Ohjelmistotuotanto

Matti Luukkainen ja ohjaajat Jami Kousa, Tero Tapio, Mauri Karlin

syksy 2019

Luento 8

19.11.2019

Miniprojektien aloitustilaisuudet

- Aloitustilaisuudet (jokainen osallistuu yhteen tilaisuuteen)
  - maanantai 18.11. klo 14-16 C222
  - tiistai 19.11. klo 14-16 A128 Chemicum
  - keskiviikko 20.11. klo 12-14 C222
  - torstai 21.11. klo 14-16 C222
- Loppudemot (jokainen ryhmä osallistuu toiseen demoista)
  - maanantai 9.12. klo 14-17
  - tiistai 10.12. klo 14-17

- Riippumatta tyylistä ja tavasta jolla ohjelmisto tehdään, ohjelmistojen tekeminen sisältää
  - vaatimusten analysoinnin ja määrittelyn
  - suunnittelun
  - toteuttamisen
  - testauksen ja
  - ohjelmiston ylläpidon
- Vaatimusmäärittelytä ja testausta käsitelty

- Riippumatta tyylistä ja tavasta jolla ohjelmisto tehdään, ohjelmistojen tekeminen sisältää
  - vaatimusten analysoinnin ja määrittelyn
  - suunnittelun
  - toteuttamisen
  - testauksen ja
  - ohjelmiston ylläpidon
- Vaatimusmäärittelytä ja testausta käsitelty
- Siirrymme käsittelemään ohjelmiston suunnittelua ja toteuttamista
  - osa suunnittelusta tapahtuu vasta toteutusvaiheessa, joten suunnittelun ja toteuttamisen käsittelyä ei voi eriyttää

- Riippumatta tyylistä ja tavasta jolla ohjelmisto tehdään, ohjelmistojen tekeminen sisältää
  - vaatimusten analysoinnin ja määrittelyn
  - suunnittelun
  - toteuttamisen
  - testauksen ja
  - ohjelmiston ylläpidon
- Vaatimusmäärittelytä ja testausta käsitelty
- Siirrymme käsittelemään ohjelmiston suunnittelua ja toteuttamista
  - osa suunnittelusta tapahtuu vasta toteutusvaiheessa, joten suunnittelun ja toteuttamisen käsittelyä ei voi eriyttää
- Suunnittelun tavoite *miten saadaan toteutettua vaatimusmäärittelyn mukaisella tavalla toimiva ohjelma*

Ohjelmiston suunnittelu

- Jakautuu kahteen vaiheeseen:
  - arkkitehtuurisuunnittelu
  - olio/komponenttisuunnittelu

Ohjelmiston suunnittelu

- Jakautuu kahteen vaiheeseen:
  - arkkitehtuurisuunnittelu
  - olio/komponenttisuunnittelu
- Ajoittuminen riippuu käytettävästä tuotantoprosessista:
  - vesiputousmallissa vaatimusmäärittelyn jälkeen, ennen toteutuksen aloittamista, tarkasti dokumentoitu
  - ketterissä menetelmissä suunnittelua tehdään tarvittava määrä jokaisessa iteraatiossa, ei suunnitteludokumenttia

Ohjelmiston suunnittelu

- Jakautuu kahteen vaiheeseen:
  - arkkitehtuurisuunnittelu
  - olio/komponenttisuunnittelu
- Ajoittuminen riippuu käytettävästä tuotantoprosessista:
  - vesiputousmallissa vaatimusmäärittelyn jälkeen, ennen toteutuksen aloittamista, tarkasti dokumentoitu
  - ketterissä menetelmissä suunnittelua tehdään tarvittava määrä jokaisessa iteraatiossa, ei suunnitteludokumenttia
- Vesiputousmallin suunnitteluprosessi tuskin on enää käytössä
  - "jäykimmissäkin" prosesseissa ainakin vaatimusmäärittely ja arkkitehtuurisuunnittelu limittyvät
- Tarkkaa ennen ohjelmointia tapahtuvaa suunnittelua toki edelleen tapahtuu ja joihinkin tilanteisiin se sopiikin

Ohjelmiston arkkitehtuuri

- *Ohjelmiston arkkitehtuuri on järjestelmän perusorganisaatio, joka sisältää järjestelmän osat, osien keskinäiset suhteet, osien suhteet ympäristöön sekä periaatteet, jotka ohjaavat järjestelmän suunnittelua ja evoluutiota* (IEEE)

