Dani Vertanen

ASIAKKAAN ROOLI ERI OHJELMISTOKEHITYSMENETELMISSÄ

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2018

TIIVISTELMÄ

Vertanen, Dani
Asiakkaan rooli eri ohjelmistokehitysmenetelmissä
Jyväskylä: Jyväskylän yliopisto, 2018, 29 s.
Tietojärjestelmätiede, kandidaatintutkielma
Ohjaaja: Seppänen, Ville

Asiakas on merkittävässä roolissa ohjelmistokehitysprosessissa. Ohjelmistoa tilatessa asiakas ei välttämättä tiedä tai osaa kertoa selvästi mitä se tarvitsee ja haluaa. Ohjelmiston määrittelyt muuttuvat usein kehitysvaiheessa, ja asiakkaan osallistuminen mahdollistaa vaatimusten muokkaamisen sekä asiakasta tyydyttävän lopputuloksen saavuttamisen. Ohjelmistokehityksessä voidaan käyttää useita eri menetelmiä. Nämä menetelmät ovat keskenään erilaisia, ja asiakasta osallistetaan eri menetelmissä eri tavoin pitkin kehitysprosessia. Tähän tutkielmaan on valittu neljä ohjelmistokehitysmenetelmää, jotka ovat vesiputousmalli, prototyyppimenetelmä, RUP ja Scrum. Tutkielmassa vertaillaan näitä menetelmiä ja erityisesti asiakkaan roolia ja osallistamista niissä. Kirjallisuuskatsauksena toteutetussa tutkielmassa pyritään vastaamaan tutkimuskysymykseen *"Miten asiakkaan osallistaminen ilmenee eri ohjelmistokehitysmenetelmissä?"*. Vesiputousmallissa asiakkaan osallistaminen painottuu prosessin alun määrittelyvaiheeseen. Prototyyppimenetelmässä asiakas osallistuu ohjelmiston kehitykseen pääasiassa prototyypin tarkastelun ja testaamisen kautta. RUP-menetelmässä asiakas on vuorovaikutuksessa kehittäjän kanssa iteraatioiden ja käyttäjätarinoiden kautta. Asiakkaan osallistaminen ilmenee eniten Scrum-menetelmässä, jossa asiakas ja kehittäjä ovat jatkuvassa vuorovaikutuksessa keskenään ja voivat näin vastata muuttuviin vaatimuksiin.

Asiasanat: ohjelmistokehitys, asiakkaan rooli, asiakkaan osallistaminen, ketterät menetelmät

ABSTRACT

Vertanen, Dani
The Role of a Customer in Software Development Methodologies
Jyväskylä: University of Jyväskylä, 2018, 29 pp.
Information System Science, Bachelor's thesis
Supervisor(s): Seppänen, Ville

The role of a customer is important in system development process. In some cases, the customer is unsure of its needs or cannot describe the requirements when the new software is ordered. The requirements of a software can usually change during the development process and customer involvement ensures that the requirements can be changed, and a successful result can be achieved. There are many different methodologies to be used in software development. These methodologies are different, and the level of customer involvement is also different during the development process. This study compares the role of a customer and the level of customer involvement in waterfall, prototyping, RUP and Scrum methodologies. The study has been carried out as a literature review and the research question is "*How customer involvement appears in different software development life cycles?*". In waterfall methodology customer is involved mainly in the early stage's requirements phase. In prototyping, customer involvement is related to observing and testing of the software prototype. RUP involves customer with interaction with the developer through iterations and user stories. Customer involvement is on the highest level in Scrum, where customer and developer are in continuous interaction to answer the changing requirements.

Keywords: software development, role of a customer, customer involvement, agile methods

KUVIOT

KUVIO 1 Tietojärjestelmien kehitysympäristö 10
KUVIO 2 Vesiputousmalli..... 14
KUVIO 3 Asiakkaan osallistaminen 20

TAULUKOT

TAULUKKO 1 Mukautetun ja paketti-kehitysympäristön merkittävät erot 12
TAULUKKO 2 Asiakkaan osallistaminen eri ohjelmistokehitysmenetelmissä . 23

SISÄLLYS

TIIVISTELMÄ
ABSTRACT
KUVIOT
TAULUKOT

1 JOHDANTO 6

2 ASIAKKAAN ROOLI OHJELMISTOKEHITYKSESSÄ 8
2.1 Ohjelmistokehitys 8
2.2 Ohjelmistokehitysprosessin osapuolet 9
2.2.1 Asiakas 9
2.2.2 Kehittäjä 10
2.2.3 Osapuolten välinen yhteistyö 10

3 MENETELMÄT 13
3.1 Vesiputousmalli 13
3.2 Prototyyppimenetelmä 15
3.3 RUP 16
3.4 SCRUM 17

4 ASIAKKAAN ROOLI MENETELMISSÄ 19
4.1 Vesiputousmalli 19
4.2 Prototyyppimenetelmä 20
4.3 RUP 21
4.4 SCRUM 22
4.5 Vertailu 22

5 YHTEENVETO 25

LÄHTEET 27

1 JOHDANTO

Asiakkaan rooli on merkittävä osa ohjelmistokehitysprosessia. Boehmin (2002) mukaan asiakkaan osallistuminen tuotteen kehitykseen parantaa sen lopputulosta. Asiakaslähtöisen ajattelutavan mukaan on tärkeää, että ohjelmisto todella täyttää asiakkaan tarpeet ja ratkaisee ongelmia. Hyvään lopputulokseen pääseminen edellyttää asiakaskeskeistä strategiaa sekä asiakkaan osallistamista ohjelmiston kehitykseen. Asiakkaan ja kehittäjän välinen vuorovaikutus erityisesti teknisten asioiden osalta on keskeisessä osassa kehitysprosessissa. (Saiedian & Dale, 2000). Asiakkaan osallistaminen ilmenee eri tavoin erilaisissa ohjelmistokehitysmenetelmissä. Etenkin nykyään suositut ketterät menetelmät painottavat ihmislähtöisyyttä ja asiakkaan osallistamista (Beck ym. 2001). Eri kehitysmenetelmiä ei niinkään voi luokitella paremmuusjärjestykseen, vaan ne soveltuvat erityyppisiin projekteihin. Asiakkaan osallistaminen on iso tekijä ohjelmistokehitysprosessissa, ja osallistamisen tason vertailu antaa tietoa eri menetelmien ominaisuuksista.

Tutkielma toteutetaan kirjallisuuskatsauksena, ja sen tarkoituksena on näyttää asiakkaan rooli eri ohjelmistokehitysmenetelmissä, ja tutkimuskysymyksenä tutkielmassa on *”Miten asiakkaan osallistaminen ilmenee eri ohjelmistokehitysmenetelmissä?”*. Kirjallisten lähteiden etsintään käytetään IEEE Xplore- ja Jyväskylän yliopiston kirjaston tietokantaa sekä Google Scholar-hakukonetta. Lähdekirjallisuutta rajataan viittausmäärän perusteella ja lisäksi lähteitä tarkastellaan myös Julkaisufoorumin julkaisutason mukaan. Lähdemateriaalia etsittäessä käytetään hakusanoja ”customer involvement”, ”software development”, ”waterfall methodology”, ”prototyping”, ”RUP” ja ”Scrum”. Lisäksi hakusanoja yhdistellään oleellisten lähteiden löytämiseksi, esimerkiksi ”waterfall AND customer involvement”.

Tutkielma koostuu viidestä luvusta. Johdannon jälkeen toisessa luvussa määritellään ohjelmistokehitystä ja asiakkaan roolia ohjelmistokehityksessä. Lisäksi tarkastellaan eri tyyppisiä asiakkaita ja asiakkaan määritelmää tarkemmin. Kolmannessa luvussa käydään läpi eri ohjelmistokehitysmenetelmiä, joita tässä tutkielmassa ovat vesiputousmalli, prototyyppimenetelmä, RUP ja Scrum.

Ohjelmistokehitysmenetelmiä on lukuisia erilaisia, ja tähän tutkielmaan rajattiin neljä menetelmää. Edellä mainitut ohjelmistokehitysmenetelmät valittiin tutkielmaan koska ne ovat laajasti käytettyjä mutta toisistaan poikkeavia, ja ne soveltuvat erityyppisten ohjelmistojen kehitykseen. Näistä menetelmistä tarkastellaan kehitysprosessin kulkua sekä hyötyjä ja haittoja. Neljännessä luvussa perehdytään asiakkaan osallistamiseen edellä mainituissa ohjelmistokehitysmenetelmissä. Ensin käydään läpi asiakkaan osallistamisen taso kussakin menetelmässä, jonka jälkeen näitä tuloksia vertaillaan keskenään. Lopuksi on yhteenveto tutkielmasta.

2 ASIAKKAAN ROOLI OHJELMISTOKEHITYKSESSÄ

Luvussa käydään läpi ohjelmistokehitysprosessia ja asiakkaan roolia kehitysprosessissa. Ensin tarkastellaan ohjelmistokehitystä, ja sen jälkeen käydään läpi asiakkaan ja kehittäjän määritelmä. Lopuksi tarkastellaan kehitysprosessin eri osapuolten välistä yhteistyötä.

