Kanditutkielma

Tietojenkäsittelytieteen kandiohjelma

Arkkitehtuurisuunnittelu ketterässä ohjelmistokehityksessä

Juuso Kotimäki

9.3.2021

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	9.3.2021	11 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 9, 2021	11 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 joukko päätöksiä koskien järjestelmän organisaatiota ja rakennetta. Arkkitehtuurilliset valinnat ohjaavat järjestelmän suunnittelua ja evoluutiota. IEEE:n standardin(Jen ja Lee, 2000) mukaan arkkitehtuuri pitää sisällään järjestelmän osat ja niiden keskenäiset suhteet sekä suhteet ympäristöön. Arkkitehtuuri määrittelee siis järjestelmän organisaation, rakenteelliset elementit sekä rajapinnat. Arkkitehtuurillisiin valintoihin vaikuttaa ainakin käyttökohde, joustavuus, uudelleenkäytettävyys ja ymmärrettävyys (Kruchten, 2004).

Ketterät menetelmät ovat nousseet suosituimmaksi ohjelmistotuotantoprosessiksi (*14th annual state of agile report* 2020). Ketterille menetelmille olennaista on sopeutuminen vaatimusten muutoksiin. Asiakkaalle arvoa tuottavaa toimivaa ohjelmistoa pidetään tärkeämpänä, kuin kattavaa dokumentaatiota, asiakkaan kanssa tehtäävää yhteistyötä tärkeämpänä kuin sopimusneuvotteluja ja muutokseen reagoimista tärkeämpänä, kuin suunnitelmien noudattamista (Fowler, Highsmith et al., 2001).

Ketterän ohjelmistokehityksen periaatteet ovat epäsuhdassa etukäteen tehtävän kattavan arkkitehtuurisuunnittelun kanssa.

Toisin kuin perinteisessä ohjelmistokehityksessä kuten vesiputousmallissa, jossa kaikki suunnittelu pyritään tekemään ennen ohjelmointia, on ketterissä menetelmissä käytäntönä suunnitella etukäteen vain sen verran, kun on välttämätöntä. Ihanteena onkin, että päätökset pyritään tekemään mahdollisimman myöhäisessä vaiheessa: mitä myöhemmin päätökset tehdään, sitä enemmän tietoa on hyödynnettävissä päätösten tueksi. Etukäteen tehdyt, turhiksi osoittautuneet suunnitelmat nähdään turhana työnä.

Ketterissä menetelmissä pyritään ohjaamaan resurssit mieluummin toimivan ohjelmiston tekemiseen, kuin dokumentaation tuottamiseen, mitä arkkitehtuurisuunnittelu voi olla. Ketterissä piireissä on kuitenkin hyväksytty arkkitehtuurisuunnittelun tärkeys laatua parantavana tekijänä. Oikealla tavalla suunniteltu arkkitehtuuri vähentää kehitykseen käytettyä aikaa ja kuluja.

Koska muutokseen reagoiminen on ketteryyden kantava voima, arkkitehtuurin pitäisi olla joustava muutoksen edessä. Tällöin puhutaan ketterästä arkkitehtuurista.

Arkkitehtuurisuunnittelusta ei juurikaan puhuta ketterissä projektinhallinnan viitekehyk-

sissä. Tämän tutkielman tarkoituksena onkin auttaa ketterää sovelluskehittäjää suhtautumaan arkkitehtuurisuunnitteluun oikealla tavalla.

Tutkielmassa kootaan menetelmiä, kuinka arkkitehtuurisuunnittelu yhdistetään osaksi ketterää ohjelmistotuotantoprosessia, käsitellään asioita, mitkä vaikuttavat menetelmän valintaan ja arvioidaan, miten tämä strategia vaikuttaa ohjelmistoprojektin onnistumiseen.

2 Ketterä ohjelmistokehitys ja ohjelmistoarkkitehtuuri

...

2.1 Ketterä ohjelmistokehitys ja arkkitehtuuri - haasteet

plaplaa ei kerrota mitään mistään yhyy

2.2 Ketterä arkkitehtuuri

Ketteryydessä on oleellista sopeutua vaatimusten muutoksiin. Ketterällä arkkitehtuurilla tarkoitetaan arkkitehtuuria, joka on suunniteltu ketterää prosessia käyttäen ja on muokattavissa, eli on muutosta suvaitseva (Waterman et al., 2015).

Ketterää arkkitehtuuria ei siis määritelmän mukaisesti suunnitella etukäteen: suunnitellaan vain se, mille on välttämätön tarve. Arkkitehtuuri todistetaan toimivaksi koodilla eli suunnittelu ja kehitys tapahtuu yhtäaikaisesti. Tällöin ei tarvita etukäteen tehtävää analysointia ja tuloksena on yleensä yksinkertaisin mahdollinen ratkaisu (Waterman, 2018a).

Ketterää arkkitehtuurin 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).

Hyvien suunnittelukäytänteiden käytön tarkoituksena on, että muutokset vaikuttavat vain mahdollisimman pieneen osaan järjestelmästä. Tämä ei varsinaisesti vähennä etukäteistyötä, mutta ketteryyden saavuttamiseksi näitä tulee kuitenkin käyttää (Waterman, 2018a).

