Kalle Ilves: Scrumban menetelmän käyttö ketterässä ohjelmistokehityksessä

Scrum ja Kanban ovat molemmat ketterän ohjelmistokehityksen menetelmiä, joiden samansuuntaiset Lean-ajattelutavat ja empirisyyden ja optimisaation periaate Kaizen mahdollistavat ominaisuukisien yhdistelyn. Tällaista ohjelmistotuotannon menetelmää kutsutaan nimellä Scrumban. Scrumban menetelmä on havaittu erityisen hyväksi projekteissa, jotka ovat tavallistakin vaikeammin ennustettavia ja joissa vaatimukset saattavat muuttua hyvin nopeasti.
Scrumin ongelma tällaisissa projekteissa on sprinttien pituus ja joustamattomuus, Scrumbanin avulla ongelma voidaan poistaa luopumalla sprinttiajattelusta ja ottamalla jatkuva suunnittelu osaksi kehitysprosessia.
Kanbanissa taas rajoitteiden poissaolo johtaa helposti huonoon kommunikaatioon ja laskevaan tehokkuuteen. Näitä ongelmia on mahdollista korjata Scrumista lainatuilla päivätapaamisilla ja retrospektiiveillä.
Yhtä oikeaa ohjelmistokehityksen menetelmää ei ole olemassa. Jokainen projekti on erilainen ja näin ollen myös parhaat menetelmät ohjelmiston toteuttamiseen ovat erilaisia. Menetelmänä Scrumban ei ole kovinkaan tarkasti määritelty, vaan siihen yhdistellään elementtejä Scrumista ja Kanbanista tarpeen mukaan.

Martin Fowler: Is Design Dead?

Fowler esittelee artikkelissaan kaksi erilaista tapaa suunnitella ohjelmistoja – suunniteltu ja evolutionaarinen. Molemmista tavoista löytyy haittansa, etenkin huonosti toteutettuna.
Suunniteltu tapa nojaa vahvaan ja perusteelliseen ennakkosuunnitteluun. Tämän suunnittelun tekevät ohjelmistoarkkitehtuurin ammattilaiset, jotka ojentavat valmiit suunnitelmansa ohjelmoijien noudetettavaksi. Ongelmiksi muodostuvat projektin aikana muuttuvat vaatimukset, joita ei ole osattu ottaa huomioon suunnitteluvaiheessa. Lisäksi arkkitehtuuriin erikoistuneet henkilöt ovat saattaneet vieraantua varsinaisesta ohjelmoimisesta siinä määrin, etteivät kykene tekemään hyviä suunnitelmia uusimpia ohjelmointitekniikoita silmällä pitäen.
Evolutionaarinessa tavassa suunnittelua tehdään jatkuvasti projektin edetessä. Huonosti toteutettuna tämä johtaa helposti tilanteeseen, jossa suunnittelua ei oikeastaan edes ole. On vain tilanteen mukaan tehtyjä päätöksiä, jotka johtavat sekavaan ja kankeaan koodin, jonka muuttaminen käy ajan kuluessa yhä vaikeammaksi.
Molemmat tavat kärsivät samasta ongelmasta. Projektin alkuvaiheessa tehty helposti korjattavissa oleva virhe on myöhemmässä vaiheessa eksponentiaalisesti kalliimpi ja työläämpi korjattava. XP pyrkii madaltamaan kyseistä eksponentiaalista käyrää mm. kattavien testien ja jatkuvan integraation avulla.
XP pyrkii myös mahdollisimman yksinkertaiseen toteutukseen. Ohjelmaan ei tehdä mitään sellaista, mitä ei tarvita juuri nyt työn alla olevia ominaisuuksia varten. Ominaisuudet toteutetaan mahdollisimman yksinkertaisella tavalla, vaikka ehkä tiedettäisiinkin ettei tapa ole optimaalinen ohjelman tulevaisuutta ajatellen. Kun ohjelmaan myöhemmin lisätään uusia ominaisuuksia, refaktoroidaan vanhaa koodia ja lisätään patterneja tarpeen mukaan.
Suunnittelua tapahtuu XP:ssä koko projektin ajan. Suunnittelusta vastaavat samat henkilöt ketkä kirjoittavat koodia. Projektissa saattaa tosin olla mukana henkilö/henkilöitä, ketkä pitävät huolta ohjelmiston suunnitelmallisuudesta/arkkitehtuurista. Näiden henkilön vastuulla on tutkia koodia läpi projektin ja jos joku osa ohjelmasta alkaa menettää teknistä laatuaan ja muuttua sekavaksi, huolehtivat nämä henkilöt siitä, että kyseinen osa refaktoroidaan siistimpään kuntoon. Koska suunnitelmat muuttuvat XP:ssä jatkuvasti,on oleellisen tärkeää, että tiimin jäsenet kommunikoivat keskenään. Kommunikointiin ei ole yhtä oikeaa menetelmää, vaan jokaisen tiimin on löydettävä oma tapansa, olkoon se sitten diagrammit, käyrät, fläppitaulu tai keskinäinen morsetus.

