Kanditutkielma

Tietojenkäsittelytieteen kandiohjelma

Arkkitehtuurisuunnittelu ketterässä ohjelmistokehityksessä

Juuso Kotimäki

20.3.2021

Matemaattis-luonnontieteellinen tiedekunta

Helsingin yliopisto

Ohjaaja(t)

Lea Kutvonen

Tarkastaja(t)

Yhteystiedot

PL 68 (Pietari Kalmin katu 5)
00014 Helsingin yliopisto

Sähkopostiosoite: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Matemaattis-luonnontieteellinen tiedekunta	Tietojenkäsittelytieteen kandiohjelma

Tekijä — Författare — Author

Juuso Kotimäki

Työn nimi — Arbetets titel — Title

Arkkitehtuurisuunnittelu ketterässä ohjelmistokehityksessä

Ohjaajat — Handledare — Supervisors

Lea Kutvonen

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Kanditutkielma	20.3.2021	14 sivua

Tiivistelmä — Referat — Abstract

Perinteisessä vesiputousmallisessa ohjelmistokehityksessä ohjelmiston arkkitehtuuri suunnitellaan ennen ohjelmoinnin aloittamista. Puhtaasti ketterän ideologian mukaisessa ohjelmistokehityksessä ei tulisi varata aikaa etukäteissuunnittelulle, vaan arkkitehtuurin tulisi muodostua inkrementaalisesti iteraatioiden aikana. Tällä tavalla muodostunut arkkitehtuuri voi kuitenkin osoittautua myöhemmin ongelmalliseksi, ja useassa tapauksessa jonkinlainen arkkitehtuurin etukäteissuunnittelu on epäketteryydestään huolimatta tarpeen.

Ketterien menetelmien kuvaukset eivät kerro tyypillisesti mitään, kuinka arkkitehtuurisuunnittelu tulisi niissä suorittaa. Tässä tutkielmassa käsitellään eri yleisesti käytössä olevia käytänteitä arkkitehtuurisuunnittelun toteuttamiseksi. Lisäksi pohditaan eri projektiympäristön tekijöiden vaikutusta siihen, kuinka arkkitehtuurisuunnittelun menetelmä tulisi valita.

ACM Computing Classification System (CCS)
Software and its engineering → Software organization and properties → Software system structures → Software architectures
Software and its engineering → Software creation and management → Software development process management → Software development methods → Agile software development

Avainsanat — Nyckelord — Keywords

Ohjelmistoarkkitehtuuri, ketterä ohjelmistokehitys

Säilytyspaikka — Förvaringsställe — Where deposited

Helsingin yliopiston kirjasto

Muita tietoja — övriga uppgifter — Additional information

Sisältö

1 Johdanto

Ohjelmiston arkkitehtuuri on ohjelmiston perusorganisaatio, joka sisältää järjestelmän osat, niiden keskinäiset suhteet ja suhteet ympäristöön (Jen ja Lee, 2000). Arkkitehtuurillisiin valintoihin vaikuttaa ainakin käyttökohde, joustavuus, uudelleenkäytettävyys ja ymmärrettävyys (Kruchten, 2004).

Ketterät menetelmät ovat nousseet suosituimmaksi ohjelmistotuotantoprosessiksi (*14th annual state of agile report* 2020). Ketteriä menetelmiä yhdistää iteratiivisesti ja inkrementaalisesti tapahtuva ohjelmiston tuotanto, pienissä erissä tapahtuvat julkaisut, tiivis tiimi sekä ominaisuus- ja tuotebacklogiin perustuva julkaisusuunnitelma (Babar et al., 2013). Ketterille menetelmille olennaista on sopeutuminen vaatimusten muutoksiin (Fowler, Highsmith et al., 2001).

Perinteisessä vesiputousmallia noudattavassa ohjelmistotuotannosa kaikki suunnittelu tehtiin tyypillisesti ennen varsinaista sovelluksen toteuttamisvaihetta. Ketterien periaatteiden mukaan etukäteissuunnittelua tulisi välttää.

