Ohjelmistoarkkitehtuurista puoliketterästi

Simo Vuorinen, TietoEnator

Ohjelmistoarkkitehtuurista on kirjoitettu useita kirjoja liittyen sen suunnittelun eri näkökulmiin, kuten suunnitteluun, dokumentointiin, ja arviointiin. Tämä artikkeli käsittelee ohjelmistoarkkitehtuuria arkkitehdin näkökulmasta esittäen perustavaa laatua olevia asioita liittyen ohjelmistoarkkitehtuurin suunnitteluun ja ohjelmiston toteutuksen aikaiseen arkkitehtuurin valvontaan.

Ohjelmistoarkkitehtuurin määritelmistä

Arkkitehtuuri-termille löytyy useilta eri tahoilta määritelmiä pienillä vivahde-eroilla höystettynä. Näistä vivahde-eroista on kuitenkin mahdollista halkoa hiuksia näin tahtoessaan vaikka kuinka. Olen tyytynyt ohjelmistoarkkitehtuurin osalta seuraavaan määritelmään:

”Arkkitehtuurilla (Software architecture) tarkoitetaan järjestelmän tai sovelluksen rakennetta tai rakenteita, jotka koostuvat ohjelmistokomponenteista, niiden näkyvistä attribuuteista, ja niiden välisistä suhteista.” [Bass et al]

Eli käytännössä ohjelmistoarkkitehtuuri koostuu ohjelmistokomponenteista ja niiden attribuuteista, sekä ohjelmistokomponenttien välisistä suhteista, jotka muodostavat ohjelmiston ja sen rakenteen.

Vaatimuksia ohjelmistoarkki–tehtuurille

Sen lisäksi, että arkkitehtuuri on järjestelmän kuvaus, on sen palveltava myös käytännöllisellä tasolla. Ei riitä, että arkkitehtuuri kuvaa järjestelmän komponentit, järjestelmän toiminnalliset ja laadulliset vaatimukset. Sen on myös mahdollistettava ohjelmiston ongelmakohtien löytäminen ennen toteutukseen siirtymistä. Lisäksi hyvän arkkitehtuurin on oltava

- helposti testattavissa toiminnallisten vaatimusten osalta
- helposti mitattavissa laadullisten vaatimusten osalta
- helposti valvottavissa ja hallittavissa.

Testattavuus

Jotta arkkitehtuuri olisi helposti testattavissa toiminnallisten vaatimusten osalta, olisi hyvä jo vaatimusten kirjaamisvaiheessa kiinnittää jokaiseen vaatimukseen testi, jolla voidaan varmistaa vaatimuksen toteutuminen [Sommerville et al]. Kun vaatimuksista päästään toiminnallisiin kuvauksiin asti, on jo kohtalaisen helppo kirjata testi, jolla toiminta voidaan myöhemmin varmistaa. Vaatimus on kelvollinen vaatimukseksi vasta sitten, kun

Huolenaihe	Laatuattribuutti	Mittari	Hyväksymiskriteeri
Kuinka nopeasti ohjelmiston komponentti voidaan vaihtaa toiseksi?	Muokattavuus	Komponentin vaihtoon kuluva aika	Komponentin vaihtoon kuluu max 2h.
...	...	...	...

Taulukko1: Laatuattribuutti ja sen mittarit

voidaan sopia, miten se testataan ja mitataan.

