Ohjelmistoala ja ryhmätyöskentely

Kenny Heinonen

Aine
Helsingin Yliopisto
Tietojenkäsittelytieteen laitos

Helsinki, 1. maaliskuuta 2013

Sisältö

1 Johdanto **2**

2 Ketterät menetelmät ja ryhmätyöskentely **2**
2.1 Extreme Programming 3
2.1.1 Kommunikointi 3
2.1.2 Pariohjelmointi 3
2.1.3 Julkaisusuunnittelun kokous 5
2.1.4 Iteraatiosuunnittelun kokous 5
2.1.5 Iteraatio 5
2.2 Scrum 6
2.2.1 Tiimi 6
2.2.2 Scrumin aloitus 7
2.2.3 Sprintin suunnittelupalaveri 8
2.2.4 Scrumin päiväpalaveri 8
2.2.5 Sprintin katselmointi 8
2.2.6 Sprintin retrospektiivi 9

3 Persoonallisuuden vaikutus **9**
3.1 Myers-Briggsin tyyppi-indikaattori 10
3.2 Hyvän ohjelmistokehittäjän piirteet 11
3.3 Persoonallisuustyypit eri kehityksen vaiheissa 12
3.3.1 Vaatimusmäärittely 12
3.3.2 Suunnittelu 13
3.3.3 Toteutus 13
3.3.4 Testaus 14
3.3.5 Ylläpito 15
3.4 Huonon ohjelmistokehittäjän piirteet 16

4 Ryhmätyötaitojen parantaminen **16**
4.1 Kevyet menetelmät 17
4.1.1 Vuorovaikutteinen luokkahuone 17
4.1.2 Tarkkailija-kommunikoija-rakentaja 17
4.2 Suuret menetelmät 17
4.2.1 Project-based learning 17
4.2.2 Rocking the boat 17

5 Yhteenveto **17**

Lähteet **17**

1 Johdanto

Ohjelmistojen kehityksen perustana on ryhmätyöskentely, koska projektien kasvava kompleksisuus ja laajuus tekee niiden toteuttamisen yksilölle vaikeaksi [16]. Ohjelmistoalalla tarvitaan monenlaisia teknisiä taitoja, jotta projekteissa saadaan toteutettua kaikki kehityksen vaiheet kattavasti. Nämä ohjelmistokehityksen vaiheet voidaan jakaa karkeasti vaatimusmäärittelyyn, suunnitteluun, toteutukseen, testaukseen ja ylläpitoon [9][1]. Sen lisäksi, että eri vaiheet vaativat eri taitoja, myös yksittäinen vaihe kysyy laajaa osaamista. Tämän seurauksena tulee tarve koota joukko osaavia ihmisiä toteuttamaan yhteistyössä kaikki kehityksen vaiheet. Onkin hyvin tavanomaista, että ohjelmistoprojektit toteutetaan ryhmätyönä.

Ryhmätyö vaatii teknisten taitojen lisäksi yhteistyö- ja kommunikointitaitoja, koska se vaikuttaa ryhmän suorituskykyyn [15]. Ryhmätyön merkitys ollaan tunnettu jo kauan ja sitä painotetaan yhä enemmän tietojenkäsittelytieteiden opetusohjelmissa [11, 16, 17, 18]. Useat prosessimallit jopa sanelevat miten ryhmätyöskentely tapahtuu, jotta kehitys sujuisi luontevammin ja tuotteliaammin. Kehittäjien välinen yhteistyö riippuu myös yksilöstä ja siitä miten heidän persoonallisuudet ja luonteenpiirteet sopivat yhteen [5, 15]. Tarkoituksena on tarkastella ketteriä menetelmiä ja tutkia miten muutamat esimerkkimenetelmät määräävät ohjelmistokehityksen luonteesta painottaen ryhmän sisäistä toimintaa. Lisäksi kuvataan millainen on ideaali ryhmän jäsen ja millä luonteenpiirteillä on vahvimmat positiiviset vaikutukset kehityksen eri vaiheisiin.

2 Ketterät menetelmät ja ryhmätyöskentely

Ketterät menetelmät pohjautuvat 12 ketterän kehityksen periaatteeseen [1]. Ryhmätyöskentelyllä ja yhteistyöllä on ketterissä menetelmissä suuri painoarvo. Esimerkiksi eräs periaate on, että *”parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itseorganisoituvissa tiimeissä”*. Tämä tarkoittaa pitkälti sitä, että tiimit työskentelevät yhdessä päättäen asioista ilman, että kukaan ulkopuolinen tulee kertomaan heille mitä tehdä ja miten. Toisin sanoen tiimiä johtaa tiimi itse, tiimillä ei ole johtajaa tai projektipäällikköä. Periaatteissa painotetaan myös informaation välittämistä kasvokkain käytävillä keskusteluilla. Ryhmillä tulisi olla yhteiset työtilat, jolloin tämä periaate toteutuisi. Kasvokkainen kommunikointi virtaviivaistaa tiedon kulkua ja pienentää väärinkäsitysten mahdollisuutta.

[1] Capretz puhuu järjestelmäanalyysista, joka on sidoksissa vaatimusmäärittelyyn. Tässä tekstissä käsitellään järjestelmäanalyysin sijaan vaatimusmäärittelyä.

Ketterissä menetelmissä toimivan ohjelmiston tuottaminen säännöllisin väliajoin on tärkeää ja sillä halutaan pitää asiakas tyytyväisenä. Ketterät menetelmät perustuvatkin iteratiiviseen ja inkrementaaliseen kehitysmalliin, jossa tuotetta kehitetään lyhyissä kehityssykleissä osa kerrallaan [10]. Kehitysmallissa ohjelmiston toteuttamiseen tarvittava työ pilkotaan osiin, jotka aikataulutetaan niin, että kukin osa kehitetään ajan myötä valmiiksi lyhyissä kehityssykleissä. Tuotetta rakennetaan valmiiksi säännöllisin väliajoin, jolloin asiakas näkee kehityksen ja voi antaa palautetta kertoen meneekö kehitys oikeaa suuntaa kohti.

Seuraavissa osioissa tutustumme kahteen tunnettuun ketterään menetelmään nimeltään *Scrum* ja *Extreme Programming* [3, 13, 21, 6]. Muitakin menetelmiä toki on, mutta ne jätetään käsittelemättä. Osioissa kerrotaan millaisen prosessin nämä menetelmät määrittelevät projektin kehitykselle ja miten niissä otetaan ryhmätyöskentely ja kommunikointi huomioon.

2.1 Extreme Programming

Extreme Programming (XP) on ketterä menetelmä, jonka tunnettu ohjelmistokehittäjä Kent Beck on luonut. XP keskittyy asiakkaan tyytyväiseen. XP:n tarkoituksena on tuottaa asiakkaalle mahdollisimman paljon arvoa mahdollisimman tehokkaasti ja nopeasti. XP:ssä kehitys tapahtuu lyhyissä, 1-4 viikon iteraatioissa, jolloin vaatimusten muutoksiin on helpompi varautua, kun tavoitteet on asetettu lähitulevaisuuteen. XP painottaa ketterien menetelmien tapaan kommunikointia ja ryhmätyötä.

