Ohjelmistoarkkitehtuuri ja kehitysprosessit

Luento 2

Oppimistavoitteet

- Kuinka ”paljon” arkkitehtuuria tarvitaan?
- Arkkitehtuuri ohjelmistokehitysprosessissa
- Ohjelmistoarkkitehdin tiedot ja taidot

KUINKA PALJON ARKKITEHTUURIA?

Kaikilla ohjelmistoilla on arkkitehtuuri

- Arkkitehtuuriset suunnittelupäätökset syntyvät jossain ohjelmistokehitysprojektin aikana - tehtiin ne tietoisesti tai ei
- Arkkitehtuuriratkaisut ovat periaatteessa tärkeitä projektien ja ohjelmistojen onnistumisen kannalta, mutta käytännöissä on paljon vaihtelua
- Kurssikirjassa Fairbanks tunnistaa kolme eri lähestymistapaa arkkitehtuurin suunnitteluun ja käyttöön

1. Arkkitehtuuri on yhdentekevä *(indifferent)*

- Monissa projekteissa ei arkkitehtuurityötä juuri tehdä eikä arkkitehtuuria erikseen suunnitella (edes uuskehityksessä)
- Syitä
  - Tietämättömyys – mennään tuurilla
  - Pieni projekti ja/tai pienet riskit - mikä vaan todennäköisesti toimii
  - *Oletusarkkitehtuurin* käyttö (presumptive architecture)

Oletusarkkitehtuurit

- Monilla toimialoilla on järjestelmä- ja ohjelmistotoimittajia, jotka ovat vakiinnuttaneet omat teknologiansa ja arkkitehtuuriratkaisunsa alan standardeiksi (*de facto*)
  - Vanha viidakon sanonta: *”nobody ever got fired for buying IBM”*
- Tarjolla on *ohjelmistokehyksiä* ja *–alustoja*, jotka
  - Kiinnittävät monet sovellusten laatuominaisuudet
  - Pyrkivät vapauttamaan sovelluskehittäjän keskittymään sovelluskohtaisen toiminnallisuuden toteuttamiseen
    - Toiminnanohjaus, asiakkuuksien hallinta, web-palvelut, mobiiliapplikaatiot jne jne.

Oletusarkkitehtuuri

- Projektin ainoaksi arkkitehtuuriratkaisuksi jää käytettävän ohjelmistokehyksen valinta, missä arkkitehtuuriasioita enemmän saattavat painaa muut seikat
  - Yhteensopivuus, käyttöympäristö, palvelinympäristö, henkilöstön osaaminen, ohjelmointikieli, markkinatilanne, lisenssi- ja tukiehdot, jne.
- Riskejä
  - Arkkitehtuurin rapautuminen ajan myötä oman arkkitehtuurisuunnittelun ohjaavan vaikutuksen ja yhteisen arkkitehtuurinäkemyksen puuttuessa
  - Monimutkaisuus, joka kehysten ja alustojen (projektille turhien) piirteiden myötä tulee ratkaisuun mukaan

Oletusarkkitehtuuri

- *Referenssiarkkitehtuuri*
  - Yleinen ratkaisumalli tietyn tyyppisten järjestelmien (tai sen osien) arkkitehtuureille
  - Kuvaa arkkitehtuuriratkaisun spesifikaation muodossa (vrt. ehdotus standardiksi)
  - Voi olla sovellusaluekohtainen tai yrityskohtainen
- Referenssiarkkitehtuurin määrittelijä toivoo usein, että siitä tulisi vallitseva oletusarkkitehtuuri

2. Arkkitehtuurikeskeinen (*focused*)

- Tunnusmerkkeinä ovat
  - *tietoinen* arkkitehtuurin valinta ja suunnittelu…
  - *laatuvaatimusten* ymmärtämisen pohjalta
- Arkkitehtuuri ei toisaalta saisi vaikeuttaa toiminnallisten vaatimusten toteuttamista
- Pyritään aktiivisesti tunnistamaan vaatimukset, jotka vaikuttavat arkkitehtuuriratkaisuihin
  - Vaatimusten kriittinen tarkastelu ja ”rivien välistä lukeminen” (implikaatiot)

Arkkitehtuurikeskeinen (*focused*)

- Ongelmanratkaisun ja päätelmien apuna käytetään usein abstraktioita ja arkkitehtuurinäkymiä
  - Järjestelmän *komponenttien* ja niiden *liitäntöjen* tarkastelu
  - Moduulinäkymä, suoritusaikaiset näkymät, …
- Ei vaadi lähtökohtaisesti minkään tietyn ohjelmistokehityksen prosessimallin noudattamista eikä täydellistä dokumentointia