Ohjelmiston arkkitehtuuri

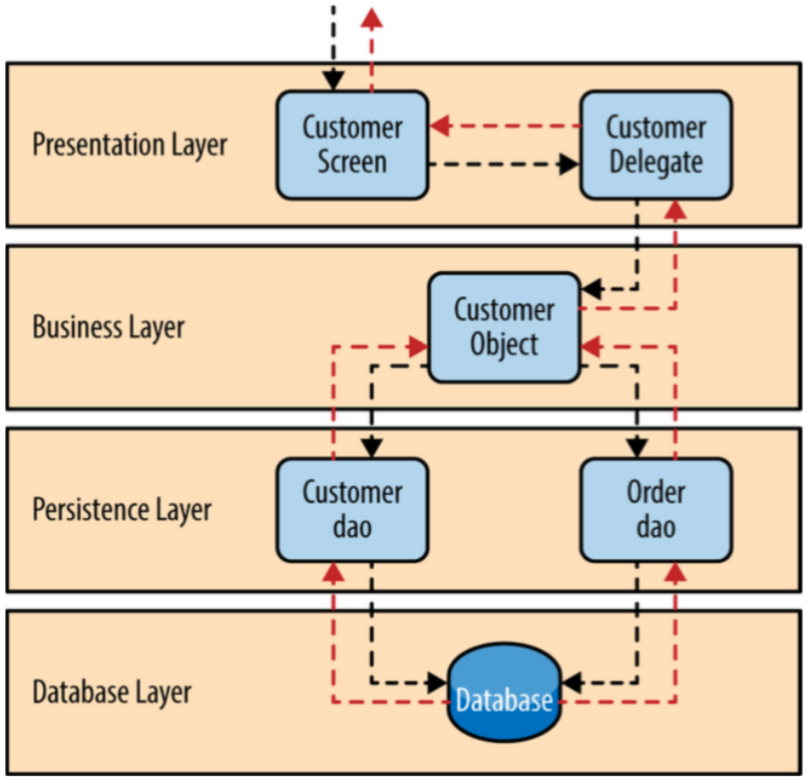
- *Ohjelmiston arkkitehtuuri on järjestelmän perusorganisaatio, joka sisältää järjestelmän osat, osien keskinäiset suhteet, osien suhteet ympäristöön sekä periaatteet, jotka ohjaavat järjestelmän suunnittelua ja evoluutiota* (IEEE)
- Järjestelmälle asetetuilla ei-toiminnallisilla laatuvaatimuksilla (engl. -ilities) on suuri vaikutus arkkitehtuuriin
  - käytettävyys, suorituskyky, skaalautuvuus, vikasietoisuus, tiedon ajantasaisuus, tietoturva, ylläpidettävyys, laajennettavuus, hinta, time-to-market, ...
- Myös toimintaympäristö vaikuttavaa arkkitehtuuriin
  - integraatiot muihin järjestelmiin, käytettävät sovelluskehykset ja tietokannat, lainsäädäntö . . .
- Arkkitehtuuri syntyy joukosta *arkkitehtuurisia valintoja* (...set of significant decisions about the organization of a software system)

Arkkitehtuurityyli

- Ohjelmiston arkkitehtuuri perustuu yleensä yhteen tai useampaan *arkkitehtuurityyliin* (architectural style)
  - hyväksi havaittua tapaa strukturoida tietyntyyppisiä sovelluksia
- Tyylejä suuri määrä
  - Kerrosarkkitehtuuri
  - MVC
  - Pipes-and-filters
  - Repository
  - Client-server
  - Publish-subscribe
  - Event driven
  - REST
  - Microservice

Kerrosarkkitehtuuri

- *Kerros* on kokoelma toisiinsa liittyviä olioita komponentteja, jotka muodostavat toiminnallisuuden suhteen loogisen kokonaisuuden
  - Kerros käyttää ainoastaan alempana olevan kerroksen palveluita



- Kerrokset omalla abstraktiotasollaan
  - Ylimmät kerrokset ovat lähellä käyttäjää: UI ja sovelluslogiikka
  - Alimmat kerrokset taas keskittyvät koneläheisiin asioihin: esim. tiedon tallennus