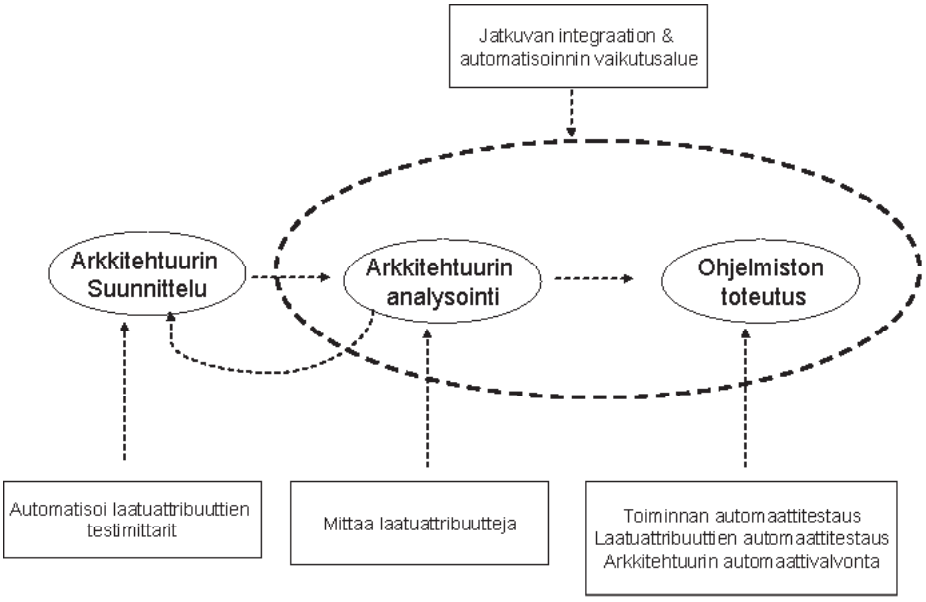
Näitä toiminnallisia testejä voidaan myöhemmin automatisoida sekä käsin kirjoitetuin yksikkötestein että erilaisilla testiroboteilla.

Mitattavuus

Laatuvaatimukset vaativat usein melko paljon työtä, jotta niistä saadaan mittauskelpoisia. Usein ne ovat aluksi lähinnä löyhiä laatutavoitteita, jotka on puserrettava vaatimuksiksi sopimalla niiden mittaustapa ja –kriteeri, jolla vaatimus on hyväksyttävissä. Taulukko 1 valaisee asiaa esimerkinomaisesti.

Laatuvaatimusten ongelmana on usein myös niiden keskinäiset ristiriitaisuudet. Ristiriidoista on päästävä yhteisymmärrykseen ennen hyväksymiskriteereistäpäättämistä. On ymmärrettävä kuinka laatuvaatimukset vaikuttavat toisiinsa, ts. aiheuttaako jonkin laatuattribuutin muokkaus negatiivisen tai positiivisen muutoksen johonkin toiseen laatuattribuuttiin. Usein joudutaan kuitenkin tapauskohtaisesti pohtimaan laatuattribuuttien vaikutusta toisiinsa.

> “Vaatimus on kelvollinen vaatimukseksi vasta sitten, kun voidaan sopia, miten se testataan ja mitataan.”



Kuva 1: Jatkuvan integraation ja automatisoidun testauksen suhde arkkitehtuurin suunnitteluun

Valvottavuus

Arkkitehtuuria ja sen toteutumista on seurattava koko arkkitehtuurin suunnittelu- että ohjelmiston toteutusvaiheen ajan. Arkkitehtuurin valvontaa voidaan hoitaa seuraavin keinoin:

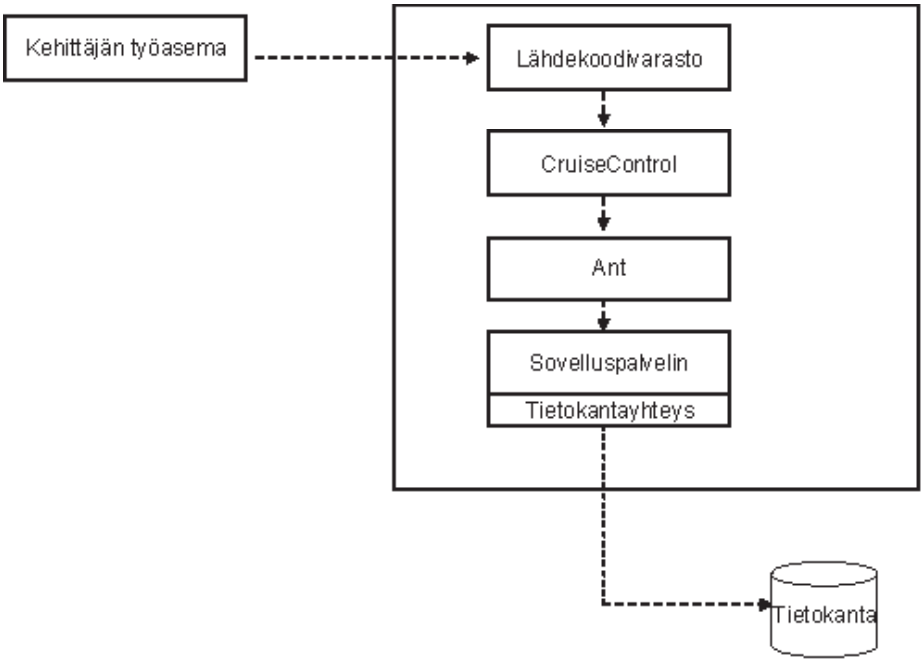
- Kattava automatisoitu testihaarniska toiminnallisille testeille
- Kattava automatisoitu testihaarniska laatuvaatimuksille
- Arkkitehtuurirakenteiden automaattinen valvonta
- Ohjelmakoodin staattinen analyysi
- Koodin ja dokumentaation katselmukset