3. Ominaisuuksien vivutus (*hoisting*)

- Arkkitehtuuriratkaisu tuodaan kehittäjien käyttöön suoraan implementaatiotasolla (esim. koodikirjastona tai konkreettisena rajoitteena)
- Valmiin koodin käyttö takaa halutut ominaisuudet ilman että kehittäjien tarvitsee erikseen tehdä mitään
  - Esim. resurssien, rinnakkaisuuden tai transaktioiden hallinta
- Suhteellisen pienellä määrällä työtä (uudelleenkäytettävä koodi) saadaan suuri (vipu-) vaikutus järjestelmän laatuominaisuuksiin

Vivutus

- Vivutuksen käyttöön liittyy usein harkintaa laatuominaisuuksien tasapainottelun kannalta (trade offs)
  - Vivutusta voi olla vaikea kiertää, joten vivutetun ominaisuuden prioriteetin on syytä olla riittävän korkea
- Vivutus vai kehittäjän hivutus?
  - Jotakuta rajoitukset voivat haitata, mutta toisaalta ne vapauttavat kehittäjän aikaa ja energiaa muihin asioihin – kaikki eivät ole experttejä hankalien laatuominaisuuksien alueella

Esimerkkejä

- *Tavoiteltu ominaisuus*: virran säästö akkukäyttöisessä mobiililaitteessa
- *Arkkitehtuuriratkaisu (Symbian OS)*: Applikaatioiden ja yleisten palveluiden (esim. tiedosto-operaatiot) toteuttaminen *tapahtumankäsittelijöinä*
  - Applikaatio- ja palvelusäikeet (thread) toimivat vain reagoidessaan tapahtumiin (kälitapahtuma, palvelupyyntö), muuten ne odottavat passiivisina (wait)
  - Kaikki kommunikointi tapahtuu asynkronista viestinvälitystä käyttäen (palvelupyyntö-vastaus -parit)
  - Käyttöjärjestelmän on helppo todeta milloin kaikki säikeet vain odottavat tapahtumia, jolloin prosessori (CPU) voidaan ajaa virransäästötilaan (sleep mode)

Esimerkkejä

- *Tavoiteltu ominaisuus*: älypuhelimessa taustalla olevien sovellusten huomaamaton sulkeminen ja niiden tilan automaattinen palauttaminen käyttäjän tuodessa sovelluksen taas esiin
  - muistin vapautus, akun latauksen säästö, käyttökokemus eli aidon moniajon simulointi
- *Arkkitehtuuriratkaisu (Android/WindowsPhone)*: sovelluskehys tarjoaa laajennospisteet, joihin sovelluskohtainen tilatiedon käsittely koodataan
  - Kehittäjä koodaa (takaisinkutsumetodina/ tapahtumankäsittelijänä) sovelluksen tilan kirjoittamisen ja lukemisen kehyksen tarjoamaan tietovarastoon
  - Sovelluskehys käynnistää tilan tallennuksen ja lukemisen (kutsuu kehittäjän koodia) automaattisesti tilanteen mukaan

Milloin arkkitehtuurityö on tärkeää?

- Pieni ratkaisuavaruus (solution space)
  - Toimivia ratkaisuja on vähän ja sellaisen löytäminen on vaikeaa
- Ohjelmistohäiriöiden (failure) vakavat seuraukset
  - Vahingot ihmisille, ympäristölle ja omaisuudelle
- Vaikeat laatuvaatimukset
  - Skaalautuvuus, ylivertainen käyttökokemus

Milloin arkkitehtuurityö on tärkeää?

- Uusi sovellusalue (domain)
  - Uudet käsitteet, toiminnot, vaatimukset, teknologiat jne.
- Tuoteperheet
  - Yhteisen tuoterungon tai komponenttikirjaston luominen monia eri tuotteita varten, joissa on kuitenkin paljon samaa toiminnallisuutta

ARKKITEHTUURI JA PROSESSIT

Kaikilla ohjelmistoilla on arkkitehtuuri

- Kurssin teesit:
  - Paitsi jos projekti on pieni ja rutiininomainen, tulisi noudattaa arkkitehtuurikeskeistä kehitystapaa, ja tehdä juuri oikea määrä arkkitehtuurityötä riskien minimoimiseksi
  - Oletusarkkitehtuureista on usein paljon hyötyä, mutta niiden vaikutus laatuominaisuuksiin ja riskeihin pitää arvioida ja ymmärtää
- Pitääkö arkkitehtuurisuunnittelu sitten erottaa omaksi muodolliseksi tehtäväkseen, jolla on
  - Syötteet, tulosteet, ohjausvaikutukset, johtamis- ja valvontamekanismit, roolit, jne.