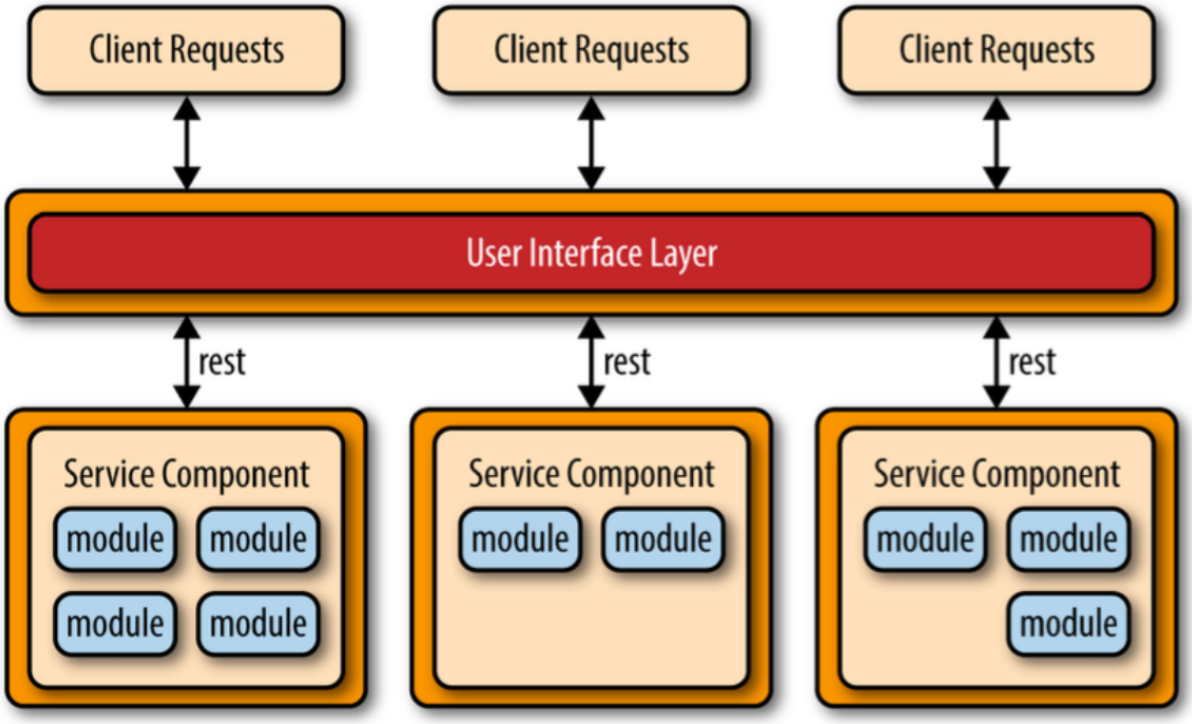
- Kerrokset omalla abstraktiotasollaan
  - Ylimmät kerrokset ovat lähellä käyttäjää: UI ja sovelluslogiikka
  - Alimmat kerrokset taas keskittyvät koneläheisiin asioihin: esim. tiedon tallennus
- Kerroksittaisuus helpottaa ylläpitoa
  - yhden kerroksen muutokset vaikuttavat korkeintaan yläpuolella olevaan kerrokseen

- Kerrokset omalla abstraktiotasollaan
  - Ylimmät kerrokset ovat lähellä käyttäjää: UI ja sovelluslogiikka
  - Alimmat kerrokset taas keskittyvät koneläheisiin asioihin: esim. tiedon tallennus
- Kerroksittaisuus helpottaa ylläpitoa
  - yhden kerroksen muutokset vaikuttavat korkeintaan yläpuolella olevaan kerrokseen
- Sovelluslogiikan riippumattomuus käyttöliittymästä helpottaa ohjelman siirtämistä uusille alustoille
- Alimpien kerroksien palveluja, voidaan osin uusiokäyttää myös muissa sovelluksissa

- Kerrokset omalla abstraktiotasollaan
  - Ylimmät kerrokset ovat lähellä käyttäjää: UI ja sovelluslogiikka
  - Alimmat kerrokset taas keskittyvät koneläheisiin asioihin: esim. tiedon tallennus
- Kerroksittaisuus helpottaa ylläpitoa
  - yhden kerroksen muutokset vaikuttavat korkeintaan yläpuolella olevaan kerrokseen
- Sovelluslogiikan riippumattomuus käyttöliittymästä helpottaa ohjelman siirtämistä uusille alustoille
- Alimpien kerroksien palveluja, voidaan osin uusiokäyttää myös muissa sovelluksissa
- Saattaa johtaa massiivisiin monoliittisiin sovelluksiin, joita on vaikea laajentaa ja skaalaata suurille käyttäjämäärille

Mikropalveluarkkitehtuuri

- Mikropalveluarkkitehtuuri (microservice) pyrkii vastaamaan näihin haasteisiin
  - sovellus koostataan useista (jopa sadoista) pienistä verkossa toimivista autonomisista palveluista
  - jotka keskenään verkon yli kommunikoiden toteuttavat järjestelmän toiminnallisuuden



Mikropalveluarkkitehtuuri

- Yksittäisistä palveluista pyritään tekemään mahdollisimman riippumattomia
  - palvelut eivät kutsu toistensa metodeja, kommunikointi aina verkon välityksellä
  - eivät käytä yhteistä tietokantaa
  - eivät jaa koodia

Mikropalveluarkkitehtuuri

- Yksittäisistä palveluista pyritään tekemään mahdollisimman riippumattomia
  - palvelut eivät kutsu toistensa metodeja, kommunikointi aina verkon välityksellä
  - eivät käytä yhteistä tietokantaa
  - eivät jaa koodia
- Mikropalveluiden ovat pieniä ja huolehtia vain “yhdestä asiasta”
- Verkkokaupan mikropalveluita voisivat olla
  - käyttäjien hallinta
  - tuotteiden suosittelu
  - tuotteiden hakutoiminnot
  - ostoskorin toiminnallisuus
  - ostosten maksusta huolehtiva toiminnallisuus

