä on haastavaa mitata, koska se voidaan määritellä eri tavoin. Lisäksi käytettävyys on monimutkaista ja kontekstisidonnaista [KaK03].

2.1 Käytettävyyden määritelmä

2.1.1 Nielsenin käytettävyys

Jakob Nielsen jakaa ohjelmiston *käyttökelpoisuuden* (usefulness) *hyödyllisyyteen* (utility) ja *käytävyyteen* (usability) [Nie93b, s. 25]. Käyttökelpoisuus tarkoittaa sitä, että ohjelmiston avulla pystytään saavuttamaan haluttuja tavoitteita. Hyödyllisyys taas kertoo sen, voiko ohjelmiston toiminnallisuudella tehdä sitä, mitä on tarkoitus. Nielsen määrittelee käytettävyyden käyttäjien onnistumisen kautta. Ohjelmistojen toiminnallisuuksien täytyy olla niin helppokäyttöisiä, että käyttäjä onnistuu suorittamaan toiminnallisuuksien vaatimat toiminnot. Nielsen on määritellyt käytettävyydelle viisi piirrettä (kuva 1) [Nie93b]: opittavuus, tehokkuus, muistettavuus, virheettömyys ja miellyttävyys. Käyttöliittymän tulee toteuttaa näiden kaikkien piirteiden vaatimat määritelmät, jotta se olisi käytettävä.



Kuva 1: Käytettävyyden piirteet [Nie93b].

Jos käyttöliittymässä *opittavuus* (easy to learn) on kunnossa, käyttäjän on helppo oppia ohjelmiston käyttö ja käyttöliittymä tuntuu intuitiiviselta. Silloin käyttöliittymä ohjaa uutta käyttäjää käyttämään toimintoja oikein. Opittavuus vaihtelee eritasoisilla käyttäjillä, joten loppukäyttäjien osaamistaso on selvitettävä ennen ohjelmiston toteutusta. Hyvän opittavuuden takaamiseksi ohjelmiston eri toiminnallisuuksissa kannattaa käyttää samankaltaista logiikkaa, esimerkiksi hae-toiminnon on hyvä toimia johdonmukaisesti ohjelmiston eri osissa. Käytettävyyden piirteistä opittavuutta on helpoin mitata [Nie93b]. Tarvitaan käyttäjiä, jotka eivät ole ohjelmistoa ennen käyttäneet, mutta kuuluvat kuitenkin ohjelmiston kohdekäyttäjryhmään. Mittaus tehdään ottamalla aikaa siitä hetkestä, kun käyttäjät ensimmäistä kertaa käyttävät ohjelmistoa, siihen hetkeen, kun käyttäjät osaavat tehokkaasti käyttää ohjelmistoa.

Ohjelmiston käytön *tehokkuus* (efficient to use) kertoo ohjelmistoa pidemmän aikaa käyttäneen käyttäjän taidosta suorittaa tietty toimintosarja haluamansa tavoitteen saavuttamiseksi. Tehokkuus vaihtelee käyttäjien tason mukaan. Esimerkiksi ohjelmistoa päivittäin käyttävä henkilö osaa suorittaa tehtäviä ohjelmistolla nopeammin kuin satunnainen tai aloitteleva käyttäjä. Käyttöliittymän tehokkuutta mitataan painallusten määränä jonkin toiminnallisuuden läpikäymisessä. Mittauksessa pieni määrä painalluksia toimintosarjan läpiviemiseksi viittaa hyvään tehokkuuteen. Tehokkuutta mitattaessa käyttäjän tulee aina olla testattavan ohjelmiston vakituinen käyttäjä [Nie93b]. Tehokkuuteen liittyvät myös näyttöjen välillä siirtyminen ja käyttäjän muistikapasiteetin kuormittuminen. Käyttöliittymä alkaa muuttua tehottomaksi, jos käyttäjä unohtaa painamiaan painikkeita ja joutuu navigoimaan tehtävässään edestakaisin.

Muistettavuuden (easy to remember) toteuttaminen käyttöliittymässä on onnistunut silloin, kun käyttäjä osaa käyttää ohjelmistoa toisella käyttökerralla tai pitkän tauon jälkeen ilman apua. Käyttäjän ei pidä joutua opettelemaan ohjelmiston käyttöä joka kerta uudestaan. Erityisesti satunnaisten käyttäjien osalta muistettavuus on tärkeää, koska heidän pitää pystyä suorittamaan haluamansa tehtävät pitkänkin tauon jälkeen ilman ongelmia. Esimerkiksi pankkiautomaatin tai junalippuautomaatin käyttö on satunnaista, ja käyttäjän pitää pystyä suoriutumaan onnistuneesti joka kerta automaattien käytössä. Muistettavuutta auttaa, että ohjelman eri osat toimivat loogisesti ja painikkeiden toiminta on selkeästi esitetty, esimerkiksi kuvien avulla.

Ohjelmiston *virheettömyys* (few errors) kertoo siitä, miten ohjelmisto reagoi virheellisiin tilanteisiin ja miten käyttäjälle ilmoitetaan niistä. Käyttöliittymä ei saa houkutella käyttäjää painamaan väärää painiketta. Jos käyttäjän toiminta silti aiheuttaa virheellisen tilanteen, käyttöliittymän pitää pystyä selkeästi kertomaan virheestä ja mahdollisuuksien mukaan palautumaan tilaan, josta käyttäjä voi jatkaa ohjelmiston käyttöä. Pahimmat virhetilanteet ovat sellai-

sia, joissa käyttäjä ei huomaa tehneensä virhettä ennen kuin toimintosarjan päätyttyä, tai joissa käyttäjän syöttämä tieto häviää ohjelman virheen seurauksena.

Viimeisenä piirteenä on *miellyttävyyys* (subjectively pleasing), joka tarkoittaa käyttäjän subjektiivista mielipidettä ohjelmiston käytöstä. Ohjelmiston käytön pitää olla käyttäjän mielestä miellyttävää, jotta tämä jaksaa ja haluaa käyttää sitä. Miellyttävyyys muodostuu erilaisista asioista riippuen tuotteen käyttäjäryhmästä. Esimerkiksi vapaa-ajan tuotteissa, kuten peleissä, tunnelma ja grafiikka tuovat tyytyväisiä asiakkaita, mutta ammattikäyttöön tarkoitetuissa työvälineissä monipuolisuus ja käytön tehokkuus lisäävät tyytyväisyyttä. Tyytyväisyyteen liittyvät erityisesti käyttöliittymän ulkonäköön liittyvät asiat sekä esimerkiksi ohjelmiston käyttöoppaan laatu. Ulkoasullisesti käyttöliittymän värimaailman pitää olla sopusointuinen, eikä käyttöliittymässä saa olla turhaan vilkkuvia tai liikkuvia kuvia tai tekstejä.

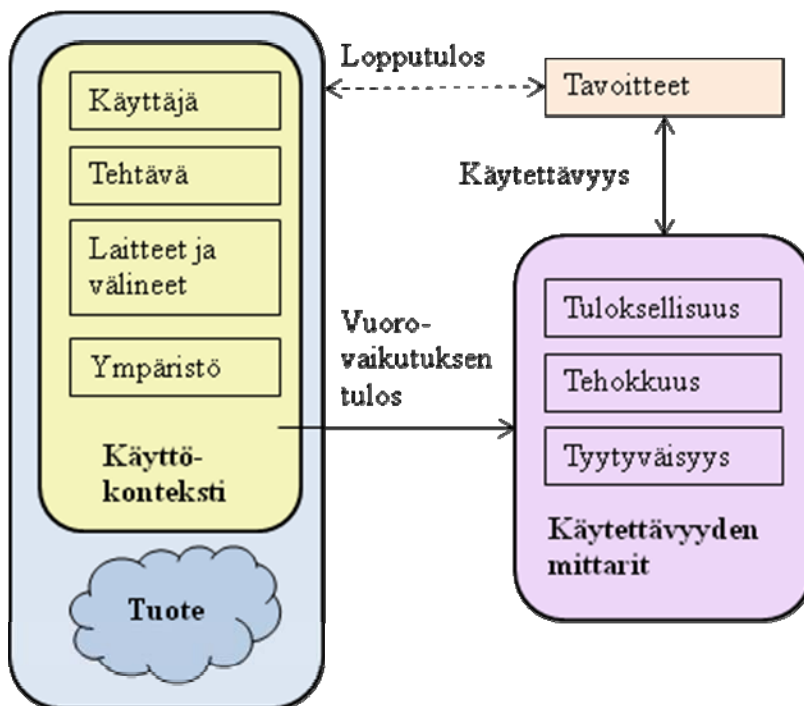
2.1.2 ISO 9241-11

Käytettävyyys on määritelty ISO 9241-11 -standardissa. Määritelmässä todetaan, että käytettävyyys mittaa, kuinka tuloksellisesti, tehokkaasti ja miellyttävästi käyttäjä saavuttaa tavoitteensa ohjelmistoa käyttäessään [ISO98]. Kuvassa 2 on esitetty ISO 9241-11 -standardin määritelmään liittyvien osien suhteet toisiinsa. Tavoitteet määritellään käyttökontekstin mukaan, ja tavoitteiden mukaan suunnitellaan käytettävyyden mittarit. Käyttökonteksti vaikuttaa mittareiden tuloksiin.

Tuotteen käyttökontekstin muodostaa se ympäristö, jossa ohjelmistoa käytetään. Käyttökonteksti muodostuu neljästä osa-alueesta: käyttäjä, tehtävä, laitteet ja ympäristö. Käyttäjistä on selvitettävä ainakin käyttäjän työtehtävät, koulutus- ja kokemustaso sekä tämän tavoitteita työn tekemiselle. Käyttäjät luokitellaan omiin ryhmiinsä ominaisuuksien perusteella. Tehtävä taas sisältää jonkin suorituspolun, jonka suorittamisen päätteeksi käyttäjä saavuttaa tavoitteensa. Tehtävät on tärkeä määritellä todellisten työtehtävien pohjalta, jotta tehtävien tulokset eivät vääristä käytettävyyttä. Laitteita osa-alueena tutkitaan arvioimalla niiden hyötyjä ja haittoja tavoitteiden saavuttamiseksi. Kaikki laitteet ja ohjelmistot, joita käyttäjä tarvitsee tuotetta käyttäessään, on kuvattava ja kirjattava muistiin. Ympäristö osa-alueena käsittää sekä fyysisen että sosiaalisen toimintaympäristön. Fyysisestä ympäristöstä kerätään tietoa esimerkiksi työtilan sisustuksesta ja tavaroiden sijainnista pöydällä ja tilassa. Sosiaaliseen työympäristöön liittyvät työkuulttuuri, asenteet ja käyttäjän työskentelytavat. Edellä mainittujen osa-alueiden tietojen yhdistämisellä saavutetaan joku haluttu tavoite ja saadaan tietoja, joiden avulla mitataan tuotteen käytettävyyttä. Nämä kaikki neljä osa-alueita pitää määritellä hyvin, jotta tuotteen ominaisuuksien käytölle voidaan asettaa tavoitteita. Tarkka määrittely on tärkeää myös siksi, koska käytettävyyys riippuu paljon käyttökontekstista. Tämän vuoksi käytettävyyys ohjelmistoille määri-

tellään eri tavoin käyttötilanteesta riippuen [Bev95, Kar97]. Esimerkiksi ulkona olevan pankki-automatin käytettävyys arvioidaan erilaisilla kriteereillä kuin tietokoneelta käytettävän verkkopankin käytettävyys.

Standardi ISO 9241 määrittelee käytettävyydelle kolme mittaustapaa: tuloksellisuus, tehokkuus ja tyytyväisyys. Nämä mittaustavat eroavat hieman Nielsenin käytettävyyden piirteistä. Opittavuutta ja muistettavuutta ei ole standardissa mainittu erikseen. Nielsenin käytettävyyden piirteistä tehokkuus, virheettius ja tyytyväisyys vastaavat lähes ISO 9241 -standardin tehokkuutta, tuloksellisuutta ja tyytyväisyyttä. ISO 9241 -standardin mukaan *tuloksellisuus* (effectiveness) mittaa sitä, miten hyvin tehtävälle annetut tavoitteet saavutetaan. Jos esimerkiksi tehtävänä on kuukauden laskujen syöttäminen järjestelmään ja tavoitteena on järjestelmässä olevien tietojen virheettömyys, niin tuloksellisuutta voidaan mitata tutkimalla järjestelmästä löytyvien virheiden määrää [ISO98]. *Tehokkuus* (efficiency) taas mittaa sitä, miten paljon resursseja tehtävän suorittaminen tarvitsee. Yleensä tehokkuudella halutaan mitata käytettyä työaikaa ja kuluja. *Tyytyväisyys* (satisfaction) mittaa käyttäjien kokemaa tunnetta ohjelmiston käytöstä. Korkea tyytyväisyys kertoo siitä, että käyttäjien asenteet ohjelmistoa kohtaan ovat positiivisia ja käyttökokemus on miellyttävä.



Kuva 2: ISO 9241-11 -standardin määritelmä käytettävyydelle [ISO98].

Käytettävyyden mittaustavat standardissa on esitetty yleisluontoisesti. Ohjelmistoprojekteissa jokaisesta mittaustavasta on valittava ainakin yksi mittari käytettävyyden arviointiin. Käytettävyttä arvioitaessa mittarit täytyy valita eri ohjelmistoille erikseen, sillä mittarit valinta riippuu

ohjelmiston luonteesta. Näin voidaan varmistua siitä, että mitataan todellista käytettävyyttä. Myös testattavat käyttötilanteet ja käyttäjät vaikuttavat mittareiden valintaan. Tärkeää on asettaa tavoitteita ohjelmiston käytölle ja niiden perusteella valita sopivia mittareita [ISO98, Nie98].

2.2 Käytettävyyden arviointi

Eri arviointitavat arvioivat käyttöliittymää eri näkökulmista, joten monet arviointimenetelmät täydentävät toisiaan. Yksistään käytettäessä erilaiset arviointimenetelmät löytävät vain tietyn-tyyppisiä käytettävyysoongelmia. Eräs tapa luokitella arviointimenetelmät on jakaa ne asiantuntijamenetelmiin sekä testimenetelmiin, joissa tarvitaan loppukäyttäjiä (taulukko 1) [Hol05].

Taulukko 1: Käytettävyyden arviointimenetelmiä [Hol05].

	Asiantuntijamenetelmät			Testimenetelmät		
	Heuristinen arviointi	Kognitiivinen läpikäynti	Toiminto-analyysi	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 arvioijat	3+	3+	1-2	1	1+	1
Vaaditut työvälineet	Vähän	Vähän	Vähän	Paljon	Kohtalaisesti	Vähän
Vaadittu ammattitaito	Kohtalainen	Korkea	Korkea	Kohtalainen	Korkea	Matala
Tunkeilevuus	Ei	Ei	Ei	Kyllä	Kyllä	Ei

Asiantuntijamenetelmissä käytettävyyden asiantuntijat testaavat käyttöliittymän käytettävyyttä [Rii98]. Nämä menetelmät edellyttävät asiantuntijalta hyvää ammattitaitoa, jotta käytettävyysongelmat löytyvät. Asiantuntijamenetelmiä ovat esimerkiksi heuristinen arviointi, kognitiiv-

vinen läpikäynti sekä toimintanalyysi. Pelkästään asiantuntijamenetelmien avulla ei voida varmistua siitä, onko käytettävyys oikeasti hyvä lopullisessa tuotteessa [Bar08], vaan tueksi tarvitaan testimenetelmiä.

Testimenetelmissä käyttöliittymää testataan loppukäyttäjien avulla. Näissä menetelmissä käyttäjä simuloi käyttöliittymää paperiversiolla tai suorittaa varsinaisella ohjelmistolla oikeita työtehtäviään käyttöliittymäsuunnittelijan kirjatessa muistiin käyttäjän kokemia ongelmia ja onnistumisia. Parhaimman lopputuloksen ohjelmiston käytettävyyden kannalta saa, kun käyttöliittymää testataan arviointimenetelmillä, jotka kuuluvat molempiin ryhmiin.

2.2.1 Asiantuntijamenetelmät

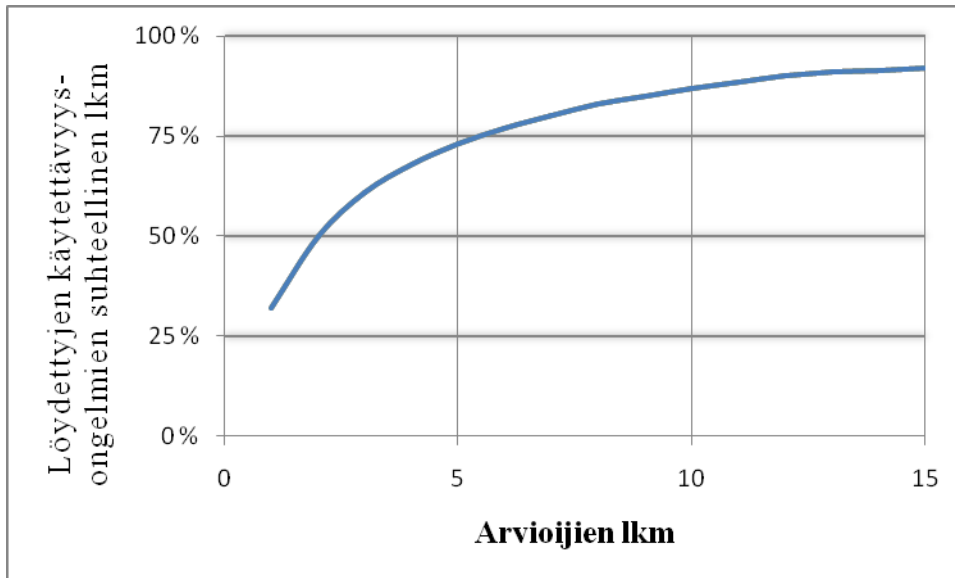
Nielsenin [NiM90, Nie93b, Nie92b] kehittämä *heuristinen arviointi* (heuristic evaluation) on järjestelmällinen käyttöliittymän käytettävyyden tarkastus, joka perustuu asiantuntija-arviointiin. Siinä käyttöliittymäsuunnittelija katsoo läpi käyttöliittymäkuvia ja vertaa niitä käytettävyyssperiaatteisiin, joita Nielsenin heuristisessa arvioinnissa on 10 kappaletta. Taulukossa 2 on listattu Nielsenin käytettävyyssperiaatteet.

Taulukko 2: Lista Nielsenin kymmenestä käytettävyyssheuristiikasta [Nie93b, Nie94, Nie02].

Nielsenin kymmenen heuristiikkaa		
1.	Järjestelmän tilan näkyvyys	Käyttäjän tulee jatkuvasti olla tietoinen ohjelmiston tilasta.
2.	Käyttäjien kielen käyttäminen	Järjestelmän tulee käyttää käyttäjien puhumaa kieltä.
3.	Käyttäjä hallitsee tekemisiään	Käyttäjän pitää pystyä poistumaan mistä ohjelmiston tilasta tahansa.
4.	Yhdenmukaisuus ja standardit	Ohjelman osissa pitää olla johdonmukainen sanasto ja ulkoasu.
5.	Virheiden ehkäisy	Käyttäjän virhetilanteita tulee välttää etukäteen.
6.	Tunnistaminen	Käyttäjän muistitarpeen minimoiminen, toiminnot ja valinnat on tehtävä näkyviksi.
7.	Joustavuus ja tehokkuus	Käyttäjät saavat itse muokata tarvitsemiaan toimintoja ja pikanäppäimet aktiivisille käyttäjille.
8.	Esteettisyys ja minimalistinen suunnittelu	Käyttäjän näkyviltä pitää ottaa turha tieto pois.
9.	Virheistä toipuminen	Virheilmoitusten pitää olla selkeitä ja sisältää ratkaisuehdotuksia ongelmaan.
10.	Opasteet ja käyttöohjeet	Opasteet pitää olla helposti löydettävissä ja sisältää konkreettisia ohjeita tehtävien suorittamiseen.

Kuvassa 3 verrataan heuristisen arvioinnin arvioijien lukumäärää löydettyjen ongelmien suhteellisen määrään. Viidenteen käyttöliittymäsuunnittelijaan asti löydettyjen ongelmien suhteellinen määrä kasvaa nopeasti, mutta sen jälkeen vauhti hidastuu. Nielsen [Nie93b] ehdottaa

heuristista arviointia tekevien käytettävyyssiantuntijoiden määräksi 3–5 henkilöä. Heuristista arviointia voivat tehdä yksittäiset asiantuntijat, mutta Nielsenin [Nie93b] mukaan silloin ei löydetä riittävän suurta osaa käytettävyysongelmista.



Kuva 3: Heuristisella arvioinnilla löydettyjen käytettävyysongelmien suhteellinen osuus verrattuna arvioijien lukumäärään [Nie93b].

Heuristisen arvioinnin hyviä puolia ovat sen nopeus ja helppo toteutus. Heuristinen arviointi on edullinen ja yleinen tapa testata käytettävyyttä, koska arviointia voi tehdä myös ohjelmistokehittäjät [JMW91, Kan03, RRH00, VMS02]. Arvioinnin heikkous on se, että siinä ei tiedetä käyttäjien oikeita käyttötarpeita, koska käyttöliittymää verrataan vain käytettävyyssperiaatteisiin. Heuristisessa arvioinnissa käyttöliittymää arvioidaan kuva kerrallaan, jolloin koko ohjelmiston käytettävyys ei kokonaisuutena välttämättä ole yhtenäinen. Heikkoutena on myös se, että arvioijia tarvitaan useampia, jotta saadaan riittävän luotettava tulos [JMW91].

Kognitiivisessa läpikäynnissä (cognitive walkthrough) käyttöliittymäsuunnittelija jäljittelee ohjelmaa tai sen prototyyppiä käyttäessään käyttäjän todellista työnkulkua. Asiantuntija on joko tutkinut käyttäjän todellisia työnkulkuja tai suunnittelee mahdollisia työnkulkuja ja tekee sen perusteella testitapaukset ja testaa niillä käyttöliittymää. Vaarana menetelmässä on, että jos todelliset työnkulut arvioidaan vääriksi, virheellisiä käytettävyysongelmia löytyy ja todelliset ongelmat jäävät piiloon. Kognitiivinen läpikäynti keskittyy erityisesti opittavuuden parantamiseen [Hol05]. Menetelmän avulla käyttöliittymäsuunnittelija pystyy määrittelemään käyttäjän tavoitteita [JMW91]. Kognitiivinen läpikäynti vaatii perehtyneisyyttä käytettävyyden testaukseen sekä testattavan ohjelmiston toimialaan, jotta testitulokset eivät löydä virheellisiä havaintoja. Kognitiivinen läpikäynti koetaan pitkästyttäväksi ja liian yksityiskohtaiseksi [JMW91].

Toimintoanalyysissa (activity analysis) tehtävien suoritusta mitataan painiketasolla. Käyttöliittymän testauksessa lasketaan kaikki painikkeiden painallukset ja mitataan suoritukseen kulunut aika. Näin saadaan selville, miten pitkään tietyn tehtävän suorittaminen kestää ja montako painallusta se vaatii. Toimintoanalyysin tekeminen vaatii työtehtävien selvittämistä ja testaaminen vie paljon aikaa. Toimintoanalyysissa ja kognitiivisessa läpikäynnissä on samoja ongelmia, kuten se että asiantuntija miettii työnkuluja ilman käyttäjää.

2.2.2 Testimenetelmät

Ääneen ajattelu -menetelmässä (think aloud) käyttäjä suorittaa tehtäviä prototyypillä tai ohjelmistolla ja kertoo jatkuvasti ääneen, mitä on ajattelemassa ja tekemässä [Nie92a]. Tällä tavoin löydetään mahdolliset virhetilanteet ja puutteelliset ohjeistukset. Asiantuntija seuraa ja kirjaa muistiin käyttäjän kommentit, mutta ei puutu tehtävien suoritukseen eikä pyri vaikuttamaan käyttäjän toimintaan. Metodien huonoja puolia ovat, että se häiritsee käyttäjän luonnollista työskentelyä ja vaatii paljon aikaa ja valmistautumista.

Käyttäjien tarkkailu (user observation) ei häiritse loppukäyttäjän työskentelyä läheskään niin paljon kuin ääneen ajattelu. Käyttäjien tarkkailu antaa tietoa ohjelmiston todellisesta hyödyllisyydestä ja käytettävyydestä [WRH02]. Siinä käyttäjä työskentelee tavallisesti omalla työpisteellään päivittäisten työtehtäviensä parissa ja asiantuntija seuraa työskentelyä mahdollisimman huomaamattomasti. Asiantuntija kirjaa muistiin käyttäjän toiminnan ja kuvaa sekä sanallisesti että kameralla esimerkiksi työpisteen sisustuksen ja tavaroiden järjestyksen. Videointia voidaan käyttää tarkkailun tukena, jolloin tarkkailutilanteeseen on helppo palata myöhemmin eikä kaikkea tarvitse tarkkailun aikana kirjoittaa muistiin. Käyttäjien tarkkailua voidaan tehdä joko aivan ohjelmistokehitysprojektin alussa, jolloin sen avulla pystytään selvittämään ja keräämään tietoa ja vaatimuksia uudelle ohjelmistolle, tai sitä voidaan käyttää toimivan ohjelmiston käytettävyyss-testauksessa [WRH02]. Huonot puolet käyttäjän tarkkailussa ovat sen pitkä kesto ja käyttäjien aikataulujen sovittaminen projektiin. Hyvä puoli on taas se, että käyttäjien tarkkailusta saadaan paljon informaatiota. Tosin sen selvittäminen ja läpikäynti vievät aikaa [WRH02].

Kysely (questionary) on yksinkertainen tapa saada käyttäjiltä tietoa ohjelmiston toimivuudesta. Se on helppo toteuttaa, koska kyselyt voidaan lähettää käyttäjille ja käyttäjät voivat vastata siihen silloin, kun heillä on aikaa. Kyselyjen hyvä puoli on se, että niiden avulla saadaan suuri määrä vastauksia koskien käyttäjien tyytyväisyyttä. Huonoja puolia kyselyssä ovat, että vastaukset ovat subjektiivisia ja kyselyistä ei voida selvittää, miksi käyttäjät olivat jotain tiettyä mieltä jostain asiasta. Hyvän kyselyn teko vaatii ammattitaitoa, jotta kysymykset ovat ymmärrettäviä ja ne mittaavat haluttuja asioita ohjelmistosta.