2.1 Ohjelmistokehitys

Ohjelmistokehityksestä ja ohjelmistotuotannosta puhutaan usein samoissa yhteyksissä. Sommervillen (2007) mukaan ohjelmistokehitys (software development) on yksi ohjelmistotuotannon (software engineering) osa-alue. Toisaalta taas ohjelmistokehitysmenetelmistä puhuttaessa käytetään termiä (SDLC) (Software Development Life Cycle), joka taas Aroran ja Aroran (2003) mukaan tarkoittaa prosessia, joka määrittelee kehitysprosessin kannalta oleellisia vaiheita niin että halutut tavoitteet saavutetaan. Ohjelmistokehityksen tavoitteena on luoda ja ylläpitää ohjelmistoja haluttua käyttötarkoitusta varten. Sommerville (2007) tunnistaa kehitysprosessista eri osa-alueita, kuten määrittely, suunnittelu, ohjelmointi, testaus ja dokumentaatio. Ohjelmistoja on monenlaisia, esimerkiksi tutkijan kehittämä taulukkolaskentasovellus datan hallintaan tai suuren logistiikkayrityksen toiminnanohjausjärjestelmä. Sommerville (2007) määrittelee ammattimaisen ohjelmistokehityksen (professional software development) prosessiksi, jossa ohjelmisto kehitetään usein tiimeissä, jotain tiettyä liiketoimintatarkoitusta varten, ulkoiselle käyttäjälle ja sitä ylläpidetään koko elinkaaren ajan. Tällaiset toisia ihmisiä varten luodot sovellukset tulee olla dokumentoitu sekä ohjeistettu loppukäyttäjiä varten. (Sommerville, 2007). Tässä tutkielmassa keskitytään juuri ammattimaisiin ohjelmistoihin ja niiden kehitysprosessiin. Sommervillen (2007) mukaan ohjelmistot voidaan jakaa kahteen eri tyyppiin: geneeriset ja kustomoidut tuotteet. Geneeriset tuotteet ovat erillisiä ohjelmistoja, joita kehittäjäorganisaatio jakelee avoimesti asiakkaille. Kustomoidut tuotteet ovat

räätälöityjä kokonaisuuksia, jotka asiakas tilaa toimittajalta omiin tarpeisiinsa. (Sommerville, 2007).

Ohjelmistokehitystä varten on olemassa useita tekniikoita ja menetelmiä, jotka soveltuvat erityyppisten ohjelmistojen kehitykseen (Sommerville, 2007). Näin ollen menetelmiä on vaikea luokitella paremmuusjärjestykseen, vaan niiden soveltuvuus riippuu kehitettävän ohjelmiston tyypistä. Lisäksi käytettävän menetelmän valintaan voi vaikuttaa myös esimerkiksi asiakaspuolen osaaminen, sillä jotkut menetelmät ovat monimutkaisia opetella ja vaativat kaikilta osapuolilta paljon tietämystä alasta.

Ammattimaisia ohjelmistoja kehitetään usein projekteissa. Hughesin ja Cotterellin (2005) mukaan projekti koostuu satunnaisista tehtävistä, joita suoritetaan vaiheittain rajoitetuilla resursseilla. Projekti suoritetaan usein jollekin ulkopuoliselle taholle, ja projektin tavoitteet ovat tarkasti määritellyt. (Hughes & Cotterell, 2005). Vaikka ohjelmistokehitysprojektia voidaan verrata tavanomaisiin projekteihin, on siinä kuitenkin monia eroja. Projektin aikana etenemistä ja sen kustannuksia voi olla vaikea seurata, sillä varsinkin alkuvaiheessa tulokset eivät ole näkyviä. Ohjelmistojen odotetaan usein mukautuvan ulkoisiin tekijöihin, jolloin projektin aikana muutokset voivat olla suuria. (Hughes & Cotterell, 2005). Ulkoisen toimittajan ja ohjelmiston tilaajan välillä saattaakin syntyä erimielisyyksiä siitä muokataanko ohjelmistoa asiakkaan prosesseihin sopivaksi vai sovitetaanko prosessit ohjelmiston mukaisiksi.

2.2 Ohjelmistokehitysprosessin osapuolet

2.2.1 Asiakas

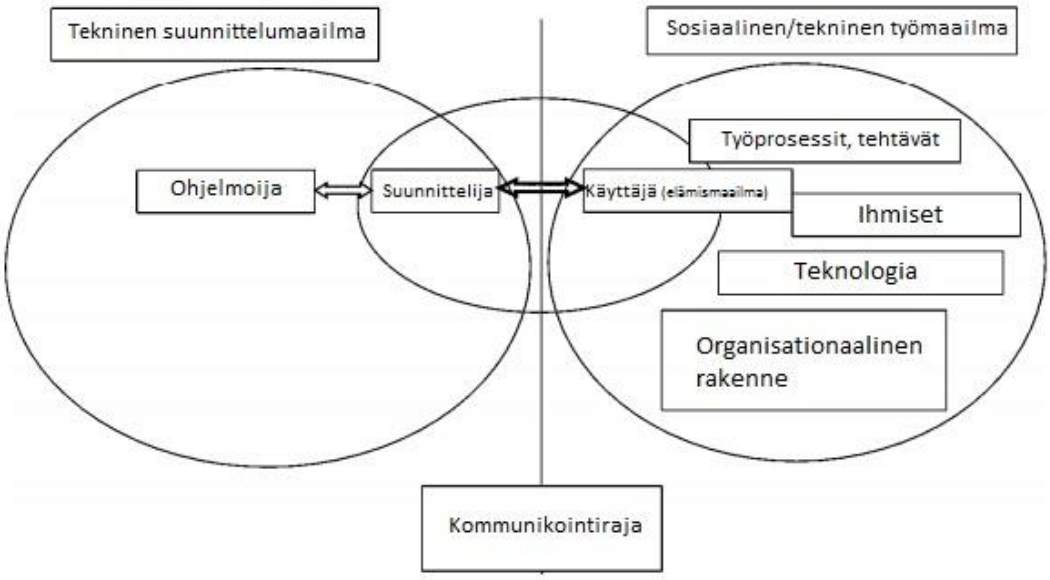
Ohjelmistokehitysprosessiin kuuluu useita sidosryhmiä. Asiakas, joka voi olla organisaation sisäinen tai ulkopuolinen taho, tilaa kehittäjältä ohjelmiston tarpeisiinsa. Saiedian ja Dale (2000) tunnistavat asiakkaan puolelta neljä avainroolia ohjelmistokehitysprojektissa. Ostaja (buyer) on vastuussa sopimusteknisistä ja rahoituspuolen asioista. Ostajat tekevät myös korkean tason päätöksiä ohjelmistoon liittyen. Loppukäyttäjä (end user) tulee käyttämään kehitettävää ohjelmistoa, joten käytettävyys ja luotettavuus ovat loppukäyttäjille tärkeimpiä ominaisuuksia. Loppukäyttäjät tietävät usein parhaiten työhön kuuluvat prosessit käytännössä. Alan asiantuntija (domain expert) ymmärtää järjestelmäympäristöä. Asiantuntijat vastaavat esimerkiksi teknisistä rajapinnoista. Ohjelmiston ylläpitäjä (software maintainer) vastaa ohjelmiston tulevista muutoksista sekä poikkeamien ratkaisemisesta ylläpitovaiheessa. (Saiedian & Dale, 2000). Edellä mainittuihin rooleihin voi kuulua useita henkilöitä asiakasorganisaatiossa. Ohjelmistokehitysprosessin kannalta on tärkeää, että roolit tunnistetaan, ja niihin kuuluvat henkilöt ovat mukana prosessissa.

2.2.2 Kehittäjä

Kuten asiakas, myös ohjelmiston kehittäjä voi olla organisaation sisäinen tai ulkopuolinen taho. Kehittäjä vastaa ohjelmiston toteutuksesta asiakkaan vaatimusten pohjalta. Saiedian ja Dale (2000) tunnistavat kehittäjän puolelta neljä keskeistä roolia. Ohjelmistojohtaja (program management) vastaa kehittäjäorganisaation tuotteiden myynnistä ja markkinoinnista sekä projektien valvonnasta. Vaatimusten määrittelijän (requirements engineer) vastuualueisiin kuuluu ohjelmiston vaatimusten tunnistaminen ja dokumentointi. Ohjelmistoinsinööri (software engineer) vastaa ohjelmiston suunnittelusta ja teknisestä toteutuksesta. Testaaja (tester) luo ja suorittaa tarvittavat testitapaukset ohjelmistolle. Eri testausvaiheisiin kuuluu esimerkiksi integraatiotestaus sekä yksittäisten toiminnallisuuksien testaus. (Saiedian & Dale, 2000).

