SUUNNITTELUPROJEKTIN HALLINTA JA VAIHEISTUS

Sovellusprojektin suunnittelumenetelmät ja niiden käytettävyys

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Ammattikorkeakoulun opinnäytetyö

Mediatekniikan koulutusohjelma

Riihimäki 31.7.2011

Saara Koivisto

HAMK
HÄMEEN AMMATTIKORKEAKOULU

OPINNÄYTETYÖ

Mediatekniikka
Riihimäki

Työn nimi	Suunnitteluprojektin hallinta ja vaiheistus
Tekijä	Saara Koivisto
Ohjaava opettaja	Raimo Hälinen
Hyväksytty	31.7.2011
Hyväksyjä	

HAMK
HÄMEEN AMMATTIKORKEAKOULU

TIIVISTELMÄ

RIIHIMÄKI
Mediatekniikka

Tekijä Saara Koivisto **Vuosi** 2011

Työn nimi Suunnitteluprojektin hallinta ja vaiheistus

TIIVISTELMÄ

Tämän opinnäytetyön tavoitteena on toteuttaa kirjallisuustutkielmana pienimuotoinen käsikirja sovellusprojektin vaiheista, sovelluskehityksessä yleisimmin käytettävistä menetelmistä sekä menetelmien käytettävyydestä. Opinnäytetyön tietoperustana käytetään useita aihetta käsitteleviä sähköisiä lähteitä. Työhön ei sisälly mitään kehitystutkimusta tai muuta käytännön projektia.

Opinnäytetyössä käsitellään aluksi projektityöskentelyä ja projektin etenemistä yleisellä tasolla. Ohjelmistotaloissa projektimuotoinen työskentely on omaksuttu osaksi arkipäivää sen tarjoaman joustavan ja tehokkaan resurssien käytön ansiosta. Ohjelmistokehitys itsessään on omaksunut piirteitä perinteisestä projektityöskentelystä eri kehitysmenetelmien muodossa, joista kaikista löytyy yhteisinä ominaisuuksina ainakin jonkinlaiset suunnittelu-, toteutus-, testaus- ja käyttöönotto- sekä ylläpitovaiheet.

Sovelluskehitysmenetelmät voidaan jakaa kahteen eri ryhmään niiden tuottaman dokumentoinnin määrän perusteella. Suunnitelmaohjautuvat kehitysmenetelmät rakentuvat tarkasti dokumentoitujen suunnitelmien varaan, kun taas ketterät kehitysmenetelmät perustuvat ihmiskeskeiselle kommunikoinnille.

Keskeisintä tässä työssä on ollut erilaisiin sovelluskehitysmenetelmiin perehtyminen ja niistä oppiminen. Sovelluskehitysmenetelmiä on tänä päivänä todella paljon, joten työn rajaaminen sai osakseen erityistä huomiota.

Avainsanat Ohjelmistokehitys, ketterät menetelmät, suunnitelmaohjautuvat menetelmät, projektityöskentely

Sivut 34 s, + liitteet 1 s.

HAMK
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Riihimäki
Media Technology of degree programme

Author Saara Koivisto **Year** 2011

Subject of Bachelor's thesis Management and Stages of Software Planning Project

ABSTRACT

The aim of this thesis is to implement a small-scale handbook as a literature survey about the different stages of a software development project, the most commonly used methods in software development and the usability of those methods. There is no empirical part in the study.

The thesis begins with a brief introduction to project management and its progression on a general level. Software development has adapted features from traditional project management in the forms of different development methods which all have at least some sort of planning, production, testing and deployment phase in common.

Software development methods can be roughly split into two groups based on the amount of documentation they produce during the development process. Plan-driven development methods are based on carefully drafted, long-term plans whereas agile development methods rely on more people-centric forms of communication.

Keywords Software development, agile methods, plan-driven methods, project management

Pages 34 p + appendices 1 p.

SISÄLLYS

1 JOHDANTO 1

2 PROJEKTITOIMINTA 2

2.1 Yleistä projekteista 2
2.2 Ideasta suunnitelmaksi 2
2.2.1 Esiselvitys ja projektiehdotus 3
2.2.2 Päätös projektin käynnistämisestä 4
2.2.3 Projektisuunnitelma 4
2.3 Seuranta 5
2.4 Ositus, vaiheistus ja aikaohjaus 6
2.5 Riskien hallinta 6
2.6 Testaus 7
2.6.1 Testaus ohjelmistokehityksessä 9
2.6.2 Ohjelmistokehityksen testaustasot ja –menetelmät 9
2.7 Raportointi ja dokumentointi 10
2.7.1 Dokumentointi ohjelmistotuotannossa 11

3 OHJELMISTOKEHITYKSEN SUUNNITTELUMENETELMÄT 13

3.1 Ohjelmistokehityksen elinkaari 14
3.2 Suunnitelmaohjautuvat kehitysmenetelmät 14
3.2.1 Vesiputousmalli 15
3.2.2 Prototyyppimalli 17
3.2.3 Spiraalimalli 19
3.3 Ketterä ohjelmistokehitys 21
3.3.1 Yleiset perusarvot ja periaatteet 22
3.3.2 Extreme Programming 23
3.3.3 Crystal-metodologiaperhe 26
3.3.4 Scrum 28
3.4 Suunnitelmaohjautuvien ja ketterien menetelmien käytettävyys ja vertailu 29

4 LOPPUSANAT 34

LÄHTEET 36

Liite 1 Agile-manifesti

1 JOHDANTO

Tässä työssä käydään läpi projektitoimintaa ja projektin etenemistä yleisellä tasolla sekä keskeisimpiä sovelluskehityksen menetelmiä ja niiden käytettävyyttä.

Sanalla projekti tarkoitetaan yleensä jotakin tarkkaan suunniteltua hanketta tietyn päämäärän saavuttamiseksi. Aikojen saatossa projekteista on tullut keskeinen osa eri alojen, kuten myös ohjelmistotuotannon, yritystoimintaa niiden joustavien sekä kustannustehokkaiden työmenetelmien johdosta. Projekti etenee erimittaisissa aikajaksoissa, joita perinteisesti ovat suunnittelu-, toteutus-, testaus- sekä käyttöönottovaihe, ja monesti ohjelmistokehityksen ollessa kyseessä, vielä ylläpitovaihe.

Ohjelmistokehitysmenetelmien kehittymisen taustalla oli laaja ohjelmistokriisi 1960-luvun lopulla, jolloin todettiin entisten ohjelmistokehitysmenetelmien olevan liian hitaita uusille tietojärjestelmille. Kehitysprojektit myöhästelivät, ylittivät kustannuksensa ja tuottivat hankalasti ylläpidettäviä järjestelmiä.

Erilaisten ohjelmistokehitysmenetelmien kehittymiseen ja erilaistumiseen ovat vaikuttaneet etenkin järjestelmän tilaajien, toisin sanoen asiakkaiden erilaiset tarpeet. Toisille nopea aikataulu on keskeistä, toiset arvostavat ennen kaikkea laatua kustannuksista ja aikatauluista välittämättä.

Sovelluskehitysmenetelmät voidaan karkeasti jakaa suunnitelmaohjautuviin ja ketteriin kehitysmenetelmiin niiden tuottaman dokumentoinnin perusteella. Suunnitelmaohjautuvat menetelmät suosivat nimensä mukaisesti pitkälle tähtäimelle tehtyjä ja tarkasti dokumentoituja toteutussuunnitelmia, joissa tulevaisuuden muutokset on pyritty ennakoimaan. Tunnetuin suunnitelmaohjautuva elinkaarimalli on vesiputousmalli.

Ketterät kehitysmenetelmät puolestaan perustuvat ihmisten väliselle kommunikaatiolle sekä ajatukselle, että tulevaisuutta ei voida ennustaa, joten pitkän tähtäimen suunnitelmat eivät pidä. Sen vuoksi suunnitellaan ja toteutetaan vain sen verran kuin on tällä hetkellä tarpeen. Tunnetuin ketterän kehityksen menetelmä on Extreme Programming.

Sovelluskehitykseen käytettävän menetelmän valinnassa on huomioitava tulevan sovelluksen käyttötarkoitus, projektin laajuus, tavoiteltu laatu sekä aika ja taloudelliset resurssit.

2 PROJEKTITOIMINTA

Projektitoiminta on etenkin ohjelmistokehityksessä yleistynyt työskentelymuoto, joka noudattaa elinkaariluonnetta, toisin sanoen sillä on alku ja loppu. Projektityöskentelyä voisi kuvata seuraavanlaisella luettelolla:

1. Projektilla on tilaaja.
2. Projekti on ajallisesti ja laajuudeltaan sidottu.
3. Projektilla on selkeä tavoite.
4. Dokumentointia harjoitetaan koko projektin ajan.
5. Projektista on laadittu kuvaus ja sen aikana noudatetaan sovittuja työskentelymuotoja, kuten
   - projektikokoukset
   - projektiin nimetyt henkilöt
   - jatkuva tiedottaminen
   - välitavoitteet ja tarkistuspisteet
   - työsuunnitelmat
   - työn- ja vastuunjaon selvitys.

(Louhelainen 2008)

2.1 Yleistä projekteista

Projektilla tarkoitetaan tarkkaan suunniteltua hanketta tietyn päämäärän saavuttamiseksi. Projekteille ominaista on eteneminen aikajanan tavoin, jolloin toiminnat suoritetaan vain kerran. Kun projekti on päätetty aloittaa, on sille asetettava selkeä tavoite, jonka saavuttamiseksi tehdään projektisuunnitelma. Tärkeimmät tavoitteet liittyvät aikaan, rahaan ja tulokseen. Tavoitteiden saavuttamista tukevat suunnitelmallisuus, organisointi, ohjaus ja työkalut, joiden avulla tavoitteet saavutetaan käytössä olevien resurssien avulla. (Ahlstedt 2010)

Toteutuksessa apuvälineiksi tarvitaan myös projektihallinnan työkaluja sekä mittareita, joiden avulla voidaan nähdä, miten tavoitteita on saavutettu. Työkalut ovat tarpeellisia myös projektien kansainvälistymisen vuoksi. Projektit ovat laajentuneet monia yrityksiä koskeviksi, jolloin niiden hallinta on vaikeutunut. (Ahlstedt 2010)

Tänä päivänä IT-yrityksissä projektit ovat yhä useammin monimutkaisia, pitkäaikaisia ja strategisia kehittämisohjelmia, joilla tähdätään tietojärjestelmien tai tuotevalikoimien muutokseen. (Ahlstedt 2010)

2.2 Ideasta suunnitelmaksi

Projekti syntyy tavallisesti ideasta. Idea taas saa alkunsa kun yrityksessä tai organisaatiossa ilmenee jokin ongelma tai tarve, johon halutaan ratkai-

su. Projektilla pyritään siis saamaan aikaan muutos, parannus tai kokonaan uusi asia yritykseen. (Louhelainen 2008)

Ideoinnista seuraava askel on visiointi. Projektityöskentelyssä visiointi ei ole suunnitelma, vaan väljä näkemys tai mielikuva asioiden mahdollisesta kulusta. Toisin sanoen visio on mielikuva halutusta tulevaisuudesta ja visio etsii ratkaisuvaihtoehtoja, joista voisi löytyä toteuttamiskelpoinen ajatus. (Louhelainen 2008)

Kun idea jostain on syntynyt, sen edellytyksiä toteuttaa projektina aletaan valmistella. Tämän päätöksen tekee asettaja, joka nimeää projektille valmistelijan omasta organisaatiosta. Valmistelija tekee esiselvityksen, jonka pohjalta muodostuu projektiehdotus ja vasta tämän jälkeen tehdään päätös projektin käynnistämisestä tai käynnistämättä jättämisestä. Päätöksen tekee asettaja, joka on projektin ylin päättävä henkilö. (Louhelainen 2008)

2.2.1 Esiselvitys ja projektiehdotus

Esiselvityksen tehtävänä on käydä läpi ne asiat, joilla varmistetaan, että projekti on toteuttamiskelpoinen. Tarvitaan tietoa taloudellisista, teknisistä, sosiaalisista, oikeudellisista ja ekologisista näkökohdista. Erityisesti on kiinnitettävä huomiota siihen, mikä on asiakkaan toive, mutta esiselvitys sisältää myös seuraavien asioiden huomioimista:

- asiakkaan tilan ja tarpeiden selvittäminen
- tutustuminen aikaisempiin vastaaviin projekteihin omassa yrityksessä ja kilpailijoilla
- aineellisten ja henkisten voimavarojen kartoitus
- sopivien projektihenkilöiden kartoitus (ammattitaito, luotettavuus, yhteistyökyky)
- tiedon haku projektin alueelta.

(Louhelainen 2008)

Esiselvitys tuottaa projektiehdotuksen, jossa tehdään jo rajaus projektiin kuuluvista asioista. Esiselvitys on strategisten valintojen paikka. Tässä vaiheessa valitaan suunnat, kumppanit sekä myös jonkin verran projektihenkilöitä. (Louhelainen 2008)

Projektiehdotuksessa kuvataan lopputuote ja sen ominaisuudet ja ehdotus ottaa myös kantaa siihen, mitä hyötyä projekti synnyttää. Projektiehdotus päätetään asettamiskirjeeseen, jolla haetaan projektille käynnistyslupaa. (Louhelainen 2008)

Heti suunnittelun alkumetreillä on hyvä pitää integraatiopalaveri organisaation eri alojen edustajien kesken. Palaverissa selvitetään osastojen valmiuksia projektin eteenpäin viemiseksi. Esimerkkinä mukana voisi olla materiaaliosasto, hankintaosasto ja taloushallinto. Projekti käyttää omiin toimintoihinsa joka tapauksessa runsaasti näiden osastojen palveluja. (Louhelainen 2008)

2.2.2 Päätös projektin käynnistämisestä

Projektin kannalta vastuullisin vaihe on siitä päättäminen. Päätöstä ei pidä missään nimessä tehdä hetken mielijohteesta, vaan se täytyy tulla johdonmukaisena prosessina. (Louhelainen 2008)

Päätöksentekoprosessissa käydään läpi valmisteluvaiheessa tehtyjä asioita. Projektipäätös sisältää:

1. ongelman tunnistamisen, tutkimisen ja määrittelyn
2. projektin ideat ongelman ratkaisemiseksi ja niistä syntyneet visiot
3. taustaselvitykset ja vaihtoehdot
4. tavoitteen määrittelyn
5. vaihtoehtojen vertailun suhteessa resursseihin, tavoitteisiin ja riskeihin
6. päätöksenteon, toimeenpanon ja seurannan.

(Louhelainen 2008)

Päätöksen projektin käynnistämisestä tekee asettaja esiselvityksen ja projektiehdotuksen perusteella. Kun päätös on tehty, käynnistetään projektisuunnitelman laatiminen. (Louhelainen 2008)

2.2.3 Projektisuunnitelma

Projektisuunnitelman tarkoitus on jäsentää kaikille mitä ollaan tekemässä, kuka tekee ja mihin mennessä. Tarkoitus on rajata mahdollisimman tarkasti projektille kuuluvat tehtävät sekä myös ne tehtävät, jotka eivät kuulu projektille. Siinä ei kuitenkaan suunnitella vielä lopputuotetta. Oleellista on, että projektisuunnitelmaa ei tehdä yhtään enempää kuin on tarpeen ja sisältö muokataan tarkoituksenmukaiseksi. Tärkeää on löytää suunnitelmasta vastauksia kysymyksiin kuka, mitä, milloin, miten, millä resurssilla ja miksi. (Louhelainen 2008)

Projektin suunnittelu käsittää seuraavat osiot:

- Organisaation ja hallinnon suunnittelu
  - organisaation luominen
  - tulosten jakautuminen organisaatiossa
  - ohjausryhmän asettaminen
  - johtamisen suunnittelu
- Resurssi- ja kustannussuunnittelu
  - resurssi- ja kustannussuunnitelman laatiminen (henkilöt, aika, raha)
  - rahoitus ja taloushallinnon järjestäminen
  - keston määrittäminen
  - riskinotto
- Toteutussuunnitelman laatiminen
  - etenemissuunnitelma
  - työvaiheitten suunnitelma.

