Ohjelmistojen testattavuuden parantaminen arkkitehtuurin avulla

Kristian Wahlroos

Kandidaatintutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 23. helmikuuta 2016

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Matemaattis-luonnontieteellinen	Tietojenkäsittelytieteen laitos

Tekijä — Författare — Author
Kristian Wahlroos

Työn nimi — Arbetets titel — Title

Ohjelmistojen testattavuuden parantaminen arkkitehtuurin avulla

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Kandidaatintutkielma	23. helmikuuta 2016	4

Tiivistelmä — Referat — Abstract

tiivistelmä (100-200 sanaa; kenelle, miksi, millaisessa ympäristössä; tutkimuskysymys; tulokset; impakti)

Avainsanat — Nyckelord — Keywords
avainsana 1, avainsana 2, avainsana 3

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Sisältö

1 Johdanto

Esimerkkilause ja lähdeviite [**?**] ja [**?**]. jäsennelty asianmukaisesti (kenelle, miksi, millaisessa ympäristössä; ratkaisun lähestymistapa; tutkimuskysymys, tulokset ja impakti; loppulukujen roolitus).

2 Body text

purkaa auki teknisemmälle yleisölle samalla käsitteitä määritellen tutkimuskysymyksen, ratkaisuille ympäristöstä esiintulevat vaatimukset ja keskeiset käsitteet näiden asioiden esittelemiseksi.

antaa riittävästi termistöä ja kysymyksenasetteluja, jotta myöhempänä teksissä on mahdollisuus analyysiin ja evaluointiin. Keskimmäiset luvut keskittyvät kukin "opettamaan"lukijalle yhden osakysymyksen tai näkökulman käsiteltävään asiaan. Lukujen työnjaon tulisi olla selkeä ja niillä tulee olla selkeä oma rakenteensa, jonka valintaperuste on lukijalle kerrottu.

Viimeistä edellinen luku on varsinaisen kontribuution koti: vertailu, soveltaminen, osien synteesi ja mahdolliset evaluointimenetelmät ja -tulokset päärooliss a.

3 Ohjelmistoarkkitehtuuri

Jeejee. Paltu.

3.1 Ohjelmistojen arkkitehtuuri käsitteenä

Arkkitehtuuri ohjelmistoissa on käsite, josta moni kehittäjä on tietoinen mutta joka ei ole kovin yksiselitteinen eikä siitä ole yksimielisiä määritelmää [**?**]. Arkkitehtuuri voidaan kuitenkin nähdä karkeasti neljästä eri näkökulmasta [**?** , s. 2-7]: ohjelmiston rakenteen kuvaajana, kuvaajana ohjelmiston komponenttien välisille suhteille, mallina joka ottaa huomioon ei-toiminnalliset (non-functional) vaatimukset ja yleisenä abstraktiona.

Rakenteen kuvaajana arkkitehtuuri määrittelee ohjelmistojärjestelmän sisäistä rakennetta, joka koostuu useista komponenteista sekä moduuleista. Erilaiset moduulit tekevät tiettyä työtä ja täten erilaiset vastuut on jaettu ohjelmistojärjestelmän sisällä loogiisiin kokoelmiin ja näistä syntyvät kokoelmat tarjoavat halutun toiminnallisuuden. Komponenttien välisen kommunikaation mallintaminen tulee vastaan, kun ohjelmistojärjestelmää jaetaan erilaisiin komponentteihin. Tavoitteena on kertoa mallintamalla, että mitkä komponentit tai moduulit ovat vuorovaikutuksessa toistensa kanssa ja millä tavoin. Yleisin kommunikaatiotapa on esimerkiksi suorat funktiokutsut komponenttien välillä. Ei-toiminnaliset vaatimukset tulevat esille vasta arkkitehtuurin avulla ja arkkitehtuurin mallinnuksen avulla voidaan määrittää

miten ohjelma suorittaa sille määrättyä tehtävää sen sijasta, että mallinnettaisiin mitä ohjelma tekee. Arkkitehtuurin avulla on mahdollista myös abstrahoida osakkaille (stakeholder) ja muille kehittäjille ohjelmistojärjestelmää helpommin lähestyttäväksi, jolloin kommunikaatio eri osapuolten välillä helpottuu. Abstrahoinnin avulla pystytään yksinkertaistamaan järjestelmän konkreettista toteutusta ja suorittamaan arkkitehtuurista erittelyä (architectural decomposition), jossa muodostetaan epärelevanteista komponenteista mustia laatikoita (black box). Mustien laatikoiden ideana on piilottaa komponenttien sisäistä toteutusta eri tasoilla, jolloin arkkitehtuurista voidaan muodostaa eri abstraktiotason malleja.