2.2.3 Osapuolten välinen yhteistyö

Ohjelmistokehitysprosessissa on useita osallistujia ja tekijöitä. Kuvio 1 kuvaa järjestelmäkehitysprosessin monimutkaisuutta, ja sitä voidaankin verrata ohjelmistokehitykseen. Kommunikointiraja erottaa toimittajan ja asiakasorganisaation, ja pääasiallinen yhteys näiden välillä kulkee suunnittelijan ja käyttäjän välillä. Tekninen suunnittelumaailma sekä sosiaalinen/tekninen työmaailma kuvaavat myös sitä, miten kehitysprosessin osapuolten osaaminen painottuu eri osa-alueisiin, ja kommunikointirajan kautta nämä tulisi siis saada yhdistettyä. Garrity (2001) painottaa, että käyttäjän ja suunnittelijan täytyy kommunikoida keskenään jatkuvasti, jotta ohjelmiston tekniset elementit sekä työmaailman työprosessit ja tehtävät saadaan sovitettua yhteen. Asiakkaan ja toimittajan välisessä kommunikaatiossa voi olla osana myös muita henkilöitä, ja esimerkiksi Scrum-menetelmä käytettäessä käyttäjän tilalla voisi olla tuotteen omistaja.



KUVIO 1 Tietojärjestelmien kehitysympäristö (Garrity, 2001, s. 109)

Hyvän lopputuloksen saavuttamiseksi asiakkaan ja kehittäjän tulee tehdä tiivistä yhteistyötä. Keil ja Carmel (1995) toteavat, että kehitysprosessissa voidaan muodostaa asiakkaan ja kehittäjän välille yhteys (customer-developer link), jonka avulla he voivat vaihtaa tietoa. Asiakkaalla on usein hyvä käsitys liiketoiminta-alasta, jolle ohjelmistoa kehitetään, ja kehittäjällä taas kokemusta tarvittavista teknologioista. Keil ja Carmel (1995) tunnistavat asiakkaan ja kehittäjän yhteydestä kaksi erilaista ympäristöä (taulukko 1). Pakettikehitysympäristössä ohjelmistoa kehitetään ulkoiseen myyntiin ja mukautetussa kehitysympäristössä ohjelmistoa kehitetään sisäiseen käyttöön, joko ulkoisen toimittajan kanssa tai organisaation sisäisesti. Taulukosta 1 nähdään myös, että ohjelmistoa voidaan kehittää monenlaisille asiakkaille. Asiakkaan tyyppi ja ominaisuudet vaihtelevat riippuen kehitettävästä tuotteesta ja organisaatiosta. Asiakkaasta puhuttaessa voidaan siis tarkoittaa esimerkiksi suuren toiminnanohjausjärjestelmän tilaajaa tai mobiilipelin yksittäistä pelaajaa.

TAULUKKO 1 Mukautetun ja paketti-kehitysympäristön merkittävät erot (Keil & Carmel, 1995, s. 34)

Kehitysdimensio	**Mukautettu**	**Paketti**
Tavoite	Sisäiseen käyttöön kehitetty ohjelmisto	Ulkoiseen käyttöön kehitetty ohjelmisto
Tyypillinen piste, jossa asiakkaat tunnistetaan	Ennen kehitysprosessin aloittamista	Kehitysprosessin jälkeen, kun tuote lanseerataan myyntiin
Asiakasorganisaatioiden lukumäärä	Usein yksi	Useita
Asiakkaan ja kehittäjän välinen fyysinen etäisyys	Usein pieni	Usein suuri
Tyypillisiä projekteja	Uusi järjestelmäprojekti; ylläpidon parantaminen	Uudet tuotteet; uudet versiot
Termit ohjelmiston kuluttajalle	Käyttäjä; loppukäyttäjä	Asiakas
Yleinen menestyksen mittari	Tyytyväisyys; hyväksyntä	Myynti; markkinaosuus; hyvät tuotearvostelut

Asiakkaan roolia ohjelmistokehitysprosessissa korostetaan erityisesti ketteristä menetelmistä puhuttaessa. Ketterät prosessit painottavat ihmisnäkökulmaa ja jokaisen yksilön taitoja. Ohjelmistokehitysprojekteissa vaatimukset muuttuvat usein nopeasti, joten on tärkeää, että kaikki osapuolet ovat tietoisia ajantasaista vaatimuksista. Ketterissä menetelmissä asiakas voidaan jopa ottaa osaksi kehitystiimiä. Yhteistyö asiakas- ja toimittajaorganisaation välillä, niin että asiakas on osa kehitystiimiä, auttaa haluttujen tulosten saavuttamisessa matalammilla kustannuksilla ja tuo kehitystiimiin lisää eri alojen osaamista (Highsmith & Cockburn, 2001). Asiakas tilaa ohjelmiston usein tiettyä liiketoimintatarkoitusta varten, jolloin asiakkaan kokemus liiketoiminnasta on tärkeässä roolissa. Erfurth ja Rossak (2007) tutkivat ohjelmistokehityksen haasteita kehittäjän näkökulmasta. Yhtenä osuutena tutkimuksessa esitetään ongelmat sidosryhmien välillä. Tutkimuksesta selviää, että haasteet kehitysprosessissa ilmenevät erityisesti vaatimusten määrittelyvaiheessa. Ongelmat liittyvät kehittäjän tietämättömyyteen asiakkaan toimialasta sekä asiakkaan epätarkkoihin määrittelyihin. (Erfurth & Rossak, 2007). Voidaankin pohtia, että asiakkaan liiketoimintaosaamisen yhdistäminen toimittajaorganisaation tekniseen osaamiseen parantaa kehitysprojektin lopputulosta.

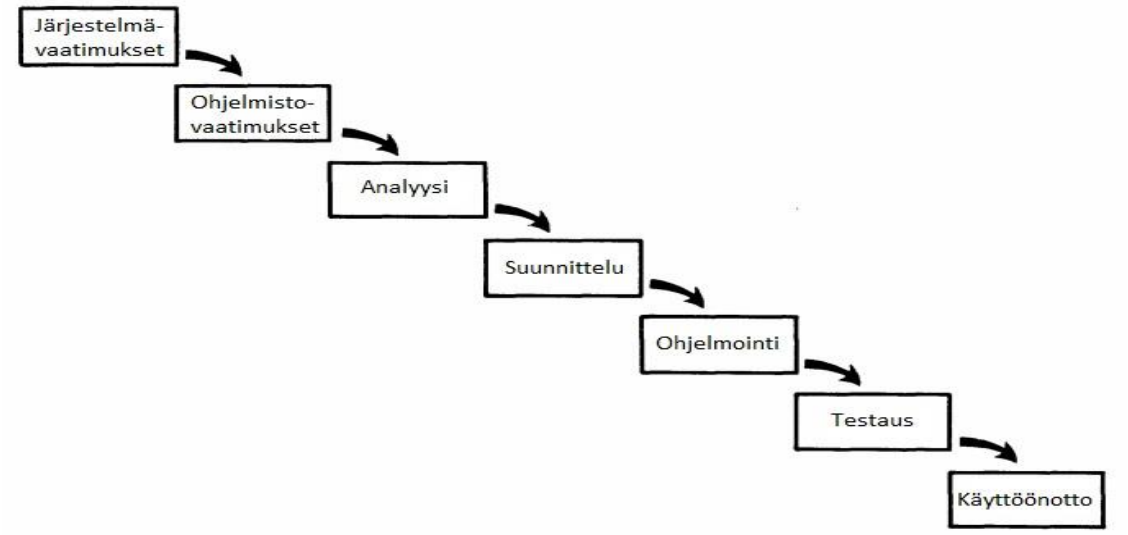
3 MENETELMÄT

Tässä luvussa kuvataan tarkemmin tarkasteltavia ohjelmistokehitysmenetelmiä, joita tutkielmassa vertaillaan. Tutkielmassa vertailtavat ohjelmistokehitysmenetelmät ovat vesiputousmalli, prototyyppimenetelmä, RUP ja Scrum. Jokaisessa kappaleessa kerrotaan ensin kunkin menetelmän ominaisuuksista ja kehitysprosessin kulusta. Lopuksi käsitellään myös menetelmien hyötyjä ja haittoja.

3.1 Vesiputousmalli

Royce (1970) määritteli alun perin suurten järjestelmien kehitysmallin, jota alettiin myöhemmin kutsua vesiputousmalliksi. Tätä mallia pidetään esimerkkinä perinteisestä ohjelmistokehitysmenetelmästä. Kuvion 2 mukainen yksisuuntainen malli on jäänyt elämään vesiputousmallina, vaikka artikkelin lopussa ehdotetaankin paluuta aiempiin vaiheisiin. Royce (1970) toteaa uskovansa malliin, mutta se myös sisältää riskejä sekä epäonnistumisen mahdollisuuksia. Royce (1970) tunnistaa kuvion 2 mallista heikkouksia, jotka liittyvät erilliseen testausvaiheeseen vasta prosessin loppuvaiheessa. Tämän takia muutokset vaatimuksissa voivat johtaa suuriin haasteisiin. Royce (1970) ehdottaakin malliin parannuksia, jossa eri kehitysvaiheiden tulisi olla vuorovaikutuksessa keskenään ylhäältä alaspäin etenemisen sijaan. Vesiputousmallissa ohjelmistokehitykseen liittyy kaksi välttämätöntä kohtaa, analyysivaihe ja sitä seuraava ohjelmointivaihe. Tarkemmin määriteltynä vesiputousmalliin kuuluu seitsemän vaihetta: järjestelmävaatimukset, ohjelmistovaatimukset, analyysi, suunnittelu, ohjel-

mointi, testaus ja käyttöönotto. (Royce, 1970).