Kaikki muut näistä ovat automatisoitavissa paitsi katselmoinnit. Projektien vauhdin lisääntyessä on nykyään vaikeaa saada varattua aika, jolloin ihmiset saadaan istumaan saman pöydän ääreen. Tämän vuoksi markkinat ovatkin pullollaan erilaisia koodin ja binäärin analysointivälineitä. Nämä välineet ovat varsin moni-

puolisia, joten niitä hyödyntämällä katselmointityö vähenee.

Käytäntöä

Jatkuva integraatio-prosessi [Fowler] yhdistettynä Test-Driven Development-menettelyyn [Beck] on todettu käytännössä toimivaksi yhdistelmäksi. On mielekkäämpää käyttää viikko tai pari arkkitehtuurin suunnitteluvaiheessa ohjelmiston kehittämis- ja valvontaprosessin automatisointiin kuin tuhlata viikkotolkulla aikaa käsityönä ohjelmiston tai sovelluksen julkaisemiseen ja laadunvarmistamiseen. Jatkuvan integraation ympäristön voi pystyttää arkkitehtuurin suunnitteluvaiheessa, jolloin sitä voidaan käyttää ennen ohjelmiston toteutuksen aloittamista laatuattribuuttien mittaamiseen.



Kuva 2: Esimerkki jatkuvan integraation ympäristöstä.

”Arkkitehtuuria ja sen toteutumista on valvottava koko arkkitehtuurin elinajan.”

Jatkuvan integraation pyörittämiseen löytyy useita open source-työkaluja, kuten esim. CruiseControl. Ko. välineet käynnistävät ajastuksella ohjelmiston rakentamisen ja julkaisevat rakentamisen lopputulokset. Jos rakentaminen epäonnistuu, ne lähettävät haluttaessa sähköpostin sovituille henkilöille. Mikäli yksikkötestit on kirjoitettu, nämä voidaan suorittaa automaattisesti jokaisen rakentamiskierroksen yhteydessä.

Yksikkötestien lisäksi kannattaa kirjoittaa myös integraatio- tai ”savutestit”, joilla voidaan varmistaa kokonaisuuden toiminta. Ja koska testikoodi päätyy lähdekoodivarastoon samaan aikaan kuin ohjelmistokoodi, tiedetään paremmalla tarkkuudella jo toimivat piirteet. Jos testikoodin rungot toimimattomien ja toteuttamattomien piirteiden osalta viedään lähdekoodivarastoon aiemmin kuin ohjelmistokoodi, tiedetään myös jäljellä olevat toteuttamattomat piirteet paremmin. Tällaisen testauksen etuna on testien automaattisen suorituksen lisäksi se, että yksikkötestit yleensä parantavat ohjelmiston koodin laatua pakottamalla kehittäjän modularisoimaan ja refaktoroimaan ohjelmakoodia paremmaksi.

Kehikkoon on mahdollista kiinnittää myös koodin testauksen laajuuden mittaus (code coverage) ja lähdekoodin sekä binäärin staattisen ja dynaamisen analyysin apuvälineet, joilla voidaan varmistaa sekä lähdekoodin laatu että arkkitehtuurin eheys. Myös keveähkö suorituskykymittaus ja erilaiset profiloinnit ovat automatisoitavissa.

Olemassaolevilla apuvälineillä ohjelmistoarkkitehdille jää huomattavasti enemmän aikaa keskittyä muihin tärkeisiin arkkitehtuurin suunnittelun seikkoihin. Myös ohjelmiston testaushenkilöstön työkuormaa voidaan jonkin verran keventää testaamalla perusasiat automaattisesti.

Lisäksi automatisoitu rakennusympäristö helpottaa toteutuksen aikaista seurantaa, koska jatkuvan integraation välineet voivat julkaista käännös-, testaus-, mittaus- ja valvontaraportit. Nämä

ovat silloin helposti ohjelmistosta vastaavien henkilöiden saatavilla.