Eero-Veikko Laine: Johtaminen perinteisissä ja ketterissä ohjelmistoprojekteissa

Ohjelmistoja tuotetaan usein projekteina, jotka ovat luonteeltaan monimutkaisia ja vaikeasti ennustettavia jatkuvasti muuttuvan teknologian ansiosta. Korkean epäonnistumisprosentin syitä on etsitty haasteellisuuden lisäksi myös projektin johdosta.
Perinteisesti projektin johdossa on toiminut projektipäällikkö, jonka kyky johtaa tiimiä on ollut projektin onnistumisen kannalta hyvin isossa roolissa. Projektipäällikön tehtävänä on ollut hallita projektia, varautua ongelmiin etukäteen ja joissain tapauksissa jopa autoritäärisellä ja kontrolloivalla otteella viedä tiimi kohti tavoitetta.
Ketterissä menetelmissä vastuu projektin johtamisesta on hajautettu. Projektilla on usein jonkinlainen johtaja, Scrum mestari tai vastaava, mutta kyseinen rooli ei ole kovin tarkasti määritelty. Tehtävänkuvaan kuuluu enemmänkin toimia mahdollistajana ja esteiden raivaajana, mutta varsinaisesta päätöksenteosta ja johtamisesta vastaa koko tiimi yhdessä. Jos jollain tiimin jäsenellä on tietyssä asiassa enemmän osaamista, ideaalitilanteessa tämä henkilö ottaa isomman roolin kyseisen asian johtamisessa.
Vaikka ketterissä menetelmissä johtamista ja vastuuta pyritään hajauttamaan kaikille tiimin jäsenille, on usein etenkin projektin alkuvaiheessa tärkeää, että tiimillä on apunaan kokenut henkilö joka mentoroi, valmentaa ja ohjaa tiimin toimintaa. Tällaisen henkilön toimenkuvaan saattaa kuulua myös koordinointi tiimin ja asiakkaan välillä, puolesta puhuminen yrityksen johdolle, sekä tiimin jäsenten motivointi, arviointi ja tarvittaessa jopa poistaminen projektista.

Kenny Heinonen: Ryhmätyö ohjelmistokehityksessä

Moderneissa ohjelmistotuotannon menetelmissä, kuten Scrumissa, on tärkeää osata työskennellä ryhmässä. Modernit menetelmät korostavat sosiaalisten- ja vuorovaikutustaitojen merkitystä ja halu työskennellä ryhmässä on isossa roolissa.
Hyvälle ryhmätyöskentelijälle ominaisia piirteitä ovat mm. hyvät kommunikointitaidot, älykkyys, joustavuus, huuomorintaju ja yleinen miellyttävyys ihmisenä. Huonoja ominaisuuksia ovat mm. itsepäisyys, taipumus tehdä kaikki itse ja kunnioituksen puute sekä haluttomuus kuunnella muita. Kunhan ryhmätyökykyjä löytyy, erilaisille persoonille on tarvetta ohjelmistokehityksen eri vaiheissa.
Ryhmätyökyvyt eivät ilmaannu ihmiselle itsestään vaan niitä täytyy kehittää. Ohjelmistoalan opetuksessa ei perinteisesti ole tätä seikkaa juurikaan huomioitu, mutta tähän on onneksi tullut muutos. Kykyjen kehittämisen tueksi on kehitetty useita menetelmiä, joissa ihmiset pääsevät kokeilemaan ryhmätyöskentelyä käytännössä. Menetelmien ei välttämättä tarvitse liittyä ohjelmoimiseen, kunhan ryhmällä on jokin yhteinen tavoite. Toiset ihmiset ovat ryhmälle erityisen edullisia, kun taas toiset jopa haitallisia. Erityisen opettavaista on harjoitella sellaisessa ryhmässä, jossa esiintyy näitä molempia ihmistyyppejä.
Teknisen osaamisen tärkeys on itsestäänselvyys ohjelmistoa kehitettäessä, mutta puutteelliset ryhmätyötaidot voivat johtaa tilanteeseen, missä muiden ryhmän jäsenten tuottavuus laskee. Parhaimmillaan ryhmän jäsenet auttavat toisiaan entistä parempiin suorituksiin ja jakavat osaamistaan toinen toiselle.