Suurin vastakkainasettelu sijaitseekin ketterän kehityksen sopeutumismentaliteetin ja perinteisen suunnittelun ennakoinnin välillä (Babar et al., 2013). Keterän manifestin (2001) mukaan muutokseen reagoimista pidetään tärkeämpänä kuin tarkkojen suunnitelmien noudattamista. Etukäteen tehdyt, mahdollisesti turhiksi osoittautuneet suunnitelmat nähdään turhana työnä. Ihanteena onkin, että päätökset tehdään mahdollisimman myöhäisessä vaiheessa: mitä myöhemmin päätökset tehdään, sitä enemmän tietoa on hyödynnettävissä päätösten tueksi.

Kuitenkin myös ketterissä piireissä on hyväksytty arkkitehtuurisuunnittelun tärkeys laatua parantavana tekijänä: oikealla tavalla suunniteltu arkkitehtuuri mm. vähentää kehitykseen käytettyä aikaa ja kuluja (Babar et al., 2013). Arkkitehtuurisuunnittelun laiminlyöminen voikin johtaa erittäin monimutkaiseen ohjelmistoon: ongelmat voivat johtaa jopa ohjelmiston kehityksen estymiseen (Vogel et al., 2011). Hyvällä arkkitehtuurisuunnittelulla vältytään suurelta määrältä uudelleenohjelmointia kehityksen aikaina (Eloranta, 2015).

Eri ketterien menetelmien kuvaukset eivät tyypillisesti kerro mitään siitä, kuinka arkkitehtuurisuunnittelu tulisi toteuttaa (Babar et al., 2013). Tämän tutkielman tarkoituksena onkin auttaa lukijaa suhtautumaan arkkitehtuurisuunnitteluun oikealla tavalla erilaisissa

ketterän sovelluskehityksen konteksteissa.

Luvussa 2 käsitellään apuvälineitä, joilla voidaan vähentää ja helpottaa arkkitehtuurisuunnittelua. Luvussa 3 keskitytään käytänteisiin, miten arkkitehtuurisuunnittelu ketterässä kehityksessä yleisesti toteutetaan. Luvussa 4 pohditaan eri tekijöiden vaikutusta arkkitehtuurisuunnittelun eli millä tavalla eri projektiympäristöissä arkkitehtuurisuunnittelu tulisi suorittaa.

2 Arkkitehtuurin apuvälineet

Merkittävää arkkitehtuurisuunnittelua esiintyy todellisuudessa ainoastaan vähän: suurin osa arkkitehtuuriin liittyvistä päätöksistä on valmiiksi sisällettyinä käytettyihin ohjelmistokehyksiin (Bellomo et al., 2014). Monissa ohjelmistoprojekteissa arkkitehtuurisuunnittelua ei tarvitse edes tehdä. Elorannan (2015) mukaan on esimerkiksi tyypillistä, että mobiiliapplikaatioiden yhteydessä ekosysteemi pakottaa käytetyn arkkitehtuurin ja webkehityksessä käytetään yleisesti täydellisen ohjelmistoarkkitehtuurin tarjoavia ohjelmistokehyksiä.

Jos soveltuvaa ohjelmistokehystä ei ole olemassa, voidaan suunnittelun pohjana usein käyttää jotain referenssiarkkitehtuuria. Ohjelmistokehysten ja referenssiarkkitehtuurien käytön etuina on se, että arkkitehtuurisuunnittelun määrä vähenee huomattavasti eli järjestelmä saadaan toimintakuntoon nopeammin (Waterman et al., 2015).

Arkkitehtuurin lähtökohdaksi voidaan toteuttaa projektin alussa alustava ohjelmarunko, joka toimii pohjana toiminnalisuuksien kehittämiseen ketterän kehityksen iteraatioissa.

2.1 Ohjelmistokehykset

Ohjelmistokehys (framework) sisältää oletusmallin arkkitehtuurista ja rajaa järjestelmän käyttämään tätä mallia (Waterman et al., 2015). Esimerkkejä ohjelmistokehyksistä ovat muunmoassa .NET, Hibernate ja Ruby On Rails.