2.3 Käytettävyys laadun mittarina

Käytettävyys on yksi laatuattribuuteista, jotka pitää ottaa huomioon ohjelmistoa suunniteltaessa. Muita laatuattributteja on esimerkiksi suorituskyky ja ohjelmiston luotettavuus [BoJ03, JMS07]. Käytettävyys on mahdollista suunnitella ohjelmistoon jo projektin alkuvaiheessa samoin kuin muiden laatuattribuuttien vaatimukset [Bev95]. Kuitenkin usein käytettävyys jää huomioimatta ohjelmistokehitysprojektin alkuvaiheessa, koska ajatellaan, että se voidaan helposti lisätä valmiiseen ohjelmistoon [SeM04, BoJ03]. Käyttöliittymän suunnittelu jo projektin alkuvaiheessa on tärkeää, jotta käytettävyys suunnitellaan käyttäjän näkökulmasta eikä sen täydy noudattaa ohjelmiston arkkitehtuuri- tai tietokantaratkaisuja, vaan tällöin kaikki eri ohjelmiston osat suunnitellaan siten, että ne ottavat toisensa huomioon. Käytettävyys on monimutkaista ja kontekstisidonnaista [Kar97]. Se vaatii perehtymistä sekä käytettävyyden periaatteisiin että ohjelmiston toimialaan ja hyvää suunnittelua. Perinteisesti käyttöliittymä on erotettu arkkitehtuurista omaksi osa-alueekseen, joka on suunniteltu ja toteutettu erikseen [BaJ01]. Tämä kuitenkin aiheuttaa ongelmia hyvän käytettävyyden saavuttamisessa, koska esimerkiksi osa käytettävyysratkaisuista vaatii tukea arkkitehtuurilta [BaJ01, BoJ03]. Käyttöliittymästä voidaan erottaa ulkoasu ja käyttäjävuorovaikutus [BoJ03]. Käyttäjävuorovaikutukseen sisältyy kaikki ohjelmiston ja käyttäjän välillä liikkuva tieto, ja käyttöliittymän toiminnot tarvitsevat tukea arkkitehtuurilta. Esimerkiksi ”kumoa”, ”edellinen” ja ”seuraava” ovat sellaisia toimintoja, joiden toteutus pitää ottaa huomioon aikaisessa vaiheessa projektia, koska niiden lisäys myöhäisessä vaiheessa projektia saattaa olla mahdotonta tai aiheuttaa paljon ylimääräistä työtä [BaJ01]. Ulkoasuun kuuluvat esimerkiksi painikkeet, värit ja toimintojen sijoittelu. Ne voidaan toteuttaa arkkitehtuurista riippumatta. Käytettävyydessä on voimassa samat lainalaisuudet kuin muissakin ohjelmistoon liittyvissä vaatimuksissa [BaJ01]. Käytettävyysvirheet, jotka löytyvät myöhäisessä vaiheessa projektia, ovat kalliit korjata. Mitä aikaisemmin virheet löytyvät, sitä nopeammin ja edullisemmin ne saa korjattua.

Pelkkä käyttöliittymäsuunnittelijoiden mukana olo projektissa ei riitä takaamaan onnistunutta käytettävyyttä, vaan lisäksi käytettävyys on otettava huomioon suunnitelmallisesti. Usein ohjelmiston prototyyppiä tai toimivaa ohjelmistoa testataan käyttäjillä liian myöhäisessä vaiheessa projektia. Jos käytettävyyden arviointimenetelmiä käytetään vasta ennen ohjelmiston julkaisua, testauksessa löytyneitä käytettävyysongelmia ei ehditä korjaamaan ennen julkaisua [Bev95]. Käyttäjakeskeisen suunnitteluprosessin käyttöönottoaminen projektissa parantaa ohjelmiston laatua, koska käytettävyys otetaan huomioon jokaisessa projektin vaiheessa. Lisäksi käyttäjakeskeisessä suunnitteluprosessissa on tärkeää ymmärtää, määritellä ja arvioida käytettävyyttä jo aikaisessa vaiheessa projektia [Bev05]. Käytettävyyttä tulee testata vähän kerrallaan ja aina, kun

jotain uutta on valmistunut joko prototyyppeihin tai toimivaan ohjelmistoon. Suunnitteluprosessi siis tuo käytettävyyttä näkyväksi osaksi ohjelmistokehitystä ja näin projektissa ollaan jatkuvasti tietoisia käytettävyyden tilanteesta, eikä suuria yllätyksiä ole tulossa ohjelmiston julkaisun jälkeen. Kun ohjelmisto julkaistaan, voidaan jo etukäteen olla varmoja ohjelmiston toimivuudesta ja käytettävyydestä.

3 Käyttäjakeskeinen suunnitteluprosessi

Käytettävyyden suunnittelu kehitettävälle tuotteelle on tärkeää projektin alusta lähtien. Käyttäjakeskeiselle suunnittelulle on olemassa ISO 13407 -standardi [ISO99], joka tukee lopputuloksen hyödyllisyyttä käyttäjän kannalta. Jotta tuotetta halutaan käyttää, sen täytyy olla sekä hyödyllinen että käytettävä. Hyödyllinen tuote sisältää tarvittut toiminnallisuudet, ja käytettävällä tuotteella toiminnallisuudet pystytään suorittamaan. Toiminnallisuudet kirjataan vaatimusmäärittelyyn vaatimuksiksi.

Käytettävyyssuunnittelun tarkoitus on löytää järkevät työnkulut halutuille toiminnallisuuksille. Käyttäjakeskeisen suunnittelun tehtävänä on varmistaa sen, että tuotteen eri osa-alueet toimivat samankaltaisesti ja että ulkonäkö on koko tuotteessa yhtenäinen. Kun käytettävyyssuunnittelu aloitetaan projektin alussa, asiakkaalle saadaan nopeasti ensimmäisiä paperiversioita käyttöliittymän ulkonäöstä. Näin asiakas pääsee heti projektin alussa arvioimaan tuotteen toiminnallisuksia sekä käytettävyyttä eikä ikäviä yllätyksiä pääse syntymään.

Käytettävyyssuunnittelun alussa käytettävyydelle täytyy määritellä vaatimukset, joiden perusteella käyttöliittymää pystytään testaamaan. Vaatimukset riippuvat valmiin tuotteen käyttötarkoituksesta. Ne määritetään Nielsenin käytettävyySPIirteiden tai ISO 9241-11 -standardin mukaan painottamalla tuotteen kannalta tärkeimpiä piirteitä. Käyttöliittymiä aletaan testata heti, kun paperiversiot käyttöliittymistä on saatu valmiiksi. Käyttöliittymiä voidaan testata kaikilla edellisessä luvussa mainituilla asiantuntija- ja testimenetelmillä. Käyttäjakeskeinen suunnittelu ei ole pelkästään käyttöliittymien ja käytettävyyden testausta eikä ohjelmistokehitystä [MVS05], vaan siihen liittyy paljon muutakin, kuten käyttäjien havainnointia, työkulujen selvittämistä ja käyttöliittymien vuorovaikutusten suunnittelua ja testausta.

3.1 *Käyttöliittymäsuunnittelijan määritelmä*

Ihmisen ja tietokoneen vuorovaikutuksen (human-computer interaction) ammattilaisilla on suuri kirjo erilaisia ammattinimikkeitä, kuten käyttöliittymäsuunnittelija, graafinen suunnittelija, käyttäjäkokeusasiantuntija ja vuorovaikutussuunnittelija [MGH07, KaK03]. Tässä tutkielmassa käytetään nimikettä käyttöliittymäsuunnittelija kuvaamaan suunnittelijaa, joka havainnoi käyttäjän tarpeet, muodostaa toimintamallin, suunnittelee ulkoasun käyttöliittymälle ja testaa käyttöliittymää. Käyttöliittymäsuunnittelijan työ on suunnitella ja kuvata sekalainen joukko erilaisia toimintoja loogisesti ja järkevästi käyttöliittymässä [Arm04]. Käyttöliittymäsuunnittelijoiden koulutuskirjo on laaja [KaK03]. Käyttöliittymäsuunnittelijalla voi olla sosiologian, psykologian, kognitiotieteiden tai ohjelmistotuotannon opinnot taustalla [DBB93, PLR07].

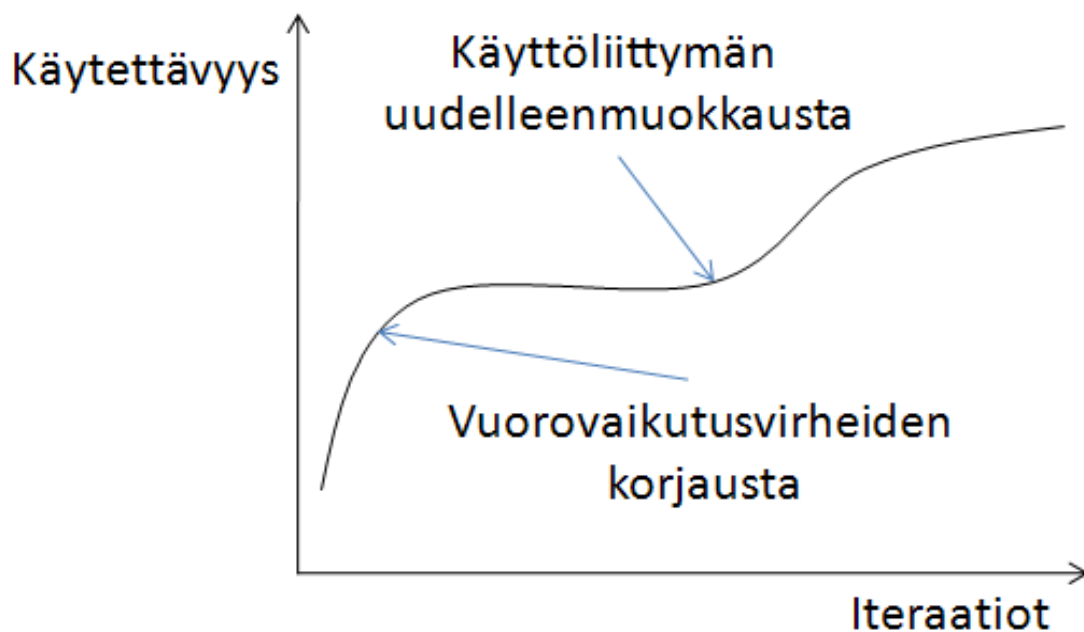
Taulukossa 3 on listattu tietoja ja taitoja, joita käyttöliittymäsuunnittelijalla täytyy olla riippumatta koulutustaustastaan.

Taulukko 3: Käyttöliittymäsuunnittelijan yleisiä tietoja ja taitoja [DBB93].

Tiedot:	
Perustiedot ja -ymmärrys ihmisen ja tietokoneen välisen vuorovaikutuksen kirjallisuudesta	Alan kirjallisuus uudistuu ripeästi, joten ajantasaisen tiedon löytäminen nopeasti ja tehokkaasti on tärkeää.
Kognitiiviset prosessit	Esimerkiksi muistin, oppimisen ja huomiokyvyn tutkimiseen liittyvien prosessien osaaminen.
Kokeellinen suunnittelu	Muuttujien hallinta, erityisesti sellaisten muuttujien hallinta, jotka eivät kuulu kokeeseen.
Nopea prototyyppien teko	Prototyyppien tekoon tarkoitettujen ohjelmistojen osaaminen ja paperiprototyyppien teko nopeasti ja tehokkaasti.
Kvantitatiiviset menetelmät	Päättelevän ja kuvailevan tilastitiikan muodostaminen tutkimustuloksista.
Taidot:	
Resurssien arviointi	Esimerkiksi suunnittelun työ- ja aika-arvioiden antaminen.
Sitoutuminen käyttäjään	Suunnittelijan täytyy osata esiintyä käyttäjän edustajana kehitysorganisaatiossa.
Ohjelmistokehitysprosessien ymmärtäminen	Käyttäjakeskeinen suunnitteluprosessi kattaa koko ohjelmistokehityksen elinkaaren.
Toteutushaasteiden ymmärtäminen	Käyttöliittymän ominaisuuksien haasteet ja kompromissit toteutuksessa.
Ominaisuudet, joita on vaikeampi saavuttaa:	
Sinnikkyys	Suunnittelijan täytyy voittaa käyttäjakeskeisen suunnittelun vastustus.
Joustavuus	Ohjelmistokehitysprosessiin täytyy saada mahtumaan erilaisia kehitystä tukevia aloja.
Empatia	Suunnittelijan täytyy osata eläytyä ja ajatella käyttäjän tavoin.
Oikea asenne	Suunnittelijan täytyy pystyä olemaan oppilas havainnoitaessa, muuttamaan suunnitelmia ja puolustamaan käyttäjää.

3.2 Käyttäjakeskeinen suunnitteluprosessi

Ihmisen ja tietokoneen välistä vuorovaikutusta (human-computer interaction) on tutkittu ja kehitetty 1980-luvulta alkaen [KaK03]. Tämän tutkimuksen evoluution aikana suunnittelun keskipiste on siirtynyt työn tuottavuuden näkökulmasta enemmän ohjelmiston käytön näkökulmaan [KaK03]. Gould [GoL85, Gou88, KaK03] määritteli 1980-luvulla hyvän suunnittelun periaatteiksi aikaisen ja jatkuvan käyttäjän huomioimisen, arvioinnin, iteratiivisen suunnittelun ja kehityksen sekä koko järjestelmän suunnittelun huomioonottamisen. Nämä periaatteet kattavat myös 1990-luvulla kehitetyn käyttäjakeskeisen suunnittelun perusmääritelmän, jonka tarkoituksena on kehittää käytettäviä ohjelmistoja [Kar97] keskittymällä tutkimaan käyttäjien tarpeita. Tarpeet muutetaan käyttöliittymäsuunnitelmaksi ja annetaan eteenpäin kehitykseen ja testaukseen [KaK03, DBB93]. Käytettävyyden suunnittelu on iteratiivinen prosessi, sillä käytettävyyden suunnittelu vaatii jatkuvaa suunnittelua ja testausta, jotta optimaalinen käytettävyys pystytään löytämään [HAH05]. Käyttöliittymän käytettävyyttä kehitetään ja parannetaan vähän kerrallaan. Kuvassa 4 näkyy käytettävyyden kehittyminen suhteessa iteraatioiden etenemiseen ja alan kulumiseen. Käytettävyyssuunnittelun iteraatioiden kesto on muutamasta tunnista päivään, joten iteraatiot on nopeita ja tiiviitä.



Kuva 4: Käytettävyyden ja suunnitteluiteraatioiden suhde [Nie93a].

Käyttäjakeskeisessä suunnittelussa pyritään ymmärtämään käyttäjän tarpeita ja tekemään niiden perusteella suunnitelmat ohjelmiston toimivuudesta [Kar97]. Käyttäjä on suunnitteluprosessin keskipisteessä ja kaikki testaus liittyy käyttäjään ja käyttäjän toimien tutkimiseen [CSM06]. Käytettävyyden luoma arvo ohjelmistolle on tärkeää huomioida käyttäjakeskeisessä suunnitte-

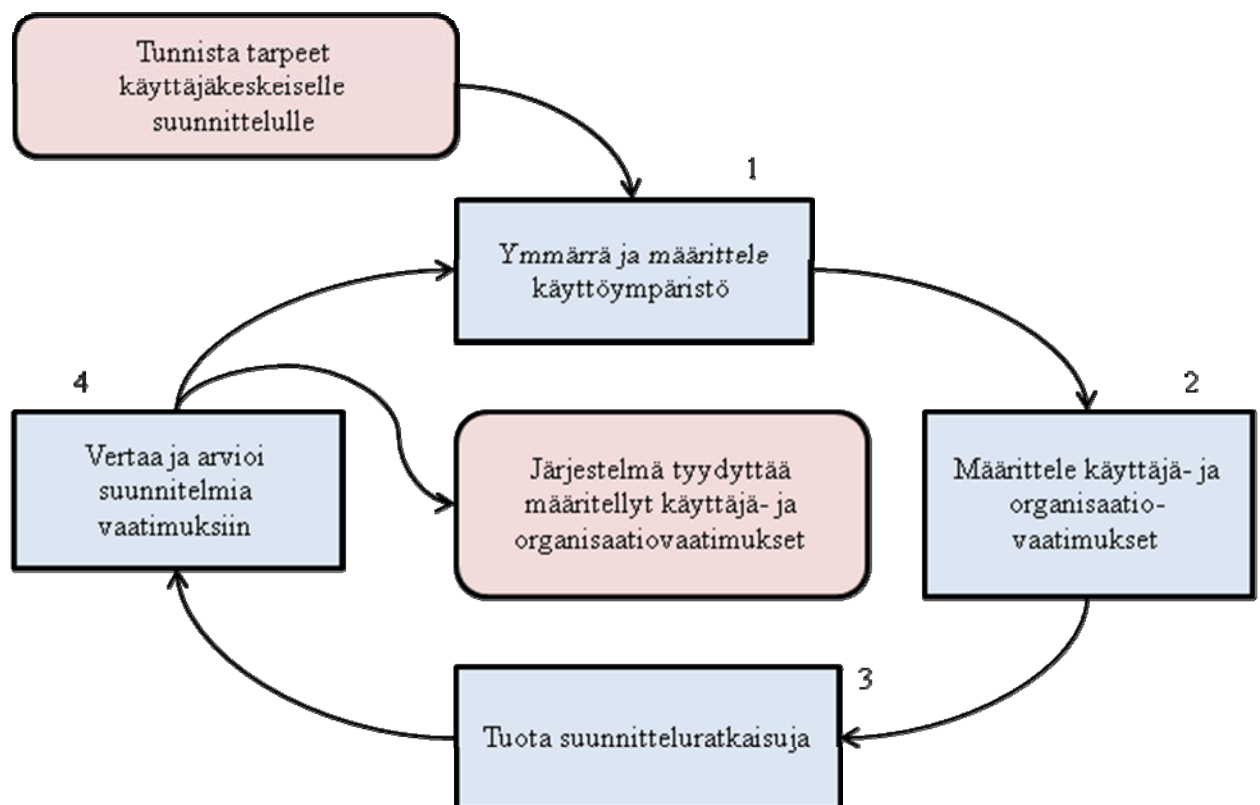
lussa ja ohjelmistojen tuottaman arvon mittaaminen tuokin uudenlaisia haasteita ohjelmistojen käyttöliittymien ja käytettävyyden suunnitteluun [KaK03].

Käytettävyyden ja oikeanlaisten toiminnallisuuksien kannalta käyttäjäkeskeinen suunnittelu (user-centered design) tuottaa toimivan ratkaisuehdotuksen. Käyttäjäkeskeisiä suunnitteluprosesseja ovat esimerkiksi Holtzblattin kontekstuaalinen suunnittelu (Contextual design) [BeH98, BeH99, WiH90] ja Cooperin tavoitepohjainen suunnittelu (Goal-Directed Design) [Coo07]. Näistä suunnitteluprosesseista lopputuloksena on yleensä kuvaus käyttöliittymästä. Tämän kuvauksen perusteella kehittäjän pitää pystyä toteuttamaan sellainen käyttöliittymä, että kuvaukseen suunniteltu toimintokulku ja käytettävyys säilyvät.

Koska käytettävyyden paremmuus ja hyvyys riippuvat käyttöympäristöstä, ne vaativat erilaiset suunnittelulähestymistavat eri käyttöympäristöissä toimiville ohjelmistoille [KaK03]. ISO 13407 -standardi [ISO99] on vuorovaikutteisten järjestelmien käyttäjäkeskeinen suunnitteluprosessi, joka ei ota kantaa käytännön työtehtäviin, vaan kuvaa prosessia yleisellä tasolla. Standardissa esitetään neljä periaatetta sekä neljä käytäntöä, jotka tukevat suunnittelua. Periaatteet kuvaavat standardissa asioita, jotka tekevät prosessista käyttäjäkeskeisen. Ensimmäinen periaate liittyy käyttäjän aktiivisesti prosessiin mukaan, jotta käyttäjältä saadaan varmasti kaikki tärkeä ja tarpeellinen tieto tarvittavista toiminnallisuuksista ja toimintosarjoista. Suunnittelijan pitää ymmärtää hyvin käyttäjä- ja toimintovaatimukset, jotta suunniteltu käyttöliittymä vastaa asiakkaan tarpeita eikä kuvaa suunnittelijan omia mielipiteitä. Toinen periaate on toiminnallisuuksien jako käyttäjien ja teknologian välillä. Tällä tarkoitetaan sitä, mitä kaikkea ohjelman toiminnoista voidaan automatisoida ja piilottaa käyttäjältä ja mitkä toiminnot välttämättä tarvitsevat käyttäjän syötettä. Käyttäjälle näytetään vain sellaista tietoa, joka on kiinnostavaa ja tarpeellista. Turhan tai tarpeettoman tiedon näkyminen käyttöliittymässä rasittaa käyttäjän muistia ja heikentää ohjelmiston tehokasta käyttöä. Kolmas periaate painottaa iteratiivista suunnittelua. Siinä ensimmäisen suunnittelukierroksen valmistuttua tuotos verifioidaan käyttäjillä, käyttäjien kerätään kommentit ja suunnitelmaa muutetaan. Jokaisen suunnitelmaan tehdyn muutoksen jälkeen kerätään käyttäjien kommentit, jotta voidaan olla jatkuvasti varmoja suunnitelman käytettävyydestä ja toimivuudesta. Viimeinen periaate peräänkuuluttaa monitaitoista suunnittelutiimiä, jotta varmasti jokainen aspekti käytettävyydestä tulee huomioitua. Tiimissä voivat olla esimerkiksi lopukäyttäjän, ostajan, käyttöliittymäsuunnittelijan, arkkitehdin sekä kouluttajan roolit. Eri roolit huolehtivat siitä, että tarvittavat vaatimukset löytyvät suunnitellusta käyttöliittymästä ja että käyttöliittymä sisältää kaikkia rooleja hyödyttäviä ominaisuuksia.

Käyttäjäkeskeisen suunnitteluprosessin neljä käytäntöä (kuva 5) ovat prosessin eri vaiheita, joita iteroidaan koko ohjelmistokehitysprojektin ajan. Ensimmäisessä vaiheessa suunnittelutiimi

opettelee ja määrittää tulevan ohjelmiston käyttöympäristön eli tutustuu käyttäjiin, työtehtäviin, organisaatioon ja käyttäjien fyysiseen työympäristöön. Käyttöympäristö on tärkeää tutkia tarkasti, jotta suunnittelussa voidaan hyödyntää käyttöympäristöstä löytyneitä ominaisuuksia. Toisessa vaiheessa määritellään käyttäjä- ja organisaatiovaatimukset, jotka liittyvät kehitettävän ohjelmiston käyttöön. Vaatimuksia kerätessä pitää ottaa huomioon ohjelmistoon toiminnallisuuksiin liittyvät lait ja muut säädökset sekä käyttäjien työtehtävät ja hyvinvointi. Kolmannessa vaiheessa suunnitellaan käyttöliittymä edellisissä vaiheissa määriteltyjen vaatimusten pohjalta. Kun ratkaisu on saatu valmiiksi, se testataan erityyppisillä testeillä, kuten asiantuntija- ja testi-menetelmillä, ja sitä muutetaan testi- ja käyttäjäpalautteiden mukaan. Pelkästään tätä vaihetta iteroidaan muutamia kertoja, kunnes suunnitelma on valmis siirtymään neljänteen vaiheeseen. Neljännessä vaiheessa aikaisemmin määritellyjä vaatimuksia verrataan syntyneeseen tuotokseen ja kerätään parannusehdotuksia tuotoksesta seuraavia iteraatioita varten. Tässä vaiheessa päätetään, toteuttiko suunnitelma sille asetetut tavoitteet [ISO99].

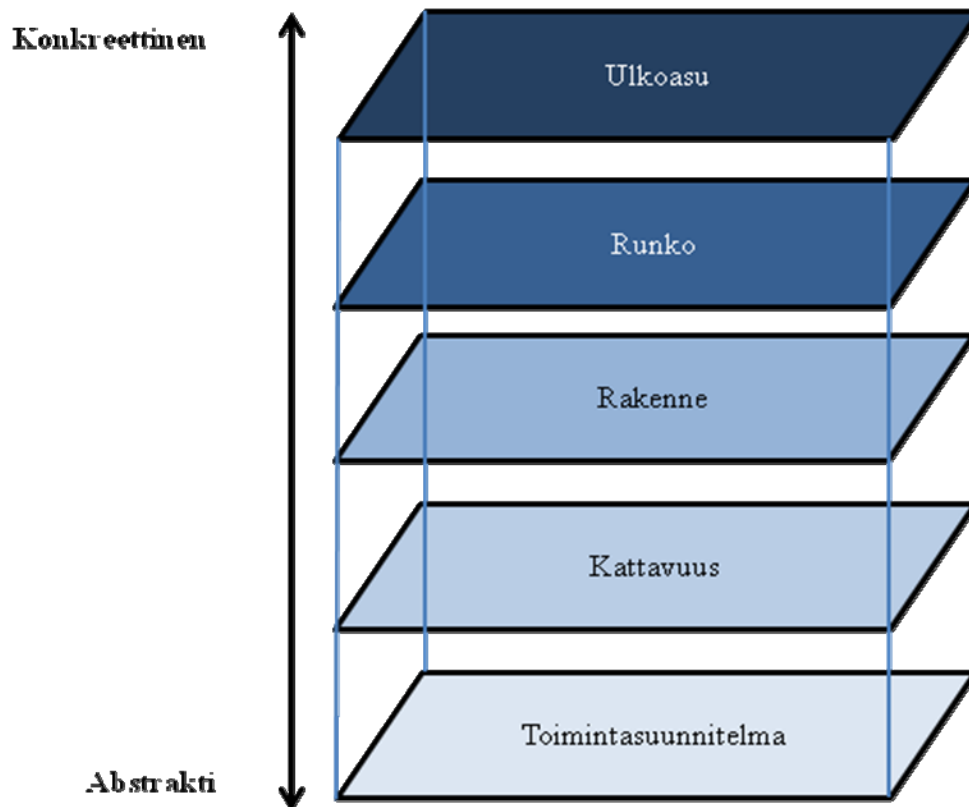


Kuva 5: Vuorovaikutteisten järjestelmien käyttäjakeskeinen suunnitteluprosessi [ISO99].

Hyviä periaatteita käyttäjakeskeiselle suunnittelulle on ottaa käyttäjät huomioon aikaisessa vaiheessa ja olla heihin yhteydessä jatkuvasti suunnittelun edetessä [Kar97]. Käyttöliittymämallien arviointi täytyy aloittaa aikaisessa vaiheessa ja jokaisen korjauskierroksen jälkeen arviointi täytyy toistaa, jotta voidaan olla jatkuvasti varmoja käyttöliittymän toimivuudesta [Kar97].

3.3 Käyttäjakeskeisen suunnitteluprosessin käytäntöjä

Käyttäjakeskeisessä suunnitteluprosessissa on erilaisia käytäntöjä eri prosessin vaiheisiin. Kuussa 6 on poikkileikkaus eri suunnittelun tasoista [Gar03]. Abstrakteimmalla tasolla ohjelmiston käyttöliittymälle määritellään toimintasuunnitelma. Siihen kuuluu esimerkiksi sellaisten käytäntöjen käyttäminen, joiden avulla määritellään käyttäjien tarpeet ja tavoitteet [Hod05] sekä ohjelmiston käyttöympäristö. Toimintasuunnitelmatasolla määritellään käytettävyyden mittarit kehitettävälle ohjelmistolle ja tunnistetaan käyttäjäroolit [Hod05]. Käyttäjäroolien tunnistamisessa kerätään tietoja ohjelmiston potentiaalisista käyttäjistä ja luokitellaan tiedot roolien mukaan. Käyttäjärooli kuvaa keskimääräistä ohjelmiston käyttäjää, eli peruskäyttäjää. Jokaiselle roolille määritellään käyttäjän koulutustausta, harrastukset, ikä ja työtehtävät.



Kuva 6: Käyttäjakeskeisen suunnittelun tasot [Gar03].

