Ohjelmiston arkkitehtuurisuunnittelu ketterissä menetelmissä

Ohjelmiston arkkitehtuuri on IEEE:n standardin (Jen ja Lee, 2000) mukaan ohjelmiston perusorganisaatio, joka sisältää järjestelmän osat, niiden keskenäiset suhteet ja suhteet ympäristöön. Määritelmän mukaan arkkitehtuuri ohjaa järjestelmän suunnittelua ja evoluutiota.

Rational Unified Process (Kruchten, 2004) määrittelee ohjelmistoarkkitehtuurin joukkona perustavanlaatuisia päätöksiä koskien järjestelmän organisaatiota, rakenteellisia elementtejä, rajapintoja, alijärjestelmiksi jakautumista, arkkitehtuurillista tyyliä ja niin edelleen. RUP:in mukaan arkkitehtuurissa ei olla kiinnostuneita pelkästään rakenteesta ja toiminnasta, mutta myös sen käyttökohteesta, suorituskyvystä, joustavuudesta, uudelleenkäytettävyydestä ja ymmärrettävyydestä. Arkkitehtuurissa pitäisi ottaa huomioon taloudelliset ja teknologiset rajoitteet.

Nykyään suosioon nousseita ketteriä menetelmiä yhdistää iteratiivinen ja inkrementaalinen elinkaari, pienissä erissä tapahtuvat julkaisut, yhtenäinen tiimi sekä ominaisuus- tai tuotebacklogiin perustuva julkaisusuunnitelma. Olennaista on myös pyrkiminen prosessin yksinkertaisuuteen ja sopeutuminen vaatimusten muutoksiin. Eri ketterien menetelmien kuvaukset ei tyypillisesti kerro mitään arkkitehtuurin suunnittelusta. (Abrahamsson et al., 2010)

Perinteisessä ohjelmistotuotannossa, kuten vesiputousmallissa, ohjelmistoon liittyvä suunnittelu tapahtuu tyypillisesti ennen varsinaista sovelluksen toteutusvaihetta. Tämä Big Design Up-Front (BDUF) -käytäntö yhdistetään myös ohjelmiston arkkitehtuurin suunnitteluun. (Abrahamsson et al., 2010)

Ketterän ohjelmistokehityksen periaatteet ovat epäsuhdassa arkkitehtuuriperustaisen ohjelmistosuunnittelun kanssa. Ketterän manifestin (Fowler, Highsmith et al., 2001) mukaan muutokseen reagoimista pidetään tärkeämpänä kuin tarkkojen suunnitelmien noudattamista. Halutaan minimoida kaikki turha työ, sillä arkkitehtuuriin saatetaan joutua tekemään muutoksia ohjelmiston elinkaaren aikana. Ihanteena onkin, että päätökset pyritään tekemään mahdollisimman myöhäisessä vaiheessa: mitä myöhemmin päätökset tehdään, sitä enemmän tietoa on hyödynnettävissä päätösten tueksi. Lisäksi asiakkaalle arvoa tuottava ohjelmiston ominaisuuksien nopea tuottaminen nähdään tärkempänä asiana kuin aikaa vievä tarkka suunnittelu ja dokumentointi.

Artikkelin Agility and architecture: Can they coexist? (Abrahamsson et al., 2010) mukaan suurin vastakkainasettelu sijaitsee ketterän kehityksen sopeutumismentaliteetin ja perinteisen suunnittelun ennakoinnin välillä. Ketterässä kehityksessä liian tarkkaa etukäteissuunnittelua vältetään. Arkkitehtuurisuunnittelu saatetaan nähdä asiana menneisyydestä, johon ei tarvi panostaa. Arkkitehtuurin nähdään rakentuvan asteittain ohjelmoin-

nin yhteydessä, iteraatioiden edetessä, refaktorointia hyödyntäen. Toisaalta perinteisen suunnitteluvetoisen ohjelmistokehityksen kannattajat ovat saattaneet nähdä ketterät menetelmät hieman amatöörimäisenä touhuna, joka soveltuu hyvin vain tietynlaisiin, tyypillisesti pieniin projekteihin. Artikkelin ja esimerkiksi saman aihepiirin kirjan Agile Software Architecture (Babar et al., 2013) kantavana teemana on tälläisen vastakkainasettelun häivyttäminen - ketteryys ja arkkitehtuuri voivat elää yhteiseloa olla jopa toisiaan tukevia asioita.

Oikealla tavalla suunniteltu arkkitehtuuri voi olla hyvä työkalu parantamaan ohjelmiston laatua sekä vähentämään kehitykseen käytettyä aikaa ja kuluja, kuten myös ketterät menetelmät voidaan nähdä laatua, tuottavuutta, kannattavuutta ja ennen kaikkea sovelluskehittäjien tyytyväisyyttä lisäävänä asiana. (Abrahamsson et al., 2010; Babar et al., 2013)

Kuinka arkkitehtuurisuunnittelua sovelletaan ketterässä ohjelmistokehityksessä

A systematic mapping study on the combination of software architecture and agile development (Yang et al., 2016) listaa 43 arkkitehtuurisuunnittelun lähestymistapaa sekä niihin liittyviä tutkimuksia. Listattuna ovat muun moassa iteratiivinen lähestymistapa sekä nollaiteraatio.