Koska monet arkkitehtuurilliset päätökset ovat toteutettuna ohjelmistokehyksiin, niiden käyttö vähentää huomattavasti arkkitehtuurin kompleksisuutta jos arkkitehtuuriin joudutaan tekemään muutoksia (Waterman et al., 2015). Muutoksiin sopeutuvissa ketterissä menetelmissä kehykset ovat siis käytännöllisiä.

2.2 Referenssiarkkitehtuurit

Referenssiarkkitehtuuri (reference- ja template architecture) on tietylle arkkitehtuurilliselle vaatimukselle syntynyt toimivaksi todettu ratkaisu. Referenssiarkkitehtuurit dokumentoivat järjestelmän rakenteen, järjestelmän rakennuspalaset sekä niiden vastuut ja vuoro-

vaikutukset (Vogel et al., 2011).

Hyödyntäessä arkkitehtuurisuunnittelussa referenssiarkkitehtuuria tulee hyödynnettyä sen suunnitelleiden ihmisten asiantuntemus ja kokemus. Sen käyttö vähentää riskiä toimimattoman arkkitehtuurin suunnitelulle, parantaa arkkitehtuurin laatua ja vähentää suunnitteluun menevää aikaa (Vogel et al., 2011).

Tässä yhteydessä referenssiarkkitehtuurilla tarkoitetaan mitä vain arkkitehtuurimallia, jota voi käyttää mallina arkkitehtuurin suunnittelussa. Tällaisia malleja voisi olla esimerkiksi Oraclen Java EE, joka tarjoaa monia malliratkaisuja Javalla tai jokin perusarkkitehtuurimalli, kuten kerrosarkkitehtuuri.

2.3 Ohjelmarungot

Kehitettävän sovelluksen pohjaksi voidaan toteuttaa alustava ohjelmarunko. Ohjelmarungon rakenne vastaa lopullisen järjestelmän arkkitehtuuria, mutta ei vielä toteuta lopullista toiminnallisuutta (Vogel et al., 2011). Ohjelmarunko kehittyy lopulliseksi järjestelmäksi lisäämällä siihen toiminnallisuuksia inkrementaalisesti iteraatioiden aikana.

Cockburn (2004) esitteleen strategian, jossa projektin alussa rakennetaan ns. kävelevä luuranko (walking skeleton). Kävelevä luuranko on pieni implemaatio järjestelmästä, joka toteuttaa end-to-end -toiminnallisuuden. Tämä ohjelmarunko muodostaa lopullisen arkkitehtuurin sitä mukaa, kun siihen lisätään uusia ominaisuuksia. Kävelevä luuranko voidaan muodostaa esimerkiksi Sprint 0 aikana.

Ideana on, että kävelevässä luurangossa kaikki arkkitehtuurin alijärjestelmien yhteydet ovat toteutettuna (Cockburn, 2004). Esimerkiksi kerrosarkkitehtuurina toteutetun websovelluksen tapauksessa tämä ohjelmarunko voisi toteuttaa jonkin toiminnallisuuden, joka käyttää kaikkea kerrosarkkitehtuurin osia: frontendiä, backendiä ja tietokantaa.

3 Arkkitehtuurisuunnittelun käytänteet

Arkkitehtuurisuunnittelun voi luokitella ajoituksen mukaan kokonaan etukäteen tehtävään suunnitteluun, osittain etukäteen tehtävään suunnitteluun (Sprint 0) sekä kehityksen aikaiseen suunnitteluun, jossa koko arkkitehtuuri muodostuu täysin inkrementaalisesti. Arkkitehtuurisuunnittelu voi tapahtua myös erillisenä prosessina tiimin ulkopuolella.

Tämän luvun käytänteet vastaa Elorannan väitöskirjassa (2015) tutkittujen ohjelmistokehitystiimien yleisesti käyttämiä käytänteitä.

3.1 Koko arkkitehtuurin suunnittelu etukäteen

Vaikka etukäteissuunnittelu on määritelmällisesti ketterän ideologian vastaista, ketterässä ohjelmistokehityksessä kuitenkin usein harjoitetaan etukäteen kokonaisuudessaan tapahtuvaa arkkitehtuurisuunnittelua (Rost et al., 2015; Eloranta, 2015). Tässä mallissa arkkitehtuuri suunnitellaan kokonaan ennen siirtymistä muun toiminnallisuuden toteuttamiseen.

