Tampereen yliopisto

Ada Happonen

YLEISIMPIÄ OHJELMISTOKEHITYSMALLEJA JA NIIDEN KÄYTÄNNÖN SOVELLUKSIA

Kandidaatintyö
Tekniikan ja luonnontieteiden tiedekunta
Ohjaaja: Veli-Pekka Pyrhönen
Toukokuu 2021

TIIVISTELMÄ

Ada Happonen: Yleisimpiä ohjelmistokehitysmalleja ja niiden käytännön sovelluksia
Kandidaatintyö
Tampereen yliopisto
Automaatiotekniikan tutkinto-ohjelma
Toukokuu 2021

Erilaisia ohjelmistokehitysmalleja on kehitetty viimeisten vuosikymmenten aikana paljon. Yritysten ohjelmistokehitysprojekteissa on tärkeää pohtia, mikä malli sopisi kehitykseen parhaiten ja miksi. Oikealla mallin valinnalla pyritään saamaan aikaan toimiva ohjelmistokehitystuote ja täyttämään asiakkaan vaatimukset mahdollisimman hyvin.

Tässä kandidaatintyössä tarkastellaan yleisimpiä ohjelmistokehitysmalleja ja niiden toimintaperiaatteita. Työ on tehty kirjallisuuskatsauksena, johon on yhdistetty lyhyt haastattelututkimus. Tutkimuksen avulla pyrittiin selvittämään, mitkä asiat vaikuttavat mallien valintaan ja minkälaisiin käyttökohteisiin eri mallit soveltuvat parhaiten. Aineistona työssä käytettiin tieteellisiä kirjoja, artikkeleja, konferenssijulkaisuja ja verkkolähteitä.

Perinteiset ohjelmistokehitysmallit, kuten vesiputousmalli, ovat olleet jo useita vuosikymmeniä ohjelmistokehityksessä käytössä. Ne pohjautuvat vaiheittaiseen kehitykseen, jossa edellisiin vaiheisiin ei palata kehityksen edetessä. Perinteinen malli ei ole kuitenkaan paras vaihtoehto, jos asiakkaan vaatimukset muuttuvat merkittävästi projektin aikana. Tällöin tarvitaan ketteriä ohjelmistokehitysmalleja, joista yleisimpiä ovat Scrum ja XP. Niissä ohjelmistoa kehitetään iteratiivisesti, mikä mahdollistaa muuttuvien vaatimusten huomioon ottamisen kehityksen kaikissa vaiheissa. Perinteisiä ja ketteriä malleja voidaan yhdistää hybridimalleiksi. Hybridimalli soveltuu tilanteeseen, jossa ketterä malli ei sovi kaikkiin projektin osiin tai organisaation toiminta halutaan muuttaa perinteisestä mallista ketterään malliin.

Avainsanat: perinteinen ohjelmistokehitys, ketterä ohjelmistokehitys,
ohjelmistokehitysmallit.

ALKUSANAT

Valitsin kandidaatintyön aiheen kiinnostukseni pohjalta. Kirjoitusprosessin alkuvaiheessa aihe liittyi yleisesti tuotekehitykseen, mutta lopulta se rajautui ohjelmistokehitysmalleihin. Haasteellisinta työssä oli tekstin kirjoitustyön aloitus, mutta kirjoitusprosessin edetessä ja aiheen ymmärryksen syventyessä työn punainen lanka ja pohjimmainen tarkoitus alkoi muotoutumaan, jonka ansiosta kirjoitusprosessin loppuvaihe sujui paremmin.

Aiheesta en entuudestaan tiennyt paljoa, mutta työtä tehdessä oli hienoa oppia uusia asioita. Kirjoitusprosessin edetessä kiinnostuin ohjelmistokehitysmalleista entistä enemmän, jonka seurauksena haluaisinkin tulevaisuudessa opiskella näitä asioita lisää ja myös mahdollisesti työskennellä ohjelmistokehitysmallien parissa.

Haluan kiittää ohjaajaani Veli-Pekka Pyrhöstä kannustavasta ohjauksesta, joka auttoi minua viemään työtä sinnikkäästi eteenpäin. Kiitän Suvi Pellistä kielenhuollon tarkastuksesta ja perhettäni saamastani tuesta koko kirjoitusprosessin ajan.

Tampereella 13.5.2021

Ada Happonen

SISÄLLYSLUETTELO

1. JOHDANTO 1
2. TUTKIMUSMENETELMÄT JA -AINEISTO 2
2.1 Tutkimusmenetelmät 2
2.2 Aineisto 2
3. PERINTEISET OHJELMISTOKEHITYSMALLIT 4
3.1 Vesiputousmallin alkuperä 5
3.2 Iteratiivinen vesiputousmalli 6
4. KETTERÄT OHJELMISTOKEHITYSMALLIT 9
4.1 Scrum 10
4.2 XP 12
5. OHJELMISTOKEHITYKSEN HYBRIDIMALLIT 15
5.1 Hybridimallien toteutus 15
5.2 Hybridimallin vertailua 17
6. KOKEMUKSIA OHJELMISTOKEHITYSMALLIEN SOVELTUVUUDESTA KEHITYSPROJEKTEISSA 18
6.1 Haastattelututkimuksen toteutus 18
6.2 Haastattelututkimuksen tulokset 19
7. YHTEENVETO 20
LÄHTEET 21
LIITTEET 23
Liite 1. 23

LYHENTEET JA KÄSITTEET

IT *(eng. information technology)*	Tietotekniikka
Iteratiivinen	Toistuva. Iteratiivisissa toimintamalleissa pyritään vuorottelemaan testaamista ja tekemistä.
JIT *(eng. just-in-time)*	Johtamisfilosofia, jonka tarkoituksena on parantaa tehokkuutta tuotanto- tai myyntiprosessin kokonaisuudessa. Suora käännös suomeksi tarkoittaa juuri oikeaan aikaan, mutta Suomessa tästä käytettään myös käännöstä: juuri oikeaan tarpeeseen (JOT).
Ketterä ohjelmistokehitys *(eng. Agile Software Development)*	Ohjelmistokehitys, jossa käytetään ketteriä menetelmiä.
Ketterät menetelmät *(eng. Agile Methods)*	Ohjelmistokehitysmalleja, joissa pyritään tekemään muutoksia ohjelmistoon vielä kehityksen loppuvaiheissa ja joissa tehdään tiivistä yhteistyötä asiakkaan kanssa.
Lean	Johtamisfilosofia, jossa pyritään vähentämään turhia komponentteja tuotannossa, ja saataisiin oikeanlaisia tuotteita oikeaan aikaan asiakkaalle. Lean- ajattelu on alun perin lähtöisin Toyota autotehtaan Toyota Production Systemin (TPS) periaatteista.
Ohjelmakehitysmalli	Kuvaa ohjelmakehityksen eri vaiheita, joita voivat olla esimerkiksi dokumentointi, suunnittelu, toteutus, testaus ja ylläpito.
Ohjelmistokehityksen elinkaari *(eng. Software Development Life Cycle)*	Kuvaa kaikkia ohjelmistokehityksen vaiheita alusta alkaen käyttöön ja ylläpitoon saakka.

Vaihepohjainen lähestymistapa *(eng. Phase-Based Approach)*	Ohjelmistokehityksen lähestymistapa, jossa kehitys kulkee vaihe kerrallaan lineaarisesti.

1. JOHDANTO

Ohjelmistokehitys on muuttunut ja kehittynyt paljon viimeisten vuosikymmenten aikana. Ohjelmistoilta vaaditaan entistä enemmän ja niiden kehityksessä on pystyttävä ottamaan huomioon asiakkaan muuttuvat vaatimukset entistä kustannustehokkaammin ja paremmin. Ohjelmistoilta edellytetään jatkuvasti parempaa tehokkuutta ja monesti myös korkeaa turvatasoa.

Ohjelmistokehityksen kulkua ja elinkaarta kuvataan erilaisilla malleilla, joista ensimmäiset kehitettiin lähes 70 vuotta sitten. Mallit ovat tärkeä osa kehitysprosessia, sillä ne luovat rakenteen ohjelmistokehitystoiminnalle. [1, s. 41] Ensimmäiset mallit olivat muodoltaan peräkkäisiä, eli niissä edettiin aina vaihe kerrallaan, yleensä askelmaisesti. Näitä malleja kutsutaan perinteisiksi ohjelmistokehitysmalleiksi. Peräkkäisiä malleja käytettäessä huomattiin kuitenkin, että ne eivät olleet tehokkaimpia silloin, kun ohjelmistotuotteen vaatimukset muuttuivat paljon. Näiden mallien ongelmiin tarvittiin uusia ratkaisuja, ja näin syntyi ketterämpiä ohjelmistokehitysmalleja, joiden iteratiivinen luonne sopii paremmin vaatimusten nopeaan muutokseen. [2, s. 1]

Tämä kandidaatintyö käsittelee yleisimpiä ohjelmistokehitysmalleja sekä näiden mallien yhdistelmiä, eli hybridimalleja. Työssä kerrotaan mallien historiasta, toiminnasta ja soveltuvuudesta eri ohjelmistokehityskohteissa. Työn keskeiset tutkimuskysymykset ovat: miten yleisimmät ohjelmistokehitysmallit toimivat, minkälaisessa ohjelmistokehityksessä niitä käytetään ja mitkä asiat vaikuttavat ohjelmistokehitysmallin valintaan.

Seuraavassa luvussa esitellään tämän kandidaatintyön tutkimusmenetelmä, ja kerrotaan aineistosta sekä sen keruutavoista. Toisen luvun jälkeen luvussa 3 kerrotaan perinteisestä ohjelmistokehityksestä ja esitellään erään perinteisen mallin toimintaa ja historiaa. Luvussa 4 tarkastellaan ketterää ohjelmistokehitystä ja kahta siihen kuuluvaa ohjelmistokehitysmallia. Viides luku käsittelee hybridimalleja ja niiden toimintaperiaatteita. Työn kuudennessa luvussa esitellään haastattelututkimus, joka ohjelmistokehitysmalleista tehtiin. Tämän jälkeen luvussa 7 on yhteenveto, jossa pohditaan, kuinka hyvin työn alussa määriteltyihin tavoitteisiin päästiin, mitä vastauksia tutkimuskysymyksiin saatiin ja mitä työssä olisi voitu tehdä vielä paremmin.

