Ohjelmistoarkkitehtuurit, syksy 2012
Harjoitus 1 (12.9.-13.9.)

Kurssikirjasta on ensimmäisellä kurssiviikolla (4.9-6.9.) käsitelty suunnilleen sivut: 1-42 (kursorisesti, ei esimerkkejä), 57-94.

Tämän viikon tehtävät ovat keskustelutehtäviä. Aiheista keskustellaan pienryhmissä ja kukin ryhmä esittää yhteenvedon keskusteluista (tärkeimmät kohdat). Valmistaudu keskustelemaan aiheista esimerkiksi kirjaamalla muistiin oleellisina pitämiäsi asioita aiheesta. Tehtävissä esiintyvät artikkelit ovat osa tenttiin liittyvää materiaalia.

Tehtävien vastauksia ei välttämättä löydy luentomateriaalista tai kurssikirjasta. Käytä nettiä tai pohdi itse.

---

1. Mitä ongelmia huonosti määritelty arkkitehtuuri voi aiheuttaa toteutusvaiheessa? Millainen ohjelmistokehitysprosessin tulisi olla, jotta näiltä ongelmilta vältyttäisiin?

Ongelmista:

- Sykliset riippuvuudet -> vaikeuttavat bugien korjausta, ohjelmiston ylläpidettävyys heikkenee
- Huonot designratkaisut yhteisen vision puuttuessa
- Hallitsemattomat/huonot arkkitehtuurimuutokset kehityksen aikana
- Hankala tulla mukaan uuden kehittäjän, jos arkkitehtuuri epäselvä
- Työnjaon epäselvydet
- Tehdään turhaa työtä -> aikataulut venyvät
- Puutteellinen modularisointi/tietomallinnus -> samoja asioita toteutetaan moneen kertaan
- Komponenttien heikko uudelleenkäytettävyys
- Kriittisten ongelmien myöhäinen havaitseminen -> muutosten tekeminen hyvin kallista -> häkätään
- Asioita jää dokumentoimatta, jos ei alunperinkään ole kunnollisia kuvauksia
- Sama luonteeltaan globaali suunnitteluongelma ratkaistu moneen kertaan periaatteessa oikein mutta eri tavoin lokaalisti (eri tiimeissä) -> keskenään kilpailevia ratkaisuja, mikä vaikeuttaa järjestelmän jatkokehitystä
- Vesiputousmallin huonot puolet korostuvat (jos sen tapaista prosessia noudatetaan)
- Toteutettujen komponenttien yhteensopivuusongelmat
- Turha päällekkäinen suunnittelutyö, jos arkkitehtuuri ei ota kantaa suunnitteluongelmiin, jotka toistuvat monissa komponenteissa

Prosessista:

- Arkkitehtuuri elää projektin aikana -> tähän varauduttava
- Laadulliset asiat huomioitava projektin alkuvaiheessa
- Myös arkkitehtuuria voidaan kehittää iteratiivisesti -> tarkistuspisteet prosessin vaiheissa
- Arkkitehti (rooli) osana kehitystiimiä -> arkkitehtuuri pois norsunluutornista
- Prosessissa pitää olla aikaa maksaa tekninen velka
- Hyvin kuvattu kehitysprosessi ja toimiva kommunikaatio helpottavat arkkitehtuuriongelmia tai arkkitehtuurin puutetta

2. Ketterässä ohjelmistokehityksessä pyritään välttämään monimutkaista dokumentaatiota ja toteuttamaan vain kunkin iteraation kannalta välttämättömät asiat ilman ylimäärää (engl. *waste*). Miten arkkitehtuurityö istuu tällaiseen prosessiin? Onko ketteryys ongelma arkkitehtuurin laatimisen kannalta tai päinvastoin arkkitehtuurin laatimistarve prosessin ketteryyden kannalta. Millaisia ongelmia voi esiintyä ja löytyykö niihin ratkaisuja. Pohdinnassa voit tukeutua seuraaviin asiaa hieman eri kannoilta tarkasteleviin artikkeleihin:

- Diego Fontdevila, Grupo Esfera: Software Architecture in the Agile Life Cycle, 2010 (http://msdn.microsoft.com/en-us/architecture/ff476940.aspx)

- Pekka Abrahamsson, Muhammad Ali Babar, Philippe Kruchten, "Agility and Architecture: Can They Coexist?," *IEEE Software*, vol. 27, no. 2, pp. 16-22, Mar./Apr. 2010 (http://www.computer.org/portal/web/csdl/doi/10.1109/MS.2010.36) Artikkeli ladattavissa vain laitoksen sisältä.