Toteuttamisvaiheessa arkkitehtuuriin tehdään enää korkeintaan pieniä muutoksia ja muutokset tekee tyypillisesti erillinen arkkitehti, ei ohjelmoija (Eloranta, 2015). Ongelmana tässä käytänteessä voidaan pitää, että projektin alkaessa on vain vähän arkkitehtuurillisia päätöksiä tukevaa informaatiota käytettäväksi (Waterman et al., 2015).

3.2 Sprint 0

Arkkitehtuurilliset päätökset tulisi tehdä mahdollisimman aikaisin (Abrahamsson et al., 2010). Mieluiten heti projektin alussa tulisi päättää, esimerkiksi onko järjestelmä hajautettu vai keskitetty, mitä teknologiastackiä käytetään ja niin edelleen (Eloranta, 2015). Ainakin jotain arkkitehtuurisuunnittelua siis tulisi tehdä heti projektin alussa.

Scrum on tällä hetkellä suosituin ketterän ohjelmistokehityksen viitemalli (*14th annual state of agile report* 2020). Sprintit ovat Scrumin kehitysjaksoja ja ne numeroidaan tyy-

pillisesti yhdestä eteenpäin.

Sprint 0:ssa on tarkoitus alustaa alkava kehitystyö eli hoitaa kaikki projektin aloituksen kannalta oleelliset toimenpiteet ennen varsinaista kehitystyön alkamista (Levison, 2008). Tässä yhteydessä Sprint 0:lla tarkoitetaan vaihetta, jossa arkkitehtuuri luodaan ensimmäisen iteraation aikana. Tyypillisesti tähän alustavaan arkkitehtuuriin tehdään muutoksia myöhempien sprinttien aikana (ks. Ohjelmistokehykset).

Sprint 0:lle on tyypillistä, että arkkitehtuurin suunnittelee ohjelmoijat itse, ei erilliset arkkitehdit (Eloranta, 2015). Sprint 0 on yleensä pituudeltaan yhden normaalin sprintin mittainen, mutta voi kestää myös useamman sprintin (Prause ja Durdik, 2012).

Sprint 0 voidaan kritisoida, koska se ei tuota välitöntä arvoa asiakkaalle. Scrum oppaan (2020) mukaan jokaisen sprintin pitäisi tuottaa potentiaalisesti julkaisukelpoinen lisäys tuotteeseen. Oppaassa ei ole mainintaa valmistelevasta sprintistä.

3.3 Kehityksen aikainen suunnittelu

Kehityksen aikaisessa suunnittelussa minkäänlaista arkkitehtuurin etukäteissuunnitteluvaihetta ei ole. Arkkitehtuuria suunnitellaan vain tarpeen mukaan järjestelmän ominaisuuksien toteuttamisen yhteydessä. Arkkitehtuuri valmistuu pala palalta kunnes se on valmis.

Ideana on tuottaa alustava arkkitehtuuri ensimmäisen iteraation aikaina samalla kuin toteutetaan tuotteeseen potentiaalisesti tulevia ominaisuuksia (Eloranta, 2015). Arkkitehtuuria suunnitellaan lähtökohtaisesti vain niitä ominaisuuksia varten, jotka on tarkoitus toteuttaa lähiaikoina (Waterman et al., 2015).

Kehityksen aikaisen arkkitehtuurisuunnittelun ajoitus vaihtelee. Sille voidaan varata aikaa iteraatiosta, sille voidaan varata koko iteraatio, arkkitehtuurisuunnittelua voi tapahtua muiden suunnitteluaktiviteettien (esim. daily scrum) yhteydessä, arkkitehtuuri voidaan toteuttaa myös kokonaan ohjelmoinnin yhteydessä (Rost et al., 2015). Suunnittelu voi kohdistua arkkitehtuurillisesti tärkeiksi arvioituihin aspekteihin, jokaiseen user storyyn, jokaiseen epiciin, jokaiseen sprintiin tai koko tuotteeseen (Rost et al., 2015).

