# 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

## Tapa 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
  - Tarjoavat perusoiminnallisuuden valmiina uudelleenkäytettävinä komponentteina/kehyksinä/kirjastoina
  - Kiinnittävät monet laatuominaisuudet (tai asettavat rajoituksia)
    - Turvallisuus, suorituskyky, ylläpidettävyys, ...
  - 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 ratkaisu tietyn tyyppisten järjestelmien (tai niiden osien) arkkitehtuureille
  - Kuvaa arkkitehtuuriratkaisun spesifikaation muodossa (vrt. ehdotus standardiksi)
  - Voi olla sovellusaluekohtainen tai yrityskohtainen
- Referenssiarkkitehtuurin määrittelijä toivoo usein, että siitä tulisi vallitseva oletusarkkitehtuuri

## Tapa 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
  - Moduulien väliset riippuvuudet, kommunikoivat prosessit, pääkäyttötapausten kulku, …
- Ei vaadi lähtökohtaisesti minkään tietyn ohjelmistokehityksen prosessimallin noudattamista eikä täydellistä dokumentointia

# Tapa 3: Arkkitehtuuri laatuvipuna (*hoisting*)

- Arkkitehtuuriratkaisu implementoidaan koodiksi ja tuodaan suoraan kehittäjien käyttöön
  - Koodikirjastona, komponentteina, tai konkreettisena automaattisesti valvottuna rajoitteena
  - Yleistä ohjelmistokehyksissä (framework)
- 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 *vipuvaikutus* järjestelmän laatuominaisuuksiin (leverage, hoisting)



# Laatuvipu

- Vivun käyttöön liittyy usein harkintaa laatuominaisuuksien tasapainottelun kannalta (trade offs)
  - Vipua voi olla pakko käyttää (vaikeaa kiertää), joten vivutetun ominaisuuden on syytä olla riittävän tärkeä
- 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 (esim. transaktioiden ja rinnakkaisuuden hallinta)

# Esimerkkejä

- *Tavoiteltu ominaisuus*: ladatun energian säästö akkukäyttöisessä mobiililaitteessa
- *Arkkitehtuuriratkaisu (Symbian OS)*: Applikaatioiden ja yleisten palveluiden (esim. tiedosto-operaatiot) toteuttaminen *tapahtumankäsittelijöinä* (asiakas-palvelin tyyli)
  - Applikaatio- ja palvelusäikeet (thread) toimivat vain reagoidessaan tapahtumiin (kälitapahtuma, palvelupyyntö), muuten ne odottavat passiivisina (wait)
  - Kaikki kommunikointi tapahtuu asynkronista viestinvälitystä käyttäen (palvelupyyntö-vastaus -parit)
  - Kaikki keskeiset KJ:n palvelut on toteutettu palvelinsäikeinä
  - Käyttöjärjestelmän on helppo todeta milloin kaikki säikeet vain odottavat tapahtumia, jolloin prosessori (CPU) voidaan ajaa virransäästötilaan (sleep mode) -> energiansäästö, parempi akun kesto



# 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)*: KJ:n sovellusarkkitehtuuri tarjoaa *laajennospisteet*, joihin sovelluskohtainen tilatiedon tallennus- ja palautuslogiikka (save/restore) koodataan
  - Kehittäjä koodaa (takaisinkutsumetodina/ tapahtumankäsittelijänä) sovelluksen tilan kirjoittamisen ja lukemisen KJ:n tarjoamaan tietovarastoon
  - KJ (sovellusarkkitehtuuri) käynnistää tilan tallennuksen ja lukemisen (eli kutsuu kehittäjän koodia) automaattisesti tilanteen mukaan
  - Käyttäjälle sovellus näyttäytyy jatkuvasti päällä olevana, mutta sen käyttämät järjestelmäresurssit voidaan kuitenkin välillä vapauttaa muuhun käyttöön
    - Monimutkaistaa hieman sovelluskehitystä