KUVIO 2 Vesiputousmalli (Royce, 1970, s. 2)

Vesiputousmalli on yksisuuntainen, ylhäältä alas kulkevien lineaaristen tapahtumien sarja (Thummadi, Shiv & Lyytinen, 2011). Ohjelmistokehitys toteutetaan iteratiivisesti, ja mallin vaiheet toteutetaan järjestyksessä. Balajin ja Sundararajan Murugaiyanin (2012) mukaan jokainen vaihe suoritetaan itsenäisesti. Seuraavaan vaiheeseen siirrytään vasta, kun edellinen on saatu valmiiksi, eikä siihen ole enää tarkoitus palata. Jokaisen vaiheen päätteeksi suoritetaan dokumentaatio ja testaus. (Balaji & Sundararajan Murugaiyan, 2012). Toisaalta taas Roycen (1970) mukaan vesiputousmallissa tulisi palata aiempiin vaiheisiin. Voidaan kuitenkin ajatella, että vesiputousmalli on nähty sellaisena menetelmänä, jossa aiempiin vaiheisiin ei ole tarkoitus palata. Näin ollen tarkoin määritellyn etenemistavan takia onkin tärkeää, että vaatimusten määrittely on tehty selkeästi ennen jokaista vaihetta.

Petersen, Wohlin ja Baca (2009) mainitsevat vesiputousmallin hyödyiksi sen keskittymisen arkkitehtuurin suunnitteluun, mikä on tärkeää erityisesti suurissa järjestelmissä. Vesiputousmalli on myös hyvin ennustettava. (Petersen ym., 2009). Balajin ja Sundararajan Murugaiyan (2012) nostavat esille vähäisen resurssien tarpeen sekä laadunvalvonnassa auttavan kattavan dokumentaation vesiputousmallin hyötyinä. Lisäksi vesiputousmallia voidaan pitää yksinkertaisena ja helposti opittavana prosessina (Alshamrani & Bahattab, 2016). Mallin käytänteet on helppo ymmärtää, ja vaiheittaisen kehitysprosessin etenemistä on helppoa seurata.

Vesiputousmalliin on katsottu liittyvän myös useita ongelmia, jotka liittyvät juuri tarkkaan vaiheistamiseen. Kun joka vaihe suoritetaan omana kokonaisuutena, jonka jälkeen siirrytään seuraavaan, syntyy kehitysvaiheessa monia haasteita. Petersen ym. (2009) nostavat esille erilaisten dokumenttien laatimisen ja hyväksymisen eri vaiheiden välillä. Muutokseen vastaaminen on erittäin hidasta, ja alkuvaiheen ongelmat kasaantuvat myöhempiin vaiheisiin ratkaistaviksi. (Petersen ym., 2009). Juuri muutosten hankaluus vesiputousmallissa onkin johtanut ketterämpien ohjelmistokehitysmenetelmien syntyyn.

3.2 Prototyyppimenetelmä

Prototyyppi on kehitettävän ohjelmiston versio, jonka avulla voidaan kokeilla erilaisia suunnitteluvaihtoehtoja, havainnollistaa tuotetta ja löytää ongelmia ja mahdollisia ratkaisuja. Prototyyppimenetelmässä eri sidosryhmät, kuten ohjelmiston tilaaja, voivat kokeilla ohjelmistoa jo sen kehitysprosessin alkuvaiheessa (Sommerville, 2007). Prototyyppi voi myös olla osa ohjelmistoa, jonka avulla testataan siihen liittyviä oletuksia (Hughes & Cotterell, 2005). Ohjelmistokehityksessä prototyypit voidaan jakaa kolmeen eri osaan. Hughes ja Cotterell (2005) jäsentelevät erilaiset prototyypit throwaway-prototyyppiin (poisheitettävä prototyyppi), evolutiiviseen prototyyppiin ja inkrementaaliseen prototyyppiin:

- Poisheitettävää prototyyppiä käytetään ainoastaan eri ideoiden testaukseen, ja sitä ei käytetä varsinaisen lopullisen ohjelmiston kehityksessä. Tällainen prototyyppi voidaan toteuttaa täysin eri tekniikalla kuin varsinainen tuotantoversio.
- Evolutiivista eli kehittyvää prototyyppiä työstetään, kunnes se on valmis tuotantokäyttöön.
- Inkrementaalista eli lisäävää prototyyppiä käytettäessä ohjelmistoa kehitetään vaiheittain, ja palaute aiemmista kehitysvaiheista otetaan huomioon, kun myöhempiä vaiheita työstetään. (Hughes & Cotterell, 2005).

Prototyyppimenetelmän hyödyt liittyvät pääasiassa siihen, että asiakkaalla on pääsy toimivaan ohjelmistoon jo kehitysprosessin alkuvaiheesta lähtien. Amlanin (2012) mukaan prototyypin avulla voidaan havainnollistaa kehitettävää konseptia eri sidosryhmille, mikä auttaa rahoituksen saamisessa. Projektin riskienhallinnan näkökulmasta prototyyppimenetelmä vähentää epäonnistumisen mahdollisuutta, koska potentiaaliset riskit voidaan tunnistaa jo alkuvaiheessa. (Amlani, 2012). Koska menetelmään kuuluu toimivan prototyypin luominen, ei dokumentaation tarvitse välttämättä olla niin kattavaa (Hughes & Cotterell, 2005). Dokumentaation puute voidaan kuitenkin nähdä myös mahdollisena riskinä.

Toisaalta prototyyppimenetelmään liittyy tiettyjä rajoitteita. Amlanin (2012) mukaan kehitysprosessi on hidas ja projektin käynnistäminen kallista koska kehitettävää prototyyppi vaatii kehitystiimin. Itse prototyyppi voi lopulta osoittautua hyödyttömäksi, kun asiakas tarkentaa määrittelyjä. Prototyypin pieniä virheitä korjatessa ohjelmiston yleisen laadun tarkkailu saattaa jäädä pienemmälle huomiolle. (Amlani, 2012). Hughes ja Cotterell (2005) nostavat esille myös heikkouden liittyen asiakkaan ja kehittäjän väliseen läheiseen yhteistyöhön. Monet tahot tilaavat koodin ohjelmistoon halvemmista maista, esimerkiksi Intiasta. Prototyyppimenetelmää käytettäessä tämä ei kuitenkaan ole mahdollista, sillä kehittäjän ja asiakkaan tulee olla tiiviissä yhteydessä toisiinsa.

3.3 RUP

The Rational Unified Processes (RUP) on Rational Software-yrityksen kehittämä viitekehys ohjelmistokehitykseen. Osorion, Chaudronin ja Heijstekin (2001) mukaan RUP kehitettiin vastaamaan perinteisten menetelmien haasteisiin, kuten riskienhallinnan puutteeseen. RUP on iteratiivinen ja arkkitehtuurikeskeinen, ja siinä yhdistyy 1990-luvun yleisiä käytäntöjä ohjelmistokehityksessä. (Osorio ym., 2011). Kehitysprosessissa eri sidosryhmät työskentelevät tiiviisti yhdessä varmistaakseen prosessin jatkuvan parantamisen ja kehittämisen viimeisimpien kokemusten ja käytäntöjen mukaan. Anwarin (2014) mukaan menetelmän tarkoituksena on varmistaa korkealaatuinen, asiakkaan tarpeet kohtaava ohjelmisto asetetun budjetin ja aikataulun puitteissa. Viitekehys tarjoaa kehitystiimille pääsyn keskeiset kehitystoiminnot sisältävään tietokantaan, mikä lisää tiimin tuottavuutta. Tämä varmistaa myös sen, että kaikki tiimin jäsenet työskentelevät samoilla menetelmillä saman asian parissa. (Anwar, 2014). RUP toimii oppaana Rational Software-yrityksen luoman standardin Unified Modeling Language-kielen (UML) käyttämisessä vaatimusten, arkkitehtuurin ja suunnittelun apuna (Kruchten, 2004).

Hirschin (2002) mukaan RUP jakaa projektin neljään vaiheeseen. Aloitusvaiheessa (inception) projektin tavoitteet määritellään. Sen jälkeen seuraa työstämisvaihe (elaboration), jossa luodaan ohjelmistolle arkkitehtuuri, tallennetaan tärkeimmät määrittelyt sekä suunnitellaan ja arvioidaan loput projektista. Tämän jälkeen seuraa rakennusvaihe (construction), jossa toteutetaan ohjelmisto edellisen vaiheen perusteella. Viimeisenä seuraa muutosvaihe (transition), johon kuuluu testausta sekä julkaistavan version valmistelua. Edellä mainittuja vaiheita jaetaan pienempiin iteraatioihin. Prosessin aikana työstettävän iteraation tavoitteet ja kesto määritellään etukäteen, ja se pohjautuu edellisen iteraation tuloksiin. (Hirsch 2002).

