Teksti ja kuvat: Pentti Virtanen

Pentti Virtanen, Systeemityön asiantuntija, Tieturi Oy. FT tietojenkäsittelytiede, Väitöskirja: "Measuring and Improving Component-Based Software Development", vuonna 2003.

Ketteryydellä tuottavuutta ja laatua

Ohjelmistotuotannon tuottavuuteen vaikuttavat tekijät ovat olleet tiedossa COCOMO-mallin myötä viimeistään jo 1980-luvulla. Fred Brooks:n kuuluisa "The Mythical Man-Month"-kirja toi vuonna 1975 esiin ohjelmistojen todellisen luonteen käsitteellisinä rakenteina. Tästä huolimatta ketterät ohjelmistotuotannon menetelmät ovat päässeet valokeilaan vasta tällä vuosituhannella ja ovat edelleen perinteisiä työtapoja harvinaisempia käytännön työssä. Kannattaakin miettiä, mikä tätä kehitystä jarruttaa.

Avainasemassa ihminen

Brooksin mukaan ohjelmistojen tekemisessä vaikeaa on ajatteleminen. Pitää siis luoda yhteinen ajattelumalli, joka ratkaisee jonkin liiketoiminnan ongelman. Ajattelemiseen tarvitaan edelleenkin aina ihmisiä. Niinpä keskeisin tuottavuuteen vaikuttava tekijä on ihminen. Tuottavuuserot eri ohjelmistokehittäjien välillä ovat erittäin suuria. Puhutaan kymmenkertaisista eroista jopa yhtä pitkän kokemuksen omaavien henkilöiden välillä.

Pariohjelmointi, koodin yhteisomistus ja aktiivinen tiimityö ovat tehokkaita menetelmiä osaamisen levittämiseen kehittäjäorganisaatiossa.

Kysymys on tarkemmin sanottuna osaamisen ja motivaation eroista. Perinteinen "palkataan vain huippuja"-ratkaisu on käytännössä jokseenkin mahdoton toteuttaa etenkin laajassa mittakaavassa. Ketterien menetelmien kuten Extreme Programmin (XP) mukana on tullut uusia vaihtoehtoja osaamisen ja motivaation parantamiseen. XP:n käytännöt kuten 40-tuntinen työviikko ja vastuiden vapaaehtoisen ottamisen periaate taas edesauttavat ohjelmoijien motivaatiota.

Kommunikaation merkitys

Tiimityöhön kuluu merkittävä osa ohjelmistotuotannon työajasta. Niinpä tiimien koon minimointia on monesti ehdotettu ratkaisuksi ohjelmistotuotannon tuottavuusongelmiin. Tässä ei ollakaan aivan väärässä, mutta laajojen ohjelmistojen tekeminen pienen tiimin avulla on vaikeaa, jos työtä ei pystytä kunnolla osittamaan. Ketterien menetelmistöjen käyttö onkin yleensä rajattu pieniin, alle 20 hengen, tiimeihin. Niiden soveltuvuutta isompiin projekteihin voidaan parantaa tehostamalla ohjelmiston osittamista paremmin suunniteltujen ohjelmistoarkkitehtuurien avulla. Ketterissä menetelmistöissä on tyypillisesti mukana arkkitehtuuritehtäviä, jotka mahdollistavat ohjelmiston osien itsenäisen kehittämisen.

Ajattelumallin siirtäminen ihmiseltä toiselle voi tapahtua vain kommunikaation avulla. Perinteinen tapa on laatia ja katselmoida kirjoitettuja dokumentteja. Tämä on kuitenkin erittäin kallista ja hidasta. Ketterissä menetelmissä koko tiimi asiakas mukaan lukien kootaan samaan työtilaan, jossa nämä pystyvät tehokkaasti käyttämään puhetta ja yksinkertaisia kaavioita ideoiden, vaatimusten ja näkemysten siirtämiseen henkilöltä toiselle. Säästö kirjallisessa dokumentaatiossa edellyttää osapuolten välistä luottamusta. Sitä taas ei ole aina mahdollista saavuttaa. Tiimien maantieteellinen hajautuminen voi myös olla joissain tilanteissa väistämätöntä. Ketterät menetelmistöt näkevät koodin ja siihen liittyvät testitapaukset ohjelmiston ensisijaisena dokumentaationa. Koodin luettavuutta parannetaan ohjelmointistandardeilla, pariohjelmoinnilla ja vaihtamalla koodin vastuuhenkilöitä usein. Tiimissä mukana oleva asiakas katselmoi ja testaa ohjelmiston prototyyppejä ja osallistuu mallintamiseen keveiden ja kertakäyttöisten luonnosten avulla. Tällä säästetään tarve kirjoittaa ohjelmisto ensin luonnollisella kielellä ja sitten uudestaan ohjelmointikielellä.