## Millaisissa projekteissa arkkitehtuurityö on erityisen tärkeää?

- Pieni ratkaisuavaruus (solution space)
  - Toimivia ratkaisuja on vähän ja sellaisen löytäminen on vaikeaa
  - Todennäköisyys, että mikä vain arkkitehtuuri toimii, on siis pieni
- Ohjelmistohäiriöiden (failure) vakavat seuraukset
  - Vahingot ihmisille, ympäristölle ja omaisuudelle
- Vaikeat laatuvaatimukset
  - Skaalautuvuus, ylivertainen käyttökokemus



## Millaisissa projekteissa arkkitehtuurityö on erityisen tärkeää?

- Uusi sovellusalue (domain)
  - Uudet käsitteet, toiminnot, vaatimukset, teknologiat jne.
  - Ei ole tuttua kaavaa tai rakennetta (conceptual model), jonka pohjalta lähteä rakentamaan ratkaisuja
- Tuoteperheet (tavoitteellinen uudelleenkäyttö)
  - Yhteisen tuoterungon tai komponenttikirjaston luominen monia eri tuotteita varten, joissa on kuitenkin paljon samaa toiminnallisuutta
  - Pyritään tunnistamaan yhteiset, eri tuotteissa tarvittavat komponentit ja implementoimaan ne vain kerran
  - Pyritään uudelleenkäyttämään mahdollisimman pitkälle myös samaa arkkitehtuuria eri tuotteissa (suunnittelutyön uudelleenkäyttö)

## Millaisissa projekteissa arkkitehtuurityö on erityisen tärkeää?

- Tee ajatuskoe - mieti, mitä seurauksia väärillä arkkitehtuuriratkaisuilla voisi olla: mikä voi mennä vikaan?
  - Jos merkittäviä riskejä ei ole, arkkitehtuurin miettimiseen ei kannata käyttää paljonkaan aikaa



## 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 silti arvioida ja ymmärtää
- Pitääkö arkkitehtuurisuunnittelu sitten erottaa omaksi muodolliseksi tehtäväkseen, jolla on
  - Syötteet, tulosteet, ohjausvaikutukset, johtamis- ja valvontamekanismit, roolit, jne.



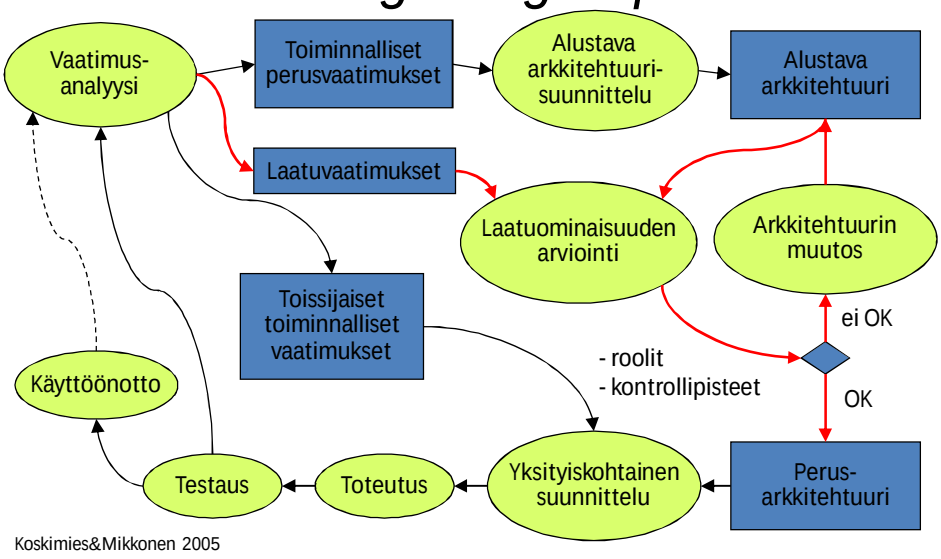
## Prosessiajattelun kaksi ääripäätä

BDUF  emergence

Suunnitelmat
Kontrollointi
Riskien hallinta

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_software_development#Adaptive_vs._predictive
  - Ketterissäkin projekteissa voidaan tehdä arkkitehtuurityötä
  - Arkkitehtuurikeskeisen kehistystavan tai laatuvivun käyttö ei vaadi (raskasta) formaalia prosessia ja täydellistä dokumentointia
  - Refaktorointia voidaan tehdä myös arkkitehtuurin parantamiseksi ja ”teknisen velan” kurissa pitämiseksi
    - Design spikes (Scrum)
  - ”Architecture backlog” ”feature backlog:n” rinnalla
  - ”Architecture owner” ”Product owner:n” lisäksi
- Molempi parempi…