Kattavuustasolla tehdään vaatimusten kartoitusta [Hod05]. Tärkein menetelmä on oikeiden käyttäjien tutkiminen ja havainnointi [NaT08]. Siinä käyttöliittymäsuunnittelija menee käyttäjän todelliseen käyttöympäristöön seuraamaan käyttäjän päivittäisiä rutiineja. Käyttöliittymäsuunnittelijalla on mukanaan lehtiö ja nauhuri tai videokamera, joilla hän tallentaa käyttäjän työpis-

teen sekä työtehtävien kulun. Käyttäjälle on tutkimuksen alkaessa tehtävä selväksi, että käyttäjä ei tutkita vaan ainoastaan työnkuluja [Kuj98], ja että käyttäjä on tutkimuksessa asiantuntija [Kuj98]. Tutkimuksen tarkoituksena on selvittää todellinen työtehtävän kulku, jotta suunniteltavan ohjelmiston toiminnallisuudet tukevat mahdollisimman hyvin käyttäjän työtehtävän suoritusta helpoimmalla ja tehokkaimmalla tavalla. Käyttäjän tarkkailussa pyritään tutkimaan käyttäjän haluja, tarpeita ja rajoitteita [DBB93]. Tässä vaiheessa on tarkoitus olla mahdollisimman avoin erilaisille toteutusvaihtoehdoille ja pyrkiä miettimään yksinkertaisinta ja tehokkainta työskentelytapaa loppukäyttäjälle. Eri toteutusvaihtoehdoista kannattaa tehdä testattavia prototyyppisiä ja kerätä vaihtoehtojen toimivuudesta palautetta loppukäyttäjältä. Jos loppukäyttäjän työpisteelle ei päästä tekemään tutkimusta, käyttäjälle voidaan tehdä työtilannetta mahdollisimman paljon vastaava tila muualle, esimerkiksi neuvotteluhuoneeseen. Loppukäyttäjän vieminen pois todellisesta työympäristöstä kuitenkin heikentää testituloksia, koska vieraassa ympäristössä käyttäjä käyttäytyy hieman eritavalla kuin oman työpöytänsä ääressä. Jos käyttäjän työpisteelle ei ole mahdollista lainkaan päästä tutkimaan, niin loppukäyttäjien mielipiteitä voidaan kerätä ryhmähaastattelulla. Ryhmähaastattelutilanteissa käyttäjät kertovat, miten he muistavat käyttävänsä ohjelmistoa, mutta muistikuvat eivät ole yhtä luotettavia kuin varsinaisen työnteon seuraamisen yhteydessä saadut. Monesti kriittisiä toimintoja saattaa unohtua, koska käyttäjät pitävät niitä niin itsestään selvinä. Nämä itsestään selvät toiminnot tulevat esiin vain käyttäjähavainnoinnissa käyttäjän varsinaisella työpisteellä.

Kolmannessa tasossa suunnitellaan käyttöliittymän rakennetta [Hod05]. Rakennetasolla määritellään ulkoasun malli ja suunnitellaan käyttöliittymän vuorovaikutus. Käyttäjätutkimuksen jälkeen siitä saadut tulokset analysoidaan ja niiden perusteella listataan käyttöliittymävaatimuksia ja tehdään ensimmäiset karkeat käyttöliittymäkuvat esimerkiksi valkotaululle liimattaville paperilapuille. Karkeaan käyttöliittymään sisällytetään tärkeimpiä toiminnallisuuksia sekä siirtymiä toiminnallisuuksien välillä.

Runkotasolla määritetään käyttöliittymän ulkoasua jo tarkemmin. Käyttöliittymä-, navigaatio- ja informaatio suunnittelu tehdään runkotasolla [Hod05]. Käyttöliittymä siirretään valkotaululta paperille. Käyttöliittymäkuvat ja siirtymät esitetään paperiversiona loppukäyttäjälle, joka niiden avulla simuloi ohjelmiston käyttöä. Tässä vaiheessa on tärkeää, että loppukäyttäjä pääsee itse käyttämään paperiprototyyppiä ohjelmasta, sillä vain näin voidaan varmistua käyttöliittymän oikeellisuudesta. Kokeessa käyttäjälle annetaan työtehtäviä, joita he yrittävät suorittaa paperiprototyypin avulla, kun taas käyttöliittymäsuunnittelija on näkymättömissä loppukäyttäjälle eikä pyri ohjaamaan tämän toimintaa eikä auttamaan ongelmatilanteissa. Paperiprototyypin testauksessa lehtiö ja nauhuri tai videokamera ovat hyödyllisiä apuvälineitä. Paperiprototyypissä on hyvä näkyä käyttöliittymänäyttöjen yhteydet toisiinsa, jolloin suunnitelmasta nähdään, mitä

mistäkin käyttöliittymän osasta tapahtuu [MeA06]. Paperiprototyypivaiheessa käyttöliittymää iteroidaan useita kertoja ja testataan joka välissä, jotta lopputuloksena on varmasti toimiva ratkaisu [DBB93]. Paperiprototyypin testauksen jälkeen käyttöliittymäsuunnittelija käy läpi testauksessa ilmenneet ongelmakohdat, ja käyttöliittymää muokataan vastaamaan aiempaa paremmin loppukäyttäjän tarpeita. Prototyypivaiheessa käyttöliittymäsuunnitelmasta kaivataan mahdollisimman monen eri sidosryhmän edustajan kommentteja, jotta muutokset saadaan vielä helposti tehtyä ja jotta kehitysvaiheessa ei enää ilmaantuisi yllättäviä ongelmia [DBB93].

Kaikista konkreettisimmassa tasossa suunnitellaan tarkasti käyttöliittymän ulkoasu graafisesti [Hod05]. Käyttöliittymä piirretään puhtaaksi, eli siihen lisätään grafiikat ja viimeistellään toimintojen sijainnit. Sen jälkeen käyttöliittymäkuva on valmis suunnitelma ja se voidaan antaa kehitystiimille toteutettavaksi. Kehityksen jälkeen valmista ohjelmistoa tai sen osaa testaan eri arviointi- ja testimenetelmillä, joiden avulla varmistetaan, että käyttöliittymä toimii oikein ja että se on käytettävä.

3.4 Käyttäjakeskeinen suunnittelu osana ohjelmistokehitystä

Usein ajatellaan, että käyttäjakeskeinen suunnittelu tehdään ohjelmistokehitysprojektin elinkaaren ulkopuolella [SeM04]. Tutkimuksissa onkin osoitettu, että ohjelmistokehityksen tukena ei useinkaan ole minkäänlaista käyttöliittymäsuunnittelijaa [SeM04]. Kuitenkin suunnitteluprosessissa on tärkeää ymmärtää, määritellä ja arvioida käytettävyyttä jo aikaisessa vaiheessa projektia. Käyttöliittymän käytettävyyttä testataan käyttäjillä heti, kun käyttöliittymäsuunnittelijat saavat ensimmäiset paperiprototyypit valmiiksi [Bev05]. Käyttäjiä ei saa unohtaa missään vaiheessa prosessia, vaan heidät tulee pitää lähellä prosessia vaatimusmäärittelystä testaukseen [Niem98]. Tuotekehitysprosessissa voidaan varmistua tuotteen käytettävyydestä vain, jos alusta alkaen panostetaan käyttöliittymän kehitykseen ja testaukseen. Projektin alkuvaiheessa suunnitteluun käytetty aika ja resurssit maksavat itsensä takaisin projektin myöhemmässä vaiheessa, kun kalliita korjauksia ei tarvitse tehdä tuotannossa olevaan ohjelmistoon. Käytettävyys on prosessi eikä pelkästään lopputulos [Bar08]. Sen takia se täytyy ottaa huomioon koko ohjelmistokehityksen ajan.

Käyttäjakeskeisiä suunnitteluprosesseja ja käytettävyyden arviointimenetelmiä ovat kehittäneet käytettävyyden asiantuntijat omissa piireissään, kun taas ohjelmistokehitysprosesseja ovat kehittäneet ohjelmistoalan asiantuntijat [SeM04]. Molemmat prosessit huolehtivat oman kokonaisuutensa kattavasta läpikäynnistä, koska molempien alojen asiantuntijat ovat kehittäneet prosesseja ottamatta huomioon toista. Esimerkiksi ohjelmistokehitysprosesseihin on kehitetty omat keinot hallita projekteja ja suunnitella käyttöliittymiä. Tärkeää kuitenkin on, että nykyisiin ohjelmistokehitysprosesseihin saisi sisällytettyä käyttäjakeskeisen suunnitteluprosessin, jolloin saavutetaan

hyötyjä molemmille osapuolille. Jos käyttäjäkeskeinen suunnittelu on integroituna muuhun ohjelmiston suunnitteluun ja ohjelmistokehitykseen, voidaan projektissa varmistua ohjelmiston kokonaisuuden toimivuudesta [Kar97]. Käyttäjakeskeisen suunnitteluprosessin integroiminen muuhun organisaatioon ja ohjelmistokehitysprosessiin vaatii mainostamista ja opastusta eri sidosryhmille ja prosessi pitää sulauttaa muuhun toimintaan määrätietoisesti, jolloin käyttäjäkeskeisestä suunnitteluprosessista tulee erottamaton osa kaikkea muuta päivittäistä toimintaa [May99].

Ohjelmistokehityksen heikoimmat lenkit ovat käyttäjävaatimukset, käytettävyys, käyttöliittymä ja vuorovaikutuksen suunnittelu käyttöliittymään [CoL03]. Yksi syy käyttäjäkeskeisten suunnittelumenetelmien puuttumiseen ohjelmistokehitysprojekteissa on se, että menetelmät ja toimintatavat ovat tuntemattomia ohjelmistokehittäjien parissa, vähän käytettyjä, haastavia hallita ja vaikeita integroida ohjelmistokehitysprosessin osaksi [SeA03, MVS05, Sam07]. Vain muutamat ohjelmistokehittäjät ovat perehtyneet käyttäjäkeskeisiin suunnittelumenetelmiin, ja sen takia niiden arkipäiväistäminen tuo haasteita projekteissa [SeA03]. Jotta käytettävyyden suunnittelu-taito kehittyisi ja laajenisi muillekin ohjelmistokehittäjille, heidän tulee seurata ja olla mukana näkemässä käyttöliittymän suunnittelua ja sen käytettävyyden arviointia [Cas97]. Ohjelmistokehitysprojektissa ei riitä, että kehittäjät oppivat jonkun arviointimenetelmän, jos he eivät ymmärrä ja osaa hahmottaa koko käyttäjäkeskeisen suunnitteluprosessin kokonaisuutta ja tarkoitusta ohjelmistokehityksen elinkaaren eri vaiheissa [SeA03]. Käyttäjakeskeiset suunnitteluprosessit ja ohjelmistokehitysprosessit myös eroavat painopisteiltään toisistaan, joten niiden yhdistäminen on haasteellista ja vaatii molempien prosessimallien hyvää osaamista [BEH06].

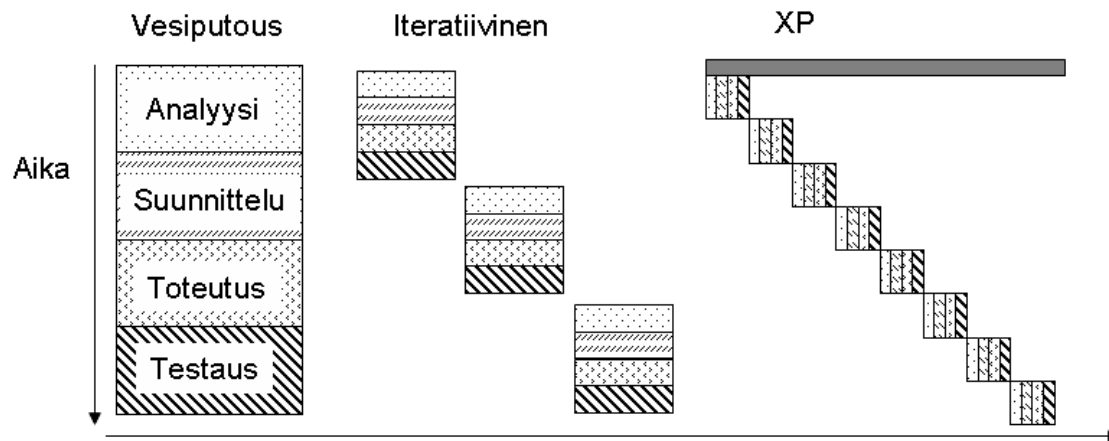
4 Scrum – ketterä ohjelmistokehitysprosessi

4.1 Ketterät menetelmät ja Scrum

4.1.1 Ketterät menetelmät

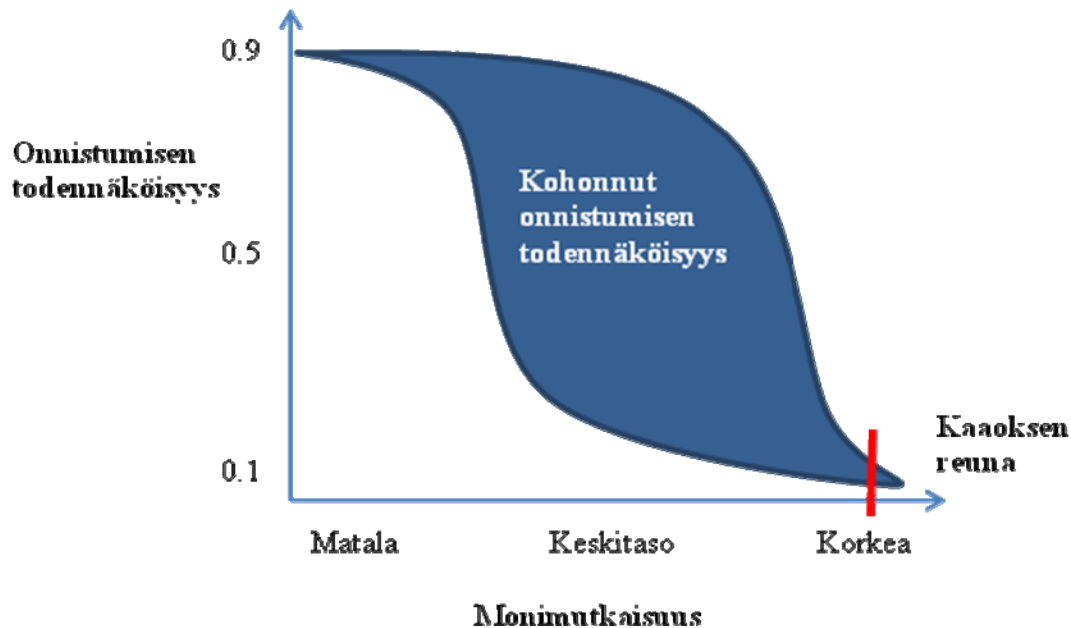
Ketterät prosessimallit ovat kehittyneet 1990-luvulla vastaamaan vaatimusten muutostarpeisiin, jotka vesiputousmallissa aiheuttivat ohjelmistokehitysprojektien epäonnistumisia. Perinteinen vesiputousmalli alkoi olla liian jäykkä, kun tuotteiden vaatimukset ja tavoitteet vaihtuivat kesken projektin markkinoiden ja asiakkaan prosessien muuttuessa. Yleisiä ongelmia ohjelmistokehitysprosesseissa ovat toistettavan ja määriteltävän ongelman, ratkaisun, kehittäjien ja organisaatioympäristön puuttuminen [BDS99]. Jokainen ohjelmistokehitysprojekti on erilainen, ja niissä on omat erityiset piirteensä, jotka vaativat erilaisia käytäntöjä ja kehitystapoja.

Kuva 7 havainnollistaa prosessimallien kehitystä. Vesiputousmallissa prosessi on jäykkä ja lineaarinen eikä jo ohitettuihin vaiheisiin palata enää myöhemmin projektin aikana [Som04]. Keskellä oleva iteratiivinen malli tuo enemmän joustavuutta prosessiin, mutta siinäkin jokainen iteraatio seuraa vesiputousmallia. Viimeinen malli kuvaa Extreme programming (XP) -prosessimallia [Bec99] ja siinä ylin vaakasuora palkki kuvaa toimintoja, joita tehdään jatkuvasti projektissa, kuten vaatimusmäärittelyä ja testausta. Lyhyissä iteraatioissa tehdään rinnakkain suunnittelua, toteutusta ja testausta eikä mitään kankeita rakenteita ole häiritsemässä, vaatimusten muuttuessa projektin aikana. XP:tä kuvaava malli sopii kuvaamaan yleisesti ketteriä menetelmiä. Iteraatiot ovat lyhyitä, ja jokaisen iteraation jälkeen tuloksena on valmiina toimiva ohjelma. Ketterät menetelmät painottavat enemmän toimivan ohjelmiston toimitusta ja ohjelmistokoodin laadukkuutta kuin perusteellista suunnittelua ja dokumentointia [PaN08]. Ne ovat keveitä prosesseja, joissa muutoksia voidaan tehdä helposti aiheuttamatta suuria lisäkustannuksia [Con01]. Ketterissä menetelmissä pyritään projektin alusta alkaen tuottamaan toimivaa ohjelmistoa ja laadukasta koodia, joita asiakas pystyy iteraatioiden jälkeen aina testaamaan [Arm04].



Kuva 7: Prosessimallien evoluutio [Bec99].

Yleensä projektin alussa on vaikea hahmottaa kaikkia vaatimuksia, eikä asiakas aina tiedä, mitä tuotteelta tarvitsee. Lisäksi eri sidosryhmillä voi olla ristiriitaiset näkemykset lopullisen tuotteen ominaisuuksista, joten vaatimusten selvittäminen ja priorisointi on haastavaa. Pitkässä projektissa vaatimukset ja lopputuotteen visio muuttuvat ympäristön muuttuessa [Arm04, RiJ00]. Kuva 8 havainnollistaa ohjelmiston monimutkaisuuden suhdetta projektin todennäköiseen onnistumiseen tavoitteissaan. Alempi käyrä kuvassa esittää perinteisten jäykästi määriteltyjen prosessien onnistumista ohjelmiston monimutkaisuuden kasvaessa. Ylempi käyrä kuvaa ketterien prosessien onnistumisen todennäköisyyttä suhteessa ohjelmiston monimutkaisuuteen. Onnistumisen todennäköisyys laskee nopeasti perinteisissä raskaissa ja joustamattomissa projekteissa, koska laajan ohjelmiston kaikkia vaatimuksia on vaikea tiedostaa projektin alussa ja vaatimusten muuttuminen kesken projektin aiheuttaa ongelmia suunnitteluun. Muuttuvat vaatimukset taas aiheuttavat hidastuksia ja haasteita projektiin, jolloin voi käydä niin, että tuote on jo julkaisuvaiheessa vanhanaikainen tai tarpeeton. Asiakas ei saa mitään arvoa tuotteesta, joka on kesken-eräinen eikä vastaa tarvetta [Arm04]. Epävarmuuden poistaminen ohjelmistokehitysprojekteista on mahdotonta, mutta ketterien menetelmien avulla sitä pystytään hallitsemaan paremmin [BDS99]. Riskejä on aina olemassa ohjelmistokehitysprojekteissa [DeL03]. Koska ketterät menetelmät pystyvät mukautumaan muuttuviin vaatimuksiin niin suunnittelun kuin toteutuksenkin osalta, todennäköisyys onnistua projektin läpiviemisessä laskee paljon hitaammin kuin perinteisissä prosesseissa. Myös riski vaatimusten osalta pienenee ketterissä projekteissa, koska niissä on sisäänrakennettuna tapa huomioida vaatimusten muuttuminen. Kun vaatimusten muuttuminen on jo huomioitu prosessissa, sen kustannukset on helpompi pitää kurissa [HiC01]. Lisäksi ketterät menetelmät pystyvät paremmin hallitsemaan kaaosta, vaikka monimutkaisuus ohjelmistossa olisi lähellä kaaoksen reunaa (edge of chaos). Kuvasta näemme, että kun tuotteen monimutkaisuus kasvaa, tarvitaan käytäntöjä, joilla pystytään kontrolloimaan riskejä sekä projektin etenemistä [Sch97].



Kuva 8: Projektin monimutkaisuuden ja onnistumisen todennäköisyyden suhde [Sch96].

Ketterissä menetelmissä ohjelmistoa kehitetään iteratiivisesti ja inkrementaalisesti. Kerralla ei pyritä saamaan koko ohjelmistoa valmiiksi, vaan ohjelmistoa julkaistaan pieninä toimivina paloina. Tärkeää on saada pieniä toimivia osia valmiiksi iteraatioiden aikana, jolloin varmistetaan siitä, että ohjelmisto todellakin kehittyy [HiC03]. Iteraation jälkeen asiakas pääsee tutustumaan ohjelmiston sen hetkiseen tilanteeseen ja voi vaikuttaa vaatimuksiin ja ohjelmiston ominaisuuksiin kesken projektin. Tällainen jatkuva palaute asiakkaalta antaa varmuutta siitä, että kehitettävä tuote vastaa asiakkaan tarpeita [HiC01]. Palautteen avulla pyritään siis estämään vääränlaisen ohjelmiston syntyminen ja edistetään tarvittavan ohjelmiston valmistumista. Erilaisia ketteriä prosessimalleja on useita, kuten Scrum, Extreme Programming (XP), Crystal, Dynamic Systems Development (DSDM), ja Lean Development [ASR02, Coc07], joista tässä tutkielmassa keskitytään Scrum-prosessimalliin ja joitain asioita poimitaan Extreme Programming -menetelmästä.

4.1.2 Ketterien menetelmien yleistyminen

Ketterät menetelmät ovat alkaneet yleistyä 2000-luvulla kaikenkokoisiin ja -tyyppisiin ohjelmistokehitysprojekteihin. Alussa ketteriä menetelmiä käyttivät vain pienet web-pohjaiset tiimit, joilla oli tarve saada nopeasti jotain julkaistua, mutta vähitellen myös isompia projekteja toteutettiin ketterien menetelmien avulla. Yritykset hakevat ketteristä prosessimalleista ratkaisua jatkuvasti muuttuvan maailman ja monimutkaisempien ohjelmistojen aiheuttamaan ongelmaan. Ketterät prosessit eivät kuitenkaan ole ratkaisu kaikkiin ohjelmistokehityksen ongelmiin ja liian

kevein perustein käytettynä aiheuttavat enemmän ongelmia ohjelmistokehitysprojekteissa kuin perinteiset jäykät prosessit. Esimerkiksi ketterät prosessit painottavat kehittäjien ammattimaisuutta ja taitavuutta, mikä aiheuttaa ensimmäiset haasteet ketterille projekteille, koska ammattilaisista koostuvaa tiimiä on vaikea löytää. Tiimien kehittyessä tiimin jäsenten ammattitaito kasvaa, mutta tämä vie aikaa ja kehittymisen ajan tiimin tuottavuus on pienempi kuin valmiin ammattilaisiin. Ketterät menetelmät ovat alun perin suunniteltu pieniin ja nopeisiin projekteihin, joissa tiimi ja asiakas työskentelevät lähellä toisiaan. Kevyiden prosessimallien skaalauksessa suuriin projekteihin sekä ulkoistettuihin tiimeihin esiintyy haasteita, kuten vuorovaikutuksen heikentyminen, kulttuurierot, ja eri aikavyöhykkeet [Sch98, ScB02]. Ketteriä prosesseja käyttävä projekti todennäköisesti onnistuu, jos kehitettävän tuotteen strategia on hyvin mietitty, kehitystiimi koostuu ammattilaisista ja se käyttää projektiin sopivia ketterien menetelmien käytäntöjä tukemaan kehitystä [ChC07].

4.2 Scrum-prosessimalli

Scrum-prosessimalli painottuu projektin- ja riskienhallintaan. Sen tarkoitus ei ole määrittellä tiukkoja toimintatapoja itse ohjelmiston kehitykseen [Sch98]. Varsinainen kehitystyö tapahtuu niin sanotun mustan laatikon sisällä, eikä sitä pyritä kontrolloimaan [Sch98], vaan kehittäjien annetaan omaa luovuuttaan käyttäen ratkaista ohjelmiston toteutusongelmia. Scrum-prosessimalli tarjoaa kehyksen ohjelmistoprojektille ja toimintatapoja projektinhallintaan. Käytännön toteutuksen tekemiseen Scrum ei tarjoa ehdotuksia, vaan jokainen Scrum-projekti ja -tiimi saa käyttää parhaiten itselleen sopivia menetelmiä. Esimerkiksi XP-prosessimallin menetelmiä, kuten parikoodausta ja refaktorointia, voidaan hyvin soveltaa Scrum-projektissa.

4.2.1 Scrumin historia

Scrumin kehitys on alkanut vuonna 1993, jolloin Jeff Sutherlandin ja Ken Schwaberin alkoivat omissa työpaikoissaan kehittää jotain parempaa ja tehokkaampaa tapaa tehdä ohjelmistoja kuin suunnitelmalähtöinen prosessi. Scrum [ScB02] on suunniteltu erityisesti hallinnolliseksi prosessimalliksi, joka tarjoaa käytäntöjä ja työkaluja prosessin seuraamiseen. Scrumissa tärkeää on se, että päätösvalta omaan tekemiseen on projektiryhmän jäsenillä. He saavat itse päättää, mitä tekevät ja milloin tekevät, ilman että muut sidosryhmät pääsevät häiritsemään heidän työskentelelyään. Scrum on suunniteltu pienille tiimeille, noin seitsemän hengen ryhmille, millä voidaan varmistaa se, että kommunikaatio tiimin sisällä on toimivaa. Ryhmäkoko on yksi keino, jolla halutaan varmistaa tiimin sisäinen kommunikaatio ja yhteistyö. Scrum-projektissa voi olla useampia Scrum-tiimejä, kunhan vain tiimikoko pysyy noin seitsemässä henkilössä.

Scrum-prosessimallissa alun perin pyrähdysten pituus on määritelty 30 päiväksi [ScB02]. Nykyään projekteissa suositaan lyhyempiä iteraatioita, joissa palaute asiakkaalta saadaan nopeammin. Projekteissa pyrähdysten pituus vaihtelee viikosta neljään viikkoon.

4.2.2 Scrum-tiimin roolit

Scrumin prosessimallissa määritetään kolme roolia: tuotteen omistaja (*product owner*), Scrum-tiimi (*Scrum team*) ja Scrum-mestari (*ScrumMaster*) [ScB02]. Tuotteen omistajalla on eniten päätösvaltaa kehitettävän ohjelmiston kannalta. Tuotteen omistaja voi olla asiakkaan edustaja tai esimerkiksi tuotepäällikkö, joka tuntee tuoteympäristön hyvin. Hänen ei tarvitse olla tekninen asiantuntija. Tuotteen omistajan tärkein vastuu on huolehtia vaatimuksista. Kaikki sidosryhmät saavat lisätä omia vaatimuksia ja kertoa mielipiteitään, mutta ainoastaan tuotteen omistajalla on oikeus priorisoida vaatimuksia. Tuotteen omistaja määrittelee tuotteelle vision. Julkaisua mietittäessä tuotteen omistajalla on suuri rooli olla määrittelemässä julkaisuun sisältyviä vaatimuksia ja seurata, että ohjelmisto kehittyy oikeaan suuntaan. Tuotteen omistajuuden voi muodostaa erillinen tiimi, mutta silloinkin yhdellä on päävastuu priorisoinnista ja tuotteen visiosta [ScB02, Sch97].

Scrum-tiimin vastuulla on toteuttaa sovitut tehtävät ja toiminnallisuudet iteraation aikana. Tiimiin kuuluvat kehittäjät, testaajat ja dokumentointi. Scrum-tiimi on moniosaajatiimi, jossa ihan-teellisessa tilanteessa kaikki tekevät kaikkea toteutuksesta testaukseen ja dokumentointiin. Scrum-tiimejä voi olla projektissa useita, jolloin jokainen tiimi toteuttaa esimerkiksi yhteen komponenttiin liittyvät vaatimukset [RiJ00]. Scrum-tiimin jäsenillä on valtaa valita tehtävät, joita tiimi haluaa tehdä seuraavan iteraation aikana. Tiimin jäseniä kannustetaan ottamaan vastuuta ja suunnittelemaan omaa työtään ja vaatimusten toteuttamista. He määrittelevät itse työ-määräarviot valitsemilleen tehtäville.