2.1.1 Kommunikointi

Kasvokkaista ja usein tapahtuvaa kommunikointia painotetaan järjestämällä tiimin jäsenet yhteiseen tilaan. Tämän lisäksi projektia varten on hankittu vähintään yksi paikan päällä oleva asiakas, joka on osana kehitystiimiä nähdäkseen projektin edistymisen. Paikan päällä olevan asiakkaan kanssa keskustellaan kehityksen jokaisesta vaiheesta ja hän päättää tuotteen vaatimuksista ja niiden priorisoinnista. Asiakkaan läsnäolo on suuri etu, sillä tiimi voi kysyä häneltä hetkessä esimerkiksi jonkin toteutettavan vaatimuksen yksityiskohdista ja asiakas vastaavasti voi antaa heti palautetta työstä. Ryhmän sisäinen kommunikointi on myös suuressa osassa sillä hyvät ratkaisut saadaan useimmiten yhteistyön tuloksena. Kommunikoinnin ja yhteistyön merkitystä kuvastaa esimerkiksi *pariohjelmointi* (pair programming), joka on yksi XP:n keskeisimpiä käytänteitä. Tutustutaan pariohjelmointiin tarkemmin.

2.1.2 Pariohjelmointi

Pariohjelmoinnissa kaksi henkilöä ohjelmoivat pareittain, jakaen saman tietokoneen [19]. Henkilöt eivät kuitenkaan ohjelmoi samaan aikaan, vaan heille

on nimetty kaksi roolia: toinen on *ajaja* (the driver), ja toinen *navigoija* (the navigator). Ajajan tehtävänä on yksinkertaisesti kirjoittaa koodia. Navigoijan tehtävänä on analysoida jatkuvasti kirjoitettua koodia ja kertoa ajajalle mitä tehtäviä heidän tulee milloinkin toteuttaa. Näin ajaja voi keskittyä pelkästään ohjelmointiin. Sovituin väliajoin henkilöt vaihtavat rooleja. Pariohjelmointi voidaan nähdä ryhmätyöskentelynä — kaksi ihmistä suorittavat yhteistä tehtävää saavuttaakseen saman päämäärän.

Pariohjelmointi korostaa yhteistyötä ja kommunikointia. Tästä on monia hyötyjä, jotka tekevät hyvää sekä ryhmän jäsenille, ryhmälle että projektille [7]. Tarkastellaan näitä hyötyjä seuraavaksi.

- **Koodin laatu**

  Kun kaksi henkilöä pariohjelmoivat ja ratkaisevat samaa ongelmaa, lopputulos on usein tehokkaampi verrattuna siihen, että yksi henkilö tekisi kaiken. Parit pystyvät helposti keskustelemaan keskenään siitä mitä heidän tulisi seuraavaksi tehdä ja he voivat jakaa ideoita saadakseen koodista laadukkaamman tai ratkaistakseen jonkin ongelman. Sivustakatsojana navigoija pystyy tekemään tärkeitä huomioita ja pohtimaan kuinka koodista saataisiin laadukkaampi, kun taas ajaja voi keskittyä ohjelmoimiseen. Navigoija tekee ajoittain ehdotuksia ajajalle, jolla koodista saataisiin parempaa. Pariohjelmoinnin ansiosta tapahtuva laajamittaisempi koodin katselmointi vähentää virheiden määrää. Vaihtoehtoisesti niitä ei edes synny, kun parit keskenään kommunikoivat miettien hyviä ratkaisuja.

  Pariohjelmointi parantaa myös keskittymiskykyä. Toisen henkilön läsnäolo estää herkemmin yksilöä laiskottelemasta tai rikkomaan XP:n vaalimia käytänteitä, kuten TDD:tä. Käytänteiden noudattaminen taas johtaa parempaan koodin laatuun ja toteutukseen.

- **Oppiminen**

  Kun henkilöt ”pariutuvat” toistensa kanssa, niin osaaminen leviää kehittäjien kesken. Monet tykkäävät neuvoa toisiaan ja ryhmän jäsenillä on eri taitoja ja tietämystä asioista. Esimerkiksi ryhmän jäsenet, jotka pääsevät heitä taitavampien ihmisten pareiksi voivat oppia paljon uusia tekniikoita. Parhaimmassa tapauksessa koko ryhmän sisällä kaikki voivat oppia toisiltaan jotakin. Oppimiseen sisältyy, että ymmärrys kehitettävästä ohjelmasta parantuu. Jos henkilöt vaihtavat pareja eri ihmisten kanssa, he pääsevät näkemään erilaisia kehityksessä olevia ohjelman osia. He myös samalla osallistuvat tämän yhden osan kehitykseen, jolloin ymmärrys koko projektista kasvaa korkeammalle tasolle.

Tästä voidaan päätellä, että pariohjelmointi on suuressa roolissa XP:ssä

ja ryhmän jäsenien keskinäisellä vuorovaikutuksella on suuri hyöty projektin laadun kannalta. Kaikkien näiden pariohjelmoinnin hyvien puolien yhteisenä tekijänä on, että parit kommunikoivat toistensa kanssa. Ilman keskinäistä vuorovaikutusta ei voi odottaa, että edellä mainitut asiat toteutuisivat. Pariohjelmointi kuitenkin rohkaisee kehittäjiä puhumaan toisilleen, joten tästä ei pitäisi olla huolta [23].

2.1.3 Julkaisusuunnittelun kokous

Kun asiakas on tehnyt listan vaatimuksia, joita kehitettävän järjestelmän pitää sisältää, aloitetaan XP:n prosessi *julkaisusuunnittelun kokouksella* (release planning meeting), jossa on tarkoituksena tehdä *julkaisusuunnitelma* (release plan). XP:ssä on tapana iteraatioiden lopuksi julkaista uusin, toimiva versio asiakkaiden käyttöön. Julkaisusuunnitelma määrittelee, mitkä vaatimukset toteutetaan missäkin iteraatiossa ja siten ovat valmiina iteraation lopuksi tehtävässä julkaisussa. Näille vaatimuksille määritellään myös päivämäärät.

Kokouksessa kehitystiimin tehtävänä on arvioida kuinka paljon yksittäinen vaatimus vie työaikaa. Estimointien perusteella asiakas tekee päätöksen vaatimusten prioriteeteista. Tämän jälkeen tiimin jäsenet yhdessä asiakkaan kanssa sijoittavat vaatimukset tietyille iteraatioille toteutettaviksi.

2.1.4 Iteraatiosuunnittelun kokous

Ennen kuin varsinainen iteraatio alkaa, jossa itse tuotteen tiettyjen toiminnallisuuksien kehitys tapahtuu, järjestetään *iteraatiosuunnittelun kokous* (iteration planning meeting), jossa tehdään *iteraatiosuunnitelma* (iteration plan). Iteraatiosuunnittelun tavoitteena on päättää mitä iteraation aikana tehdään, joka kirjataan iteraatiosuunnitelmaan.

Suunnittelun alussa asiakas valitsee julkaisusuunnitelmasta korkeimman prioriteetin omaavat vaatimukset alkavaan iteraatioon. Kehittäjät pilkkovat nämä vaatimukset pieniksi, toteutettaviksi tehtäviksi. Tämän jälkeen kehittäjät päättävät ketkä toteuttavat minkäkin tehtävän ja he estimoivat tehtäviin kuluvan työmäärän. Lopuksi vaatimukset, tehtävät ja niiden estimaatit kirjataan iteraatiosuunnitelmaan.