Yksi tapa toteuttaa arkkitehtuuri sprinttien aikana on määritellä arkkitehtuurisuunnittelu kuten käyttäjätarinat (user storyt) arkkitehtuuritarinoina (Jensen et al., 2006).

Kehityksen aikaisen suunnittelun etuna on nopea arvontuotto, mutta arkkitehtuurin uu-

delleensuunnitteluun saatetaan joutua käyttämään aikaa myöhemmin. Watermanin (2015) mukaan kehityksen aikaisen suunnittelun käyttäminen lisää suunnitteluun käytettyä kokonaisaikaa: jokaisessa iteraatiossa täytyy miettiä, onko nykyinen arkkitehtuuri sopiva toteutettaville vaatimuksille: jos ei, arkkitehtuuria pitää uudelleensuunnitella.

Elorannan (2015) mukaan tätä suunnittelukäytäntöä käytti menestyksekkäästi kokeneet tiimit. Kokeneemattomalla tiimillä arkkitehtuurin etukäteissuunnittelun puute johti jatkuvaan refaktorointiin ja lopulta siihen, että koko arkkitehtuuri piti tehdä uudelleen.

3.4 Erillinen arkkitehtuuriprosessi

Tässä mallissa arkkitehtuurisuunnittelu on eriytetty omaksi prosessikseen. Prosessi tapahtuu tyypilliset täysin erillisessä arkkitehtitiimissä, jossa voi kuitenkin olla samoja jäseniä, kuin itse kehitystiimissä (Eloranta, 2015).

Arkkitehtuuritiimi ajoittaa arkkitehtuurillisten ominaisuuksien julkaisunsa milestonejen mukaan. Nämä taas määrittelevät sen, milloin varsinaisen tuotteen toiminnallisia ominaisuuksia voidaan alkaa kehittämään (Eloranta, 2015).

Elorannan (2015) mukaan syy erilliselle arkkitehtuuriprosessille oli usein, että ei haluttu sotkea arkkitehtuurisuunnittelua Scrum-prosessiin.

4 Arkkitehtuurisuunnittelun määrä

Vaikka monessa projekteissa, esimerkiksi websovelluksissa, arkkitehtuurisuunnittelua ei tarvitse juurikaan tehdä, joudutaan monessa yhteydessä arkkitehtuuri suunnittelemaan hyvinkin tarkasti. Esimerkkeinä paljon arkkitehtuurisuunnittelua vaatineista projekteista Eloranta (2015) mainitsee monimutkaiset kohteet, kuten työkoneiden ohjausjärjestelmät ja lääketieteellisten laitteiden ohjelmisot.

Siihen, kuinka paljon arkkitehtuurisuunnittelua tulisi tehdä etukäteen, vaikuttaa ainakin riski, odotettavissa oleva vaatimusten epävakaus, tiimin kulttuuri ja osaaminen sekä asiakkaan suhtautuminen ketteryyteen.

4.1 Riskin vaikutus

Arkkitehtuurisuunnittelun määrän tulisi määräytyä sen perusteella, että riski saadaan minimoitua riittävän tyydyttävälle tasolle (Fairbanks, 2010). Kyse ketteryyden ja riskin välillä tasapainottelusta: jos käytetään liian vähän aikaa arkkitehtuurin suunnitteluun, on riski ja todennäköisyys epäonnistua suurempi; jos suunnitteluun käytetään liikaa aikaa, arvon tuotto asiakkaalle viivästyy ja kyky vastata muutokseen vaikeutuu (Waterman et al., 2015).

Jos arkkitehtuurin etukäteissuunnittelu on liian vähäistä, saatetaan päätyä vahingolliseen arkkitehtuuriin, jossa ongelmien korjaamiseen uppoutuu enemmän aikaa kun toiminnallisuuksien toteuttamiseen (Waterman et al., 2015). Näin voi käydä esimerkiksi, jos arkkitehtuurisuunnittelua tapahtuu ainoastaan kehityksen aikana (Eloranta, 2015).