2. TUTKIMUSMENETELMÄT JA -AINEISTO

Kandidaatintyön alkuvaiheessa on valittava tutkimusmenetelmä sekä etsittävä aineistoa. Tutkimusmenetelmänä voi olla kirjallisuuskatsaus tai esimerkiksi työn laatijan itse tekemä oma tutkimus, josta vielä ei ole muualla tietoa. Alaluvuissa esitellään tämän kandidaatintyön tutkimusmenetelmä, aineistonkeruutavat ja hakumenetelmät.

2.1 Tutkimusmenetelmät

Tutkimusmenetelmänä tässä työssä on käytetty kirjallisuuskatsausta sekä kirjoittajan tekemää haastattelututkimusta. Aihe määräytyi kirjoittajan mielenkiinnon mukaan. Tietoa on etsitty sähköisistä kirjoista, artikkeleista, konferenssijulkaisuista sekä nettilähteistä, ja eri lähteiden tuloksia on vertailtu toistensa kanssa. Tietolähteiden luotettavuutta on arvioitu tutkimalla niiden mahdollisia vertaisarviointeja, vertaamalla esitettyä tietoa toisten lähteiden tietoon, arvioimalla tekstin kirjoitustyyliä ja selvittämällä, kuinka ammattitaitoisia ja kokeneita lähteiden kirjoittajat ovat esimerkiksi heidän koulutuksensa perusteella.

Aiheesta löytyy paljon uutta tietoa ja tutkimuksia, minkä ansiosta työssä pystyttiin vertailemaan eri näkökulmia ja aiheeseen liittyviä tuloksia laajasti. Tiedon paljous aiheutti haasteita hakusanojen määrittämisessä, mutta erilaisten hakukokeilujen jälkeen tarkemmin aiheeseen liittyvää aineistoa löytyi. Valitut aineistot käytiin läpi ja tutkittiin, löytyykö niistä lähteitä. Lähteiden luotettavuutta arvioitiin yllä mainituilla tavoilla.

2.2 Aineisto

Aineistoa haettiin Tampereen yliopiston tietokannoista käyttäen Andor- ja Google Scholar -hakupalveluja. Luotettavia nettilähteitä löydettiin tutkimusartikkeleiden avulla. Julkaisuja etsittiin englanninkielisillä hakusanoilla ja lähes kaikki työssä käytettävä aineisto on englanniksi. Tällä tavoin löydettiin lähivuosina tehtyjä kansainvälisiä julkaisuja. Aiheeseen liittyvä tieto on nopeasti kehittyvää ja muuttuvaa, minkä takia suurin osa aineistoista tuli olla viime vuosien aikana tehtyjä ja julkaistuja, jotta työ ei sisältäisi vanhentunutta tietoa.

Työn alkuvaiheessa aineiston etsinnässä käytettiin monenlaisia hakusanoja. Perustietoa haettaessa kohdennettiin haku kirjoihin, ja uudempaa sekä yksityiskohtaisempaa tietoa etsittiin artikkeleista, konferenssijulkaisuista ja verkkolähteistä. Hakusanoja aineistoa etsittäessä olivat ”Software Development”, “Life Cycle”, ”Agile”, ”Waterfall”, “Scrum”, “Ext-

reme Programming”, ”Agile Hybrids” ja “Agile-Waterfall Hybrids” sekä näiden yhdistelmät, kuten esimerkiksi “Agile AND Software Development” tai ”Agile AND Scrum”. Hakupalvelun eri toiminnoilla saatiin tarkennettua hakua ja termien yhdistämisellä hakutulosten määrää saatiin tiivistettyä.

3. PERINTEISET OHJELMISTOKEHITYSMALLIT

Ohjelmistokehityksen ajatellaan joskus olevan vain itse kehitettävän systeemin ohjelmointia. Se sisältää kuitenkin paljon enemmän asioita tämän lisäksi. Ohjelmistokehityksen elinkaareen (eng. Software Development Life Cycle) kuuluvat kehitettävän ohjelmiston määrittäminen, kehittäminen, testaaminen ja toimittaminen sekä myös sen käyttäminen ja ylläpito. Ohjelmiston laatu riippuu siitä, kuinka hyvin ohjelmistoprosessin metriikka on määritelty ja analysoitu koko elinkaaren ajan. Ohjelmistokehitysprosessin onnistumiseen vaikuttaa myös se, missä vaiheessa mahdollisia vikoja on huomattu ja korjattu. Vikojen varhainen havaitseminen edistää onnistuneeseen lopputulokseen pääsemistä. [3, s. 1]

Ohjelmistokehityksen elinkaarta kuvataan erilaisilla malleilla, jotka voivat olla esimerkiksi lineaarisia, iteratiivisia tai näiden kahden yhdistelmiä. Erilaiset ohjelmistot voivat erota toisistaan paljon, eikä siksi niiden kehityksen ja elinkaaren kuvaamiseen voida käyttää samoja malleja. Ohjelmisto voi tarjota palveluja loppukäyttäjälle tai toiselle sovellukselle. Loppukäyttäjälle se voi myös tuottaa graafisen käyttöliittymän, joka koostuu visuaalisista käyttöliittymäkomponenteista. [4, s. 8]

Monia uusia ohjelmistokehitysmalleja on syntynyt ja olemassa olevia malleja on kehitetty viimeisten vuosikymmenten aikana [3, s. 1]. Perinteiset ohjelmistokehitysmallit olivat ensimmäisiä malleja, joilla ohjelmistokehityksen elinkaarta kuvattiin useita vuosikymmeniä sitten. Yksi tunnetuin perinteinen ohjelmistokehitysmalli on vesiputousmalli [5, s. 316]. Vesiputousmalli koostuu yleisesti viidestä eri vaiheesta, jotka suoritetaan yksi kerrallaan loppuun ennen siirtymistä seuraavaan vaiheeseen [6, s. 16]. Kaikki ohjelmistokehitysmallit pohjautuvat ohjelmistokehityksen elinkaareen, mutta ne kuvaavat sitä eri tavalla.

Perinteisessä ohjelmistokehityksessä suunnittelu ja dokumentointi on suuressa osassa koko prosessia. Sen alkuvaiheessa laaditaan tarkkoja vaatimuksia ohjelmistotuotteelle. Tämän takia perinteisessä ohjelmistokehityksessä käytetyt mallit eivät sovellu parhaiten prosesseihin, joissa vaatimukset muuttuvat paljon ja usein. Perinteisessä ohjelmistokehityksessä myös usein määritellään tarkat rooli jokaiselle jäsenelle, eikä tiimityöskentely ole siinä olennaista. [7, s. 189]

Ohjelmistokehityksen vaiheille on määritelty tietty paikka kehitysprosessista, jossa ne sijaitsevat. Niihin on määritelty tehtäviä ja asioita, jotka tulee toteuttaa ohjelmistokehitysprosessin etenemiseksi ja valmiiksi saamiseksi. Vaikka ohjelmistokehitysmallit perustuvat pohjimmiltaan ohjelmistokehityksen elinkaareen, saatetaan perinteisissä malleissa

elinkaarta tulkita melko suppeasti [8, s. 326]. Englantilaisen kirjailijan ja teknologia-ammattilaisen Ruparelian mukaan perinteiset ohjelmistokehitysmallit ovat yleensä lineaarisia eli peräkkäisiä. Niissä suoritetaan yksi vaihe kerrallaan ja kyseisen vaiheen loputtua aloitettaan seuraava vaihe. [4, s. 8]

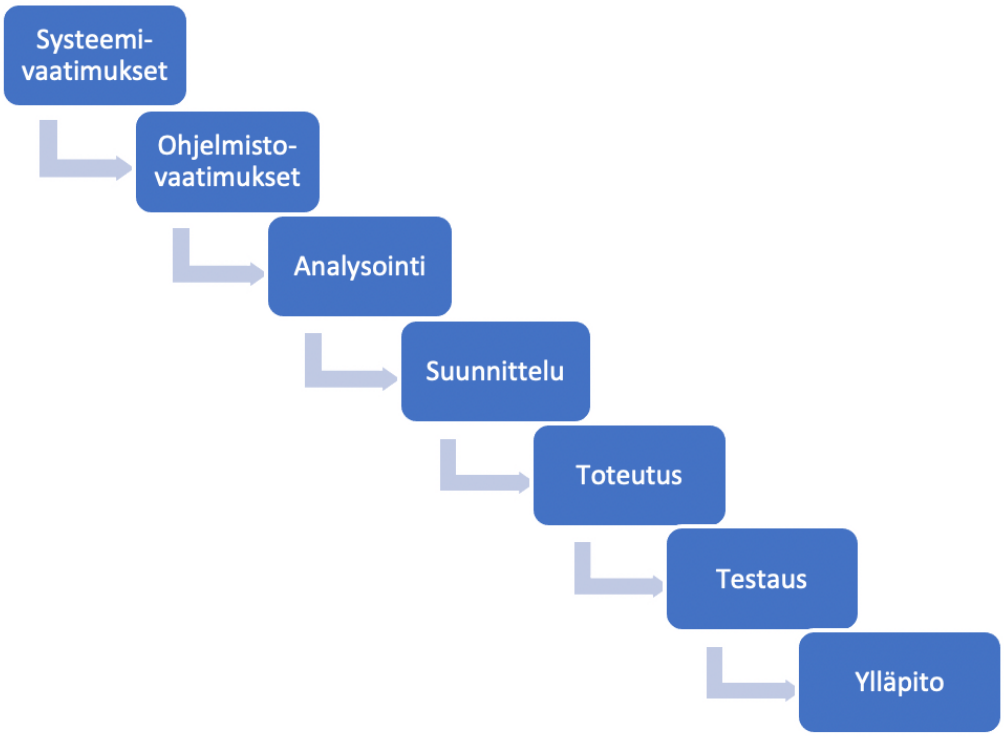
Perinteiset ohjelmistokehitysmallit soveltuvat usein parhaiten yksinkertaisiin prosesseihin [8, s. 326]. Mikäli tiedetään, että ohjelmiston kehitys tulee olemaan monimutkaista ja projekti on kooltaan todella suuri, kannattaa pohtia, olisiko jokin modernimpi malli tähän parempi, kuten jokin ketteristä ohjelmistokehitysmalleista.