2.1.5 Iteraatio

Iteraatiosuunnittelun ja onnistuneen suunnitelman tekemisen jälkeen voidaan aloittaa itse iteraatio. Iteraatio kestää 1-4 viikkoa, jonka aikana iteraatiosuunnitelmaan kirjatut vaatimukset ja niihin liittyvät tehtävät pitää toteuttaa. Kehitystyö tapahtuu pariohjelmoiden, jonka merkitystä ryhmätyöskentelyyn tarkasteltiin luvussa 2.1.2. Iteraatioon liittyy lisäksi päivittäinen palaveri,

jossa ryhmän jäsenet raportoivat toisilleen mitä saivat viime palaverin jälkeen aikaan, mitä aikovat saada aikaan ennen seuraavaa palaveria ja onko heidän etenemisessään ongelmia. Näin tiimin jäsenet ovat jatkuvasti tietoisia muiden jäsenten sekä koko tiimin tilanteesta. Kuten edellä on mainittu, iteraation aikana on vähintään yksi asiakas läsnä joka päivä, jonka kanssa voi puhua ongelmatilanteissa toteutettavien vaatimusten yksityiskohdista ja neuvotella kehityksestä.

Iteraation lopuksi suoritetaan *hyväksymätestaus* (acceptance testing), jossa testataan iteraation aikana toteutettuja vaatimuksia ja varmistutaan siitä, että ne toimivat oikein. Asiakas on määritellyt tarkemmin mitä skenaarioita testien tulisi testata. Jos yksittäinen vaatimus läpäisee kaikki siihen kohdistuvat testit, on vaatimus toteutettu valmiiksi. Asiakas itse tarkistaa testitulokset ja päättää niiden perusteella onko viimeisin versio tuotteesta julkaisukelpoinen. Jos testauksen aikana löytyy ohjelmointivirheitä, ne kirjataan ylös ja korjataan seuraavan iteraation aikana.

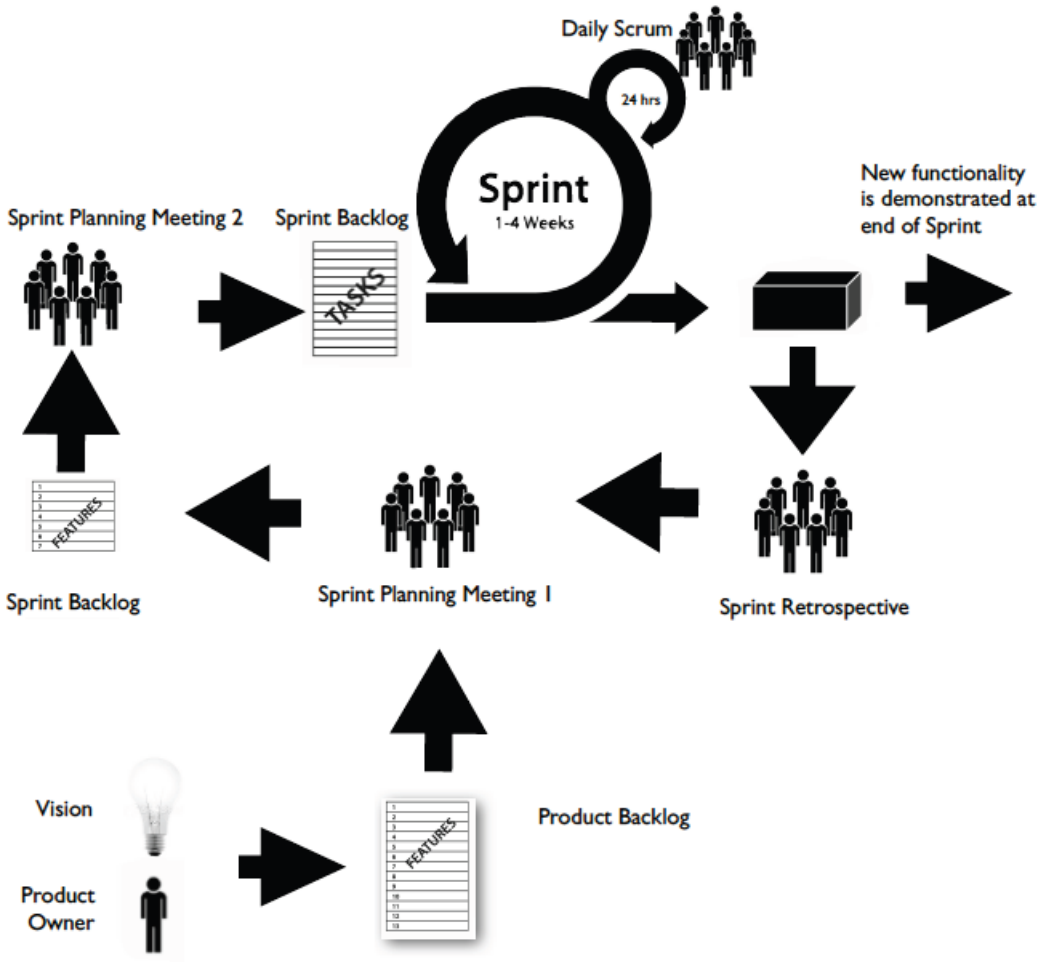
Ketteränä menetelmänä XP on iteratiivinen, joten kun yksi iteraatio on loppunut, aloitetaan edellä mainittu prosessi uudestaan alkaen julkaisusuunnittelun kokouksella. Näin XP:n prosessia jatketaan, kunnes tuote on valmis. XP painottaa hyvin paljon asiakkaan kanssa kommunikointia ja läsnäoloa, jotta kehityksessä ei päädytä sivuraiteille. Asiakas on osana ryhmää ja hänen kanssaan tapahtuva kommunikointi ja yhteistyö vaikuttaa suuresti projektin onnistumiseen yhdessä ryhmän sisäisen kommunikoinnin kanssa.

2.2 Scrum

Scrum on prosessimalli, joka painottaa tiimien yhteistyötä projektin kehityksessä [2]. Scrum, kuten XP, on myös iteratiivinen ja inkrementaalinen menetelmä. Kehitys tapahtuu lyhyissä 1-4 viikon sykleissä, *”sprinteissä”*, joissa toteutetaan kehitettävää tuotetta tietyt toiminnallisuudet kerrallaan. Lyhyet kehityssyklit sallivat tiimien mukautuvan asiakkaalta saatavaan palautteeseen ja vaatimusten muutoksiin. Näin tuotteeseen voidaan tehdä ajoissa muutoksia ilman suuria kuluja ja tuote ”hioutuu” yhä lähemmäs sitä mitä asiakas sen haluaa olevan. Scrum tekee projektin kehityksestä monivaiheisen, kuten kuvassa 1 näkyy. Nämä vaiheet tullaan käsittelemään myöhemmin. Seuraavissa osioissa kerrotaan millainen Scrum-tiimi on ja millainen projektin elinkaari on Scrumissa.

2.2.1 Tiimi

Scrumissa tiimit koostuvat *monitaitoisista* (cross-functional) jäsenistä, joilta löytyy tarpeellinen tekninen osaaminen, jota vaaditaan tuotteen toteuttamiseksi. Tiimillä ei ole johtajaa tai projektipäällikköä, jotka määräisivät tiimin



Kuva 1: Scrumin prosessi [13]

tekemisistä. Sen sijaan tiimi itse saa päättää tavoitteet joka sprintille ja sen miten nämä tavoitteet saavutetaan. Toisin sanoen tiimi on *itseorganisoituva* (self-organizing). Tämä on Scrumille olennaista ja siitä näkee kuinka suuressa arvossa tiimin jäsenten välistä yhteistyötä pidetään [21].