(Louhelainen 2008)

Projektin suunnittelu etukäteen on olennaista, sillä projektilla on joka tapauksessa vain rajallisesti resursseja käytettävissä ja tietty tavoite täytyy saavuttaa niiden avulla määrätyssä ajassa. Suunnittelun tarkoituksena on koordinoida resursseja sekä hyödyntää niitä oikein, arvioida aikatauluja, lisätä tehokkuutta, välttää hätiköintiä ja havaita poikkeamia tavoitteissa. (Louhelainen 2008)

Aluksi suunnitellaan projektia vain sen verran kuin on projektin läpiviemiseksi välttämätöntä. Projekti elää ja kehittyy joka tapauksessa matkan varrella, jolloin liian tarkat suunnitelmat romuttuvat ja kuluttavat näin aikaa sekä rahaa turhaan.

Projektisuunnitelman perustana toimii sen runko, jossa hahmottuvat projektin yleisosat ja sen työkokonaisuudet. Runko puretaan osatehtäviksi jakamalla rungossa kuvatut työkokonaisuudet ensin tehtäväkokonaisuuksiksi ja ne edelleen osatehtäviksi. Kun tehtäväkohtaiset työsuunnitelmat suhteutetaan toisiinsa, saadaan kokonaisaikataulu. Aina kun työsuunnitelma tarkentuu, se päivitetään myös runkosuunnitelmaan. (Louhelainen 2008)

Projektisuunnitelmaa täydennetään myöhemmin tarvittavilla suunnitelmilla, joita voivat olla esimerkiksi testaussuunnitelma tai käyttöönottosuunnitelma. Muita projektisuunnitelmaa täydentäviä dokumentteja ovat budjetit, riskiarviot ja laatudokumentit, jotka yleensä on hyvä lisätä omiksi liitteiksi niiden jatkuvien muutosten takia. (Louhelainen 2008)

Projektisuunnitelman ja työsuunnitelmien tehtävänä on lopputuotteen valmistuttamisen lisäksi toimia seurannan ja valvonnan apuvälineenä sekä palvella perusorganisaatiota kokonaissuunnittelussa. (Louhelainen 2008)

Valmis projektisuunnitelma täytyy hyväksyä ja katselmoida. Katselmoinnin suorittaa koko projektihenkilöstö ja sen tavoitteita ovat:

- projektihenkilöstön tutustuttaminen toisiinsa
- projektin taustan, tavoitteiden ja viitekehyksien tiedottaminen
- projektin lähtötietojen ja projektisuunnitelman läpikäynti
- osaamis- ja koulutustarpeiden analysointi
- projektin periaatteiden ja menetelmien todentaminen
- resurssien todentaminen (tilat, välineet, varat yms.)
- projektin viestintäsuunnitelman ja dokumentoinnin periaatteiden läpikäynti
- projektin tietoturvamenettelystä ja julkisuusasteesta tiedottaminen.

(Louhelainen 2008)

2.3 Seuranta

Seuranta on osa projektinhallintaa, jolla tarkoitetaan ajan tasalla pysymistä tavoitteiden saavuttamiseksi. Seuranta voidaan jakaa eri muuttujien, kuten esimerkiksi ajan ja rahan seurantaan, joka tapahtuu ohjausryhmän valvon-

nan sekä toistuvan raportoinnin avulla. Suunnitelmat projektin suhteen muuttuvat ja tarkentuvat työn edetessä, jonka vuoksi pitkäkestoisissa projekteissa seuranta on entistä tärkeämpää. Seurannan avulla varmistetaan projektisuunnitelman mukainen eteneminen ja työn laatu vertaamalla projektin toteutumista suunnitelmaan jo projektin aikana. Tällöin on mahdollista puuttua projektin etenemiseen, mikäli eroavaisuuksia huomataan.

Projektiryhmä voi seurata työn etenemistä viikoittaisten kokousten avulla, jotka etenevät etukäteen suunniteltujen asialistojen mukaan. Kokousten avulla huomataan poikkeamat suunnitelluista aikatauluista. Mahdollisten poikkeamien syyt käsitellään projektipäällikön toimesta ja jälkikäteen kokouksesta tehdään muistio, joka välitetään osallistujille. (Ahlstedt 2010)

2.4 Ositus, vaiheistus ja aikaohjaus

Tärkein työkalu pitkäkestoisen projektin jäsentämiseen ja seurannan suorittamiseen on osittaminen eli projektin jakaminen pienempiin itsenäisiin osaprojekteihin (Ahlstedt 2010). Osittelu voidaan tehdä tuotteen rakenteen tai sen toteutusvaiheiden mukaan tai sitten toiminnallisten kokonaisuuksien mukaan (Louhelainen 2008). Osittaminen helpottaa myös vastuun ja tehtävien jakamista sekä aikataulujen suunnittelua. Jokaiselle projektin osavaiheelle asetetaan oma tavoite, jonka tulos on tultava esiin vaiheen päättyessä. (Ahlstedt 2010)

Vaiheistuksen toteuttamisen apuna voidaan käyttää aikaohjausta, jossa tehtäville määritellään työntekijän kokemuksen mukaan suoritusaika ja aloituspäivämäärä. Jokaiselle tehtävälle on jätettävä myös pelivara odottamattomia muutoksia varten, sillä projekti voi myöhästyä myös yrityksen ulkopuolisista ja yrityksestä riippumattomista tekijöistä johtuen. (Ahlstedt 2010)

Aikataulujen laatiminen on sitä vaativampaa, mitä laajemmasta projektista on kyse. Projektilla voi samanaikaisesti olla käytössä useampia aikatauluja kuten yleis- tai päiväkohtaisia aikatauluja. Yleisaikataulu on jaettu päätehtävien ja välietappien mukaisesti ja aikajaksot saattavat olla useammankin kuukauden mittaisia. Pitkissä, yli vuoden projekteissa suunnitelmat tarkentuvat projektin edetessä ja tästä syystä aikataulujen tarkistaminen ajoittain on tärkeää. (Ahlstedt 2010)

2.5 Riskien hallinta

Riskien hallinnassa nimetään ja analysoidaan riskit, jotka ovat todennäköisiä ja tapahtuessaan aiheuttavat projektille haittaa sekä lisäkustannuksia. Riskit tulee kartoittaa jo suunnitteluvaiheessa, mikä auttaa niiden ehkäisyä. (Ahlstedt 2010) Tavallisimpia riskityyppejä ovat:

- henkilöriskit
- yhteistoiminnan konfliktiriskit
- projektin sisäinen riski (aikataulu, tietotekniikka, talous, suunnitteluvirheet, toteutusvirheet)

- markkinariskit
- teknologiariskit (valitun teknologian vanheneminen)
- ympäristötekijöiden aiheuttamat riskit
- sopimusriskit
- poliittinen riski
- asiakasriski (tuote jää ilman ostajaa)

(Louhelainen 2008)

Riskit on asetettava järjestykseen, mikä tapahtuu kertomalla riskin vaikutus sen todennäköisyydellä. Käytännöllinen apuväline arviointiin on alla oleva riskitaulukko.

Tapahtuman todennäköisyys	Tapahtuman seuraukset		
	Vähäiset	Haitalliset	Vakavat
Epätodennäköinen	1. Merkityksetön riski	2. Vähäinen riski	3. Kohtalainen riski
Mahdollinen	2. Vähäinen riski	3. Kohtalainen riski	4. Merkittävä riski
Todennäköinen	3. Kohtalainen riski	4. Merkittävä riski	5. Sietämätön riski

Kuvio 1. Riskitaulukko (PK-RH 2009)

Tehtyjen selvitysten perusteella valitaan ensiksi seurausten vakavuus taulukon ylimmältä riviltä ja sen jälkeen tapahtuman todennäköisyys ensimmäisestä sarakkeesta. Riski on valittujen kohtien leikkauspisteessä olevan arvon suuruinen. Näin riskin suuruus saa pienimmillään arvon 1 (Merkityksetön riski) ja suurimmillaan arvon 5 (Sietämätön riski). (PK-RH 2009)

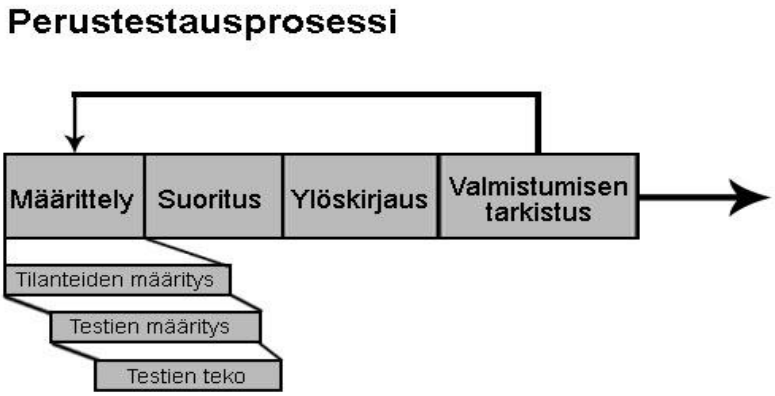
Mitä vakavammasta riskistä on kysymys, sitä enemmän se tulee huomioida niin suunnittelussa kuin toteutuksessakin. Mikäli riskit ovat suuria, saattaa koko projekti jäädä käynnistymättä. Riskit tulee tarkistaa säännöllisin väliajoin, sillä riskien vaikutus projektiin muuttuu työn edetessä. (Ahlstedt 2010)

2.6 Testaus

Testaus on vikojen järjestelmällistä etsimistä, laadun varmistamista, todentamista ja kelpuuttamista. Päällimmäisin syy on tuotteen toiminnallisuuden ja laadun varmistaminen sekä tuotekehityksen vauhdittaminen. (Stankovic 2010)

Testauksen aloittaminen mahdollisimman aikaisin säästää paljon vaivaa ja rahaa. Mitä varhaisemmassa vaiheessa virheitä löydetään, sitä helpommin ja halvemmalla ne ovat yleensä korjattavissa. Testaussuunnitelma on tämän takia hyvä tehdä heti projektin alussa, samalla kun projektille tehdään määrittelyä. Hyvällä, oikein kohdennetulla testauksella optimoidaan liiketoiminnalle aiheutuvia kustannuksia ja maksimoidaan siitä saatava hyöty. (Stankovic 2010)

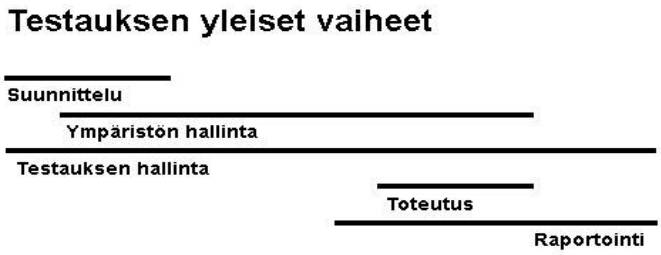
Testausprosessi voidaan jaotella ISEB:n (Informational Systems Examinations Board) määrittelemän perustestausprosessin mukaisesti (kuvio 2).



Kuvio 2. Perustestausprosessi (ISEB – Information Systems Examinations Board). (Stankovic 2010)

Määrittelyvaiheessa määritellään testaustilanteet, esimerkiksi sovellusjärjestelmän eri osa-alueet. Tämän jälkeen suunnitellaan erilaisia testejä, joita tullaan suorittamaan. Tässä vaiheessa päätetään, mitä ja kuinka testataan, priorisoidaan tärkeimmät testitapaukset sekä asetetaan aikataulu. Määrittelemisen jälkeen siirrytään varsinaiseen suoritukseen eli suunniteltujen testien tekoon. (Stankovic 2010)

Testauksen yleiset vaiheet voidaan jakaa suunnitteluun, ympäristön hallintaan, testauksen hallintaan sekä testaukseen ja raportointiin, kuten kuviossa 3 on esitetty.



Kuvio 3. Testauksen yleiset vaiheet. (Stankovic 2010)

Suunnittelu, ympäristön hallinta ja toteutuksen hallinta kuuluvat kuviossa 2 esitettyyn määrittelyvaiheeseen. Suunnitteluvaiheessa määritellään kuinka hallitaan ympäristöä ja testausta, ts. suunnitellaan testausprosessin kulku.

Toteutuksella tarkoitetaan suunniteltujen testien tekemistä ja toteutus kuuluu kuviossa 2 esitettyyn suoritusvaiheeseen.

Raportointivaiheessa tehdään testiraporttipohjat, testiraportit täytetään ja virheraportit luovutetaan ohjelmoijalle. Raportointi kuuluu kuviossa 2 esitettyyn suoritus-, ylöskirjaus- ja valmistumisen tarkistusvaiheeseen.