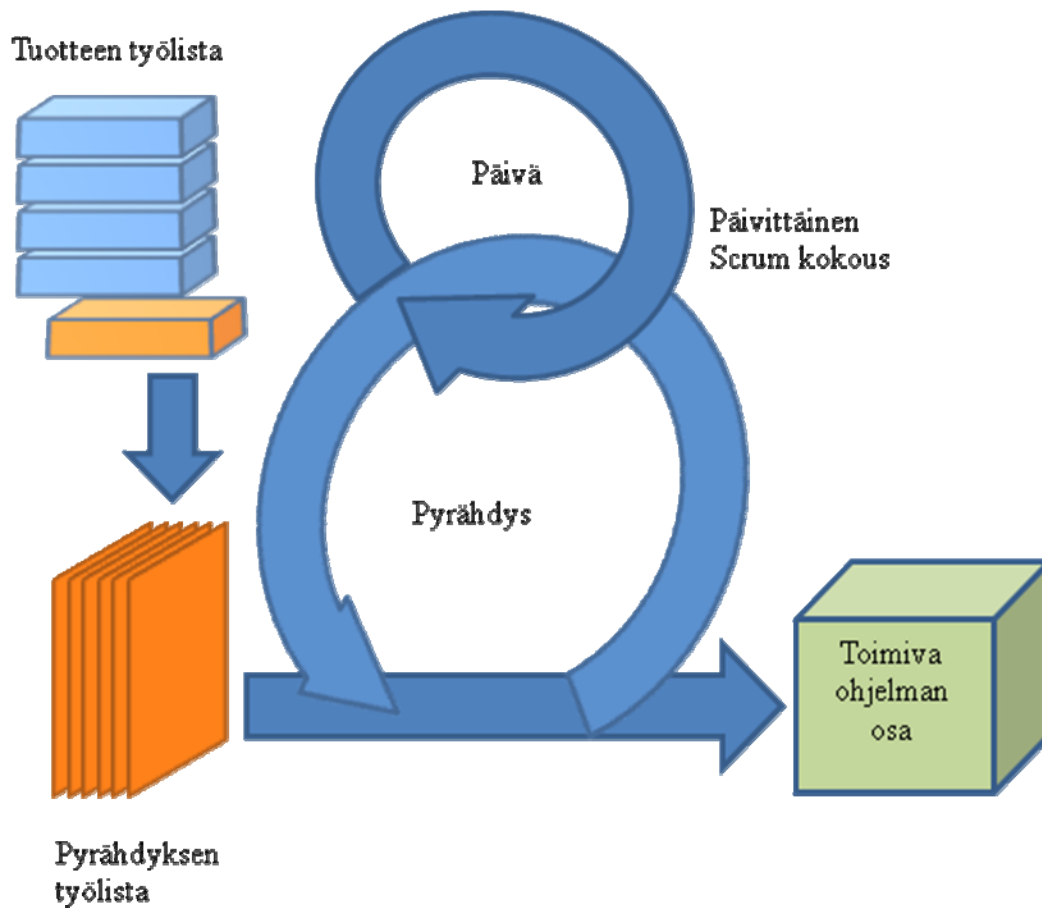
Scrum-mestari on tiimin valmentaja. Hän voi olla joku tiimin jäsenistä tai ulkopuolinen henkilö, mutta Scrum-mestari ei saa olla tiimin jäsenten esimies. Scrum-mestari pitää huolen, että projektissa noudatetaan Scrum-prosessimallin käytäntöjä ja ohjeita. Hän varmistaa, että tiimi pystyy työskentelemään esteettömästi ja häiriöttömästi iteraation aikana. Scrum-mestarin vastuualueeseen kuuluvat teknisten ja muiden esteiden poistaminen. Hän myös valvoo, että tiimit noudattavat prosessin käytäntöjä. Scrum-mestari on päivittäin yhteydessä Scrum-tiimin kanssa ja pystyy näin seuraamaan ja auttamaan tiimiä tehtävien teossa. Scrum-mestarin tehtävässä on tärkeää, että tehtävään valitulla on valtaa tehdä nopeita päätöksiä itsenäisesti.

4.2.3 Prosessin kuvaus

Pyrähdysten (sprint) syöttö- ja tulostiedot on tarkoin määritelty. Näin pystytään kontrolloimaan pyrähdyksissä tapahtuvaa kehitystä. Pyrähdystä käsitellään mustana laatikkona, jota seurataan ulkopuolelta erilaisin mittarein, kuten tuntiarvioiden kehittymisenä pyrähdysten kuluessa. Pyrähdysten aikana tiimin jäsenet saavat rauhassa tehdä pyrähdykseen määriteltyjä vaatimuksia, eivätkä tiimin ulkopuoliset sidosryhmät pysty vaikuttamaan käynnissä olevaan pyrähdysten sisältöön [BDS99]. Muutoksia voi ilmaantua vaatimukseen milloin vain projektin elinkaaren aikana, paitsi kun julkaisun viimeinen pyrähdys on loppunut ja ohjelmisto on asiakastestauksessa ennen julkaisua. Näin varmistetaan se, että lopputulos on mahdollisimman lähellä asiakkaan todellisia tarpeita ja tilausta. Tuotteen luonne voi muuttua projektin aikana useita kertoja, jos vaatimukset muuttuvat [Sch97]. Pyrähdysten tavoitteena on saada valmiiksi mahdollisimman paljon laadukasta ohjelmistoa, joka toimii ja voidaan asentaa asiakkaan ympäristöön [BDS99]. Kuvassa 9 on Scrumin prosessimalli. Prosessin perusta on *tuotteen työlista* (product backlog), joka sisältää listan tuotteeseen liittyvistä vaatimuksista. Tuotteen työlista on tarkoitettu julkiseksi listaksi, johon kuka tahansa projektiin liittyvä taho voi lisätä haluamiaan vaatimuksia. Listan vaatimukset ovat yleisellä tasolla eivätkä ota kantaa toteutukseen. Listaa kehitetään jatkuvasti projektin kuluessa. Sen sisältö muuttuu ja kehittyy koko tuotteen elinkaaren ajan.

Ohjelmistotuotteen julkaisua suunniteltaessa täytyy ottaa huomioon asiakkaiden vaatimukset, aika, kilpailu, laatu, tuotteen visio ja resurssit. Näiden pohjalta tuotteen omistajan vastuulla on miettiä julkaisun ajankohta ja sisältö. Tuotteen työlistaa priorisoidaan vastaamaan edellä mainittujen muuttujien painotuksia. Tärkeimmällä prioriteetilla olevat vaatimukset työlistasta valitaan julkaisuun [Sch97]. Yhden julkaisun valmiiksi tekemiseen voi mennä yksi tai useampia pyrähdyksiä. Kun pyrähdyksiä lähdetään suunnittelemaan, tuotteen työlistasta valitaan korkeimmin priorisoituja vaatimuksia *pyrähdysten työlistaan* (sprint backlog). Tärkeimpiä vaatimuksia valitaan pyrähdysten työlistaan ainoastaan sen verran, mitä yhdessä pyrähdyksessä ehditään saada valmiiksi. Ennen pyrähdysten alkua pidetään pyrähdysten suunnittelukokous, johon osallistuvat Scrum-tiimin jäsenet, Scrum-mestari ja tuotteen omistaja. Suunnittelukokouksessa sovitaan pyrähdysten tavoite, vaatimukset mitä halutaan saada valmiiksi, ja käydään läpi ne vaatimukset, jotka valitaan seuraavan pyrähdysten työlistalle. Tiimin itsenäisyys ja omatoimisuus ovat Scrumin ideologian pääperiaatteita, jotka kannustavat tiimin jäseniä ottamaan haasteita ja vastuuta omasta tekemisestään. Tuotteen työlista ja pyrähdysten työlista eroavat toisistaan siinä, että tuotteen työlistalla vaatimukset ovat yleisemmällä tasolla, kun taas pyrähdysten työlistassa vaatimukset pilkotaan pienempiin osiin, jotka on helppo hahmottaa ja joille on helpompi antaa työmääräarviot. Esimerkiksi, jos tuotteen työlistassa on vaatimus ”haku-toiminto kirjaston kirjoille”, tämä pilkotaan pyrähdysten työlistalle pienemmiksi vaatimuksiksi, kuten ”yksinkertai-

nen haku kirjan nimellä tai ISBN-tunnisteella”, ”tarkempi haku kirjoittajan nimellä ja kirjan julkaisuvuodella” ja ”hakutulosten selaaminen” [Coh07].



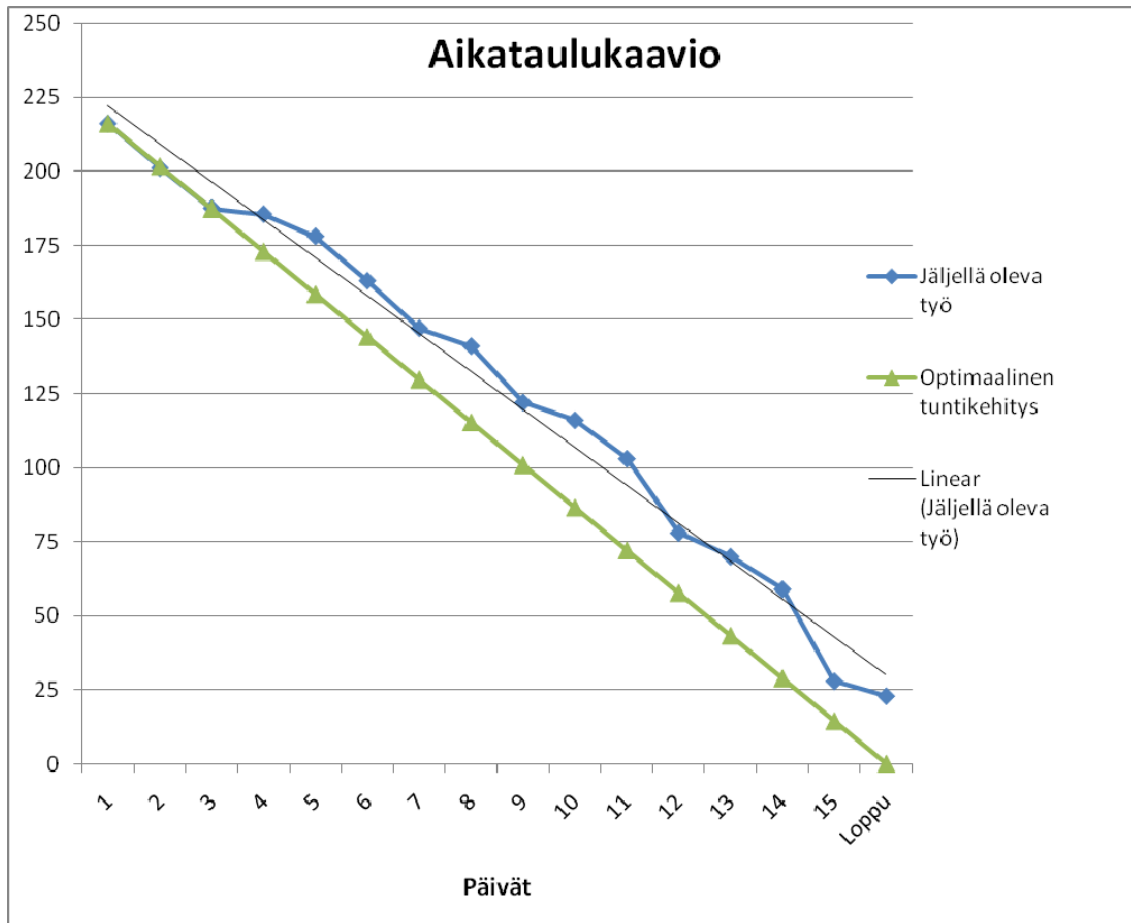
Kuva 9: Scrum-prosessimalli [ScB02].

Pyrähdyksen käynnistyessä alkavat päivittäiset Scrum-palaverit. Näiden palaverien tarkoitus on seurata meneillään olevaa pyrähdystä ja jakaa tietoa tiimin sisällä ja sidosryhmille. Kaikki sidosryhmät saavat osallistua kokoukseen, mutta heillä on lupa ainoastaan kuunnella ja seurata. Scrum-mestari huolehtii, että päivittäinen Scrum -palaveri pidetään, ja että kaikki ovat paikalla. Palaveri on erittäin tarkoin määritelty. Sen kesto saa enintään olla 15–30 minuuttia, ja se pidetään joka päivä samaan aikaan ja samassa paikassa. Kokouksessa puheoikeus on ainoastaan tiimin jäsenillä. Scrum-palaverissa jokainen tiimiläinen vastaa seuraaviin kolmeen kysymykseen: mitä tein eilisen kokouksen jälkeen, mitä aion tehdä tänään ennen huomista kokousta ja mitä esteitä minulla on tehtävien suorittamisessa. Taulukossa 4 on esitetty Scrum-palaverin sisältö.

Taulukko 4: Päivittäinen Scrum -palaveri [Sch99, Sut04, Sut05]

Päivittäisen Scrum -palaverin sisältö:
1. Mitä teit edellisen Scrum-palaverin jälkeen?
2. Mitä aiot tehdä ennen seuraavaa Scrum palaveria?
3. Onko jotain esteitä?

Pyrähdyksen tehtävien etenemistä seurataan *aikataulukkaaviolla* (burndown chart) (kuva 10), jonka avulla nähdään, miten paljon pyrähdys on edellä tai jäljessä optimaalista toteutusaikataulua. Aikataulukkaavion avulla pystytään arvioimaan, miten monta tuntia on vielä jäljellä pyrähdykseen kuuluvia tehtäviä. Kaaviosta ei nähdä, miten monta tuntia on kulutettu tehtävien tekoon, koska Scrumin ideologian mukaan ollaan kiinnostuneita jäljellä olevasta työmäärästä eikä tehtyyn työhön menneestä ajasta. Kaavion avulla seurataan visuaalisesti jäljellä olevan työmääräarvion kehittymistä. Kuvassa 10 alempi viiva kuvaa suunnitelmaa, jonka mukaan tehtävien pitäisi optimaalisesti edetä, jotta pyrähdysten lopussa kaikki tehtävät olisivat valmiita. Ylempi viiva kuvaa pyrähdysten toteutunutta tilannetta. Kuvan tilanteessa 25 tunnin verran tehtävistä jäi joko suorittamatta tai kesken. Syitä siihen, miksi kaavion suunnitelmasta toteutunut tulos jäi jälkeen, voi olla useita. Eräs syy on esimerkiksi tehtävien koon ja keston arviointivirheet. Tehtävien kestoa on sitä helpompi arvioida, mitä pienemmiksi palasiksi tehtävät on pilkottu ja mitä tarkemmin ne on määritelty. Erityisesti uusien asioiden toteuttamisessa tehtävien pilkkominen on tärkeää, jotta työmääräarvio voidaan antaa. Tärkein tehtävä Scrum-mestarilla iteraatiossa on kokouksissa ilmenevien esteiden poistaminen. Esteet voivat liittyä organisaatioon, laitteisiin, tiimityöhön tai tuotteeseen. Scrum-palaverissa ei ole tarkoitus keskustella ohjelmointiteknisistä ratkaisuksista, vaan palaverin tarkoitus on välittää tietoa muille. Kokouksen jälkeen tiimiläiset voivat keskustella tehtävien toteutukseen liittyvistä ongelmista.



Kuva 10: Esimerkki Scrumin aikataulukaaviosta [ScB01].

Pyrähdyksen kuluessa Scrum-tiimi toteuttaa pyrähdysten työlistaan valittuja tehtäviä, eikä sidosryhmien jäsenillä ole lupaa tulla suoraan tiimiläisten luo vaatimaan uusien toiminnallisuuksien toteuttamista heti. Kaikki uudet vaatimukset täytyy ensin laittaa tuotteen työlistaan, josta tuotteen omistaja voi priorisoida niitä. Jos pyrähdysten työlistaa halutaan muuttaa kesken pyrähdystä, pyrähdys täytyy keskeyttää ja aloittaa uusi pyrähdys. Uutta pyrähdystä varten tarvitaan pyrähdysten suunnittelukokous. Tuoteomistaja voi keskeyttää pyrähdysten. Keskeytyksen syytä voi olla esimerkiksi prioriteettien muuttuminen tuotelistalla tai kriittisen ohjelmistovirheen korjaamistarve. Pyrähdysten loputtua Scrum-prosessissa on pyrähdysten katselmointikokous, jossa paikalla ovat tiimiläiset, Scrum-mestari, tuotteen omistaja sekä edustajia kaikista sidosryhmistä, kuten asiakas, markkinointi ja myynti [Sch97]. Näissä kokouksissa tiimiläiset esittävät valmiiksi saamansa osat ja sidosryhmien edustajat saavat kommentoida tuotosta. Katselmointikokouksilla on tärkeä rooli prosessissa, koska ne ovat ainut paikka, jossa varmistetaan, että tekeillä oleva tuote on varmasti sellainen, minkä asiakas haluaa. Kokouksessa tuotteen omistajalla on oikeus hyväksyä tai hylätä pyrähdysten tuotos. Jos katselmointikokouksessa ilmenee puutteita tuotteessa tai tuote ei toimi halutulla tavalla, muutospyynnöt tallennetaan ensin tuotteen

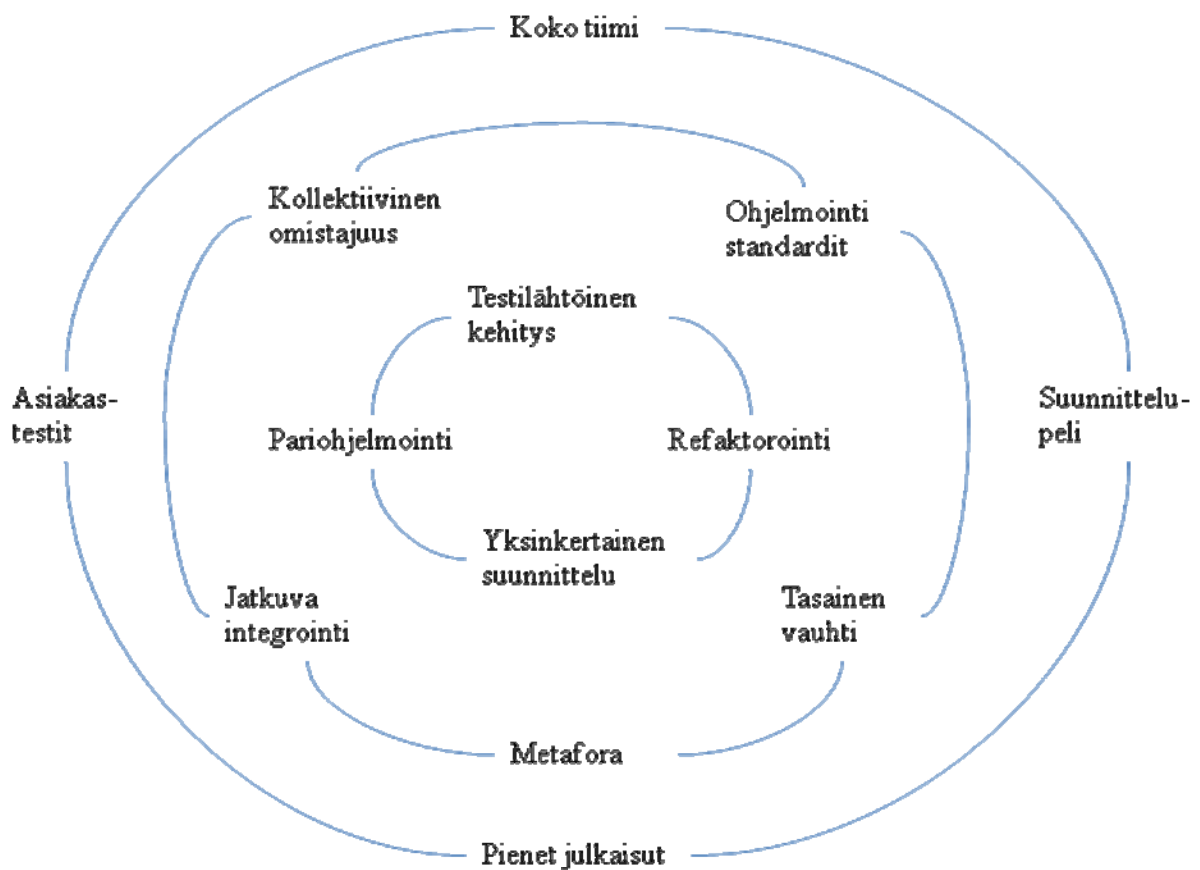
työlistaan ja vasta sen jälkeen ne voivat tulla tiimiläisille toteutettavaksi uudestaan. Pyrähdyksen katselmointikokouksessa käydään läpi tiimin ja johdon kanssa pyrähdyksen sujuminen ja esitellään ohjelmiston uudet toimivat osat. Kokouksessa tarkistetaan mahdolliset ongelmaosat, kuten tekniset ja muutoksista johtuvat ongelmat [Sch97]. Katselmoinnissa on tarkoitus antaa sidosryhmille mahdollisuus vaikuttaa lopputulokseen [BDS99]. Kokouksen aikana kaikille osallistujille tulee tietoa siitä, miten projekti on edennyt viime pyrähdyksen aikana [BDS99]. Pyrähdyksen päättyessä pidetään tiimin ja Scrum-mestarin kesken palaveri, jossa käydään läpi asiat, jotka onnistuivat hyvin pyrähdyksen aikana ja asiat, jotka tarvitsevat parannusta. Samalla seurataan, ovatko edellisessä arviointikokouksessa löydetty ongelmat vielä ajankohtaisia vai ovatko ne jo ratkaistu. Tämä prosessinparannukseen ja tiimin itsensä tehokkuuden lisäämiseen liittyvä palaveri on tärkeä, jotta ongelmalliset kohdat tunnistetaan ja löydetään entistä parempi tapa tuottaa ohjelmistoa.

Scrumin on sanottu olevan tehokas siitä syystä, että se keskittyy tiimityöskentelyyn. Prosessissa korostetaan ja kannustetaan tiimiläisten omaa vastuuta ja omia päätöksiä. Pyrähdyksen aikana kukaan ei tule häiritsemään tiimiä työnteossa, ja näin he saavat rauhassa keskittyä toteuttamaan vaadittavia tehtäviä. Scrum-mestarilla on tärkeä rooli pitää huolta siitä, että esteitä ei ole. Scrum ei ota kantaa siihen, miten työtä käytännössä tehdään, vaan tarjoaa välineet projektin seuraamiseen ja hallintaan. Scrum-prosessiin voi ja on helppo integroida käytäntöjä, jotka tukevat tehtävien suorittamista ja ryhmätyötä. Näitä käytäntöjä voidaan ottaa muista ketteristä prosesseista, kuten Extreme Programming (XP) -prosessimallista.

Scrum-projektin ominaisuuksia ovat joustava lopputulos ja aikataulu, pienet tiimit, tiheät katselmoinnit, yhteistyö ja oliopohjaisuus [Sch97]. Näiden ominaisuuksien avulla projektin toimivuutta voidaan parantaa ja lopputuotteen sisältöä kohdentaa vastaamaan tarpeita. Scrumin hyvä puoli on se, että prosessi tukee ja sallii muutokset sekä projektiin että kehitteillä olevaan ohjelmistoon missä vaiheessa projektia tahansa [Sch97]. Kehittäjät saavat pyrähdyksen aikana vapaasti suunnitella ja miettiä parhaita ratkaisuja ohjelmistoon. Pyrähdyksen aikana kukaan tiimin ulkopuolinen jäsen ei saa tulla häiritsemään tiimiä, jolloin tiimillä on täysi työrauha kehittää paras mahdollinen ratkaisu työn alla oleviin tehtäviin. Pienet tiimit ja tiimin sisäinen kommunikaatio auttavat ja nopeuttavat oppimista ja hiljaisen tiedon siirtymistä kaikille tiimin jäsenille. Scrum-prosessimalli on tehty soveltumaan oliopohjaisiin projekteihin [Sch97]. Scrum-prosessimalli tuo haasteita tiimille. Tiimin jäsenten pitää pystyä työskentelemään ja kommunikoimaan tiiviisti keskenään, jotta projekti onnistuu.

4.3 XP:n ideoita yhdistettynä Scrumiin

Scrum antaa kehyksen projektille, mutta ei ota kantaa käytäntöihin ja työntekoon. Scrumiin voi hyvin yhdistää toisen ketterän menetelmän, Extreme Programmingin. XP:n kehittäjä on Kent Beck [Bec99], joka aloitti prosessin kehityksen 1990-luvun puolivälissä. XP:ssä pilkotaan isoja ja monimutkaisia ominaisuuksia mahdollisimman pieniksi ja yksinkertaisiksi kokonaisuuksiksi, jotka on mahdollista saada valmiiksi yhden iteraation aikana [Arm04]. XP:aan kuuluu erilaisia käytäntöjä, jotka auttavat kehitystiimiä toteuttamaan ohjelmistoa. Kaksitoista pääkäytäntöä antavat selkeitä suuntaviivoja ja käytännön ohjeita kehitykseen. Käytännöt voidaan jakaa neljään ryhmään: palautteen anto, jatkuva prosessi, jaettu ymmärrys ja kehittäjän hyvinvointi. Kuvassa 11 on esitetty XP:n käytäntöjä.



Kuva 11: XP:n käytännöt [Jef01].

XP-projektissa on tärkeää, että tiimillä on avoin työtila, jossa kaikki tiimin jäsenet istuvat lähellä toisiaan ja kommunikointi on helppoa [Bec99]. Tiimiin kuuluu jokainen, joka on osallisena projektissa, kuten asiakas, kehittäjä, testaaja ja tiiminvetäjä. Tiimin jäsenet tekevät projektissa kaikkia mahdollisia tehtäviä, joita osaavat [Jef01].

Suunnittelupeli (planning game) on kokous, jossa kehitystiimi ja asiakas käyvät läpi ohjelmiston toiminnallisuuksiin liittyviä tarinoita ja valitsevat niistä tärkeimmät ja arvokkaimmat toteutettavaksi seuraavassa iteraatiossa. Tarinoissa pyritään kuvaamaan käyttäjien tarpeita ja rooleja [MGH07]. Suunnittelupeleissä asiakas seuraa projektin etenemistä ja vauhtia. Asiakas voi kokouksessa vaikuttaa ominaisuuksien toteutusjärjestykseen ja tärkeyteen.

Pieniä julkaisuja (small releases) tehdään mahdollisimman usein, kun jokin toimiva osa on saatu iteraation aikana valmiiksi. Julkaisuja voidaan tehdä muutaman päivän välein tai kuukausittain. Tämän tarkoitus on taata, että asiakas pääsee nopeasti testaamaan toimivaa tuotetta ja näin varmistaa tuotteen oikeellisuus [Jef01].

Jatkuvasti tavoitettavissa oleva asiakas on tärkeä XP-projektille, koska asiakas määrittelee tarinat ja hyväksymistestit sekä hyväksyy tai hylkää valmistuneet toiminnallisuudet. Kehittäjät voivat milloin tahansa kysyä asiakkaalta lisätietoja kehityksen alla oleviin toiminnallisuuksiin [Bec99].

Ohjelmointistandardien käyttö projektissa takaa koodin ulkoasun yhtenäisyyden [Jef01]. Tiimin jäsenet pystyvät helposti lukemaan ja ymmärtämään toisten kirjoittamaa ohjelmakoodia. Ohjelmointistandardien käyttö nopeuttaa uusien tiimin jäsenten integroitumisen ryhmään, koska he oppivat standardien avulla kirjoittamaan ja lukemaan koodia helposti.

Uusi ohjelmakoodi integroidaan mahdollisimman usein kehitettävään järjestelmään. Integroinnin yhteydessä koko järjestelmä käännetään uudestaan ja suoritetaan kaikki automatisoidut testit. Tällöin pystytään varmistumaan siitä, ettei ohjelma ole päässyt hajoamaan missään vaiheessa [Bec99].