- Perusjännite *Big Design Up Front (BDUF)* < -- > *Emergence*
- Jokin arkkitehtuuri oltava joka tapauksessa
- Arkkitehtuuria ei määritellä alussa supertarkasti, vaan annetaan sen muuttua projektin aikana
- Dokumentointia ”reaaliajassa” ei jälkikäteen
- Konteksti vaikuttaa, asiakkaan vaateet -> esim julkishallinnon kilpailuttama projekti
- Kehittäjien rooli ja osallistumishalukkuus arkkitehtuurityöhön -> kehittäjät ymmärtävät mitkä asiat ovat arkkitehtuuriluonteisia ja vaativat koordinointia ja yhteistä näkemystä
- Kevytarkkitehtuuri alkuun, sitten iteraatiot, arkkitehtuuri-impaktin tarkistus iteraation lopussa
- Iteraation *alussa* valitaan toteutettavat taskit ja tarkistetaan siinä vaiheessa mahdollinen arkkitehtuuri-impakti
- Kehitystiimien yli vaikuttavat arkkitehtuuriasiat vaativat jonkin tiimien ulkopuolisen koordinaatiot -> muuten syntyy helposti kilpailevia, sinänsä hyviä, ratkaisuja samaan ongelmaan
- Valitut teknologiat ja työkalut tarjoavat yleensä sovelluksen perusarkkitehtuurin -> monet arkkitehtuuriratkaisut tulevat valmiina -> pitää tehdä valintoja, jos työkalut tarjoavat vaihtoehtoisia ratkaisuja asioiden toteuttamiseen -> miten työkalun uusien versioiden tuomat paremmat ratkaisut saadaan tuotteeseen mukaan (miten perustellaan liiketaloudellisesti parannusten käyttöönotto)

3. Alustava arkkitehtuuri laaditaan toiminnallisten perusvaatimusten pohjalta, laadulliset (ei-toiminnalliset) vaatimukset vaikuttavat kuitenkin tyypillisesti enemmän. Miksi?

Huom – tehtävän muotoilu oli hämäävä. Tarkoituksena oli kysyä, miksi laadulliset vaatimukset vaikuttavat arkkitehtuurin suunnitteluun tyypillisesti enemmän kuin toiminnalliset.

- Toiminnalliset helppo toteuttaa ja määritellä, laadullisillekin voi löytyä tarkat vaatimukset (asiakkaan määrittelemä standardi)

- Laadullinen vaatimus koskettaa joukkoa komponentteja, jolloin sitä ei voi toteuttaa vain paikallisesti yhden komponentin sisäisellä suunitteluratkaisulla -> pakko miettiä arkkitehtuurin tasolla
- Loppukäyttäjien käyttökokemus ja kehittäjien työkokemus ratkaisee ohjelmiston menestyksen -> tyypillisesti näihin vaikutetaan nimenomaan laadullisilla ominaisuuksilla -> koskettavat useita komponentteja tai järjestelmää kokonaisuutena
- Laadullinen vaatimus saattaa karsia suuren määrään suunnittelumalleja ja ratkaisuvaihtoehtoja pois -> suuri vaikutus arkkitehtuuriin
- Tuoteperheet -> suunniteltava etukäteen uudelleenkäytettävyys ja varioituvuus arkkitehtuuriin -> miten saadaan mahdollisimman vähällä työllä toteutettua tuotevariantteja samasta perusrungosta ja komponenttikokoelmasta