Ohjelmistotuotanto

Luento 7

3.4.

Testaus ketterissä menetelmissä, nopea kertaus..

Ketterien menetelmien testauskäytänteitä

- Testauksen rooli ketterissä menetelmissä poikkeaa huomattavasti vesiputousmallisesta ohjelmistotuotannosta
  - Testaus on integroitu kehitysprosessiin ja testaajat työskentelevät osana kehittäjätiimejä
  - Testausta tapahtuu projektin ”ensimmäisestä päivästä” lähtien
  - Toteutuksen iteratiivisuus tekee regressiotestauksen automatisoinnista erityisen tärkeää
- Viimeksi puhuimme neljästä ketterästä testaamisen menetelmästä:
  - **Test driven development (TDD)**
  - **Acceptance Test Driven Development / Behavior Driven Development**
    - Etenkin TDD:ssä on on kyse enemmän ohjelman suunnittelusta kuin testaamisesta. sivutuotteena syntyy toki kattava joukko testejä
  - **Continuous Integration (CI)** suomeksi jatkuva integraatio
    - Moderni kehitys on kulkenut kohti **Continuous deploymentiä** eli automaattisesti tapahtuvaa jatkuvaa tuotantoonvientiä
  - **Exploratory testing**, suomeksi tutkiva testaus

Ohjelmiston suunnittelu

Ohjelmiston suunnittelu ja toteutus

- Riippumatta tyylistä ja tavasta jolla ohjelmisto tehdään, ohjelmistojen tekeminen sisältää
  - vaatimusten analysoinnin ja määrittelyn
  - suunnittelun
  - toteuttamisen
  - testauksen ja
  - ohjelmiston ylläpidon
- Olemme käsitelleet vaatimusmäärittelyä ja testaamista erityisesti ketterien ohjelmistotuotantomenetelmien näkökulmasta
- Siirrymme kurssin ”viimeisessä osassa” käsittelemään ohjelmiston *suunnittelua ja toteuttamista*
  - Osa suunnittelusta tapahtuu vasta toteutusvaiheessa, joten suunnittelun ja toteuttamisen käsittelyä ei ole järkevä eriyttää
- **Ohjelmiston suunnittelun tavoitteena on määritellä miten saadaan toteutettua vaatimusmäärittelyn mukaisella tavalla toimiva ohjelma**

Ohjelmiston suunnittelu

- Suunnittelun ajatellaan yleensä jakautuvan kahteen vaiheeseen:
  - **Arkkitehtuurisuunnittelu**
    - Ohjelman rakenne karkealla tasolla
    - Mistä suuremmista rakennekomponenteista ohjelma koostuu?
    - Miten komponentit yhdistetään, eli komponenttien väliset rajapinnat
  - **Oliosuunnittelu**
    - yksittäisten komponenttien suunnittelu
- Suunnittelun ajoittuminen riippuu käytettävästä tuotantoprosessista:
  - Vesiputousmallissa suunnittelu tapahtuu vaatimusmäärittelyn jälkeen ja ohjelmointi aloitetaan vasta kun suunnittelu valmiina ja dokumentoitu
  - Ketterissä menetelmissä suunnittelua tehdään tarvittava määrä jokaisessa iteraatiossa, tarkkaa suunnitteludokumenttia ei yleensä ole
- Vesiputousmallin mukainen suunnitteluprosessi tuskin on enää juuri missään käytössä, ”jäykimmissäkin” prosesseissa ainakin vaatimusmäärittely ja arkkitehtuurisuunnittelu limittyvät
  - Tarkkaa ja raskasta ennen ohjelmointia tapahtuvaa suunnittelua (BDUF eli Big Design Up Front) toki edelleen tapahtuu ja tietynlaisiin järjestelmiin (hyvin tunnettu sovellusalue, muuttumattomat vaatimukset) se osittain sopiikin

Arkkitehtuurisuunnittelu

Ohjelmiston arkkitehtuuri

- Termiä ohjelmistoarkkitehtuuri (software architecture) on käytetty jo vuosikymmeniä
- Termi on vakiintunut yleiseen käyttöön 2000-luvun aikana ja on siirtynyt mm. ”tärkeää työntekijää” tarkoittavaksi nimikkeeksi
  - Ohjelmistoarkkitehti engl. Software architech
- Useimmilla alan ihmisillä on jonkinlainen kuva siitä, mitä ohjelmiston arkkitehtuurilla tarkoitetaan
  - Kyseessä ohjelmiston rakenteen suuret linjat
- Termiä ei ole kuitenkaan yrityksistä huolimatta onnistuttu määrittelemään siten että asiantuntijat olisivat määritelmästä yksimielisiä
- IEEE:n standardi *Recommended practices for Architectural descriptions of Software intensive systems* määrittelee käsitteen seuraavasti
  - *Ohjelmiston arkkitehtuuri on järjestelmän perusorganisaatio, joka sisältää järjestelmän osat, osien keskinäiset suhteet, osien suhteet ympäristöön sekä periaatteet, jotka ohjaavat järjestelmän suunnittelua ja evoluutiota*

Ohjelmiston arkkitehtuuri, muita määritelmiä

- Krutchten:
  - An architecture is the **set of significant decisions about the organization of a software system**, the selection of structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements, the composition of these elements into progressively larger subsystems, and the **architectural style** that guides this organization -- these elements and their interfaces, their collaborations, and their composition.
- McGovern:
  - The software architecture of a system or a collection of systems consists of all the important design decisions about the software structures and the interactions between those structures that comprise the systems. **The design decisions support a desired set of qualities that the system should support to be successful**. The design decisions provide a conceptual basis for system development, support, and maintenance.