Jokainen kehittäjä voi korjata kenen tahansa muun tiimin jäsenen kirjoittamaa ohjelmakoodia. Tämä perustuu kollektiiviseen koodin omistajuuteen, jossa koko tiimillä on vastuu ohjelmakoodin laadukkuudesta. Ohjelmakoodia voi parantaa milloin vain, kun siinä huomataan vanhentunut tai monimutkaista koodia.

Kehittäjän hyvinvoinnin ja jaksamisen kannalta sopiva työtahti on tärkeä. XP:n mukaan työtehtävät pitää pystyä tekemään työajalla. Ylitöitä ei pidetä hyvänä asiana, sillä ne kertovat projektin ongelmista [Bec99]. Ylitöiden teko on lyhytnäköistä. Ne auttavat hetkellisesti, mutta myöhemmin aiheuttavat ongelmia työntekijöiden väsyessä ja ollessa ylityövapailta.

Metafora on kuvaus tuotteesta, jonka ymmärtävät sekä asiakas että tuotekehitystiimi. Tiimin jäsenten välillä on ymmärrys kehitystehtävien sisällöstä ja tiimin ja asiakkaan välillä ymmärrys lopputuloksesta. Tuotteen toiminnan kuvaus auttaa asiakasta ja tiimiä puhumaan samaa kieltä tuotteesta.

Testaus on yksi XP:n kulmakivistä. XP:ssä suositetaan testilähtöistä lähestymistapaa, jossa yksikö- ja hyväksymistestit määritellään ennen varsinaista toteutusta. Testien määrittelyn jälkeen valitaan joku testeistä ja toteutetaan sitä vastaava ohjelman osa ja testataan, meneekö osa testistä läpi. Kehittäjät tekevät itse yksikkötestit ja asiakas määrittelee hyväksymistestejä tarinoille. Testaus tulee automatisoida mahdollisimman pitkälle, jotta yksikkötestejä voidaan ajaa aina, kun ohjelma käännetään.

Refaktorointi on kiinteä osa toteutusta. Aina kun tiimin jäsen huomaa järjestelmän rakenteessa jotain, joka aiheuttaa turhaa monimutkaisuutta, niin ohjelmalogiikka pitää toteuttaa uudestaan helpommalla tavalla. Refaktoroinnin avulla ohjelmakoodi pysyy kehityksen ajan yksinkertaisena.

Pariohjelmoinnissa kaksi kehittäjää toteuttaa samalla koneella tiettyä toiminnallisuutta. Toinen kehittäjää kirjoittaa ohjelmakoodia ja toinen katselmoi koodia samalla. Pariohjelmoinnissa tiimin jäsenten välillä tieto leviää helposti ja nopeasti. Pareja vaihdetaan välillä, jolloin tiimin jäsenet tekevät yhteistyötä kaikkien muiden tiimin jäsenten kanssa.

XP kannustaa yksinkertaiseen suunnitteluun, jossa vältetään tulevien ominaisuuksien ennakkointia, koodia, joka esiintyy useasti eri paikoissa ja ylimääräisiä luokkia. Iteraation ollessa käynnissä tehdään vain sellaiset toiminnallisuudet, jotka on mainittu asiakkaan määrittelemissä tarinoissa. Suunnittelua XP:ssä ei tehdä etukäteen, vaan sitä mukaan, kun projekti etenee [Jef01].

XP:stä kaikki käytännöt on mahdollista yhdistää Scrum-projektiin. Käytännöt tukevat Scrum-projektin pyrähdyksissä tapahtuvaa kehitystä. Ne tuovat varmuutta ohjelmiston koodin laadukuuteen, koska niiden avulla voidaan varmistaa hyvä kommunikoinnin taso ja kehittäjien oma vastuunotto projektin tuloksesta [ScB01]. Käytännöt auttavat uusia tiimin jäseniä integroitumaan nopeammin tehokkaammiksi tiimin työntekijöiksi, koska ne tukevat tiivistä vuorovaikutusta ja yhteistyötä.

4.4 Scrum ja tarve käyttäjälähtöiselle suunnittelulle

Ketterien menetelmien, kuten Scrumin, avulla on saatu nopeutettua ohjelmistokehitysprojekteja. Projekteissa on helpompi pysyä aikataulussa ja budjetissa, ja asiakkaan vaatimukset saadaan toteutettua. Ketteristä menetelmistä huolimatta valmiissa ohjelmistoissa on ollut toiminnallisuksia, joita loppukäyttäjä ei käytä lainkaan ja niistä on puuttunut toiminnallisuksia, jotka ovat olleet kriittisiä loppukäyttäjän työn kannalta [Pat02]. Ohjelmistokehitysprojektissa on edelleenkin käytetty paljon aikaa sellaisten toiminnallisuuksien tekoon, joita loppukäyttäjä ei haluaakaan käyttää. Vaikka asiakas ketterissä menetelmissä tekeekin tiivistä yhteistyötä

projektiryhmän kanssa, ei hän välttämättä osaa kertoa vaatimuksia, joita loppukäyttäjä tarvitsisi. Asiakas on harvemmin tuotteen todellinen loppukäyttäjä. Vaikka asiakas olisikin loppukäyttäjä, niin oltuaan läheisessä yhteistyössä kehitystiimin kanssa, asiakas oppii teknisestä rakenteesta ja ohjelmiston toteutuksesta paljon uusia asioita, ettei enää kykene toimimaan todellisena loppukäyttäjänä. Jotta valmis ohjelmisto olisi toimiva loppukäyttäjän kannalta, pitää käyttäjakeskeisiä menetelmiä ottaa mukaan Scrum-prosessiin, jotta voidaan varmistaa ohjelmiston käytettävyys jo projektin kuluessa.

Ohjelmistokehittäjät harvoin osaavat kunnolla käyttäjakeskeisen suunnitteluprosessien käytäntöjä tai käytettävyydestä eriyttäviä eri menetelmiä, minkä takia ohjelmistokehitysprosessiin tarvitaan erityistä käytettävyysoosaamista. Lisäämällä ohjelmistokehitysprosessiin käyttäjakeskeisen suunnitteluprosessin tärkeimpiä kohtia voidaan turvata valmiin ohjelmiston käytettävyys [CSM06].

Scrumissa ei, kuten ei muissakaan ketterissä menetelmissä, oteta kantaa käyttäjakeskeisen suunnittelun käyttöön [MGH07]. Kuitenkin laadukkaan ohjelmiston kehittäminen vaatii muidenkin ohjelmistokehitysvaiheiden suunnitelmallisen käytön. Käyttäjakeskeisen suunnittelun ja käyttöliittymäsuunnittelijan avulla voidaan Scrum-prosessissa tuottaa parempaa käyttäjätyytyväisyyttä ja käytettävyyttä [MGH07].

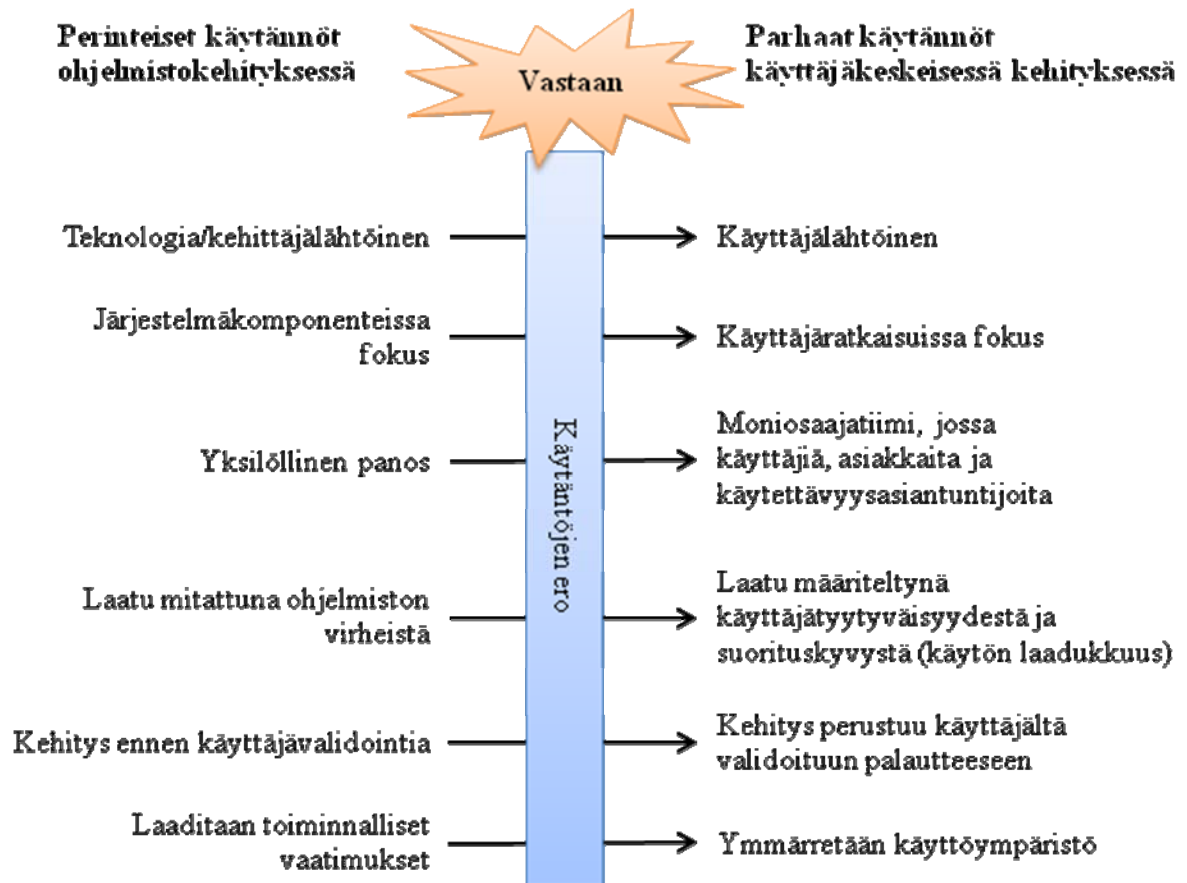
5 Käyttäjakeskeinen suunnitteluprosessi ja Scrum

Käytettävyys on tärkeä ominaisuus vuorovaikutteisissa ohjelmistoissa. Käyttäjakeskeisiä suunnitteluprosesseja käyttämällä varmistetaan, että ohjelmisto on käytettävä ja hyödyllinen käyttäjälle. Prosessi on työläs ja vaatii ammattitaitoisen käyttöliittymäsuunnittelijan. Scrum keskittyy ominaisuuksia, jotka tuovat eniten arvoa asiakkaalle. Vaatimusmäärittely ja suunnittelu ovat kevyitä ja pyrähdyn jälkeen asiakas validoi valmistuneet ominaisuudet. Molemmat prosessit pyrkivät laadukkaaseen tuotteeseen, mutta erillisinä prosesseina niissä on puutteita. Puutteet heikentävät tuotteen laatua. Scrumissa liian myöhään tehtävä ominaisuuksien validointi aiheuttaa lisätyötä projektissa. Käyttäjakeskeisissä suunnitteluprosesseissa liian raskas suunnitteluvaihe hidastaa projektin etenemistä. Vaatimukset saattavat muuttua kesken suunnittelun, jolloin käyttöliittymän suunnittelu kestää eikä kehitystyö pääse alkamaan.

Prosessien yhdistäminen lisää tuotteen laadukkuutta ja käytettävyyttä. Kun Scrumissa otetaan huomioon käytettävyys ja käyttäjakeskeinen suunnittelu, suunnittelun määrä kasvaa. Vaatimukset löydetään paremmin suunnitteluvaiheessa ja toteutus etenee suoraviivaisesti. Ominaisuudet validoidaan ennen toteutusta, joten käyttäjakeskeinen suunnittelu nopeuttaa kehitystyötä. Tärkeää on löytää molempien prosessien hyvät ominaisuudet, jotta niitä voidaan korostaa prosessimalleja yhdistäessä. Käyttäjakeskeisen suunnitteluprosessin ja Scrumin erilaisuudet aiheuttavat haasteita prosessimallien yhdistämiseen. Prosessien erilaisuus täytyy ottaa huomioon ja löytää sopiva kompromissi, joka tyydyttää molempia prosesseja.

5.1 Haasteet käyttäjakeskeisessä suunnittelussa ja Scrumissa

Käyttäjakeskeisen suunnitteluprosessin ja Scrumin yhdistämisen haasteet tulevat siinä, että toinen on keskittynyt projektin alkupäässä massiiviseen suunnitteluun ja loppukäyttäjien tarkkailuun, kun taas toinen keskittyy kehittäjien työn tehostamiseen ja vahvistamiseen [BHB04]. Käytettävyyden huomioiminen Scrumissa ei sisälly prosessiin, vaikka asiakas on läheisessä yhteistyössä kehitystiimin kanssa. Scrumissa jokaisen pyrähdyn tavoitteena on saada jotain valmista. Silloin usein jätetään käyttöliittymän ulkoasu viimeiseksi vaiheeksi ja käytetään siihen pyrähdyn loppuun jäänyt aika [Sin08]. Kuvassa 12 on lueteltu eroja ohjelmistokehityksen käytäntöjen ja käyttäjakeskeisen suunnittelun käytäntöjen välillä. Käytännöt liittyvät vahvasti prosesseihin ja niiden erojen pienentäminen tuo haasteita prosessien yhdistämiseen.



Kuva 12: Käytäntöjen erot ohjelmistokehityksessä ja käyttäjakeskeisessä kehityksessä [SeM04].

5.1.1 Oikea aika ja paikka käyttäjakeskeiselle suunnittelulle Scrumissa

Ongelmana käyttäjakeskeisen suunnitteluprosessin ja ohjelmistokehitysprosessien yhdistämisessä on löytää ohjelmistokehitysprosessin oikeat kohdat, joihin käyttäjakeskeisen suunnitteluprosessin menetelmät voidaan luontevasti liittää [SeM04]. Käyttäjakeskeisen suunnitteluprosessin leviämisen ja ohjelmistokehitysprosessiin integroitumisen esteenä on ollut käyttäjakeskeisen suunnitteluprosessin tuntemattomuus, käyttämättömyys ja hallinnoimattomuus tavalliselle kehittäjälle ja pienelle tiimille [SeM04]. Käyttäjakeskeisen suunnittelun integroiminen Scrum-prosessin oikeisiin kohtiin vaatii tiimiltä ja käyttöliittymäsuunnittelijalta perusteellista osaamista molemmista prosesseista.

Käyttäjakeskeistä suunnittelua on tehtävä ennen kehityksen aloittamista. Etukäteissuunnittelu alentaa kustannuksia ja aikaa, koska hyvän käyttöliittymäsuunnitelman pohjalta on luotettavampi arvioida työmäärää ja helpompi priorisoida vaatimuksia kuin sanallisesti kuvattujen vaatimusten pohjalta [FNB07b]. Ohjelmiston käyttöliittymää ei kannata suunnitella etukäteen täydellisesti, koska toteutusvaiheessa todennäköisesti ilmenee haasteita, joihin ei ole pystytty varautumaan [FNB07b]. Scrumissa seuraavien pyrähdysten suunnittelu vaikuttaa käyttäjakes-

keiseen suunnitteluun, koska pyrähdyn suunnittelussa vasta päätetään, mitä seuraavaan pyrähdykseen otetaan mukaan [FNB07a]. Jos käyttäjakeskeisessä suunnittelussa halutaan varmistua siitä, että turhaa suunnittelutyötä ei tehdä, tuotelistan prioriteettien on ainakin tärkeimpien vaatimusten osalta oltava kohtalaisen vakaa ja ainakin seuraavaan julkaisuun tulevat asiat on tiedettävä.

Ennen toteutusvaihetta tehtävä käyttäjakeskeinen suunnittelu parantaa lopullisen tuotteen käyttäjätyytyväisyyttä ja yhtenäisyyttä [FNB07b]. Koko käyttöliittymän suunnittelua ei kannata eristää pelkästään käyttöliittymäsuunnittelijalle, vaan ottaa päätöksentekoon koko tiimi mukaan. Näin tiimi pystyy käyttöliittymän suunnitteluvaiheessa ottamaan kantaa teknisiin käyttöliittymäratkaisuihin, jolloin toteutusvaiheessa haasteita esiintyy vähemmän. Kun aikanaan käyttöliittymä siirtyy käyttöliittymäsuunnittelusta vaatimuksena kehitystiimille [FNB07b], niin tiimi tietää etukäteen, mitä seuraavaan pyrähdykseen on tulossa ja se pystyy varautumaan siihen ajoissa.

5.1.2 Vaatimusten kerääminen

Vaatimusmäärittely on koko suunnittelun perusta. Sen aikana tehdyt virheet ovat kalliita korjata myöhemmin [Kuj98]. Sen takia vaatimusten keräämiseen kannattaa käyttää aikaa ja vaivaa. Myös suunnittelun laatu paranee, kun vaatimusmäärittely on tehty tarkasti [Kuj98]. Vaatimusmäärittelyssä vaatimuksia kerätään skenaarioiksi, jotka kuvaavat käyttöympäristön, käyttäjät, käyttäjäroolit, tehtävät ja toiminnot [MGH07]. Asiakkaat eivät osaa kuvailla tarpeitaan riittävän tarkasti, jotta kehitystiimi pystyisi niiden perusteella toteuttamaan halutunlaista ohjelmistoa. Asiakas ei osaa välttämättä ottaa kantaa käytettävyyteen ja kehitystiimikin mieluummin keskittyy koodin laatuun ja toiminnallisuuksien valmistumiseen eikä käyttöliittymän laatuun [Ban03, WBJ97].

Ketterissä menetelmissä asiakkaan pitäisi pystyä kertomaan, millainen ohjelmisto tarvitaan, mitä sillä pitää pystyä tekemään ja miten ohjelmiston tulee vaikuttaa työntekoon [BHB04]. Kuitenkin Scrumissa käyttäjän käytettävyystarpeiden määrittely jää vajaaksi vaatimusten määrittelyvaiheessa sekä inkrementaalisen kehityksen edetessä [Sin08], koska riittävää osaamista ei välttämättä asiakkaalta löydy. Asiakkaiden tekemän vaatimusten keräämisen tueksi tarvitaan käyttäjien havainnointia, työnkuvan mallintamista sekä vision luomista, jotta asiakkaat ymmärtävät, mitä haluavat ja kehittäjät tietävät, millainen ohjelmisto pitää toteuttaa [BHB04]. Ketterät menetelmät eivät huomioi sitä, että asiakas ja käyttäjä eivät ole suunnittelijoita, joten he eivät tiedä kaikki teknisiä tai tulevaisuuden mahdollisuuksia, joita ohjelmiston suunnittelussa on tarpeen ottaa huomioon [BHB04].

Ketterissä menetelmissä vaatimuksia kerätään sitä mukaa, kun projekti etenee. Projektin alussa ei huolehdita siitä, onko kaikki tarpeelliset vaatimukset jo löydetty, vaan varaudutaan siihen, että projektin kehittyessä ilmaantuu lisää vaatimuksia [DZN07]. Käyttäjakeskeisessä suunnittelussa pitää ennakkoon tietää, millaisia ominaisuuksia ollaan kehittämässä, jotta käyttäjältä voidaan varata aikaa käyttäjähavainnoinnin tekoon [Hod05]. Käyttäjähavainnointien tuloksena on suuri määrä erilaisia vaatimuksia. Käyttäjähavainnointiin pitää etukäteen valmistautua hyvin ja löytää yhteinen aika käyttäjän kanssa. Aikaa tarvitaan valmistautumiseen ja havainnointiin useita viikkoja, ennen kuin tuleva ominaisuus pääsee toteutusvaiheeseen. Tämä aiheuttaa ristiriitaa prosessien yhdistämisessä. Käyttäjakeskeisten suunnitteluprosessien ja ketterien menetelmien yhdistämisessä pitää löytää jonkinlainen välimuoto, jossa vaatimuksia ei tarvitse suunnitella liian aikaisin. Täytyy kuitenkin varmistaa, että riittävästi tietoa vaatimuksista on saatavilla muutamaa iteraatiota ennen toteutusvaihetta, jotta tarvittavat käyttäjätutkimukset ehditään suorittaa hyvin.

5.1.3 Suunnittelun määrä

Suunnittelun määritelmä on yksi pääasioista, jotka eroavat ketterissä menetelmissä ja käyttäjakeskeisessä suunnittelussa [FNB07a]. Käyttäjakeskeisissä suunnitteluprosessimalleissa yleensä suositaan koko käyttöliittymän suunnittelua ennen toteutusta. Ketterien menetelmien manifestissa todetaan, että ketterissä menetelmissä suositaan ”vastausta muutokseen, ei suunnitelman seuraamista” ja ”toimivaa ohjelmistoa yli kattavan dokumentaation” [Bec01]. Käytettävyyssuunnittelija määrittelee suunnittelun käyttöliittymäsuunnitteluksi, jossa mietitään, miten käyttäjä näkee ja kokee ohjelmiston. Ohjelmistokehittäjät määrittelevät suunnittelun tekniäksi rakenteen suunnitteluksi [FNB07a]. Ketterissä menetelmissä ei erikseen mainita käyttöliittymäsuunnittelun huomioonottamista suunnitteluvaiheessa. Käyttäjakeskeisessä suunnittelussa kerätään alussa kaikki mahdollinen käyttöliittymään vaikuttava tieto käyttäjiltä. Käyttöliittymän suunnittelussa käyttäjän havainnointi on välttämätöntä, koska käyttäjä ei osaa kuvailla suullisesti toimintatapojaan todenmukaisesti [BHB04].

Scrumissa ohjelmistoa suunnitellaan pyrähdys ja julkaisu kerrallaan eikä oteta kantaa myöhempiin vaiheisiin. Scrumissa pyritään aloittamaan toteutus heti, kun ensimmäiset tärkeimmät vaatimukset ovat selvillä, ja projektin edetessä tutustutaan paremmin ohjelmiston toimintaan [FSM06]. Ketterissä menetelmissä liian perusteellinen suunnittelu on kaikki pois toteutuksesta [FSM06]. Käyttäjakeskeisessä suunnittelussa kannatetaan kattavan käyttäjien havainnoinnin tekoa prosessin alussa, koska käyttäjän työkäytännöt muuttuvat hitaasti, vaikka tekniikka ja toiminnallisuuksien toteutus muuttuvat nopeasti [BHB04]. Tällöin käyttäjähavainnointien tulokset eivät ole turhaa työtä, vaikka havainnoinnit tehtäisiinkin paljon toteutusta aikaisemmin.

Scrumissa ei ole kattavaa etukäteissuunnittelua käyttöliittymälogiikalle eikä käyttäjien palautetta saada pyrähdysten ollessa käynnissä [Sin08]. Siksi mahdollisuus tehdä prototyyppisiä ja validoida vaatimuksia ennen pyrähdystä tai sen aikana ei onnistu [Sin08]. Käyttäjäkeskeinen suunnittelu ohjaa tiimiä tutkimaan ja ymmärtämään käyttäjää niin paljon kuin mahdollista ennen toteutusta, jotta ohjelmisto vastaisi käyttäjän todellisia tarpeita [FSM06]. Suunnitteluvaiheessa on tärkeää, että käyttöliittymäsuunnittelija pystyy kuvaamaan ideansa ja ajatuksensa paperille, jotta muu tiimi pystyy jatkuvasti näkemään ja testaamaan käyttöliittymäsuunnitelmaa ja antamaan palautetta siitä [MGH07].

Käyttäjäkeskeisessä suunnittelussa tuotetaan käyttöliittymään liittyviä dokumentteja, jotka sisältävät kaiken tarvittavan tiedon käyttöliittymän toiminnasta. Dokumentit tukevat käytettävyyssiantuntijoiden ja ohjelmistokehittäjien välistä vuorovaikutusta [FSM06]. Ketterät menetelmät pyrkivät välttämään kaikkea turhaa dokumentointia, joten prosessien yhdistämisessä pitää pyrkiä kompromissiin dokumentoinnin määrän suhteen. Dokumentoinnissa tärkeintä on tehdä vain sellaisia muistiinpanoja, joita joku tarvitsee ja lukee. Ketterissä menetelmissä tiimin tiivis kommunikaatio vähentää dokumentoinnin tarvetta, koska tieto liikkuu suullisesti tiimin sisällä.

5.1.4 Käyttöliittymäsuunnitteluun käytettävä aika

Käyttöliittymäsuunnittelu on suurelta osin luovaa työtä, jolle on hankalaa määritellä työmääräarviota tunneissa [Hod05]. Käyttöliittymäsuunnittelijoille täytyy antaa riittävästi aikaa ennen toiminnallisuuksien toteutusta selvittää käyttäjän tarpeet ja iteroida prototyyppisiä [FSM06]. Käyttöliittymäsuunnittelussa iteroidaan useita kertoja lyhyessä ajassa ja käyttöliittymää parannetaan joka kierroksella testien perusteella. Kuitenkin liian pitkää aikaa ei kannata käyttää suunnitteluun, koska myös käyttäjäkeskeisessä suunnittelussa toimiva ohjelmisto on tärkein asiakkaalle ja loppukäyttäjälle [FNB07b]. Koko tiimin on hyvä yhdessä määritellä, millainen taso käyttöliittymän toimivuudessa riittää suunnittelun iteroinnin lopettamiseen ja käyttöliittymäsuunnitelman siirtymisestä suunnittelusta toteutukseen [Hod05].

Käyttäjäkeskeisen suunnitteluprosessin mukaan tehty käyttöliittymäsuunnitelma nopeuttaa toteutusvaihetta. Se myös parantaa ohjelmiston laatua, jolloin valmiissa ohjelmassa on vähemmän käytettävyysongelmia. Käyttöliittymäsuunnittelussa hyvä puoli on siinä, että käyttöliittymän prototyypin pystyy piirtämään paperilla muutamassa minuutissa ja prototyyppiä voi heti testata esimerkiksi loppukäyttäjällä.

5.1.5 Käyttöliittymän testaus ennen toteutusta

Jos kunnollista ja testattua käyttöliittymäsuunnitelmaa ei tehdä ennen toteutusta, kehittäjät joutuvat toteutusvaiheessa arvailemaan ominaisuuden todellisen käyttötarkoituksen ja suunnittele-

maan itse käyttöliittymän koodauksen sivutuotteena. Tällöin kehitysvaiheessa valmistunutta käyttöliittymää ei ole verifioitu todellisilla käyttäjillä ennen toteutusta [Mil05]. Jos käyttöliittymää testataan vasta sen ollessa valmis, seuraavissa pyrähdyksissä täytyy varautua korjaamaan käyttöliittymän ulkoasua ja ohjelmiston toiminnallisuuksia [MeA06]. Scrumissa käyttöliittymien testaus tehdään vasta sitten, kun käyttöliittymä on toteutettu ja käyttäjä voi sitä testata käyttämällä toimivaa ohjelmaa [McM05, DZN07]. Käyttäjä arvioi toimivaa ohjelmaa ja hyväksyy ohjelman sellaisena kuin se on jo toteutettu tai vaatii muutoksia toteutukseen [McM05].

