Ohjelmistoarkkitehtuurit

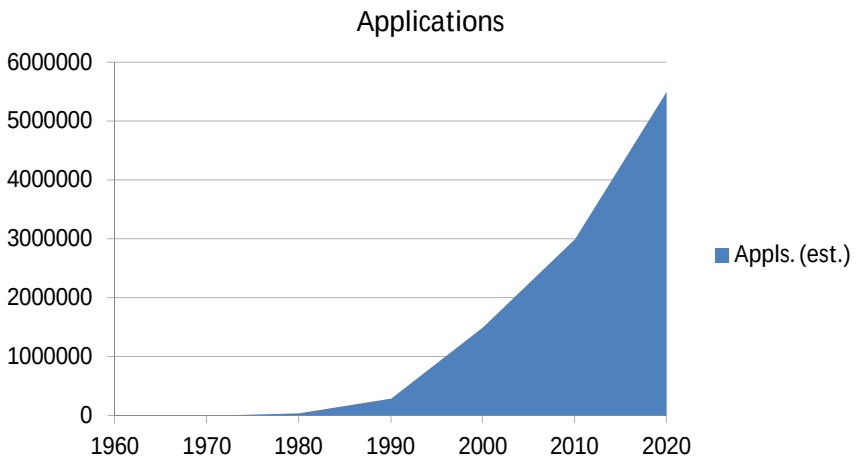
Luento 1

Oppimistavoitteet

- Mitä ohjelmistoarkkitehtuuri on?
- Mitä hyötyä siitä on?
- Miten aihetta kurssilla käsitellään?

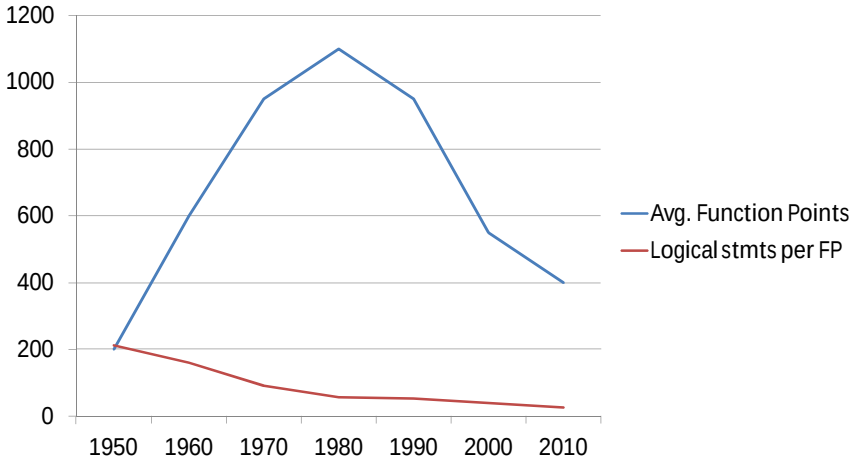
ALUKSI

Sovellusten määrä USA:ssa eri vuosikymmeninä (arvio)*



*) Capers Jones, *The Technical and Social History of Software Engineering*. Addison-Wesley Professional, 2013.

Sovellusten keskim. koko USA:ssa *Function Point* -kokomittarilla (arvio)*



*) Capers Jones, *The Technical and Social History of Software Engineering*. Addison-Wesley Professional, 2013.