Arkkitehtuuriin kuuluu

- Vaikka arkkitehtuurin määritelmät hieman vaihtelevat, löytyy määritelmistä joukko samoja teemoja
- Lähes jokaisen määritelmän mukaan arkkitehtuuri määrittelee ohjelmiston rakenteen, eli jakautumisen erillisiin osiin ja osien väliset rajapinnat
- Arkkitehtuuri ottaa kantaa rakenteen lisäksi myös käyttäytymiseen
  - Arkkitehtuuritason rakenneosien vastuut ja niiden keskinäisen kommunikoinnin muodot
- Arkkitehtuuri keskittyy järjestelmän tärkeisiin/keskeisiin osiin
  - Arkkitehtuuri ei siis kuvaa järjestelmää kokonaisuudessaan vaan on isoihin linjoihin keskittyvä abstraktio
  - Tärkeät osat voivat myös muuttua ajan myötä, eli arkkitehtuuri ei ole muuttumaton
  - http://www.ibm.com/developerworks/rational/library/feb06/eeles/

”Arkkitehtuuri on ne asiat joita on vaikea vaihtaa”

- Artikkelissa ”Who needs arcitect” Martin Fowler toteaa seuraavasti
  - you might end up defining architecture as **things that people perceive as hard to change**
  - https://martinfowler.com/ieeeSoftware/whoNeedsArchitect.pdf
- Melkein sama hieman toisin ilmaistuna oli Krutchtenin määritelmässä
  - **set of significant decisions about the organization of a software system**
- Konkreettinen esimerkki tälläisestä arkkitehtoonisesta päätöksestä tuli esiin miniprojektien ensimmäisellä viikolla
- Monissa projekteissa oli seuraava User story
  - *Ohjelmaa tulee voida käyttää useilta eri koneilta*
- Story ei varmasti ole millään projektilla priorisoitu kovin korkealle, ja ei ole ainakaan ensimmäisen sprintin tavoitteissa

”Arkkitehtuuri on ne asiat joita on vaikea vaihtaa”

- Kaikki ryhmät tekevät kuitenkin jo ensimmäisellä viikolla tärkeän ratkaisun **(significant decisions about …)** sen suhteen miten ohjelma toteutetaan:
  - konsolisovelluksesta
  - Java Swingillä
  - Web-sovelluksena
- Tällä ”arkitehtoonisella päätöksellä” on erittäin suuri vaikutus miten story

  Ohjelmaa tulee voida käyttää useilta eri koneilta

  voidaan toteuttaa siinä vaiheessa kun sen toteututuksen aika tule
- Tämä ensimmäisen viikon tärkeä arkitehtooninen päätös siis johtaa tilanteeseen, joka on myöhemmin **hard to change**
  - Esim. siirtyminen konsolisovelluksesta WebMVC-sovellukseen ei ole suinkaan mahdoton, mutta se edellyttää paljon töitä

Arkkitehtuuriin vaikuttavia tekijöitä

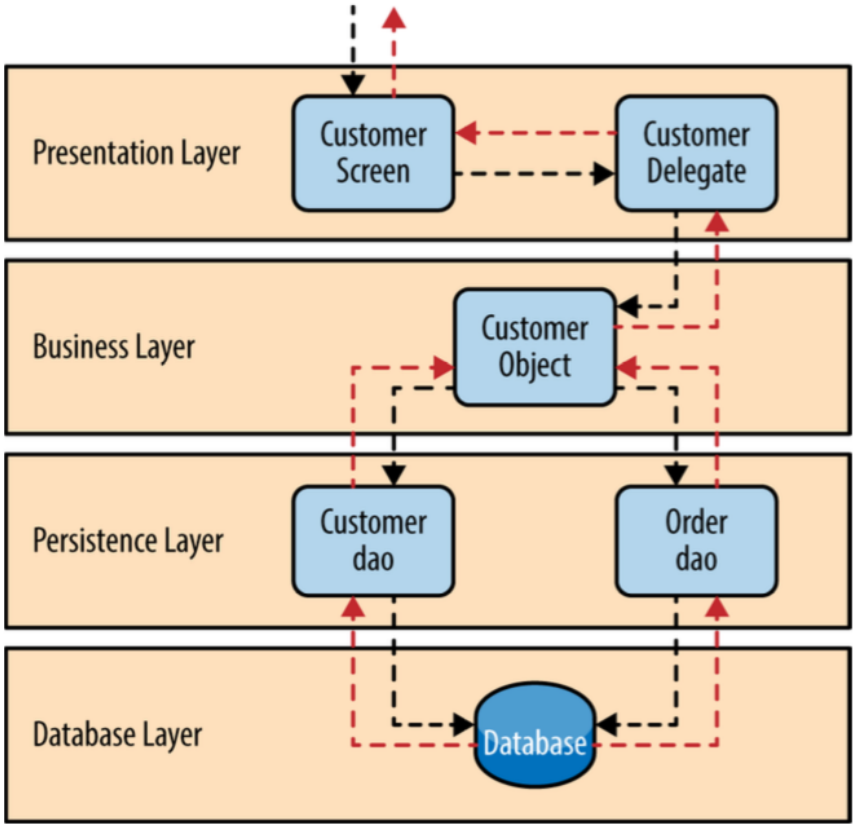
- Järjestelmälle asetetuilla *ei-toiminnallisilla laatuvaatimuksilla* (engl. -ilities) on suuri vaikutus arkkitehtuuriin
  - Käytettävyys, suorituskyky, skaalautuvuus, vikasietoisuus, tiedon ajantasaisuus, tietoturva, ylläpidettävyys, laajennettavuus, hinta, time-to-market, ...
- Laatuvaatimukset ovat usein ristiriitaisia, joten arkkitehdin tulee hakea kaikkia sidosryhmiä tyydyttävä kompromissi
  - Esim. time-to-market lienee ristiriidassa useimpien laatuvaatimusten kanssa
  - Tiedon ajantasaisuus, skaalautuvuus ja vikasietoisuus ovat myös piirteitä, joiden suhteen on pakko tehdä kompromisseja, kaikkia ei voida saavuttaa ks. http://en.wikipedia.org/wiki/CAP_theorem
- Myös järjestelmän toimintaympäristö ja valitut toteutusteknologiat muokkaavat arkkitehtuuria
  - Organisaation standardit
  - Integraatio olemassaoleviin järjestelmiin
  - Toteutuksessa käytettävät sovelluskehykset

Arkkitehtuurimalli

- Järjestelmän arkkitehtuuri perustuu yleensä yhteen tai useampaan **arkkitehtuurimalliin** (architectural pattern), jolla tarkoitetaan hyväksi havaittua tapaa strukturoida tietyntyyppisiä sovelluksia
  - Samasta asiasta käytetään joskus nimitystä **arkkitehtuurityyli** (architectural style)
- Arkkitehtuurimalleja on suuri määrä, esim:
  - Kerrosarkkitehtuuri
  - MVC
  - Pipes-and-filters
  - Repository
  - Client-server
  - publish-subscribe
  - event driven
  - REST
  - Microservice
  - SOA
- Useimmiten sovelluksen rakenteesta löytyy monien arkkitehtuuristen mallien piirteitä