Mikä on hyväksyttävä riski vaihtelee paljon: esimerkiksi jos verkkokauppa kaatuu, voidaan menettää asiakkaita, mutta lääketieteellisessä järjestelmässä voi riskinä olla jopa ihmishenkien menettäminen. Tälläisissä tilanteissa on tyypillistä panostaa enemmän etukäteiseen arkkitehtuurisuunnitteluun (Waterman, 2018b).

Tiimi voi vähentää riskiä hyödyntämällä tutkimusta, mallinnuksella ja analyysillä, tekemällä kokeita tai rakentamalla toimivan ohjelmarungon, esim. kävelevän luurangon (ks. Ohjelmarungot) (Waterman et al., 2015).

4.2 Vaatimusten epävakaus

Vaatimusten epävakaus johtuu yleensä epämääräisesti määritellyistä tai vaihtelevista vaatimuksista (Waterman et al., 2015). Epämääräiset vaatimukset johtuvat siitä, ettei asiakas tiedä, mitä haluaa tai heiltä tulee uusia ideoita kehityksen aikana. Vaihtelevat vaatimukset johtuvat siitä, että asiakas muuttaa mieltään tai että käyttötapaukset muuttuvat.

Mitä enemmän voi olettaa vaatimusten muuttuvan, sitä muutosta suvaitsevampi arkkitehtuurin tulisi olla (Waterman et al., 2015). Muutosta suvaitseva arkkitehtuuri aikaansaadaan käyttämällä hyväksi hyviä suunnittelukäytänteitä, kuten kapselointia ja selkeää vastuunjakoa, päätösten tekoa viivyttelemällä sekä suunnittelemalla arkkitehtuuri niin, että vaihtoehdoille jätetään tilaa (Waterman, 2018a).

Hyvien käytänteiden käyttö ei varsinaisesti vähennä etukäteistyötä, mutta ketteryyden ylläpidettävyyden saavuttamiseksi näitä tulisi kuitenkin käyttää (Waterman, 2018a).

4.3 Aikainen arvontuotto

Jos tarkoituksena on aikainen arvontuotto, täytyy nopeuttaa ensimmäistä julkaisua vähentämällä arkkitehtuurisuunniitteluun käytettyä aikaa (Waterman et al., 2015). Käytännössä tämä voi tarkoittaa esimerkiksi kehityksen aikaista arkkitehtuurisuunnittelua.

Lean Startup -kontekstissa, jossa periaatteena on luoda tuote asiakkaan käyttäytymisestä tehtävien hypoteesien avulla - tuote (MVP, minimum viable product) hylätään tai hyväksytään (Reis, 2011). Tällaisessa tilanteessa suunnitteluun tulisi käyttää mahdollisimman vähän aikaa: jos tuote päädytään hylkäämään, on kaikki suunnitteluun käytetty aika mennyt hukkaan. Jos tuote menestyy ja päätetään ottaa laajempaan käyttöön, voidaan arkkitehtuuri suunnitella uudestaan vastaamaan paremmin laajempia käyttäjämääriä (Reis, 2011).

4.4 Tiimin kulttuurin ja kokemuksen vaikutus

Ketterän manifestin (2001) periaatteiden mukaan parhaat arkkitehtuurit syntyvät itseorganisoituvan tiimin sisällä. Elorannan (2015) mukaan tiimit, jotka arvostivat ketteriä arvoja valitsivat itse, minkälaista lähestymistapaa arkkitehtuurisuunniitteluun he käyttivät. Tyypillisesti tällöin lähestymistapa oli kehityksen aikainen suunnittelu. Sen sijaan perin-

teisiä toimintamalleja suosivissa yrityksissä lähestymistapa usein saneltiin johtoportaasta.

Tiimin arkkitehtuurillinen osaaminen vaikuttaa etukäteissuunnittelun määrään. Mitä parempi osaaminen on, sitä vähemmän suunnitteluun tarvitaan käyttää aikaa. Arkkitehtuurillisesti kokeneella tiimillä on ymmärrys siitä, mikä toimii ja mikä ei (Waterman et al., 2015).

Kokenut tiimi voi valita strategiakseen suorittaa arkkitehtuurisuunnittelun kokonaan kehityksen aikana. Elorannan (2015) mukaan kokeneet tiimit käyttivät tätä lähestymistapaa menestyksekkäästi.