2.2.2 Scrumin aloitus

Scrumin ensimmäinen askel on luoda visio tuotteen vaatimista ei-toiminnallisista ja toiminnallisista vaatimuksista. Vaatimukset kootaan listaksi, jota kutsutaan *tuotteen kehitysjonoksi* (product backlog) [3]. Kehitysjono on priorisoitu siten, että tärkeimmät vaatimukset ovat kehitysjonon kärjessä. Kehitysjono on jatkuvan muutoksen alaisena: uusia vaatimuksia voi tulla lisää asiakkaan toimesta, tarpeettomia vaatimuksia karsitaan ja olemassaole-

via muokataan tai tarkennetaan. Edellä mainitut tehtävät ovat niin kutsutun *tuoteomistajan* (product owner) vastuulla, joka on yksittäinen henkilö ja pitää huolen tuotteen arvon ja kehitystiimin työn arvon maksimoimisesta. Tuoteomistaja voi kuitenkin pyytää kehitystiimiä auttamaan edellä mainituissa tehtävissä.

2.2.3 Sprintin suunnittelupalaveri

Jokaisen sprintin aluksi järjestetään kokous, jota kutsutaan *sprintin suunnittelupalaveriksi* (sprint planning meeting) [21]. Palaveri on kaksiosainen. Ensimmäisessä osassa tuoteomistaja ja kehitystiimi neuvottelevat siitä mitkä vaatimukset tiimin tulisi toteuttaa alkavassa sprintissä. Päätösvastuu on kuitenkin tiimin jäsenillä.

Toisessa osassa tiimi valitsee toteutettavat vaatimukset, jotka he sitoutuvat tekemään sprintin loppuun mennessä. Vaatimukset valitaan aina kehitysjonon kärjestä ylhäältä alas järjestyksessä. Kun vaatimukset on valittu, ne hajotetaan pienemmiksi, teknisiksi tehtäviksi. Nämä tehtävät kirjataan *sprintin tehtävälistaan* (sprint backlog) aika-arvioineen, mitä tiimi käyttää hyödykseen sprintin ajan. Tiimi saa siis itse valita toteutettavat vaatimukset alkavalle sprintille ja suunnitella miten ne toteutetaan.

2.2.4 Scrumin päiväpalaveri

Kun sprintti alkaa, sen aikana harjoitetaan erästä Scrumin käytäntöä: *päiväpalaveria* (daily Scrum). Tämä on lyhyt, noin 15 minuuttia kestävä kokoontuminen joka päivä samaan aikaan, johon kaikki tiimin jäsenet osallistuvat. Jokainen tiimin jäsen saa mahdollisuuden raportoida kolme asiaa muille tiimin jäsenille: mitä henkilö on saanut aikaan viime tapaamisen jälkeen, mitä henkilö aikoo saada aikaan ennen seuraavaa tapaamista ja onko mitään esteitä työn etenemiselle? Näin tiimin jäsenet säännöllisesti tietävät kuinka muiden työ edistyy ja palaverin jälkeen mahdollisia raportoituja ongelmia voidaan yhdessä ratkoa. Huomionarvoista on, että on yleisesti suositeltua, että päiväpalaveriin ei osallistu esimiehiä tai muita ulkopuolisia henkilöitä [21]. Jos näin käy, on riski, että henkilöt tuntevat olevansa tarkkailun alaisena ja tämä voi tuottaa heille paineita oman edistymisen tai ongelmien raportoinnista.

2.2.5 Sprintin katselmointi

Sprintin lopussa järjestetään *sprintin katselmointi* (sprint review), jossa tiimi esittelee sprintin aikana toteuttamiaan vaatimuksia tuoteomistajalle ja asiakkaalle. Asiakas ja tuoteomistaja tietävät näin miten projekti on edistynyt ja tiimi vastaavasti saa arvokasta palautetta kyseisiltä henkilöiltä. Tämä

on motivoivaa kaikille osapuolille. Annettu palaute kirjataan mahdollisina muutoksina vaatimuksiin tai uusina vaatimuksina tuotteen kehitysjonoon.

2.2.6 Sprintin retrospektiivi

Viimeisenä asiana sprintin lopuksi järjestetään *sprintin retrospektiivi* (sprint retrospective), jossa tiimi keskustelee kuinka heidän oma työprosessinsa sujui sprintin aikana [13]. Käytännössä jäsenet keskustelevat yhdessä mikä sprintissä sujui hyvin ja missä asioissa pitäisi parantaa. Jäsenien antama palaute ja kritiikki voi esimerkiksi kohdistua yksittäisen jäsenen työskentelytapoihin. Jos mahdollisia ongelmakohtia on, tiimin tulisi yhdessä päättää miten nämä ongelmat ratkaistaan. Retrospektiivi on tärkeä vaihe ryhmätyöskentelyn kannalta sillä se on Scrumin pääasiallisin mekanismi tuoda tiimin ongelmat näkyville ja ratkaista ne siten, että ryhmän työskentely vahvistuu ja paranee.

Kun nämä kaikki edellä mainitut vaiheet on käyty läpi, alkaa uusi sprintti. Käytännössä aloitetaan uusi sykli aloittaen sprintin suunnittelupalaverista. Tätä periaatteessa jatketaan niin kauan, kunnes tuote on valmis eli asiakkaan kaikki vaatimukset on toteutettu. Kuten monista asioista kävi ilmi, kehitystiimin jäsenillä on paljon valtaa sen suhteen mitä he tekevät ja miten he sen tekevät. Suunnittelutyö ja toteutus on asia, jonka tiimin jäsenten pitää yhteisymmärryksessä päättää kommunikoiden toistensa kanssa.

3 Persoonallisuuden vaikutus

Kehitystiimit koostuvat erilaisista ihmisistä ja useissa tutkimuksissa on huomattu, että tietyt luonteenpiirteet ja persoonallisuustyypit vaikuttavat positiivisesti tiimin suorituskykyyn sekä projektin onnistumiseen [5, 14, 8, 9]. Sen lisäksi, että tiimien olisi hyvä koostua monitaitoisista jäsenistä, moninaiset persoonallisuustyypit tiimin sisällä ovat hyväksi projektille. Esimerkiksi suurempia ratkaisuja mietittäessä harvoin hyväksytään ensimmäinen ehdotus, joka esitetään, vaan puntaroidaan monen ehdotuksen välillä arvioiden niiden hyviä ja huonoja puolia. Myös yksilön oma tekninen osaaminen vaikuttaa ratkaisun miettimiseen. Eri tavalla ajattelevat ihmiset kykenevät tuomaan joukon eri näkökulmia tuotetta kehitettäessä, joka johtaa parempiin ratkaisuihin.

Edellä mainitun lisäksi henkilön persoonallisuus vaikuttaa myös siihen kuinka mieluista kyseisen henkilön kanssa on työskennellä, miten henkilö lähestyy annettua tehtävää ja minkälaisiin tehtäviin hän soveltuu parhaiten [7, 9]. Eräs tapa mitata henkilön persoonallisuutta on käyttää Myers-Briggsin tyyppi-indikaattoria. Seuraavissa osioissa tarkastellaan millaisia piirteitä hyvällä ohjelmistokehittäjällä on ja mitkä erilaiset persoonallisuustyypit sopivat parhaiten projektien eri kehitysvaiheisiin. Ennen tätä kuitenkin

kerrotaan millainen Myers-Briggsin tyyppi-indikaattori on, jonka avulla pohjustetaan tulevia osioita.