Iteratiivisuudesta apua

Dokumenttipohjainen systeemityö tuottaa usein huonolaatuisia ohjelmistoja, vaikka dokumenttien tekemiseen ja katselmoimiseen on käytetty merkittäviä työpanoksia. Syynä tähän on se, että ohjelmistotuotanto on luonteeltaan lähempänä tuotekehitystä kuin tuotteen tehdasmaista monistamista. Tuotekehityksen ja ohjelmistotuotannon projekteissa tuotetaan jotain ainutkertaista, jolloin tarkan vaatimusmäärittelyn tekeminen on mahdotonta. Ohjelmiston soveltuvuus liiketoiminnan tarpeeseen voidaan varmistaa vasta käyttämällä sitä tuotannossa. Samoin uuden arkkitehtuurin keskeiset ominaisuudet kuten suorituskyky, luotettavuus ja ylläpidettävyys voidaan monesti arvioida vasta kun ohjelmisto on jo merkittäviltä osiltaan tehty. Tämän kaiken lisäksi vielä asiakkaiden vaatimukset muuttuvat mm. heidän liiketoimintansa muutosten seurauksena. Viimemainitusta syystä laaja vaatimusmäärittely on usein valmistuessaan jo vanhentunut. Ratkaisu näihin perustavaa laatua oleviin ongelmiin on iteratiivinen ohjelmistotuotanto, jossa ohjelmistoa tuotetaan pienissä osissa valmiiksi saakka. Tyypillinen iteraation kesto on 2-

8 viikkoa, jona aikana tuotetaan seuraava mahdollisuuksien mukaan tuotannossa asti testattava ohjelmiston osa.

Iteratiivinen ohjelmistotuotanto on eräs ratkaisu ohjelmistotuotannon riskeihin, jotka ovat keskeinen kustannustekijä. Edellä mainittujen riskien lisäksi projekteissa on itsessään riskejä, joihin varautuminen on usein puutteellista. Riskilistojen laatiminen on vanhan kokemuksen perusteella kohtuullisen helppoa. Ongelmana on uskallus ottaa riskivaraukset mukaan aikatauluihin ja kustannusarvioihin. Koska aikataulu on monille projekteille tärkeämpää kuin toimitettava toiminnallisuus, voidaan vaatimusten priorisoinnin avulla varautua yllättäviin tilanteisiin.

Laadun parantamisen keinot

Ohjelmistotuotannon prosessien kehittäminen on ollut painopistealueena jo pitkään. Tämä on usein liitetty laadun kehittämiseen ja laatusertifiointeihin. Laadun ja tuottavuuden parantaminen prosesseja standardoimalla on yleisesti hyväksytty toimintatapa monilla toimialoilla. Yleispätevät standardiprosessit eivät kuitenkaan ole aina tuottaneet haluttuja tuloksia. Byrokratia on lisääntynyt, eivätkä lopputulokset ole niin laadukkaita kuin on odotettu. Prosessien sovittaminen kuhunkin projektiin on osaratkaisu, joka ei pysty muuttamaan prosessin peruslähtökohtia. Vahvaan ja kurinalaiseen ennakkosuunnitteluun perustuva prosessi ei toimi, jos projektin haluttu lopputulos ei ole ennakoitavissa ja muuttuu jatkuvasti. Niinpä ketterät ohjelmistotuotannon menetelmistöt soveltuvat nykyaikaiseen nopearytmiseen liiketoimintaan kurinalaisia menetelmistöjä paremmin.

Perinteisen laatuajattelun kulmakivi, virhekustannusten välttäminen, on kuitenkin edelleen validi. Virhekustannus kasvaa virheen synnyn ja sen havaitsemisen välisen ajan kasvaessa. Ketterät ja perinteiset menetelmistöt ratkaisevat virhekustannusongelman kuitenkin eri tavoin. Perinteiset menetelmistöt luottavat dokumenttien laadun parantamiseen katselmointien avulla. Tämä johtaa massiiviseen dokumenttien ylläpitoon. Katselmoinneissa ei kuitenkaan voida löytää virheitä, joita kukaan katselmoijista ei osaa ennakoida.