Tero Huomo: Ohjelmistoarkkitehtuurin sisällyttäminen ketteriin ohjelmistotuotantomenetelmiin

Arkkitehtuurilähtöiset ohjelmistotuotantomenetelmät nähdään usein vastakohtana ketterille menetelmille. Äärimmilleen vietynä ketterät menetelmät luottavat siihen, että arkkitehtuuri rakentuu ajan myötä tarpeita vastaavaksi, mutta on olemassa useita tapoja liittää arkkitehtuurista suunnittelua myös ketteriin menetelmiin.
Yksi tällainen tapa on sprint 0, joka on ensimmäistä sprinttiä edeltävä jakso, jossa ohjelmoijat itse suunnittelevat ohjelman rakenteen. Tämän rakenteen ei ole tarkoitus olla lopullinen, ainoastaan suuntaa antava. Toinen tapa on eriyttää arkkitehtuuri omaksi prosessikseen, joka synkronoidaan ohjelmiston kehityksen kanssa. Arkkitehdit miettivät kokonaisuutta ja suunnittelevat sopivan rakenteen seuraavaa työvaihetta varten. Tätä tapaa käytettäessä osa kehittäjistä toimii tyypillisesti sekä arkkitehtinä, että ohjelmoijana.
Vaihtoehtona on myös käyttää suunnittelupiikkejä, jotka on ajallisesti rajattuja ja joihin osallistuu tyypillisesti vain osa tiimin jäsenistä. Tämän laajennettu versio on arkkitehtuurijakso, missä koko tiimi pohtii yhdessä ohjelman rakennetta.
Yhtä oikeaa tapaa toimia ei ole, vaan valitut menetelmät riippuvat aina tekijöistä ja heidän taitotasostaan, toimintaympäristöstä ja projektista itsestään. Edellä mainitut tavat liittää suunnittelua mukaan ketterään ohjelmistokehitykseen saattavat rikkoa joitain ketterän kehityksen periaatteita, mutta monet ohjelmistoyritykset kokevat tällaisen hybridimallin hyväksi.

Lauri Suomalainen: Ohjelmistotuotantomenetelmien kehittyminen 1950-luvulta nykypäivään

Ohjelmistotuotannon menetelmät ovat kehittyneet vuosien saatossa huomattavasti. Virheistä on opittu, mutta myös toimintaympäristön ja tekniikan kehittyminen on luonut tarpeen uusille menetelmille.
Ohjelmointi oli alkuaikoinaan 1940-luvun puolivälistä 1950-luvun lopulle hyvin yksinkertaista johtuen laitteiston asettamista rajotteista. Ohjelmointi keskittyi lähinnä sen maksimoimiseen, mitä tämän ajan tekniikalla oli mahdollista saavuttaa.
Tekniikan kehittyessä 1960-luvulla syntyi ohjelmointiongelmien monimutkaistumisen myötä myös tarve uusille tuotantomenetelmille. Tähän tarpeeseen vastasivat 1970-luvulla niin kutsutut perinteiset ohjelmointimenetelmät johon kuuluu insinööritieteistä vaikutuksia saanut vesiputosmalli. Malli muistuttaa rakennussuunnitteluprosessia missä suunnitteluvaihe on hyvin perusteellinen ja kattavasti dokumentoitu. Suunnitelmien valmistuttua ne toteutettiin mahdollisimman tarkasti ja koko prosessi suoritetaan ennalta määrätyn kaavan mukaisesti.
Vesiputosmallin huomattiin olevan liian kankea ohjelmistotuotannon tarpeisiin ja näitä ongelmia varten kehitettiin inkrementaaliset ja iteratiiviset menetelmät. Näissä menetelmissä ohjelmistoa kehitetään vaiheittain kohti valmista tuotetta. Erityisesti nämä menetelmät nousivat suosioon sen jälkeen, kun yhdysvaltain puolustusministeri julkaisi vuonna 1994 uuden standardin, joka salli inkrmentaalisten ja iteratiivisten menetelmien käytön vesiputosmallin rinnalla.
Internet aika on 2000-luvulla kiihdyttänyt tarvetta saada nopeasti valmista ja muutoksiin täytyy pystyä reagoimaan entistä pikaisemmin. Tätä tarvetta vastaamaan on kehitetty ketterät menetelmät, joista yksi tunnetuimpia on Scrum. Ne lainaavat toimivaksi havaittuja elementtejä iteratiivisista malleista, mutta painotukset ovat erilaisia. Iteraatiot (tai Scrumissa sprintit) ovat lyhyempiä, kommunikaatio tiimin sisällä ja tiimin sekä asiakkaan välillä ovat korostetussa roolissa. Ketterissä menetelmissä turhaa työtä vältetään kaikessa tekemisessä ohjelmoimisesta dokumentaatioon.

Ketterillä menetelmillä on tutkitusti saatu aikaisempia menetelmiä parempia tuloksia sekä nopeuden, että projektien onnistumisen suhteen. Ketterät menetelmät ovat saaneet osakseen myös kritiikkiä mm. puuttellisen dokumentaation osalta. Myös soveltuvuus isoihin projekteihin on kyseenalaistettu ja jonkinlainen hybridimalli saattaisikin olla isoissa projekteissa paras ratkaisu.