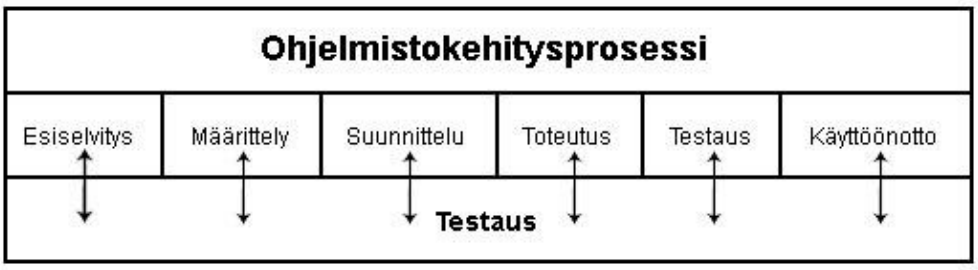
Testauksen hallinta kestää koko testausprosessin ajan. Siihen kuuluvat kaikki niin kuviossa 2 kuin kuviossa 3:kin esitetyt perusprosessin vaiheet. (Stankovic 2010)

2.6.1 Testaus ohjelmistokehityksessä

Ohjelmistokehityksessä testausta tehdään koko ohjelmiston elinkaaren ajan, kuten kuviossa 4 on esitetty. Ohjelmistokehitysprosessin aikana tehtävä testaus voidaan jakaa seuraaviin vaiheisiin:

- määrittelyn ja suunnittelun aikainen testaus
- kehityksen aikainen testaus
- ylläpidon aikainen testaus

(Stankovic 2010)



Kuvio 4. Testaus ohjelmistokehitysprosessissa. (Stankovic 2010)

Määrittelyn ja suunnittelun aikainen testaus suoritetaan staattisena ja dynaamisena testauksena. Staattiseen testaukseen kuuluu vaatimusmäärittelyn, toiminnallisen määrittelyn, teknisen suunnittelun ja testaussuunnittelun dokumenttien testaus katselmoinnin avulla. Dynaamiseen testaukseen kuuluu mallien simulointi, suorituskyvyn analysointi ja prototyyppien testaus. (Stankovic 2010)

2.6.2 Ohjelmistokehityksen testaustasot ja –menetelmät

Ohjelmistokehityksessä testausmenetelmät ja –tasot usein lajitellaan kehitysprosessin vaiheiden tai testattavan kohteen mukaan. Testitapausten määrittely tapahtuu valitsemalla ensin testimenetelmä. Yleisimmin käytetään lasilaatikkotestausta (White Box Testing), mustalaatikkotestausta (Black Box Testing) tai harmaalaatikkotestausta (Grey Box Testing). (Stankovic 2010)

Moduulitestaus on järjestelmän pienin testattavissa oleva osa, joka on kooltaan tyypillisesti alle 1000 koodiriviä ja yleensä yhden ohjelmoijan tekemä. Testauksella pyritään osoittamaan moduulin vaatimusten mukai-

suus tai vaihtoehtoisesti korjauksen tarve. Virheitä kutsutaan yksikkövirheiksi. Koska moduulitestauksessa keskitytään vain pienempien osien testaukseen, se ei sovellu järjestelmän ainoaksi testausmenetelmäksi. Moduulitestauksen yleisin menetelmä on *lasilaatikkotestaus* (White box testing), joka perustuu ohjelmakoodin tutkimiseen. Lasilaatikkotestauksen tavoitteena on valita testitapaukset siten, että ohjelman eri polkuja käydään läpi mahdollisimman monta. (Stankovic 2010)

Integraatiotestauksessa moduuleja yhdistetään suuremmiksi kokonaisuuksiksi, rajapinnoiksi, joita testaamalla varmistetaan, että moduulit toimivat oikein yhdessä. Moduuleja voidaan lisätä kahdella tavalla: joko yksittäin (incremental) tai kaikki kerralla (non-incremental). Yksittäin lisäävässä menetelmässä testataan aina lisättävä moduuli ja kun se on todettu toimivaksi, lisätään seuraava. Vaihtoehtoisesti yhdistettäessä kaikki moduulit kerralla, tulee koko järjestelmä testata. Tämä on kuitenkin haastavampi ja samalla heikompi menetelmä, sillä virheiden jäljittäminen on vaikeampaa. Integraatiotestauksen yleisin menetelmä on *mustalaatikkotestaus* (Black box testing), jossa testitapaukset valitaan usein määrittelyiden perusteella. Menetelmä on tehokas suuremmille koodimäärille ja testitapauksia tehdään myös käyttäjän näkökulmasta. Testitapausten valintamenetelmät ovat ekvivalenssiosoitus, arvotestaus, syy-seurausanalyysi ja virheenarvaus. Mustalaatikkotestausta käytetään integraatiotestauksen lisäksi myös järjestelmä- ja hyväksymistestauksissa. (Stankovic 2010)

Systeemitestauksessa (järjestelmätestaus) järjestelmää testataan kokonaisuudessaan sen käyttötarkoitusta vastaavassa ympäristössä. Järjestelmää testataan loppukäyttäjän näkökulmasta ja testauksella pyritään varmistamaan, että koko järjestelmä toimii määriteltyjen vaatimusten mukaisesti. Systeemitestaukseen kuuluu myös *ei-toiminnallinen testaus* (Non-functional testing), jolla testataan järjestelmän suorituskykyä, turvallisuutta ja käyttöoikeuksia. Systeemitestauksen yleisin menetelmä on *harmaalaatikkotestaus* (Grey box testing), jossa on pyritty yhdistämään lasi- ja mustalaatikkotestausmenetelmän hyviä puolia. Harmaalaatikkotestaus on tärkeä testaustapa esimerkiksi web-sovelluksissa. (Stankovic 2010)

Hyväksymistestaus suoritetaan ajallisesti viimeisimpänä. Hyväksymistestauksen suorittavat asiakkaat ja/tai käyttäjät ja testauksen tarkoituksena on lisätä käyttäjän luottamusta järjestelmään. (Stankovic 2010)

Virheiden korjausten jälkeen korjattuja järjestelmän osia testataan uudelleen, jotta ne voidaan varmistaa virheettömiksi. Tällaista testausta kutsutaan *regressiotestaukseksi* (Regression testing). Mikäli regressiotestaus suoritetaan huolimattomasti, voi seurauksena olla suurempia ja työläämmin korjattavissa olevia ongelmia. (Stankovic 2010)

2.7 Raportointi ja dokumentointi

Raportointi on ensisijaisesti projektin johtamisen työkalu. Tunnetuimmat raportit lienevät määräajoin tehtävät tilanneraportit sekä johtoryhmän tilannekatsaukset. Tilanneraporttien avulla nähdään projektisuunnitelman ja

toteutuneiden suunnitelmien mahdolliset erot. Kuitenkin lukujen erojen sijaan olisi hyvä panostaa niiden ennaltaehkäisyyn. (Ahlstedt 2010)

Raportit tulisi tehdä siten, että sama raportti toimii tilannekatsausraporttina niin projektiryhmälle kuin ohjausryhmällekin. Tiedon on oltava pitkälle jalostettua, missä paneudutaan vain tärkeimpiin tavoitteisiin. Raporttien on oltava selkeitä ja totuudenmukaisia ja tekemättömien tehtävien on tultava selkeästi esille. (Ahlstedt 2010)

Dokumentti on yleiskielessä synonyymi käsitteille asiakirja, todistuskappale. Perinteisesti dokumentilla tarkoitetaan joko kirjoitettua tai tulostettua paperia tai artefaktia, joka tuo esiin todistusvoimaista informaatiota, esimerkiksi passi, sopimus tai lasku. Laajemmin ymmärrettynä dokumentti sanalla voidaan viitata mihin tahansa kirjalliseen tuotokseen, jolla on informatiivista arvoa. (Määttä & Rautio 2010)

Dokumentit arkistoidaan projektikansioon, joka on projektille perustettu sähköinen hakemisto. Projektikansio toimii osana projektiviestintää ja samalla kansio auttaa projektia koskevan materiaalin hallintaa, sekä siihen palaamista projektin jälkeen. (Ahlstedt 2010)

2.7.1 Dokumentointi ohjelmistotuotannossa

Ohjelmistotyölle on tyypillistä, että projektin aikana kertynyttä tietoa kirjataan dokumentin muotoon. Ohjelmistosuunnittelu koostuu kahdesta vaiheesta. Näistä ensimmäinen on *toiminnallinen määrittely*, jossa kuvataan kaikki järjestelmän toteuttamat toiminnot ja liitännät järjestelmän ulkopuolelle. Vaiheen tuloksena syntyvä *määrittelydokumentti* kuvaa siis mitä kaikkea järjestelmällä voi tehdä ja miten sitä käytetään. Dokumentti ei kuitenkaan ota kantaa siihen, miten kaikki toiminnot tulee toteuttaa. (Määttä & Rautio 2010; Wikipedia 2011)

Toiminnallisen määrittelyn vastakohta on *tekninen määrittely*. Tekninen määrittely eli *arkkitehtuurisuunnittelu* seuraa toiminnallisen määrittelyn jälkeen ja siihen on sisällytetty kuvaus siitä, miten kaikki halutut toiminnot tullaan toteuttamaan. Tekniseen määrittelyyn sisältyy esimerkiksi ohjelmiston toteutuksessa käytettävä ohjelmointikieli, ohjelmistokomponentit, kuten kirjastot, ja niiden välinen hierarkia. (Wikipedia 2011)

Toiminnallisen määrittelyn ja teknisen määrittelyn pohjalta voidaan laatia *testaussuunnitelmat*, joita käytetään myöhemmin ohjelmiston testausvaiheessa etsittäessä virheitä. Mikäli testaustulosten perusteella täytyy palata muuttamaan esimerkiksi toiminnallista määrittelyä, se saattaa kieliä huolimattomasti tehdystä suunnittelutyöstä. (Määttä & Rautio 2010; Wikipedia 2011)

Projektin päättyessä tuotedokumentointi muutetaan ylläpitoa palvelevaan muotoon ylläpitodokumentiksi. Tekniseen dokumenttiin kootaan kaikki tiedot, joita ylläpidossa tarvitaan: toiminnallinen määrittely, tekninen määrittely sekä testaukseen ja tuotteen hallintaan liittyvät ohjeistukset. Ohjelmistoa korjattaessa ja päivitettäessä täytyy dokumentointi saattaa myös

ajan tasalle. Tämän vuoksi dokumentointi kannattaa muokata mahdollisimman helposti ylläpidettävään muotoon. Tätä tukemaan dokumenttimallit kannattaa standardisoida yhdenmukaisiksi yrityksen sisällä. (Määttä & Rautio 2010)

Tyypillinen ohjelmistoalan ongelma on, että määrittelydokumentteja ei jakseta kirjoittaa riittävän tarkasti. Ajatellaan, että dokumentin kirjoittamiseen kuluva aika käytettäisiin tehokkaammin ohjelmoimalla itse tuotetta. Tosiasia on kuitenkin, että asioita on helpompi muuttaa luonnollisella kielellä kirjoitetusta dokumentista kuin ohjelmakoodista ja tarkasti tehty dokumentointi helpottaa tulevaisuudessa ohjelmiston päivittämistä uudempaan versioon. (Määttä & Rautio 2010; Wikipedia 2011)

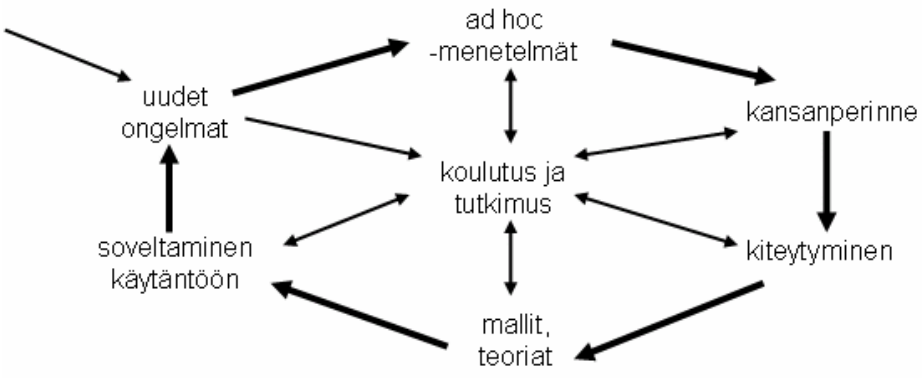
3 OHJELMISTOKEHITYKSEN SUUNNITTELUMENETELMÄT

Ohjelmistokehitys (engl. software engineering, usein suomennettu myös ohjelmistotuotanto) esiteltiin terminä ensimmäisen kerran 1960-luvun lopun ohjelmistokriisin aikaan. Kriisin taustalla oli se, etteivät entiset ohjelmistokehitysmenetelmät enää sopineet uusille, tehokkaammille tietokoneille, vaan kehitysprojektit olivat jopa vuosia myöhässä, ylittivät kustannuksensa ja tuloksena syntyneet ohjelmistot olivat hankalia ylläpitää. (Boström 2010)

Ensimmäiset ohjelmistokehitysmenetelmät alkoivat syntyä muista insinööritieteistä haettujen menetelmämallien pohjalta. Ohjelmistokehitystyön systemaattiset työskentelytavat ovat myös hyvin läheisesti yhteydessä muiden insinööritieteiden systemaattisiin toimintatapoihin. (Boström 2010)

Ohjelmistotuotannon kehittymisen vaiheita voidaan kuvata seuraavalla tavalla:

- Aluksi ongelman ratkaisuksi kelpaa mikä tahansa ongelman ratkaiseva tapa. Tällöin sovelletaan ns. ad hoc- menetelmiä.
- Löydetyistä ratkaisuista, jotka toimivat useammassa kuin yhdessä tapauksessa, syntyy kansanperinnettä.
- Kun kansanperinteenä välittyvä osaaminen kehittyy järjestelmällisemmäksi, siitä syntyy tutkimus- ja työskentelymenetelmiä.
- Ajan kuluessa nämä menetelmät kehittyvät riittävästi ja muodostavat malleja sekä teorioita uusille sovelluksille.
- Kun kehittyneitä malleja sovelletaan käytännössä, löydetään uusia ongelmia ja kokonaan uusia sovellusalueita, joille entiset ratkaisumallit eivät enää sovikaan. Näin on aloitettava alusta ja palattava ad hoc- menetelmiin. (Boström 2010)



Kuvio 5. Ohjelmistotuotannon kehittyminen (Boström 2010.)

Mallit ja teoriat sekä käytäntö yhdessä muodostavat ohjelmistokehityksessä käytettävän ohjelmistokehitysmenetelmän. Menetelmä terminä määritellään kokoelmaksi erilaisia teknisiä työvaiheita sekä ohjeita siitä, missä

järjestyksessä ja millä tavalla nämä työvaiheet tulee suorittaa tietyn tavoitteen saavuttamiseksi. Ohjelmiston kehitysmenetelmän valinta tapahtuu sovelluksen käyttötarkoituksen, laajuuden, tavoitellun laadun sekä käytettävissä olevan ajan ja taloudellisten resurssien mukaan. (Boström 2010)

Ohjelmistokehitysmenetelmät voidaan jakaa kahteen lähestymistapaan tuottamansa dokumentoinnin määrän perusteella: suunnitelmaohjautuviin kehitysmenetelmiin (plan-driven methods) ja ketteriin kehitysmenetelmiin (agile methods). (Määttä & Rautio 2010)

Suunnitelmaohjautuvissa kehitysmenetelmissä ohjelmiston kehitys alkaa tarpeiden määrittelystä ja päättyy ylläpitoon. Näissä menetelmissä tehtävät, merkkipaalut, määrittelyt sekä arkkitehtuurisuunnitelmat suunnitellaan ja dokumentoidaan mahdollisimman tarkasti. (Määttä & Rautio 2010)

Vastapainoksi raskaille suunnitelmaohjautuville kehitysmenetelmille kehitettiin joukko kevyempiä menetelmiä, jotka perustuvat dokumentoinnin sijaan asiakasyhteistyön tuloksena syntyvään toimivaan ohjelmistoon. Tähän malliin kuuluvia menetelmiä kutsutaan ketteriksi ohjelmistokehitysmenetelmiksi. (Määttä & Rautio 2010)

3.1 Ohjelmistokehityksen elinkaari

Ohjelmiston elinkaarella (engl. life cycle) tarkoitetaan aikaa, joka kuluu ohjelmiston kehittämisen aloittamisesta sen poistamiseen käytöstä. Elinkaarimalli taas on tapa, jolla ohjelmiston kehitystyö jaetaan vaiheisiin. Oikean elinkaarimallin valinnalla voidaan vaikuttaa ratkaisevasti ohjelmistoprojektin kehitykseen. Sillä voidaan sekä parantaa projektin kehittämisnopeutta, laatua ja hallintaa että minimoida turhaa työtä ja riskien ottamista. Väärällä elinkaarimallin valinnalla tai mallin valitsematta jättämisellä voidaan aiheuttaa paljon tarpeetonta työtä, joka osaltaan hidastaa projektia ja johtaa helposti turhautumiseen. (Määttä & Rautio 2010)