RUP on valmis kokonaisuus ohjelmistokehitykseen, ja sitä käyttämällä voidaan saavuttaa monia hyötyjä kehitysprosessissa. Osorion ym. (2001) mukaan iteraatiot mahdollistavat projektipäälliköille pienempiä kokonaisuuksia hallittavaksi, joka auttaa aikatauluttamisessa. Alkuvaiheen tehtävien perusteella projektin ennustettavuus kasvaa. Iteraatiot mahdollistavat myös sen, että tuloksia voidaan näyttää aikaisessa vaiheessa. (Osorio ym., 2011). Myös Amlani (2012) nostaa esille iteraatioiden tuoman edun kehitysprosessissa. Lisäksi viitekehys on kaikille avoin, ja sen tueksi on tarjolla paljon koulutusmateriaaleja ja kursseja internetissä. RUP tukee iteratiivista prosessia sekä ohjelmiston vaatimusten muuttamista halutun lopputuloksen saavuttamiseksi. Komponenttiperustaisen arkkitehtuurin ansiosta myös ohjelmiston testaus on helpompaa. (Amlani, 2012).

RUP-viitekehyksessä on nähtävissä kuitenkin myös useita ongelmia, jotka liittyvät pääasiassa prosessin monimutkaisuuteen. Osorio ym. (2001) toteavat, että RUP menetelmän jalkauttaminen organisaation käyttöön vaatii paljon perehdyttämistä ja käytänteiden omaksumista, mikä vaatii paljon henkilöstön

kouluttamista. Lisäksi RUP vaatii keskittymistä riskeihin sekä epävarmuuksien sietokykyä. (Osorio ym., 2011). Amlanin (2012) mukaan prosessi on monimutkainen ja haastava oppia. Voidaankin todeta, että RUP-viitekehyksen tarjoamat hyödyt saattavatkin hukkua sen käyttöönoton haasteisiin.

3.4 SCRUM

Scrum on ketterä ohjelmistokehitysmenetelmä, ja niille tyypillisesti jatkuva iteraatio ja testaus ovat läsnä koko kehitysprosessin ajan. Ensimmäisenä Scrum-menetelmän määritelmänä pidetään Takeuchin ja Nonakan (1986) ajatusta, jossa nopeuden ja joustavuuden tarve vaatii erilaista lähestymistä uusien tuotteiden kehitykseen. Tähän uuteen holistiseen ajatukseen kuului itseohjautuvat projektitiimit sekä perinteisistä menetelmistä eroavat päällekkäiset kehitysvaiheet. Scrum-tiimiä voidaan verrata rugbyjoukkueeseen: joukkueen on tarkoitus päästä maaliin yhdessä, ja palloa voidaan heitellä eteen- ja taaksepäin. (Takeuchi & Nonaka, 1986). Scrum-menetelmän kolme tärkeintä kohtaa ovat tuotteen kehitysjono (product backlog), sprintit ja Scrum-tiimi (Mundra, Misra & Dhawale 2013).

Scrum-tiimi koostuu kolmesta roolista ja noin viidestä yhdeksään jäsenestä. Mundra ym. (2013) määrittelevät tiimin roolit seuraavasti:

- Scrum -mestarin tehtäviin kuuluu poistaa mahdollisia esteitä tiimin jäsenten tieltä. Rooliin kuuluu myös tiimin avustaminen Scrum-käytäntöjen omaksumisessa. Scrum-mestari siis helpottaa ja mahdollistaa tiimin työskentelyä ja vastaa siitä, että päivittäiset tehtävät hoituvat.
- Tuotteen omistaja vaikuttaa siihen, mitkä ominaisuudet kehitettävään ohjelmistoon aiotaan sisällyttää. Tuotteen omistaja tarkastelee kehitystä asiakkaan ja loppukäyttäjien näkökulmasta. Tämä rooli on enemmänkin liiketoimintakeskeinen kuin ohjelmistokehityskeskeinen.
- Scrum -tiimi vastaa itse ohjelmiston kehityksestä. Tiimiin kuuluu ohjelmistokehittäjiä, testaajia sekä muita kehitysprosessin vaatimia asiantuntijoita. Kehitettävän ohjelmiston luonteesta riippuen muita tiimin rooleja voisi olla esimerkiksi käyttöliittymäsuunnittelija. (Mundra ym., 2013).

Tuotteen kehitysjonoon on koottu tuotteen omistajan vaatimukset. Srivastavan, Bhardwajin ja Saraswatin (2017) mukaan tuotteen kehitysjono jakaantuu sprinttien kehitysjonoiksi (sprint backlog), jotka sisältävät sprintissä suoritettavat tehtävät. Sprintit ovat yhdestä kolmeen viikkoa kestäviä jaksoja, jolloin tiimi työstää heille annettua tehtävää. Sprintin aikana suunnitellaan, rakennetaan, testataan sekä arvioidaan kehitettävä tuote, ja tavoitteena on luoda asiakkaalle toimitettava versio aina sprintin päätteeksi. Sprintin tavoitteet valitaan sprintin

kehitysjonon perusteella, eikä tavoitteita vaihdeta kesken sprintin. (Srivastava ym., 2017).

Scrum-menetelmä on nopea ja joustava, ja se kykenee vastaamaan erilaisiin muutostarpeisiin kehitysprosessin aikana. Scrum tarjoaa työkalut tuotteen suunnittelemiseen sekä muuttujien hallitsemiseen kehityksen edetessä. Tämän ansiosta kehitettävää tuotetta voidaan muuttaa jatkuvasti halutun lopputuloksen saavuttamiseksi (Schwaber & Beedle, 2002). Myös Amlani (2012) nostaa esille muutosten hallinnan helppouden Scrumin avulla. Lisäksi kehitysvaiheen ongelmat voidaan havaita jo alkuvaiheessa ja käsitellä päivittäisissä tapaamisissa. (Amlani, 2012).

Myös ketterissä menetelmissä on havaittavissa kuitenkin tiettyjä heikkouksia. Balaji ja Sundararajan Murugaiyan (2012) toteavat että päätöksenteko ketterässä projektissa voi olla haastavaa kokemattomille ohjelmoijille, jolloin tilaa jää ainoastaan kokeneemmille henkilöille. Takeuchin ja Nonakan (1986) mukaan holistisiin menetelmiin kuuluvan Scrumin ongelmia ovat poikkeuksellisen suuret työmäärät työntekijää kohden sekä sen heikko soveltuvuus poikkeuksellisen suuriin projekteihin. Sprinttien suuren työmäärän takia kehitys voi viivästyä, jos työntekijät eivät ole sitoutuneita projektiin. (Takeuchi & Nonaka, 1986). Myös Amlani (2012) nostaa esille mahdollisen henkilöstön sitoutumattomuuden Scrumin ongelmana. Lisäksi Scrum-mestarin liiallinen kontrollointi voi johtaa ongelmiin kehitysprosessissa. (Amlani 2012). Ihmiset ja yksilöt nähdään ketterien menetelmien keskeisinä tekijöinä, ja voidaankin ajatella, että henkilöstö voi myös aiheuttaa suurimpia rajoitteita ja ongelmia kehitysprosessiin.