3.1 Myers-Briggsin tyyppi-indikaattori

Persoonallisuustyyppejä määriteltäessä käytetään apuna Myers-Briggsin tyyppi-indikaattoria, joka jakaa ihmisen persoonallisuuden neljään eri osioon: sosiaaliseen vuorovaikutukseen, tiedonkeruuseen, päätöksentekoon ja elämäntyyliin [8, 9, 12]. Joka osiossa on kaksi eri luonteenpiirrettä. Kussakin osiossa jokainen ihminen kallistuu enemmän toiseen piirteeseen kuin toiseen eli Myers-Briggsin tyyppi-indikaattorilla saadaan yksittäiselle henkilölle näistä osioista neljä luonteenpiirrettä, jotka määrittävät hänen persoonallisuuden. Yhteensä persoonallisuustyyppejä on $2^4 = 16$ erilaista.

1. Sosiaalinen vuorovaikutus: Ekstrovertti (E) — Introvertti (I)

   Ekstrovertit ovat sosiaalisia, ulospäinsuuntautuneita ja nauttivat muiden ihmisten seurasta. Introvertit sen sijaan ovat sisäänpäinkääntyneitä, hiljaisia, varautuneita ja ovat mieluummin omissa oloissaan.

2. Tiedonkeruu: Tosiasiallinen (S) — Intuitiivinen (N)

   Tosiasialliset ihmiset etsivät yksityiskohtaista tietoa ja tunnettuja tosiasioita. He uskovat enemmän konkreettisiin asioihin, jotka voivat itse omin aistein todistaa. Intuitiiviset etsivät asioille yhteyksiä teoreettisemman ja abstraktisemmankin tiedon pohjalta ja he miettivät eri asioista aiheutuvia mahdollisuuksia.

3. Päätöksenteko: Ajatteleva (T) — Tunteva (F)

   Ajattelevat perustavat päätöksensä miettimällä tarkasti päätöksen syitä, seurauksia ja loogisuutta. He perustavat päätöksensä puhtaaseen järjen käyttöön. Tuntevilla on taipumusta tehdä päätös henkilökohtaisten arvojen perusteella ja sen mukaan miten päätös vaikuttaa muihin ihmisiin. Päätöksenteko-ulottuvuus vaikuttaa siihen minkälaisista tehtävistä henkilö kiinnostuu ja kuinka tyytyväinen hän on niihin.

4. Elämäntyyli: Järjestelmällinen (J) — Spontaani (P)

   Järjestelmälliset ihmiset ovat sananmukaisesti järjestelmällisiä ja täsmällisiä. He pitävät aikamääreistä kiinni ja suunnittelevat asioita etukäteen. Spontaanit taas ovat joustavia ja elävät hetkessä välittämättä suunnitelmallisuudesta ja järjestelmällisyydestä. Elämäntyyli-ulottuvuus vaikuttaa henkilön työskentelytapoihin.

3.2 Hyvän ohjelmistokehittäjän piirteet

Usein todetaan, että hyvälle ohjelmistokehittäjälle on aina tilaa ohjelmistoalalla. Siinä missä tekninen osaaminen ja hyvä tietämys asioista on tärkeää, myös ihmistaidot ja kommunikointi on alkanut keräämään huomiota alalla [15]. Esimerkiksi pariohjelmoinnissa työskentely vaatii kommunikointia ja sitä, että tulee toisten ihmisten kanssa toimeen. Seuraavaksi tarkastellaan ohjelmistokehittäjän piirteitä, jotka tekevät henkilöstä mieluisen työkumppanin teknisten taitojen lisäksi. Tutkimuksissa on haastateltu ohjelmistoalan ammattilaisia ja kysytty heiltä mitkä piirteet ja asiat tekevät hyvän ohjelmistokehittäjän [5, 7, 15]. Haastattelujen perusteella seuraavat piirteet ovat tärkeimpiä:

- **Joustava & mukautuva**

  Joustava henkilö on avaramielinen. Henkilö kuuntelee mielellään muiden ideoita ja kykenee katsomaan asioita eri näkökulmista sen sijaan, että puolustelisi vain omaa kantaansa. Lisäksi hän on halukas yhteistyöhön ja pystyy mukautumaan eri työtapoihin.

- **Hyvät kommunikointitaidot**

  Ohjelmiston kehittäminen vaatii paljon ryhmätyöskentelyä ja kommunikointia tiimin jäsenten kesken. Hyvä kommunikoija kykenee kuuntelemaan muiden ideoita, ilmaisee omat mielipiteensä selkeästi ja uskaltaa kysyä apua, jos hänellä on ongelmia. Kun tiimi koostuu jäsenistä, jotka kommunikoivat usein ja hyvin, se vaikuttaa positiivisesti ilmapiiriin ja työn laatuun. Esimerkiksi kommunikoinnin myötä tieto leviää tiimin jäsenten kesken, joka kasvattaa kaikkien osaamista.

- **Älykäs**

  Ohjelmistokehitys on pohjimmiltaan teknistä työtä, joten ihmistaitojen lisäksi myös älykkyys on erittäin tärkeä piirre. Älykkyydellä tarkoitetaan tässä tapauksessa ohjelmistokehittäjää, jolla on hyvä tekninen osaaminen ja hän on nopea ajatuksissaan. Henkilön ongelmanratkaisutaidot ovat erinomaiset ja hän kykenee ajattelemaan asioita abstraktilla tasolla sekä ottaa vastuuta työstään.

- **Mukava**

  Mukava ohjelmistokehittäjä on sosiaalinen ja hänen kanssaan on helppo työskennellä. Hänellä on huumorintajua ja sopivaa tilannetajua eli esimerkiksi pystyy huomauttamaan virheistä ilman, että toinen pahastuu. Lyhyesti sanottuna kyseisen henkilön kanssa on hauska työskennellä.

  Muita mainittuja piirteitä olivat muun muassa, että ohjelmistokehittäjä on tietäväinen, innovatiivinen, itsenäinen, kykenee keskittymään työhönsä, ja noudattaa käytettyä prosessia.

3.3 Persoonallisuustyypit eri kehityksen vaiheissa

Ohjelmiston kehitys on monivaiheinen prosessi ja usein se on jaettu vaatimusmäärittelyyn, suunnitteluun, toteutukseen, testaukseen ja ylläpitoon. Nämä eri vaiheet vaativat erilaisia teknisiä taitoja, jotta saadaan paras mahdollinen lopputulos. Kuitenkin teknisen osaamisen lisäksi myös persoonallisuustyyppi vaikuttaa siihen kuinka hyvin henkilö suoriutuu tietyistä tehtävistä tai kuinka hyvin hän soveltuu niihin [9]. Kun luonteiltaan oikeanlaiset henkilöt valitaan suorittamaan näitä tiettyjä kehityksen vaiheita, tuloksena on parempi lopputulos projektin kannalta.

Seuraavaksi käydään edellä mainitut kehityksen vaiheet läpi yksi kerrallaan ja tarkastellaan millaiset ohjelmistokehittäjät sopivat parhaiten mihinkin vaiheeseen. Oletetaan, että kehittäjillä on hyvä tekninen osaaminen kaikissa vaiheissa, jolloin tarkastelu voidaan rajoittaa ainoastaan siihen kuinka hyvin he persoonallisuustyypeiltään soveltuvat suorittamaan tiettyjä kehityksen vaiheita.