Perinteisiä ohjelmistokehitysmalleja vesiputousmallin lisäksi ovat esimerkiksi V-malli ja Spiral-malli. Näissä kaikissa malleissa suoritetaan vaiheittain ohjelmistokehitystä. Spiralmalli on vesiputousmallin kaltainen, mutta se on nimensä mukaan spiraalin muotoinen ja sisältää useita silmukoita. V-mallin muoto tulee myös sen nimestä. Se kulkee samanlaisia peräkkäisiä vaiheita pitkin kuin vesiputousmalli. V-mallissa sen vasemmalla puolella on prosessin alkuvaihe, johon kuuluvat esimerkiksi vaatimusten määrittely, analysointi ja suunnittelu. Loppuvaiheessa, joka on oikealla puolella mallia, on eri osien integrointia ja testausta. [8, s. 325–327] Seuraavassa alaluvussa esitellään tarkemmin vesiputousmallin historiaa ja toimintaperiaatteita.

3.1 Vesiputousmallin alkuperä

Vesiputousmalli dokumentoitiin ensimmäisen kerran Berningtonin johdosta vuonna 1956. Berningtonin alkuperäisessä mallissa määritellyt vaiheet olivat toiminnallisten analyysien ja todistusten teko, suunnittelun ja ohjelmoinnin tietojen määritys, kehitys, testaus, käyttöönotto ja arviointi. [4, s. 8] Toinen henkilö, jota myös pidetään vesiputousmallin kehittäjänä, on ohjelmistokehittäjä Royce. Hän tutki 1970-luvulla tietokoneohjelmistojen kehitystä ja piti analysointi- ja ohjelmointivaihetta kahtena olennaisimmista osana ohjelmistokehitysprosessia. Hän piti näitä kahta vaihetta kehityksessä jopa riittävinä osina koko prosessia, mikäli kehitettävä ohjelma on hyvin yksinkertainen ja tulee käyttöön henkilölle, joka sen kehitti. Royce kuitenkin huomasi, että kehitettäessä laajempaa ohjelmistosysteemiä eivät nämä kaksi vaihetta pelkästään riitä ja ilman useampaa ohjelmistokehitysmallin osaa kehitys tulisi epäonnistumaan. [9, s. 328]

Royce jatkoi vesiputousmallin kehittämistä ja lisäsi siihen tarvittavia osia. Hänen seuraava versionsa mallista sisälsi seitsemän kohtaa, jotka esitellään kuvassa 1. Hän erotti analyysi- ja toteutusvaiheesta erikseen vaatimusten määrityksen sekä suunnittelu-, testaus- ja ylläpitovaiheen, koska ne tulisi suunnitella eri tavalla resurssien parhaan mahdollisen käytön vuoksi. [9, s. 328]

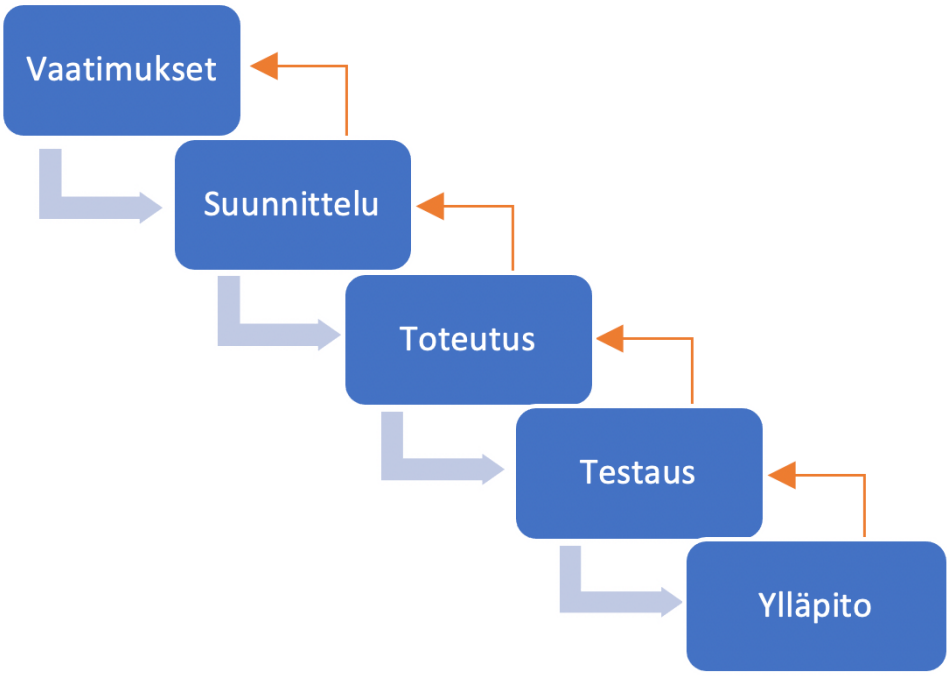


Kuva 1. *Roycen vesiputousmalli ilman takaisinkytkettyjä silmukoita, perustuu lähteeseen [9, s. 329]*

Royce edelleen huomasi, että kuvan 1 malli sisälsi paljon riskejä, jotka johtaisivat ohjelmistokehityksen epäonnistumiseen. Mikäli testausvaiheessa huomattaisiin suuria ongelmia, tulisi tehdä uudelleensuunnittelua ja muutoksia. Nämä muutokset kuitenkin saattaisivat vaikuttaa niin paljon ohjelmistokehitykseen, ettei alussa määritettyjä vaatimuksia saataisi enää täytettyä eikä lopputuloksesta tulisi enää oikeanlainen. Royce jatkoi mallin kehitystä ja lisäsi siihen iteratiivisen osion. Hän sisälsi malliin takaisinkytkeytyvät silmukat, joiden avulla mallista saatiin parempi ja aiemmin tunnistettuja riskejä poistettua. [9, s. 329]

3.2 Iteratiivinen vesiputousmalli

Royce kehitti vesiputousmallin, jossa mukana olevat, kuvassa 2 oranssilla näkyvät, takaisinkytketyt silmukat tekevät siitä iteratiivisemman ja hieman paremmin muutoksiin soveltuvan [4, s. 8–9]. Silmukoiden avulla pystytään tekemään vaiheita päällekkäin, mikä muuttaa alkuperäistä vesiputousmallia hieman joustavammaksi. Roycen seitsemänvaiheisesta mallista on muotoiltu myös viisi- ja kuusivaiheisia versioita [8, s. 326]. Tässä työssä mallista käytetään viisivaiheista muotoa, jossa vaatimukset on kuvattu vain yhdellä vaiheella, eikä analysointivaihetta ole erikseen.



Kuva 2. *Vesiputousmalli takaisinkytketyillä silmukoilla, perustuu lähteisiin [6, s. 16][9, s. 330]*

Ensimmäisenä vesiputousmallissa määritellään vaatimukset. Vaatimusten määritys voidaan jakaa vielä kolmeen eri osaan, jotka ovat vaatimusten kokoaminen, niiden analysointi ja dokumentointi. Vaatimuksista keskustellaan sidosryhmien kanssa, joita voivat olla esimerkiksi yksittäiset henkilöt tai ryhmät, jotka ovat kiinnostuneita projektista ja sen lopputulemasta niin positiivisella kuin negatiivisellakin tavalla. Vaatimusten kokoamisessa tunnistetaan sidosryhmät ja kirjataan heidän vaatimuksensa ylös. Tämän jälkeen niitä analysoidaan ja dokumentoidaan. Vaatimukset voivat yleensä olla toiminnallisia eli kuvata käyttäjän vuorovaikutusta tuotteeseen tai ei-toiminnallisia eli olla esimerkiksi ratkaisuja laatuominaisuuksiin. [6, s. 17]

Seuraava kuvassa 2 näkyvä vesiputousmallin vaihe on suunnittelu. Tässä vaiheessa suunnitellaan jokainen ohjelmiston komponentti sekä koko järjestelmän yksityiskohdat. Suunnittelun tulee olla sen mukaista, että sen jälkeen pystytään siirtymään toteutusvaiheeseen eli itse ohjelmakoodin tekemiseen. Mallin ensimmäisellä tasolla, vaatimustasolla, tehdään kuvauksia erilaisista käyttötapauksista. Nämä käyttötapaukset otetaan tarkempaa muotoiluun suunnitteluvaiheessa ja niitä tarkennetaan. [6, s. 18–22]

Suunnittelun jälkeen ohjelma toteutetaan. Ohjelmakoodin kirjoittajat käyttävät edellisissä vaiheissa tuotettuja käyttötapauksia ja muita dokumentteja mallina ja tekevät ohjelmistoa niiden perusteella. Jokainen ohjelmistokomponentti testataan erikseen, ja ohjelmoinnin loppupuolella aloitetaan integraatio, jossa komponenteista luodaan ehjä kokonaisuus. Tässä vaiheessa voi tulla ongelmia, jolloin joudutaan palaamaan suunnitteluvaiheeseen,

joka johtaa aikataulujen venymiseen. Ongelmia voidaan yrittää välttää lisäämällä toteutusvaiheeseen virstanpylväitä, jolloin tietyin ajoin ohjelman tulee olla määrätyllä tasolla ja integroitu. [6, s. 23]