Kerrosarkkitehtuuri

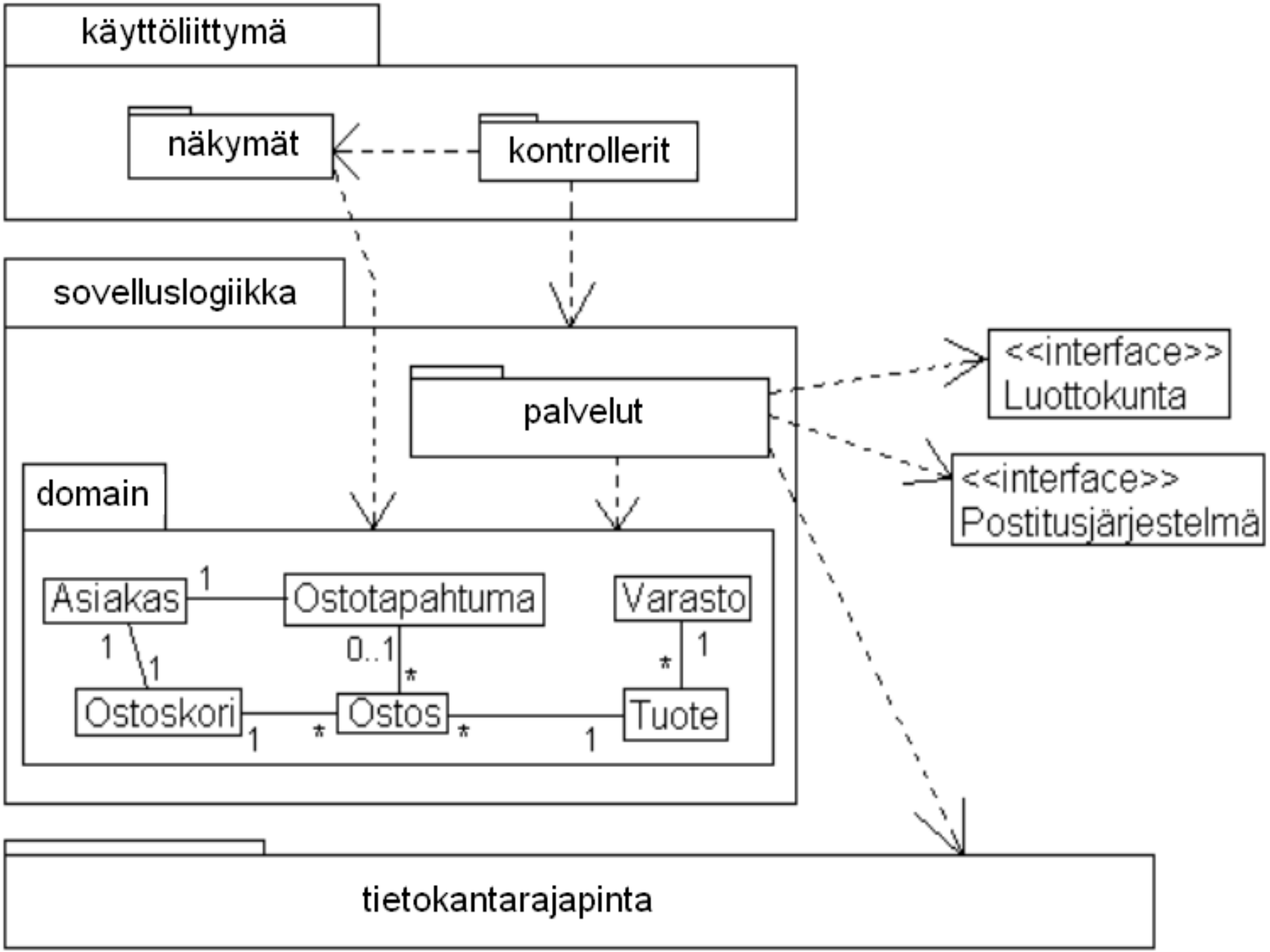
- Eräs tunnetuimmista on *kerrosarkkitehtuurimalli*
  - Kerros on kokoelma toisiinsa liittyviä olioita tai alikomponentteja, jotka muodostavat toiminnallisuuden suhteen loogisen kokonaisuuden
  - Pyrkimyksenä järjestellä komponentit siten, että ylempänä oleva kerros käyttää ainoastaan alempana olevien kerroksien tarjoamia palveluita



- Kerrosarkkitehtuuri on sovelluskehittäjän kannalta selkeä mutta saattaa johtaa massiivisiin monoliittisiin sovelluksiin, joita on lopulta vaikea laajentaa ja joiden skaalaaminen suurille käyttäjämäärille voi muodostua ongelmaksi

Kumpulabeershop

- Seuraavalla sivulla kuvaus Kumpulabiershopin arkkitehtuurista
- Arkkitehtuuri on mukaelma kerrosarkkitehtuuria (layered architecture) ja MVC-mallia
  - Kuvaus on UML-pakkauskaaviona, näyttäen osin myös pakkausten sisäisiä luokkia
  - Luokkatasolle ei yleensä arkkitehtuurikuvauksissa mennä
- Koodi osoitteessa https://github.com/mluukkai/BeerShop
- Koodin tasolla arkkitehtuuri ilmenee luokkien sijoittelusta pakkauksiin
- Ohjelman kaikki koodi on nyt yhdessä projektissa
- Laajempien ohjelmien tapauksessa voi olla tarkoituksenmukaista jakaa koodi useampaan eri projektiin, erityisesti jos sovelluksessa on komponentteja, joita hyödynnetään useimmissa ohjelmissa
- Jos sovellus koostuu eri projekteissa toteutetuista komponenteista, ”pääprojekti” sisällyttää silloin aliprojekteissa toteutetut komponentit esim. gradle-riippuvuuksina