Osaamisen lisäksi tiimin kyky kommunikoida vaikuttaa siihen, kuinka paljon tarvitaan dokumentaatiota ja etukäteispanostusta ohjelmistokehityksen ohjaamiseksi. Kommunikoinnin kykyyn vaikuttaa kulttuurin lisäksi tiimin koko: mitä suurempi tiimi, sitä enemmän vaaditaan rakennetta ja etukäteissuunnittelua (Waterman et al., 2015).

4.5 Asiakkaan vaikutus

Watermanin (2015) mukaan asiakkaan ketteryydellä on suuri vaikutus siihen, kuinka paljon etukäteissuunnittelua pitää tehdä eli kuinka ketterää arkkitehtuurisuunnittelu on. Prosessi-orientoitunut asiakas, joka ei usko ketterään ajatusmalliin vähentää huomattavasti tiimin mahdollisuuksia olla ketterä. Asiakas voi haluta hyväksyä kaikki mahdolliset suunnitelmat tai haluaa pakottaa oman prosessimallinsa tiimin sisälle.

Elorannan (2015) tutkimuksessa tuli ilmi, että arkkitehtuurin etukäteissuunnittelua käytettiin eniten projekteissa, joissa kehitettiin sulautettuja järjestelmiä. Elorannan mukaan tämä voi johtua siitä, että laitteistovalmistajat, joiden kanssa yhteistyö tapahtuu, eivät yleensä ole kovin ketteryysmyönteisiä.

Etukäteisbudjettien hyväksymisen tarve voi pakottaa tiimin suunnittelemaan arkkitehtuurin etukäteen, jotta he tietävät, paljonko heillä menee tuotteen valmistamiseen aikaa (Waterman et al., 2015). Abrahamsson et al. (2010) mainitsee mahdollisena ratkaisuna inkrementaalisen rahoitusmallin.

5 Yhteenveto

Tässä työssa käytiin läpi arkkitehtuurisuunnittelua helpottavista apuvälineistä ohjelmistokehykset, referenssiarkkitehtuurit sekä ohjelmarungot. Käsiteltiin arkkitehtuurisuunnitelussa yleisesti käytössä olevat käytänteet: koko arkkitehtuurin suunnittelu etukäteen, sprint 0, kehityksen aikainen suunnittelu sekä erillinen arkkitehtuuriprosessi. Lopuksi pohdittiin, mitkä asiat vaikuttavat arkkitehtuurisuunnittelun määrää.

Lähtökohtana voidaan pitää, että jos arkkitehtuurista on olemassa jonkinlainen malli, ohjelmistokehys tai referenssiarkkitehtuuri, kannattaisi niitä ehdottamasti hyödyntää. Varsinkin ketterän ohjelmistokehyksen kontekstissa edut ovat oleellisia: koodin tai konseptien uudelleenkäyttö vähentää kehitykseen käytettyä aikaa ja tekee arkkitehtuurista luotettavamman muutosten edessä.

Usein arkkitehtuurisuunnittelu on kuitenkin välttämätöntä. Suunnittelun lähestymistavassa pitää tehdä kompromissi ketteryyden ja riskin välillä. Tietyissä projektikonteksteissa riski on suurempi kuin toisissa. Suuren riskin kohteissa pitäisi panostaa enemmän arkkitehtuurinsuunnitteluun. Riskiä voidaan pienentää mm. arkkitehtuurin analyysillä ja kokeilla.

Mitä enemmän voidaan olettaa vaatimuksien muuttuvan ajan kuluessa, sitä vähemmän arkkitehtuurillisia päätöksiä tulisi tehdä etukäteen. Päätokset kannattaakin tehdä mahdollisimman myöhäisessä hyväksyttävässä vaiheessa. Muutoksiin sopeutuva arkkitehtuuri on kapseloitu, osilla on selkeä vastuunjako ja tulevaisuuden vaihtoehdoille tulee jättää tilaa.

Kun tavoitteena on aikanen arvontuotto, kannattaa luonnollisesti välttää kaikkea etukäteissuunnittelua. Lean Startup -konsepti on yksi esimerkki, jossa arkkitehtuurisuunnitteluun ei kannata panostaa liikaa.