Päätösten viivyttelyllä tarkoitetaan, että päätökset tehdään mahdollisimman myöhäisessä vaiheessa. Tällä maksimoidaan vaatimusten ymmärrys kertyneen kokemuksen ja asiakkaalta saadun palautteen vaikutuksena. Viivyttelemällä päätöksen tekoa päätöksiä tulee myös

tehtyä vähemmän, ja sitä vähemmän arkkitehtuuriin tulee tehtyä muutoksia (Waterman, 2018a).

Ketterä arkkitehtuuri tulisi myös suunnitella "vaihtoehtoja varten", eli niin, ettei suljeta mahdollisuuksia tulevaisuuden vaihtoehdoilta. Tässä voi helpottaa se, että tiedostetaan ne asiat, joita saatetaan joutua muuttamaan myöhemmin ja myöskin vältetään arkkitehtuurin liiallista ennenaikaista optimointia tiettyä käyttötarkoitusta varten (Waterman, 2018a).

3 Arkkitehtuurisuunnittelun ajoitus

Ohjelmistokehitystiimin pitää pitää tehdä valinta, kuinka ketterä näkökulma heidän on järkevintä ottaa projektinsa arkkitehtuurisuunnittelun lähtokohdaksi. Ketteryyteen vaikuttaa suoraan se, kuinka paljon etukäteissuunnittelua tulee tehdä. Valintaan vaikuttaa ainakin riski, (odotettavissa oleva) vaatimusten epävakaus, tiimin kulttuuri ja osaaminen sekä myös asiakkaan suhtautuminen ketteryyteen.

3.1 Riskin vaikutus

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. Arkkitehtuurisuunnittelun määrän tulisi määräytyä sen perusteella, että riski saadaan minimoitua riittävän tyydyttävälle tasolle (Fairbanks, 2010). Kyse on ketteryyden ja riskin välillä tasapainottelusta.

Mitä enemmän tiimi haluaa vähentää riskiä, sitä aikaisemmin päätökset pitää tehdä. Riskin vähentäminen vaikuttaa negatiivisesti tiimin kykyyn suunnitella ketterä arkkitehtuuri (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).

Tilanteissa, joissa järjestelmän vaatimuksista ei ole minkäänlaista varmuutta, esimerkiksi täysin uutta tuotetta tai ominaisuutta luodessa, voi olla tärkeämpää saada tuote ulos ja testattua mahdollisimman nopeasti. Jos tarkoituksena on aikainen arvontuotto, täytyy nopeuttaa ensimmäistä julkaisua vähentämällä arkkitehtuurisuunniitteluun käytettyä aikaa (Waterman et al., 2015). Tällöin voi olla järkevää käyttää hyväksi malliarkkitehtuureja sekä inkrementaalista arkkitehtuuria.

3.2 Riskin tunnistaminen

...

3.3 Vaatimusten epävakauden vaikutus

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 ketterämpi arkkitehtuurin tulisi olla eli sen pitäisi olla muutosta suvaitseva.

3.4 Muutokseen vastaaminen

...

3.5 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).

3.6 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.

4 Arkkitehturoinnin käytänteet

Arkkitehtuurisuunnittelun ajoituksen voi luokitella yleisesti täysin etukäteen tehtävään suunnitteluun, osittain etukäteen tehtävään suunnitteluun sekä täysin inkrementaaliseen arkkitehtuuriin. Lisäksi voidaan käyttää hyväksi malliarkkitehtuuria.

Arkkitehtuuri voidaan pyrkiä suunnittelemaan kokonaan etukäteen erillisessä arkkitehtuurisuunnitteluvaiheessa. Tällöin puhutaan Big design up front (BDUF) -suunnittelusta.

Osittain etukäteen tehtävässä suunnittelussa voidaan esimerkiksi omistetaan iteraatio (Scrumin sprintti) arkkitehtuurisuunnittelulle, jossa luodaan alustava arkkitehtuuri (minimun viable architecture). Tässä vaiheessa voidaan toteuttaa ns. Walking Skeleton. Tässä tutkielmassa käytetään tästä käytänteestä Scrumista tuttua nimitystä Sprint 0.

Inkrementaalisessa arkkitehtuurissa arkkitehtuuri luodaan iteraatioiden yhteydessä. Arkkitehtuurisuunnittelulle voidaan varata omia aikaikkunoita iteraatiosta, arkkitehtuurisuunnittelua voi tapahtua muiden suunnitteluaktiviteettien yhteydessä tai se voidaan tehdä kokonaan samanaikaisesti ohjelmoinnin yhteydessä Rost et al., 2015.

Arkkitehtuurisuunnittelu voi kohdistua arkkitehtuurillisesti tärkeiksi arvioituihin aspekteihin, jokaiseen user storyyn, jokaiseen epiciin, jokaiseen sprintiin tai koko tuotteeseen Rost et al., 2015.

Malliarkkitehtuurissa (template architecture) käytetään hyväksi esim. valmiita frameworkkeja.

4.1 Big Design Up Front

4.2 Sprint 0

4.3 Inkrementaalinen arkkitehtuuri

4.4 Malliarkkitehtuurit

5 Arkkitehdin rooli

5.1 Dokumentaatio

6 Yhteenveto

Lähteet

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

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.

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.

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.