Tunnetuin ohjelmistojen elinkaarimalleista on vesiputousmalli, jossa suunnittelu- ja toteutusprosessi etenevät vaihe vaiheelta alaspäin kuten vesiputouksessa. Ohjelmistolle on kuitenkin tyypillistä *inkrementaalinen* kehitys ts. ohjelmisto kasvaa koko ajan kohti lopullista muotoaan. Näin ollen tuotantoprosessi on usein *iteratiivinen* eli suunnittelua ja toteutusta tehdään pienissä osissa ja prosessia toistetaan. Vesiputousmalli kuvaa huonosti ohjelmiston iteratiivisuutta, sillä mallissa pyritään suunnittelemaan koko tuote kerralla toteutuskuntoon. Tästä syystä vesiputousmallin rinnalle on kehitetty useita iteratiivisia prosessimalleja kuten spiraalimalli ja ketterät kehitysmenetelmät. (Määttä & Rautio 2010)

3.2 Suunnitelmaohjautuvat kehitysmenetelmät

Kaikki suunnitelmaohjautuvat ohjelmistokehitysmenetelmät voidaan nähdä kokoelmana lineaarisesti eteneviä vaiheita, jotka seuraavat toinen toistaan ohjelmiston toteutuksessa. Yleensä nämä vaiheet ovat kaikissa suunnitelmaohjautuvissa malleissa perusajatukseltaan samoja, mutta niiden

esiintymistiheys ja toteutustapa vaihtelevat. Vaiheet voidaan esitellä seuraavasti:

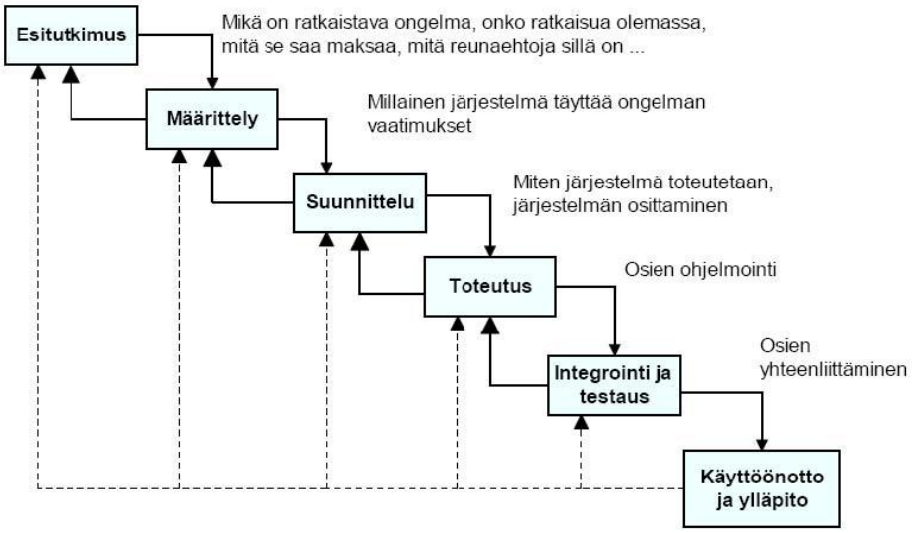
- Määrittely
- Suunnittelu
- Ohjelmointi (toteutus)
- Testaus
- Käyttöönotto ja ylläpito

Nämä vaiheet, joita voidaan kutsua myös ohjelmiston elinkaaren vaiheja-oksi, etenevät järjestyksessä ensimmäisestä viimeiseen ja lopputuloksena saadaan asiakkaan tarpeita ja määrittelyjä vastaava ohjelmisto. Yleensä kaikkiin edellä kuvattuihin vaiheisiin liittyy myös dokumentaatio, joka valmistuu kunkin vaiheen päättyessä ja ohjaa osaltaan ohjelmistokehitystä eteenpäin seuraavaan vaiheeseen. (Huttunen 2006)

3.2.1 Vesiputousmalli

Vesiputousmalli (eng. waterfall model) on erittäin käytetty, perinteinen suunnitelmaohjautuva ohjelmistokehitysmenetelmä, jonka Winston W. Royce esitteli nykyisessä muodossaan vuonna 1970. Vesiputousmalli toimii ideologialtaan pohjana useille muille malleille, jotka yleensä ovat saaneet alkunsa vesiputousmallin käytännön sovelluksista.

Vesiputousmallissa suunnittelu on jaettu osiin, joista jokainen tulee suorittaa loppuun ennen kuin siirrytään seuraavaan vaiheeseen. Tästä periaatteesta aiheutuvat vesiputousmallin hyvät ja huonot puolet. Käytännössä ohjelmistoa kehitettäessä on yleensä joissakin tapauksissa tarpeen palata takaisin edelliseen vaiheeseen ja muuttaa sen toteutusta kuten kuviossa 6 on kuvattu.



Kuvio 6. Vesiputousmalli (Määttä & Rautio 2010)

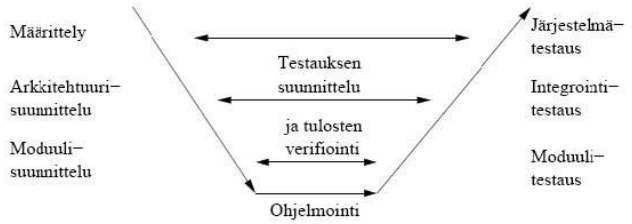
Vesiputousmallin *esitutkimusvaiheen* tehtävänä on asettaa yleiset vaatimukset toteutettavalle järjestelmälle. Näitä vaatimuksia kutsutaan usein myös asiakasvaatimuksiksi, sillä ne määrittelevät asiakkaan tarpeet toteutettavalle järjestelmälle, mutta eivät ota vielä kantaa, millainen ja miten toteutettu täyttää nämä tarpeet. Esitutkimuksen voidaan katsoa olevan myös osa määrittelyvaihetta, koska asiakastarpeita analysoidaan ja tarkennetaan yleensä käytännössä koko määrittelyvaiheen ajan. Esitutkimukseen liittyy usein myös alustava projektisuunnitelma. (Huttunen 2006)

Määrittelyvaiheessa kerätään ohjelmistolle asetetut toiminnalliset ja tekniset vaatimukset sekä ei-toiminnalliset vaatimukset ja rajoitukset. Ei-toiminnallisiksi vaatimuksiksi kutsutaan esimerkiksi suoritustehoon, vasteaikaan sekä käytettävyyteen liittyviä vaatimuksia (Huttunen 2006). Toiminnalliset vaatimukset taas kertovat, mitä järjestelmän odotetaan tekevän. Määrittelyn tuloksena syntynyttä dokumenttia kutsutaan vaatimusmäärittelyksi tai toiminnalliseksi määrittelyksi. (Määttä & Rautio 2010)

Suunnitteluvaiheessa (design) suunnitellaan, miten järjestelmä toteutetaan ja sen tarkoituksena on muuntaa asiakkaan tarpeisiin mukautuva toiminnallinen määrittely tekniseksi määrittelyksi (technical specification), joka kuvaa järjestelmän toteutuksen. Suunnitteluvaihe voidaan jakaa kahteen pienempään kokonaisuuteen: arkkitehtuurisuunnitteluun (architechtural design) ja moduulisuunnitteluun (module design). Arkkitehtuurisuunnittelussa järjestelmä pilkotaan pieniin, toisista riippumattomiin osiin eli moduuleihin. Moduulisuunnitteluvaiheessa suunnitellaan jokaisen moduulin sisäinen rakenne. (Määttä & Rautio 2010)

Toteutusvaiheessa (ohjelmointivaiheessa) kirjoitetaan varsinainen ohjelman lähdekoodi. Vaiheen tuloksena on varsinainen ohjelma sekä koodiin liittyvä dokumentaatio. (Huttunen 2006).

Testausvaiheessa (testing) on tarkoitus löytää ohjelmistosta virheitä. Ohjelmistojen testaus tapahtuu yleensä ns. V-mallin mukaisesti. V-mallissa testaus jaetaan kolmeen osaan: moduulitestaukseen, integrointitestaukseen ja järjestelmätestaukseen (kuvio 7). Moduulitestauksessa etsitään yksittäisten moduulien vikoja, integrointitestauksessa vikaa etsitään kaikkien moduulien yhteistoiminnoista sekä järjestelmätestauksessa koko järjestelmän toiminnoista ja suorituskyvystä. Järjestelmätestauksen suunnittelu tehdään alustavasti jo määrittelyvaiheessa ja testaus tehdään vertaamalla valmista järjestelmää sen määrittelyyn. (Määttä & Rautio 2010)



Kuvio 7. Ohjelmistojen testauksen V-malli (Määttä & Rautio.)

Käyttöönotossa valmis ohjelmisto toimitetaan asiakkaan käyttöön. *Ylläpidolla* tarkoitetaan valmiin järjestelmän käytössä havaittujen virheiden korjaamista, ohjelman laajentamista ja muuttamista. Ylläpitoon liittyy usein myös erilaisia ylläpitosopimuksia ja asiakaspalvelun organisointitehtäviä. (Huttunen 2006)

Vesiputousmallin keskeinen ominaispiirre, sen heikkous ja vahvuus, keskittyy monilta osin prosessin alkupuoleen. Mallin ideologiaan kuuluu, että varsinaiseen toteutustyöhön, ohjelmointiin halutaan käyttää niin vähän energiaa kuin mahdollista. Tähän pyritään mahdollisimman perusteellisella suunnittelulla: Asiakkaalta pyritään saamaan selville järjestelmän määrittelyt (vaatimusmäärittelydokumentti) täydellisinä ennen toteutustyön aloittamista. Todellisuudessa tähän tilanteeseen pääseminen on hyvin vaikeaa, ellei jopa mahdotonta. Siksi toteutuksen aikanakin on tarpeen varmistaa asiakkaalta aika ajoin toteutuksen vastaavuutta todellisiin tarpeisiin. (Huttunen 2006)

Vesiputousmalli soveltuu hyvin sellaisiin tilanteisiin, joissa on hyvin tarkasti tiedossa, mitä toteutettavalta lopputuotteelta halutaan ja mihin sitä aiotaan käyttää. Todellisuudessa asetettujen määrittelyiden muuttumisriski on hyvin suuri, jonka vuoksi vesiputousmallin käyttäminen ideaalisena, ilman takaisinkytkentöjä edellisiin vaiheisiin, on käytännössä mahdollista erittäin harvoin. (Huttunen 2006)

3.2.2 Prototyyppimalli

Prototyyppimallilla voidaan tarkoittaa melkein mitä tahansa työskentelymallia, jossa jotain tuotteen piirrettä kokeillaan ennen varsinaisen tuotteen rakentamista. Prototyyppimallit soveltuvat erityisesti uuden teknisen ratkaisun vaatiman kokeilun tekemiseen tai etsimään epäselviä asiakasvaatimuksia. (Määttä & Rautio 2010)

Kehitettävien ohjelmistojen prototyypit pyritään toteuttamaan mahdollisimman edullisesti ja nopeasti. Tämä voi tarkoittaa prototyypin eroamista varsinaisesta järjestelmästä kahdella tavalla esimerkiksi toteutusvälineiden (ohjelmointikieli) ja toteutettavien toimintojen osalta. (Huttunen 2006)

Toteutusvälineiden osalta on kyse yleensä varsinaisen ohjelmointikielen sijasta käytettävästä jostakin toisesta, prototyypin kannalta tehokkaammasta ohjelmointikielestä (Huttunen 2006). Erityisesti ohjelmistotuotteissa prototyyppien käyttäminen on liittynyt käyttöliittymien suunnitteluun, ja tässä prototyypit ovatkin osoittautuneet erityisen käyttökelpoisiksi. Prototyyppien avulla voidaan nopeasti myös kehittää uusia kehitysversioita kunnes asiakas on lopputulokseen tyytyväinen. (Boström 2010)

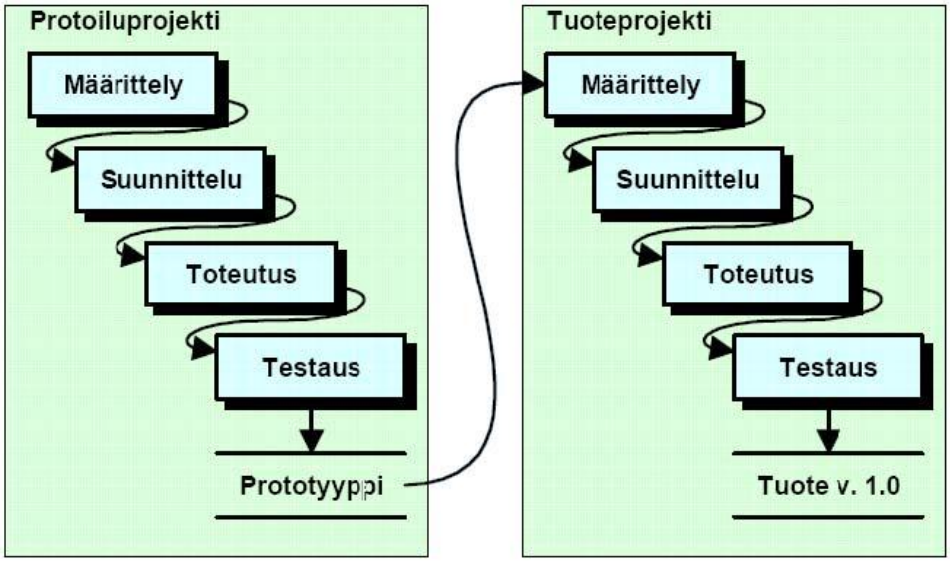
Toteutettavien toimintojen osalta eroavaisuudet todelliseen järjestelmään voivat koskea esimerkiksi tietojen hakua ja tallennusta. Prototyypissä ei välttämättä tarvitse olla näille toiminnoille mitään sovelluslogiikkaa, etenkään silloin mikäli mallinnus koskee ainoastaan käyttöliittymäsuunnittelua. Jos sovelluslogiikkaa kuitenkin toteutetaan jo prototyyppiin, voi sen

ominaisuudet ja toiminnallisuus olla merkittävästi tuotantoversiota heikommat. (Huttunen 2006)

Prototyypin toteuttamisen jälkeen on tarjolla kaksi päävaihtoehtoa, joiden mukaan ohjelmistokehitysprojekti voi edetä (Huttunen 2006):

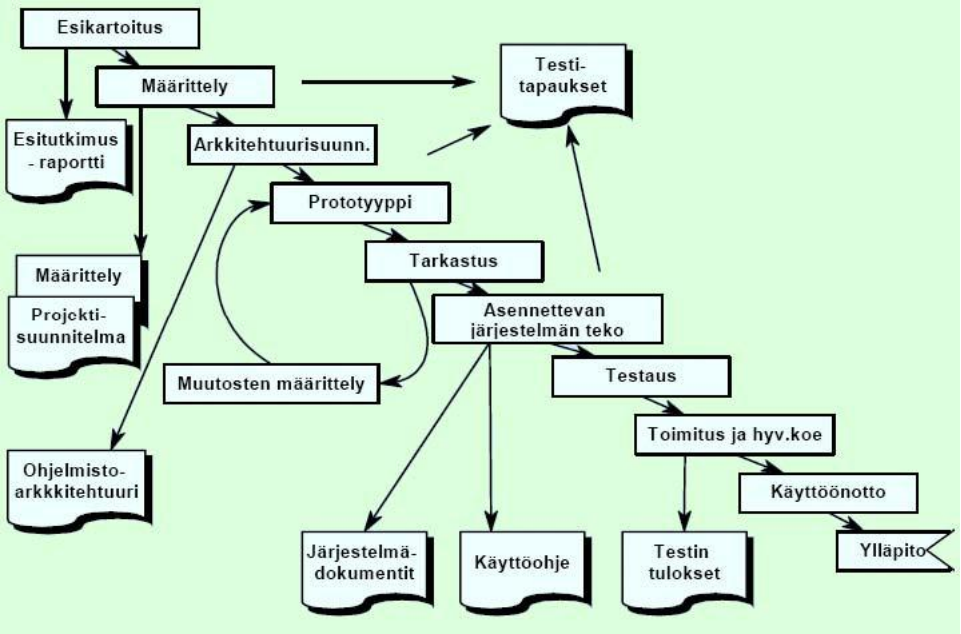
a) Valmistuneen prototyypin perusteella määritellään toteutettava järjestelmä, joka toteutetaan alusta asti uudelleen ja prototyyppi hylätään.
b) Prototyyppi kehitetään valmiiksi järjestelmäksi.

