Kanditutkielma

Tietojenkäsittelytieteen kandiohjelma

Arkkitehtuurisuunnittelu ketterässä ohjelmistokehityksessä

Juuso Kotimäki

18.3.2021

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	18.3.2021	14 sivua, 1 liitesivua

Tiivistelmä — Referat — Abstract

Tähän tiivistelmä

Opiskelija Työn aihe luokitellaan ACM Computing Classification System (CCS) mukaisesti, ks. https://www.acm.org/about-acm/class, käyttäen komentoa `\classification{}`. Käytä muutamaa termipolkua (1–3), jotka alkavat juuritermistä ja joissa polun tarkentuvat luokat erotetaan toisistaan oikealle osoittavalla nuolella.

ACM Computing Classification System (CCS)
General and reference → Document types → Surveys and overviews
Applied computing → Document management and text processing → Document management → Text editing

Avainsanat — Nyckelord — Keywords

ulkoasu, tiivistelmä, lähdeluettelo

Säilytyspaikka — Förvaringsställe — Where deposited

Helsingin yliopiston kirjasto

Muita tietoja — övriga uppgifter — Additional information

Ohjelmistojärjestelmien erikoistumislinja

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Batchelor's Programme in Computer Science

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
Bachelor's thesis	March 18, 2021	14 pages, 1 appendix pages

Tiivistelmä — Referat — Abstract

Use this otherlanguage environment to write your abstract in another language if needed.

Topics are classified according to the ACM Computing Classification System (CCS), see https://www.acm.org/about-acm/class: check command `\classification{}`. A small set of paths (1–3) should be used, starting from any top nodes referred to bu the root term CCS leading to the leaf nodes. The elements in the path are separated by right arrow, and emphasis of each element individually can be indicated by the use of bold face for high importance or italics for intermediate level. The combination of individual boldface terms may give the reader additional insight.

ACM Computing Classification System (CCS)
General and reference → Document types → Surveys and overviews
Applied computing → Document management and text processing → Document management → Text editing

Avainsanat — Nyckelord — Keywords

ulkoasu, tiivistelmä, lähdeluettelo

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Software Systems specialisation line

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 tällaista etukäteissuunnittelua tulisi välttää.

Suurin vastakkainasettelu sijaitsee 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). Hyvällä arkkitehtuurisuunnittelulla vältytään suurelta määrältä uudelleenohjelmointia kehityksen aikaina (Eloranta, 2015). Arkkitehtuurisuunnittelun laiminlyöminen voikin johtaa erittäin monimutkaiseen ohjelmistoon: ongelmat voivat johtaa jopa ohjelmiston kehityksen estymiseen (Vogel et al., 2011).

Eri ketterien menetelmien kuvaukset eivät tyypillisesti kerro mitään arkkitehtuurin suunnittelusta (Babar et al., 2013). Tämän tutkielman tarkoituksena onkin auttaa lukijaa suhtautumaan arkkitehtuurisuunnitteluun oikealla tavalla erilaisissa ketterän sovelluske-

hityksen 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

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

2.1 Ohjelmistokehykset

Ohjelmistokehys (software framework) on konkreettinen tai konseptuaalinen alusta, joka toteuttaa yleiskäyttöisiä toiminnallisuuksia, joita voi erikoistaa tai korvata.

2.2 Referenssiarkkitehtuurit

Referenssiarkkitehtuuri on tietylle arkkitehtuurilliselle vaatimukselle syntynyt toimivaksi todettu ratkaisu. Referenssiarkkitehtuurit dokumentoivat järjestelmän rakenteen, järjestelmän rakennuspalaset sekä niiden vastuut ja vuorovaikutukset (Vogel et al., 2011).

Jos tilanteeseen sopivaa arkkitehtuuria ei ole käytettävissä, on sellainen usein muokattavissa olemassaolevasta referenssiarkkitehtuurista.

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 (LÄHDE) 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 (Eloranta, 2015).