Ohjelmistotekniikan kehityksestä

2010	Websovellusalustat, HTML5, pilvipalvelut, mobiili, Big data	Lean, ekosysteemit
2000	*Tuoteperhearkkitehtuurit, MDA*, väliohjelmistot, ohjelmistoalustat, aspektit, dynaamiset (skripti-) kielet	Ketterät menetelmät (agile methods)
	CASE-välineet: uudelleenkäyttö, testaus, mittaus	
1995	*Patterns (ratkaisumallit), sovelluskehykset, ohjelmistoarkkitehtuurit, arkkitehtuurityylit, arkkitehtuurien kuvauskielet, UML*	Liiketoiminta, organisaatio, Unified Process & inkrementaalinen kehitys
1990	CASE-välineet: käyttöliittymien piirtäminen	Graafiset käyttöliittymät, käytettävyys
	CASE-välineet: kaaviotyökalut (oliotekniikat)	
1985	CASE-välineet: koodingenerointi (5GL)	Prosessit: CMM, spiraalimalli & riskien hallinta
1980	Oliot, uudelleenkäyttö, oliosuunnittelu Tietokone- ja järjestelmäarkkitehtuuri, Relaatiotietokannat (4GL) Suunnittelumenetelmät: tietovirtakaaviot, ER-malli	
1970	*Modulaarisuus, tiedon kätkeminen,* (2GL, 3GL)	Prosessit: vesiputous
1950 - 60	Aliohjelmat (1GL)	

Ohjelmistojen merkityksestä

- 2010-luvun tietoyhteiskunta: lukematon määrä palveluita toteutetaan tietojärjestelminä, joista monet ovat kooltaan ja käyttäjämääriltään maailmanlaajuisia
- Tietokonelaitteistot kehittyneet huimasti: laskentateho, nopea tiedonsiirto, valtavien tietomäärien hajautettu tallennus ja rinnakkainen käsittely
- Virtualisoinnin ja ”pilvestä” saatavan laskenta- ja tallennuskapasiteetin ansiosta kokonaisia järjestelmiä voidaan toteuttaa nyt puhtaasti *ohjelmistoratkaisuina* (ei omia kiinteitä laiteinvestointeja)
- Laadultaan hyvien ohjelmistojen kustannustehokas toteuttaminen on entistäkin tärkeämpää
  - Tarkoituksenmukaisuus, tehokkuus, luotettavuus, saatavuus, käyttökokemus, muunneltavuus, skaalautuvuus, turvallisuus, ...
  - Kilpailu on kovaa ja markkinat avoimet, time-to-market

OHJELMISTOARKKITEHTUURI

Mitä OA on?

- Määritelmä (Software Engineering Inst. / CMU)

> ” The software architecture of a system is the *set of structures needed to reason about the system*, which comprise software elements, relations among them, and properties of both.”

> ” Järjestelmän ohjelmistoarkkitehtuuri tarkoittaa rakenteita, joita tarvitaan päätelmien tekemiseksi järjestelmästä ja sen ominaisuuksista. Rakenteet koostuvat ohjelmistoelementeistä, niiden välisistä suhteista, ja näiden molempien piirteistä.”

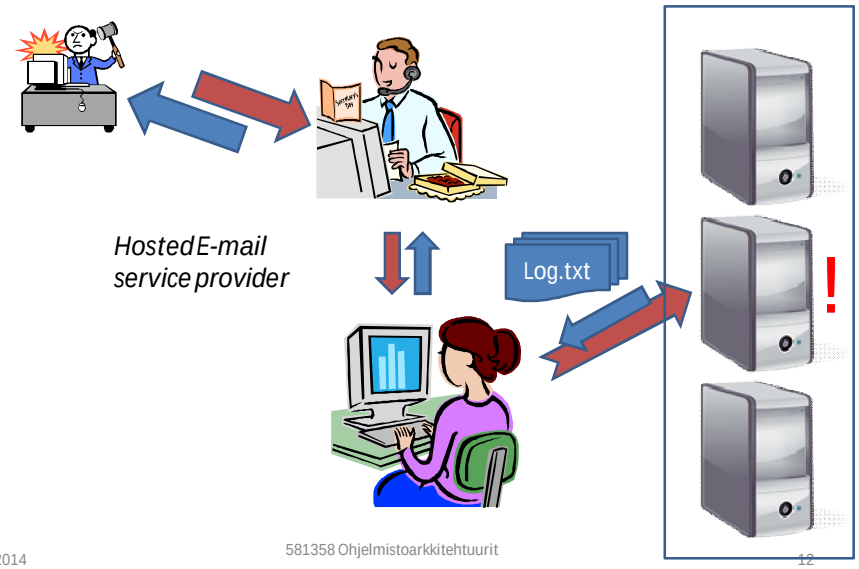
...mutta mitä se tarkoittaa?