Kaksi viimeistä vaihetta vesiputousmallissa on testaus ja ylläpito. Ohjelmistokehityksen ollessa täysin valmis integroidaan kaikki eri komponentit yhdeksi systeemiksi. Usein vesiputousmallia käytettäessä olisi hyvä olla erikseen integrointivaihe toteutuksen ja testauksen välissä, jolloin itse ohjelman kehittäjät testaisivat valmista systeemiä integroinnin aikana ja varmistaisivat, ettei siinä ilmene selviä vikoja, ennen kuin systeemi luovutetaan kokonaan testivaiheeseen. Testivaiheen aikana ohjelmistosysteemistä pyritään löytämään jäljelle jääneet ongelmat, joita ei ole muissa vaiheissa huomattu, ja edelleen varmistamaan, että ohjelmisto olisi mahdollisimman hyvä asiakasta varten. Virheistä syntyy vähiten kustannuksia, kun ne löydetään jo kehitysvaiheessa. Kustannukset kasvavat, kun virheitä löydetään vasta testausvaiheessa. Kaikista kalleinta kuitenkin on, jos ohjelmiston luovutuksen jälkeen asiakas huomaa, että jokin osa ohjelmistosta ei toimi. Koskaan ei voida olla täysin varmoja, onko kaikki systeemin viat löydetty. Testivaiheessa löydettyjä vikoja tulee kirjata muistiin ja tutkia niiden esiintymistä esimerkiksi kuvaajalla. Kun huomataan, että kuvaajan käyrä vikojen saapumisesta tasaantuu, voidaan ohjelmistosysteemi luovuttaa asiakkaalle. [6, s. 37–27]

Viimeinen kuvassa 2 näkyvä vesiputousmallin vaihe on ylläpito. Ylläpitovaiheeseen siirryttäessä vastuu siirtyy kehittäjiltä ylläpidosta vastaavalle tiimille, joka aloittaa tukeen liittyvät tehtävät heti, kun tuote on luovutettu asiakkaalle. Tuen tasoja voi olla erilaisia. Ylläpidossa voidaan auttaa puhelinkeskustelujen kautta, jolloin pystytään ratkaisemaan yksinkertaisia ongelmia. Tukea voi myös saada tuotteen asiantuntijoilta, jotka pystyvät auttamaan, jos heidän ei tarvitse muuttaa itse ohjelmistokoodia. Viimeisenä ja korkeimpana tukitasona on itse kehittäjän apu, jolloin virheitä voidaan korjata ohjelmakoodia muuttamalla. [6, s. 27]

Vesiputousmallissa nähdään niin hyviä kuin huonojakin puolia. Jos alussa suunnittelu saadaan tehtyä täysin lopullisen ohjelmistotuotteen vaatimusten perusteella, eikä siinä ilmene ongelmia, vesiputousmalli voi olla hyvin tehokas tapa toteuttaa kyseinen projekti. Kuitenkin, kuten jo aikaisemmin on mainittu, mikäli kehitysprosessi on todella laaja ja asiakkaan vaatimukset muuttuvat useaan kertaan, ei vesiputousmallilla pystytä kustannustehokkaasti tuottamaan tällaisiin muutoksiin vastaavaa systeemiä. [6, s. 27–28]

4. KETTERÄT OHJELMISTOKEHITYSMALLIT

Ohjelmistokehityksessä 1990-luku nähdään aikana, jolloin oli paljon ongelmia. Ohjelmistokehitysprojekteissa oli vaikeuksia täyttää määrättyjä vaatimuksia ja pysyä suunnitelluissa aikatauluissa sekä budjeteissa. Ohjelmistoja kehitettiin vesiputousmallin kaltaisella vaihepohjaisella lähestymistavalla (eng. Phase-based Approach), jossa kehityksen vaiheet tehtiin yksi kerrallaan alussa määritetyssä järjestyksessä. Tämä ei kuitenkaan enää tuottanut parhainta ja tehokkainta tulosta yrityksille teknologian kehittyessä ja asiakasvaatimusten kasvaessa, mikä johti monilla yrityksillä ohjelmistokehitysprojektien keskeyttämiseen. [10, luku 1] Vuonna 1994 The Standish Group -nimisen organisaation tekemän The CHAOS -raportin [11] mukaan kolmasosa IT (eng. Information Technology) -projekteista keskeytettiin ja hylättiin jo ennen kuin ne saatiin valmiiksi. Vaihepohjaisten mallien ongelmiin tarvittiin ratkaisu, ja näin syntyi yksinkertaisempia ja kevyempiä menetelmiä (eng. Lightweight Methods), kuten XP (eng. Extreme Programming) ja Scrum, joita kutsutaan ketteriksi menetelmiksi (eng. Agile Methods) [10].

Ketterä ajattelu ohjelmistokehityksessä sai tarkemman määritelmän vuonna 2001, kun 17 kevyiden ohjelmistokehitysmenetelmien kannattajaa kokoontui yhteen [10, luku 1] ja julkaisi ketterän ohjelmistokehityksen julistuksen (eng. Agile Manifesto) [12], jossa he määrittelivät ketterän ohjelmistokehityksen arvot ja julistuksen taustalla olevat 12 periaatetta. Ketterän ajattelun historiaan nähdään kuitenkin vaikuttaneen monet eri suuntaukset jo ennen julistuksen julkaisua, kuten JIT (eng. just-in-time) ja lean-valmistus [13, luku 2]. Näin ollen ketterän ohjelmistokehityksen julistusta ei voida täysin pitää ketterän kehityksen alkuna.

Julistuksen laatijoiden Beckin et al. [12] mukaan ketterän ohjelmistokehityksen neljä arvoa ovat

1. yksilöiden ja kanssakäymisten korostaminen
2. dokumentoinnin sijaan keskittyminen toimivan ohjelmiston aikaansaamisessa
3. asiakasyhteistyön tärkeys sopimusneuvottelujen sijaan
4. tarkassa suunnitelmassa pysymisen sijaan muutoksien tekeminen.

Arvojen laatijat painottavat erityisesti listan ensimmäisten asioiden tärkeyttä. Niiden lisäksi ketterän ohjelmistokehityksen julistuksen takana olevat periaatteet kertovat konkreettisemmin, miten kyseisen kehityksen tulisi tapahtua. Julistuksen takana olevia periaatteita ovat muun muassa asiakkaan tyytyväisyyden takaaminen toimittamalla tälle jo

aikaisessa vaiheessa vaatimukset täyttäviä versioita ohjelmistosta ja muutosten hyväksyminen sekä tekeminen ohjelmistoon myös sen kehityksen loppuvaiheessa. [12] Yleinen virhe on noudattaa näitä arvoja ja periaatteita sanatarkasti. Parhaimpaan lopputulokseen päästään, kun ajatellaan niiden laajempaa merkitystä ja tulkitaan niitä tietyn projektin yhteydessä. [13, luku 2] Näin ketterää ajattelua ja metodeja voidaan käyttää muuallakin kuin ohjelmistokehityksessä.

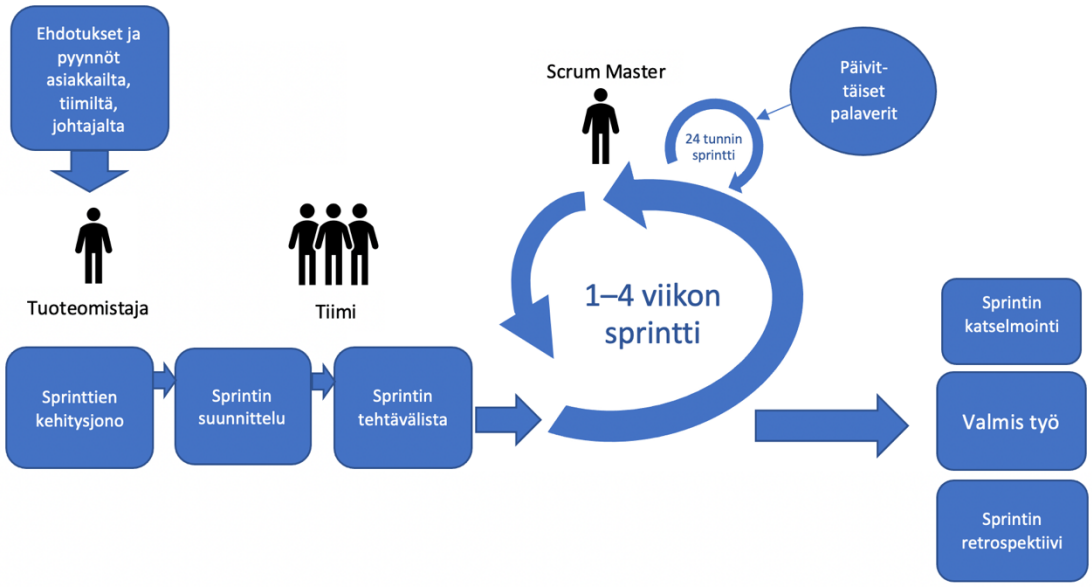
Ketterässä ohjelmistokehityksessä on useita erilaisia malleja, jotka eroavat toisistaan esimerkiksi keskittymällä kehityksessä eri asioihin. Kun ketterää mallia valitaan, tulee ottaa huomioon, onko kyseessä esimerkiksi täysin uuden vai olemassa olevan ohjelmiston kehitys, onko sidosryhmiä saatavilla mukaan kehitysvaiheeseen ja montako jäsentä kehitysprojektiin on tulossa mukaan. Malleista yleisimmin käytettyjä ovat Scrum ja XP. [14, s. 92–96] Nämä mallit eroavat toisistaan tietyillä toimintatavoillaan, mutta niissä on myös samankaltaisia piirteitä. Molemmissa malleissa on paljon lyhyitä kehitysvaiheita, joiden avulla muutoksista aiheutuvia kustannuksia pyritään vähentämään mahdollisimman paljon. Myös malleja käyttävien tiimien koko tulee pysyä melko pienenä, sillä ne eivät sovellu projekteihin, jotka sisältävät useita kymmeniä työntekijöitä. [15, s. 2] Näiden mallien toiminnasta, periaatteista ja historiasta kerrotaan seuraavissa alaluvuissa.