# CRASH Report 2014

- Suurten yritysohjelmistojen mittaukseen ja analysointiin erikoistunut CAST –yhtiö julkaisee muutaman vuoden välein asiakasyritystensä ohjelmistojen analysoinnista keräämästään datasta CRASH –raportin (CAST Report on Application Software Health)
- Vuoden 2014 raportin* mukaan: ”Agile/Waterfall mix exhibits higher structural quality”
  - ”Structural quality” koostuu joukosta staattisesti mitattavia indikaattoreita ohjelmakoodista ja konfiguraatio-ym. tiedostoista
  - Perustuu hvyien arkkitehtuuri- ja koodauskäytäntöjen vastaisten rakenteiden tunnistamiseen**
- Kyseessä ovat pääasiassa yritysten ja julkishallinnon suuret kriittiset järjestelmät (mission critical), jotka on toteutettu monenlaisilla teknologioilla (http://www.castsoftware.com/products/appmarq)
- Katso myös Don Reiferin blogi: http://reifer.com/news-flash/
  - Suunnitteluvirheiden suhteellinen osuus näyttää olevan suurempi ketterissä kuin ”perinteisissä” projekteissa (vaikka ketterien kokonaislaatu on usein parempi)

*) http://www.castsoftware.com/news-events/event/crash-report-webinar

**) http://www.castsoftware.com/research-labs/crash-reports

# Agile Software Quality

- Reifer Consultans, 15.10.2013
  - Kyselytutkimus, n. 500 ketterää projektia 10 eri sovellusalueelta (koko keskim. n. 1000 FP)
  - Verrattu vastaavanlaisiin ”perinteisiin” projekteihin
  - Johtopäätös: *ketterien laatu parempi*
    - Jos toimitettujen piirteiden/user storyjen lukumäärä suhteessa alun perin vaadittujen määrään jätetään huomiotta
  - Raportin voi ostaa ISBSG:n sivulta http://www.isbsg.com/collections/analysis-reports



# Agile Productivity, Cost and Quality Benchmarks

- Reifer Consultants, 23.6.2013 (v3.3)
  - 800 projektia 60 yrityksestä 10 vuoden ajalta
  - 10 sov. aluetta, 250 ketterää, 550 perinteistä projektia
- Johtopäätökset
  - Ketterien plussat
    - Parempi tuottavuus
    - Alemmat kustannukset
    - “Pehmeitä” tekijöitä johtaen korkeaan motivaatioon ja mielialaan
  - Miinukset
    - Hiukan huonompi laatu (ei paljon)
    - Kysymyksiä: skaalautuvuus isoihin projekteihin, sopimusten ja riskien hallinta, ylläpito
- Raportin voi ostaa ISBSG:n sivulta http://www.isbsg.com/collections/analysis-reports

# Kehitysmenetelmät ja laatu

- Capers Jones:in (Namcook Analytics) dataa
  - Katso diat 50 ja 51 tästä powerpointista: http://namcookanalytics.com/wp-content/uploads/2013/10/SQA2013Long.pdf
- Datan lähteet, katso dia 2 samasta esityksestä
- Tutkimusta aiheesta tarvitaan vielä
  - Appelsiineja ja omenoita ei voi suoraan verrata
  - Ohjelmistokehityksen mittaus ja raportointi yleisesti ottaen melko kehnoa (kaikissa organisaatioissa)
  - Lisää asiasta Ohjelmistoprosessit ja Laatu -kurssilla