- Määritelmä ei kerro suoraan mitä rakenteet ovat
  - Ei yksi vaan *joukko* rakenteita
  - Mikä tahansa tarkoitukseen sopiva kokoelma ”elementtejä” (koodimoduuleja, komponentteja, prosesseja, virtuaalikoneita) ja niiden välisiä suhteita (riippuvuuksia, käyttösuhteita, omistussuhteita) voi muodostaa tarkasteltavan rakenteen
  - Kurssilla opimme, minkälaisia rakenteita yleensä kannattaa tarkastella
    - Riippuu myös sovellusalueesta ja käytetyistä teknologioista

...siis mitä?

- Oleellista on, että rakenteiden perusteella voidaan tehdä päätelmiä jollekin *sidosryhmälle tärkeistä ohjelmiston makrotason ominaisuuksista*
  - Asiakas, loppukäyttäjä, kehittäjä, ylläpitäjä, liiketoiminnan johto, viranomainen, ...
  - Tarkoituksenmukaisuus, luotettavuus, tehokkuus, helppokäyttöisyys, turvallisuus, muutettavuus, ulkoiset riippuvuudet, skaalautuvuus, ...
- Arkkitehtuuri ei ole itsetarkoitus vaan *väline*

Esimerkki - Rackspace



Tutkittava järjestelmä

- Sähköpostipalvelimien tuottamien lokitietojen tallennus ja käsittely asiakastukea varten
- Tehtävä (missio)
  - Asiakkaan sähköpostin toimintaa koskevien ongelmien selvittämisen tukeminen sähköpostipalvelimien tuottamien lokitietojen perusteella
  - Tarjottava ajantasaista tietoa riittävän pitkältä ajanjaksolta
- Kolme järjestelmäversiota, kolme erilaista arkkitehtuuria

V. 1

- *S-postipalvelinten paikalliset lokitiedostot*
  - Teknisen tuen pitää pyytää operaattoria ottamaan yhteys asiakkaan postipalvelimeen ja tutkimaan lokitietoja ongelmien selvittämiseksi
  - Rackspace toteutti skriptin, joka automaattisesti ottaa yhteyden joukkoon palvelimia ja suorittaa haun (grep) halutuntyyppisen palvelimen lokista
- Ongelmia:
  - Palvelinten, asiakkaiden ja tukipyyntöjen määrän kasvaessa yhä useammin suoritettavat haut alkoivat vaikuttaa palvelinten suorituskykyyn
  - Hakujen tekemiseen tarvittiin aina operaattoria, mikä vei jatkuvasti kasvavan osan heidän työajastaan

V. 2

- *Lokitietojen keskittäminen yhteen tietokantaan*
  - S-postipalvelimet lähettävät muutaman minuutin väliajoin lokitietonsa ladattavaksi samaan relaatiotietokantaan
  - Teknikoilla on suora web-yhteys tietokantapalvelimelle ennalta ohjelmoitujen hakujen suorittamista varten
  - Yksittäisten palvelimien lähettämät päivitykset kootaan yhteen säännöllisesti suoritettavaksi massapäivitykseksi (suoritetaan 10 min. välein) tietokantapalvelimen suorituskyvyn ylläpitämiseksi
- Ongelmia:
  - Datan ja kyselyiden määrän jatkuvasti kasvaessa tietokantapalvelimen suorituskyky ja kapasiteetti muodostuivat pullonkaulaksi, häiriöt lisääntyivät
  - Haut hidastuivat ja dataa hävisi välillä, vain muutaman päivän edestä lokitietoja kyettiin säilyttämään, varmuuskopioita ei ollut