Etuja

- Kun järjestelmään lisätään toiminnallisuutta, se yleensä tarkoittaa uusien palveluiden toteuttamista tai ainoastaan joidenkin palveluiden laajentamista
  - Sovelluksen laajentaminen voi olla helpompaa kuin kerrosarkkitehtuurissa

Etuja

- 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
- Mikropalveluiden käyttö mahdollistaa sen, että sovellus voidaan helposti koodata “monella kielellä”, toisin kuin monoliittisissa projekteissa

Haasteita

- Sovelluksen jakaminen järkeviin mikropalveluihin on vaikeaa
- Testaaminen ja debuggaus voi olla vaikeaa koska asioita tapahtuu niin monessa paikassa

Haasteita

- Sovelluksen jakaminen järkeviin mikropalveluihin on vaikeaa
- Testaaminen ja debuggaus voi olla vaikeaa koska asioita tapahtuu niin monessa paikassa
- Kymmenistä tai jopa sadoista mikropalveluista koostuvan ohjelmiston operoiminen tuotantopalvelimilla on haastavaa ja vaatii pitkälle menevää automatisointia
- Sama koskee sovelluskehitysympäristöä ja jatkuvaa integraatiota

Haasteita

- Sovelluksen jakaminen järkeviin mikropalveluihin on vaikeaa
- Testaaminen ja debuggaus voi olla vaikeaa koska asioita tapahtuu niin monessa paikassa
- Kymmenistä tai jopa sadoista mikropalveluista koostuvan ohjelmiston operoiminen tuotantopalvelimilla on haastavaa ja vaatii pitkälle menevää automatisointia
- Sama koskee sovelluskehitysympäristöä ja jatkuvaa integraatiota
- Mikropalveluiden menestyksekäs soveltaminen edellyttää vahvaa DevOps-kulttuuria
- Mikropalveluiden yhteydessä käytetäänkin paljon ns kontainereja eli käytännössä dockeria

Arkkitehtuurin kuvaamisesta

- On tilanteita, missä sovelluksen arkkitehtuuri täytyy dokumentoitava jollain tavalla
- Arkkitehtuurien kuvaamiselle ei olemassa vakiintunutta formaattia
  - Useimmiten käytetään epäformaaleja laatikko/nuoli-kaavioita
  - UML:n luokka- ja pakkauskaaviot sekä komponentti- ja sijoittelukaaviot joskus käyttökelpoisia

Arkkitehtuurin kuvaamisesta

- On tilanteita, missä sovelluksen arkkitehtuuri täytyy dokumentoitava jollain tavalla
- Arkkitehtuurien kuvaamiselle ei olemassa vakiintunutta formaattia
  - Useimmiten käytetään epäformaaleja laatikko/nuoli-kaavioita
  - UML:n luokka- ja pakkauskaaviot sekä komponentti- ja sijoittelukaaviot joskus käyttökelpoisia
- Arkkitehtuurikuvaus kannattaa tehdä useasta eri tarpeita palvelevasta *näkökulmasta*
  - korkean tason kuvauksen voi olla hyödyksi esim. vaatimusmäärittelyssä
  - tarkemmat kuvaukset toimivat ohjeena tarkemmassa suunnittelussa ja ylläpitovaiheen aikaisessa laajentamisessa
- Hyödyllinen arkkitehtuurikuvaus dokumentoi ja perustelee tehtyjä *arkkitehtuurisia valintoja*

Arkkitehtuuri ketterissä menetelmissä

- Ketterien menetelmien kantava teema on toimivan, asiakkaalle arvoa tuottavan ohjelmiston nopea toimittaminen:
  - *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*

Arkkitehtuuri ketterissä menetelmissä

- Ketterien menetelmien kantava teema on toimivan, asiakkaalle arvoa tuottavan ohjelmiston nopea toimittaminen:
  - *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*

Arkkitehtuuri ketterissä menetelmissä

- Ketterien menetelmien kantava teema on toimivan, asiakkaalle arvoa tuottavan ohjelmiston nopea toimittaminen:
  - *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*
- Arkkitehtuuriin suunnittelu ja dokumentointi on perinteisesti pitkäkestoinen, ohjelmoinnin aloittamista edeltävä vaihe
- Ketterät menetelmät ja “arkkitehtuurivetoinen” ohjelmistotuotanto ovat siis jossain määrin keskenään ristiriidassa

Arkkitehtuuri ketterissä menetelmissä

- Ketterien menetelmien yhteydessä puhutaan inkrementaalisesta suunnittelusta ja arkkitehtuurista
- Arkkitehtuuri mietitään riittävällä tasolla projektin alussa
  - Jotkut projektit alkavat ns. nollasprintillä ja alustava arkkitehtuuri määritellään tällöin