Iteratiivinen arkkitehtuuri (Emergent Architecture) sulauttaa arkkitehtuurisuunnittelun osaksi ketterän tuotannon iteraatoita. Arkkitehtuurisuunnittelua tehdään vain tarpeellinen määrä kyseisen iteraation tavoitteiden, eli käytännössä uusien käyttäjätoiminnallisuuksien toteuttamiseksi. Tässä mallissa arkkitehtuuri rakentuu refaktoroinnin kautta: sitä ei olla määritelty etukäteen vaan se kehittyy koodin mukana. (Abrahamsson et al., 2010; Babar et al., 2013)

Abrahamsson et al. (2010) artikkelissa todetaan, että arkkitehtuurin suunnittelu tulisi tehdä mahdollisimman aikaisin, sillä se kattaa merkittäviä päätöksiä järjestelmän rakenteesta ja käyttäytymisestä - näitä päätöksiä on vaikea kumota tai muuttaa myöhemmissä vaiheissa projektia.

Nollaiteraatiossa (Iteration Zero) on tarkoitus siirtää osa arkkitehtuurisuunnittelua tapahtuvaksi ennen varsinaista tuotantovaihetta olevaan iteraatioon sekä parantaa arkkitehtuuria tulevissa iteraatioissa (Babar et al., 2013). Tässä vaiheessa on mahdollista tehdä ns. walking skeleton, eli eräänlainen prototyyppi arkkitehtuurista, jonka tarkoitus on rakentua järjestelmän kasvaessa. Esimerkiksi Scrumiin on ollut ennen määriteltynä ns. pregame-vaihe, jossa korkean tason arkkitehtuuri on määritelty. Tämä on kuitenkin poistettu myöhemmistä Scrumin kuvauksista (Babar et al., 2013).

Ketterässä kehityksessä muutoksen pitäisikin olla ns. tavoiteltavissa oleva asia, joten etukäteen tehtävää arkkitehtuurisuunnittelua ei pitäisi periaatteessa esiintyä.

Babar et al. (2013) kirjassa viitatun tutkimuksen mukaan Scrum-ohjelmistokehityksessä

kuitenkin yleisesti hyödynnetään nollasprinttejä, BDUF-henkistä suunnittelua ja erillisten arkkitehtuuritiimien käyttöä. Abrahamsson et al. (2010) mainitsee Davide Falessin tutkimuksen, jonka mukaan ketterää hyödyntävät ohjelmistokehittäjät yleisesti pitävät arkkitehtuuria merkityksellisenä apuvälineenä, joka muun muassa helpottaa kommunikointia ja suunnitteluvaihtoehtojen arviointia.

Ketterän manifestin (Fowler, Highsmith et al., 2001) mukaan parhaat arkkitehtuurit syntyvät itsenäisten kehitystiimin sisällä. Tämän voi nähdä lisäävän arkkitehtuuriin sitoutumista ja vähentävän dokumentaatioita. Toisaalta varsinkin suuremmissa organisaatioissa tämä ideaali ei välttämättä voi olla mahdollinen - arkkitehtuurilliset päätökset voi esimerkiksi tulla tiimin ulkopuolelta. Tällöin taas esimerkiksi dokumentaation tärkeys voi korostua.

Se, miten arkkitehtuuri ja ketterä ohjelmistotuotanto yhdistetään, ei ole itsestäänselvyys. Tästä on onneksi tehty tutkimusta - hyviä lähtökohtia aihepiirin tutkimukseen vaikuttaisi olevan esimerkiksi lähteissä mainitut Yang et al. (2016) kartoitustutkimus ja Babar et al. (2013) aiheesta koottu kirja.

Lähteet

Abrahamsson, P., Babar, M. A. ja Kruchten, P. (2010). "Agility and architecture: Can they coexist?" *IEEE Software* 27.2, s. 16–22.

Babar, M. A., Brown, A. W. ja Mistrık, I. (2013). *Agile Software Architecture: Aligning Agile Processes and Software Architectures*. Newnes.

Fowler, M., Highsmith, J. et al. (2001). "The agile manifesto". *Software Development* 9.8, s. 28–35.

Jen, L. R. ja Lee, Y. J. (2000). "Working Group. IEEE recommended practice for architectural description of software-intensive systems". Teoksessa: *IEEE Architecture*. Citeseer.

Kruchten, P. (2004). *The rational unified process: an introduction*. Addison-Wesley Professional.

Yang, C., Liang, P. ja Avgeriou, P. (2016). "A systematic mapping study on the combination of software architecture and agile development". *Journal of Systems and Software* 111, s. 157–184.