Kuviossa 8 esitetään esimerkki vaihtoehdosta a. Sekä prototyypin että varsinaisen järjestelmän suunnittelu ja toteutus etenee samaan tapaan. Tuotettavan järjestelmän määrittelyssä on kuitenkin käytettävissä jo prototyypin avulla selvitettyä informaatiota, jolloin lähtökohta on erilainen. (Huttunen 2006)



Kuvio 8. Prototyyppimalli (Määttä & Rautio 2010)

Kuviossa 9 esitetään vaihtoehto b. Prototyyppimalli sisältää jo kaikki tärkeimmät toiminnot lukuun ottamatta virhetarkastuksia, opastuksia ja tehokkaasti toimivaa tietokantaa. Ennen järjestelmän lopullista toteutusta huomioidaan asiakkaan palaute käyttöliittymän suhteen sekä varmistetaan järjestelmän sisältävän kaikki tarpeelliset toiminnot. (Määttä & Rautio 2010)



Kuvio 9. Prototyyppi kehitetään valmiiksi järjestelmäksi. (Määttä & Rautio 2010)

Prototyyppimallia on hyvä käyttää tilanteissa, joissa asiakkaan antamat määrittelyt ovat epäselviä tai sellaisissa järjestelmissä, joissa käyttöliittymällä on suuri rooli. Ongelmana on kuitenkin, että asiakas saattaa luulla järjestelmää lähes valmiiksi prototyypin ulkoasun perusteella, vaikka valtaosa työstä olisi vielä tekemättä. Tästä syystä asiakkaalle onkin syytä korostaa, että prototyyppi ei ole varsinainen järjestelmä, eikä prototyypin valmistuminen tarkoita koko järjestelmän valmistumista. Prototyyppien toteutuksen on myös oltava suunnitelmallista, joten jonkin ohjelmistokehitysmallin käyttäminen myös prototyypin kehityksessä on järkevää. (Huttunen 2006)

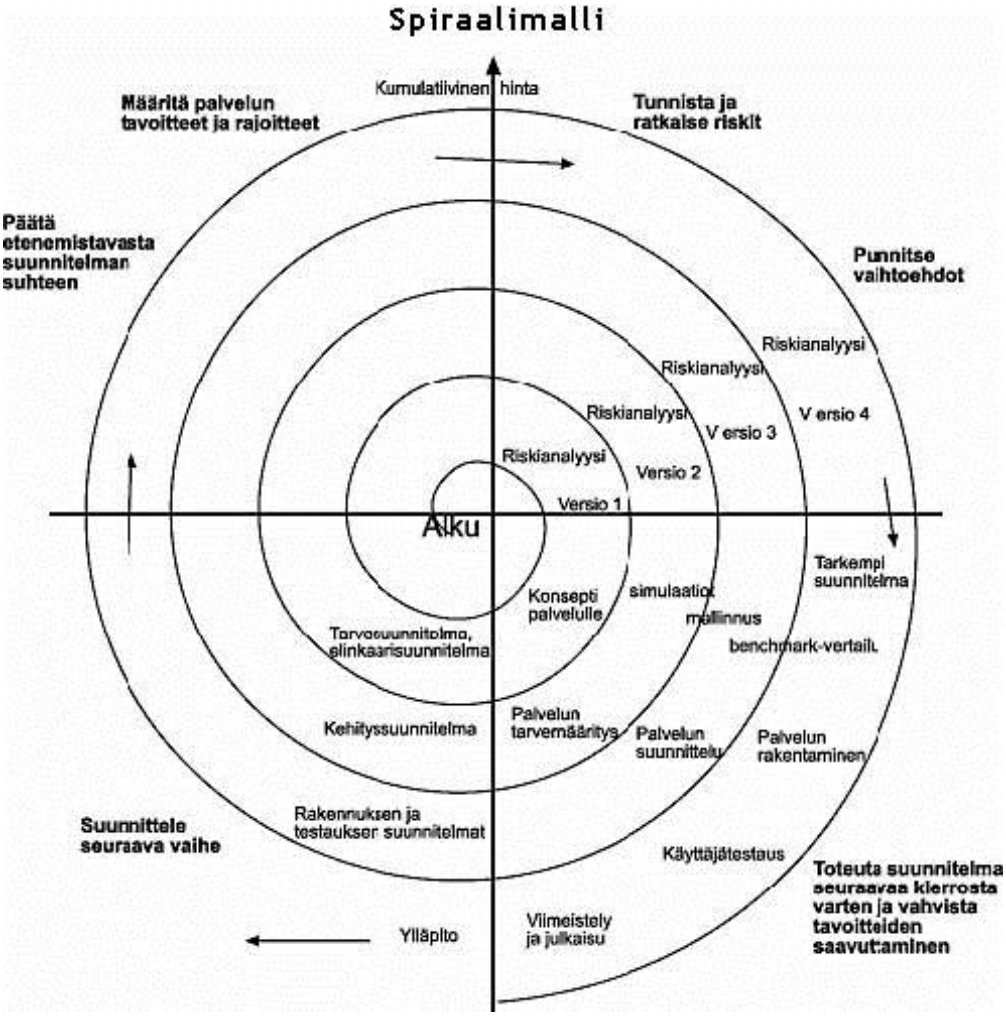
3.2.3 Spiraalimalli

Spiraalimalli perustuu voimakkaasti vesiputousmalliin, mutta yhdistää myös muita ohjelmistosuunnittelun menetelmiä kuten evoluutio- (evolutionary development) ja ohjelmoi-korjaa (code-fix) –menetelmiä. Spiraalimallia eivät kuitenkaan ohjaa dokumentit vaan riskit ja tästä syystä mallia voidaankin kutsua myös riskiohjatuksi prosessiksi. (Huttunen 2006.) Riskeillä tarkoitetaan tässä yhteydessä mm. huonosti määriteltyjä ohjelmiston vaatimuksia ja arkkitehtuuria tai ohjelmistoon liittyviä teknisiä ongelmia. Spiraalimallille on tyypillistä iteratiivinen eteneminen sykleissä, riskien jatkuva analysoiminen sekä prosessin uudelleenohjaaminen riskianalyysin tuotosten mukaan. (Määttä & Rautio 2010)

Spiraalimallissa edetään siis sykleissä ja jokainen sykli sisältää neljä eri vaihetta, jotka toistuvat kaikissa sykleissä (Huttunen 2006). Ensimmäinen vaihe on suunnittelu, jossa määritellään rakennettavalle ohjelmistolle tavoitteet, vaihtoehdot ja rajoitukset. Toisessa vaiheessa tehdään riskianalyysi, jossa arvioidaan eri vaihtoehtoihin liittyviä ongelmia. Kolmannessa vaiheessa ohjelmisto menee tuotantoon, josta syntyy uusi ohjelmaversio. Neljännessä vaiheessa suoritetaan asiakkaan kanssa arviointi, jonka perus-

teella tehdään ohjelmiston jatkamis- tai lopettamispäätös. (Määttä & Rautio 2010)

Kuviossa 10 spiraalimalli lähtee etenemään kuvan keskeltä päätyen kehän ulkoreunaan. Etäisyys keskipisteestä kuvaa projektin kokonaiskustannuksia ja sen kokonaisvaihetta. Mitä kauempana keskipisteestä ollaan, sitä valmiimpi on lopputulos. (Huttunen 2006)



Kuvio 10. Ohjelmistokehityksen spiraalimalli. (Huttunen 2006)

Spiraalimallin käyttöä voidaan jatkaa koko ohjelmiston elinkaaren ajan. Tällöin voidaan ajatella, että ensimmäinen kierros kuvion 10 spiraalilla tarkoittaa vain alkuperäisen ohjelmiston suunnittelua ja toteutusta, ja seuraavat kierrokset ovat ohjelmiston kehitystyötä ja ylläpitoa. (Huttunen 2006.) Spiraalimalli ei myöskään edellytä jotain tiettyä menetelmää tuotantovaiheessa, vaan sallii esimerkiksi vesiputousmallin tai prototyyppilähestymistavan soveltamisen. Prosessi on mahdollista keskeyttää, jos jossakin vaiheessa havaitaan riskien kasvaneen liian suuriksi, mutta tavoitteena on kuitenkin pienentää analyysin avulla riskejä joka iteraatiolla. (Määttä & Rautio 2010)

Spiraalimallista on kuitenkin suhteellisen vähän käytännön kokemuksia. Mallin suurimpana ongelmana pidetään sen sopeutumista talousnäkökoh-

tiin, sillä iteratiivinen prosessi on aikaavievä ja kehityksen eteneminen sykleissä saattaa hankaloittaa sopimusmäärittelyitä asiakkaan kanssa. Lisäksi mallin soveltaminen vaatii riskianalyysin hallitsemista. (Huttunen 2006; Määttä & Rautio 2010)

3.3 Ketterä ohjelmistokehitys

1990-luvun lopulla useat ohjelmistokehitysmenetelmät alkoivat saada ohjelmistotekniikan alalla huomiota. Jokainen menetelmä yhdisti vanhoja sekä uusia ideoita, kuten myös muunneltuja vanhoja ideoita. Jokainen menetelmä painotti kuitenkin läheistä yhteistyötä ohjelmoijien ja liiketoiminnan ammattilaisten kesken, kasvokkain tapahtuvaa kommunikointia, säännöllistä uuden vastineen tuottamista asiakkaan investoinneille sekä ohjelmointikäytäntöjä ja tiimejä, jotka kykenevät vastaamaan muuttuviin vaatimuksiin. (Huttunen 2006)

Vuonna 2001 USA:ssa monet näiden eri menetelmien kehittäjät ja käyttäjät kokoontuivat miettimään, mitä yhteistä nämä menetelmät pitivät sisällään. Kokouksen seurauksena otettiin käyttöön termi ketterä (agile), joka kuvaa näille menetelmille yhteisiä piirteitä. Tässä samassa tapahtumassa julkaistiin myös ketterän sovelluskehityksen manifesti (Agile Software Development Manifesto), joka määrittelee yleiset arvot ja periaatteet ketterälle sovelluskehitykselle. (Huttunen 2006; Määttä & Rautio 2010)

Ketterää ohjelmointia perustellaan muun muassa sillä, että liiketoiminnassa vaaditaan yhä enemmän ketteryyttä, etenkin kustannusten ja aikataulujen suhteen. Myös järjestelmien teknologia muuttuu jatkuvasti. Tämän vuoksi kiinnostus nopeampia ja joustavia ohjelmistokehitysmenetelmiä kohtaan on jatkuvasti kasvanut, sillä näiden menetelmien avulla sovelluskehittäjät voivat kasvattaa tuottavuutta säilyttäen samalla ohjelmistojen laadun ja muokattavuuden korkeana. (Huttunen 2006; Määttä & Rautio 2010)

Ketterässä kehityksessä suunnittelu perustuu raskaan dokumentoinnin sijasta ihmisten välisellä keskustelulla tapahtuvaan tiedonhankintaan. Projektin aikaisten dokumenttien ainoa tehtävä on vähentää epävarmuustekijöitä ja tehostaa kommunikaatiota. Muutoin ne ovat turhia. (Ketterät käytännöt 2009a; Määttä & Rautio 2010)

Ketterässä ohjelmistokehityksessä ei pyritä määrittelemään kaikkia ohjelmiston yksityiskohtia kerralla, vaan suunnittelua ja toteutusta tehdään pienemmissä osissa ja prosessia toistetaan. Jokainen iteraatio luo jotakin uutta tietoa tai yksityiskohtia, jolloin ohjelmisto kehittyy koko ajan kasvaen kohti lopullista muotoaan. Käytännön toteutuksessa ohjelmistoon tehdään koko ajan pieniä muutoksia ja uusia versioita julkaistaan nopeassa tahdissa. Tämä kysyy korkeaa teknistä osaamista. Yleisesti ketterät menetelmät ovat hyvin suoraviivaisia ja mukautuvia, helposti opittavia, hyvin ohjeistettuja sekä mahdollistavat viime hetken muutokset. (Määttä & Rautio 2010)

Yleisesti tunnetuimpana pidetty ketterien menetelmien ohjelmistokehitysmenetelmä on Extreme Programming, jonka määritteli vuonna 1999 amerikkalainen ohjelmistosuunnittelija Kent Beck. XP-menetelmää on pidetty monissa yhteyksissä ketterien ohjelmistokehitysmenetelmien uranuurtajana ja ketterien menetelmien kehityksen alkuna (Huttunen 2006). Muita keskeisiä ketteriä ohjelmistokehitysmenetelmiä ovat esimerkiksi Crystal Methods ja Scrum, joita esitellään tässä työssä tuonnempana.

3.3.1 Yleiset perusarvot ja periaatteet

Ketterässä kehityksessä lähdetään siitä, että ei ole olemassa yhtä oikeaa tapaa saavuttaa haluttu lopputulos. Harvat ketterät menetelmät sisältävät tarkoin määriteltyjä toimintamalleja eri tilanteisiin, mutta kaikki menetelmät noudattavat kuitenkin ketterän kehityksen manifestissa (Agile Alliance 2011a) julkaistuja perusarvoja ja periaatteita.

Manifestin mukaan ketterän kehityksen neljä yleistä arvoa ovat:

- **Yksilöt ja vuorovaikutus** *ovat tärkeämpiä kuin prosessit ja työkalut*
- **Toimiva ohjelmisto** *on tärkeämpää kuin kattava dokumentointi*
- **Asiakasyhteistyö** *on tärkeämpää kuin sopimusneuvottelut*
- **Muutokseen reagoiminen** *on tärkeämpää kuin suunnitelman noudattaminen*

Ketterissä menetelmissä arvostetaan enemmän vasemmanpuoleisia (lihavoidut) arvoja kuin oikeanpuoleisia (kursivoidut). Tämä ei kuitenkaan tarkoita, että prosessit, työkalut, dokumentoinnit sopimukset sekä suunnitelman eivät olisi tärkeitä. Oikeanpuoleisten arvojen voidaan ajatella olevan rinnastuksia alan yleiseen ajattelutapaan, kun taas ketterät menetelmät suosivan enemmän asiakkaan vaatimuksen mukaisena syntyviä lopputuloksia. (Agile Alliance 2011a.)