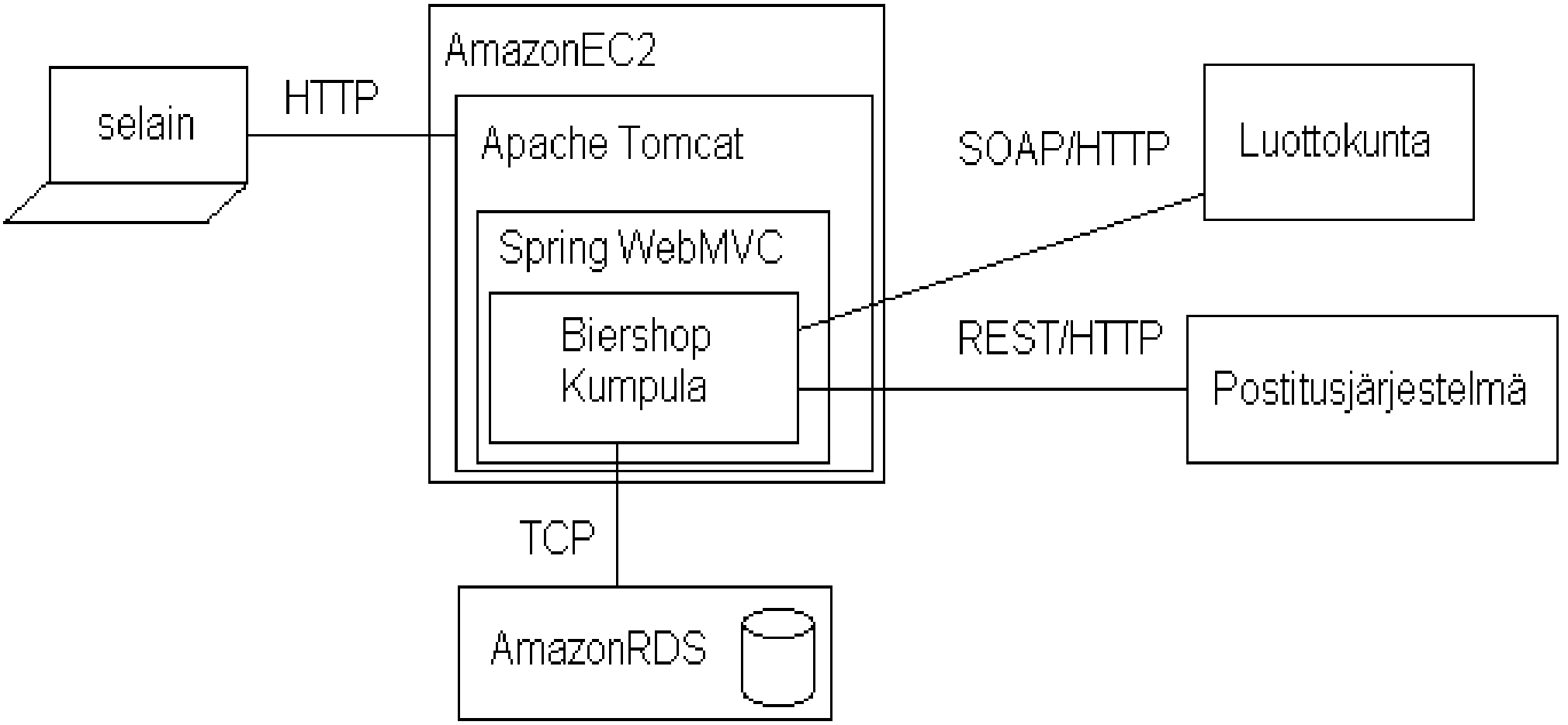

käyttöliittymä
näkymät
kontrollerit
sovelluslogiikka
palvelut
<<interface>>
Luottokunta
<<interface>>
Postitusjärjestelmä
domain
Asiakas
Ostotapahtuma
Varasto
Ostoskori
Ostos
Tuote
1
1
1
1
0..1
*
*
1
*
*
1
tietokantarajapinta

Kumpula biershopin arkkitehtuuri

- Arkkitehtuurikuvaus näyttää järjestelmän jakaantumisen kolmeen kerroksittain järjestettyyn komponenttiin
  - Käyttöliittymä
  - Sovelluslogiikka
  - Tietokantarajapinta
- Sovelluslogiikkakerros on jaettu vielä kahteen alikomponenttiin, sovellusalueen käsitteistön sisältävään *domainiin* ja sen olioita käyttäviin sekä tietokantarajapinnan kanssa keskusteleviin palveluihin
- Käyttöliittymäkerros on myös jakautunut kahteen osaan, näkymään ja kontrollereihin
  - Käytännössä näkymällä tarkoitetaan HTML-tiedostoja
  - Kontrollereilla taas tarkoitetaan main-metodin sisällä olevia selaimen tekemien pyyntöjen käsittelymetodeja
- Kuva tarjoaa **loogisen näkymän** arkkitehtuuriin mutta ei ota kantaa siihen mihin eri komponentit sijoitellaan, eli toimiiko esim. käyttöliittymä samassa koneessa kuin sovelluksen käyttämä tietokanta

Kumpula biershopin arkkitehtuuri

- Alla **fyysisen tason kuvaus,** josta selviää että kyseessä on selaimella käytettävä, SpringWebMVC-sovelluskehyksellä tehty sovellus, jota suoritetaan AmazonEC2-palvelimella ja tietokantana on AmazonRDS
  - Myös kommunikointitapa järjestelmän käyttämiin ulkoisiin järjestelmiin (Luottokunta ja Postitusjärjestelmä) selviää kuvasta