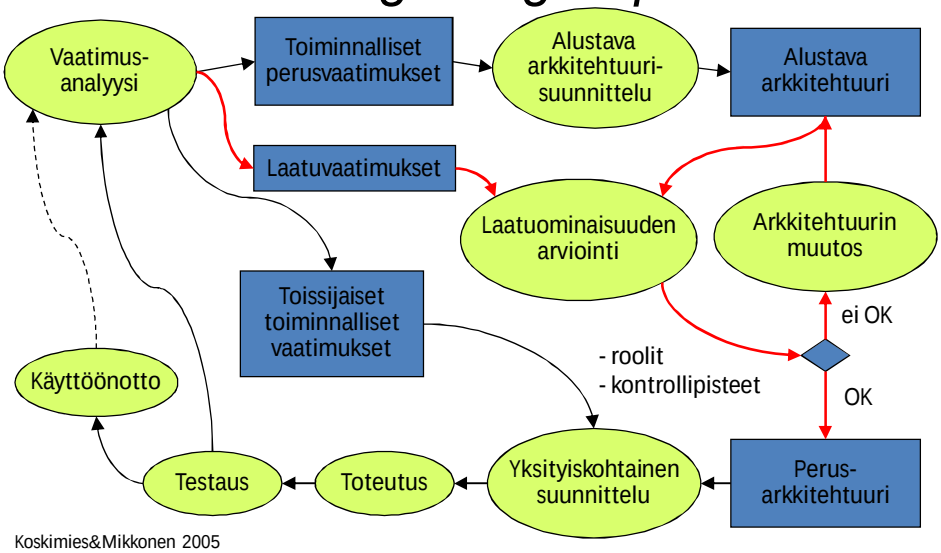
Prosessiajattelun kaksi ääripäätä

BDUF ⟷ emergence

Suunnitelmat
Kontrollointi
Riskit

Muutos
Reagointi
Mukautuminen

BDUF – *Big Design Up Front*



Koskimies&Mikkonen 2005

Ketterä kehitys

- *Agile manifesto*[1]

> Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät [*emerge*] itseorganisoituvissa tiimeissä.

[1]http://agilemanifesto.org/iso/fi/

Kumpi vai kampi?

- Ennakkosuunnittelun tarpeellisuus riippuu projektin luonteesta:
  - http://en.wikipedia.org/wiki/Agile_Manifesto#Suitability
- Ketterissäkin projekteissa voidaan tehdä arkkitehtuurityötä
  - Arkkitehtuurikeskeisen ja vivuttavan kehistystavan käyttö ei vaadi (raskasta) formaalia prosessia ja täydellistä dokumentointia
  - ”Architecture backlog” ”feature backlog:n” rinnalla
  - ”Architecture owner” ”Product owner:n” lisäksi
- Molempi parempi…

Inkrementaalinen kehitys

- Jäykkien vesiputousprosessien ja joustavien ketterien menetelmien ohella on kehitetty monia inkrementaalisia, ohjelmiston asteittaista *kasvattamista* korostavia prosessimalleja
  - Spiral model
  - RUP, Rational Unified Process

Spiraalimalli[1]

- ”Metaprosessimalli” projektikohtaisen kehitysmenetelmän löytämiseksi
- Kehitys tapahtuu sykleissä, joissa
  1. selvitetään projektin menestymisen kannalta kriittisten sidosryhmien asettamat tavoitteet
  2. kehitetään ja evaluoidaan vaihtoehtoisia ratkaisuja
  3. tunnistetaan ja ratkaistaan valittuun ratkaisuun liittyvät riskit
  4. haetaan sidosryhmien hyväksyntä ja lupa seuraavaan kierroksen käynnistämiseen
- Projekti voi yhdistellä haluamiaan osia eri kehitysmenetelmistä sillä perusteella, miten ne sopivat projektin riskien voittamiseen

[1] http://en.wikipedia.org/wiki/Spiral_model

Spiraalimalli

- *Life Cycle Architecture* –kontrollipiste (myöhempi lisäys alkuperäiseen malliin)
  - Onko olemassa riittävän hyvin määritelty ja kaikkien sidosryhmien tavoitteet täyttävä ratkaisu ja onko kaikki riskit eliminoitu tai minimoitu?
  - "'Hazardous spiral look-alikes' that violate this invariant include evolutionary and incremental processes that commit significant resources to implementing a solution with a poorly defined architecture."[1]

