Referaatti Martin Fowlerin artikkelista ”Is Design Dead”

Pete Saarela

http://martinfowler.com/articles/designDead.html

Artikkelissaan Fowler pohdiskelee ohjelmistosuunnittelun asemaa XP-tyyppisessä (Extreme Programming) ohjelmistokehityksessä. Hän käyttää siitä termiä Evolutionary Design, vertailukohteena perinteinen Planned Design, jossa esimerkiksi ohjelmistoarkkitehti on suunnitellut toteutuksen ennakkoon ja koodaajille jää tehtäväksi ainoastaan toteuttaa suunnitelma mahdollisimman tarkasti.

XP:n suurimmat ongelmat aiheutuvat eri iteraatioiden aikana ”lennossa” tehdyistä taktisista päätöksistä, jotka johtavat kokonaisuuden rikkoutumiseen. Koodista tulee vaikeasti muokattavaa ja virhealtista. Tuloksena on koodaa-ja-korjaa –painajainen, jossa virheiden paikkaamisesta tulee yhä työläämpää projektin edetessä. Hintakäyrää voidaan kuitenkin tasoittaa XP:n opinkappaleita noudattamalla. Niistä tärkeimmät ovat yksinkertaisuuteen pyrkiminen ja ylimääräisen työn välttäminen.

Ketterän kehityksen manifesti toteutetaan tiettyjä kriteereitä tarkkailemalla. Niitä ovat testien läpäisy, toisteisuuden välttäminen, läpinäkyvyys ja luokkien/metodien määrän minimointi. Teknisiä menetelmiä ovat jatkuva testaaminen, integrointi ja refaktorointi. Huolimatta tietyistä vasta-argumenteista, Fowler suosittelee ohjelmointimallien harkittua käyttämistä.

Ohjelmistoarkkitehtuuri tulisi valita niin, että sen muuttaminen on mahdollista projektin edetessä. Myös UML-kaavioita voi käyttää, pitäen mielessä että niiden tarkoituksena on kommunikointi; niiden tulisi olla kevyitä ja helposti muokattavia. Muita merkillepantavia asioita ovat muun muassa päätösten peruutettavuus, perinteisten tittelien ja tehtävien uudelleen ajattelu sekä kokonaislaadun tarkkailu. Vastauksena artikkelin otsikon kysymykseen todetaan, että suunnittelu ei suinkaan ole kuollut, mutta sen luonne on ratkaisevasti muuttunut ja uuden polven suunnittelijoilta vaaditaan uudenlaisia taitoja ja tahtotilaa.

Referaatti Kenny Heinosen kandidaatintutkielmasta ”Ryhmätyö ohjelmistokehityksessä”

Pete Saarela

Ohjelmistoala ja ryhmätyöskentely

Tyypillisesti nykyajan laajat ja kompleksiset ohjelmistoprojektit toteutetaan ryhmätöinä. Ryhmän suorituskyky määräytyy sen jäsenten yhteistyö-, kommunikointi- ja teknisten taitojen pohjalta. Ne ovat avainasemassa ketteriä prosessimalleja käytettäessä, joista tutkielmassa kuvaillaan kaksi esimerkkiä: Extreme Programming (XP) ja Scrum. Eroistaan huolimatta molemmat painottavat yksilöiden välistä vuorovaikutusta.

Etenkin Scrum vaatii tiimeiltä itseorganisoituvuutta ja jäseniltä monitaitoisuutta (cross-functionality). Tiimillä ei ole johtajaa tai projektipäällikköä. Prosessin jouheva eteneminen ja laadukas työn jälki varmistetaan lukuisilla ja säännöllisillä kokouksilla (suunnittelu- ja päiväpalaverit, katselmoinnit sekä retrospektiivit), joissa hyvät kommunikointitaidot ovat välttämättömiä.

Ohjelmistoprojektin sujuvuudessa myös ihmisten luonteenpiirteillä on merkitystä. Myers-Briggsin tyyppi-indikaattorien avulla on selvitetty erilaisten persoonallisuuksien soveltuvuutta tuotantoprosessin eri vaiheisiin. Haastattelujen perusteella työtoverit luonnollisesti arvostavat sellaisia piirteitä kuin joustavuus, älykkyys ja mukavuus. Silti kaikki indikaattorien määrittelemistä luonteenpiirteistä ovat hyödyksi jossakin ohjelmiston kehitysvaiheessa – myös introverttiys, intuitiivisuus, tuntevuus ja spontaanius.

On lisääntyvää tarvetta menetelmille, joilla rohkaistaan ryhmätyöhön ja kehitetään yksilöiden kommunikointitaitoja. Heinonen kuvaa useita erityisesti opiskelijoille kehitettyjä menetelmiä, jotka sisältävät muun muassa vuorovaikutukseen aktivointia, sanallisia harjoituksia, vertaisarviointeja, projektioppimista ja iteratiivista ryhmätyöskentelyä.

Referaatti Eero-Veikko Laineen kandidaatintutkielmasta ”Johtaminen perinteisissä ja ketterissä ohjelmistoprojekteissa”

Pete Saarela

Johtaminen perinteisissä ja ketterissä ohjelmistotuotantoprojekteissa

2000-luvulla ketterät ohjelmistotuotannon prosessimallit ottivat haasteen vastaan: kuinka parantaa monimutkaistuvien IT-projektien alati huononevaa tuottavuutta. Yksi tärkeimmistä painopisteistä on vastuun jakaminen toisin kuin perinteisessä projektijohtamisessa, joka on perustunut tarkkoihin etukäteissuunnitelmiin. Arkkityyppiset tiimirakenteet perustuivat joko pääohjelmoijajohtoiseen tai ”egottomaan” tiimiin, joista jälkimmäisestä lainattiin paljon elementtejä ketterään ohjelmistokehitykseen.

Riskianalyysien ja kattavan dokumentaation sijaan tavoitteeksi otettiin mahdollisimman joustava reagointi muutoksiin. Asiakas haluttiin mukaan ohjelmiston kehitykseen ja projektipäällikön rooli muuttui kontrolloijasta eräänlaiseksi tienraivaajaksi (facilitator). Johtamisesta muodostettiin koko tiimin yhteinen kollektiivinen ja muuttuva prosessi.

Scrum-menetelmässä aiemman projektipäällikön roolin korvaa Scrum Master, joka yhdessä Product Ownerin kanssa vastaa projektista. Vastuu ulottuu kuitenkin vielä syvemmälle; tiimin jäsenet ovat erityisasiantuntijoita, joille hajautettu johtajuus määrittyy kulloisenkin haasteen mukaan. Kun aiemmin tiimin jäsenten taitojen päällekkäisyyttä (redundancy) pidettiin tehottomana, ketterissä työryhmissä siihen suorastaan pyritään.

Yksi ketteryyden pääperiaatteista on itseohjautuvuus. Vallan ja vastuun jakamisen on todettu kannustavan jäseniä sitoutumiseen, luovuuteen, tuottavuuteen ja ongelmanratkaisun nopeuteen. Jaetun johtamisen soveltaminen ei kuitenkaan ole aina helppoa. Tiimit eivät edelleenkään voi toimia täysin vailla ulkopuolista kontrollia, jotta ketteryyteen kuuluva epävakaus ei johtaisi kaaokseen. Yksilön autonomia ei saa ohittaa ryhmän autonomiaa. Muun muassa tuon riskin pienentämiseksi on eritelty erilaisia epävirallisia rooleja, joita jäsenet voivat omaksua, esimerkiksi mentori, koordinoija, tulkki, puolestapuhuja, markkinoija ja terminaattori.