Perusarvojen lisäksi manifestissa kuvataan 12 periaatetta (Agile Alliance 2011b), joita ketterä kehitys noudattaa:

1. Tärkeintä on täyttää asiakkaan tarpeet jatkuvilla ja riittävän aikaisilla ohjelmistotoimituksilla.

2. Vaatimusten muuttumiset ovat tervetulleita ja ne hyväksytään jopa myöhäisessä kehitysvaiheessa. Ketterät menetelmät valjastavat muutoksen asiakkaan kilpailueduksi.

3. Toimivia ohjelmaversioita toimitetaan säännöllisesti mielellään lyhyellä aikavälillä, muutamasta viikosta muutamaan kuukauteen.

4. Liiketoiminnan ammattilaisten ja kehittäjien täytyy työskennellä yhdessä päivittäin koko projektin ajan.

5. Projektit rakennetaan motivoituneiden ihmisten ympärille. Annetaan heille ympäristö ja tuetaan heidän tarpeitaan, sekä luotetaan siihen, että he saavat työn tehtyä.

6. Kaikkein tehokkain tapa välittää tietoa kehitystiimille ja –tiimin sisällä on kasvokkain tapahtuva keskustelu.

7. Toimiva ohjelmisto on ensisijainen työn edistymisen mittari.

8. Ketterät menetelmät suosivat kestävää kehitystä. Rahoittajien, kehittäjien ja käyttäjien tulisi pitää jatkuvasti yllä tasaista työtahtia.

9. Jatkuva huomion kiinnittäminen tekniseen osaamiseen ja hyvään suunnitteluun lisää ketteryyttä.

10. Yksinkertaisuus – taito maksimoida tekemätön työ – on olennaista.

11. Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat nousevat itseorganisoituvista tiimeistä.

12. Tiimi arvioi tasaisin väliajoin, kuinka tulla entistä tehokkaammaksi ja muokkaa toimintaansa sen mukaisesti.

Huttusen mukaan (2006) näitä edellä kuvattuja arvoja ja periaatteita voidaan käyttää seuraavanlaisina ketterän kehityksen tunnusmerkkeinä: Ketterä ohjelmistokehitys on *inkrementaalista* (pienet versiojulkaisut tiheään tahtiin), *yhteistyössä tapahtuvaa* (asiakas ja kehittäjä toimivat jatkuvasti yhdessä ja yhteydenpito on tiivistä), *suoraviivaista* (menetelmä itsessään on helppo oppia ja se on hyvin dokumentoitu) ja *sopeutuvaa* (on mahdollista tehdä viime hetken muutoksia).

3.3.2 Extreme Programming

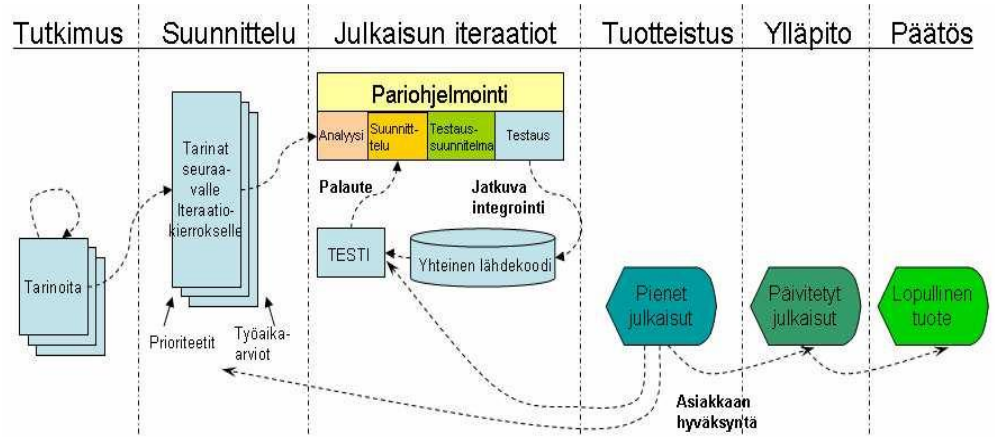
Nimensä mukaisesti Extreme Programming (XP) on hyvin ohjelmointikeskeinen menetelmä. Muista ketteristä menetelmistä poiketen, XP tarjoaa joukon käytäntöjä, joita tulee noudattaa yhdessä, sillä käytännöt on rakennettu tasapainottamaan toisiaa. Käytäntöjä voidaan ajatella palapelin tavoin: yksin ja erillään palasissa ei ole järkeä, mutta yhdessä ne muodostavat selkeän kokonaisuuden. Vaikka XP:n pääpaino onkin ohjelmistoteknisessä toteutuksessa, siinä on kuitenkin mukana myös jonkin verran projektihallintaan liittyviä käytäntöjä, kuten suunnittelupeli. (Extreme Programming 2009a; Ketterät käytännöt 2009b)

Extreme Programming –menetelmä rakentuu ajatukselle kokonaisesta yhteisöstä, tiimistä, jossa niin asiakkaat kuin ohjelmistokehittäjätkin ovat kaikki tasa-arvoisia kumppaneita keskenään. Tällainen yksinkertainen, mutta tehokas ympäristö tarjoaa tiimeille mahdollisuuden toimia erittäin tuottavasti ja itseohjautuvasti. Lisäksi XP –menetelmä pyrkii parantamaan ohjelmistoprojektia seuraavalla viidellä keskeisellä tavalla: *kommunikoinnilla, yksinkertaisuudella, palautteella, kunnioituksella ja rohkaisulla*. XP-ohjelmoijat kommunikoivat jatkuvasti asiakkaidensa ja toisten ohjelmointikollegojensa kanssa. Ohjelmisto suunnitellaan selkeäksi ja yksinkertaiseksi ja palautetta tehdystä työstä saadaan heti ensimmäisestä päivästä lähtien testauksen avulla. Ohjelmisto toimitetaan asiakkaalle mahdollisimman

nopeasti ja mikäli muutoksille on tarvetta, ne toteutetaan. Jokainen onnistuminen syventää osaltaan tiimin ryhmähenkeä ja parantaa työmotivaatiota, mikä osaltaan rohkaisee tiimiä vastaamaan alati muuttuvien vaatimusten ja teknologian luomiin haasteisiin. (Extreme Programming 2009a; Huttunen 2006; Määttä & Rautio 2010)

Extreme Programming –elinkaarimalli koostuu kuudesta päävaiheesta (Huttunen 2006), jotka etenevät numerojärjestyksessä ensimmäisestä viimeiseen, mutta mahdollistavat myös paluun takaisin edellisiin vaiheisiin (kuvio 11):

1. Tutkimus (Exploration phase)
2. Suunnittelu (Planning phase)
3. Julkaisun iteraatiot (Iterations to release phase)
4. Tuotteistus (Productionizing phase)
5. Ylläpito (Maintenance phase)
6. Päätös (Death phase)



Kuvio 11. XP –prosessin elinkaari (Huttunen 2006)

Tutkimusvaiheessa asiakkaat kirjoittavat tarinakorteille (story card) toivomuksiaan ohjelmiston ensimmäiseen julkaisuun sisällytettävistä ominaisuuksista. Jokainen tarinakortti kuvaa ainoastaan yhtä järjestelmän ominaisuutta kerrallaan. Samalla tutkimusvaiheen aikana ohjelmoijat tutustuvat toteutuksessa käytettävään tekniikkaan, työvälineisiin ja käytäntöihin. Vaiheen kesto vaihtelee muutamista viikoista muutamaan kuukauteen. (Huttunen 2006; Määttä & Rautio 2010)

Suunnitteluvaiheessa päätetään ensimmäisen julkaisun sisällöstä määrittelemällä asiakkaan antamien tarinoiden (ominaisuuksien) tärkeysjärjestys. Ohjelmoijat arvioivat toteuttamiseen tarvittavan työmäärän ja merkitsevät sen kortteihin, jonka jälkeen ominaisuuksien toteutusaikataulu voidaan sopia tarkemmin. Suunnitteluvaihe kestää muutamia päiviä. (Huttunen 2006; Määttä & Rautio 2010)

Julkaisun iteraatiovaiheessa aikataulu pilkotaan pienempiin iteraatioihin, joista jokaisen toteuttamiseen kuluu aikaa yhdestä neljään viikkoa. Ensimmäinen iteraatiokierros luo järjestelmän arkkitehtuurin ja seuraavat rakentavat hiljalleen ohjelmiston muita ominaisuuksia. Jokaisen iteraa-

tiokierroksen päätteeksi asiakas laatii hyväksymistestit kullekin ominaisuudelle. Mikäli testatut ominaisuudet täyttävät annetut kriteerit, ne luokitellaan valmiiksi ja seuraavalle kierrokselle voidaan valita uusia ominaisuuksia toteutettavaksi. Iteraatioiden edetessä asiakas voi milloin tahansa lisätä uusia tarinakortteja kierrokselle, muuttaa niiden tärkeysjärjestystä, pilkkoa yhden kortin useampaan osaan tai poistaa kortteja. Asiakas kuitenkin aina päättää seuraavaan iteraatioon otettavat toiminnot niiden tärkeyden ja hyväksymistestien perusteella. Viimeisen iteraatiokierroksen lopussa järjestelmä on valmis tuotantoon. (Extreme Programming 2009b; Huttunen 2006; Määttä & Rautio 2010)

Tuotteistusvaiheessa järjestelmälle tehdään testausta esimerkiksi suorituskykyyn liittyvissä asioissa. Tämä vaihe sisältää myös useita iteraatiokierroksia. (Huttunen 2006)

Ylläpitovaiheeseen siirrytään, kun tuotteen ensimmäinen versio on toimitettu asiakkaalle. Käytössä huomattuja virheitä korjataan ja ohjelmaversioita päivitetään toimivammiksi käyttäjien toiveiden mukaisesti. (Määttä & Rautio 2010)

Päätösvaiheessa käyttäjillä ei ole enää uusia vaatimuksia järjestelmää kohtaan. Järjestelmästä kirjoitetaan dokumentaatio ja se todetaan valmiiksi. Jatkuvasti kehittyvän järjestelmän tapauksessa tämä voi tarkoittaa myös järjestelmän sulkemista ja lopettamista tilanteessa, jossa se ei enää täytä asiakkaan tarpeita tai sitä ei muuten haluta enää kehittää. (Huttunen 2006)

Samoin kuin ketterille menetelmille yleisesti, on XP:llekin määritelty sen keskeiset kaksitoista käytäntöä (Huttunen 2006; Määttä & Rautio 2010):

- *Suunnittelupeli (Planning game)* on tiivistä yhteistyötä asiakkaan ja ohjelmoijien välillä. Asiakas kuvaa tarinakorteilla ohjelmistoon tarvittavat ominaisuudet ja ohjelmoijat arvioivat tarvitun työmäärän, jonka jälkeen asiakas arvioi vielä ominaisuuksien kiireellisyyden ja laajuuden.

- *Pienet julkaisut (Small releases)* tarkoittavat uusien ohjelmistoversioiden toimittamista asiakkaan käyttöön tiheään tahtiin – jopa päivittäin tai ainakin kuukausittain, koska kaikkia ongelmia ei tarvitse ratkaista kerralla.

- *Järjestelmävertauskuva (System metaphor)* tarjoaa kokonaiskuvan järjestelmästä ja sen toiminnasta, jolloin kaikkien osapuolien on helpompi ymmärtää projektin tavoitteet.

- *Yksinkertainen rakenne (Simple design)* vaatii ohjelmoijia toteuttamaan vain tarpeelliset asiat mahdollisimman yksinkertaisella tavalla. Ylimääräinen monimutkaisuus ja koodi tulee poistaa sekä tulevaisuudessa mahdollisesti tarvittaviin ominaisuuksiin ei tarvitse varautua nyt.

- *Testit ensin (Tests first)*-toimintatapa tarkoittaa sekä kehittäjien määrittelemien yksikkötestien että asiakkaan määrittelemien toiminnallisten testien tekemistä ennen toiminnon varsinaista toteutusta. Kun toiminto läpäisee testin, se täyttää tarkoituksensa ja on valmis.

- *Refaktoroinnilla (Refactoring) eli rakenteen parantamisella* koodia muokataan yksinkertaisemmaksi ja helpommin ylläpidettäväksi muuttamatta kuitenkaan sen toiminnallisuutta.

- *Pariohjelmointi (Pair programming)* on työtapa, jossa kaksi ohjelmoijaa istuu yhdessä saman tietokoneen ääressä toisen kirjoittaessa koodia ja toisen seuratessa, miettiessä vaihtoehtoisia ratkaisutapoja ja valvoessa koodin laatua.

- *Jatkuvalla integroinnilla (Continuous integration)* varmistetaan ohjelmiston eri komponenttien yhteensopivuus. Järjestelmän on läpäistävä kaikki aiemmin luodut testit missä tahansa ohjelmiston kehitysvaiheessa ennen kuin uudet muutokset ja lisäykset voidaan hyväksyä osaksi järjestelmää.

- *Koodin yhteisomistus (Collective ownership)* valtuuttaa kenet tahansa ohjelmoijista korjaamaan tai muutamaan tarvittavia kohtia ohjelmistosta.

- O*hjelmointikäytäntöjä (Code conventions)* tulee määritellä, jotta kaikkien tiimiläisten tuottama koodi on samanäköistä, jolloin sitä on helppo lukea ja ylläpitää.

- *Asiakkaan läsnäolo (On-site customer)* kuvaa sitä, että asiakkaan edustajan on oltava ohjelmiston toteuttajien käytössä kysymyksiä ja tarkennuksia varten.

- *40-tuntiset työviikot (40-hour weeks)* on listattu myös XP:n kantaviin perusajatuksiin. Jatkuvaa ylityötä ei sallita, jotta voidaan varmistaa riittävä virkistäytyminen ja vapaa-aika ohjelmoijille. Samaa järkevää työtahtia tarkoittavaa ajatusta kutsutaan toisissa lähteissä myös *säilytettävissä olevaksi tahdiksi (Sustainable pace).*

Extreme Programming on osoittautunut varsin onnistuneeksi ohjelmointitavaksi erikokoisissa yrityksissä, koska se korostaa asiakastyytyväisyyttä tuottamalla näkyviä tuloksia heti ohjelmiston kehityksen alusta lähtien. Lisäksi XP vastaa muuttuviin asiakasvaatimuksiin hyvin vielä myöhemmässäkin elinkaaren vaiheissa. (Määttä & Rautio 2010)

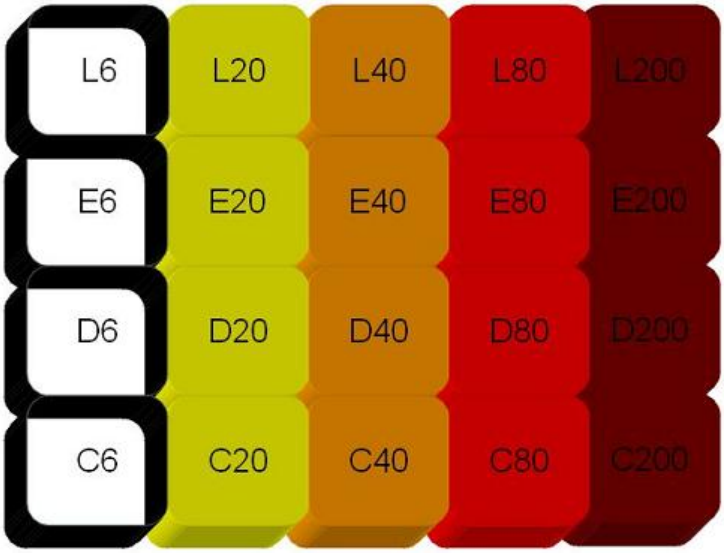
3.3.3 Crystal-metodologiaperhe

Crystal on Alistair Cockburnin ja Jim Highsmithin kehittämä ketterien kehitysmenetelmien perhe. Se on kokoelma erilaisia menetelmiä, joista voidaan valita sopivin ja tarkoituksenmukaisin menetelmä hyvin erilaisiin

projekteihin. Menetelmän valintavaiheessa projekteja arvioidaan kahdella akselilla: kehitystiimin koko ja projektin kriittisyys. Projektiin osallistuvien henkilöiden lukumäärä vaikuttaa siihen, millaisia käytäntöjä, kuten kommunikaatiomenetelmiä, on mielekästä noudattaa. Projektin kriittisyys taas voidaan luokitella erityyppisiin ryhmiin sen mukaan, millaista vahinkoa projekti saa aikaan epäonnistuessaan. Kriittisyysryhmät ovat (Huttunen 2006, Ketterät käytännöt 2009c):

- C = Mukavuuden heikkeneminen (Comfort): virhe aiheuttaa käsityötä, kun automaatio ei toimi.
- D = Kohtuullinen rahan menetys (Discretionary money): virhe aiheuttaa taloudellisia menetyksiä, jotka ovat kuitenkin korvattavissa.
- E = Merkittävä rahan menetys (Essential money): virheet ovat kohtuuttomia ja voivat aiheuttaa korvaamattomia taloudellsia menetyksiä.
- L = Ihmishengen menetys (Life): virhe voi vaarantaa ihmishenkiä.