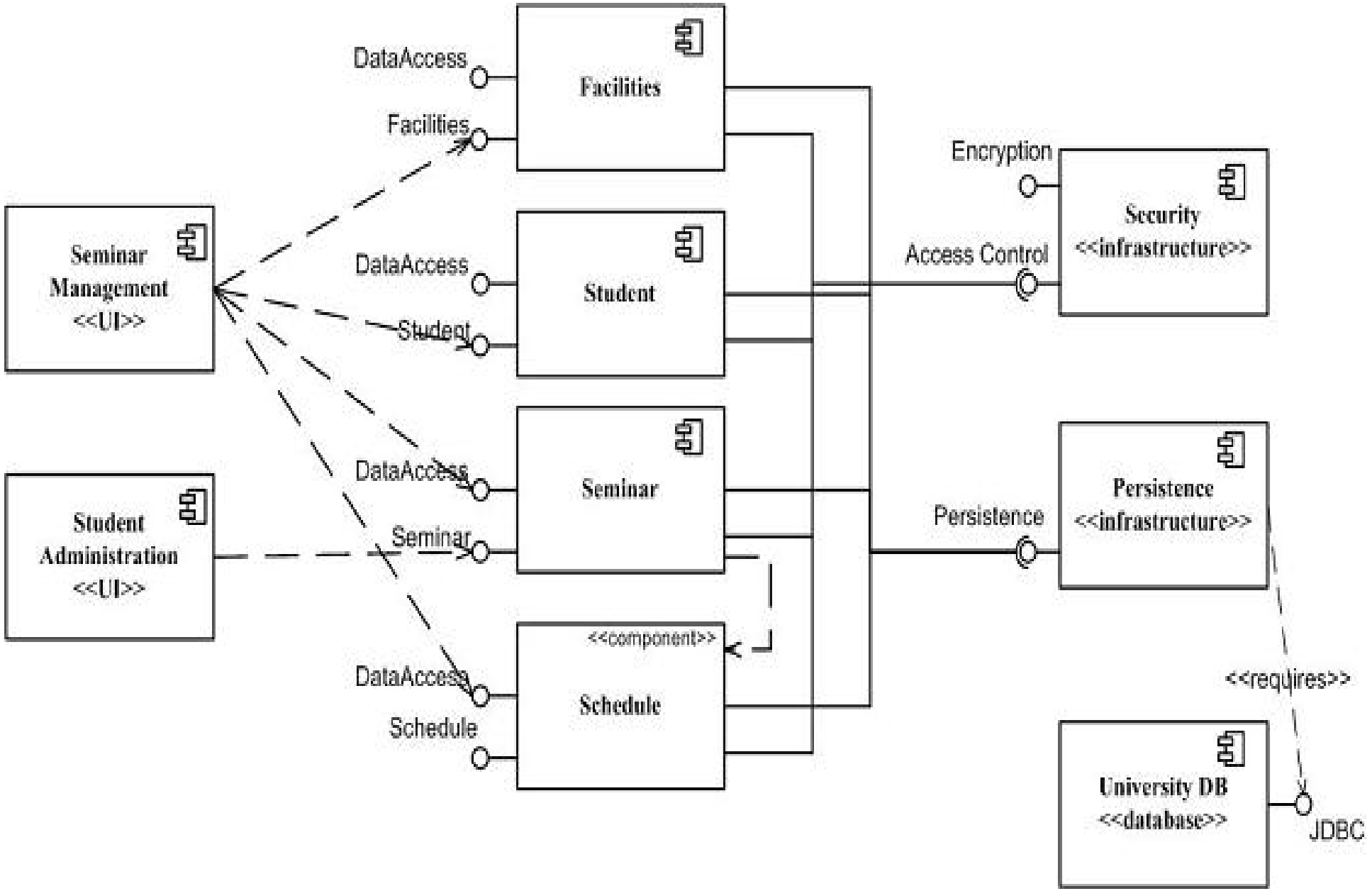


Arkkitehtuurin kuvaamisesta

- UML:n lisäksi arkkitehtuurikuvauksille ei ole vakiintunutta formaattia
  - Luokka ja pakkauskaavioiden lisäksi UML:n komponentti- ja sijoittelukaaviot voivat olla käyttökelpoisia (ks. seuraavat kalvot)
  - Usein käytetään epäformaaleja laatikko/nuoli-kaavioita
- Arkkitehtuurikuvaus kannattaa tehdä useasta eri *näkökulmasta*, sillä eri näkökulmat palvelevat erilaisia tarpeita
  - Korkean tason kuvauksen avulla voidaan strukturoida keskusteluja eri sidosryhmien kanssa, esim.:
    - Vaatimusmäärittelyprosessin jäsentäminen
    - Keskustelut järjestelmäylläpitäjien kanssa
  - Tarkemmat kuvaukset toimivat ohjeena järjestelmän tarkemmassa suunnittelussa ja ylläpitovaiheen aikaisessa laajentamisessa
- Arkkitehtuurikuvaus ei suinkaan ole pelkkä kuva: mm. komponenttien vastuut tulee tarkentaa sekä niiden väliset rajapinnat määritellä
  - Jos näin ei tehdä, kasvaa riski sille että arkkitehtuuria ei noudateta
  - Hyödyllinen kuvaus myös perustelee tehtyjä arkkitehtuurisia valintoja