[1]http://en.wikipedia.org/wiki/Spiral_model

Rational Unified Process[1]

- Räätälöitävä prosessikehys iteratiivisten ohjelmistokehitykseen
  - Prosessi koostuu neljästä peräkkäisestä vaiheesta, jotka jakautuvat kiinteän mittaisiin iteraatioihin, jotka tuottavat inkrementaalisesti lisäarvoa (business value)
  - Kussakin vaiheessa ja iteraatiossa voidaan tehdä kaikkia ohjelmistokehityksen aktiviteetteja, mutta eri painolla

[1]http://en.wikipedia.org/wiki/RUP

Rational Unified Process

- Arkkitehtuurityö tapahtuu pääasiassa *Elaboration* –vaiheessa
  - Tuotoksina mm. arkkitehtuurin kuvaus ja ”suoritettava arkkitehtuuri” -malli (executable architecture)

Fairbanks – *Risk-driven approach*

- Idea:
  - Arkkitehtuurisuunnittelua tehdään vain sen verran kuin projektiin liittyvien riskien voittaminen vaatii
- Riskit voivat liittyä ohjelmiston *käyttöön ja sen laatuominaisuuksiin* (tuoteriski)
  - suorituskyky, turvallisuus, skaalautuvuus, uhka ihmisille ja omaisuudelle, ...
- ... tai sen *kehitykseen* (projektiriski)
  - teknologiat, henkilöstö, asiakas, aikataulu, vaikeat suunnitteluhaasteet, ...

Risk-driven approch

- Fairbanks ei esitä yhtä kaiken kattavaa prosessimallia, vaan kokoelman erilaisia tekniikoita ja lähestymistapoja riskien identifiointiin, ratkaisujen suunnitteluun ja ratkaisujen evaluointiin
- Ei edellytä mitään tiettyä kehitysprosessia, vaan voidaan soveltaa monenlaisissa kehitysprojekteissa

Yhteenveto

- Arkkitehtuurityötä voidaan tehdä monenlaisia prosesseja seuraavissa projekteissa
- Käytetty prosessimalli pitäisi valita projektin tarpeista lähtien
- Tuote- ja projektiriskien pitäisi ohjata arkkitehtuurityötä
- Vältettävä ”paralysis by analysis”, ”design until perfect” ja ”modeling for modeling’s sake” –tilanteita,

OHJELMISTOARKKITEHDIN TIEDOT JA TAIDOT

Mitä arkkitehdit tekevät?

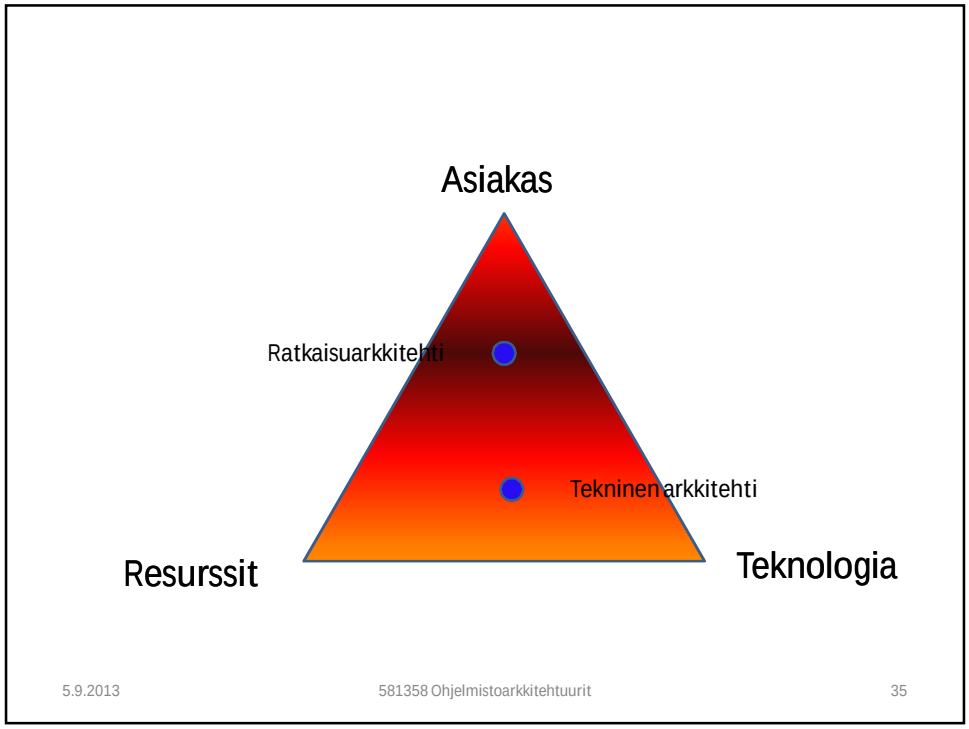
- Seuraavat arkkitehdin toimen kuvaukset perustuvat globaalin palvelu- ja konsultointiyrityksen Accenture:n ”standardirooleihin”
  - Yrityksellä on noin pari kymmentä (!) Architect –nimikkeen sisältävää standardiroolia