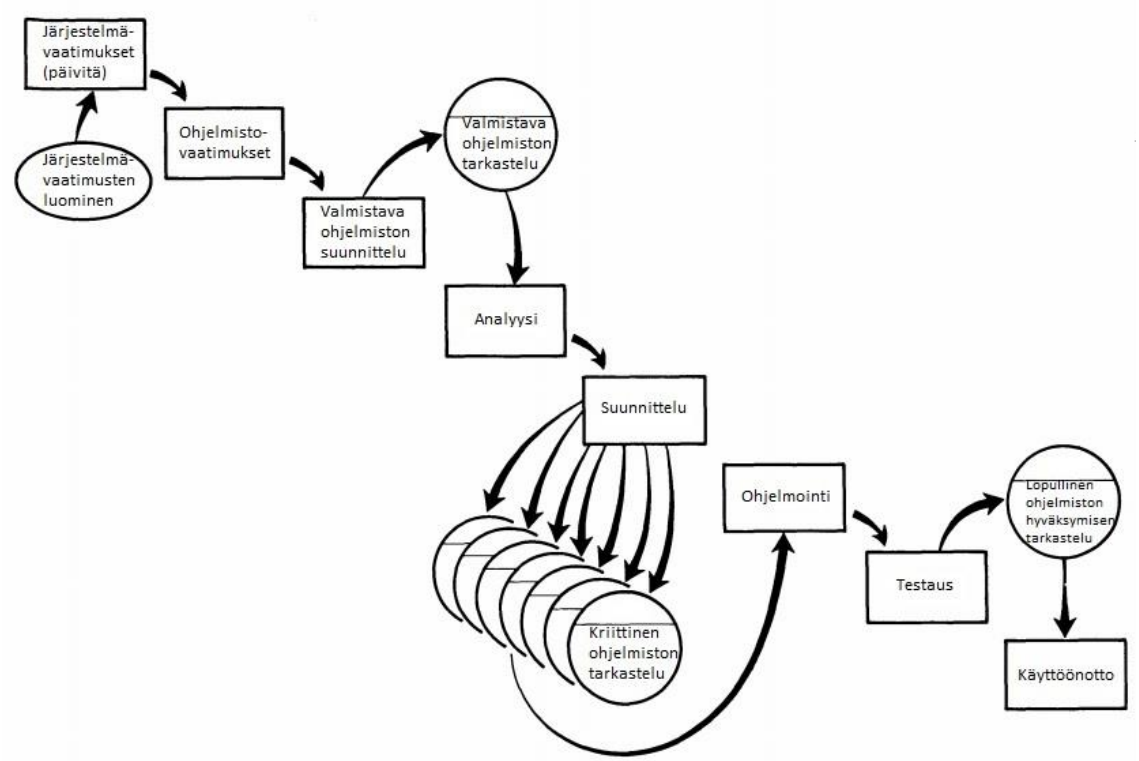
4 ASIAKKAAN ROOLI MENETELMISSÄ

Luvussa käydään läpi asiakkaan roolia eri ohjelmistokehitysmenetelmissä. Jokainen edellisessä luvussa käyty menetelmä käydään erikseen läpi tarkastellen sitä, miten asiakas osallistetaan ohjelmistokehitykseen kyseisessä menetelmässä. Lopuksi vertaillaan yhteenvetona asiakkaan roolia eri menetelmissä.

4.1 Vesiputousmalli

Vesiputousmallissa asiakas osallistuu usein kehitykseen pääasiassa vaatimusten määrittelyvaiheessa. Vesiputousmalli ei ole vaatimusten suhteen kovinkaan joustava. Kehitysprosessi etenee ylhäältä alaspäin vaiheittain, jolloin asiakkaan vaatimusten muuttaminen kesken prosessin on haastavaa. (Amlani, 2012). Voidaankin ajatella, että vesiputousmallissa asiakasta ei osallistu kehitysprosessiin kovinkaan suuresti. Asiakkaan muuttaessa vaatimuksia täytyy kehitysprosessin vaiheissa mennä taaksepäin, mikä mahdollisesti hidastaa projektia ja lisää sen kustannuksia. Ohjelmiston testaaminen kehitysvaiheessa asiakkaan toimesta on hyvä keino saada selville mahdollisia ongelmia. Vesiputousmallissa testaaminen on tarkoitus suorittaa ainoastaan omassa vaiheessaan (Balaji & Sundararajan Murugaiyan, 2012). Näin ollen asiakas ei voi osallistua ohjelmiston testaukseen kesken kehitysvaiheen, jolloin mahdolliset ongelmat voitaisiin saada selville. Vesiputousmalli ei tue asiakkaan osallistamista kesken kehitysprosessin kovinkaan paljon. Alshamranin ja Bahattabin (2016) mukaan vesiputousmallia käytettäessä asiakas ei välttämättä tiedä omia tarpeitaan. Lisäksi asiakkaalla ei ole juuri mahdollisuuksia tarkastella tuotetta ennen kuin se on valmis. Asiakasorganisaatiolla ei ole aina käytettävissä teknistä osaamista omaavaa henkilöstöä. Ohjelmiston tulevan loppukäyttäjän on hyvä osallistua kehitysprosessiin, ja Sasankarin ja Chavanin (2011) mukaan myös tällaisten henkilöiden on helppo omaksua vesiputousmallin käytänteet. Näin ollen asiakas voi osallistua kehitysprosessiin ilman korkean tason teknistä osaamista.

Toisaalta taas Royce (1970) nostaa esille asiakkaan osallistamisen vesiputousmallissa. Lopputuloksen kanssa tulee ongelmia, jos ohjelmiston toimittaja saa vapaat kädet määrittelyvaiheen jälkeen. Kuviosta 3 nähdään päivitetyn vesiputousmallin vaiheet, johon on lisätty asiakkaan osallistamisen elementtejä. Asiakkaan tulisi olla mukana vaatimusten määrittelyn jälkeen valmistavassa ohjelmiston suunnitteluvaiheessa, suunnitteluvaiheessa ohjelmiston kriittisenä tarkastelijana sekä testausvaiheessa lopullisena hyväksyjänä (Royce, 1970). Voidaankin ajatella, että vesiputousmallissa olisi tarkoitus olla myös asiakasta osallistavia piirteitä kesken kehitysprosessin, mutta prosessi mielletään usein vain kuvion 2 mukaiseksi malliksi.



KUVIO 3 Asiakkaan osallistaminen (Royce, 1970, s. 10)

4.2 Prototyyppimenetelmä

Prototyyppimenetelmä luo asiakkaalle mahdollisuuden tutustua tuotteeseen jo kehitysprosessin alkuvaiheessa. Amlanin (2012) mukaan prototyypin näyttäminen asiakkaalle auttaa vaatimusten ymmärtämisessä ja hahmottamisessa. Tämän takia menetelmä sopii hyvin henkilöille, jotka eivät osaa määritellä haluttuja vaatimuksia kunnolla. Pienen prototyypin avulla asiakas voi myös saada hyvän käsityksen toimittajapuolen taidoista. Hughesin ja Cotterellin (2005) mukaan prototyypin käyttö kasvattaa kehittäjän ja asiakkaan välisen vuorovaikutuksen määrää sekä lisää käyttäjien osallistamista kehitykseen. Dokumentaation lukemisen sijaan käyttäjät voivat tutustua prototyyppiin, joka havainnollistaa

toiminnallisuuksia paremmin. Lisäksi prototyyppiin tutustuminen voi auttaa käyttäjiä havaitsemaan puutteita tai tarvittavia lisäyksiä järjestelmässä. (Hughes & Cotterell, 2005). Prototyyppimenetelmää käyttämällä asiakas siis osallistuu kehitysprosessiin usealla eri tavalla.

Evolutiivista prototyyppiä käytettäessä asiakas näkee vaatimukset jatkuvasti, ja kehittäjä ja asiakas pysyvät tietoisena toistensa odotuksista jatkuvasti (Sasankar & Chavan 2011). Voidaankin todeta, että asiakas pääsee osallistumaan kehitysprosessiin aktiivisesti. Prototyypin avulla asiakas ja tulevat loppukäyttäjät pääsevät jatkuvasti käyttämään ohjelmistoa jo kehitysvaiheessa. Tällainen asiakkaan osallistaminen kasvattaa loppukäyttäjien taitoja käyttää ohjelmistoa, ja vähentää tarvetta koulutuksille. (Sasankar & Chavan 2011). Näin ollen loppukäyttäjien osallistaminen kehitysprosessiin saattaa myös vähentää työmäärää käyttöönottovaiheessa ja sen valmistelussa.

4.3 RUP

RUP on iteratiivinen menetelmä, ja iteratiiviset menetelmät lisäävät tyypillisesti asiakastyytyväisyyttä (Anwar, 2014). Iteraatioiden avulla vuorovaikutus kehitystiimin ja asiakkaan välillä on korkealla tasolla, mikä auttaa vastaamaan muuttuviin määrittelyihin. Osorion ym. (2011) mukaan asiakkaan aktiivinen osallistaminen ei kuitenkaan ilmene kokonaisvaltaisesti kehitysprosessin aikana, vaan kehittäjän ja asiakkaan välinen vuorovaikutus rajoittuu palautteen antamiseen ja vaatimuksista tiedottamiseen.

RUP mahdollistaa sen, että ohjelmistokehitysprosessi kehittyy jatkuvasti kokemusten ja parhaiksi todettujen käytäntöjen mukaan (Anwar, 2014). Iteratiivisuuden ansiosta asiakas voi antaa suoraa palautetta kehitystiimille koko kehitysprosessin aikana, joka auttaa kehittäjää vastaamaan muuttuneisiin asiakastarpeisiin ja luomaan tuotteen joka todella vastaa asiakkaan odotuksia (Osorio ym., 2011). Constantinen ja Lockwoodin (1999) mukaan kuitenkin RUP-menetelmässä käyttäjät eivät pääse todella vaikuttamaan itse kehitysprosessiin, vaan heitä osallistetaan ikään kuin konsultteina, jotka tarjoavat tietoa, kun sitä tarvitaan (Constantine & Lockwood, 1999). Näin ollen RUP ei välttämättä sovellu tiivistä asiakasyhteistyötä vaativiin kehitysprojekteihin.