Käyttäjän toimintojen kannalta ohjelmistoa voi ainoastaan testata joko toimivan käyttöliittymän tai prototyypin kautta [BHB04]. Käyttöliittymän prototyyppejä testataan käyttäjillä ennen toteutusta, jotta käyttäjiltä saadut palautteet voidaan ottaa huomioon käyttöliittymän uudistetussa prototyypissä [BHB04]. Vaatimuksia ja käyttöliittymäprototyyppejä pitää testata ja niiden suunnittelua iteroida ennen kehitykseen menoa [BHB04]. Testaajina tarvitaan ohjelmiston oikeita käyttäjiä, jotta saadaan mahdollisimman varmaa ja luotettavaa tietoa ohjelmiston toimivuudesta. Prototyypin teossa on tärkeää, että sen tekeminen on helppoa ja nopeaa, jotta prototyyppien käyttäminen ei vie liikaa aikaa ketterässä projektissa [Con00].

5.1.6 Asiakas ja käyttäjä

Käyttäjakeskeisessä suunnitteluprosessissa on tärkeää päästä tutkimaan ja seuraamaan oikean loppukäyttäjän työskentelyä ja työtapoja. Scrumissa tuoteomistaja toimii asiakkaan roolissa ja määrittelee tulevan ohjelmiston toimintoja. Käyttäjä ja asiakas ovat usein eri henkilöitä. Käyttäjä on ohjelmiston loppukäyttäjä, kun taas asiakas on laajempi määritelmä ja saattaa sisältää käyttäjän [BHB04]. Asiakas määrittelee toiminnallisuuksia ohjelmistolle ja hyväksyy ohjelmiston ostamisen, mutta asiakas ei välttämättä tiedä, mitkä työtehtävät ovat kriittisiä ja mitkä aiheuttavat ongelmia loppukäyttäjien joukossa [BHB04, PLR07]. Scrumissa ei välttämättä saada oikeilta loppukäyttäjiltä tietoa käyttäjien tarpeista, jolloin vaatimukset perustuvat oletuksiin käyttäjien tarpeista [MGH07]. Käyttäjakeskeisessä suunnittelussa on tärkeää selvittää, keitä käyttäjät ovat ja miten he käyttävät ohjelmistoa [BHB04]. Käyttäjakeskeisessä suunnittelussa keskitytään loppukäyttäjään [FSM06]. Scrumissa pyrähdysten lopussa olevassa pyrähdysten katselmointikokouksessa mukana ovat projektiryhmän sidosryhmiin kuuluvat henkilöt. Kokouksessa sidosryhmien edustajat katselmoivat pyrähdysten tuotoksen ja antavat siitä palautetta tiimille ja tuoteomistajalle [CSM06]. Käyttäjien pitää olla mukana ohjelmistokehitysprosessissa, jotta ohjelmistoa voidaan kehittää käyttäjää paremmin tukevaan suuntaan [FSM06]. Projektissa on hyvä olla muita loppukäyttäjää tukevia rooleja, jotta loppukäyttäjän ei tarvitse liian tiiviisti seurata ohjelmistokehitystä [FSM06]. Scrumissa tuoteomistaja huolehtii ohjelmiston toimivuudesta realistisissa tilanteissa, jolloin käyttäjää ei tarvita päivittäisessä projektityöskentelyssä.

5.1.7 Roolien vastuut

Scrumissa tuoteomistaja vastaa ohjelmiston visiosta ja sen määrittelystä. Yhtenä osana ohjelmiston määrittelyä on käytettävyyden huomioonottaminen. Tuoteomistajan työaika kuluu tuotelistan priorisoinnissa ja markkinoinnin ja myynnin tarpeiden huomioimisessa, jolloin käytettävyydelle ei usein jää riittävästi aikaa [Sin08]. Tuoteomistajalla ei ole riittävästi taitoa ottaa huomioon kaikkia käytettävyyteen liittyviä piirteitä priorisointia tehdessään ja vaatimuksia määritellessään [Sin08]. Tuoteomistajan tärkein tehtävä on pitää huolta vaatimusten priorisoinnista tuotelistassa, jotta kaikki sidosryhmät tietävät, missä tulevien vaatimusten suhteen menään. Sidosryhmät voivat yrittää vaikuttaa tuoteomistajaan priorisointien muuttamiseksi, mutta sidosryhmät eivät pysty aiheuttamaan riskejä projektin etenemiseen [DeL03]. Tehokkaan ja toimivan käyttöliittymän määrittely ja suunnittelu vaatii paljon osaamista ja aikaa, joten tuoteomistajalla ei usein ole kunnolliseen käyttöliittymäsuunnitteluun aikaa [Sin08].

Scrumissa tiimin sisällä ei ole määriteltyjä rooleja [PaN08, Hod05], mutta käyttäjakeskeinen suunnittelu vaatii paljon osaamista, jolloin on helpompi määritellä siihen oma rooli kuin opettaa koko tiimi tekemään käyttöliittymäsuunnittelua. Käyttäjakeskeisessä suunnittelussa käyttöliittymäsuunnittelijat ovat kaikki oman alansa erityisosaajia. Eri suunnitteluprosessin vaiheille ovat omat asiantuntijat [McM05]. Scrumissa tiimin sisällä tiimiläisillä ei ole määriteltyjä rooleja, vaan tarkoituksena on, että tiimi koostuu moniosaajista [McM05]. Koska tiimin jäsenet pystyvät ottamaan mitä tahansa tehtäviä pyrhdykseen on määritelty, tiimin tuottavuus ja tehokkuus lisääntyvät [Hod05]. Tiimiin ei tarvita niin paljon jäseniä, kun kaikki tiimin jäsenet hallitsevat eri ohjelmistokehityksen tehtäviä. Tämä aiheuttaa haasteita Scrumin ja käyttäjakeskeisen suunnittelun yhdistämisessä, koska on helpompaa pitää tiimissä erillistä käyttöliittymäsuunnittelijaa kuin perehdyttää koko tiimi huolehtimaan ohjelmiston käyttöliittymäsuunnittelusta ja käytettävyydestä. Toisaalta käyttöliittymäsuunnittelijan rooli tiimissä rikkoo Scrumin sääntöä moniosaajista. Käyttäjakeskeiseen suunnitteluun käyttöliittymäsuunnittelijan työn tueksi voidaan ottaa muut tiimin jäsenet, jolloin tiimi kokee olevansa osa käyttäjakeskeistä suunnittelua ja kommunikaatio tiimin ja käyttöliittymäsuunnittelijan välillä on tiivistä ja avointa [McM05].

Scrum-tiimin erikoistuminen eri työtehtäviin voi johtaa siihen, että pyrhdykseen määriteltyjä tehtäviä ei saada valmiiksi, kun joku tiimin jäsenistä esimerkiksi on kuormitettu muilla tehtävillä ja muut eivät osaa jatkaa keskeneräisiä tehtäviä [Hod05]. Erikoistuminen johtaa siihen, että pyrhdyksen suunnitteluvaiheessa tehtävät pilkotaan sellaisiin osiin, että jokaiselle löytyy sopivia työtehtäviä [Hod05]. Scrum-tiimeissä tärkeää on niiden itseohjautuvuus ja vastuunotto [Hod05]. Tämän vuoksi on tärkeää, että käyttöliittymäsuunnittelija on osa tiimiä eikä osa muualta organisaatiosta määriteltyä tiimiä.

5.1.8 Tiimin jäsenten sijainti ja yhteistyö

Ketterissä menetelmissä ketteryys perustuu tiimin tiiviiseen yhteistyöhön, jossa tieto liikkuu jatkuvasti ja kaikki tietävät, missä mennään [Hod05]. Paras tulos ohjelmistokehitysprojektille saadaan, kun käyttöliittymäsuunnittelijat ja ohjelmistokehitystiimi ovat mahdollisimman lähellä toisiaan. Käyttöliittymäsuunnittelijoiden on hyvä kuulua tiimiin, jotta vuorovaikutus on mahdollisimman tiivistä ja kitkatonta. Jos käyttöliittymäsuunnittelija ei kuulu tiimiin, yhteistyö ei toimi riittävän saumattomasti ja ketterästi. Toisaalta jos tiimit on hajautettu eri puolille maapalloa, kommunikaation sujuvuudesta täytyy pitää erityistä huolta, jotta kommunikaatiokatkokset ja väärinymmärrykset eivät aiheuta ongelmia ja hidastuksia projektissa [NaT08]. Kun käyttöliittymäsuunnittelija on osa Scrum-tiimiä, tuottavuudessa ja tehokkuudessa saavutetaan hyötyjä [Hod05]. Päivittäinen kommunikaatio käytettävyyssiantuntijoiden ja kehittäjien välillä on tärkeää projektin onnistumisen kannalta [Hod05]. Ilman päivittäistä kommunikaatiota ja vuorovaikutusta käytettävyyssiantuntijoiden ja kehittäjien välillä projektin tilasta ja työn etenemisestä syntyy epäselvyyttä [Hod05].

Molemmissa prosessimalleissa tiimin sisäinen yhteistyö on tärkeää [FSM06]. Scrumissa tiimin jäsenet valitsevat työtehtävänsä pyrähdyn aikana ja päivittäin kommunikoivat etenemisestään toisilleen. Käyttäjakeskeisessä suunnittelussa tiimi keskittyy käyttäjään koko projektin ajan [FSM06]. Käyttäjakeskeisessä suunnittelussa alkuvaiheessa käyttöliittymäsuunnittelijat pitävät aivoriihiä ideoinnin tukena, jolloin koko suunnittelutiimin ajatukset suunniteltavasta käyttöliittymästä käydään läpi.

5.2 Yhteiset käytännöt prosessien yhdistämisessä

Jotta käyttäjälähtöinen suunnitteluprosessi ja Scrum toimisivat yhdessä, joidenkin asioiden täytyy prosessien ja asiantuntijoiden välillä toimia. Tärkeimmät näistä asioista ovat molempien prosessien iteratiivisen luonteen yhdistäminen yhdeksi ja tiimin saumaton työskentely yhdessä käytettävyyden parantamisen puolesta. Vuorovaikutus prosessien ja ihmisten välillä on välttämätöntä, jotta prosessien yhdistämisen hyödyt auttavat ja parantavat kehitettävää ohjelmistoa. Molemmissa prosesseissa on tärkeää, että käyttäjä tai asiakas on helposti saatavilla koko ohjelmistokehitysprojektin ajan [MGH07].

5.2.1 Iteratiivinen luonne

Käyttäjakeskeisessä suunnittelussa pitää muuttaa käyttäjätiedon keräämisen ajankohtaa ja sitä, miten usein tietoa kerätään. Näin suunnittelu pystyy paremmin tukemaan ketterää kehitystä [Mil05]. Ketterien menetelmien periaate aikaisesta ja jatkuvasta tuotejulkaisusta vaatii ohjelmistokehitykseltä sitä, että iteraatioissa keskitytään muutamaa kokonaisuuteen, jotka on mahdol-

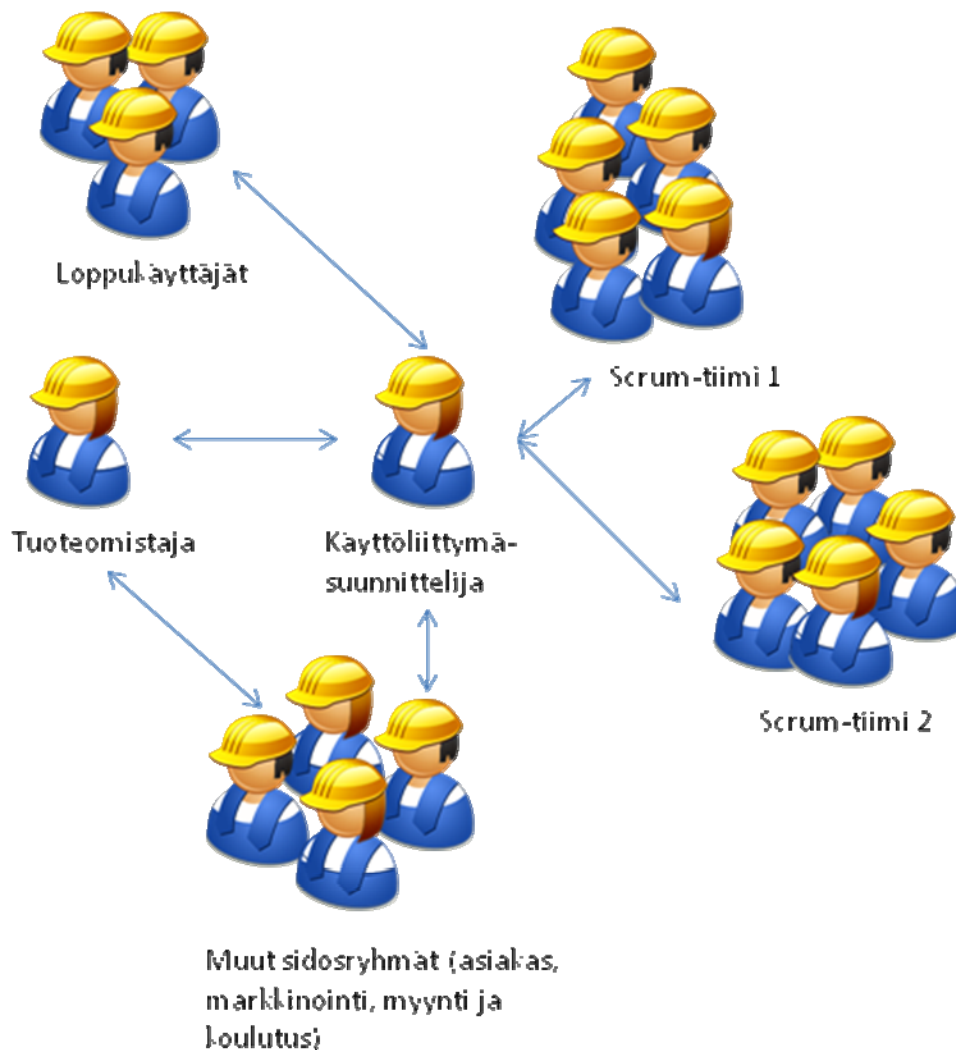
lista saada valmiiksi iteraation aikana. Tämä helpottaa käyttäjakeskeistä suunnittelua, koska ennen iteraation alkua voidaan suunnitella muutama käyttöliittymä eikä tarvitse suunnitella koko ohjelmiston käyttöliittymää yhdellä kertaa [Mil05]. Scrumin iteratiivinen luonne aiheuttaa sen, että käyttäjakeskeisen suunnittelun on sopeuduttava Scrumin iteraatioihin, jotta suunnittelusta saadaan paras hyöty ohjelmistokehitykseen. Molemmissa prosesseissa empiiristä tietoa kerätään edellisistä iteraatioista ja kerätyillä tiedoilla pyritään parantamaan lopputulosta [CSM06].

5.2.2 Kommunikaatio sidosryhmien välillä

Yhdistetyssä prosessimallissa täytyy löytää paras tapa yhteistyölle ja kommunikaatiolle käyttöliittymäsuunnittelijoiden ja ohjelmistokehittäjien välillä [BEH06]. Kanssakäyminen ja kommunikaatio tiimissä täytyy olla toimivaa [NaT08]. Jos vuorovaikutus tiimin ja käytettävyyssiantuntijoiden välillä ei toimi, siitä aiheutuu puutteita tiedonkulussa ja ongelmia käytettävyyden varmistamisessa [FSM06]. Päivittäinen kommunikaatio kehitystiimin ja käyttöliittymäsuunnittelun välillä takaa toimivan ohjelmiston [Mil05, Ban03, FNB07a, LeM07]. Ongelmat kehitystiimin ja käyttöliittymäsuunnittelijan kommunikoinnissa saattavat johtua erilaisesta merkintätavasta, kielestä, työvälineistä sekä käytettävyyteen liittyvien asioiden priorisoinnista [SeM04]. Sekä käytettävyyssiantuntijoiden että Scrum-tiimin tulee ymmärtää toistensa tavoitteita ja pyrkimyksiä [CSM06]. Kommunikaatio kehitystiimin ja käyttöliittymäsuunnittelijan välillä tarvitsee tukea organisaatiolta, jotta esimerkiksi ilmaisutavat ovat ymmärrettävissä molemmissa ryhmissä [JMW91]. Kuvassa 13 näkyy erilaisia sidosryhmiä, jotka liittyvät ohjelmiston kehitykseen sekä sidosryhmien välistä vuorovaikutusta. Käyttöliittymäsuunnittelija on asiakkaan ja kehitystiimin välissä [Ban03] ja suunnittelija tekee yhteistyötä yrityksen sisällä esimerkiksi markkinoinnin kanssa [FNB07b]. Käyttöliittymäsuunnittelija on Scrum-tiimeille käyttäjän edustaja.

Käyttöliittymäsuunnittelijan tulee olla nopeasti kehitystiimin saatavilla, jos kehityksen aikana ilmenee käytettävyyteen tai käyttöliittymiin liittyviä haasteita tai kysymyksiä. Scrum-tiimin ja käytettävyyssiantuntijoiden täytyy työskennellä päivittäin yhdessä, jotta tieto eri asiantuntijoiden välillä siirtyy mahdollisimman vaivattomasti [FSM06, FNB07c]. Käyttöliittymäsuunnittelijat kommunikoivat jatkuvasti Scrum-tiimin kanssa käyttöliittymämuutoksista, -testauksesta ja epäselvistä suunnitteluratkaisuista [FNB07c]. Kehittäjät voivat antaa palautetta käyttöliittymäsuunnittelijoille, jos käyttöliittymän toteutusvaiheessa huomataan jonkin ominaisuuden olevan erittäin työläs toteuttaa. Näin käyttöliittymäsuunnittelija voi reagoida kehittäjien palautteeseen ja miettiä vaihtoehtoja ratkaisua toteutukselle.

Tuoteomistajan tulee olla aktiivinen projektissa ja huolehtia vaatimusten priorisoinnista ja hyväksymistestauksesta. Kaikkien projektiryhmän jäsenten tulee olla projektiryhmän käytettävissä koko projektin ajan [FSM06]. Jatkuva vuorovaikutus käyttöliittymäsuunnittelijan ja Scrum-tiimin välillä takaa sen, että käyttöliittymäsuunnittelijat voivat koko projektin ajan varmistaa ohjelmiston ja toteutettujen käyttöliittymien yhtenäisyyden tyylioppaiden sääntöjen kanssa [FNB07c].



Kuva 13: Käyttöliittymäsuunnittelijan ja sidosryhmien välinen vuorovaikutus [muokattu Sin08].

Jos käyttöliittymäsuunnittelijat eivät ole osa ohjelmistokehitystiimiä, he eivät välttämättä tiedä teknisiä rajoitteita, jotka estävät käyttöliittymän toteutuksen suunnitellulla tavalla. He eivät silloin tiedä syitä tehtyjen toteutusten takana [JMW91]. Toisaalta jos ohjelmistokehitystiimille vain annetaan suunnitellut käyttöliittymäkuvat eikä toteutusvaiheen aikana käyttöliittymäsuunnittelijat tue kehitystiimiä, lopputulos saattaa olla jotain aivan muuta kuin suunnitelma. Jos ohjelmistokehitystiimi ei ymmärrä joitain suunnitteluratkaisuja, he kehittävät niihin omat

ratkaisunsa, ellei käyttöliittymäsuunnittelija ole tukemassa heidän työtään. Käyttöliittymäsuunnittelijan ei tarvitse tehdä täydellistä käyttöliittymäsuunnitelmaa, koska hänen ei tarvitse tietää tekniikasta ja toteutustavoista kaikkea. Kehitystiimi kommentoi oman tietämyksensä valossa käyttöliittymäsuunnittelijan ehdotusta, jolloin käyttöliittymäsuunnittelija voi tiimin palautteen perusteella kehittää käyttöliittymää [FNB0b].

5.2.3 Käytettävyydestuki kehittäjille

Tyylioppaat, käytettävyyssmallit, heuristinen arviointi ja tehtävien kulkukaaviot auttavat pienissä tiimeissä käytettävyyden parantamisessa ja huomioonottamisessa [MeS04]. Standardien ja tyyli-sääntöjen käyttö projektissa tehostaa käyttöliittymäsuunnittelua [Hod05, Kan03]. Tyylioppaiden haasteena on niiden tekemiseen kuuluva aika, ellei niitä ole valmiiksi määritelty esimerkiksi katamaan kaikkien yrityksen ohjelmistojen ulkoasua [Kan03]. Tyylioppaat ja -ohjeistukset auttavat sekä kehittäjiä että käytettävyyssuunnittelijoita ohjelmiston kehityksen elinkaaren aikana. Ne tukevat pitemmällä aikajänteellä ohjelmiston yhdenmukaisuutta ja toimivuutta. Tyylioppaista kannattaa tehdä kevennetyt versiot, jotka sisältävät tärkeimmät asiat käytettävyydestä ja käyttöliittymien toimintalogiikasta, jolloin aikaa ei kulu liikaa etukäteistyöhön [MGH07].

Työvälineitä tarvitaan tukemaan ohjelmistokehittäjien osaamista käyttöliittymäsuunnittelun saralla, jotta kehittäjät saavat lisätietoa itselleen ja voivat jakaa sitä muille [MeS04]. Käyttäjakeskeisten suunnittelumenetelmien esittely ja ohjaus muulle ohjelmistokehitystiimille avartaa näiden käsitystä käyttöliittymäsuunnittelun tärkeydestä ohjelmistolle ja tehostaa tiimin yhteistyötä käytettävyyden parantamiseksi [Hod05]. Kun Scrum-tiimin osaaminen käytettävyydestä kasvaa, sen jäsenet ymmärtävät paremmin käyttäjiä ja näiden työtapoja, jolloin ohjelmiston kehityksessä otetaan enemmän huomioon käyttäjän tarpeet [LeM07].

5.2.4 Aikataulut

Vaatimusten aikataulutaminen on tärkeää projektissa, koska se tehostaa projektin etenemistä [Hod05]. Scrumissa tuotelistassa priorisoidut vaatimukset aikataulutetaan julkaisuihin ja pyrähdykseen. Seuraavan julkaisun pyrähdysten sisällöt ovat tuotelistassa tarkalla tasolla aikataulutettu. Seuraavien julkaisujen sisällöt voivat muuttua vapaasti ja ominaisuudet on määritelty pintapuolisesti. Tuotelistan ominaisuuksien aikataulutamisessa pitää ottaa huomioon ominaisuudet, jotka vaativat käyttöliittymäsuunnittelun. Esimerkiksi käyttäjien havainnointiin täytyy varata riittävästi aikaa, jotta sen valmistelu ja tutkimus ehditään tehdä kattavasti ja rauhassa [Hod05]. Ajoissa tehtävät käytettävyyssuunnittelut täytyy sijoittaa muutamaaan edeltävään iteraatioon, jotta niiden suorittamiseen jää riittävästi aikaa.

Aikataulua tehtäessä pitää huomioida se, että vaatimusten prioriteetti vaihtelee ja vaatimukset muuttuvat projektin edetessä. Sen takia liian yksityiskohtaista ja monen vuoden aikataulua ei kannata lähteä tekemään. Aikataulua suunnitellaan niin pitkälle eteenpäin kuin se projektissa tuntuu järkevältä. Turhan työn tekoa pyritään välttämään.

5.2.5 Visio tuotteesta

Kehitettävä ohjelmisto vaatii taustalleen toimivan vision lopullisesta tuotteesta. Visio ohjaa ohjelmistokehitysprojektin aikana vaatimusten priorisointia ja suunnittelua. Ilman visiota kehitetty ohjelmisto saattaa olla kasa irrallisia toimintoja [MeA06]. Sekä käytettävyydelle että ohjelmistolle projektin alussa pitää määritellä oma visio ja tavoite. Niiden avulla varmistetaan, että projektiryhmä on etenemässä oikeaan suuntaan koko projektin ajan. Toteutettavien vaatimusten tulee vastata visiota. Visio auttaa ohjelmistokehitystiimiä ja käyttöliittymäsuunnittelijaa asettamaan tavoitteensa samaan suuntaan.

5.2.6 Ohjelmiston käytettävyyden testaus

Käytettävyyden testausta tehdään käyttöliittymän prototyypivaiheessa, toteutuksen aikana ja toteutuksen jälkeen. Jokainen testaus voi aiheuttaa muutosehdotuksia ohjelmistoon [FNB07a]. Käyttöliittymään saattaa tulla muutosehdotuksia, vaikka vaatimukset eivät olisi muuttuneet taustalla tai toteutusvirheitä ei olisi löydetty. Muutokset johtuvat siitä, että käyttöliittymän toimivuus on riippuvainen käyttöympäristöstä ja käyttäjistä. Erilaisessa käyttöympäristössä testatusta ohjelmistosta voi löytyä aivan uudenlaisia käytettävyyshaasteita, joita ei esimerkiksi prototyypitestauksessa ole huomattu.

Käyttöliittymäsuunnittelijoiden tekemät prototyypit ja käyttäjien palautteet esitellään muulle Scrum-tiimille päivittäisten Scrum-palaverien yhteydessä [FSM06]. Prototyyppien suunnittelun iterointi täytyy sovittaa Scrumin iteraatioihin [FSM06]. Kehitystiimi voi testata prototyypeillä ohjelmiston toimivuutta. Kehitystiimin testauksen aikana löydetään teknisiä haasteita, jotka voivat aiheuttaa muutoksia prototyyppiin. Testauksessa voi myös löytyä asioita, jotka vaikuttavat tekniseen suunnitteluun.

5.2.7 Johdon tuki

Käyttäjakeskeisen suunnitteluprosessin ja Scrumin toimiminen yhdessä vaatii tukea johdolta [FMS06]. Projektijohdon täytyy toimia yhtenäisesti ja tukea tasapuolisesti molempia prosesseja, jotta kumpaakaan prosessia suosita toisen varjolla. Projektijohdon pitää antaa riittävästi tilaa ja vastuuta käyttäjakeskeistä suunnittelua tekeville käytettävyydsiantuntijoille sekä Scrum-tiimiläisille, jotta nämä voivat itsenäisesti parhaaksi näkemillään tavoilla kehittää yhtenäisiä

toimintatapoja [FSM06]. Johdon täytyy varmistaa, että molempien osa-alueiden asiantuntijoille annetaan yhtäläisesti valtaa päättää ohjelmiston kehityksestä [FSM06].

Projektinhallinnan tehtävänä on sovittaa käyttäjakeskeinen suunnittelu ja ohjelmistokehitys sellaiseksi, että kumpaankaan osa-alueeseen ei projektin aikana synny toisen osa-alueen odotettua [FSM06]. Käyttäjakeskeisen suunnittelun ja ohjelmistokehityksen tasapainottaminen on johdon tasolta tärkeää, koska molempia tarvitaan laadukkaan ohjelmiston tuottamisessa [MGH07].

6 Käyttäjälähtöinen Scrum -prosessi

Käyttöliittymäsuunnittelu on tärkeä osa kokonaisvaltaista ohjelmistokehitystä. Toimivan käyttöliittymän avulla varmistetaan, että käyttäjät haluavat käyttää ohjelmaa ja pystyvät tekemään sen avulla tehtäviään. Toisaalta on tärkeää, että ohjelman lähdekoodi on laadukasta eikä ohjelman käyttäminen aiheuta yllätyksiä. Ohjelman laadukkuuteen vaikuttavat ohjelmistoprosessimalli ja sen noudattaminen projektissa.

Käyttäjäkeskeinen suunnittelu ja ohjelmistokehitys voidaan jakaa omiksi linjoiksi ohjelmistokehitysprojektissa [Mil05]. Näin käyttäjäkeskeisessä suunnittelussa pystytään paremmin huolehtimaan oikea-aikaisesta ja tarpeellisista käyttäjätiedoista. Käyttäjäkeskeinen suunnittelu etenee askeleen ohjelmistokehitystä edellä, jotta suunnittelu ehditään tekemään kunnolla ja kehitystiimi saa paremmin tehdyn suunnitelman toteutettavakseen.