V. 3

- *Lokitiedostojen rinnakkainen indeksointi hajautetussa tiedostojärjestelmässä*
  - S-postipavelimet syöttävät lokitiedostonsa usealle yksinkertaiselle palvelimelle (cluster) hajautettuun tiedostojärjestelmään (Hadoop DFS)
  - Hadoop map-reduce –operaatio indeksoi lokitiedostot ja muodostaa kaiken kattavan indeksin 15 min. välein
  - Teknikoilla on web-liittymä hakujen tekemiseen kuten ennenkin
  - Hakujen suoritus indeksistä on nopeaa, uudenlaisen haun ohjelmointi ja suoritus kestää kuitenkin joitain tunteja
  - Täydellinen varmuuskopiointi, lokitietojen säilytys 6 kk:n ajalta

Ratkaisujen vertailua

	V. 1	V. 2	V. 3
Toiminnallisuus	✔	✔	✔
Skaalautuvuus – datan ja hakujen määrä	heikko	välttävä	hyvä
Viive – lokitietojen saanti s-palvelimilta	välitön	10 min.	15 min.
Joustavuus – uudet haut	hyvä	hyvä	tyydyttävä

Huomioita 1

- Järjestelmän tekniselle tuelle näkyvät *toiminnot ovat käytännössä samat* kaikissa versioissa
  - Tukihenkilö sai kaikissa tapauksissa tarvitsemansa (samat) tiedot lokeihin kirjautuneista tapahtumista
- Järjestelmän toiminnallisuus (*mitä* järjestelmä tekee) ei siis ohjannut teknisiä ratkaisuja
  - 1. versioonkin olisi voinut toteuttaa web-liittymän hakujen pyytämiseen operaattoreilta

Huomioita 2

- Esimerkkijärjestelmän *arkkitehtuuriin* vaikuttavat eniten vaaditut *laatuominaisuudet* (quality attributes, -ilities, 'non-functional requirements')
  - Skaalautuvuus, tiedonsaannin viive, joustavuus uusien hakujen määrittelyssä
- Ylivoimaisesti tärkein järjestelmän ominaisuus on *skaalautuvuus* datan ja hakujen määrän suhteen
  - Huonosti skaalautuva järjestelmä ei lainkaan kykene täyttämään tehtäväänsä toiminnan laajentuessa

Huomioita 3

	V. 1	V. 2	V. 3
Toiminnallisuus	✔	✔	✔
Skaalautuvuus – datan ja hakujen määrä	heikko	välttävä	hyvä
Viive – lokitietojen saanti s-palvelimilta	välitön	10 min.	15 min.
Joustavuus – uudet haut	hyvä	hyvä	tyydyttävä

Huomioita 3

- Skaalautuvuuden parantuessa versiosta toiseen muut ominaisuudet hieman kärsivät
  - Viive kasvaa (lokidatan tilannekuvan ikä) -> 15 min
  - Joustavuus vähenee, koska uusien hakujen lisääminen vaatii enemmän työtä
- Joistakin vaadituista ominaisuuksista täytyy tinkiä toisten hyväksi (*trade off*)
  - Laatuominaisuuksien *priorisointi* esimerkissä:

Skaalautuvuus > Viive > Joustavuus

...entäs se arkkitehtuuri?

- Esimerkin järjestelmäversioiden kuvausten yhteydessä mainitut tekniset ratkaisut (suunnittelupäätökset) kuuluvat järjestelmän arkkitehtuuriin
- Ohjelmistoarkkitehtuuriin kuuluvat siis ne ohjelmiston *suunnittelupäätökset*, jotka *merkittävästi vaikuttavat* ohjelmiston *laadullisten ominaisuuksien* saavuttamiseen
  - Suunnittelupäätösten pohjalta voidaan rakentaa järjestelmä, jolla on halutut ominaisuudet
- *Sovellusalue* ja kunkin sovelluksen/järjestelmän *vaatimukset* vaikuttavat siihen, mitkä tekniset seikat milloinkin nousevat arkkitehtuuriasioiksi ja mitä rakenteita on syytä tarkastella (OA:n määritelmä)