# Inkrementaalinen kehitys

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

# 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 (syklin) käynnistämiseen
- Projekti voi yhdistellä osia eri kehitysmenetelmistä sillä perusteella, miten ne sopivat projektin riskien voittamiseen
  - Eri sykleissä voidaan noudattaa eri menetelmiä

[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
  - Iteraatiot tyypillisesti pitempiä kuin ketterissä menetelmissä

[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)
  - Korkea suorituskyky, turvallisuus, skaalautuvuus, uhka ihmisille ja omaisuudelle, …
- … tai sen *kehitykseen* (projektiriski)
  - teknologiat, henkilöstön määrä ja osaaminen, asiakas, aikataulu, työkalut, …
- Kysytään: ”*Mikä voi mennä pieleen* ohjelmistoa käytettäessä ja kehitettäessä”?



# Riski

Asiaan *A* liittyvä Riski =
häiriön todennäköisyys *A*:ssa x häiriön aiheuttama haitta

- Riskien tunnistaminen on luonnollisesti kaiken lähtökohta
  - Ei aina helppoa, vaatii kokemusta sovellusalueesta ja käytetyistä toteutusteknologioista
  - Vaatimukset ovat kuitenkin hyvä lähtökohta riskianalyysille
    - Vaikeasti toteutettavilta tuntuva asiat ovat riskikandidaatteja
    - Tunnistamattomat tai epämääräiset vaatimukset ovat sinänsä aina jonkinasteisia riskejä ja virheiden lähteitä

# Yleisiä riskikategorioita

| Sovellusalue | Tyypillisiä riskejä |
|---|---|
| Tietotekniikkapalvelut (IT) | Kompleksinen, huonosti ymmärretty ongelma-alue<br>Epävarmuus siitä, ollaanko ratkaisemassa oikeaa ongelmaa<br>Väärien COTS ohjelmien valinta<br>Integrointi olemassaoleviin mutta huonosti tunnettuihin ohjelmistoihin<br>Sovellusalueen tuntemus hajallaan organisaatiossa<br>Muunneltavuus ja räätälöinitarpeet |
| Kriittiset järjestelmät | Suorituskyky, luotettavuus, koko, turvallisuus<br>Rinnakkaisuus<br>Koostaminen ja konfigurointi |
| Web | Turvallisuus<br>Skaalautuvuus käyttäjämäärän ja datamäärän mukaan<br>Kehittäjien tuottavuus |



# Riskien hallinta

- Tunnistetut riskit pitää kirjoittaa auki siten, että voidaan todeta, ovatko riskin pienentämiseksi tehtävät toimet tehokkaita
  - Sen sijaan, että sanotaan vain tiettyyn laatuominaisuuteen (esim. suorituskykyyn) liittyy riski, kirjoitetaan muutama konkreettinen häiriöskenaario (failure scenario), jossa häiriötilanne kuvataan kvantitatiivisesti
    - Esim: ”Aloitussivun latautuminen kestää yli 10 sekuntia 10%:lla käyttäjistä”

# Riskien hallinta

- Perusidea
  1. Tunnista ja *priorisoi* riskit
  2. Valitse ja käytä tekniikoita riskien lieventämiseksi (risk mitigation)
  3. Evaluoi toimenpiteiden vaikutus riskeihin
- Tätä sykliä toistetaan, kunnes riskit on saatu siedettävälle tasolle (subjektiivinen arvio)
- Arkkitehtuurityön osalta tämä tarkoittaa, että arkkitehtuurisuunnittelu ja -analyysi kohdistetaan *riskeihin liittyviin osa-alueisiin*



# *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
  - RUP ja Boehmin spiraalimalliin perustuvat prosessit ovat myös riskilähtöisiä: *The most risky things first!*
  - Sopii hyvin ohjenuoraksi myös ketterään kehitykseen, joskin niissä fokus on yleensä käyttäjille näkyvässä toiminnallisuudessa

# 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, eli suunnittelua suunnittelun vuoksi!

!