4.1 Scrum

Ketterän ohjelmistokehitysmallin Scrumin kehitti 1990-luvulla Sutherland ja Schwaber. He pitävät Scrumia enemmänkin viitekehyksenä, kuin tarkkana mallina, jota tulisi noudattaa jokaisen arvon ja periaatteen pohjalta sanatarkasti. Vuonna 2010 Sutherland ja Schwaber julkaisivat myös oppaan nimeltä The Scrum Guide, jonka tarkoituksena on auttaa ihmisiä ymmärtämään kyseisen mallin toimintatapaa ja periaatteita. Viimeisin päivitetty versio oppaasta on julkaistu vuonna 2020. Oppaassa kerrotaan Scrumin määritelmä, sen teoriaa ja kuinka mallia tulee käyttää, sekä myös sen viisi arvoa, jotka ovat sitoutuminen, keskittyminen, rohkeus, kunnioitus ja avoimuus. Näiden arvojen omaksuminen vaikuttaa siihen, kuinka hyvällä menestyksellä Scrumin käyttö onnistuu. [16]

Kuvassa 3 esitellään, kuinka Scrum-viitekehitys toimii ja etenee sitä käytettäessä. Scrumissa on monia eri vaiheita, mutta tärkein niistä on kuvassa näkyvä 1–4 viikkoa kestävä sprintti, joka sisältää neljä muodollista tapahtumaa. Scrum-tiimi koostuu yleensä alle kymmenestä jäsenestä, jotka jakautuvat kolmeen vastuualueeseen, joita ovat Scrum Master, tuoteomistaja ja kehittäjät. Scrum Masterin tehtävä on esimerkiksi huolehtia, että Scrumia toteutetaan oppaan mukaisesti, toimia johtavana esimerkkinä sekä valmentajana, joka ohjaa tiimiä jäseniä itseohjautuvuuteen, auttaa tuoteomistajaa määrittelemään tuotteen tavoitteet ja palvella organisaatiota suunnittelemalla sekä neuvomalla Scrumin

toteutuksessa. Tuoteomistajan vastuualueeseen kuuluu yhtenä osana huolehtia, että Scrum-tiimi tuottaa työllään arvoa mahdollisimman paljon. Scrum-tiimistä kehittäjät ovat henkilöitä, jotka tekevät konkreettisesti itse kehitystä heille annettujen ohjeiden mukaan ja vastaavat esimerkiksi sprinttien kehitysjonon suunnittelusta. Kehitysjono on työstä oleva reaaliaikainen kuva, jota pyritään kehittämään Scrumin sprinttien aikana säännöllisesti. Tämän avulla kehitystä pystytään tarkastelemaan päivittäin. Koko Scrum-tiimissä ideana on se, että jokainen osa on yhdenvertainen, eikä osia jaotella arvojärjestykseen. [16]



Kuva 3. *Scrum-viitekehyksen toimintamalli, perustuu lähteeseen [17, s. 10].*

Scrumissa olennaisimpana osana ovat sprintit. Ne sisältävät tuotteen tavoitteen saavuttamiseksi tehtävän työn. Sprintti aloitetaan kuvassa 3 näkyvällä Sprintin suunnittelu -vaiheella, joka kestää maksimissaan kahdeksan tuntia. Suunnittelussa pyritään vastaamaan kolmeen kysymykseen: miksi sprintti on arvokas, mitä sprintissä voidaan saada aikaan ja miten kyseisen sprintin ajalle suunniteltu työ saadaan tehtyä. Sprinttien aikana ideoista pyritään tuottamaan arvoa ja esimerkiksi tarvittaessa jalostetaan kehitysjonoa. Sprinttien pituus pidetään tarpeeksi lyhyenä, jotta haluttua tavoitetta kohti etenemistä pystytään tarkastelemaan usein. Tämä johtaa parempaan ennustettavuuteen, joka on sprinttien yksi parhaista puolista. [16]

Kuvassa 3 näkyvät sprintin päivittäiset palaverit ovat osa jokaista sprinttiä, ja ne kestävät noin 15 minuuttia. Palavereiden tarkoituksena on tarkastaa, että edetään kohti sprintin tavoitetta, ja suunnitella, mitä seuraavana työpäivänä tulisi tehdä. Kun sprintti saadaan päätökseen, toteutetaan sen katselmointi. Katselmoinnissa tutkitaan sprintin tuloksia, pohditaan muutostarpeita seuraavaa sprinttiä varten ja esitellään tulokset tärkeimmille

sidosryhmille. Sprintin jälkeen on myös sen retrospektiivi, eli vaihe, jossa pyritään miettimään, miten laatua saataisiin parannettua ja miten ihmisiin sekä yhteistyöhön liittyen sprintti sujui. Sprintin retrospektiivin pituuteen vaikuttaa itse sprintin pituus. Sprintin ollessa kokonaisen kuukauden pituinen on retrospektiivi tällöin noin kolmen tunnin mittainen. Tämä vaihe päättää sprintin. [16]

Yllä mainittujen kuvassa 3 näkyvien Scrumin eri vaiheiden ja sen tiimin avulla malli toimii enemmänkin työkaluna ohjelmistokehityksessä, mikä auttaa kehitystä kulkemaan haluttuun suuntaan nopeasti ja tehokkaasti. Kaikkien suunniteltujen sprinttien ja niihin kuuluvien toimintojen jälkeen tulisi olla koossa asiakkaan vaatimukset täyttävä ohjelmistotuote. [16]

4.2 XP

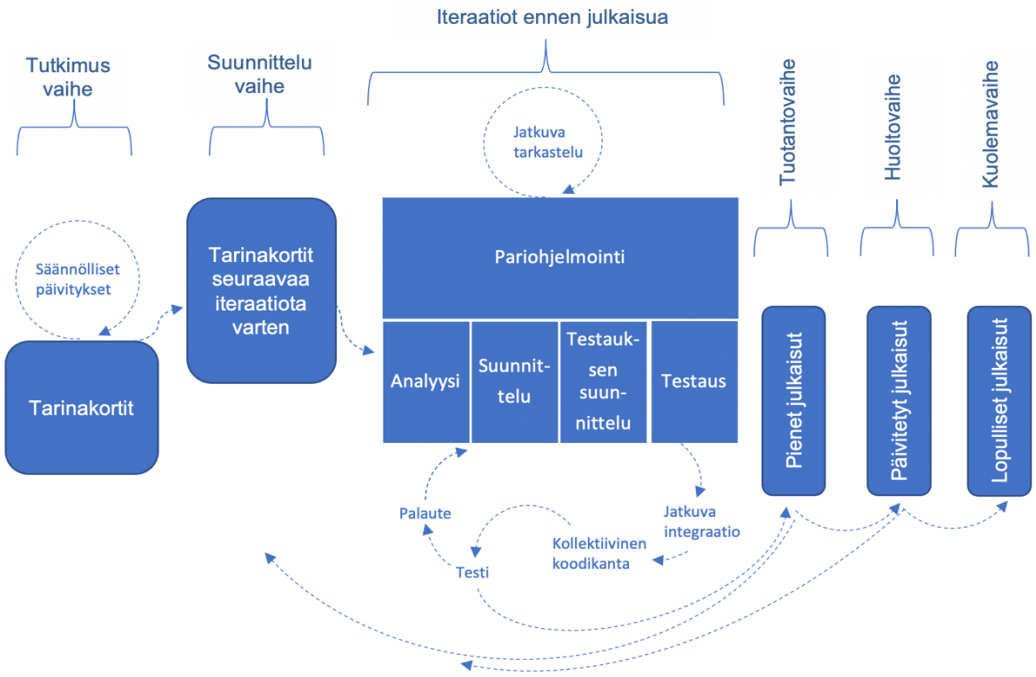
XP on toinen yleisesti käytetty ohjelmistokehitysmalli. Se on hyvin samantapainen toiminnaltaan kuin Scrum, mutta mallissa erityisesti keskitytään ohjelmakoodiin syntyvien virheiden huomioimiseen ja korjaamiseen. Asiakkaan muuttuvat vaatimukset johtavat ohjelmakoodin muuttumiseen, joka on usein virheiden aiheuttaja ohjelmassa. XP sisältää, samalla tavalla kuin Scrum, erilaisia arvoja, periaatteita ja käytäntöjä. Se toimii kaikista tehokkaimmin, kun jokainen tiimin jäsen sisäistää sen oikealla tavalla. [18, luku 6]

XP:n kehitti Kent Beck [15, s. 2][19], joka oli myös yksi ketterän ohjelmakehityksen julistuksen laatijoista. Beckin mukaan XP:n yhtenä ideana on se, että mallin tiettyjen toimintojen ajatellaan tuottavan erityistä arvoa ohjelmistokehityksessä, kun niihin keskitytään ja niitä työstetään eteenpäin enemmän kuin aikaisemmin. [19] XP:ssä määriteltyjen arvojen avulla kehitystä pyritään ohjaamaan haluttuun suuntaan. Arvoja ovat kommunikaatio, yksinkertaisuus, palaute, rohkeus ja kunnioitus. Malliin liittyy myös käytäntöjä, joista ensisijaisia ovat testilähtöinen ohjelmointi (eng. test-first programming), pariohjelmointi (eng. pair programming) ja esimerkiksi viikoittainen sykli (eng. weekly cycle). Ensisijaisten käytäntöjen avulla voi aloittaa XP:n käyttöönoton valiten, mitä käytäntöä kyseinen kehitysympäristö tarvitsee. [20, luku 1]

XP-mallin käyttö ohjelmistokehityksessä aloitetaan siitä kohtaa kehitystä, jossa kyseisellä hetkellä ollaan. Malli voidaan siis ottaa mukaan kesken ohjelmakehitystä, eikä ainoastaan ennen kehittämisen alkua. XP:n avulla kehittäjän kokemuksia sekä käynnissä olevaa ohjelmistokehitystä voidaan parantaa. Malli otetaan käyttöön lisäämällä ohjelmistokehitykseen XP:n käytäntöjä, jotka tulee valita niin, että ne parantaisivat mahdollisia

kehityksen heikkoja kohtia, jotka voivat olla itse ohjelmoinnissa tai esimerkiksi tiimityöskentelyssä ja kommunikoinnissa. Muutoksia XP:n avulla on helpoin tehdä, kun otetaan yksi käytäntö kerrallaan mukaan kehitykseen eikä kaikkia samaan aikaan. [20, luku 1]

XP:n toimintamallin eri vaiheet on esitelty kuvassa 4. Ensimmäisenä vaiheena mallissa on tutkimusvaihe. Tutkimusvaiheen aikana määritellään vaatimuksia, arkkitehtuuria ja esimerkiksi työkaluja, joita ohjelmistokehityksessä tullaan käyttämään. Vaiheen aikana asiakas laatii tarinakortteja, joihin se kirjoittaa haluttuja käyttäjäkertomuksia, joiden avulla kehittäjä saa kuvan järjestelmävaatimuksista. Mallin seuraava vaihe on suunnitteluvaihe. Sen aikana valitaan minkä tarinakortin tarina tullaan seuraavaksi toteuttamaan. [21, s. 2] Suunnitteluvaiheessa myös tehdään päätös ensimmäisen pienen julkaisun sisällöstä [22, s. 20]. Beck [20] korostaa, että suunnittelussa tulisi jokaisen tiimin jäsenen olla mukana, koska se vaatii yhteistyötä ja jokaisen tulisi tulla kuulluksi.