Arkkitehtuuri ketterissä menetelmissä

- Ketterien menetelmien yhteydessä puhutaan inkrementaalisesta suunnittelusta ja arkkitehtuurista
- Arkkitehtuuri mietitään riittävällä tasolla projektin alussa
  - Jotkut projektit alkavat ns. nollasprintillä ja alustava arkkitehtuuri määritellään tällöin
- 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
  - *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 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 informaalisillä tiimi tuntee joka tapauksessa arkkitehtuurin hengen ja pystyy

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

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 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 kehittäjiltä kurinalaisuutta, muuten seurauksena on kaaos

Olio-komponenttisuunnittelu

- Sovelluksen arkkitehtuuri antaa raamit, jotka ohjaavat sovelluksen tarkempaa suunnittelua ja toteuttamista
- *Olio- tai komponenttisuunnittelu*
  - tarkentaa arkkitehtuuristen komponenttien väliset rajapinnat sekä hahmottelee ohjelman luokka- tai moduulirakenteen

Olio-komponenttisuunnittelu

- Sovelluksen arkkitehtuuri antaa raamit, jotka ohjaavat sovelluksen tarkempaa suunnittelua ja toteuttamista
- *Olio- tai komponenttisuunnittelu*
  - tarkentaa arkkitehtuuristen komponenttien väliset rajapinnat sekä hahmottelee ohjelman luokka- tai moduulirakenteen
- Vesiputousmallissa komponenttisuunnittelu voi olla dokumentoitu tarkkaan esim. UML:ää käyttäen
- Ketterässä tarkka suunnittelu tehdään vasta ohjelmoitaessa

Olio-komponenttisuunnittelu

- Sovelluksen arkkitehtuuri antaa raamit, jotka ohjaavat sovelluksen tarkempaa suunnittelua ja toteuttamista
- *Olio- tai komponenttisuunnittelu*
  - tarkentaa arkkitehtuuristen komponenttien väliset rajapinnat sekä hahmottelee ohjelman luokka- tai moduulirakenteen
- Vesiputousmallissa komponenttisuunnittelu voi olla dokumentoitu tarkkaan esim. UML:ää käyttäen
- Ketterässä tarkka suunnittelu tehdään vasta ohjelmoitaessa
- Suunnittelussa pyritään maksimoimaan *koodin sisäinen laatu*
  - helppo ylläpidettävyys ja laajennettavuus

Olio-komponenttisuunnittelu

- Sovelluksen arkkitehtuuri antaa raamit, jotka ohjaavat sovelluksen tarkempaa suunnittelua ja toteuttamista
- *Olio- tai komponenttisuunnittelu*
  - tarkentaa arkkitehtuuristen komponenttien väliset rajapinnat sekä hahmottelee ohjelman luokka- tai moduulirakenteen
- Vesiputousmallissa komponenttisuunnittelu voi olla dokumentoitu tarkkaan esim. UML:ää käyttäen
- Ketterässä tarkka suunnittelu tehdään vasta ohjelmoitaessa
- Suunnittelussa pyritään maksimoimaan *koodin sisäinen laatu*
  - helppo ylläpidettävyys ja laajennettavuus
- Ohjelmistosuunnittelu on “enemmän taidetta kuin tiedettä”, kokemus ja hyvien käytänteiden opiskelu toki auttaa
  - kehitetty monia suunnittelumenetelmiä, mikään niistä ei ole vakiintunut

- Ylläpidettävyyden ja laajennettavuuden kannalta hyvällä koodilla joukko yhteneviä ominaisuuksia, tai *laatuattribuutteja*, esim. seuraavat:
  - kapselointi
  - korkea koheesion aste
  - riippuvuuksien vähäisyys
  - toisteettomuus
  - testattavuus
  - selkeys

- Ylläpidettävyyden ja laajennettavuuden kannalta hyvällä koodilla joukko yhteneviä ominaisuuksia, tai *laatuattribuutteja*, esim. seuraavat:
  - kapselointi
  - korkea koheesion aste
  - riippuvuuksien vähäisyys
  - toisteettomuus
  - testattavuus
  - selkeys
- *Suunnittelumallit* auttavat luomaan koodia, joissa sisäinen laatu kunnossa
  - kurssin aikana nähty jo *dependency injection*, *singleton*, *data access object*
  - lisää kurssimateriaalissa ja laskareissa

Koodin laatuattribuutti: kapselointi

- Ohjelmoinnin peruskurssilla *kapselointi* määritellään seuraavasti
  - *oliomuuttujat tulee määritellä privaateiksi ja niille tulee tehdä tarvittaessa setterit ja getterit*

Koodin laatuattribuutti: kapselointi

- Ohjelmoinnin peruskurssilla *kapselointi* määritellään seuraavasti
  - *oliomuuttujat tulee määritellä privaateiksi ja niille tulee tehdä tarvittaessa setterit ja getterit*
- Olion sisäisen tilan lisäksi kapseloinnin kohde voi olla mm. *käytettävän olion tyyppi, käytetty algoritmi, olioiden luomisen tapa, käytettävän komponentin rakenne*