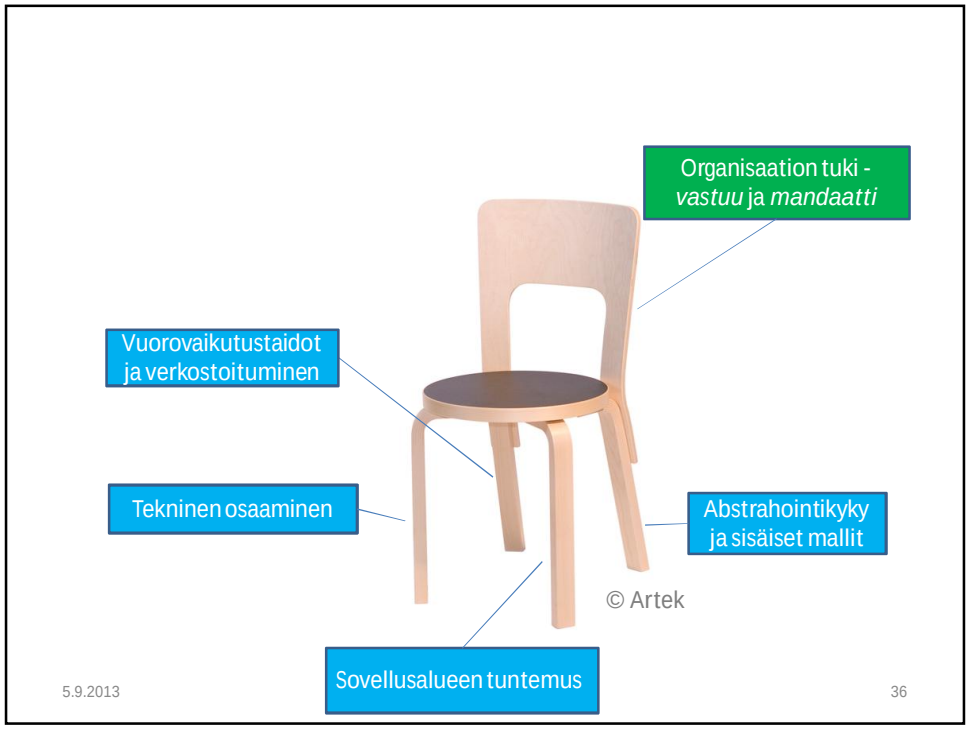
Ratkaisuarkkitehti

- Toimii asiakasrajapinnassa (*client-facing role*)
- Tulkitsee asiakasvaatimukset ja muodostaa niiden pohjalta ratkaisusuunnitelman (*solution plan*), joka voidaan koota tarjolla olevista (standardi-) rakennusosista
- Osallistuu työmäärien ja kustannusten arviointiin
- Tavoitteena on globaalien resurssien ja aikaisempien ratkaisujen sekä organisaation osaamisen kustannustehokas käyttö

Tekninen arkkitehti

- Tekninen asiantuntija, joka vastaa tietyn teknologia-alueen kehityssuunnasta ja arkkitehtuurista
- Muodostaa teknisiä vaatimuksia ja ohjelmistosuunnitelmia liiketoiminta- ja asiakasvaatimusten perusteella
- Kehittää arkkitehtuurikomponentteja
- Osallistuu yksityiskohtien suunnitteluun ja koodikatselmointeihin
- Analysoi suorituskyky- ja tehokkuusongelmia
- Ohjaa, valmentaa ja tukee kehittäjiä
- Määrittelee yleisiä käytäntöjä ja periaatteita (standardeja) sekä valvoo niiden noudattamista
- Implementoi itsekin


Asiakas
Ratkaisuarkkitehti
Tekninen arkkitehti
Resurssit
Teknologia


Organisaation tuki - *vastuu* ja *mandaatti*
Vuorovaikutustaidot ja verkostoituminen
Tekninen osaaminen
Abstrahointikyky ja sisäiset mallit
Sovellusalueen tuntemus
© Artek