Ketterien menetelmistöjen ratkaisu on pakkomielteeseen asti viety testaus yhdessä inkrementaalisen kehittämisen kanssa. Perinteistä korttitaloa ei synny, koska ohjelmistoa testataan aktiivisesti koko kehitystyön ajan. Jos virheitä on perinteiseen tapaan lisätty virheiden päälle ja niiden havaitsemien on jätetty projektin loppuun integraatiovaiheeseen, on projektin pelastaminen vaikeaa. Syntyy paniikkivaihe, jossa kaikki mitä laadukkaasta ohjelmistotuotannosta on opittu, heitetään romukoppaan. Paniikissa syntyy lisää virheitä ja niille entistäkin suurempia kustannuksia.

Uudelleenkäytön haasteet

Ohjelmistokomponenttien uudelleenkäyttöä on pitkään pidetty mahdollisena menestystekijänä, mutta toivottua läpimurtoa ei ole saavutettu. Kehitysympäristöjen mukana tulevien peruskomponenttien uudelleenkäyttö on monesti ainoa tulos tällä alueella. Ongelmana on tarvittavien investointien tarve. Jonkun pitää ensin löytää, ymmärtää ja arvioida uudelleenkäytettävät komponentit. Sen jälkeen ne pitää vielä sovittaa uusiin käyttötarkoituksiin. Monesti tämä on yksittäistapauksissa kalliimpaa kuin komponentin kirjoittaminen alusta alkaen.

Perinteisen tavan edut

Perinteisillä menetelmistöillä on kuitenkin pakko olla jotain etuja ketteriin verrattuna, koska niiden käyttö on niinkin laajaa. Kaikkein selvimmin esiin nousee mahdollisuus kilpailuttaa ohjelmistotoimittajia ja saada ratkaisuille kiinteä hinta. Kilpailuttaminen on pakollista julkisissa hankkeissa ja laajalti käytössä myös muualla. Kiinteän hinnan edellytyksenä on kiinteä ja yksiselitteisesti määritelty tehtävä. Niinpä dokumenttipohjainen ratkaisu tuntuu ainoalta mahdolliselta. Iteratiivisissa projekteissa tarvitaankin ostajan suojaksi mahdollisuus vetäytyä epäedulliseksi osoittautuneesta yhteistyöstä. Projektin lopettaminen on käytännössäkin mahdollista iteraatioiden päättyessä, jos tuloksena syntyvät tuotokset ovat ostajalle käyttökelpoisia ja kenties sellaisenaan liiketoimintahyötyjä tuottavia. Kilpailuttamisessa on myös mahdollista käyttää muitakin valintakriteereitä kuin pelkkä hinta. Hintakilpailu johtaa helposti siihen, että ohjelmistotoimittajat tinkivät laadusta. Vaatimusdokumenteilla ja jälkikäteen tehtävällä laadunvarmistuksella on vaikea estää tätä. Tilaaja voi myös käyttää budjettihintaa ja aikatauluja, joita käytetään yhdessä priorisoitujen vaatimusten kanssa rajaamaan projektin tehtävät.

Ketterät menetelmistöt eivät ole niin tunnettuja kuin perinteiset, eikä niihin liittyvä osaaminen ole levinnyt kovin laajalle. Intuitiivisesti ajateltuna iteratiivisia projekteja on vaikea hallita ja lopettaa eikä niiden lopputuloksia voida ennakoida samalla tavoin kuin perinteisiä projekteja. Käytäntö voi olla päinvastainen.

Johtopäätös

Kun rakennettavaa ohjelmistoa ei ole mahdollista määritellä ja suunnitella etukäteen riittävän hyvin, tarvitaan tuotantotapa, jossa ohjelmistoa rakennetaan osissa ja jossa aktiivisesti hyödynnetään aikaisemmista vaiheista kerättävää todellista palautetta. Syntyvän ohjelmiston laadun paraneminen johtaa myös tuottavuuden paranemiseen, kun virheiden korjaamiseen käytettävä aika vähenee.