Kuva 4. *XP-toimintamalli, perustuu lähteeseen [22, s. 21].*

XP:n kolmannessa vaiheessa, joka näkyy kuvan 4 keskellä, suoritetaan iteraatioita, eli kehitetään tuotetta useiden kierrosten ajan jatkuvasti sitä tarkkaillen. Iteraatiot ennen julkaisua -vaiheen osia ovat pariohjelmointi, analyysien teko, suunnittelu ja testaus. [22, s. 21–22] Pariohjelmoinnissa kaksi henkilöä ohjelmoi yhdessä samalla laitteella, jonka avulla työskentelystä saadaan dynaamista [23, luku 10]. Iteraatioiden aikana ohjelmistotuotetta kehitettään valittujen tarinakorttien tarinoiden avulla [22, s. 22]. Yksi iteraatio voi

kestää 1–4 viikkoa, jonka jälkeen kierroksen onnistuessa asiakkaalle esitellään tuotokset. Mikäli iteraatiokierroksen aikana huomataan ongelmia, tulee niitä korjata ja huomioida seuraavalla kierroksella. Viimeisen iteraation jälkeen voidaan siirtyä neljänteen vaiheeseen, joka on tuotantovaihe. [23, luku 21]

Tuotantovaiheessa tyypillisesti iteraation pituus lyhenee noin viikon pituiseksi. Vaiheessa tulee todistaa erilaisten testien avulla, että ohjelmisto on valmis luovutettavaksi asiakkaalle. Sen kehitystä jatketaan, mutta hieman hitaammin kuin aikaisemmin. Tuotantovaiheen lopussa ohjelmisto on valmis asiakkaan käyttöön. Ohjelmiston luovutuksen jälkeen siirrytään toiseksi viimeiseen vaiheeseen, huoltoon. [23, luku 21]

Huoltovaiheessa asiakkaan käyttöön luovutettua ohjelmisto kehitetään, parannetaan ja ylläpidetään. Tämä on XP:n yleisin vaihe, jossa usein ollaan. Vaiheen aikana tiimin jäsenet saattavat muuttua ja mukaan voi tulla uusia ohjelmistokehittäjiä. Kun kehitystä ja ylläpitoa on tehty niin pitkään, että asiakas ei saa aikaan enempää tarinakortteja, joihin kirjoittaisi uusia vaatimuksia ja tarpeita ohjelmiston kehitykselle, ja asiakas pitää ohjelmistoa kaikilta osin täydellisenä, siirrytään kuolemavaiheeseen, joka näkyy myös kuvassa 4. Tämä on XP:n viimeinen vaihe, jonka aikana tehdään lopulliset dokumentoinnit, eikä enää jatketa ohjelmiston kehittämistä. [23, luku 21]

5. OHJELMISTOKEHITYKSEN HYBRIDIMALLIT

Usein ohjelmistokehityksessä valitaan yksi tietty malli, jonka pohjalta kehitystä aloitetaan tekemään. Nykyään ohjelmistokehityksessä käytetään suurimmaksi osaksi ketteriä malleja, kuten Scrumia, koska ne toimivat parhaiten muuttuvassa ympäristössä. Yhdysvaltalaisen projektinhallinnan sekä ketterien menetelmien ammattilaisen Bellingin mukaan yritys voi kuitenkin joskus huomata olevansa tilanteessa, jossa projektissa käytettävä malli on enemmänkin perinteisen ja ketterän välimuoto. Tällöin on tärkeää osata arvioida, minkä verran mallissa on ketteriä ominaisuuksia sekä minkä verran perinetisen mallin piirteitä, ja ymmärtää, miksi näin on. [24, luku 1]

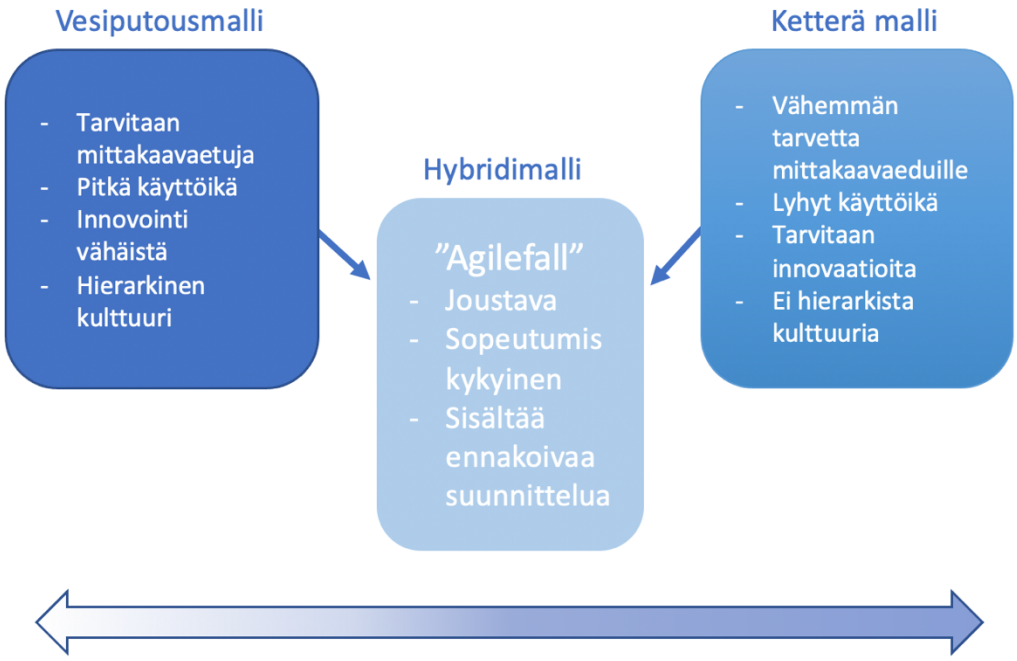
Näitä eri mallien yhdistelmiä kutsutaan hybridimalleiksi. Yleisimpiä hybridimalleja ovat esimerkiksi vesiputousmallin sekä ketterien mallien yhdistelmät, kuten vesiputous-Scrum hybridimalli. Seuraavissa alaluvuissa esitellään, miten hybridimalli muodostuu, mitä osia siihen kuuluu ja minkälaisissa ohjelmistokehitysprojekteissa hybridimalleja käytetään.

5.1 Hybridimallien toteutus

Hybridimalleja, kuten kaikkia muitakin kehitysmalleja, voidaan käyttää esimerkiksi projektinhallinnan työkaluna. Hybridimallin muoto vaihtelee sen mukaan, minkälaisesta projektista on kyse. Ohjelmistokehitysprojekteissa hybridimallissa on yleensä enemmän ketterän mallin ominaisuuksia, kun taas esimerkiksi konkreettisen laitteen tai koneen kehityksessä saattaa olla perinteisen mallin, kuten vesiputouksen, piirteitä enemmän. Monet asiat, kuten kehitettävän tuotteen käyttöikä, projektin suuruus ja riskitaso, vaikuttavat kuitenkin siihen, minkä mallin ominaisuuksia hybridimallissa on laajimmin. [24, luku 1]

Vaikka ohjelmistokehitys on usein enemmän ketterien mallien kaltaista, voi siinäkin olla paljon perinteisten mallien ominaisuuksia, jos esimerkiksi ohjelmistoa kehitetään systeemiin tai ympäristöön, jossa kaikki mahdolliset riskit on minimoitava. Tällöin tarvitaan esimerkiksi tarkempaa suunnittelua jo ennen kehityksen alkua, jotta jo aikaisessa vaiheessa ohjelmistosta saadaan suunniteltua mahdollisimman turvallinen. Kuvassa 5 näkyy, minkälaisista ominaisuuksista hybridimalli voi koostua, sekä nuolella kuvattu jana, jonka avulla organisaatio voi tutkia, minkä verran heidän hybridimallissaan on ketteriä ja perinteisiä piirteitä. Mikäli organisaatio tunnistaa sijaitsevansa esimerkiksi enemmän janan oikealla puolella, tarkoittaa se sitä, että organisaation käyttämässä mallissa on

enemmän ketterän mallin ominaisuuksia. Organisaatiossa on hyvä osata tunnistaa, missä se janalla sijaitsee. [24, luku 1]



Kuva 5. *Vesiputousmallin, ketterän mallin sekä hybridin erot ja yhtäläisyydet, perustuu lähteeseen [24, luku 1]*