Tiimin kulttuuri vaikuttaa arkkitehtuurisuunnitteluun. Mitä arkkitehtuurillisesti kokeneempi tiimi, sitä ketterämmän rkkitehtuurisuunnittelumenetelmän se voi valita. Mitä parempi kommunikaatio on, sitä vähemmän tarvitaan arkkitehtuurisuunnittelua.

Asiakkaan suhtautumisella ketteryyteen on suuri vaikutus siihen, kuinka ketterästi tiimi voi toimia. Projektin rahoitusmalli voi pakottaa tiimin käyttämään arkkitehtuurin etukäteissuunnittelua, jotta se voi arvioida työmääränsä.

Ei ole selvää todistusaineistoa, jonka mukaan jokin tietty asia vaikuttaisi yksinään siihen,

mikä tapa toimia olisi paras.

Lähteet

14th annual state of agile report (2020). URL: https://explore.digital.ai/state-ofagile/14th-annual-state-of-agile-report.

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 Mistrík, I. (2013). *Agile Software Architecture: Aligning Agile Processes and Software Architectures*. Newnes.

Bellomo, S., Kruchten, P., Nord, R. L. ja Ozkaya, I. (2014). "How to agilely architect an agile architecture". *Cutter IT Journal* 27.2, s. 12–17.

Cockburn, A. (2004). *Crystal clear: A human-powered methodology for small teams: A human-powered methodology for small teams*. Pearson Education.

Eloranta, V.-P. (2015). "Techniques and Practices for Software Architecture Work in Agile Software Development".

Fairbanks, G. (2010). *Just enough software architecture: a risk-driven approach*. Marshall & Brainerd.

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.

Jensen, R. N., Møller, T., Sönder, P. ja Tjørnehøj, G. (2006). "Architecture and design in extreme programming; introducing "developer stories"". Teoksessa: *International Conference on Extreme Programming and Agile Processes in Software Engineering*. Springer, s. 133–142.

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

Levison, M. (syyskuu 2008). *What is Sprint Zero? Why was it Introduced?* URL: https://www.infoq.com/news/2008/09/sprint_zero/ (viitattu 19. 03. 2021).

Prause, C. R. ja Durdik, Z. (2012). "Architectural design and documentation: Waste in agile development?" Teoksessa: *2012 international conference on software and system process (icssp)*. IEEE, s. 130–134.

Reis, E. (2011). "The lean startup". *New York: Crown Business* 27.

Rost, D., Weitzel, B., Naab, M., Lenhart, T. ja Schmitt, H. (2015). ”Distilling Best Practices for Agile Development from Architecture Methodology: Experiences from Industrial Application”. eng. Teoksessa: *Software Architecture*. Lecture Notes in Computer Science. Cham: Springer International Publishing, s. 259–267. ISBN: 978-3-319-23726-8.

Sutherland, J. ja Schwaber, K. (marraskuu 2020). *The Scrum guide – the definitive guide to Scrum: The rules of the game*. URL: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf (viitattu 19. 03. 2021).

Waterman, M. (2018a). ”Agility, Risk, and Uncertainty, Part 1: Designing an Agile Architecture”. *IEEE Software* 35.2, s. 99–101. DOI: 10.1109/MS.2018.1661335.

– (2018b). ”Agility, Risk, and Uncertainty, Part 2: How Risk Impacts Agile Architecture”. *IEEE Software* 35.3, s. 18–19. DOI: 10.1109/MS.2018.2141017.

Waterman, M., Noble, J. ja Allan, G. (2015). ”How Much Up-Front? A Grounded Theory of Agile Architecture”. Teoksessa: *Proceedings of the 37th International Conference on Software Engineering - Volume 1*. ICSE ’15. event-place: Florence, Italy. IEEE Press, s. 347–357. ISBN: 978-1-4799-1934-5.

Vogel, O., Arnold, I., Chughtai, A. ja Kehrer, T. (2011). *Software architecture: a comprehensive framework and guide for practitioners*. Springer Science & Business Media.