Ideana on, että kävelevässä luurangossa kaikki arkkitehtuurin alijärjestelmien yhteydet ovat toteutettuna. Esimerkiksi kerrosarkkitehtuurina toteutetun web-sovelluksen 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 arkkitehti, ei ohjelmoija (Eloranta, 2015). Ongelmana tässä mallissa 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 esimerkiksi päättää, 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 on tarkoitus alustaa alkava kehitystyö eli hoitaa kaikki projektin aloituksen kannalta oleelliset toimenpiteet ennen varsinaista kehitystyön alkamista. Tässä yhteydessä Sprint 0 tarkoittaa 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).

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 sprintin 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).

Arkkitehtuurisuunnittelulle voidaan varata aikaa iteraatiosta tai sille voidaan varata koko iteraatio, arkkitehtuurisuunnittelua voi tapahtua muiden suunnitteluaktiviteettien (esim. daily scrum) yhteydessä, arkkitehtuuri voidaan tehdä myös kokonaan ohjelmoinnin yhteydessä (Rost et al., 2015).

Yksi toteuttaa arkkitehtuuri sprinttien aikana on määritellä arkkitehtuurisuunnittelu kuten käyttäjätarinat (user storyt) arkkitehtuuritarinoiksi. (Jensen, R. N., Møller, T., Sönder, P. ja Tjørnehøj, developer stories). Scrumissa voidaan myös toteuttaa oma arkkitehtuurisuunnitteluun tarkoitettu sprintti, ns. architecture sprint (Leffingwell). Kehityksen aikainen suunnittelu voi kohdistua arkkitehtuurillisesti tärkeiksi arvioituihin aspekteihin, jokaiseen user storyyn, jokaiseen epiciin, jokaiseen sprintiin tai koko tuotteeseen (Rost et al., 2015).

Elorannan (2015) mukaan tätä suunnittelukäytäntöä käytti menestyksekkäästi kokeneet tiimit. Kokeneemattomalla tiimillä arkkitehtuurin etukäteissuunnittelun puute johti jat-

kuvaan refaktorointiin ja lopulta siihen, että 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. Tämä taas määrittää 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. Eloranta (2015) mainitsee esimerkkinä monimutkaiset projektit, kuten työkoneiden ohjausjärjestelmät ja lääketieteellisten laitteiden ohjeArkkitehtuuriprototyyppi on toiminnallinen osa järjestelmästä, jolla on tarkoitus saada aikaista palautetta sidosryhmiltä. Prototyyppejä käytetään tyypillisesti suorituskyvyn, muokattavuuden ja rakennettavuuden analysointiin (Babar et al., 2013).

Spike solutioninka paljon arkkitehtuurisuunnittelua tulisi tehdä etukäteen, vaikuttaa ainakin riski, (odotettavissa oleva) vaatimusten epävakaus, tiimin kulttuuri ja osaaminen sekä myös asiakkaan suhtautuminen ketteryyteen.

4.1 Riskin vaikutus ja tunnistaminen

Arkkitehtuurisuunnittelun määrän tulisi määräytyä sen perusteella, että riski saadaan minimoitua riittävän tyydyttävälle tasolle (Fairbanks, 2010). Waterman, 2018b mukaan riskillä on suuri vaikutus siihen, kuinka paljon arkkitehtuurisuunnittelua tulisi tehdä etukäteen: mitä suurempi riski ja sen seuraukset, sitä aikaisemmin siihen pitäisi puuttua ja sitä yksityiskohtaisemmin arkkitehtuuri pitäisi suunnitella. Kyse on siis ketteryyden ja riskin välillä tasapainottelusta.

Mitä enemmän tiimi haluaa vähentää riskiä, sitä aikaisemmin päätökset pitää tehdä. Riskin vähentäminen vaikuttaa negatiivisesti tiimin ketteryyteen (Waterman, 2018b). Jos arkkitehtuurisuunnitteluun käytetään liikaa aikaa, arvon tuotto asiakkaalle viivästyy. Jos suunnittelua ei tapahdu etukäteen, on todennäköisyys epäonnistumiseen suurempi.

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

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.

Erityisesti Lean Startup -kontekstissa, jossa periaatteena on luoda tuote asiakkaan käyttäytymisestä tehtävien hypoteesien avulla, järjestelmän vaatimuksista ei ole mitään varmuutta . (Reis, 2011)

Mitä enemmän voi olettaa vaatimusten muuttuvan, sitä ketterämpi arkkitehtuurin tulisi olla eli sen pitäisi olla muutosta suvaitseva.