Tyypillisiä arkkitehtuuripäätöksiä

- Järjestelmän osittaminen osajärjestelmiin (pääkomponentteihin) ja niiden roolien, toimintojen ja keskinäisten riippuvuuksien määrittely
  - Osajärjestelmien jako päätason komponentteihin jne.
- J:n käsittelemän ja säilyttämän tiedon tallennus- ja saantiratkaisut
- J:n rajapintojen (käyttöliittymät, ohjelmalliset rajapinnat) tunnistaminen ja erottaminen niiden kautta käytettävien toimintojen sisäisestä toteutuksesta
- J:n ajoaikaiseen suorituskykyyn, resurssien käyttöön ja turvallisuuteen vaikuttavat ratkaisut
  - Palvelimet, prosessit ja säikeet, tieto- ja tapahtumavirrat, etäyhteydet, valvonta, tunnistaminen, sijoittelu laitteistoon
- J:n kehitystä ja ylläpitoa tukevat ratkaisut
  - Uudelleenkäyttö, komponenttien vaihdettavuus ja siirrettävyys, diagnostiikka, testattavuus
- Jonkin teknologia-alustan ja sen referenssiarkkitehtuuri(e)n käyttö

Arkkitehtuuri ja muu suunnittelu (design, detailed design)

- Suunnittelu (design) – päätetään, minkälaisista yksiköistä ohjelmisto toteutus koostuu
  - Yksiköiden vastuut, rajapinnat, riippuvuudet
  - Esim. oliot ja niiden roolit ja rajapinnat, metodit ja niiden semantiikka, patternien soveltaminen, keskeiset algoritmit, luokkien paketointi, tietomallit, konfigurointi jne.
- Miten vedetään raja arkkitehtuurisuunnittelun ja muun suunnittelun (detailed design) välille?
  - Täytyykö raja vetää?
  - Joskus detaljit voivat olla tärkeitä; esimerkiksi komponenttistandardin muotovaatimukset ladattavien komponenttien rajapinnan koodaukselle ja niiden ulkoisten riippuvuuksien määrittelylle
- Arkkitehtuuriset suunnittelupäätökset
  - Vaikuttavat ohjelmiston keskeisten *laatuvaatimusten toteutumiseen*
  - Tai muuten *vaikuttavat laajasti* ohjelmiston osien suunnitteluun

Arkkitehtuuriratkaisut ja muut suunnittelupäätökset

Arkkitehtuuri ja muu suunnittelu

- Rajaa arkkitehtuurin ja (yksityiskohtien) suunnittelun välille ei ole aina helppo vetää, mutta järjestelmän laatuominaisuuksiin vaikuttavat suunnittelupäätökset ovat yleensä tunnistettavissa
  - Kaikilla ohjelmistoilla siis on arkkitehtuuri
  - On toki parempi, että arkkitehtuuriratkaisut ovat *harkittuja* kuin sattumalta syntyneitä (vrt. ”Big Ball of Mud”)
  - Kokeneella ammattilaisella on mielessään tietynlaisten *ongelmien hahmoja* ja niiden *ratkaisutapoja* (conceptual model), jotka auttavat häntä tunnistamaan arkkitehtuurin kannalta tärkeät päätökset -> näkee metsän puilta

Ohjelmistoarkkitehtuuri vs. Arkkitehtuuri

?

Suomen arkkitehtiliiton puheenjohtaja Erkki Rautio *Tietoviikon* haastattelussa 29.4.2012: ”Ei se herätä suuttumusta, vaan ehkä enemmänkin sääliä, kun it-ammattilaisilla ei ole omia ammattinimikkeitä. ”

Huippuarkkitehtuuri on *taidemuoto*, jossa rakennusten käyttäjien tarpeet usein jäävät vähemmälle huomiolle kuin taiteelliset näkökohdat

Venustas (!) , Firmitas(?), Unistas (??)