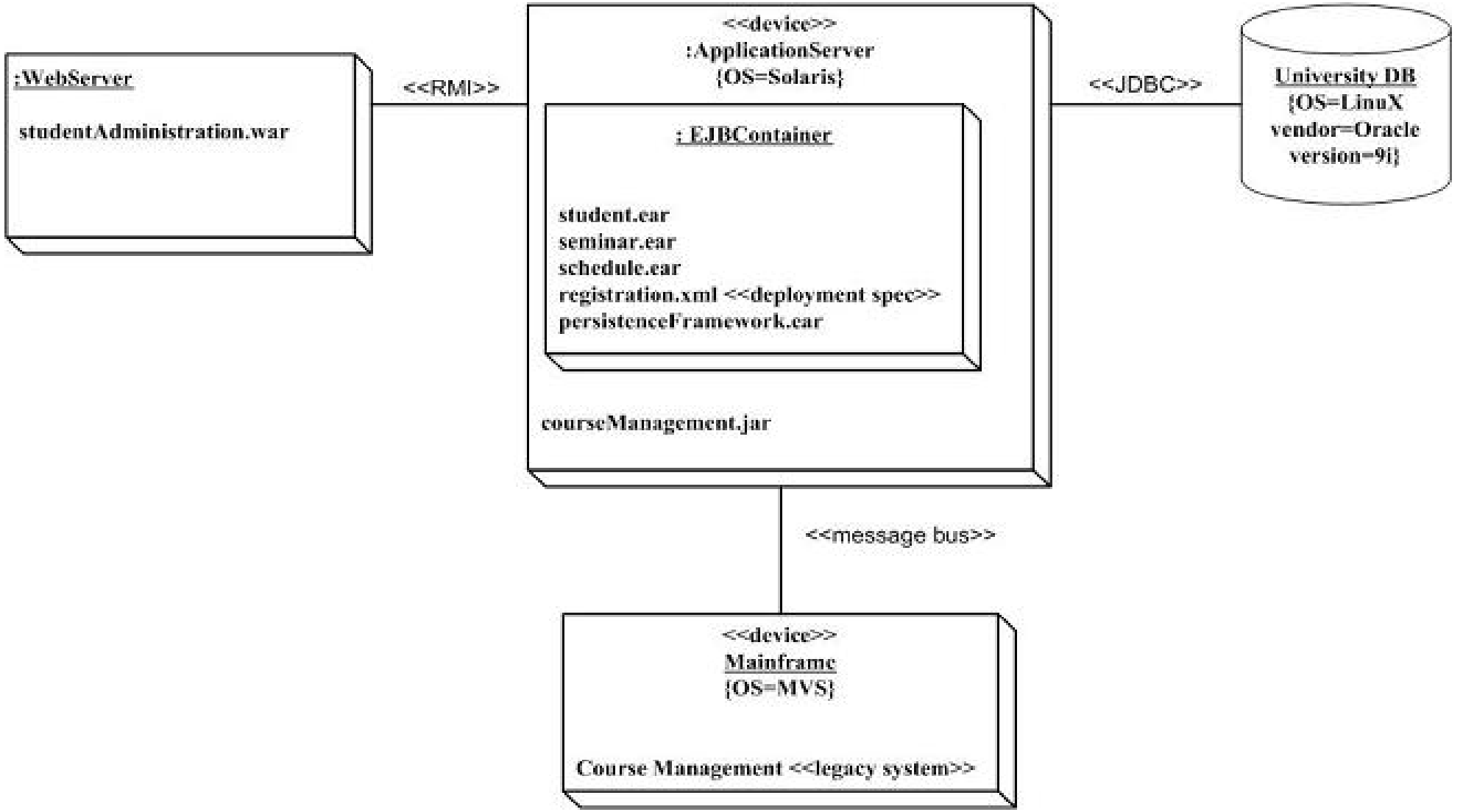
UML komponenttikaavio

- http://www.agilemodeling.com/artifacts/componentDiagram.htm



UML:n sijoittelukaavio

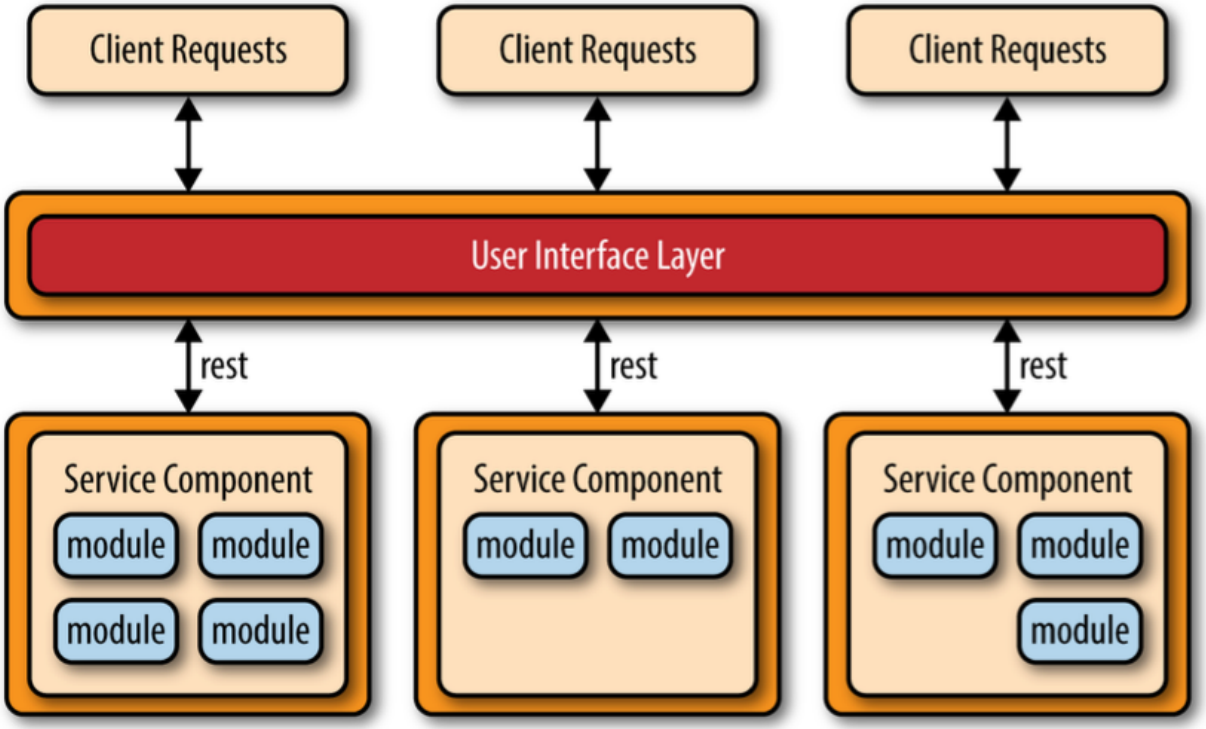
- http://www.agilemodeling.com/artifacts/deploymentDiagram.htm



- UML:n komponentti- ja sijoittelukaavio ovat jossain määrin käyttökelpoisia mutta melko harvoin käytännössä käytettyjä

Hieman lisää arkkitehtuurimalleista

- Tarkastellaan vielä hieman paria arkkitehtuurimallia
- Muutama kalvo sitten todettiin että kerrosarkkitehtuuri saattaa johtaa massiivisiin monoliittisiin sovelluksiin, joita on lopulta vaikea laajentaa ja joiden skaalaaminen suurille käyttäjämäärille voi muodostua ongelmaksi
- Viime aikoina nopeasti yleistynyt **mikropalvelumalli** (microservices) pyrkii vastaamaan näihin haasteisiin koostamalla sovelluksen useista (jopa sadoista) pienistä verkossa toimivista autonomisista palveluista jotka keskenään verkon yli kommunikoiden toteuttavat järjestelmän toiminnallisuuden



Mikropalvelut

- Mikropalveluihin perustuvassa sovelluksessa yksittäisistä palveluista pyritään tekemään mahdollisimman riippumattomia
  - Palvelut eivät esim. käytä yhteistä tietokantaa
  - Palvelut on toteutettu omissa koodiprojekteissaan ja ne eivät jaa koodia
  - Palvelut eivät kutsu toistensa metodeja vaan ne keskustelevat keskenään verkon välityksellä
- Mikropalveluiden on tarkoitus olla pieniä ja huolehtia vain ”yhdestä asiasta”
  - Kun järjestelmään lisätään toiminnallisuutta, se yleensä tarkoittaa uusien palveluiden toteuttamista tai ainoastaan joidenkin palveluiden laajentamista
  - Sovelluksen laajentaminen voi olla helpompaa kuin kerrosarkkitehtuurissa
- Mikropalveluja hyödyntävää sovellusta voi olla helpompi skaalata
  - suorituskyvyn pullonkaulan aiheuttavia mikropalveluja voidaan suorittaa useita rinnakkain