Käyttäjätarinat ovat suuressa osassa RUP-menetelmässä. Ruparelian (2010) mukaan käyttäjätarinat kuvaavat ohjelmiston yksittäisiä käyttötapauksia, ja ne ovat pohjana suunnittelulle ja testaukselle. Asiakkaan kanssa tehdyt käyttötapaukset ovatkin hyvä keino asiakkaan osallistamiseen ohjelmistokehitykseen niin, että asiakas kuvailee haluamiaan vaatimuksia käyttötapausten muodossa. Käyttäjätarinat liittyvät kuitenkin usein yleisesti tiedossa oleviin ongelmiin. Näin ollen RUP saattaakin tarjota enemmän johtajatason kuin loppukäyttäjien tarpeiden mukaista näkemystä ohjelmistokehitykseen (Hull, Taylor, Hanna & Millar, 2002).

4.4 SCRUM

Ketterät menetelmät, kuten Scrum, tukevat vahvasti ihmislähtöistä näkökulmaa. Ketterien menetelmien taustalla on ajatus siitä, että projektin monimutkaisuuden takia asiakas ei välttämättä osaa määritellä kehitettävää tuotetta heti alkuvaiheessa, ja määrittelyt voivat muuttua kehitysprosessin aikana. Beck ym. (2001) koostivat Agile Manifesto-teoksessa ketterien menetelmien ominaispiirteitä. Ketterän ohjelmistokehityksen neljä arvoa liittyvät vahvasti asiakkaan osallistamiseen kehitysprosessissa. Prosessin sijaan kehitysprosessissa tulisi painottaa vuorovaikutusta ihmisten välillä sekä yksilöiden osaamista. Toisen arvon mukaan ketterien menetelmien tulisi tähdätä enemmän toimivan ohjelmiston kehittämiseen dokumentaation sijaan. Kolmanneksi arvoksi mainitaan yhteistoiminta asiakkaan ja kehittäjän välillä, johon tulisi panostaa enemmän kuin sopimusneuvotteluun ohjelmiston vaatimuksista. Neljäntenä arvona nostetaan esille muutokseen vastaaminen. (Beck ym., 2001). Voidaankin ajatella, että ketterissä menetelmissä asiakkaan todelliset tarpeet huomioidaan kattavasti, ja tavoitteena on tuottaa oikeasti toimiva ratkaisu asiakkaan käyttöön. Tämä onnistuu asiakkaan vahvalla osallistamisella itse kehitysprosessiin.

Ketterissä menetelmissä asiakkaan osallistumisaste kehitysprosessiin on suuri. Sillittin ja Succin (2005) mukaan asiakas on mukana kehitysvaiheessa mahdollisten ongelmien tunnistamiseksi, ja asiakas voidaan jopa ottaa mukaan itse kehitystiimiin. Määrittelyssä tulee varmistua siitä, että asiakas ja kehittäjä puhuvat samaa kieltä ja asiat tulevat oikein ymmärretyksi, ja kehittäjän tulee olla perillä kehitettävän tuotteen liiketoiminta-alasta. (Sillitti & Succi, 2005). Tämä tarkoittaa sitä, että liian teknistä ja formaalia kielenkäyttöä tulee välttää. Beckin ym. (2001) mukaan ketterissä menetelmissä painotetaan toimivaa ohjelmistoa dokumentaation sijaan. Asiakkaan ollessa osa kehitystiimiä voidaankin välttyä liialliselta dokumentoinnilta, ja asiakas voi antaa jatkuvasti palautetta kehitysprosessin tilasta (Sillitti & Succi, 2005). Scrum-menetelmässä tuotteen omistaja vastaa asiakkaan vaatimusten hallinnasta (Paasivaara, Heikkilä & Lasenius, 2012). Asiakas osallistuu kehitysprosessiin ilmoittamalla haluttuja vaatimuksia, jotka tuotteen omistaja priorisoi eri sprintteihin. Lisäksi asiakas osallistuu usein tuotteen omistajan valitsemiseen (Abrahamsson, Salo, Ronkainen & Warsta, 2002). Voidaankin ajatella, että tuotteen omistajan rooli kehittäjän ja asiakkaan välisessä vuorovaikutuksessa ja asiakkaan osallistamisessa on suuri.

4.5 Vertailu

Tutkielmassa käsiteltävät ohjelmistokehitysmenetelmät ovat erilaisia prosesseja, joissa myös asiakkaan rooli kehitysprosessissa on erilainen. Asiakkaan mahdollisuus osallistua kehitysprosessin eri vaiheisiin riippuu vahvasti käytettävästä menetelmästä. Nykyään suositummat ketterät menetelmät pohjautuvat vahvasti ihmiskeskeiseen näkökulmaan, ja sitä kautta asiakkaan osallistamiseen. Toi-

saalta taas tässä tutkielmassa tarkasteltu vanhin viitekehys, vesiputousmalli, on melko kankea asiakkaan osallistamisen osalta. Näin ollen vesiputousmallia ja Scrum-menetelmää voidaan pitää tämän tutkielman osalta eräänlaisina ääripäinä.

TAULUKKO 2 Asiakkaan osallistaminen eri ohjelmistokehitysmenetelmissä (Abrahamssonin ym., 2002, Amlanin, 2012, Constantinen ja Lockwoodin 1999, Hughesin ja Cotterellin 2005, Osorion ym., 2011, mukaan)

Asiakkaan osallistaminen	**Vesiputousmalli**	**Prototyyppimenetelmä**	**RUP**	**SCRUM**
Osallistamisen aste	Matala	Korkea	Matala	Korkea
Mahdollisuus muuttaa vaatimuksia kesken prosessin	Pieni	Suuri	Keskinkertainen	Suuri
Asiakkaan pääasiallinen rooli	Määrittely	Prototyypin tarkastelu	Palaute kehitettävästä tuotteesta, määrittely	Vaatimusten määrittely tuotteen omistajan kautta

Vesiputousmallissa asiakkaan rooli painottuu vahvasti kehitysprosessin alkuvaiheeseen. Asiakkaan tulisi tietää lopulliset ohjelmistolta vaaditut ominaisuudet heti prosessin alkuvaiheessa, sillä vaatimusten muuttaminen kesken kehitysprosessin on haastavaa. Usein asiakas ei kuitenkaan tiedä, tai osaa kertoa vaatimuksiaan. Tällaisessa tilanteessa asiakkaan tarpeita palvelee paremmin prototyyppimenetelmä, jossa asiakas voi tutustua prototyyppiin hahmottaakseen paremmin toiminnallisuuksia ja vaatimuksia. Myös Scrum-menetelmä vastaa paremmin asiakkaan muuttuviin vaatimuksiin, sillä se painottaa vuorovaikutusta kehitystiimin ja asiakkaan välillä. Toisaalta taas vesiputousmallin hyötynä voidaan pitää sen yksinkertaisuutta. Esimerkiksi RUP-viitekehys vaatii paljon panostuksia sen opetteluun ja ymmärtämiseen. Asiakkaan osallistuminen kehitysprosessiin voi vaikeutua, jos käytettävissä olevia työkaluja ei täysin tunneta. Vesiputousmalli on helppo ymmärtää ja käyttää, ja sen erillisin vaihein etenevän prosessin myötä myös helppo hallita.

Prototyyppimenetelmän osalta asiakkaan osallistaminen ilmenee eniten kehitettävän prototyypin tarkastelemisella ja testaamisella. Prototyyppimenetelmää käytettäessä asiakas osallistuu kehitysprosessiin monella eri tavoin. Prototyyppiä tarkastelemalla asiakkaalla on mahdollisuus saada hyvä käsitys kehitettävästä tuotteesta ja antaa palautetta. Näin ollen lopullinen tuote hahmottuu jo kehitysprosessin aikana, toisin kuin esimerkiksi vesiputousmallissa, jossa

lopputulos selviää vasta kehitysprosessin lopuksi. (Arora & Arora, 2003). Asiakkaan tutustuminen prototyyppiin kehitysprosessin aikana voi vähentää tarvetta loppukäyttäjien koulutukselle, toisin kuin esimerkiksi vesiputousmallissa, jossa valmis ohjelmisto voidaan kouluttaa loppukäyttäjille vasta kehitysprosessin loppuvaiheessa.

Scrum-menetelmässä asiakasta osallistetaan eniten, koko kehitysprosessin ajan. Ketterien menetelmien mukaisesti Scrum pohjautuukin ihmislähtöiseen ajattelutapaan, ja sen keskeisiä tekijöitä on vuorovaikutus asiakkaan ja kehitystiimin välillä. Scrum-menetelmää käytettäessä asiakas on tiiviisti yhteydessä kehitystiimiin, ja joissain tapauksissa jopa osana kehitystiimiä. Menetelmää käytettäessä asiakas ja kehittäjä ovat jatkuvassa vuorovaikutuksessa keskenään, ja palautteen antaminen on helppoa, samaan tapaan kuin esimerkiksi prototyyppimenetelmässä. Scrum-menetelmän osalta palaute kuitenkin pohjautuu yhteistoimintaan ja keskusteluun enemmän kuin prototyyppimenetelmän konkreettisen prototyypin tarkasteluun. Kuten Sillitti ja Succi (2005) toteavat, Scrum painottaa toimivaa ohjelmistoa sekä asioiden molemminpuolista ymmärtämistä tarkan dokumentaation sijaan. Asiakkaan osallistuminen kehitysprosessiin on helpompaa, jos huolehditaan siitä, että kaikki osapuolet todella ymmärtävät toisiaan. Vesiputousmalli taas pohjautuu vahvasti tarkkaan dokumentaatioon.