6.1 *Prosessinäkökulma*

Käyttäjälähtöinen Scrum on ketterä ja kevyt prosessimalli perinteisen Scrum-prosessin tapaan. Muokattuun prosessimalliin ehdotetut muutokset pyrkivät noudattamaan Scrumin periaatteita. Suurin muutos käyttäjälähtöisen Scrumin ja perinteisen Scrumin välillä on käyttöliittymäsuunnittelijan roolin lisäys Scrum-tiimiin. Tämä lisäys rikkoo Scrumin periaatetta moniosaajatiimistä, jossa kaikki tekevät kaikkea. Kuitenkin edut käyttöliittymäsuunnittelijan lisäämisestä tiimiin ovat paremmat kuin käyttöliittymäsuunnittelun opettaminen koko tiimille. Käyttöliittymäsuunnittelija työskentelee tuoteomistajan, käyttäjän ja tiimin kanssa, ja välittää tietoa näille sidosryhmille. Käyttäjäkeskeistä suunnittelua ei voi täysimittaisena ottaa mukaan Scrum-prosessiin, koska se on liian raskas ja aikaa vievä. Käyttäjäkeskeiseen suunnittelu -prosessiin liittyviä menetelmiä on integroitu käyttäjälähtöiseen Scrumiin varmistamaan hyvä käytettävyys tuotteessa, mutta rikkomatta Scrumin ketteryyttä.

Käyttäjälähtöinen Scrum -prosessimalli esitetään kolmesta eri näkökulmasta. Projektinäkökulmassa prosessia katsotaan koko projektin kannalta. Heti projektin alussa otetaan käytettävyys ja käyttäjät huomioon. Projektin alussa tehdyt valmistelut tukevat käyttöliittymäsuunnittelua ja ohjelmistokehitystä koko projektin ajan. Julkaisunäkökulmassa käydään läpi menetelmät, joita julkaisuittain tarvitsee projektissa tehdä. Käytettävyys on otettava ajoissa huomioon julkaisua suunniteltaessa, jotta käyttöliittymäsuunnittelu ehditään tehdä ajoissa ja tarkasti. Iteraationäkökulmassa käydään läpi tehtävät, joita yhden iteraation aikana tehdään käytettävyyden varmistukseksi.

Tässä tutkielmassa esitetyssä käyttäjälähtöisessä Scrum -prosessissa oletetaan, että projektissa on yksi käyttöliittymäsuunnittelija jatkuvasti Scrum-tiimien käytettävissä. Pyrähdysten mitta on kolme viikkoa ja julkaisu on neljä kertaa vuodessa. Yhteen julkaisuun mahtuu neljä pyrähdystä.

6.2 Projektinäkökulma prosessiin

Projektin alkaessa on tärkeää määrittellä käytettävyyteen vaikuttavia asioita, jotka vaikuttavat koko projektin ajan. Käyttäjäpersoonien määrittely ja luokittelu on tärkeää projektin alussa, koska myöhemmin käyttäjakeskeinen suunnittelu pohjautuu käyttäjäpersooniin. Persoonat ovat käyttäjien perustyyppisiä [Sin08, FNB07c, CRC07]. Ne selvitetään yhdistämällä tietoja markkinatutkimuksista, etnografiatutkimuksista ja kertomuksiin perustuvista havainnoista käyttäjistä [Sin08]. Persoonat tukevat koko projektiryhmää hahmottamaan, millaiseen käyttöön ohjelmistoa toteutetaan [FNB07c]. Persoonien määrittely ja tutkiminen auttaa sekä käyttöliittymäsuunnittelijan että Scrum-tiimin työtä. Esimerkiksi havainnoinnissa käyttäjäpersoonaluokista valitaan havainnoitavat loppukäyttäjät ja käyttöliittymäsuunnittelija pystyy persoonien avulla valitsemaan sopivimmat loppukäyttäjät käyttäjähavainnoiteihin ja käytettävyydesteihin. Scrum-tiimille persoonat taas tuovat käsityksen ohjelmiston loppukäyttäjistä, ja tiimi pystyy kehitystyössä paremmin eläytymään loppukäyttäjän rooliin. Prosessin aikana syntyneet käytettävyyksivaatimukset kirjataan suoraan tuotelistaan, jossa ne priorisoidaan normaalisti muiden vaatimusten tapaan [PaN08].

Käytettävyyksstandardien ja ulkoasuohjeiden määrittely projektin alussa on tärkeää, jotta tietoa käyttöliittymäsuunnittelusta saadaan jaettua koko projektiryhmälle. Vastuuta koko ohjelmiston käytettävyydestä jakautuu koko projektiryhmälle, kun kaikilla on sama päämäärä ohjelmiston lopputuloksesta. Ulkoasu-, käytettävyyks- ja vuorovaikutusmäärittelyt kannattaa laittaa kaikkien helposti saataville ja antaa niihin kaikille muokkausmahdollisuus [LMW05], jotta kaikki projektiin osallistuvat kokevat kehittävänsä ohjelmiston käytettävyyttä. Navigaatiomallin ja tyyliohjeiden teko projektin alussa vähentää riskiä siitä, että ohjelmiston yhtenäisyys katoaisi projektin aikana [FNB07c]. Etukäteen tehtävät yleisluontoiset ohjeet käyttöliittymän rakenteesta ja toiminnasta helpottavat ja nopeuttavat käyttöliittymän suunnittelua projektin aikana, kun nopeasti pitää saada käyttöliittymä suunniteltua toteutusta varten. Erilaisia ohjeita on hyvä tehdä etukäteen juuri sen verran kuin niitä projektin kuluessa tarvitaan, jotta aikaa ei kuluteta turhaan kattavien ohjeistuksien määrittelyyn. Ohjeiden sijoittaminen kaikkien helposti saataville on tärkeää, jotta ohjeita seurataan.

Tärkeää käyttäjälähtöisessä Scrum -prosessissa on, että käyttöliittymäsuunnittelijat ovat tiiviissä yhteistyössä jatkuvasti kehitystiimin kanssa, jotta tieto liikkuu nopeasti tiimien välillä ja kaikki tietävät, missä mennään [NaT08]. Käyttöliittymäsuunnittelija on tiiviissä yhteistyössä tuote-

omistajan ja loppukäyttäjien kanssa, jotta tuoteomistaja voi käyttöliittymäsuunnitelmien avulla tarkemmin priorisoida tuotelistaa ja loppukäyttäjän mielipiteet ja kokemukset ohjelmistosta tulevat huomioiduiksi. Käyttöliittymäsuunnittelija välittää loppukäyttäjien mielipiteet ja palautteen tiedoksi kehitystiimille, jolloin kehitystiimikin saa enemmän lisätietoa ohjelmiston todellisesta käyttökokemuksesta ja käytöstä.

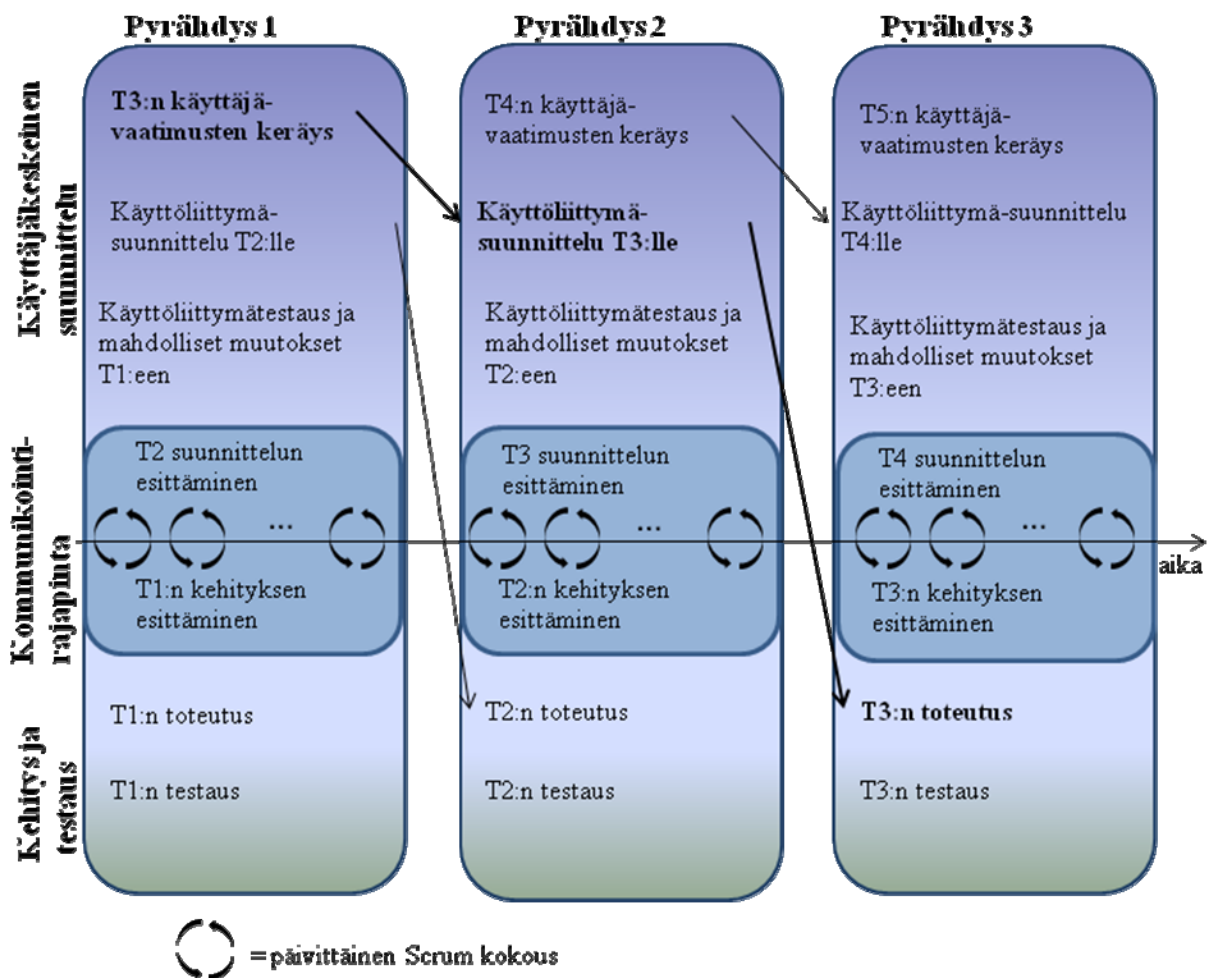
6.3 Julkaisunäkökulma prosessiin

Käyttöliittymän suunnittelulle pitää varata aikaa, jotta se ehditään tehdä ja testata rauhassa. Käyttöliittymät pitää suunnitella ohjelmiston julkaisuittain. Kuvassa 14 on kuvattu vaatimusten siirtyminen pyrähdysten välillä. Kuvassa T1, T2, T3 ja T4 ovat ohjelmiston toiminnallisuuksia ja vaatimuksia. Uuden ohjelmiston osa-alueen suunnittelussa lähdetään ensin keräämään vaatimuksia [BEH06]. Suunniteltavasta osa-alueesta määritellään sen vaikeustaso sekä tärkeimmät käyttäjäroolit. Näiden tietojen pohjalta suunnitellaan käyttäjien havainnointi [BHB04] tai käyttäjien haastattelu. Tutkimuksen suunnitteluun ja aikataulujen sopimiseen täytyy varata aikaa 2–3 viikkoa [BHB04]. Käyttäjien havainnointiin menee viikko. Yksi käyttöliittymäsuunnittelija ehtii viikon aikana havainnoida neljää käyttäjää [BHB04]. Käyttäjien havainnointiin menee yhden pyrähdys verran aikaa.

Toisen pyrähdysen aikana käyttöliittymäsuunnittelija purkaa auki havainnointien tulokset ja muodostaa niiden perusteella tehtävämallit, joista käy ilmi, miten käyttäjän työtehtävä tehdään [BHB04]. Käyttöliittymäsuunnittelija kehittää tehtävämallien mukaan ensimmäisen käyttöliittymän prototyypin, jossa näkyvät karkealla tasolla päätoiminnallisuudet [BEH06]. Alustava suunnitelma esitellään sidosryhmille, ja näiltä kerätään kehitysehdotukset ja -ideat suunnitelman parantamista varten [BHB04]. Käyttäjätarinat ja vaatimukset rakennetaan suunnitelman perusteella. Niiden perusteella suunnitellaan yksityiskohtainen käyttöliittymä [BHB04]. Tehty käyttöliittymämalli on paperiversio, jota on helppo muokata tarvittaessa. Valmiiksi suunniteltu käyttöliittymä testataan oikeilla käyttäjillä, ja siihen tehdään tarvittavat muutokset, ennen kuin se toteutetaan [BEH06]. Kaikki suunnitelmat, joita käyttäjähavainnointien perusteella tehdään, ovat näkyvissä tiimin yhteisen työtilan seinällä, jotta kuka tahansa tiimistä voi tulla kysymään tai antamaan vinkkejä suunniteltavasta käyttöliittymästä [MeA06]. Tehtävämallien teon, testauksen ja käyttöliittymän suunnitteluun menee toinen pyrähdys, jolloin kolmannessa pyrähdyksessä kehittäjätiimi pääsee toteuttamaan ohjelmiston uutta osaa.

Pyrähdysten kuluessa kehitystiimi ja käyttöliittymäsuunnittelija työskentelevät samassa tilassa, jolloin kommunikaatio on mahdollisimman nopeaa ja helppoa kehityksen ja suunnittelun välillä [Mil05]. Päivittäisissä Scrum -kokouksissa tiimi käy läpi edellisen kokouksen jälkeen tekemät

työt, jolloin käyttöliittymäsuunnittelija kertoo omien töidensä edistymisen muulle tiimille, ja samalla kuulee muun tiimin edistymisestä ja esteistä.



Kuva 14: Käyttäjälähtöinen Scrum [muokattu Mil05, FSM08, Sy07].

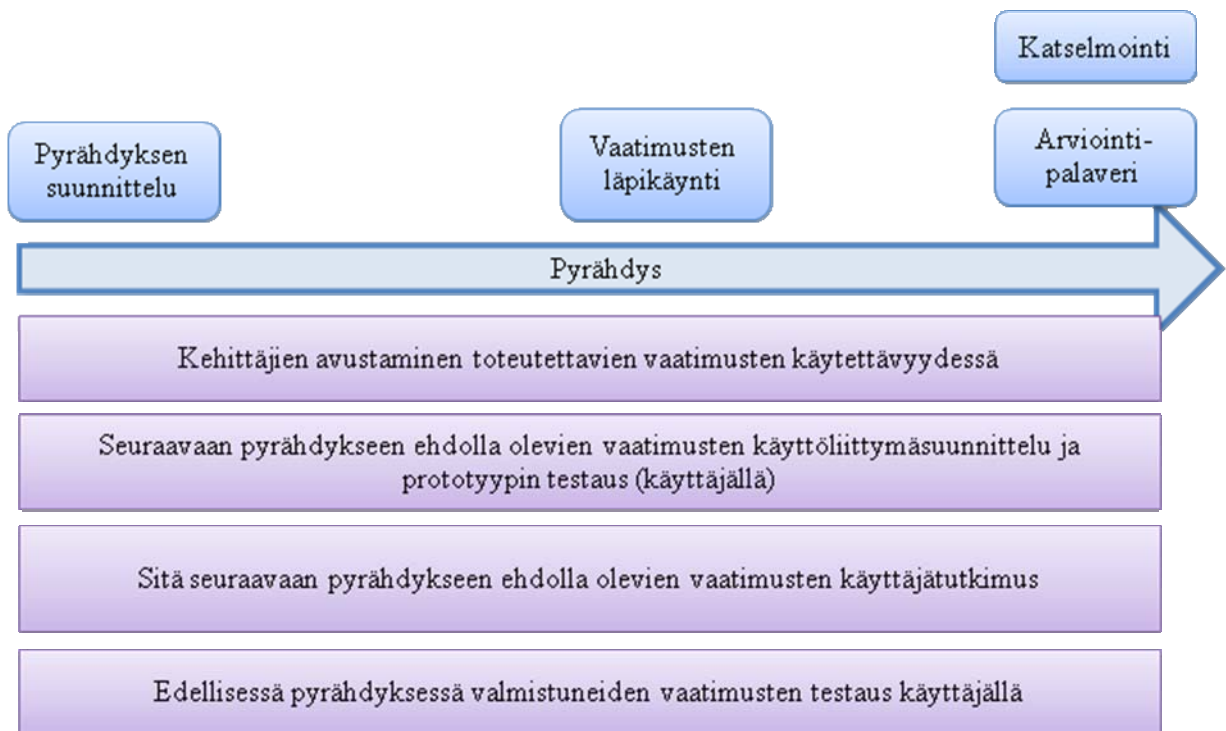
Kehityspyrahdyksen lopussa käyttöliittymälle tehdään hyväksymistestit. Jos hyväksymistestit onnistuvat ennen pyrahdyksen päättymistä, kehitystiimi voi todeta onnistuneensa pyrahdyksessä. Hyväksymistestit voidaan tehdä esimerkiksi käyttämällä heuristisia listoja apuna.

6.4 Iteraationäkökulma prosessiin

Ensimmäisessä iteraatiossa käyttöliittymäsuunnittelu tehdään sellaisille ominaisuuksille, joiden tiedetään menevän kehitykseen seuraavassa iteraatiossa. Suunnitteluiteraatioissa käyttöliittymistä tehdään ensin prototyypit, jotka testataan testikäyttäjillä. Käytettävyyden arviointimenetelmissä huomatu ongelmat korjataan käyttöliittymäprototyyppeihin ja korjaus-testaus -vaihtelua jatketaan niin pitkään, kunnes käyttöliittymäprototyyppi saavuttaa sille asetetut suunnittelutavoitteet. Koska käyttöliittymä testataan prototyyppivaiheessa, suunnittelun tuloksen oikeellisuuteen voidaan luottaa kehitystyön alkaessa [Mil05]. Tarkoituksena on, että suunnittelu tehdään sen verran

myöhäisessä vaiheessa, että suunnittelua ei tehdä turhaan eivätkä vaatimukset ehdi muuttua ennen toteutusvaihetta [Mil05]. Ensimmäisen iteraation aikana kerätään käyttäjiltä tietoa kolmannen iteraation aikana toteutettaviin ominaisuuksiin [Mil05].

Päivittäisten Scrum-kokousten yhteydessä käyttöliittymäsuunnittelija esittelee muille tiimiläisille käyttöliittymäsuunnitelmaansa ja muut voivat kommentoida sitä. Näin muut tiimiläiset näkevät jo ennakkoon, mitä seuraavaan iteraatioon on tulossa kehitettäväksi. Tiimin kehitystyö nopeutuu, koska he tietävät miten ominaisuus toimii ja millaiset vaatimukset siihen liittyvät [Hod05]. Pyrähdysten alussa kehitystiimi voi heti aloittaa ominaisuuden toteutuksen, eikä aikaa kulu ominaisuuden toiminnallisuuden selvittämiseen. Käyttöliittymäsuunnittelija kommentoi kehitysvaiheessa olevia käyttöliittymiä. Näin voidaan jatkuvasti olla varmoja siitä, että käyttöliittymän toteutus vastaa suunnitelmaa [Mil05, NaT08]. Kuvassa 15 on yhden pyrähdysten sisältö käyttäjakeskeisen suunnittelun näkökulmasta. Siinä on nuolen alapuolelle listattu käyttöliittymäsuunnittelijan tehtäviä yhden pyrähdysten aikana. Sinisissä laatikoissa ovat pyrähdysten liittyvät kokoukset, joissa käydään läpi myös käyttöliittymäsuunnittelun tuloksia.



Kuva 15: Yhden pyrähdysten sisältö käyttäjakeskeisen suunnittelun näkökulmasta.

Käyttöliittymäsuunnitelma on valmis aina pyrähdysten suunnittelukokoukseen mennessä, jossa kehitystiimi ja käyttöliittymäsuunnittelija pilkkovat käyttöliittymän ominaisuuksiksi. Ominaisuudet pilkotaan vielä tehtäviksi, joille annetaan työmääräarviot. Tehtävät ovat sellaisia kokonaisuuksia, jotka kehitystiimi saa pyrähdysten aikana tehtyä. Suunnittelukokouksessa

käyttöliittymän toteutukselle alustavasti mietitään jo toteutussuunnitelmaa. Käyttöliittymäsuunnittelija ja tuoteomistaja käyvät läpi julkaisun työlistaa, josta valitaan sillä hetkellä tärkeimmät vaatimukset, jotka tarvitsevat käyttöliittymäsuunnittelun. Pyrähdysten suunnittelukokouksessa työmääräarviot annetaan käyttöliittymäsuunnittelijan tehtäville.

Pyrähdysten keskellä olevassa vaatimusten läpikäynti -kokouksessa käydään alustavasti läpi seuraavaan pyrähdykseen ehdolla olevia vaatimuksia. Jos käyttöliittymäsuunnittelija on saanut tähän kokoukseen tehtyä käyttöliittymistä prototyyppejä, käytetään niitä vaatimusten esittelyn tukena. Kokouksessa tiimi tutustuu ehdolla oleviin vaatimuksiin ja käyttöliittymien prototyyppeihin. Tiimin jäsenet ja käyttöliittymäsuunnittelija käyvät läpi prototyypit, ja keskustelevat käyttöliittymien prototyyppien suunnitteluratkaisuista. Käyttöliittymäsuunnittelija muokkaa prototyyppejä tiimin palautteen perusteella.

Pyrähdysten lopussa olevassa katselmointikokouksessa käyttöliittymäsuunnittelija esittelee pyrähdysten aikana tekemänsä käyttöliittymäsuunnitelmat muille sidosryhmille. Tuoteomistaja voi katselmointikokouksessa kommentoida käyttöliittymäsuunnitelmaa. Pyrähdysten arviointikokouksessa tiimi käy läpi pyrähdysten onnistumiset, ja kohdat, jotka vaativat parannusta. Kokouksessa otetaan huomioon myös käyttöliittymäsuunnittelu ja pyritään sitä parantamaan ja tehostamaan seuraavaan pyrähdykseen.

Taulukossa 5 on eritelty Scrumin eri vaiheissa tapahtuvat käyttäjakeskeisen suunnittelun menetelmät, joiden avulla varmistetaan, että ohjelmiston käytettävyys on hyvä ja että ohjelmisto vastaa käyttäjien tarpeisiin.

Taulukko 5: Käyttäjakeskeisten suunnittelumenetelmien käyttö Scrumin eri vaiheissa.

Pyrähdysten tapahtuma	Käyttäjakeskeisen suunnittelun menetelmät
Projektin ensimmäinen pyrähdys	Käyttäjäroolien määrittäminen ja tyyliohjeiden teko tukemaan käyttöliittymäsuunnittelua ja -toteutusta.
Pyrähdysten suunnittelu	Valmiin käyttöliittymäsuunnitelman käyttö vaatimuksena pyrähdysten työlistassa.
Vaatimusten läpikäynti	Prototyypin läpikäynti käyttöliittymäsuunnittelijan ja kehitystiimin kanssa. Kehitystiimiltä palautetta käyttöliittymästä.
Päivittäinen Scrum	Käyttöliittymäsuunnittelija esittelee lyhyesti edellisen kokouksen jälkeen tehdyt muutokset käyttöliittymäsuunnitelmaan.
Pyrähdysten katselmointi	Tuoteomistaja tekee hyväksymisestäit pyrähdysten aikana valmistuneille ominaisuuksille

	ja päättää hyväksyykö pyrähdysten tuloksen.
Pyrähdysten aikana:	
- Tiedon keruu vaiheessa	Käyttäjätarkkailu ja ääneen ajattelu
- Käyttöliittymän suunnittelun vaiheessa	Kognitiivinen läpikäynti, toimintoanalyysi
- Kehityksen tukeminen	Käyttöliittymäsuunnitelman toteutumisen varmistaminen ja kehitystiimin tukeminen. Käyttöliittymäsuunnitelman muokkaaminen mahdollisten teknisten ongelmien takia.
- Toteutettu käyttöliittymä	Kognitiivinen läpikäynti ja ääneen ajattelu kehitetylle käyttöliittymälle.
Kehityksen valmistuttua	Heuristinen arviointi, jonka suorittaa kehitystiimi.
Projektin tai julkaisun valmistuttua	Kysely ohjelmiston tyytyväisyydestä loppukäyttäjiltä.

6.5 Käyttäjälähtöisen Scrum-prosessin erot Scrumiin

Käyttäjälähtöinen Scrum -prosessi yhdistää käyttäjakeskeisen suunnitteluprosessin vaiheita Scrumiin. Muokattu prosessimalli noudattaa perinteistä Scrumin prosessia, mutta sitä on muutettu käyttäjakeskeisempään suuntaan. Scrum-prosessin keveys ja ketteruus on pyritty säilyttämään käyttäjälähtöisessä Scrumissa.

Käyttäjä otetaan paremmin huomioon käyttäjälähtöisessä Scrumissa kuin perinteisessä Scrumissa. Käyttäjälähtöisessä Scrumissa etukäteen tehtävä suunnittelu tehdään mahdollisimman lähellä pyrähdystä, jolloin suunniteltu ominaisuus kehitetään. Tällä varmistetaan se, että turhaa suunnittelutyötä ei tehdä. Taulukossa 6 on eritelty eroavaisuudet käyttäjälähtöisen Scrumin ja perinteisen Scrumin välillä.

Taulukko 6: Erot Scrumin ja käyttäjälähtöisen Scrumin välillä.

Ero perinteiseen Scrumiin	Etu käyttäjälähtöisessä Scrumissa
Käyttöliittymäsuunnittelija tiimissä	Käyttöliittymäsuunnittelija hallitsee käyttäjakeskeisen suunnittelun ja kommunikoi muun tiimin kanssa päivittäin käyttöliittymäratkaisuihin. Muu tiimi saa keskittyä ohjelmiston kehitykseen ja testaukseen.
Käyttäjän kanssa tiivis yhteistyö	Käyttäjä tai käyttäjän edustaja otetaan tiiviisti mukaan ohjelmiston suunnitteluun ja toteutukseen. Pelkkä asiakas ei riitä takaamaan tuotteen hyödyllisyyttä.
Validointi ennen kehitystä	Käyttäjä varmistaa käyttöliittymän ulkoasun ja toiminnallisuuden käyttöliittymäprototyypin testaamalla

	ennen kehitystä.
Käytettävyyden arviointimenetelmien käyttöönotto projektin eri vaiheissa	Suunnitteluvaiheessa käyttöliittymäsuunnittelija testaa käyttöliittymäprototyypin asiantuntija- ja testimenetelmillä. Toteutuksen valmistuttua tiimi tekee heuristisen arvioinnin ja käyttäjä hyväksymistestit.
Prototyyppien käyttö kehityksen vaatimuksina	Käyttöliittymäsuunnittelun tuloksena ovat prototyypit, jotka siirtyvät vaatimuksina pyrähdysten työlistään.
Käytettävyyden korkea prioriteetti projektissa	Käytettävyys pidetään näkyvillä koko projektin ajan. Tietoutta käytettävydestä kerrotaan kehitystiimille ja muille sidosryhmille.
Käyttöliittymän testaus pyrähdysten jälkeen	Hyväksymistestauksen yhteydessä testataan ja arvioidaan myös käyttöliittymien käytettävyys.