Mikropalvelut

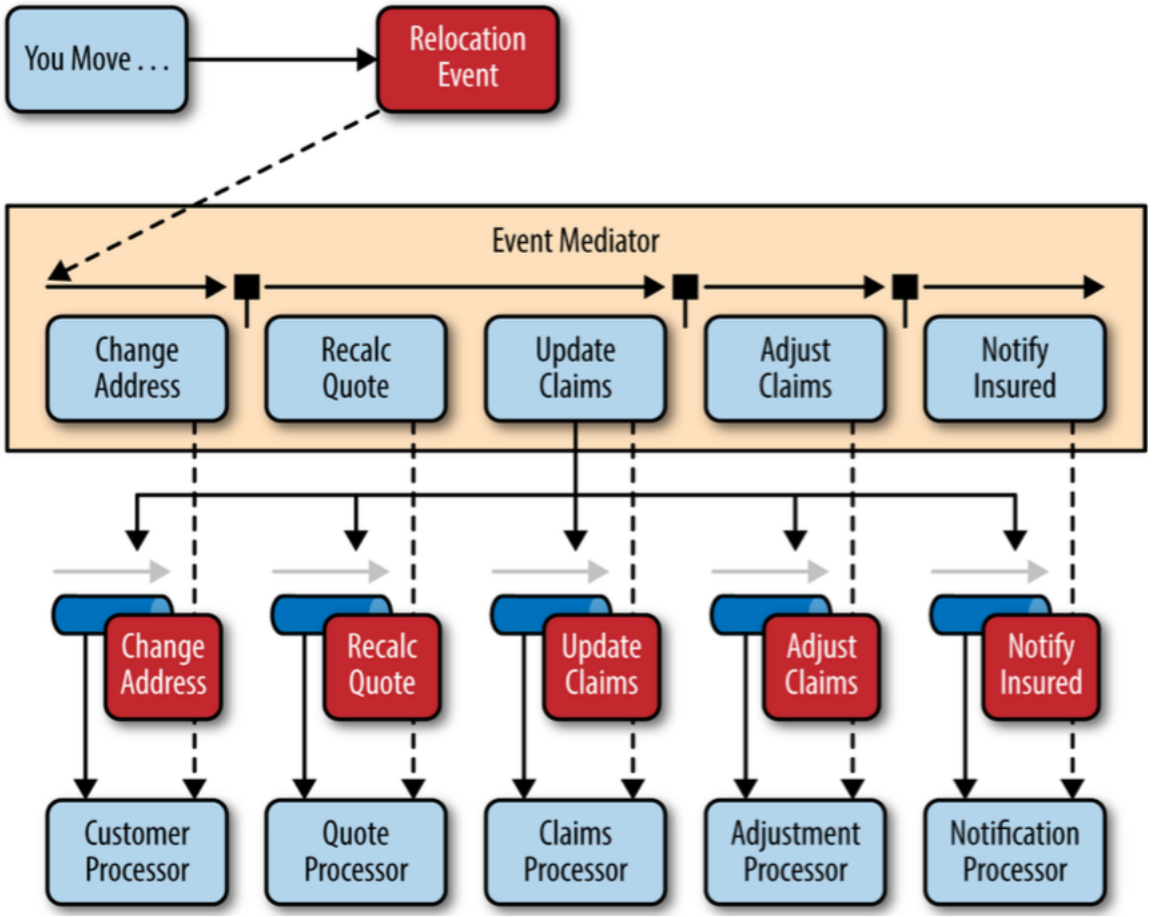
- Mikropalveluiden käyttö mahdollistaa sen, että sovellus voidaan helposti koodata ”monella kielellä”, toisin kuin monoliittisissa projekteissa, mikään ei edellytä, että kaikki mikropalvelut olisi toteutettu samalla kielellä
- Sovelluksen jakaminen järkeviin mikropalveluihin on haastavaa
- Kymmenistä tai jopa sadoista mikropalveluista koostuvan ohjelmiston operoiminen eli ”käynnistäminen” tuotantopalvelimilla on haastavaa ja vaatii pitkälle menevää automatisointia
  - Sama koskee sovelluskehitysympäristöä ja jatkuvaa integraatiota
- Mikropalveluiden yhteydessä käytetäänkin paljon ns *kontainereja* eli käytännössä dockeria
  - Kontainerit ovat hieman yksinkertaistaen sanottuna kevyitä virtuaalikoneita, joita voi suorittaa yhdellä palvelimella suuren määrän rinnakkain
  - Jos mikropalvelu on omassa kontainerissaan, vastaa se käytännössä tilannetta, jossa mikropalvelua suoritettaisiin omalla koneellaan
  - Aihe on tärkeä, mutta emme valitettavasti voi mennä siihen tämän kurssin puitteissa ollenkaan...

Mikropalveluiden kommunikointi

- Mikropalvelut kommunikoivat keskenään verkon välityksellä
- Kommunikointimekanismeja on useita. Yksinkertainen vaihtoehto on käyttää kommunikointiin HTTP-protokollaa, eli samaa mekanismia, jonka avulla web-selaimet keskustelevat palvelimien kanssa
  - Tällöin puhutaan usein että mikropalvelut tarjoavat kommunikointia varten REST-rajapinnan
  - Viikon 4 laskareissa haettiin suorituksiin liittyvää dataa palautusjärjestelmän tarjoamasta REST-rajapinnasta
- Vaihtoehtoinen, huomattavasti joustavampi kommunikeino on ns. viestikanavien käyttö, tällöin voi ajatella, että mikropalveluita on höystetty *event-driven*-arkkitehtuurilla
- Tällöin verkkoon käynnistetään eräänlainen viestinvälityspalvelu, johon muut palvelut voivat lähettää tai **julkaista** (publish) viestejä
  - Viesteillä on tyypillisesti joku aihe (topic) ja sen lisäksi datasisältö
  - Esim: *topic*: new_user, *data*: (username: Arto Hellas, age: 21)
- Palvelut voivat **tilata** (subscribe) viestipalvelulta viestit joista ne ovat kiinnostuneita
  - Esim. käyttäjähallinnasta vastaava palvelu tilaa viestit joiden aihe on *new_user*
- Viestinvälitysjärjestelmä välittää vastaanottamansa viestit kaikille, jotka ovat aiheen tilanneet