3.3.1 Vaatimusmäärittely

Perinteisesti ohjelmistotuotanto alkaa vaatimusmäärittelyllä [4, 20]. Tämän vaiheen tavoitteena on selvittää mitä vaatimuksia tuotteella eli valmiilla ohjelmalla on. Vaatimuksilla voidaan tarkoittaa toiminnallisia vaatimuksia eli mitä ohjelma tekee tai mitä sillä pystytään tekemään, tai ei-toiminnallisia vaatimuksia eli esimerkiksi millä kielellä ohjelma kirjoitetaan, kuinka käytettävä tai tehokas sen tulee olla ja niin edelleen.

Vaatimusmäärittely tehdään usein asiakkaan kanssa, jolla on etukäteen jonkinlainen kokonaiskuva valmiista ohjelmasta. Kehittäjien tehtävänä on haastatella asiakasta saadakseen selville millaisesta ohjelmasta on kyse ja mitä sen tulisi tehdä. Vaatimusmäärittelyssä tulee kartoittaa ensinnäkin keitä ovat ohjelman käyttäjät ja millaisia käyttötilanteita käyttäjillä on ohjelmaan liittyen. Yhteistyössä asiakkaan kanssa saadaan lopuksi lista vaatimuksia toteutettavalle ohjelmalle, jotka seuraavaksi analysoidaan. Analysoinnilla tutkitaan ovatko vaatimukset tarpeeksi kattavat eli määrittävätkö ne asiakkaan haluaman tuotteen sekä tarkistetaan, että vaatimukset eivät ole ristiriidassa toistensa kanssa. Jos vaatimukset vaikuttavat kattavilta ja ristiriidattomilta, niin lopuksi vielä varmistetaan asiakkaalta, että vaatimukset ovat hänen mieleensä ja edustavat sitä mielikuvaa tuotteesta, joka hänellä on. Lopuksi vaatimukset dokumentoidaan, jotta kehittäjät tietävät ja muistavat jatkossakin mitä heidän tulee toteuttaa sekä testata.

Capretz ja Ahmed ovat tutkineet millainen kehittäjä soveltuu parhaiten järjestelmäanalyysiin, joka on sidoksissa vaatimusmäärittelyyn [9]. Järjestelmä-

analyysiin tarvitaan kehittäjiä, jotka ymmärtävät asiakkaan tarpeet ja käsittävät mitkä ovat ohjelman keskeisimmät toiminnot. Järjestelmäanalyysi vaatii paljon kommunikointia asiakkaan kanssa, joten ekstrovertit (E) soveltuvat tähän hyvin, koska he ovat parempia puhumaan ja saamaan asiakkaasta vastauksia irti toisin kuin introvertit (I), jotka eivät ole yhtä hyviä ilmaisemaan asioita selkeästi. Kehittäjän tulee myös osata asettumaan käyttäjän rooliin ja miettimään käyttäjän tarpeita. Tähän tarpeeseen on tuntevat (F) ihmiset hyviä. Vaatimusmäärittelyssä keskustellaan samaan tapaan asiakkaan kanssa, joissa tämän kaltaiset henkilöt ovat tarpeellisia. Optimaalisinta olisi määrätä vaatimusmäärittelyyn kehittäjiä, jotka ovat sekä ekstrovertteja että tuntevia (EF).

Gorlan ja Lamin mielipiteet eroavat edellä mainitusta [14]. Heidän mukaan ajattelevat (T) kehittäjät ovat tuntevia (F) parempia tässä työssä, jos kyse on pienestä tiimistä. Pienemmissä tiimeissä vaatimusten määrittelyn lisäksi kehittäjä voi joutua tekemään paljon muutakin, kuten suunnittelua ja ohjelmointia, jolloin analyyttinen ajattelutapa on parempi.

3.3.2 Suunnittelu

Suunnittelu on prosessi, jossa määritellään ohjelman arkkitehtuuri, komponentit, rajapinnat ja muut ohjelmaan liittyvät asiat [4]. Suunnittelussa hahmotellaan vaatimusten perusteella mistä komponenteista ja pienemmistä osasista järjestelmä koostuu ja mitkä ovat niiden tehtävät. Suunnittelussa mietitään miten järjestelmän eri osat ovat vuorovaikutuksessa toistensa kanssa, millaiset rajapinnat niillä on ja mitä palveluita ne tarvitsevat muilta järjestelmän osilta. Tarkoituksena on luoda ohjelmasta malli, joka toteuttaa kirjatut vaatimukset, ja jonka voi muuttaa ohjelmakoodiksi.

Suunnittelu on luovaa ja tutkivaa työtä, jossa suunnittelija miettii mitkä ovat järjestelmän avainkomponentit, ja hahmottelee parhaita ratkaisuja erilaisten käyttötilanteiden avulla. Suunnittelijalla tulisi olla kokonaiskuva järjestelmästä ja kyky erotella relevantit asiat annetusta datasta, kuten vaatimuksista. Tämä vaatii intuitiota. Suunnittelijoiden tulisi siis olla intuitiivisia (N), koska tällaiset henkilöt ovat luovia ja innostuvat uusien asioiden luomisesta ja kokeilevat erilaisista asioista tapahtuvien mahdollisuuksien yhdistelyä. Hyvän tuloksen saamiseksi tarvitaan ongelmanratkaisutaitoja, joita ajattelevilla (T) on. On suositeltavaa, että suunnittelija on sekä intuitiivinen ja ajatteleva (NT) [9].

3.3.3 Toteutus

Kun suunnittelu on ohi, siirrytään toteutukseen [4]. Suunnitteluvaiheessa ohjelmasta ollaan luotu jonkinlainen malli siitä millaisista komponenteista

se koostuu ja mitkä ovat eri komponenttien tehtävät, voidaan ryhtyä toteuttamaan tätä mallia. Käytännössä tämä tarkoittaa mallin muuntamista koodiksi ja siten konkreettiseksi ohjelmaksi.

Toteutus on suurimmaksi osaksi ohjelmointia, johon liittyy suunnittelua ja testausta. Ohjelmoijan tulee tunnistaa ohjelman osille tärkeät muuttujat, rakenteet ja tietorakenteet. Ohjelmoijan tulee luonnollisesti hallita ohjelmointikieli, jolla ohjelma kirjoitetaan. Siinä missä ohjelmalle tehty malli voi olla hyvinkin korkeatasoinen, koodin kirjoittaminen vaatii paneutumista yksityiskohtiin ja mallin sekä vaatimusten muuntaminen suoraviivaiseksi koodiksi vaatii loogista ajattelutapaa. Ajatteleva (T) henkilö soveltuu loogiseen ajatteluun paremmin kuin tunteva (F). Lisäksi yksityiskohtiin ja faktatietoihin paneutuminen on tosiasiallisen (S) ihmisen aluetta enemmän kuin intuitiivisen (N). Vaikka aiemmin on todettu kehityksen vaativan ryhmätyötä ja kommunikointia, Capretz ja Ahmed näkevät ohjelmoinnin teknisenä työnä, joka loppupeleissä vaatii itsenäistä työskentelyä ja keskittymiskykyä enemmän kuin ryhmätyöskentelyä. Täten introvertit (I) sopivat ohjelmoijiksi ekstrovertteja (E) paremmin. Ihanteellinen yhdistelmä ohjelmoijalle olisi, että hän on introvertti, tosiasiallinen ja ajatteleva (IST). Tätä väitettä tukee se, että monien tutkimusten mukaan useimmat ohjelmistokehittäjät ovat persoonallisuustyyppiä ISTJ [8, 9].