Koodin laatuattribuutti: kapselointi

- Ohjelmoinnin peruskurssilla *kapselointi* määritellään seuraavasti
  - *oliomuuttujat tulee määritellä privaateiksi ja niille tulee tehdä tarvittaessa setterit ja getterit*
- Olion sisäisen tilan lisäksi kapseloinnin kohde voi olla mm. *käytettävän olion tyyppi, käytetty algoritmi, olioiden luomisen tapa, käytettävän komponentin rakenne*
- Näkyy myös arkkitehtuurin tasolla
  - kerrosarkkitehtuuri: ylempi kerros käyttää ainoastaan alemman kerroksen ulospäin tarjoamaa rajapintaa, muu kapseloitu näkymättömiin
  - mikropalvelut: yksittäinen palvelu kapseloi sisäisen logiikan, tiedon säilytystavan ja tarjoaa ainoastaan verkon välityksellä käytettävän rajapinnan

Koodin laatuattribuutti: koheesio

- *Koheesio*:
  - kuinka pitkälle metodissa, luokassa tai komponentissa oleva ohjelmakoodi keskittyy tietyn yksittäisen toiminnallisuuden toteuttamiseen.
  - Hyvänä pidetään mahdollisimman korkeaa koheesion astetta

Koodin laatuattribuutti: koheesio

- *Koheesio*:
  - kuinka pitkälle metodissa, luokassa tai komponentissa oleva ohjelmakoodi keskittyy tietyn yksittäisen toiminnallisuuden toteuttamiseen.
  - Hyvänä pidetään mahdollisimman korkeaa koheesion astetta
- Korkeaan koheesioon tulee pyrkiä kaikilla ohjelman tasoilla, metodeissa, luokissa, komponenteissa ja arkkitehtuurissa

Koodin laatuattribuutti: koheesio

- *Koheesio*:
  - kuinka pitkälle metodissa, luokassa tai komponentissa oleva ohjelmakoodi keskittyy tietyn yksittäisen toiminnallisuuden toteuttamiseen.
  - Hyvänä pidetään mahdollisimman korkeaa koheesion astetta
- Korkeaan koheesioon tulee pyrkiä kaikilla ohjelman tasoilla, metodeissa, luokissa, komponenteissa ja arkkitehtuurissa
- Luokkatason koheesiossa pyrkimyksenä on, että luokan *vastuulla* on vain yksi asia, tunnetaan myös nimellä *single responsibility periaate*

Koodin laatuattribuutti: koheesio

- *Koheesio*:
  - kuinka pitkälle metodissa, luokassa tai komponentissa oleva ohjelmakoodi keskittyy tietyn yksittäisen toiminnallisuuden toteuttamiseen.
  - Hyvänä pidetään mahdollisimman korkeaa koheesion astetta
- Korkeaan koheesioon tulee pyrkiä kaikilla ohjelman tasoilla, metodeissa, luokissa, komponenteissa ja arkkitehtuurissa
- Luokkatason koheesiossa pyrkimyksenä on, että luokan *vastuulla* on vain yksi asia, tunnetaan myös nimellä *single responsibility periaate*
- Arkkitehtuurin tasolla
  - kerrosarkkitehtuurin kerrokset samalla abstraktiotasolla, esim. käyttöliittymä tai tietokantarajapinta
  - mikropalvelu tiettyyn liiketoiminnan tason toiminnallisuuden, esim. suosittelualgoritmi

Koodin laatuattribuutti: riippuvuuksien vähäisyys

- Pyrkimys korkeaan koheesioon johtaa ohjelmiin, joissa suuri määrä olioita/komponentteja
  - Olioiden oltava keskenään vuorovaikutuksessa toteuttaakseen ohjelman toiminnallisuuden, *paljon keskinäisiä riippuvuuksia*

Koodin laatuattribuutti: riippuvuuksien vähäisyys

- Pyrkimys korkeaan koheesioon johtaa ohjelmiin, joissa suuri määrä olioita/komponentteja
  - Olioiden oltava keskenään vuorovaikutuksessa toteuttaakseen ohjelman toiminnallisuuden, *paljon keskinäisiä riippuvuuksia*
- *Riippuvuuksien vähäisyyden* periaate
  - eliminoidaan tarpeettomat riippuvuudet
  - sekä riippuvuudet konkreettisiin asioihin

Koodin laatuattribuutti: riippuvuuksien vähäisyys

- Pyrkimys korkeaan koheesioon johtaa ohjelmiin, joissa suuri määrä olioita/komponentteja
  - Olioiden oltava keskenään vuorovaikutuksessa toteuttaakseen ohjelman toiminnallisuuden, *paljon keskinäisiä riippuvuuksia*
- *Riippuvuuksien vähäisyyden* periaate
  - eliminoidaan tarpeettomat riippuvuudet
  - sekä riippuvuudet konkreettisiin asioihin
- Konkreettisten riippuvuuksien eliminointi kulkee eri nimillä:
  - program to an interface, not to an implementation
  - depend on abstractions, not on concrete implementation
- Hyödynnetään usein meille tuttua *dependence injection* -suunnittelumallia

Koodin laatuattribuutti: toisteettomuus

- Aloittelevaa ohjelmoijaa pelotellaan toisteisuuden vaaroista uran ensiaskelista alkaen: älä copypastaa koodia!