OHJELMISTOARKKITEHTUURIN HYÖDYT JA KÄYTTÖ

Arkkitehtuurin ilmeneminen

- Arkkitehtuuri on siis valikoitu joukko ohjelmisto-elementtien ja niiden välisten suhteiden muodostamia rakenteita
- Konkreettisesti nämä rakenteet voivat ilmetä
  - Ideoina ja periaatteina (kehittäjien mielessä)
  - Konkreettisina rajoitteina (suunnittelulle ja toteutukselle)
  - Dokumentteina (muodollisina tai vapaamuotoisina)
  - Malleina (UML, formaalit mallinnuskielet)
  - Koodina (kirjastot, kehykset, alustat, esimerkit ja prototyypit, lähdekoodin struktuuri ja riippuvuudet, paketointi ja nimentä) ja suoritusaikaisina objekteina
    - Implementaation ilmentämä arkkitehtuuri on tietysti lopullinen totuus (ei välttämättä vastaa dokumentoituja suunnitelmia)

Arkkitehtuurin käyttö

- Arkkitehtuurin käyttötavat ohjelmistokehityksessä voidaan jakaa karkeasti kahteen kategoriaan
  - *Preskriptiivinen* eli *ohjaava* käyttö
  - *Deskriptiivinen* eli *kuvaileva* käyttö

Ohjaava käyttö

- Arkkitehtuuri määrittää järjestelmän perusrakenteen
  - Analogiana eläimen luuranko
  - Arkkitehtuurin lähtökohdaksi voidaan valita *tyyli* (architectural style) tai *patterni* (architectural pattern), joka kiinnittää elementtien vastuut/roolit ja niiden välisten liitosten ja yhteyksien ominaisuudet
  - Ei voida sanoa, onko jokin tyyli absoluuttisesti parempi kuin toinen, vaan täytyy arvioida tyylin *sopivuutta* kehitettävän ohjelmiston tarpeisiin nähden

Ohjaava käyttö

- Arkkitehtuuri vaikuttaa laatuominaisuuksiin
  - Arkkitehtuuri mahdollistaa haluttujen ominaisuuksien saavuttamisen
  - Sopimaton arkkitehtuuri voi myös estää sen
- Arkkitehtuuri on (enimmäkseen) riippumaton toiminnallisuudesta
  - Saman toiminnallisuuden voi toteuttaa melkein minkälaisella arkkitehtuurilla tahansa
  - *Mutta*: huonosti valittu arkkitehtuuri voi tehdä toiminnallisuuden toteuttamisen ja laatuvaatimusten saavuttamisen vaikeaksi ja kalliiksi (joskus jopa mahdottomaksi)

Ohjaava käyttö

- Arkkitehtuuri ohjaa toteutusta rajoitteiden avulla (guide rails)
  - Esimerkiksi käytettävyyden tai tietoturvallisuuden vuoksi halutaan suoraan kieltää eräät ominaisuuden kannalta huonoksi tiedetyt ratkaisut
  - Rajoitteet auttavat kehittäjiä monin tavoin
    - Kokemuksen siirto tiivistetyssä muodossa asiantuntijoilta, jotka ovat perustelluista syistä asettaneet rajoitteet
    - Käsitteellisen eheyden (conceptual integrity[1]) vahvistaminen ja kompleksisuuden vähentäminen ”tarpeetonta luovuutta” rajoittamalla
    - Koodin ajonaikaisen käyttäytymisen helpompi ymmärtäminen

[1] *”A single good idea consistently applied is better than several brilliant ideas scattered across a system”* (Fred Brooks)

Kuvaileva käyttö

- Ohjelmiston ja suunnitteluratkaisujen ymmärtäminen
  - Abstrahointi yksityiskohtia pois suodattamalla
  - Sopivasti valitut rakenteet ja niitä tietystä näkökulmasta esittävät *näkymät* (view) ovat erinomaisia ymmärryksen lähteitä
  - Uudet kehittäjät, johto, asiakkaat, alihankkijat, jne.
- Liiketoiminnallisten tavoitteiden toteutumisen seuraaminen
  - Tuoteperheet, COTS-komponenttien käyttö, integrointi ulkoisiin palveluihin, standardointi, lisensointi jne.