Gorlan ja Lamin havainnot poikkeavat ideaalista ohjelmoijasta [14], kun kyse on pienistä tiimeistä. Tiimin suorituskyky on parempi, jos ohjelmoijat ovat ekstrovertteja (E). Pienissä tiimeissä ohjelmoijien pitää kommunikoida useiden osapuolten kanssa, kuten esimerkiksi muiden tiimin jäsenten kanssa. Lisäksi pienissä tiimeissä ohjelmoija voi joutua tekemään monia muitakin tehtäviä kuin vain ohjelmointia, kuten ohjelman suunnittelua ja kommunikointia asiakkaan tai käyttäjien kanssa. Tällöin ekstroverttius on eduksi.

3.3.4 Testaus

Testausvaiheessa on tarkoituksena tehdä testejä, joilla varmistetaan ohjelman toimivuus [4]. Vaatimukset ovat ensisijaisia testauskohteita eli testien avulla tarkistetaan, että ohjelmalle asetetut vaatimukset, jotka ovat toteutettu, toimivat moitteettomasti. Testauksella pyritään myös löytämään ohjelmasta mahdollisimman paljon virheitä tekemällä testitilanteita, joissa yritetään tehdä jotain luvatonta toimintaa, jota ohjelman ei tulisi sallia. Esimerkiksi, jos ohjelma on yksinkertainen laskin, joka pyytää syöttämään yhteenlaskua varten kaksi lukua ja käyttäjä syöttääkin kirjaimista koostuvan merkkijonon, ohjelman tulee osata varautua tähän ilman, että se hajoaa.

Testausta on monenlaista, joita ovat muun muassa yksikkötestaus, integrointitestaus ja järjestelmätestaus [9]. Yksikkötestauksessa testataan ohjel-

man sisältämiä komponentteja, luokkia ja metodeita yksilöllisesti. Integraatiotestauksessa varmistetaan, että nämä komponentit, luokat ja metodit toimivat niiden ollessa keskinäisessä vuorovaikutuksessa toistensa kanssa, pyytäen ja tarjoten palveluita toisilleen. Järjestelmätestauksessa ohjelmaa testataan sen rajapinnan eli useimmiten käyttöliittymän kautta. Järjestelmätestaus liittyy vahvasti vaatimuksiin, jotka määrittävät, että järjestelmällä pitäisi pystyä tekemään jotain ja järjestelmän tulisi vastata tähän toimintoon vaatimusten mukaisesti.

Testaukseen on monenlaisia strategioita, jotka noudattavat järjestelmällisiä lähestymistapoja. Testaus vaatii paneutumista yksityiskohtiin, jotta testaaja varmasti tarkistaa, että pienimmätkin ohjelman osat toimivat oikein. Tarkkuus ja järjestelmällisyys ovat haluttavia ominaisuuksia testaajalle, joita tosiasialliset (S) ja järjestelmälliset (J) henkilöt edustavat. Menestyvät testaajat olisivat sekä tosiasiallisia että järjestelmällisiä (SJ) [9].

3.3.5 Ylläpito

Ennen ylläpitoa ohjelma on jo julkaistu kohderyhmän käyttöön valmiina tuotteena, joka toteuttaa vaatimukset [4]. Ylläpidon tarkoituksena on parannella tai muokata ohjelmaa, jolla tarkoitetaan lähinnä ohjelmointivirheiden korjaamista ja optimointia. Ylläpidossa saatetaan lisätä myös aivan uusia toiminnallisuuksia ohjelmaan. Kaiken tämän tarkoituksena on pidentää ohjelman käyttöikää ja varmistaa käyttäjien tyytyväisyys.

Ohjelman ylläpitäminen ja parantelu sopii tosiasiallisille (S) kehittäjille, koska he suosivat tuttujen tehtävien tekoa, jotka eivät vaadi uusien asioiden kokeilua. Intuitiiviset (N) sen sijaan haluavat kokeilla jotain uutta ja luultavasti tylsistyisivät ylläpitoon, jossa tehdään jonkinlaisia parannuksia ja pieniä ohjelmointivirheiden korjauksia. Tosiasialliset suosivat töitä, joissa aiemmin opitun tiedon käyttö on riittävää tehtävän suorittamiseen sen sijaan, että tulisi keksiä uusia ratkaisuja. He myös osaavat keskittyä yksityiskohtiin ja haluavat tietää miten asiat toimivat, joten heillä olisi hyvä kokonaiskuva järjestelmästä, jota ylläpitäminen vaatii. Spontaanit (P) sopeutuvat muutoksiin, jolloin heitä ei haittaisi järjestelmän jatkuva muokkaaminen. Siten tosiasialliset ja spontaanit kehittäjät ovat hyviä ylläpitäjiä (SP) [9].

Edellä mainitut vaiheet ovat kaikki oleellisia ketterissä menetelmissä. Ketterät menetelmät perustuvat iteratiiviseen ja inkrementaaliseen kehitysmalliin, jonka ideana on tehdä ohjelmisto sykleissä, joissa ohjelmistosta tuotetaan vain osa. Tämä osa tuotetaan valmiiksi niin, että sykli sisältää vaatimusmäärittelyn, suunnittelun, toteutuksen, testauksen ja ylläpidon. Käytännössä tämä tarkoittaa, että yhden syklin aikana sama tiimi työstää yhdessä näitä vaiheita eli tiimin jäseniä ei eroteta ottamaan vastuuta vain tie-

tystä vaiheesta. Kuten on todettu, monipuolinen tiimi eri persoonallisuuksilla täytettynä tuottaa paljon eri näkökulmia ja ratkaisuja. Persoonallisuutta katsottaessa yksittäinen henkilö saattaa olla toista henkilöä parempi tietyssä kehityksen vaiheessa, joten ketterissä menetelmissä on hyvä olla tiimi, jonka jäsenet ovat erilaisia luonteiltaan.

3.4 Huonon ohjelmistokehittäjän piirteet

Ohjelmistoalalla on väistämättä huonojakin ohjelmistokehittäjiä, jotka tekevät yhteistyöstä epämiellyttävää. Hallin ja kumppaneiden haastatteluissa kysyttiin millaisia ovat huonot kehittäjät [15]. Tuloksena oli, että huono ohjelmistokehittäjä on tekniseltä osaamiseltaan heikko ja hänellä on huonot kommunikointitaidot. Huonoilla kommunikointitaidoilla haastateltavat tarkoittivat seuraavia asioita:

1. Ei raportoi ongelmista.
2. Haluton kuuntelemaan muiden ideoita.
3. Itsepäinen.
4. Ei kunnioita ylempiarvoisia.

Kun tähän lisätään luvun 3.2 hyvien piirteiden vastakohtia, on huono kehittäjä huumorintajuton, ei ota työstään vastuuta ja hänellä on vaikeuksia omaksua uusia työtapoja tai rooleja. Tekniseltä osaamiseltaan kehittäjän ongelmanratkaisutaidot ovat heikot eikä hänellä ole kykyä ajatella asioita abstraktisti.