Koodin laatuattribuutti: toisteettomuus

- Aloittelevaa ohjelmoijaa pelotellaan toisteisuuden vaaroista uran ensiaskelista alkaen: älä copypastaa koodia!
- Alan piireissä toisteisuudesta varoittava periaate kulkee nimellä DRY, don't repeat yourself
  - *Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.*

Koodin laatuattribuutti: toisteettomuus

- Aloittelevaa ohjelmoijaa pelotellaan toisteisuuden vaaroista uran ensiaskelista alkaen: älä copypastaa koodia!
- Alan piireissä toisteisuudesta varoittava periaate kulkee nimellä DRY, don't repeat yourself
  - *Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.*
- koodin lisäksi periaate ulottuu koskemaan järjestelmän muitakin osia, kuten tietokantaskeemaa, testejä, build-skriptejä

Koodin laatuattribuutti: toisteettomuus

- Aloittelevaa ohjelmoijaa pelotellaan toisteisuuden vaaroista uran ensiaskelista alkaen: älä copypastaa koodia!
- Alan piireissä toisteisuudesta varoittava periaate kulkee nimellä DRY, don't repeat yourself
  - *Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.*
- koodin lisäksi periaate ulottuu koskemaan järjestelmän muitakin osia, kuten tietokantaskeemaa, testejä, build-skriptejä
- Suoraviivainen copypaste on helppo eliminoida esim. metodien avulla
  - Kaikki toisteisuus ei ole yhtä ilmeistä, monissa suunnittelumalleissa kyse hienovaraisempien toisteisuuden muotojen eliminoinnista

Koodin laatuattribuutti: toisteettomuus

- Aloittelevaa ohjelmoijaa pelotellaan toisteisuuden vaaroista uran ensiaskelista alkaen: älä copypastaa koodia!
- Alan piireissä toisteisuudesta varoittava periaate kulkee nimellä DRY, don't repeat yourself
  - *Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.*
- koodin lisäksi periaate ulottuu koskemaan järjestelmän muitakin osia, kuten tietokantaskeemaa, testejä, build-skriptejä
- Suoraviivainen copypaste on helppo eliminoida esim. metodien avulla
  - Kaikki toisteisuus ei ole yhtä ilmeistä, monissa suunnittelumalleissa kyse hienovaraisempien toisteisuuden muotojen eliminoinnista
- Hyvä vs. paha copypaste
  - *three strikes and you refactor*

Koodin laatuattribuutti: testattavuus

- Koodi on helppo testata kattavasti yksikkö- ja integraatiotestein
  - seuraa yleensä siitä, että koodi koostuu löyhästi kytketyistä, selkeän vastuun omaavista komponenteista
- Jos koodin kattava testaaminen on vaikeaa, on se usein seurausta siitä, että komponenttien vastuut eivät ole selkeät, ja niillä on liikaa riippuvuuksia

Koodin laatuattribuutti: testattavuus

- Koodi on helppo testata kattavasti yksikkö- ja integraatiotestein
  - seuraa yleensä siitä, että koodi koostuu löyhästi kytketyistä, selkeän vastuun omaavista komponenteista
- Jos koodin kattava testaaminen on vaikeaa, on se usein seurausta siitä, että komponenttien vastuut eivät ole selkeät, ja niillä on liikaa riippuvuuksia
- Kurssin alusta asti pyritty hyvään testattavuuteen esim. purkamalla turhia riippuvuuksia rajapintojen ja riippuvuuksien injektoinnin avulla

Koodin laatuattribuutti: selkeys ja luettavuus

- Perinteisesti koodin on ajateltu olevan kryptistä ja vaikeasti luettavaa
  - yleistä C-kielessä, pyritty esim. optimoimaan tehokkuutta ja muistinkäyttöä

Koodin laatuattribuutti: selkeys ja luettavuus

- Perinteisesti koodin on ajateltu olevan kryptistä ja vaikeasti luettavaa
  - yleistä C-kielessä, pyritty esim. optimoimaan tehokkuutta ja muistinkäyttöä
- Nykytrendin mukaista on pyrkiä kirjoittamaan koodia, joka nimeämisen sekä rakenteen kautta ilmaisee mahdollisimman hyvin sen, mitä koodi tekee

