Esimerkkiotsikko

Kenny Heinonen

Aine
Helsingin Yliopisto
Tietojenkäsittelytieteen laitos

Helsinki, 9. helmikuuta 2013

Sisältö

1 Johdanto

2 Ryhmätyöskentelyn merkitys ketterissä menetelmissä

Ohjelmistoalalla tarvitaan monenlaisia teknisiä taitoja, jotta etenkin suuremmissa projekteissa saadaan toteutettua kaikki kehityksen vaiheet kattavasti. Nämä ohjelmistokehityksen vaiheet voidaan jakaa karkeasti vaatimusmäärittelyyn, suunnitteluun, toteutukseen, testaukseen ja ylläpitoon [4]. 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. Useat prosessimallit jopa sanelevat miten ryhmätyöskentely tapahtuu, jotta kehitys sujuisi luontevammin ja tuotteliaammin.

Ryhmätyöskentelyä voi tehdä monin tavoin. On esimerkiksi tilanteita, joissa ryhmät sijaitsevat samoissa oloissa, mutta työskentelevät silti täysin erillään toisistaan. Tämän seurauksena kommunikointi kärsii ja voi olla epäsäännöllistä. 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 näistä asioista ilman, että kukaan ulkopuolinen tulee kertomaan heille mitä tehdä. 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. Ketterissä menetelmissä toimivan ohjelmiston tuottaminen säännöllisin väliajoin on tärkeää ja sillä halutaan pitää asiakas tyytyväisenä. Tuotteen kehitys siis tehdään iteratiivisesti ja inkrementaalisesti. Tämä tarkoittaa sitä, että tuotetta kehitetään lyhyissä kehityssykleissä pala kerrallaan, saaden jatkuvaa palautetta asiakkaalta. Seuraavissa osioissa tarkastellaan muutamia ketteriä prosessimalleja nähdäksemme miten ryhmätyöskentely ja kommunikointi otetaan niissä huomioon.

2.1 Extreme Programming

Extreme Programming (XP) on ketterä menetelmä, jonka tunnettu ohjelmistokehittäjä Kent Beck on luonut. XP on iteratiivinen ja inkrementaalinen prosessimalli, jonka tarkoituksena on tuottaa asiakkaalle mahdollisimman paljon arvoa mahdollisimman tehokkaasti. XP:ssä kehitys tapahtuu lyhyissä

iteraatioissa, jolloin vaatimusten muutoksiin on helpompi varautua, kun tavoitteet on asetettu lähitulevaisuuteen. Yhteistyö ja kommunikointi on XP:n tärkeimpiä piirteitä. Kommunikointia tapahtuu säännöllisesti asiakkaan kanssa, jotta kehittäjät voivat saada arvokasta palautetta tekemästään työstä ja siten tehdä parannuksia. Asiakas pysyy tyytyväisenä, kun saa olla jatkuvasti osana projektia ja näkee työn edistymistä. Myös ryhmän sisäinen kommunikointi on 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.

2.1.1 Pariohjelmointi

Pariohjelmoinnissa kaksi henkilöä ohjelmoivat pareittain, jakaen saman tietokoneen. 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. [6]

Pariohjelmointi korostaa yhteistyötä ja kommunikointia. Tästä on monia hyötyjä, jotka tekevät hyvää sekä ryhmän jäsenelle, ryhmälle että projektille [3]. 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 usein 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 myös se, 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 se, 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 [8].

2.2 Scrum

Scrum on prosessimalli, joka painottaa tiimien yhteistyötä projektin kehityksessä [2]. Scrum, kuten XP, on myös iteratiivinen ja inkrementaalinen eli kehitys tapahtuu lyhyissä 1-4 viikon sykleissä, *”Sprinteissä”*, rakentaen 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.

2.2.1 Projektin elinkaari

Tämä osio vielä paloihin?

Projektin alussa luodaan lista toiminnallisuuksista, joita asiakas haluaa toteutettavan. Lista on niin sanottu *tuotteen kehitysjono* (”product backlog”). Kehitysjonon sisältämät asiat priorisoidaan siten, että tärkeimmät toiminnallisuudet ovat listan kärjessä. Yksittäinen tiimi valitsee Sprinttiin kehitysjonon kärjestä sen verran toteutettavia asioita, jonka verran he uskovat pystyvänsä toteuttamaan Sprintin loppuun mennessä. Tavoitteena on toteuttaa nämä vaatimukset niin, että tuote on ”valmis” eli toimii näiden vaatimusten osalta

moitteettomasti, vaikka tuote itsessään ei sisältäisikään vielä kaikkia niitä toiminnallisuuksia, joita asiakas haluaa. Sprintin lopuksi on *Sprintin katselmus* ("Sprint review"), jossa tiimi näyttää asiakkaalle aikaansaannoksiaan. Näin asiakas näkee projektin edistymistä säännöllisesti ja voi samalla antaa mielipiteensä tiimin tekemästä työstä. Asiakkaan antama palaute otetaan huomioon ja asiakkaan vaatimat muutokset kirjataan kehitysjonoon. [7]

2.2.2 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 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. [7]

2.2.3 Sprintin retrospektiivi

Sprintin lopuksi järjestetään *Sprintin retrospektiivi* ("Sprint retrospective"), jossa tiimi keskustelee kuinka heidän oma työprosessinsa sujui Sprintin aikana. 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. [5]

2.3 Lean/FDD?

3 Persoonallisuuden vaikutus

3.1 Hyvän ohjelmistokehittäjän piirteet

3.2 Persoonallisuustyypit eri kehityksen vaiheissa

3.2.1 Vaatimusmäärittely

3.2.2 Suunnittelu

3.2.3 Toteutus

3.2.4 Testaus

3.2.5 Ylläpito

4 Ryhmätyötaitojen parantaminen

5 Lähteet

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

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

[3] 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.

[4] 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.

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

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

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

[8] 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.