Sopivan kehitysmenetelmän valinta tapahtuu yhdistämällä kriittisyyskoodi alla esitetyn matriisin (kuvio 12) pystyakselilta ja ohjelmiston kehitysryhmän koko vaaka-akselilta. Esimerkiksi C10 tarkoittaisi tilannetta, jossa kehitystyössä työskentelee 10 henkilöä ja tuotettavan ohjelmiston virheellinen toiminta voi pahimmillaan aiheuttaa käsityötä. (Huttunen 2006)



Kuvio 12. Crystal-metodologiamatriisi. (Ketterät käytännöt 2009c)

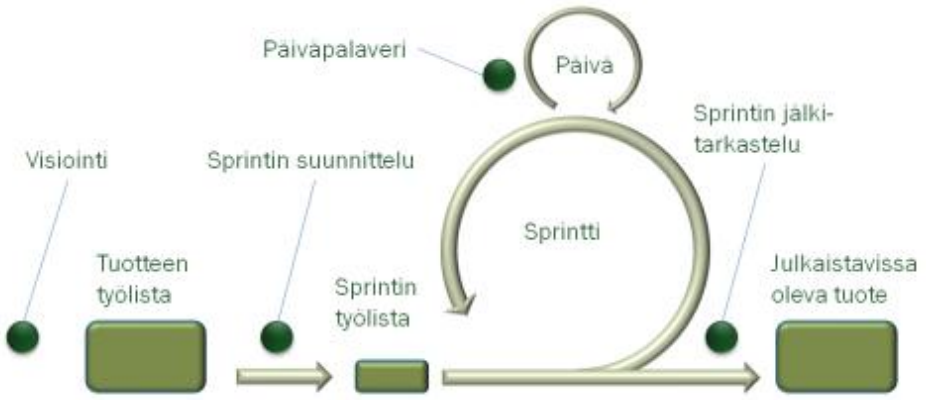
Crystal-perheen menetelmät on nimetty eri värisävyillä kehitysryhmän koon mukaan. Kevyin ja parhaiten dokumentoitu menetelmä on Crystal Clear (kirkas), jota seuraa keltainen (yellow), oranssi (orange), punainen (red) ja viininpunainen (maroon). Kirkas on suunniteltu 1-6:n, keltainen alle 20:n, oranssi 20-40:n, punainen 40-80:n ja viininpunainen luonnollisesti yli 80:n hengen projekteille. (Huttunen 2006, Ketterät käytännöt 2009c)

Crystal-metodologioiden ydinelementit ovat ihmis- ja kommunikaatiokeskeisyys, menetelmän sovitus tarpeen mukaan, inkrementaalinen kehitys ja lyhyt kehityssykli sekä arviointipalaverit ennen ja jälkeen sykliä. On kuitenkin tärkeää huomata, että Crystal-metodologiat eivät ota kantaa siihen, miten varsinainen sovelluskehitystyö konkreettisesti tulisi tehdä. Sovelluskehitystä voidaan hyvin siis toteuttaa esimerkiksi XP-menetelmällä tai vaikka suunnitelmaohjautuvilla sovelluskehitysmenetelmillä. (Huttunen 2006)

3.3.4 Scrum

Jeff Sutherlandin 1990-luvun alkupuolella kehittämä Scrum-menetelmä tarjoaa sovelluskehitykseen mallin, jonka mukaan projektia ohjataan. Scrum keskittyy ennen muuta projektin vaiheistamiseen ja jatkuvaan kontrolliin projektin etenemisestä. Kuten muidenkin ketterien menetelmien, myös Scrumin kantaviin perusperiaatteisiin luetaan kommunikaation maksimoiminen, muutoksiin sopeutuminen, toteutustyön inkrementaalisuus ja runsas testaaminen. (Huttunen 2006; Ketterät käytännöt 2009d)

Vesiputousmallin mukaisissa projekteissa on yleensä ainakin suunnittelija, ohjelmoija, testaaja ja projektipäällikkö. Scrum-projektissa rooleja on vain kolme: Tuotteen omistaja (product owner), Scrum-mestari (ScrumMaster) ja tiimi. Tuotteen omistaja on henkilö, joka viime kädessä vastaa tuotteen ominaisuuksista, toisin sanoen *omistaa* tuotteen. Yhdessä Scrum-tiimin kanssa hän luo aikataulun ominaisuuksien toteuttamiselle. Scrum-mestarin tehtävänä on huolehtia siitä, että tiimi voi tehdä työtään optimaalisella tavalla, ratkoa tiimin työtä hidastavia ongelmia sekä johtaa päivittäiset sprintti-palaverit. Tiimiin kuuluvat kaikki henkilöt, jotka ovat projektia tekemässä. Tiimiin sisältä ei erikseen nimitetä ohjelmoijia tai testaajia, vaan tiimiin kasataan henkilöitä, joilla on tarvittava osaaminen. Kukin tiimiläinen on yhtä tärkeä ja tiimi yhdessä vastaa tuotteen kaikista puolista, ei koskaan yksittäinen henkilö. (Ketterät käytännöt 2009d; Määttä & Rautio 2010)



Kuvio 13. Scrum-prosessi (Ketterät käytännöt 2009d)

Scrum-menetelmän eteneminen esitetään kuviossa 13. Ennen projektin aloittamista luodaan visio siitä, mitä projektilta halutaan. Visioinnin jäl-

keen muodostetaan *tuotteen työlista (product backlog)* tuotteeseen tarvittavista ominaisuuksista. (Huttunen 2006; Ketterät käytännöt 2009e)

Työlistaa lähdetään purkamaan yhdestä neljään viikkoa kestävissä iteraatioissa, joita kutsutaan sprinteiksi. Sprintti sisältää määrittelyä, toteutusta, testausta ja toimittamista. Jokaisen sprintin alussa pidetään *sprintin suunnittelupalaveri (sprint planning meeting),* jossa tuotteen omistaja määrittelee tärkeyden työlistassa luetelluille töille. Tämän jälkeen Scrum-tiimi valitsee tulevan sprintin aikana tehtävät työt ja nämä työt siirretään tuotteen työlistasta *sprintin työlistaan*. Sprintin aikana vaatimusten muuttaminen on kiellettyä ja tiimiläisillä on täysi vapaus käyttää tarpeelliseksi katsomiaan toimenpiteitä, jotta sovittu sprintin päämäärä voidaan saavuttaa. Näin tiimi organisoi itsensä parhaaksi katsomallaan tavalla. (Huttunen 2006; Ketterät käytännöt 2009e; Määttä & Rautio 2010)

Sprintin aikana Scrum-tiimi pitää päivittäisiä palavereja. Palaverin kesto on noin 5-15 minuuttia ja palaverissa jokainen vastaa kolmeen kysymykseen (Ketterät käytännöt 2009e):

1. Mitä teit edellisen päivän aikana?
2. Mitä aiot tehdä seuraavan päivän aikana?
3. Mitkä tekijät estävät tai hidastavat sinua saavuttamasta sprintin tavoitteita?

Palaverit ovat avoimia kaikille, jotka ovat projektista kiinnostuneita, mutta puheenvuorot on varattu vain tiimiläisille ja Scrum-mestarille. Palaverin perimmäisenä tarkoituksena on jakaa kaikille tieto siitä, missä vaiheessa projekti etenee ja mitä ongelmia on noussut esiin. (Ketterät käytännöt 2009e)

Jokaisen sprintin päätteeksi pidetään *sprintin loppupalaveri (sprint review meeting),* jossa tiimi esittelee tuotteen omistajalle valmista tuotetta. Omistaja arvioi toteutettuja toimintoja ja antaa tiimille palautetta. Lisäksi käydään läpi, mikä kuluneessa sprintissä meni hyvin ja mitä voitaisiin parantaa. Lopuksi priorisoidaan kehityskohteet ja uusi sprintti voidaan käynnistää palaamalla alkuun uuden sprintin suunnitteluvaiheeseen. (Huttunen 2006; Ketterät käytännöt 2009e)

3.4 Suunnitelmaohjautuvien ja ketterien menetelmien käytettävyys ja vertailu

Kuten tämän luvun alkupäässä todetaan, ohjelmistokehitysmenetelmät voidaan jakaa karkeasti kahteen ryhmään niiden tuottaman dokumentaation määrän perusteella. Nimensä mukaisesti suunnitelmaohjautuvat menetelmät rakentuvat hyvin pitkälti suunnitelmadokumentaation päälle, kun taas ketterät menetelmät hyödyntävät kasvotusten tapahtuvaa kommunikaatiota sekä väliaikaisia muistiinpanoja ja tarkat dokumentoinnit tuotetaan vasta tuotteen valmistuttua. Huttusen mukaan (2006) dokumentaation lisäksi menetelmille voidaan esittää myös muita vertailullisia piirteitä, kuten alla olevasta taulukosta (Taulukko 1) voidaan nähdä.

Taulukko 1. Ketterien ja suunitelmaohjautuvien menetelmien tyypillisiä piirteitä. (Huttunen 2006)

Osa-alue	Ketterät menetelmät	Suunnitelmaohjautuvat menetelmät
Kehittäjät	Ketteriä, asiantuntevia, rinnakkaiseen työhön kykeneviä, yhteistyökykyisiä	Suunnitelmaorientoituneita, keskitasoiset taidot, pääsy ulkoisiin tietolähteisiin
Asiakkaat	Omistautuneita, asiantuntevia, rinnakkaiseen työhön kykeneviä, yhteistyöhaluisia, riittävät valtuudet	Saavat tietonsa asiantuntevilta henkilöiltä, yhteistyöhaluisia, valtuutettuja
Määrittelyt	Pitkälti työn ohessa esiin tulevia, nopeasti muuttuvia	Aikaisin tiedossa olevia, pitkälti muuttumattomia
Järjestelmän arkkitehtuuri	Suunniteltu tämän hetkisiin tarpeisiin	Suunniteltu tämän hetkisiin ja tulevaisuuden tarpeisiin
Rakenteen parantaminen	Edullista	Kallista
Koko	Pienemmät tiimit ja tuotteet	Suuremmat tiimit ja tuotteet
Ensisijainen tavoite	Nopea hyödyn tuottaminen	Korkea luotettavuus

Kehittäjille (ohjelmoijille, suunnittelijoille tai laajemmin koko toteuttajayritykselle) asetettavat vaatimukset poikkeavat ketterissä ja suunnitelmaohjautuvissa menetelmissä merkittävästi. Avainsanoja ovat asiantuntemus ja kommunikointitaito. Ketterissä menetelmissä kehittäjiltä edellytetään laajaa asiantuntemusta, sillä selkeitä roolituksia (ohjelmoija, testaaja, sovellusarkkitehti) ei ole, vaan kaikki toteutustiimin jäsenet suorittavat yhdessä asiakkaan kanssa suunnittelun, ohjelmoinnin ja testauksen. Suunnitelmaohjautuvat menetelmät eivät nojaa yksittäisten henkilöiden kykyihin niin merkittävästi, vaan osaamista täydennetään laajemmilla suunnitelmilla, dokumentaatiolla ja tiedon hakemisella ulkoisista lähteistä, esimerkisi kirjoista, Internetistä tai konsulteilta. Ketterissä menetelmissä periaatteena tiedon siirrossa ja kommunikoinnissa ylipäätään on ihmisten välinen keskustelu, suunnitelmaohjautuvissa tieto välitetään paperein ja tiedostoin. Tästä nousee esiin kommunikoinnin viestinnän suunnat: keskustelu on yleensä kaksisuuntaista, kun dokumentit siirtävät viestiä vain yhteen suuntaan. Yhtenä ketterien menetelmien ongelmana pidetään osaavien kehittäjien löytämistä. Huttusen mukaan (2006) noin puolet maailman ohjelmistokehittäjistä ovat keskitasoa huonompia työssään. Tämä tilanne rajaa osaltaan merkittävästi ketterien menetelmien käyttömahdollisuuksia, sillä henkilöstön tietojen ja taitojen puute saattaa puoltaa suunnitelmaohjautuvien menetelmien käyttöä ketterien menetelmien sijasta. (Huttunen 2006)

Huttusen mukaan (2006) menestyksekäs ohjelmistoprojekti edellyttää *asiakkaalta* seuraavia piirteitä: yhteistyökykyä, asiantuntemusta, sitoutumista ja valtaa. Etenkin ketterissä menetelmissä nämä ominaisuudet nousevat esiin, sillä asiakkaan tulee olla jatkuvasti kehittäjien käytettävissä ja hänellä tulee olla riittävät valtuudet hyväksyä muutoksia nopeasti. Ketterille menetelmille aiheutuu näistä vaatimuksista kuitenkin se riski, että asiakkaalta ei saada käyttöön riittävän pätevää ja käytettävissä olevaa asiantuntemusta. Monasti yritykset tarjoavat kehittäjien avuksi sellaisen henkilön, jolla on eniten vapaata aikaa, ei eniten asiantuntemusta. (Huttunen 2006)

Suunnitelmaohjautuvat menetelmät edellyttävät asiakasta, jonka kanssa pystytään etukäteen tekemään tarkat sopimukset, suunnitelmat ja määrittelyt. Edellä kuvatut hyvän asiakkaan ominaisuudet pätevät myös suunnitelmaohjautuvissa menetelmissä, mutta eivät ole niin välttämättömiä kuin ketterissä menetelmissä. Suunnitelmaohjautuvissa menetelmissä suurin asiakasriski on byrokratia: liian tarkka sopimuksien laatiminen ja seuraaminen voi johtaa viivästyksiin asiakkaan edustajan etsiessä valtuutusta omasta organisaatiostaan. (Huttunen 2006)