Ohjelmistoarkkitehtuurin hallinnointiprosessista

Arkkitehtuuria ylläpidetään ja päivitetään läpi järjestelmän tai ohjelmiston eliniän. Jos ohjelmiston arkkitehti vaihtuu, on arkkitehtuurivastuu siirrettävä selkeästi toiselle henkilölle. Kuten muidenkin vastuiden osalla, myös arkkitehtuurivastuu on oltava jollakulla nimetyllä henkilöllä, eikä sitä voida kollektiivisesti jakaa.

titeetteihin. Jos mitään estettä ei uudelle vaateelle löydy, päästään laskemaan työmääräarvioita.

Yhteenveto

Ohjelmistoarkkitehtuurilla tarkoitetaan ohjelmiston komponentteja, ohjelmistokomponenttien näkyviä attribuutteja, ja niiden välisiä suhteita. Ohjelmistoarkkitehtuurin tarkoitus on varmistaa ohjelmiston toiminta yrityksen tai organisaation toiminnan tukena kaikkien sidosryhmien osalta, huolehtia ohjelmiston yhteensopivuudesta

> ”Jonkin strategisen arkkitehtuurikokonaisuuden hallinnointiprosessi voi olla melkoisen monimutkainen ja aikaa vievä.”

Arkkitehtuurin hallinnoinnista vastaavat hallinnointiprosessissa määrätyt henkilöt. Arkkitehtuurin hallinnointiprosessi kuvaa tavat, joilla arkkitehtuuriin ja ohjelmistoon tehdään lisäyksiä, muutoksia tai poistoja. Riippuen arkkitehtuurin strategisesta merkityksestä tämä prosessi vaihtelee paljonkin. Jonkin strategisen arkkitehtuurikokonaisuuden hallinnointiprosessi voi olla melkoisen monimutkainen ja aikaa vievä, kun taas vähemmän kriittisessä ohjelmistossa tai järjestelmäkokonaisuudessa se voi käytännössä olla vain tuotepäällikön tai järjestelmän omistajan ilmoitus uudesta piirteestä tai prosessista arkkitehdille. Tämän jälkeen ohjelmiston arkkitehti tarkistaa piirteen vaikutukset ohjelmistoon ja muihin sidosen- ja kommunikointikyvystä muiden sisäisten ja ulkoisten järjestelmien kanssa, ja mahdollistaa ohjelmiston ongelmakohdat löytäminen ennen siirtymistä toteutukseen. Lisäksi sen on oltava helposti testattavissa toiminnallisten vaatimusten osalta, helposti mitattavissa laadullisten vaatimusten osalta, ja helposti valvottavissa ja hallittavissa. Ohjelmistoarkkitehtuurin valvontaa ja laatua voidaan osin parantaa jatkuvan integraation käytännöllä sekä Test-Driven Development-menettelyllä. Automatisoituun valvontaan voidaan lisätä myös muiden laatuattribuuttien mittaus. Automatisoitua rakennus- ja mittausympäristöä käyttämällä voidaan parantaa sekä arkkitehtuurin että ohjelmiston laatua ja säästää arkkitehtuurin eheyden ja ohjelmiston laadun tarkkailuun kuluvaa aikaa. Arkkitehtuurin hallinnointiprosessi kuvaa tavat, joilla arkkitehtuuriin ja ohjelmistoon tehdään lisäyksiä, muutoksia tai poistoja.

Lähdeluettelo ja luettavaa

Agile Manifesto. http://agilemanifesto.org

[Bass et al]: Software Architecture in practice, 2nd ed. L.Bass, P.Clements, R.Katzman. Addison-Wesley 2003.

[Beck]: Test-Driven Development: By Example. K. Beck. Addison-Wesley, 2003.

[Fowler]: Continuous Integration. http://www.martinfowler.com/articles/continuousIntegration.html

[Sommerville et al]: Requirements Engineering: A good practice guide. I. Sommerville, P.Sawyer. Wiley 2000.

Planning Extreme Programming. K. Beck, M.Fowler. Addison-Wesley 2001.

Simo Vuorinen toimii konsulttina TietoEnator Telecom & Mediassa erikoisalueenaan ohjelmistoarkkitehtuurit.