Kuvassa 5 esitetään, kuinka vesiputousmallista ja ketterästä mallista voidaan saada aikaan hybridimalli. Ketterästä mallista otetaan usein sen joustavia ominaisuuksia sekä sopeutumiskykyä, johon yhdistetään vesiputousmallin ennakoivaa suunnittelua. Kun tämänlaista hybridimallia luodaan, organisaation on ensimmäisenä tärkeä olla tietoinen sen omasta kulttuurista, rakenteesta ja projekteista. Tietoisuuden avulla pystytään suunnittelemaan hybridimalli, joka vastaa organisaation tarpeisiin. Toisena organisaation on tutkittava, minkälaisia varoja sillä jo on eli onko käytössä jo mahdollisesti toimivia malleja tai toimintatapoja, joita hybridimallissa voitaisiin käyttää. Hybridimalliin siirryttäessä kaikkea ei tarvitse vaihtaa, vaan toimivat ja hyväksi todetut piirteet organisaation toiminnassa kannattaa säästää. [24, luku 3]

Seuraavaksi organisaation tulee pohtia, mitä ja ketä he tarvitsevat. Tarkastelun avulla, jossa pohditaan, mitä organisaatio tarvitsee, saadaan selville, mihin osiin projektia tarvitaan parannusta ja mitkä osat projektin toiminnasta jo toimivat. Organisaation toiminnan muutoksiin tarvitaan vahva henkilö, esimerkiksi tuotepäällikkö, joka tukee muutosta ja ajaa sitä eteenpäin. Muutokset on myös saatava myytyä organisaation muille vaikutusvaltaisille jäsenille sekä henkilöille, joihin hybridimallin käyttöönotto tulee vaikuttamaan. Aina on mahdollisuus, että muutos organisaatiossa voi aiheuttaa vastarintaa, minkä takia

on tärkeä lisätä koko organisaatiossa tietoisuutta muutoksen tarpeesta, jolloin uuteen menetelmään pystytään mukautumaan paremmin. Näiden tapojen avulla yritys pystyy ottamaan hybridimallin käyttöön toiminnassaan ja saamaan sillä aikaan positiivisia muutoksia. [24, luku 3]

5.2 Hybridimallin vertailua

Hybridimallien käyttöä on tutkittu useissa tutkimuksissa viimeisten vuosien aikana. Vuonna 2016 Kuhrmann et al. [25, s. 38] tekivät HELENA-nimisen tutkimuksen ensimmäisen osan, jossa selvitettiin, mitä ohjelmistokehitysmalleja organisaatiot käyttävät ja kuinka yleistä hybridimallien käyttö on. Tutkimuksen mukaan esimerkiksi yrityksen koko ei vaikuttanut hybridimallin käyttöön, eli niin pienet kuin suuretkin yrityksen käyttävät niitä. Myöskään viitteitä siitä, että erilaiset standardit ohjaisivat hybridimalleja, ei löydetty. HELENA-tutkimuksen mukaan usein yritykset haluavat hybridimallin avulla tuoda perinteisten mallien ominaisuuksia esimerkiksi riskienhallintaan ja ketterän mallin piirteitä vaatimusten suunnitteluun, integrointiin ja testaukseen. [25, s. 38]

Vaikka Kuhrmannin et al. [25] tekemän tutkimuksen mukaan hybridimallit ovat yleisiä ja niillä yritys saa monenlaisia etuja käyttöönsä, huomioi Belling [24, luku 1] kuitenkin sen, että vaikka hybridimallia käyttäen voidaan saada aikaan entistä paremmin vaatimuksiin vastaava lopputulos ohjelmistokehitysprojekteissa, voi sen tarkka ja harkittu käyttö myös johtaa epäonnistumiseen. Jos esimerkiksi organisaatio ei suostu ottamaan käyttöön toista hybridimallin puolta omien tottumustensa takia, hankaloittaa tämä mallin toteutusta. Belling kuitenkin myös toteaa, että esimerkiksi vesiputousmallin ja ketterän mallin yhdistelmän avulla voidaan parhaimmassa tapauksessa tuottaa todella hyvä lopputulos, jos asiakas on valmis kasvattamaan budjettiaan ja projektin kestoa. Tällöin ketterän mallin sprinttien avulla asiakas pystyy entistä paremmin näkemään, mitä todella ohjelmistotuotteelta haluaa, ja muuttamaan sekä lisäämään vaatimuksia useita kertoja. [24, luku 1]

6. KOKEMUKSIA OHJELMISTOKEHITYSMALLIEN SOVELTUVUUDESTA KEHITYSPROJEKTEISSA

Tässä luvussa esitellään tähän kandidaatintyöhön tehty haastattelututkimus. Tutkimuksen tarkoituksena oli selvittää, mitä ohjelmistokehitysmalleja nykyään ohjelmistokehityksessä suositaan ja miksi, millä perusteilla malli tiettyyn projektiin valitaan, kuinka yleistä hybridimallien käyttö ohjelmistokehityksessä on ja miten hyötyjä tai haittoja ohjelmistokehittäjät näkevät hybridimalleissa.

6.1 Haastattelututkimuksen toteutus

Tutkimus suoritettiin haastattelemalla kahta ohjelmistokehittäjää. Haastateltavat valittiin eri ikäryhmistä, jolla tutkimukseen pyrittiin saamaan mukaan eri näkökulmia ja kokemuksia. Tutkimuksen haastattelua varten laadittiin yhteensä 12 kysymystä, jotka ovat työn liitteenä.

Haastattelukysymysten tarkoituksena oli selvittää haastateltavien taustaa ohjelmistokehityksessä, millaisissa ohjelmistokehitysprojekteissa he ovat olleet mukana sekä mitä malleja niissä on käytetty, ovatko haastateltavat käyttäneet hybridimalleja ja mitä hyötyjä tai haittoja he mallissa ovat kokeneet. Haastateltavilta kysyttiin myös yhteen heidän valitsemaansa projektiin liittyen tarkempia kysymyksiä, joiden avulla yllä mainittuihin kysymyksiin pyrittiin saamaan vastauksia. Tutkimus suoritettiin kirjallisesti, eli haastateltavat vastasivat sähköpostin kautta heille lähetettyihin kysymyksiin.

Ensimmäinen haastateltava on nuorempi henkilö, joka on vasta muutaman vuoden työskennellyt ohjelmistokehittäjänä. Hän on käyttänyt ainoastaan ketteriä ohjelmistokehitysmalleja, eikä ole ollut osallisena projekteissa, joissa käytettäisiin perinteisiä malleja tai hybridimallia. Haastateltava on ollut mukana ohjelmistokehitysprojektissa, joissa kehitetään finanssialan ohjelmistoa.

Toinen haastateltava on työskennellyt ohjelmistokehitykseen liittyvissä tehtävissä 12 vuoden ajan. Ensimmäiset vuodet hän toimi itse ohjelmiston kehittäjänä ja tämän jälkeen erilaisissa ohjelmistokehitystä ohjaavissa rooleissa. Tällä hetkellä hän ohjaa erään automaatiojärjestelmän kehitystä, joka koostuu yli 30 eri palvelusta. Haastateltava on ollut mukana erilaisia ohjelmistokehitysmalleja käyttävissä projekteissa. Hänellä on kokemusta perinteisestä vesiputousmallista, ketterästä Scrum-mallista ja vesiputouksen sekä ketterän malli hybridistä.

6.2 Haastattelututkimuksen tulokset

Haastateltavien taustan selvittämisen jälkeen tutkimuksessa otetiin selvää, miten ohjelmistokehitysprojekteissa käytettävä malli valitaan ja mitä hyötyä sekä haittoja kehitysmallin ominaisuuksilla on. Vastausten perusteella saatiin selville, että ohjelmistokehitysmallin valintaan vaikuttaa hyvin moni asia, kuten esimerkiksi miten eri organisaatiot tekevät yhteistyötä. Jos koko kehitysprosessiin kuuluu ohjelmistokehityksen lisäksi esimerkiksi tuotehallintaa, joka käsittää konkreettisen laitteen tai järjestelmän, ei malli voi koskaan olla täysin ketterä. Tällöin hybridimalli voisi olla paras vaihtoehto tämänkaltaiseen prosessiin, sillä se mahdollistaa ohjelmistokehityksen ketterän tuottamisen, mutta lisää perinteisen mallin vaiheittaista muotoa. Perinteisen mallin ominaisuuksia tarvitaan, kun koko prosessiin liittyy esimerkiksi koneiston tai laitteiston suunnittelu ja toteutus ohjelmistokehityksen lisäksi.

Hybridimallin hyvä puoli on se, että ohjelmistoa voidaan kehittää teknisestä ja julkaisun näkökulmasta ketterästi, mutta samaan aikaan toimintamalli saadaan yhdistettyä hitaammin kääntyvään loppuorganisaatioon, eli organisaation osaan, joka ei haluaisi ketterää mallia käyttöön. Mallia voidaan käyttää myös silloin, kun perinteisestä mallista halutaan siirtyä ketterään malliin. Tällöin hybridimalli helpottaa siirtymävaiheessa organisaatiota sopeutumaan muutoksiin paremmin. Haasteena mallissa on tämä muun organisaation ketteryyden puute.

Ketteristä ohjelmistokehitysmalleista haastattelututkimuksella saatiin selville, että kehitettäessä ohjelmistoa, jonka toiminnan tarkoitus ei ole ohjata mitään laitetta, vaan ohjelmisto on itsessään tuote, toimii sen kehityksessä parhaiten jokin ketterä malli, esimerkiksi Scrum. Tällöin saadaan aikaan mahdollisimman nopeasti toimivaa ohjelmistoa, joka tuo lisäarvoa asiakkaalle ilman laajaa määrittelyä ja suunnittelua. Ketterää menetelmää käyttäessä myös hierarkiatasot ovat matalat, joka mahdollistaa sen, että ohjelmistokehittäjät voivat tehdä itse päätöksiä myös ilman projektipäällikön lupaa, joka lisää nopeutta kehitykseen. Vapaus ei tuo kuitenkaan lisäarvoa silloin, jos kyseessä on nuori ohjelmistokehittäjä, jolla ammattitaito ja uskallus eivät riitä päätösten tekoon itsenäisesti ja muutokset tulee varmistaa projektipäälliköltä, jotta virheitä ei synny.