# 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



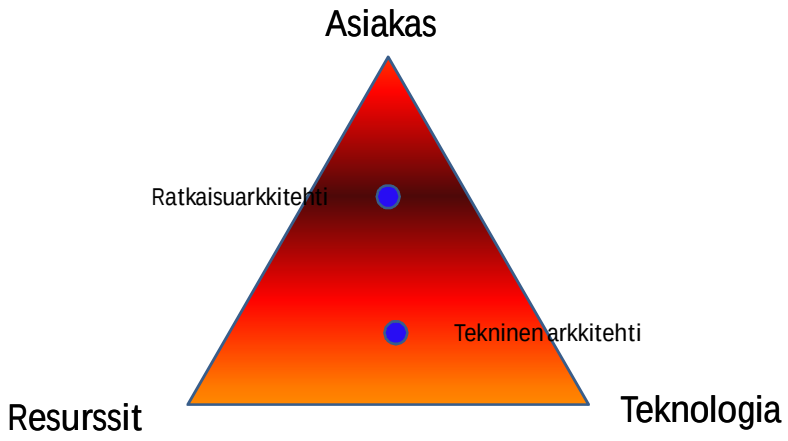
# Ratkaisuarkkitehti

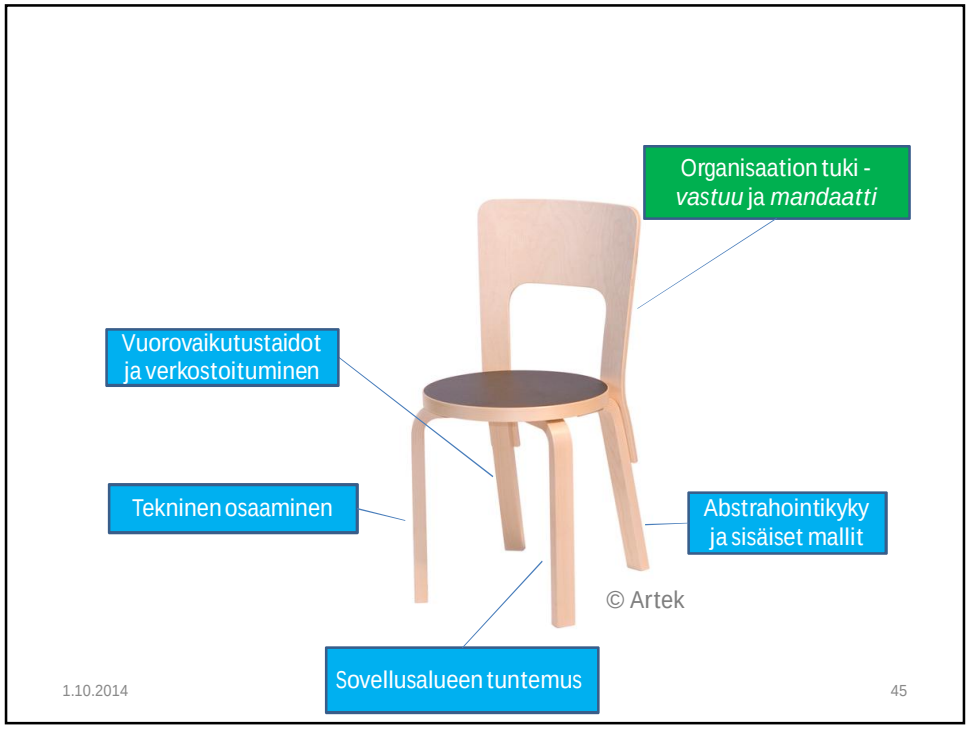
- Toimii asiakasrajapinnassa (*client-facing role*)
- Tulkitsee asiakasvaatimukset ja muodostaa niiden pohjalta ratkaisusuunnitelman (*solution plan*), joka voidaan koota tarjolla olevista (standardi-) rakennusosista
- Osallistuu työmäärien ja kustannusten arviointiin
- Tavoitteena on globaalien resurssien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






Organisaation tuki -
*vastuu* ja *mandaatti*
Vuorovaikutustaidot
ja verkostoituminen
Tekninen osaaminen
Abstrahointikyky
ja sisäiset mallit
Sovellusalueen tuntemus
© Artek
1.10.2014
45