Mikropalveluiden kommunikointi viestien välityksellä

- Kaikki viestien (tai joskus puhutaan myös tapahtumista, *event*) välitys tapahtuu viestinvälityspalvelun (kuvassa *event mediator*) kautta
- Näin mikropalveluista tulee erittäin löyhästi kytkettyjä, ja muutokset yhdessä palvelussa eivät vaikuta mihinkään muualle, niin kauan kuin viestit säilyvät entisen muotoisina



- Viestinvälitykseen perustuvat mikropalvelut eivät ole ilmainen lounas, erityisesti debuggaus voi olla välillä melko haastavaa

Arkkitehtuuri ketterissä menetelmissä

Arkkitehtuuri ketterissä menetelmissä

- Ketterien menetelmien kantava teema on toimivan, asiakkaalle arvoa tuottavan ohjelmiston nopea toimittaminen (agile manifestin periaatteita):
  - Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
  - Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Ketterät menetelmät suosivat yksinkertaisuutta suunnitteluratkaisuissa
  - Simplicity--the art of maximizing the amount of work not done--is essential.
  - YAGNI eli ”you are not going to need it”-periaate
- Arkkitehtuuriin suunnittelu ja dokumentointi on perinteisesti ollut melko pitkäkestoinen, ohjelmoinnin aloittamista edeltävä vaihe
  - BUFD eli Big Up Front Design
- Ketterät menetelmät ja ”arkkitehtuurivetoinen” ohjelmistotuotanto ovat siis jossain määrin keskenään ristiriidassa

Arkkitehtuuri ketterissä menetelmissä

- Ketterien menetelmien yhteydessä puhutaan *inkrementaalisesta tai evolutiivisesta suunnittelusta ja arkkitehtuurista*
- Arkkitehtuuri mietitään riittävällä tasolla projektin alussa
- Jotkut projektit alkavat ns. nollasprintillä ja alustava arkkitehtuuri määritellään tällöin
  - Scrumin varhaisissa artikkeleissa puhuttiin ”pre game”-vaiheesta jolloin mm. alustava arkkitehtuuri luodaan
  - Sittemmin koko käsite on hävinnyt Scrumista ja Ken Schwaber (Scrumin kehittäjä) jopa eksplisiittisesti kieltää ja tyrmää koko ”nollasprintin” olemassaolon: http://www.scrum.org/assessmentdiscussion/post/1317787
- Ohjelmiston ”lopullinen” arkkitehtuuri muodostuu iteraatio iteraatiolta samalla kun ohjelmaan toteutetaan uutta toiminnallisuutta
  - Esim. kerrosarkkitehtuurin mukaista sovellusta ei rakenneta ”kerros kerrallaan”
  - Jokaisessa iteraatiossa tehdään pieni pala jokaista kerrosta, sen verran kuin iteraation toiminnallisuuksien toteuttaminen edellyttää
  - http://msdn.microsoft.com/en-us/architecture/ff476940

Walking skeleton

- Yleinen lähestymistapa inkrementaaliseen arkkitehtuuriin on *walking skeletonin* käyttö
  - A Walking Skeleton is a **tiny implementation of the system that performs a small end-to-end function**. It need not use the final architecture, **but it should link together the main architectural components**. The architecture and the functionality can then evolve in parallel.
- What constitutes a walking skeleton varies with the system being designed
  - For a layered architecture system, it is a working connection between all the layers
- The walking skeleton is not complete or robust (it only walks, pardon the phrase), and it is missing the flesh of the application functionality. Incrementally, over time, the infrastructure will be completed and full functionality will be added
- A walking skeleton, is permanent code, built with production coding habits, regression tests, and is intended to grow with the system
- Eli tarkoitus on toteuttaa arkkitehtuurin rungon sisältävä Walking skeleton jo ensimmäisessä sprintissä ja kasvattaa se pikkuhiljaa projektin edetessä
- Katso lisää esim http://alistair.cockburn.us/Walking+skeleton

Arkkitehtuuri ketterissä menetelmissä

- Perinteisesti arkkitehtuurista on vastannut ohjelmistoarkkitehti ja ohjelmoijat ovat olleet velvoitettuja noudattamaan arkkitehtuuria
- Ketterissä menetelmissä ei suosita erillistä arkkitehdin roolia, esim. Scrum käyttää kaikista ryhmän jäsenistä nimikettä *developer*
- Ketterien menetelmien ideaali on, että kehitystiimi luo arkkitehtuurin yhdessä, tämä on myös yksi agile manifestin periaatteista:
  - The best architectures, requirements, and designs emerge from self-organizing teams.
- Arkkitehtuuri on siis koodin tapaan tiimin *yhteisomistama*, tästä on muutamia etuja
  - Kehittäjät sitoutuvat paremmin arkkitehtuurin noudattamiseen kuin ”norsunluutornissa” olevan tiimin ulkopuolisen arkkitehdin määrittelemään arkkitehtuuriin
  - Arkkitehtuurin dokumentointi voi olla kevyt ja informaali (esim. valkotaululle piirretty) sillä tiimi tuntee joka tapauksessa arkkitehtuurin hengen ja pystyy sitä noudattamaan

Inkrementaalinen arkkitehtuuri

- Ketterissä menetelmissä oletuksena on, että parasta mahdollista arkkitehtuuria ei pystytä suunnittelemaan projektin alussa, kun vaatimuksia, toimintaympäristöä ja toteutusteknologioita ei vielä tunneta
  - Jo tehtyjä arkkitehtonisia ratkaisuja muutetaan tarvittaessa
- Eli kuten vaatimusmäärittelyn suhteen, myös arkkitehtuurin suunnittelussa ketterät menetelmät pyrkii välttämään liian aikaisin tehtävää ja myöhemmin todennäköisesti turhaksi osoittautuvaa työtä
- Inkrementaalinen lähestymistapa arkkitehtuurin muodostamiseen edellyttää koodilta hyvää sisäistä laatua ja toteuttajilta kurinalaisuutta
- Martin Fowler http://martinfowler.com/articles/designDead.html:
  - Essentially evolutionary design means that the design of the system grows as the system is implemented. Design is part of the programming processes and as the program evolves the design changes.
  - **In its common usage, evolutionary design is a disaster**. The design ends up being the aggregation of a bunch of ad-hoc tactical decisions, each of which makes the code harder to alter
- Seuraavaksi siirrymme käsittelemään oliosuunnittelua