5 YHTEENVETO

Tutkielmassa käytiin läpi asiakkaan roolia eri ohjelmistokehitysmenetelmissä. Vertailtaviksi ohjelmistokehitysmenetelmiksi valittiin vesiputousmalli, prototyyppimenetelmä, RUP ja Scrum. Tutkielman tavoitteena oli vastata tutkimuskysymykseen *"Miten asiakkaan osallistaminen ilmenee eri ohjelmistokehitysmenetelmissä?"*. Tutkielman viidestä luvusta ensimmäisenä oli johdanto, jonka jälkeen toisessa luvussa käsiteltiin tutkielman kannalta keskeisiä käsitteitä, kuten ohjelmistokehitystä, asiakkaan roolia sekä asiakkaan ja kehittäjän välistä yhteistyötä. Kolmannessa luvussa käytiin läpi tähän tutkielmaan valittuja ohjelmistokehitysmenetelmiä. Näistä menetelmistä kerrottiin kehitysprosessin kulku, historiaa sekä hyviä ja huonoja puolia. Tämän jälkeen neljännessä luvussa käytiin läpi edellä mainittuja menetelmiä keskenään asiakkaan osallistamisen näkökulmasta. Luvussa tarkasteltiin asiakkaan roolia ja eri tapoja asiakkaan osallistamiseksi eri kehitysmenetelmissä. Neljännen luvun lopussa vielä vertailtiin ja esiteltiin tuloksia, eli asiakkaan roolia ja osallistamista eri ohjelmistokehitysmenetelmissä.

Kirjallisuuskatsauksen tulosten pohjalta voidaan todeta, että vertailtavista menetelmistä asiakasta osallistetaan eniten Scrum-menetelmässä, kun taas vesiputousmallin osalta asiakkaan osallistaminen oli vähäisintä. Myös RUP-menetelmässä asiakkaan osallistaminen oli alhaisella tasolla, ja vesiputousmallin tavoin se painottui kehitysprosessin alkupäähän. Prototyyppimenetelmässä asiakasta osallistettiin koko kehitysprosessin ajan, kuten myös Scrum-menetelmässä. Asiakkaan osallistamisen tasolla ei kuitenkaan suoraan voida perustella jonkin menetelmän paremmuutta, sillä eri tyyppisiin projekteihin ja ohjelmistoihin soveltuu erilaiset ohjelmistokehitysmenetelmät. Vesiputousmallin ominaisuuksien osalta tutkielmat tulosten vertailu oli mielenkiintoista. Vaikuttaisi siltä, että vesiputousmalliksi on määrittynyt malli, jossa vaiheet suoritetaan ylhäältä alaspäin, eikä aiempiin vaiheisiin enää palata. Useissa lähteissä mainittiin vesiputousmalli tällaisena prosessina, ja varsinkin sen huonoja puolia perusteltiin tällä vesiputousmaisella etenemistavalla. Royce (1970) kuitenkin toteaa artikkelissa huolensa kuvattua menetelmää kohtaan, ja esittää myöhem-

min artikkelissa parannellun mallin, jossa on tarkoitus palata taaksepäin aiempiin vaiheisiin.

Jatkotutkimusaiheena esitetään asiakkaan osallistamisen vaikutusta ohjelmistokehitysprojektin menestykseen ja lopputuloksen laatuun. Ohjelmistokehityksessä kohdataan yhä suuria haasteita, ja laatuun keskittyvien tekijöiden tutkiminen olisi varmasti hyödyllistä. Lisäksi mielenkiintoinen jatkotutkimusaihe voisi olla kehitysprosessin eri roolien lähempi tarkastelu, esimerkiksi johdon toimien vaikutukset ohjelmistoa tilatessa. Kehitysprosessia tarkasteltaessa puhutaan usein enemmän ohjelmoijista ja suunnittelijoista, joiden työ näkyy selvemmin. Niin asiakkaan kuin kehittäjänkin puolen johtajien tekemillä päätöksillä on kuitenkin suuri merkitys ohjelmistojen kehityksessä.

LÄHTEET

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2017). Agile software development methods: Review and analysis. *arXiv Preprint arXiv:1709.08439,*

Alshamrani, A., & Bahattab, A. (2015). A comparison between three SDLC models waterfall model, spiral model, and incremental/iterative model. *International Journal of Computer Science Issues (IJCSI), 12*(1), 106.

Amlani, R. D. (2012). Advantages and limitations of different SDLC models. *International Journal of Computer Applications & Information Technology, 1*(3), 6-11.

Anwar, A. (2014). A review of rup (rational unified process). *International Journal of Software Engineering, 5*(2), 8-24.

Arora, R., & Arora, N. (2016). Analysis of SDLC models. *International Journal of Current Engineering and Technology, 1*

Balaji, S., & Murugaiyan, M. S. (2012). Waterfall vs. V-model vs. agile: A comparative study on SDLC. *International Journal of Information Technology and Business Management, 2*(1), 26-30.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning J., Highsmith J., Hunt A., Jeffries R. & Jeffries, R. (2001). The agile manifesto.*The Agile Manifesto*

Boehm, B. (2002). Get ready for agile methods, with care. *Computer, 35*(1), 64-69.

Constantine, L. L., & Lockwood, L. A. (1999). *Software for use: A practical guide to the models and methods of usage-centered design* Pearson Education.

Erfurth, I., & Rossak, W. R. (2007). A look at typical difficulties in practical software development from the developer perspective--A field study and a first solution proposal with UPEX. Paper presented at the *Null,* pp. 241-248.

Garrity, E. J. (2001). Synthesizing user centered and designer centered IS development approaches using general systems theory. *Information Systems Frontiers, 3*(1), 107-121.

Highsmith, J., & Cockburn, A. (2001). Agile software development: The business of innovation. *Computer, 34*(9), 120-127.

Hirsch, M. (2002). Making RUP agile. Paper presented at the *OOPSLA 2002 Practitioners Reports,* pp. ff.

Hughes, B., & Cotterell, M. (2005). *Software project management* (4th ed ed.). London: McGraw-Hill Publishing.

Hull, M. E. C., Taylor, P. S., Hanna, J. R. P., & Millar, R. J. (2002). Software development processes—an assessment. *Information and Software Technology, 44*(1), 1-12.

Keil, M., & Carmel, E. (1995). Customer-developer links in software development. *Communications of the ACM, 38*(5), 33-44.

Kruchten, P. (2004). *The rational unified process: An introduction* Addison-Wesley Professional.

Mundra, A., Misra, S., & Dhawale, C. A. (2013). Practical scrum-scrum team: Way to produce successful and quality software. Paper presented at the *Computational Science and its Applications (ICCSA), 2013 13th International Conference On,* pp. 119-123.

Osorio, J. A., Chaudron, M. R., & Heijstek, W. (2011). Moving from waterfall to iterative development: An empirical evaluation of advantages, disadvantages and risks of RUP. Paper presented at the *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference On,* pp. 453-460.

Paasivaara, M., Heikkilä, V. T., & Lassenius, C. (2012). Experiences in scaling the product owner role in large-scale globally distributed scrum. Paper presented at the *Global Software Engineering (ICGSE), 2012 IEEE Seventh International Conference On,* pp. 174-178.

Petersen, K., Wohlin, C., & Baca, D. (2009). The waterfall model in large-scale development. Paper presented at the *International Conference on Product-Focused Software Process Improvement,* pp. 386-400.

Royce, W. W. (1970). Managing the development of large software systems. , *26.* pp. 1-9.

Ruparelia, N. B. (2010). Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes, 35*(3), 8-13.

Saiedian, H., & Dale, R. (2000). Requirements engineering: Making the connection between the software developer and customer. *Information and Software Technology, 42*(6), 419-428.

Sasankar, A. B., & Chavan, V. (2011). SWOT analysis of software development process models. *International Journal of Computer Science Issues (IJCSI), 8*(5), 390.

Schwaber, K., & Beedle, M. (2002). *Agile software development with scrum* Prentice Hall Upper Saddle River.

Sillitti, A., & Succi, G. (2005). Requirements engineering for agile methods. *Engineering and managing software requirements* (pp. 309-326) Springer.

Sommerville, I. (2007). *Software engineering* Addison-wesley.

Srivastava, A., Bhardwaj, S., & Saraswat, S. (2017). SCRUM model for agile methodology. Paper presented at the *Computing, Communication and Automation (ICCCA), 2017 International Conference On,* pp. 864-869.

Takeuchi, H., & Nonaka, I. (1986). The new new product development game. *Harvard Business Review, 64*(1), 137-146.

Thummadi, B. V., Shiv, O., & Lyytinen, K. (2011). Enacted routines in agile and waterfall processes. Paper presented at the *Agile Conference (AGILE), 2011,* pp. 67-76.