Määrittelyt kuvataan useimmissa ketterissä menetelmissä joustaviksi ja epämuodollisiksi tarinoiksi. Ketterät menetelmät olettavat, että tulevaisuuden suunnittelu pitkälle ei kannata, sillä muutoksia tulee aina ja näin suunnitelmat vanhentuvat nopeasti. Suunnitelmaohjautuvat menetelmät suosivat yksityiskohtaisia ja muodollisia määrittelyjä, jotka esitetään kirjallisessa dokumentissa. Niiden heikkoutena on ohjelmiston ominaisuuksien priorisoinnin vaikeus sekä muutokseen sopeutumattomuus. Toisaalta suunnitelmaohjautuvien menetelmien määrittelytyön käytännöissä on myös tiettyjä vahvuuksia: epäfunktionaaliset määritteet, kuten luotettavuus, suorituskyky ja skaalautuvuus tulevat paremmin esiin suunnitelmaohjautuvissa menetelmissä. Näillä on keskeinen merkitys erityisesti suurissa, kriittisiä tehtäviä hoitavissa järjestelmissä. (Huttunen 2006)

Järjestelmän arkkitehtuurilla tarkoitetaan toteutettavan järjestelmän teknisiä ratkaisuja. Ketterät menetelmät suosivat tekniikkaa, joka täyttää vain tämän hetkisen tarpeen. Tämä on hyödyllistä etenkin silloin, kun tulevaisuutta ei voida ennustaa. Suunnitelmaohjautuvissa menetelmissä varaudutaan alusta lähtien etukäteismäärittelyillä ja arkkitehtuurisuunnitelmilla ennustettavissa olevaan muutokseen. Tämä antaa toteuttajille mahdollisuuden tehdä sovellukseen liityntöjä ja ominaisuuksia, jotka auttavat toteuttamaan uudet ominaisuudet nopeasti tulevaisuudessa. (Huttunen 2006)

Rakenteen parantamisen (refaktoroinnin) edullisuus ketterissä menetelmissä perustuu ideaan, että jatkuvat parannukset ehkäisevät isojen ja kalliiden muutostöiden tarvetta. Menetelmä toimii silloin, jos kyseessä on suhteellisen pieni järjestelmä ja ohjelmoijat ovat todellisia asiantuntijoita. Heikompitaitoisilla ohjelmoijilla ja laajemmilla järjestelmillä tilanne voi olla toinen, jolloin rakenteen parantamiseen joudutaan panostamaan paljon, mikä vuorostaan tarkoittaa suurta määrää suunnittelutyötä. Järjestelmien kokojen kasvaessakaan ei voida luottaa edulliseen rakenteen kehitystyöhön. Esimerkiksi jotkin aluksi hyvänä pidetyt yksinkertaiset arkkiteh-

tuuriratkaisut eivät välttämättä enää riitäkään suurempiin järjestelmiin. (Huttunen 2006)

Ketterillä menetelmillä toteutettavien projektien *koko* on yleensä pienempi kuin suunnitelmaohjautuvilla menetelmillä. Yleisesti ketterissä menetelmissä tiimin optimikokona pidetään noin kymmentä henkilöä menetelmien ihmiskeskeisen viestintätavan vuoksi. Ketterien menetelmien skaalaaminen suurempiin projekteihin tapahtuu pilkkomalla projekti sopiviin osiin ja pienet ketterät kehitystiimit toimivat itsenäisesti omaa osaansa kehittäen. Osien yhteensopivuudesta varmistutaan liittämällä paloja yhteen tiheään tahtiin ja toimittamalla uusia ohjelmaversioita usein. Vaikka ketterien menetelmien käyttö onkin näin mahdollista myös laajemmissa projekteissa, ollaan yleisesti sitä mieltä, että suunnitelmaohjautuvat menetelmät ovat tehokkaampia suurissa projekteissa. (Huttunen 2006)

Ensisijainen tavoite ketterissä menetelmissä on Agile Manifeston mukaan ”täyttää asiakkaan vaatimukset jatkuvilla ja riittävän aikaisilla ohjelmistotoimituksilla.” Tämä periaate toimii hyvin, kunhan järjestelmien koko ei ole kovin suuri. Suurten järjestelmien ollessa kyseessä keskittyminen konkreettisten lopputulosten saamiseen asiakkaan nähtäville saattaa johtaa pahimmassa tapauksessa merkittävän suuriin ongelmiin, mikäli mittakaavaa ei voidakaan suurentaa olemassa olevalla rakenteella. Suunnitelmaohjautuvien menetelmien tavoitteita ovat perinteisesti olleet *ennustettavuus, toistettavuus, ja optimointi*. Nämä tavoitteet ovat toimivia ainoastaan tilanteissa, joissa tulevaisuus on tiedossa. Samojen tavoitteiden puolesta suunnitelmaohjautuvat menetelmät sopivat erityisen hyvin tilanteisiin, joissa vaaditaan korkeaa luotettavuutta, kuten elintärkeitä järjestelmiä ylläpitävissä ohjelmistoissa. (Huttunen 2006)

Arvioitaessa erityisesti ketterien menetelmien soveltuvuutta käsillä olevaan työhön, on olemassa muutamia käytännön asioita, joihin on syytä kiinnittää huomiota. Huttunen listaa (2006) kuusi keskeistä asiaa, jotka asettavat rajoitteita ketterien menetelmien hyödyntämiselle.

1. **Hajautetut kehitysympäristöt**.
   Työskenneltäessä hajautetuissa kehitysympäristöissä, kasvokkain tapahtuva kommunikointi on hankalaa, ellei jopa mahdotonta, vaikkakin tilannetta voidaan parantaa videoneuvotteluilla ja muilla nykyaikaisilla viestintävälineillä.

2. **Alihankinta**.
   Alihankinnan haasteet liittyvät sopimusteknisiin asioihin: toteutettavaksi sovittavan osakokonaisuuden rajaaminen on ketterien menetelmien työtavoilla haastavaa.

3. **Uudelleen käytettävät ohjelmistokomponentit**.
   Ketterillä menetelmillä kehitettyjen ohjelmistojen komponenttien uudelleenkäytettävyys kyseenalaistetaan usein, koska perustavoitteiden mukaisesti komponentit suunnitellaan täyttämään nykyhetken tarpeet, eikä tulevaisuutta mietitä. Mikäli on tiedossa, että kyseisiä ohjelmisto-

komponentteja tullaan käyttämään myös jatkossa, saattaa tilanteeseen sopia jokin suunnitelmaohjautuva menetelmä.

4. **Suuret kehitystiimit**.
   Suurien kehitystiimien kohdalla organisaation hallinta edellyttää useampia kommunikaatioväyliä kokonaisuuden hallintaan ja tähän ketterien menetelmien keskusteluun pohjautuva viestintä ei välttämättä sovellu.

5. **Turvallisuuskriittiset ohjelmistot**.
   Turvallisuuskriittisiin ohjelmistoihin liittyy haaste laadunvarmistuksesta. On epäilty, etteivät ketterien kehitysmenetelmien tavat ole riittäviä turvallisuuskriittisille ohjelmistoille. Suunnitelmaohjautuvien menetelmien yksityiskohtaisesti määritellyt testitapaukset ja testisuunnitelmat tarjoavat tähän paremman, mutta myös kalliimman tavan.

6. **Laajat, monimutkaiset ohjelmistot**.
   Kuten tässä työssä aiemmin on mainittu, jatkuva koodin parantelu ei poista suunnittelun tarvetta suurissa järjestelmissä. Kasvaessaan liian suuriksi, ohjelmistokomponenttien uudelleenohjelmointi ei ole enää järkevää. Siksi komponentit tulisi suunnitella perusteellisesti etukäteen, jotta niitä voitaisiin hyödyntää jatkossa tehokkaammin.

(Huttunen 2006)

Valinnan tekeminen menetelmäsuuntausten välillä nähdään usein puolueellisena, samasta syystä myös vaikeana. On kuitenkin esitetty, että tällaisesta kaksijakoisuudesta olisi hyvä luopua, sillä ohjelmistokehityksessä voidaan hyvin yhdistää piirteitä niin ketteristä kuin suunnitelmaohjautuvistakin menetelmistä. (Huttunen 2006)

4 LOPPUSANAT

Tämän työn tavoitteena oli tuottaa pienen käsikirjan tyyppinen toteutus sovellusprojekteista sekä sovelluskehitysmenetelmistä. Lähtiessäni hahmottelemaan työtä, huomasin aihealueen olevan varsin laaja. Luonnollisesti työn rajaamisesta tuli ensimmäinen haaste, sillä erilaisia sovelluskehitysmenetelmiä on todella paljon. Rajaukseen vaikuttivat lopulta pääasiassa sovelluskehitysmenetelmien tunnettavuus sekä oma aikatauluni.

Lähdin rakentamaan työtäni projektin määrittelemisellä. Yleisellä tasolla projektit voidaan jakaa karkeasti suunnittelu-, toteutus-, testaus- ja käyttöönotto- sekä ylläpitovaiheisiin. Yllätykseksenі havahduin huomaamaan tässä jo tutun kaavan: vesiputousmallin.

Tutkiessani eri sovelluskehitysmenetelmiä ja niiden toimintatapoja, en voinut olla huomaamatta, että kaikki menetelmät ainakin jossakin määrin hyödyntävät vesiputousmallin elementtejä. Esimerkiksi ketteriin kehitysmenetelmiin kuuluvat Scrum ja Extreme Programming sisältävät kumpikin omanlaisensa suunnittelu-, toteutus- ja testausvaiheet, tosin suppeammassa mittakaavassa. Tästä seikasta päädyin johtopäätökseen, että vaikka vesiputousmalli sellaisenaan onkin raskas menetelmä toteuttaa kaiken vaatimansa etukäteissuunnittelun vuoksi, menetelmästä on poimittavissa paljon käyttökelpoisia ja hyväksi todettuja käytäntöjä nykyajan projekteihin. Vesiputousmalli tarjoaa myös luotettavat puitteet projekteille, joissa teknінen osaaminen on vähäistä.

Miettiessäni ketterien ja suunnitelmaohjautuvien menetelmien käytettävyyttä, mielessäni heräsi kysymys, että mikä mahtaa todellisuudessa olla kumpaisenkin menetelmäryhmän käyttöaste? Huttusen mukaan (2006) monet ohjelmistoalan yritykset edelleen harkitsevat, kannattaisiko heidän käyttää suunnitelmaohjautuvia vai ketteriä menetelmiä. Tekniikka kehittyy kuitenkin vauhdilla ja ketteristä menetelmistäkin on jo runsaasti enemmän kokemusta. Liikemaailman tahti on vain kiihtynyt samoin kuin asiakkaiden vaatimukset ja aikataulut siinä ohessa. Näistä seikoista voisi nopeasti päätellä, että ketterällä kehityksellä olisi nyt otolliset ajat. Näin varmasti onkin, ainakin osassa ohjelmistoyrityksistä, joissa on riittävästi osaamista. Kuitenkin työn kirjoittamisen aikana nousi monta kertaa esiin, että ketterät menetelmät soveltuvat ennen kaikkea pienille projekteille niin laajuudeltaan kuin henkilömääriltäänkin. Karrikoidusti ajatellen tämä sulkisi kaikki isommat järjestelmät pois ketterän kehityksen piiristä.

Kuten Huttunen (2006) työssään totesi, tästä menetelmien kaksijakoisuudesta olisi hyvä luopua, sillä ohjelmistokehityksessä voidaan helposti yhdistellä eri piirteitä niin suunnitelmaohjautuvista kuin ketteristäkin menetelmistä. Itse kannatan myös tätä ajatusta. Menetelmät eivät ole kuitenkaan kiveen hakattuja sääntöjä vaan ennemminkin hyväksi todettuja, eri tilanteisiin sopivia käytäntöjä, joilla saavuttaa asetetut päämäärät.

Kaiken kaikkiaan opinnäytetyöprosessi on ollut opettavainen, vaikka aluksi minulla ei ollut aiheesta juuri minkäänlaista käsitystä. Olen oppinut paljon projektitoiminnasta ja sovelluskehityksestä teoriapohjalla. Kenties jonain päivänä pääsen toteuttamaan oppejani myös käytännössä.

LÄHTEET

Agile Alliance 2011a. Manifesto for Agile Software Development. Viitattu 26.7.2011.
http://www.agilemanifesto.org/

Agile Alliance 2011b. Principles behind the Agile Manifesto. Viitattu 26.7.2011.
http://www.agilemanifesto.org/principles.html

Ahlstedt, Nina 2010. Ohjelmistoprojektin etenemisen ja projektiviestinnän tarkastelu. Lahden ammattikorkeakoulu. Liiketalouden koulutusohjelma. Opinnäytetyö.

Boström, Karl-Erik 2010. Ohjelmistokehitysprosessi Cadpool oy:ssä. Metropolia Ammattikorkeakoulu. Tuotantotalouden koulutusohjelma. Insinöörityö.

Extreme Programming 2009a. Extreme Programming: A gentle introduction. Viitattu 27.7.2011.
http://www.extremeprogramming.org/

- 2009b. Iteration planning. Viitattu 27.7.2011.
http://www.extremeprogramming.org/rules/iterationplanning.html

Huttunen, Janne 2006. Ketterän ohjelmistokehitysmenetelmän määrittely, vertailu ja käyttäjäkysely. Helsingin teknillinen korkeakoulu. Sähkö- ja tietoliikennetekniikan osasto. Diplomityö.

Ketterät käytännöt 2009a. Vaatimuksia vai lisäarvoa? Onko projekti vaatimusten täyttämistä vai lisäarvon tuottamista?
Viitattu 26.7.2011.
http://www.ketteratkaytannot.fi/fi-FI/Ketteryys/VaatimuksiaVaiLisaarvoa/

- 2009b. Menetelmät. XP. Viitattu 27.7.2011.
http://www.ketteratkaytannot.fi/fi-FI/Menetelmat/XP/

-2009c. Menetelmät. Crystal.Viitattu 27.7.2011.
http://www.ketteratkaytannot.fi/fi-FI/Menetelmat/Crystal/

-2009d. Menetelmät. Scrum. Viitattu 27.7.2011
http://www.ketteratkaytannot.fi/fi-FI/Menetelmat/Scrum/

-2009e. Menetelmät. Scrum. Aktiviteetit. Viitattu 27.7.2011.
http://www.ketteratkaytannot.fi/fi-FI/Menetelmat/Scrum/Aktiviteetit/

Louhelainen, Timo 2008. Kuinka projekti toimii? Lahden ammattikorkeakoulu. Kone- ja tuotantotekniikan koulutusohjelma. Opinnäytetyö.

Määttä, T. & Rautio, P. 2010. Toiminnallinen määrittelydokumentti ketterästi. Rovaniemen ammattikorkeakoulu. Tietojenkäsittelyn koulutusohjelma. Opinnäytetyö.

PK-RH 2009. Pk-yrityksen riskienhallinta. Viitattu 29.6.2011 http://www.pk-rh.fi/startti-riskienhallintaan/mita-riskienhallinta-on/riskien-suuruuden-arviointi/riskien-suuruuden-arviointi

Stankovic, A., 2010. Web- projektin aikainen testaus. Vaasan ammattikorkeakoulu. Liiketalous ja matkailu. Opinnäytetyö.

Wikipedia 2011. Ohjelmistotuotanto. Viitattu 28.7.2011. http://fi.wikipedia.org/wiki/Ohjelmistotuotanto

Liite 1

Agile-manifesti