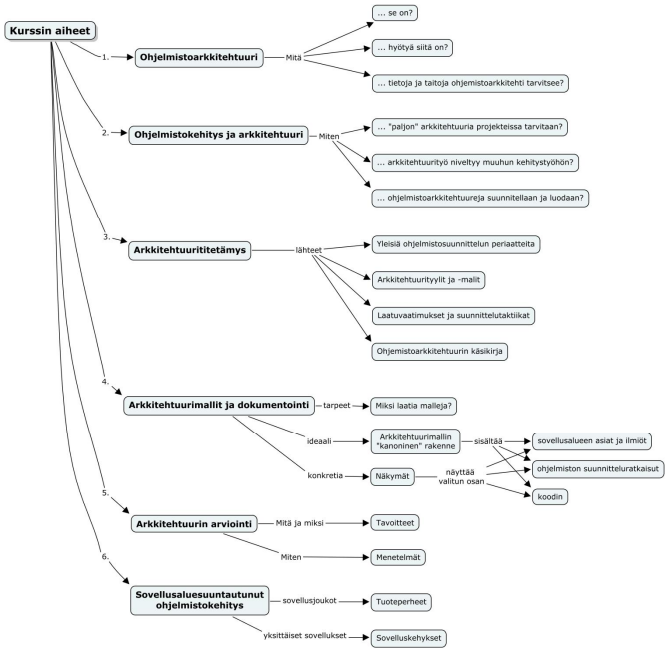
Kuvaileva käyttö

- Rajoitteiden noudattamisen valvonta
  - Esimerkiksi näkymän muodostaminen rakenneosien (komponenttien, kerrosten) välisistä riippuvuuksista, jolloin voidaan havaita arkkitehtuurin rajoitteiden kieltämät riippuvuudet
    - Esim. kerrosarkkitehtuurissa alemman kerroksen komponentti kutsuu ylemmän kerroksen palvelua muuten kuin palvelun asettaman takaisinkutsurajapinnan kautta
- Organisaation kehittäminen
  - Vahvasti toisiinsa kytkeytyvien elementtien kehittämisvastuun jako kannattaa miettiä tarkkaan kommunikaatio-ongelmien välttämiseksi
    - Conwayn laki – organisaatio ja arkkitehtuuri muistuttavat ennen pitkää toisiaan

Muita hyötyjä

- Riskien hallinta
  - Arkkitehtuurityö kannattaa keskittää tunnistettujen riskien eliminoimiseen tai lieventämiseen
    - Kiinnitetään jatkuvaa huomiota tärkeimpiin teknisiin uhkiin
- Vaatimusten täsmentäminen
  - Vaadittujen laatuominaisuuksien analysointi ja arkkitehtuurin suunnittelu niiden saavuttamiseksi auttaa huomaamaan ristiriitaisuuksia ja epätäsmällisyyksiä vaatimuksissa ja määrittelyissä
  - Arkkitehtuurin suunnitteluun liittyvä *laatuominaisuuksien tasapainottelu* (trade off) pakottaa *priorisoimaan laatuvaatimukset*

KURSSIN SISÄLTÖ



Pääoppimistavoitteet

- Tietää mitä ohjelmistoarkkitehtuuri on ja mihin sitä käytetään
- Tietää ja osaa soveltaa muutamia keskeisiä arkkitehtuurityylejä ja –patterneja ohjelmiston arkkitehtuurin suunnitteluun
- Tietää miksi ja milloin arkkitehtuurimalleja laaditaan ja osaa muodostaa ”kanonisen rakenteen” mukaisia näkymiä arkkitehtuuriin
- Hallitsee arkkitehtuurin arvioinnin ja ohjelmistokehysten (framework) suunnittelun perusideat (syvennetään harjoitustyössä)