6.6 Muokatun prosessimallin rajoitukset

Käyttäjälähtöinen Scrum -prosessimalli on suunniteltu projektiin, jossa on yksi käyttöliittymäsuunnittelija projektin käytettävissä jatkuvasti koko projektin ajan. Koska projektissa on käytössä ainoastaan yksi käyttöliittymäsuunnittelija, tiimien lukumäärä ei voi olla kovin suuri. Jos käyttöliittymäsuunnittelijan täytyy kommunikoida monen tiimin kanssa, vuorovaikutus ei yhden tiimin osalta ole enää riittävän tiivistä ja tietokatkoksia saattaa esiintyä käyttöliittymäsuunnittelijan ja tiimin välillä. Käyttäjälähtöisessä Scrum -prosessimallissa yksi käyttöliittymäsuunnittelija tekee kaikki käyttäjakeskeisen suunnittelun työvaiheet itse. Vaatimusten keräysvaiheessa ja toiminnallisuuksien määrittelyssä käyttöliittymäsuunnittelija tekee yhteistyötä tuoteomistajan kanssa. Käyttöliittymän suunnitteluvaiheessa tiimit osallistuvat käyttöliittymän arviointiin, jotta käyttöliittymäsuunnittelija saa tukea omaan työhönsä.

Käyttäjälähtöinen Scrum -prosessimalli on suunniteltu pitkäkestoiseen useamman vuoden kestäväan projektiin, jossa ei ole mahdollista tehdä kattavaa käyttäjakeskeistä suunnittelua ennen toteutuksen aloittamista. Vaatimukset, joiden prioriteetti on korkea, otetaan käyttöliittymäsuunnitteluun. Alhaisella prioriteetilla olevien vaatimusten järjestys muuttuu tuotteen työlistassa, joten niiden suunnittelua ei kannata aloittaa liian aikaisessa vaiheessa. Jos kehitettävä ohjelmisto on pieni eikä sisällä kymmeniä käyttöliittymiä, suurimman osan käyttäjakeskeisestä suunnittelusta voidaan mahdollisesti tehdä ennen toteutusvaihetta. Pienessä ja lyhyessä ohjelmistokehitysprojektissa vaatimusten prioriteetti ja käyttöliittymäsuunnitelmat eivät ehdi muuttua useita kertoja projektin aikana.

Käyttäjälähtöinen Scrum -prosessimalli on tarkoitettu ohjelmistokehitysprojekteihin, joissa käyttöliittymällä ja ohjelmiston käytettävyydellä on suuri merkitys ja painoarvo ohjelmiston kehityksessä. Käyttöliittymät kehitettävissä tuotteissa ovat vuorovaikutteisia. Niiden käyttö

pitää olla tehokasta ja mielekästä loppukäyttäjän työnteon kannalta. Käyttäjälähtöinen Scrum tuo lisää työtä projektiin, mutta varmistaa johdonmukaisesti koko projektin ajan hyvän käytettävyyden lopputuotteessa.

6.7 Jatkokehitystä käyttäjälähtöiseen Scrum -prosessimalliin

Haasteena tässä tutkielmassa esitetyssä käyttäjälähtöisessä Scrumissa on saada tuotelista priorisoitua ainakin julkaisutasolla sellaiseksi, että seuraava julkaisu tiedetään ennen julkaisun teon aloittamista. Scrumissa vaatimusten muuttuminen ja niiden prioriteetin vaihtuminen on sallittua koko projektin ajan, mutta käyttäjakeskeinen suunnittelu tarvitsee ajoissa tiedot tulevissa pyrhdyksissä tehtävistä vaatimuksista, jotta käyttöliittymäsuunnittelu ja -testaus ehditään tehdä rauhassa. Julkaisun pyrhdyksissä toteutetaan siihen julkaisuun suunnitellut vaatimukset. Julkaisun sisältö täytyy olla sidottu sen takia, että turhaa käyttöliittymäsuunnittelua ja -testausta ei tehdä ja että käyttöliittymäsuunnitelma osataan aloittaa oikeaan aikaan ja saada valmiiksi juuri ennen toteutusta. Jos tuotelistan priorisointi vaihtelee pyrhdyksittäin paljon, etukäteissuunnittelu on haastavaa, ja se aiheuttaa ongelmia käyttöliittymäsuunnittelulle ja -testaukselle. Suurin ongelma käyttöliittymäsuunnittelulle on turhan työn tekeminen, joka aiheuttaa kustannuksia projektiin. Käyttöliittymäsuunnitelman pitää siirtyä mahdollisimman nopeasti käyttöliittymäsuunnittelusta toteutukseen, jotta vältetään turhaa työtä. Turha työ tulee siitä, kun ajan kuluessa viimeistely käyttöliittymäsuunnitelma muuttuu. Tiimi unohtaa yksityiskohtia käyttöliittymäsuunnitelmasta ja saattaa tehdä muutoksia, jotka rikkovat alkuperäistä toimintalogiikkaa. Esimerkiksi syyt joidenkin käyttöliittymäratkaisujen takana unohtuvat ja käyttöliittymää muutetaan ilman perusteellista testausta.

Loppukäyttäjän saaminen jokaiseen pyrhdykseen aiheuttaa omia haasteita käyttäjälähtöiselle Scrumille. Projektiin pitää löytää muutamia sellaisia loppukäyttäjää, jotka pystyisivät osallistumaan ohjelmiston testaukseen yhtenä päivänä pyrhdyksen aikana. Muutoin ohjelmiston todellisen käytettävyyden tasoa on vaikea tietää. Toinen vaihtoehto on käyttää loppukäyttäjän testausta kerran julkaisun aikana, jolloin loppukäyttäjää ei tarvita niin usein, mutta silloin tulee muita ongelmia. Jos loppukäyttäjään ollaan yhteydessä kerran julkaisun aikana, täytyy silloin seuraava julkaisu ja pyrhdykset olla jo staattisia, jotta ei tehdä turhaa käyttäjätutkimusta. Toisaalta välttämättä kaikkia prototyyppjeä ei ehditä testaamaan loppukäyttäjällä, jolloin suunnitelmat menevät suoraan käyttöliittymäsuunnittelijalta kehitystiimille. Kolmanneksi, jos valmiin ohjelmiston testauksessa loppukäyttäjää löytää käytettävyyso ongelmia, niiden korjaamiseen ei välttämättä ole aikaa seuraavassa julkaisussa, vaan korjaukset siirtyvät tulevaisuuteen. Loppukäyttäjää voidaan tarvittaessa kiireellisissä tilanteissa korvata loppukäyttäjän edustajalla, eli henkilöllä, joka tuntee

käyttäjät ja heidän työtapansa. Tällaisia henkilöitä on esimerkiksi tuoteomistaja, kouluttajat, myyjät ja konsultit, jotka työskentelevät lähellä asiakasrajapintaa.

7 Yhteenveto

Tutkielmassa on tutkittu käyttäjakeskeisen suunnittelun menetelmien integroimista Scrum-prosessimalliin. Nykyään uudet ohjelmistot täytyy saada nopeasti markkinoille ja käyttökokemus on yksi ohjelmistojen erottava tekijä, joten sekä ketteriin menetelmiin että käyttäjakeskeiseen suunnitteluun on panostettava, jotta kehitettävä tuote onnistuu ja myy.

Ketterät prosessimallit pyrkivät välttämään kattavaa etukäteen tehtävää suunnittelua. Niiden päätarkoituksena on tuottaa nopeasti laadukas ohjelmisto asiakkaalle ja käyttäjälle. Scrum on ketterä prosessi, joka keskittyy projektinhallintaan. Se pyrkii tuomaan projektin epäkohdat ilmi ja tehostamaan tiimien työskentelyä joka pyrähdysten aikana. Scrumissa korkean tason vaatimusmäärittelyn tekee tuoteomistaja, joka priorisoi tuotelistassa olevia vaatimuksia. Priorisoidusta tuotelistasta tiimi valitsee tärkeimmät vaatimukset seuraavaan pyrähdykseen työnalle. Tiimi tekee pyrähdysten työlistaan valituille tehtäville kevyen suunnittelun, jonka perusteella toteutus aloitetaan. Käyttöliittymäsuunnittelua tai käytettävyyttä ei Scrumissa erikseen mainita lainkaan, joten on mahdollista, että toteutusvaiheessa käyttöliittymä syntyy kehitystyön sivutuotteena.

Käytettävyys on laatuattribuutti, jonka avulla voidaan mitata ohjelmiston toimivuutta käyttäjän näkökulmasta. Nielsen jakaa käytettävyyden viiteen eri osa-alueeseen: opittavuuteen, tehokkuuteen, muistettavuuteen, virheettömyyteen ja tyytyväisyyteen. Jokaisella osa-alueella on omat mittarinsa, joiden avulla pystytään varmistamaan käytettävyyden taso ohjelmistossa. Käytettävyyden arviointimenetelmistä voidaan erotella kaksi erilaista tapaa testata käytettävyyttä. Asiantuntijamenetelmissä käyttöliittymäsuunnittelija arvioi ilman käyttäjää ohjelmiston käytettävyyttä ja testimenetelmissä ohjelmiston käyttäjän käytöstä tutkitaan ja tehdään arviointia käyttäytymisen pohjalta ohjelmiston käytettävyydestä.

Käyttäjakeskeinen suunnitteluprosessi on määritelty ISO 13407 -standardissa. Käyttäjakeskeisessä suunnittelussa käyttäjä on suunnittelun keskipiste. Siinä pyritään ymmärtämään käyttäjää, tämän tarpeita ja työympäristöä ja niiden pohjalta suunnittelemaan ohjelmisto, joka on käytettävä ja hyödyllinen. Käyttäjakeskeisessä suunnitteluprosessissa painotetaan etukäteen tehtävää kattavaa tutkimusta ja suunnittelua. Suunnittelu tehdään lyhyissä iteraatioissa ja sen käytettävyyttä testataan jatkuvasti eri käytettävyyden arviointimenetelmillä. Kun ohjelmiston käyttöliittymä on suunniteltu ja testattu, se on valmis siirtymään kehitykseen.

Prosesseissa on tiettyjä ominaispiirteitä, jotka aiheuttavat haasteita prosessien yhdistämisessä. Scrumissa ei suosita laajojen suunnitelmien tekoa ennen toteutusta ja ohjelmisto validoidaan kehityksen jälkeen. Käyttäjakeskeisessä suunnittelussa halutaan ymmärtää mahdollisimman

hyvin käyttäjää ja tämän tarpeita ennen kuin mitään suunnitellaan ja toteutetaan. Käyttöliittymäsuunnitelmat validoidaan jatkuvasti käyttäjillä, jotta koko suunnittelun ajan tiedetään suunnitelman noudattavan paremmin käyttäjän tarpeita.

Scrumissa ja käyttäjäkeskeisissä suunnitteluprosesseissa samankaltaisuuksia ovat prosessien iteratiivinen luonne, asiakkaan tai käyttäjän tärkeä rooli koko kehitysprojektin ajan sekä tiivis tiimityöskentely. Käyttäjäkeskeisen suunnitteluprosessin yhdistäminen Scrumiin lähtee yhteisten toimivien asioiden tarkastelusta ja painottamisesta muokatussa käyttäjälähtöisessä Scrum -prosessimallissa. Scrumin ja käyttäjäkeskeisten suunnitteluprosessien eroavaisuuksissa täytyy tehdä kompromisseja, jotta saadaan joustava ja mukautuva prosessimalli, jossa yhdistyy sekä Scrumin että käyttäjäkeskeisten suunnitteluprosessien hyvät puolet.

Tutkielman tuloksena on, että käyttäjälähtöisessä Scrum -prosessimallissa käyttäjäkeskeinen suunnittelu ja toteutus on erotettu toisistaan omiksi linjoiksi, joilla on yhtä pitkät iteraatiot. Käyttöliittymäsuunnittelija kuuluu kehitystiimiin ja päivittäisissä Scrum -palavereissa kehitystiimi kuulee käyttäjäkeskeisen suunnittelun edistymisestä ja käyttöliittymäsuunnittelija tietää missä mennään kehityksessä. Tärkeää käyttäjälähtöisessä Scrum-prosessimallissa on tiivis yhteistyö käyttöliittymäsuunnittelijan ja kehitystiimin välillä. Kehitystiimi kommentoi käyttöliittymäsuunnitelmia ja käyttöliittymäsuunnittelija tukee toteutusta. Käyttäjälähtöinen Scrum -prosessimalli nopeuttaa ja parantaa kehitysvaihetta, koska suunnittelu on tehty tavallista Scrum-prosessia tarkemmin ja kehitystiimillä on jo ennen pyrähdyn alkua selkeä kuva toteutettavasta ominaisuudesta. Käyttäjälähtöinen Scrum keskittyy enemmän käyttäjään eikä pelkästään asiakkaaseen. Käyttöliittymästä tehtyjä suunnitelmia, eli prototyyppijä, testataan tiiviisti käyttäjillä tai käyttäjien edustajilla. Näin pystytään jo ennen toteutusvaihetta varmistamaan toteutettavan ominaisuuden käytettävyyden.

Lähteet

- ASR02 Abrahamsson, P., Salo, O., Ronkainen, J., Warsta J., Agile software development methods, review and analysis. Otamedia Oy, Espoo, 2002.
- Arm04 Armitage, J., Are agile methods good for design? *Interactions*, 11, 1 (2004), sivut 14-23.
- BaJ01 Bass, L., John, B., E., Supporting usability through software architecture. *Computer*, 34, 10 (2001), sivut 113-115.
- Bar08 Barnum, C., M., What do you mean when you say "usability"? *eLearn Magazine*, 2008, 2 (2008).
- BDS99 Beedle, M., Devos, M., Sharon, Y., Schwaber, K., Sutherland, J., SCRUM: An extension pattern language for hyperproductive software development. *Pattern Languages of Program Design 4*, Addison-Wesley Software Patterns Series, 1999.
- Bec99 Beck, K., Embracing change with extreme programming. *Computer*, 32, 10 (1999), sivut 70-77.
- Bec01 Beck, K., Beedle, M., Bennekum A. et al. Manifesto for Agile Software Development, <http://agilemanifesto.org/>, 2001. [29.3.2009]
- BeH99 Beyer, H., Holtzblatt, K., Contextual design, *Interactions*, 6, 1 (1999), sivut 32-42.
- BEH06 Breen, S., Enkerud, T., Husoy, K., Applying Usability Engineering in ABB. *Proc. of the 4th Nordic conference on Human-computer interaction*, ACM International Conference Proceedings Series, Oslo, Norja, 2006, sivut 491-492.
- Bev95 Bevan, N., Usability is Quality of Use, *Proc. of the 6th International Conference on Human Computer Interaction*, Yokohama, Japani, 1995.
- BHB04 Beyer, H., Holtzblatt, K., Baker, L., An Agile Customer-Centered Method: Rapid Contextual Design. *Proc. 4th conference on extreme programming and agile methods*, Calgary, Canada, , elokuu 2004, sivut 50-59.
- BoJ03 Bosch, J., Juristo, N., Designing software Architecture for Usability. *Proc. of the 25th International Conference on Software Engineering*, IEEE Computer Society, Portland, USA, toukokuu 2003, sivut 757-758.

- Cas97 Casaday, G., Notes on a Pattern Language for Interactive Usability. *Proc. Conference on Human Factors in Computer Systems*, Atlanta, USA, maaliskuu 1997, sivut 289-290.
- ChC07 Chow, T., Cao, D-B., A survey study of critical success factors in agile software projects. *The Journal of Systems and Software*, 81, 2 (2007), sivut 961-971.
- Coc07 Cockburn, A., *Agile Software Development – The Cooperative Game*. Pearson Education Inc., USA, 2007.
- Coh07 Cohn, M., *User Stories Applied for Agile Software Development*. Pearson Education, Inc., USA, 2007.
- CoL03 Constantine L. L., Lockwood L. A. D., Usage-Centered Software Engineering: An Agile Approach to Integrating Users, User Interfaces, and Usability into Software Engineering Practice. *Proc. of the 25th International Conference on Software Engineering*, International Conference on Software Engineering, Portland, USA, 2003, sivut 746-747.
- Con01 Constantine, L., L., Process Agility and Software Usability: Toward Lightweight Usage-Centered Design. *Software Development*, 9, 6 (2001).
- CRC07 Cooper A., Reinmann R. ja Cronin D., *About Face 3 - The Essentials of Interaction Design*. Wiley Publishing, Inc., Indianapolis, USA, 2007.
- CSM06 Chamberlain, S., Sharp, M., Maiden, N., Towards a Framework for Integrating Agile Development and User-Centred Design. *Proc. of the 7th International Conference on extreme programming and agile processes in software development*, Oulu, Suomi, kesäkuu 2006, sivut 143-153.
- DBB93 Dayton, T., Barr, B., Burke, P., A., et Al., Skills needed by user-centred design practitioners in real software development environments: report on the CHI'92 workshop. *SIGCHI Bulletin*, 25, 3 (heinäkuu 1993), sivut 16-31.
- DeL03 DeMarco, T., Lister, T., Risk Management during Requirements. *IEEE Software*, 20, 5 (2003), sivut 99-101.
- DZN07 Düchting, M., Zimmermann, D., Nebe, K., Incorporating User Centered Requirement Engineering into Agile software Development, *Proc. of the 12th International Conference Human-Computer Interaction*, Peking, Kiina, heinäkuu 2007, sivut 58-67.

- FNB07a Ferreira J., Noble J., Biddle R., Agile Development Iterations and UI Design. *Proc. of the AGILE 2007*, Washington, USA, Elokuu 2007, sivut 50-58.
- FNB07b Ferreira, J., Noble, J., Biddle, R., Up-Front Interaction Design in Agile Development. *Proc. of the 8th International Conference on Agile Processes in Software Engineering and Extreme Programming*, Como, Italia, kesäkuu 2007, sivut 9-16.
- FNB07c Ferreira, J., Noble, J., Biddle, R., Interaction Designers on eXtreme Programming Teams: Two Case Studies from the Real World. *Proc. of the 5th New Zealand Computer Science Research Student Conference*, Hamilton, New Zealand, 2007.
- FSM08 Fox, D., Sillito, J., Maurer, F., Agile Methods and User-Centered Design: How These Two Methodologies Are Being Successfully. *Proc. of Agile 2008 Conference*, IEEE computer society, Toronto, Kanada, elokuu 2008, sivut 63-72.
- Gar03 Garrett, J. J., *The Elements of User Experience*. AIGA/NewRiders Publishing, New York, USA, 2003.
- GoL85 Gould J. D., Lewis, C., Designing for usability: key principles and what designers think. *Communications of the ACM*, 28, 3 (maaliskuu 1985).
- Gou88 Gould, J. D., How to design usable systems. Teoksessa *Human-Computer Interaction*, Editor Elsevier Science Publishers, Amsterdam, Alankomaat, 1988, sivut 757-789.
- HAH05 Helms, J. W., Arthur, J. D., Hix, D., Hartson, H. R., A field study of the wheel - a usability engineering process model. *The Journal of Systems and Software*, 79, 6 (2005), sivut 841-858.
- HiC01 Highsmith J., Cockburn A., Agile Software Development: The Business of Innovation. *Computer*, 34, 9 (syyskuu 2001), sivut 120-122.
- Hod05 Hodgetts, P., Experiences Integrating Sophisticated User Experience Design Practices into Agile Processes. *Proc. of the Agile Development Conference*, IEEE Computer Society, Denver, USA, heinäkuu 2005, sivut 235-242.
- Hol05 Holzinger, A., Usability Engineering Methods for Software Developers. *Communications of the ACM*, 48, 1 (tammikuu 2005), sivut 71-74.
- ISO98 International Standards Organisation & Internal Electrotechnical Commission, *ISO 9241-11 Guidance on Usability*. Technical report, 1998.

- ISO99 International Standards Organisation & Internal Electrotechnical Commission, *ISO 13407 Human-centered design processes for interactive systems*. Technical report, 1999.
- Jef01 Jeffries, R., What is extreme programming? *XP Magazine*, <http://www.xprogramming.com/xpmag/whatisxp.htm>, 8.11.2001. [29.3.2009]
- JMS07 Juristo, N., Moreno, A. M., Sanchez-Segura, M-I., Analysing the impact of usability on software design. *The Journal of Systems and Software*, 80, 9 (syyskuu 2007), sivut 1506-1516.
- JMW91 Jeffries, R., Miller, J., Wharton, C., Uyeda, K. M., User Interface Evaluation in the Real World: A comparison of four techniques. *Proc. of the SIGCHI conference on Human factors in computing systems*, New Orleans, USA, 1991, sivut 119-124.
- Kan03 Kane, D., Finding a Place for Discount Usability Engineering in Agile Development: Throwing Down the Gauntlet. *Proc. of the Agile Development Conference*, Salt Lake City, USA, kesäkuu 2003, sivut 40-46.
- Kar97 Karat, J., Evolving the Scope of User-Centered Design. *Communications of the ACM*, 40, 7 (heinäkuu 1997), sivut 33-38.
- KaK03 Karat, J., Karat, C. M., The evolution of user-centered focus in the human-computer interaction field, *IBM Systems Journal*, 42, 4 (2003).
- Kuj98 Kujala, S., Käyttäjätutkimukset vaatimusmäärittelyn apuna. *Systemityö*, 5, 4 (1998), sivut 10-12.
- LeM07 Lee, J. C., McCrickard, D. S., Towards Extreme(ly) Usable Software: Exploring Tensions Between Usability and Agile Software Development. *Proc. in AGILE 2007 Conference*, Washington, USA, elokuu 2007, sivut 59-71.
- LMW05 Leacock, M., Malone, E., Wheeler, C., Implementing a Pattern Library in the Real World: A Yahoo! Case Study, *IA Summit 2005*, Quebec, Canada, maaliskuu 2005.
- May99 Mayhew, D., J., Strategic Development of the Usability Engineering Function. *Interactions*, 6, 5 (1999), sivut 27-34.
- McM05 McInerney, P., Maurer, F., UCD in Agile Projects: Dream Team or Odd Couple? *Interactions*, 12, 6 (2005), sivut 19-23.

- MeA06 Meszaros, G., Aston, J., Adding Usability Testing to an Agile Project. *Proc. of AGILE 2006 Conference*, Minneapolis, USA, heinäkuu 2006.
- MGH07 Memmel, T., Gundelsweiler, F., Reiterer, H., Agile Human-Centered Software Engineering. *Proc. of the 12th International Conference on Human-Computer Interaction*, Peking, Kiina, heinäkuu 2007.
- Mil05 Miller, L., Case Study of Customer Input for a Successful Product. *Proc. of Agile 2005 Conference*, Denver, USA, heinäkuu 2005, sivut 225-234.
- MVS05 Mao, J-Y, Vredenburg, K., Smith, P., Carey, T., The State of User-Centered Design Practice. *Communications of the ACM*, 48, 3 (2005), sivut 105-109.
- NaT08 Najafi, M., Toyoshiba, L., Two Case Studies of User Experience Design and Agile Development. *Proc. of Agile 2008 Conference*, Toronto, Kanada, elokuu 2008, sivut 531-536.
- Nie92a Nielsen, J., The Usability Engineering Life Cycle. *Computer*, 25, 3 (maaliskuu 1992).
- Nie92b Nielsen, J., Finding Usability Problems Through Heuristic Evaluation. *Proc. of Conference on Human Factors in Computing Systems*, Monterey, USA, toukokuu 1992.
- Nie93a Nielsen, J., Iterative User-Interface Design. *Computer*, 26, 11 (marraskuu 1993).
- Nie93b Nielsen, J., *Usability Engineering*. Academic Press, 1993.
- Nie94 Nielsen, J., Enhancing the Explanatory Power of Usability Heuristics. *Proc. of Human factors in Computing Systems*, Boston, USA, huhtikuu 1994.
- Nie02 Nielsen, J., Ten Usability Heuristics. http://www.useit.com/papers/heuristic/heuristic_list.html, 2005. [29.3.2009]
- NiM90 Nielsen, J., Molich, R., Heuristic Evaluation of User Interfaces. *Proc. of the ACM CHI'90 Human factors in Computing systems conference*, Seattle, USA, huhtikuu 1990, sivut 249-256.
- PaN08 Paelke, V., Nebe, K., Integrating Agile Methods for Mixed Reality Design Space Exploration. *Proc. of Designing interactive systems conference*, Cape Town, Etelä-Afrikka, helmikuu 2008, sivut 240-249.

- Pat02 Patton, J., Hitting the target: adding interaction design to agile software development. *Proc. of Conference on Object Oriented Programming Systems Languages and Applications*, Seattle, USA, marraskuu 2002.
- PLR07 Parsons, D., Lal, R., Ryu, H., Software Development Methodologies, Agile Development and Usability Engineering. *Proc. 18th Australasian Conference on Information Systems*, Toowoomba, Australia, joulukuu 2007.
- Rii98 Riihiaho, S., Käytettävyyden arviointi ilman käyttäjiä. *Systeemityö*, 5, 4 (1998).
- RiJ00 Rising, L., Janoff, N., S., The Scrum Software Development Process for Small Teams. *IEEE Software*, 17, 4 (2000), sivut 26-32.
- RRH00 Rosenbaum, S., Rohn, J., A., Humburg, J., A Toolkit for Strategic Usability: Results from Workshops, Panels, and Surveys. *Proc. of the SIGCHI, conference on Human factors in computing systems*, Haag, Alankomaat, huhtikuu 2000.
- Sam07 Sampson, F., Who Said "Usability Is Free"? *Interactions*, 14, 4 (2007), sivut 10-11.
- SeA03 Seffah, A., Andreevskaia, A., Empowering Software Engineers in Human-Centered Design. *Proc. of the 25th International Conference on Software Engineering*, Portland, Oregon, toukokuu 2003, sivut 653-658.
- SeM04 Seffah, A., Metzker, E., The obstacles and myths of usability and software engineering. *Communications of the ACM*, 47, 12 (2004), sivut 71-76.
- ScB02 Schwaber K., Beedle, M., *Agile Software Development with Scrum*. Prentice Hall, 2002.
- Sch97 Schwaber, K., Scrum Development Process. *OOPSLA Business Object Design and Implementation Workshop*, Atlanta, USA, lokakuu 1997.
- Sin08 Singh, M., U-Scrum: An Agile Methodology for Promoting Usability. *Proc. Agile 2008 Conference*, IEEE Computer Society, Toronto, Kanada, elokuu 2008, sivut 555-560.
- Som04 Sommerville, I., *Software Engineering 7*. Pearson Education, 2004.
- Sut04 Sutherland, J., Agile Development: Lessons Learned from the First Scrum. *Cutter Agile Project Management Advisory Service: Executive update*, 5, 20 (2004), sivut 1-4.

- Sut05 Sutherland, J., Future of Scrum: Parallel Pipelinig of Sprints in Complex Projects. *Proc. of the Agile Development Conference*, Denver, USA, heinäkuu 2005.
- Sy07 Sy, D., Adapting Usability Investigations for Agile User-Centered Design. *Journal of Usability Studies*, 2, 3 (toukokuu 2007), sivut 112-132.
- WBJ97 Wilson, S., Bekker, M., Johnson, P., Johnson,H., Helping and Hindering User Involvement – a Tale of Everyday Design. *Proc. of the SIGCHI conference on Human factors in computing systems*, Atlanta, USA, maaliskuu 1997, sivut 178-185.
- WiH90 Wixon, D., Holtzblatt, K., Contextual Design: An Emergent view of System Design. *Proc. of the SIGCHI conference on Human factors in computing systems*, Seattle, USA, huhtikuu 1990, sivut 329-336.
- VMS02 Vredenburg, K., Mao, J-Y., Smith, P., W., Carey, T., A Survey of User-Centered Design Practice. *Proc. of the SIGCHI conference on Human factors in computing systems*, Minneapolis, USA, huhtikuu 2002, sivut 471-478.
- WRH02 Wixon, D., R., Ramey, J., Holtzblatt, K., et Al., Usability in Practice: Field Methods evolution and revolution. *Proc. of Conference on Human factors in Computing Systems*, Minneapolis, USA, huhtikuu 2002, sivut 880-884.