4.3 Aikainen arvontuotto

Jos tarkoituksena on aikainen arvontuotto, täytyy nopeuttaa ensimmäistä julkaisua vähentämällä arkkitehtuurisuunniitteluun käytettyä aikaa (Waterman et al., 2015). Tällöin voisi olla järkevää käyttää hyväksi malliarkkitehtuureja sekä inkrementaalista arkkitehtuuria.

4.4 Tiimin kulttuurin ja kokemuksen vaikutus

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).

Tiimin kokemus toistensa kanssa työskentelystä vaikuttaa: dokumentaation ja suunnittelun tarve vähenee, kun tiimin jäsenet tulevat kokeeneemmaksi toistensa kanssa työskentelysyä. Uusi tiimi tarvitsee enemmän etukäteisuunnittelua, kuin ketterän ajatusmallin sisäistänyt tiimi (Waterman et al., 2015).

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

Kokenut tiimi voisi valita strategiakseen suorittaa arkkitehtuurisuunnittelun kokonaan ke-

hityksen aikana. Elorannan (2015) mukaan kokeeneet tiimit käyttivätkin tätä lähestymistapaa menestyksekkäästi. Tiimeillä, joilla ei ollut kokemusta sovellusalueesta, strategia ei toiminut.

4.5 Projektin koko

Elorannan (2015) mukaan BDUF suunnittelu on tyypillistä varsinkin koodimäärän mukaan mitattuna suurissa projekteissa.

4.6 Asiakkaan vaikutus

(Waterman et al., 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.

Etukäteisbudjettien hyväksymisen tarve voi pakottaa tiimin suunnittelemaan arkkitehtuurin etukäteen, jotta he tietävät, paljonko heillä menee tuotteen valmistamiseen aikaa.

Joskus asiakkaalla ei ole aikaa osallistua projektiin, jolloin suunnitelmat pitää hyväksyttää etukäteen.

5 Yhteenveto

Elorannan (2015) mukaan ei ole selvää todistusainoeistoa, jonka mukaan jokin tietty asia vaikuttaisi yksinään siihen, mikä tapa toimia olisi paras.

6 Roskalaatikko

Kapseloinnin tarkoituksena on, että muutokset vaikuttavat vain mahdollisimman pieneen osaan järjestelmästä.

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

Ketterällä arkkitehtuurilla tarkoitetaan arkkitehtuuria, joka on suunniteltu ketterää prosessia käyttäen ja on muokattavissa, eli on muutosta suvaitseva (Waterman et al., 2015). Ketteränarkkitehtuurin tärkein ominaisuus on muutokseen sopeutuminen. Tämä aikaansaadaan käyttämällä hyväksi hyviä suunnittelukäytänteitä, kuten kapselointia ja selkeää vastuunjakoa, päätösten tekoa viivyttelemällä sekä suunnittelemalla arkkitehtuuri niin, että vaihtoehdoille jätetään tilaa (Waterman, 2018a).

Hyvistä suunnittelu/koodauskäytänteistä (Xp jne)

Päätösten viivyttelystä

Ketterä arkkitehtuuri tulisi suunnitella niin, että tulevaisuudessa eteen tulevat muutokset ja uudet vaatimukset ovat mahdollista sisällyttää sovellukseen. Tässä auttaa, kun tiedostaa asiat, joita saatetaan joutua muuttamaan myöhemmin sekä se, että vältetään arkkitehtuurin liiallista ennenaikaista optimointia jotain tiettyä käyttötarkoitusta varten (Waterman, 2018a).

Arkkitehtuuriprototyyppi on toiminnallinen osa järjestelmästä, jolla on tarkoitus saada aikaista palautetta sidosryhmiltä. Prototyyppejä käytetään tyypillisesti suorituskyvyn, muokattavuuden ja rakennettavuuden analysointiin (Babar et al., 2013).

Spike solution

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.

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.

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

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.

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.

Liite A Sample Appendix

usually starts on its own page, with the name and number of the appendix at the top. The appendices here are just models of the table of contents and the presentation. Each appendix Each appendix is paginated separately.

In addition to complementing the main document, each appendix is also its own, independent entity. This means that an appendix cannot be just an image or a piece of programming, but the appendix must explain its contents and meaning.