4 Ryhmätyötaitojen parantaminen

Hyvät ryhmätyö- ja kommunikointitaidot eivät ole itsestäänselvyys. Itse asiassa monesti on todettu, että tietojenkäsittelytieteen opiskelijoiden ryhmätyö- ja kommunikointitaidot ovat puutteelliset johtuen siitä, että opetuksessa näitä taitoja ei ole huomioitu tarpeeksi [11, 16, 22]. Ryhmätyöskentelyn merkitys on huomattu ja opetusohjelmiin on lisätty enemmän ryhmä- ja projektitöitä, mutta varsinaisia kommunikointitaitoja ei opiskelijoille opeteta, jotka auttaisivat heitä menestymään paremmin näissä töissä. Tämän seurauksena opiskelijat eivät välttämättä osaa toimia ryhmässä ja kokemus jää epämiellyttäväksi, joka vaikuttaa negatiivisesti asenteisiin ryhmätyöskentelyä kohtaan [17]. Luvussa 3.4 todettiin huonon ohjelmistokehittäjän piirteitä. Henkilö, joka täyttää osankin kyseisistä piirteistä, ei ole houkutteleva ryhmän jäsen. Kuitenkin haastattelujen perusteella tällaisia ihmisiä löytyy ohjelmistoalalta.

Puutteelliset ja jopa huonot ryhmätyötaidot huomioon ottaen on tarve menetelmille, joilla parantaa yksilön ryhmätyötaitoja. Seuraavaksi tarkastellaan erilaisia menetelmiä, joilla kehittää yksilöiden kommunikointi- ja ryhmätyötaitoja, jotta he pystyisivät toimimaan tehokkaasti ryhmänä. Menetelmät on kehitetty suurimmaksi osaksi opetuskäyttöön opiskelijoille, jotka eivät ole vielä työelämässä.

4.1 Kevyet menetelmät

4.1.1 Vuorovaikutteinen luokkahuone

4.1.2 Tarkkailija-kommunikoija-rakentaja

4.2 Suuret menetelmät

4.2.1 Project-based learning

4.2.2 Rocking the boat

5 Yhteenveto

Lähteet

[1] *Agile Manifesto*, 2001. `http://agilemanifesto.org/`.

[2] *What is Scrum?*, 2009. `http://www.scrum.org/Resources/What-is-Scrum`.

[3] *The Scrum Guide*, 2011. `http://www.scrum.org/Portals/0/Documents/Scrum%20Guides/Scrum%20Guide%20-%20FI.pdf#zoom=100`.

[4] Abran, Alain, Pierre Bourque, Robert Dupuis, James W. Moore ja Leonard L. Tripp: *Guide to the Software Engineering Body of Knowledge - SWEBOK*. IEEE Press, Piscataway, NJ, USA, 2004, ISBN 0769510000.

[5] Acuña, Silvia T., Marta N. Gómez ja Juan de Lara: *Empirical study of how personality, team processes and task characteristics relate to satisfaction and software quality*. Teoksessa *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, ESEM '08, sivut 291–293, New York, NY, USA, 2008. ACM, ISBN 978-1-59593-971-5. `http://doi.acm.org/10.1145/1414004.1414056`.

[6] Beck, Kent ja Cynthia Andres: *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004, ISBN 0321278658.

[7] Begel, Andrew ja Nachiappan Nagappan: *Pair programming: what's in it for me?* Teoksessa *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, ESEM '08, sivut 120–128, New York, NY, USA, 2008. ACM, ISBN 978-1-59593-971-5. http://doi.acm.org/10.1145/1414004.1414026.

[8] Capretz, Luiz Fernando: *Personality types in software engineering.* Int. J. Hum.-Comput. Stud., 58(2):207–214, helmikuu 2003, ISSN 1071-5819. http://dx.doi.org/10.1016/S1071-5819(02)00137-4.

[9] Capretz, Luiz Fernando ja Faheem Ahmed: *Making Sense of Software Development and Personality Types.* IT Professional, 12(1):6–13, tammikuu 2010, ISSN 1520-9202. http://dx.doi.org/10.1109/MITP.2010.33.

[10] Cockburn, Alistair: *Using Both Incremental and Iterative Development.* 2008. http://www.crosstalkonline.org/storage/issue-archives/2008/200805/200805-Cockburn.pdf.

[11] Cushing, Judy, Kate Cunningham ja George Freeman: *Towards best practices in software teamwork.* J. Comput. Sci. Coll., 19(2):72–81, joulukuu 2003, ISSN 1937-4771. http://dl.acm.org/citation.cfm?id=948785.948797.

[12] Da Cunha, Alessandra Devito ja David Greathead: *Does personality matter?: an analysis of code-review ability.* Commun. ACM, 50(5):109–112, toukokuu 2007, ISSN 0001-0782. http://doi.acm.org/10.1145/1230819.1241672.

[13] Deemer, Pete ja Gabrielle Benefield: *The Scrum Primer. An Introduction to Agile Project Management with Scrum.* 2007. http://www.rallydev.com/documents/scrumprimer.pdf.

[14] Gorla, Narasimhaiah ja Yan Wah Lam: *Who should work with whom?: building effective software project teams.* Commun. ACM, 47(6):79–82, kesäkuu 2004, ISSN 0001-0782. http://doi.acm.org/10.1145/990680.990684.

[15] Hall, Tracy, David Wilson, Austen Rainer ja Dorota Jagielska: *Communication: the neglected technical skill?* Teoksessa *Proceedings of the 2007 ACM SIGMIS CPR conference on Computer personnel research: The global information technology workforce*, SIGMIS CPR '07, sivut 196–202, New York, NY, USA, 2007. ACM, ISBN 978-1-59593-641-7. http://doi.acm.org/10.1145/1235000.1235043.

[16] Jun, Huang: *Improving undergraduates' teamwork skills by adapting project-based learning methodology.* Teoksessa *Computer Science and*

Education (ICCSE), 2010 5th International Conference on, sivut 652 –655, aug. 2010.

[17] Lingard, R. ja E. Berry: *Teaching teamwork skills in software engineering based on an understanding of factors affecting group performance.* Teoksessa *Frontiers in Education, 2002. FIE 2002. 32nd Annual*, nide 3, sivut S3G–1 – S3G–6 vol.3, nov. 2002.

[18] Pieterse, Vreda, Lisa Thompson, Linda Marshall ja Dina M. Venter: *Participation patterns in student teams.* Teoksessa *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, SIGCSE '12, sivut 265–270, New York, NY, USA, 2012. ACM, ISBN 978-1-4503-1098-7. `http://doi.acm.org/10.1145/2157136.2157218`.

[19] Shore, James ja Shane Warden: *The art of agile development*, luku 5. O'Reilly, first painos, 2007, ISBN 9780596527679.

[20] Sommerville, Ian: *Integrated Requirements Engineering: A Tutorial.* IEEE Softw., 22(1):16–23, tammikuu 2005, ISSN 0740-7459. `http://dx.doi.org/10.1109/MS.2005.13`.

[21] Sutherland, Jeff: *Scrum Handbook*, 2010. `http://jeffsutherland.com/scrumhandbook.pdf`.

[22] Waite, William M., Michele H. Jackson, Amer Diwan ja Paul M. Leonardi: *Student culture vs group work in computer science.* SIGCSE Bull., 36(1):12–16, maaliskuu 2004, ISSN 0097-8418. `http://doi.acm.org/10.1145/1028174.971308`.

[23] Zarb, Mark: *Understanding communication within pair programming.* Teoksessa *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*, SPLASH '12, sivut 53–56, New York, NY, USA, 2012. ACM, ISBN 978-1-4503-1563-0. `http://doi.acm.org/10.1145/2384716.2384738`.