Hieman samanlainen jaottelu on määritelty myös Solmsin tutkimuksessa [**?** , s. 369]. Siinä ohjelmistoarkkitehtuuri-määritelmä jaetaan kolmeen eri määrittelyluokkaan: korkean tason abstraktioon ohjelmistojärjestelmästä, rakenteita sekä ulkoisia näkyviä ominaisuuksia korostavaan määritelmään ja peruskäsitteistöön sekä rajoitteisiin joiden puitteissa ohjelmistojärjestelmää kehitetään. Näistä kaksi ensimmäistä määritelmää vastaavat samoja kuin [**?** , s. 2-7], mutta tarkentaen kumpaakin omat kuvaustavat. Korkean tason abstraktioita mallinnetaan erilaisten näkymien kautta tai käyttämällä arkkitehtuurallisia malleja, jotka IEEE on määritellyt [**?**]. Rakenteita korostettaessa mallinnuksessa käytetään UML-kaavioita, koska UML määrittelee ohjelmiston arkkitehtuuria osina, joita pystytään rekursiivisesti tarkentamaan haluttaessa. Tarkennuksen avulla osat pystytään kuvaamaan kommunikoivan keskenään erilaisten rajapintojen kautta, osia yhdistäviä suhteita pystytään tarkastelemaan ja erilaisia rajoitteita pystytään luomaan osien välille. Kolmas määritelmä on paljon laajempi näkemys ohjelmistoarkkitehtuurista, koska se kuvaa ohjelmistojärjestelmän peruskäsitteistön ja ominaisuudet siinä ympäristössä, jossa ne ilmeentyvät elementtien, suhteiden ja suunnittelun periaatteiden kautta. Peruskäsitteistö tarjoaa sen käsitteistön, jonka avulla sovelluslogiikka voidaan määritellä. Ominaisuudet liittyvät usein ohjelmistojärjestelmän laadullisiin ominaisuuksiin. Periaatteet voidaan nähdä järjestelmän keskeisinä suunnittelurajoitteina (core design constraints), joiden kokonaisuus muodostaa järjestelmän arkkitehtuurisen tyylin. Ohjelmistojärjestelmän arkkitehtuurin tyyli voi esimerkiksi olla väylät ja suodattimet (pipes and filters).

Näistä kahdesta eri jaottelusta ohjelmistoarkkitehtuuriin voidaan nähdä merkittävinä yhtäläisyyksinä tarpeen kuvata ohjelmiston rakenteellisuutta, komponenttien välistä kommunikaatiota ja laadullisia vaatimuksia. Kaikki nämä edellämainitut ominaisuudet on saatava upotettua ohjelmistojärjestelmästä luotavaan arkkitehtuuriseen malliin niin, että ne vastaavat muunmuassa seuraaviin kysymyksiin [**?** , s. 31 - 33]: mitkä ovat arkkitehtuurisi toiminnalliset elementit; miten nämä elementit kommunikoivat keskenään ja ulkomaailman kanssa; mitä tietoa käsitellään, talletetaan ja esitetään; ja mitä fyysisiä ja ohjelmallisia elementtejä tarvitaan tukemaan näitä elementtejä. Kuitenkaan arkkitehtuurista luotava malli ei saisi olla monoliittinen

malli, joka pyrkii kuvamaan kaiken yhdessä mallissa, koska arkkitehtuuria ei ole mahdollista kuvata vain yhden mallin avulla. Tämä johtuu siitä, että monoliittinen malli on erittäin vaikea ymmärtää, siitä on vaikeaa löytää arkkitehtuurin tärkeimmät ominaisuudet (features) ja se on usein puutteelinen, jäljessä eikä vastaa enää nykyistä ohjelmistojärjestelmää. Ratkaisu tähän on jakaa malli useisiin toisiinsa liittyviin näkymiin (views), jotka esittävät jokainen yhden näkökulman (viewpoint) järjestelmän arkkitehtuuriin keräämällä yhteen toiminnalliset piirteet sekä laadulliset ominaisuudet [**?** , s. 33-34; **?** , s. 8-9]. Tämän avulla voidaan tarkastella saavuttaako järjestelmä sille asetetut tavoitteet.

3.2 Näkymät

Arkkitehtuuristen näkymien tehtävänä on kuvata ne näkökannat arkkitehtuurista, jotka ovat merkityksellisiä itse asialle, jota näkymä haluaa painottaa [**?**]. Yksi tunnetuimmista määritelmistä arkkitehtuuriselle näkymälle on Krutchenin 4+1 näkymämalli (4+1 View Model) [**?** , s.7]. Siinä arkkitehtuuri kuvataan neljän näkymän avulla: looginen, prosessi, fyysinen ja kehitys. Looginen näkymä kuvailee esimerkiksi luokkakaavioiden avulla ohjelmistojärjestelmän elementtejä ja niiden välisiä suhteita tarkentaen järjestelmän rakennetta, prosessinäkymä keskittyy ajonaikaisen suorituksen kuvaamiseen tarkentamalla muunmuassa miten samanaikaisuuden hallinta tapahtuu järjestelmässä, fyysinen näkymä keskittyy siihen miten järjestelmän eri komponentit kuvautuvat fyysiselle laitteistolle jossa komponenttia ajetaan ja lopuksi kehitysnäkymä (development view) keskittyy järjestelmän sisäiseen toteutukseen tarkemmalla tasolla kuvailemalla sisäkkäisia pakkauksia tai luokkahierarkiaa. Jokainen näkymä voidaan liittää osaksi toista näkymää skenaarioiden avulla, jotka heijastelevat järjestelmälle asetettuja vaatimuksia.

3.3 Arkkitehtuuri osana ohjelmistonkehitystä

3.4 Vaatimukset

Vaatimusten keräys vs ei-toiminnalliset vs toiminnalliset.

3.5 Historia

4 Ohjelmistojen laadulliset tekijät

4.1 Arviointi

4.2 Tradeoffit

5 Testattavuus laadullisena tekijänä

5.1 Testattavuuden merkitys

6 Arkkitehtuurin vaikutus testattavuuteen

6.1 Jotain

7 Yhteenveto

Yleensä hieman johdantoa lyhyempi.

Muistuttaa mieleen tutkimuskysymyksen, mainitsee tärkeimmät tulokset ja niiden perusteet. Keskittyy impaktiin ja esimerkiksi suosituksiin. Ei vain summeeraa luku kerrallaan aikaisempaa tekstiä.