Haastattelututkimuksessa viimeiseksi selvitettiin, että kokevatko ohjelmistokehittäjät, että jossakin ohjelmistokehityksessä olisi voitu käyttää jotain muuta mallia alkuperäisen mallin sijasta. Kumpikaan haastateltava ei kuitenkaan osannut sanoa toista mallia, joka olisi heidän kehitysprosesseihinsa toiminut. Tämän perusteella voidaan todeta, että hybridimalli sopii parhaiten laajan automaatiosysteemin kehittämiseen ja täysin ketterä malli puhtaan ohjelmistotuotteen kehittämiseen.

7. YHTEENVETO

Työn tavoitteena oli esitellä yleisimpiä ohjelmistokehitysmalleja ja kertoa niiden toiminnasta sekä historiasta. Työ suoritettiin kirjallisuuskatsauksena, johon yhdistettiin kirjoittajan itse tekemä haastattelututkimus. Kansainvälisistä julkaisuista löydettiin ajankohtaista tietoa työn aiheesta, ja haastattelututkimuksen avulla saatiin selville, kuinka eri ohjelmistokehitysmallit soveltuvat kehitysprojekteihin.

Työssä perehdyttiin perinteiseen ohjelmistokehitykseen ja sen yhteen tunnetuimpaan ohjelmistokehitysmalliin, vesiputousmalliin. Työssä käsiteltiin myös ketterää ohjelmistokehitystä ja sen kahta mallia, Scrumia ja XP:tä. Myös näiden mallien yhdistelmistä eli hybridimalleista kerrottiin ja lopuksi esiteltiin haastattelututkimuksen avulla, mitkä asiat vaikuttavat mallin valintaan ja miksi.

Työssä saatiin selville, että ennen 1990-lukua käytettiin lähinnä perinteisiä ohjelmistokehitysmalleja. Niissä kehitys oli vaiheittaista ja dokumentointi hallitsevaa. Näiden ominaisuuksien takia ei kuitenkaan pystytty kehittämään ohjelmistoa tehokkaasti, jos sen vaatimukset muuttuivat merkittävästi kehityksen aikana. Tämän takia 1990-luvulla alettiin käyttämään uusia ketterämpiä malleja, jotka vastasivat paremmin asiakkaiden muuttuviin vaatimuksiin.

Kirjallisuus- ja haastattelututkimuksella saatiin selville, että ketterät ohjelmistokehitysmallit soveltuvat parhaiten puhtaaseen ohjelmistotuotteen kehittämiseen. Molempien työssä esiteltyjen mallien Scrumin ja XP:n avulla ohjelmistoa voidaan kehittää iteratiivisesti useissa kierroksissa, jolloin ohjelmistoon voidaan lisätä vaatimuksia myös kehityksen aikana. Perinteinen vesiputousmalli soveltuu taas pieniin kehitysprojekteihin, joissa muutoksia ei tule lähes ollenkaan.

Hybridimallit sopivat parhaiten ohjelmistokehitykseen, jossa useat erilaiset organisaatiot, esimerkiksi laitteiden tuotehallinta ja ohjelmistokehitys, yhdistyvät. Niitä voidaan käyttää myös muutosvaiheessa, jossa siirrytään täysin perinteisestä mallista ketterään malliin. Hybridimallin haasteena on kuitenkin se, että jompikumpi puoli mallista, ketterä tai perinteinen, voi jää kokonaan pois, jos organisaation jäsenet eivät ole avoimia muutokselle.

Työn haastattelututkimukseen osallistui kaksi henkilöä, jonka takia sen perusteella tehdyt johtopäätökset ovat vain suuntaa antavia. Tutkimusta voisi kehittää haastattelemalla suurempaa joukkoa ohjelmistokehittäjiä, jolloin saaduista tuloksista voitaisiin tehdä luotettavampia johtopäätöksiä.

LÄHTEET

[1] R. Kneuper. Sixty Years of Software Development Life Cycle Models. IEEE Ann. Hist. Comput. 2017. vol. 39(3) pp. 41–54.

[2] V. Hema, S. Thota, S. Naresh Kumar, C. Padmaja, CB. Rama Krishna, K. Mahender. Scrum: An Effective Software Development Agile Tool. IOP conference series. Materials Science and Engineering. IOP Conf.Ser. Mater.Sci.Eng. 2020. vol. 981(2).

[3] S. Ergasheva, A. Kruglov. Software Development Life Cycle early phases and quality metrics: A Systematic Literature Review. Journal of physics. Conference series 2020. vol. 1694(1).

[4] NB. Ruparelia. Software development lifecycle models. Software Eng. Notes 2010. vol. 35(3) pp. 8–13.

[5] AM. Dima, MA. Maassen. From Waterfall to Agile software: Development models in the IT sector, 2006 to 2018. Impacts on company management. Journal of International Studies 2018. vol. 11(2).

[6] T. Stober, U. Hansmann. Agile Software Development: Best Practices for Large Software Development Projects. 1st ed. Berlin, Heidelberg: Springer-Verlag 2010.

[7] G. Kumar, P. Kumar Bhatia. Comparative Analysis of Software Engineering Models from Traditional to Modern Methodologies. Fourth International Conference on Advanced Computing & Communication Technologies 2014.

[8] RA. Haraty, G. Hu. Software process models: A review and analysis. International journal of engineering & technology 2018. vol. 7(2) pp. 325–331.

[9] W. Royce. Managing the development of large software system: concepts and techniques. In Proceedings of the 9th international conference on Software Engineering (ICSE '87). IEEE Computer Society Press 1987. pp. 328–338.

[10] J. Shore, S. Warden. The Art of Agile Development, 2nd ed. O'Reilly Media 2021.

[11] Chaos, tech. report, Standish Group Int'l 1994.

[12] K. Beck et al. Manifesto for Agile Software Development 2001. Luettu: 3.4.2021, saatavilla: https://agilemanifesto.org

[13] CG. Cobb. The Project Manager's Guide to Mastering Agile : Principles and Practices for an Adaptive Approach. New York: John Wiley & Sons Incorporated 2015.

[14] JL. Cooke. Everything you want to know about Agile : how to get Agile results in a less-than-agile organization. 1st ed. Ely, Cambridgeshire: IT Governance Publishing 2012.

[15] R. Kaur, J. Sengupta. Software Process Models and Analysis on Failure of Software Development Projects. IJSER 2013. vol. 2(2).

[16] K. Schwaber, J. Sutherland. The Scrum Guide 2010. Luettu: 6.4.2021, saatavilla: https://scrumguides.org/scrum-guide.html

[17] MS. Merkow. Secure, Resilient, and Agile Software Development. 1st ed. Milton: CRC Press 2020.

[18] A. Stellman, J. Greene, E. Volckhausen. Learning agile understanding Scrum, XP, Lean, and Kanban. 1st ed. Sebastopol, Calif: O'Reilly 2015.

[19] P. Krill. XP inventor talks about agile programming; Kent Beck touts shorter development cycles and says agile methods can detect unworkable software projects quicker. InfoWorld.com 2007.

[20] C. Andres, K. Beck. Extreme programming explained: embrace change. Addison-Wesley Professional 2004.

[21] F. Anwer, S. Aftab, S. Muhammad, U. Waaheed. Comparative Analysis of Two Popular Agile Process Models: Extreme Programming and Scrum. International Journal of Computer Science and Telecommunications 2017. vol. 8 pp. 1–7.

[22] P. Abrahamsson, O. Salo, J. Ronkainen, J. Warsta, Agile software development methods: Review and analysis, *VTT publ.* 2020. pp. 3–107.

[23] K. Beck. Extreme programming explained : embrace change. 1st ed. Boston: Addison-Wesley 2000.

[24] S. Belling. Succeeding with Agile Hybrids: Project Delivery Using Hybrid Methodologies. Apress 2020. pp. 3–14.

[25] M. Kuhrmann et al. Hybrid software and system development in practice: waterfall, scrum, and beyond. In Proceedings of the 2017 International Conference on Software and System Process(ICSSP 2017) Association for Computing Machinery 2017. pp. 30–39.

LIITTEET

Liite 1.

Haastattelukysymykset

1. Minkälaisissa ohjelmistokehitysprojekteissa olet ollut mukana / minkälaista ohjelmistoa olet kehittänyt ja kuinka kauan olet alalla työskennellyt?

2. Mitä kehitysmallia yleisimmin käytätte ohjelmistokehityksessä/ohjelmistokehitysprojekteissa?

3. Onko sinulla kokemusta monista eri ohjelmistokehitysmalleista (esim myös perinteisistä, kuten vesiputous), ja jos on niin mistä malleista?

4. Miten valitsette mallin, jota käytätte ohjelmistokehityksessä/projektissa?

Seuraavat kysymykset on tarkoitettu vastattavaksi yhteen tiettyyn valitsemaasi ohjelmistokehitysprojektiin liittyen, jossa olet ollut mukana:

5. Minkälaista ohjelmistoa kyseisessä projektissa kehitettiin ja mihin tarkoitukseen?

6. Mitä hyötyjä kyseiseen projektiin valitulla ohjelmistokehitysmallilla tavoiteltiin?

7. Oliko mallissa jo ennen ohjelmistokehityksen alkua huomattavissa joitakin mahdollisesti heikkoja puolia?

8. Mitä hyötyä mallilla huomattiin saavutettavan, kun ohjelmistokehitys oli valmis (esim laatuun tai nopeuteen liittyen)?

9. Mitä huonoja puolia mallista huomattiin ohjelmistokehityksen loputtua?

10. Oletko käyttänyt hybridimalleja, eli esimerkiksi onko jossain projektissa ollut käytössä ketterän ja perinteisen mallin yhdistelmä (esim Agile-Vesiputous hybridi)? Jos olet, niin miksi kyseiseen projektiin valittiin hybridimalli?

11. Mitä mieltä olet hybridimalleista, mitä hyviä ja huonoja puolia niissä mielestäsi on tai olet huomannut niissä olevan?

12. Olisiko jossain ohjelmistokehitysprojektissa, jossa olet ollut mukana, voitu mielestäsi käyttää hybridimallia yhden mallin sijaan, jotta olisi saavutettu parempi lopputulos?