Koodin laatuattribuutti: selkeys ja luettavuus

- Perinteisesti koodin on ajateltu olevan kryptistä ja vaikeasti luettavaa
  - yleistä C-kielessä, pyritty esim. optimoimaan tehokkuutta ja muistinkäyttöä
- Nykytrendin mukaista on pyrkiä kirjoittamaan koodia, joka nimeämisen sekä rakenteen kautta ilmaisee mahdollisimman hyvin sen, mitä koodi tekee
- Miksi selkeä koodi on tärkeää?
  - joidenkin arvioiden mukaan jopa 90% “ohjelmointiin” kuluvasta ajasta menee olemassa olevan koodin lukemiseen
  - oma aikoinaan niin selkeä koodi, ei enää olekaan yhtä selkeää parin kuukauden kuluttua

Code smell

- Martin Fowlerin mukaan
  - *koodihaju* (code smell) on helposti huomattava merkki siitä että koodissa on jotain pielessä
  - jopa aloitteleva ohjelmoija saattaa pystyä havaitsemaan koodihajun, sen takana oleva todellinen syy voi olla jossain syvemmällä

Code smell

- Martin Fowlerin mukaan
  - *koodihaju* (code smell) on helposti huomattava merkki siitä että koodissa on jotain pielessä
  - jopa aloitteleva ohjelmoija saattaa pystyä havaitsemaan koodihajun, sen takana oleva todellinen syy voi olla jossain syvemmällä
- Koodihaju siis kertoo, että syystä tai toisesta *koodin sisäinen laatu* ei ole parhaalla mahdollisella tasolla.

Koodihajuja

- Koodihajuja on hyvin monenlaisia ja monentasoisia, esimerkkejä helposti tunnistettavista hajuista:
  - toisteinen koodi
  - liian pitkät metodit
  - luokat joissa on liikaa oliomuuttujia
  - luokat joissa on liikaa koodia
  - metodien liian pitkät parametrilistat
  - epäselkeät muuttujien, metodien tai luokkien nimet
  - kommentit

Koodihajuja

- Koodihajuja on hyvin monenlaisia ja monentasoisia, esimerkkejä helposti tunnistettavista hajuista:
  - toisteinen koodi
  - liian pitkät metodit
  - luokat joissa on liikaa oliomuuttujia
  - luokat joissa on liikaa koodia
  - metodien liian pitkät parametrilistat
  - epäselkeät muuttujien, metodien tai luokkien nimet
  - kommentit
- Pari monimutkaisempaa
  - Primitive obsession
  - Shotgun surgery

Refaktorointi

- Lääke koodin sisäisen laadun ongelmiin on *refaktorointi*
  - muutos koodin rakenteeseen, joka pitää sen toiminnallisuuden ennallaan

Refaktorointi

- Lääke koodin sisäisen laadun ongelmiin on *refaktorointi*
  - muutos koodin rakenteeseen, joka pitää sen toiminnallisuuden ennallaan
- Erilaisia koodin rakennetta parantavia refaktorointeja on lukuisia
  - *rename variable/method/class*
  - *extract method*
  - *move field/method*
  - *extract interface*
  - *extract superclass*
- Osa pystytään tekemään sovelluskehitysympäristön avustamana

Miten refaktorointi kannattaa tehdä

- Refaktoroinnin melkein ehdoton edellytys on kattavien testien olemassaolo
- Kannattaa ehdottomasti edetä pienin askelin
  - Yksi hallittu muutos kerrallaan
  - Testit suoritettava mahdollisimman usein

Miten refaktorointi kannattaa tehdä

- Refaktoroinnin melkein ehdoton edellytys on kattavien testien olemassaolo
- Kannattaa ehdottomasti edetä pienin askelin
  - Yksi hallittu muutos kerrallaan
  - Testit suoritettava mahdollisimman usein
- Refaktorointia kannattaa suorittaa lähes jatkuvasti
  - Koodin ei kannata antaa rapistua pitkiä aikoja
- Lähes jatkuva refaktorointi on helppoa
  - pitää koodin rakenteen selkeänä ja helpottaa sekä nopeuttaa koodin laajentamista

Miten refaktorointi kannattaa tehdä

- Refaktoroinnin melkein ehdoton edellytys on kattavien testien olemassaolo
- Kannattaa ehdottomasti edetä pienin askelin
  - Yksi hallittu muutos kerrallaan
  - Testit suoritettava mahdollisimman usein
- Refaktorointia kannattaa suorittaa lähes jatkuvasti
  - Koodin ei kannata antaa rapistua pitkiä aikoja
- Lähes jatkuva refaktorointi on helppoa
  - pitää koodin rakenteen selkeänä ja helpottaa sekä nopeuttaa koodin laajentamista
- Osa refaktoroinnista on helppoa ja suoraviivaista, aina ei näin ole
- Joskus tarve tehdä isoja, jopa viikkojen kestoisia refaktorointeja joissa ohjelman rakenne muuttuu paljon

Tekninen velka

- Koodi ei ole aina laadultaan optimaalista, joskus on jopa asiakkaan kannalta tarkoituksenmukaista tehdä vähemmän laadukasta koodia
- Huonoa suunnittelua tai/ja ohjelmointia kuvaa käsite *tekninen velka* (engl. technical debt).

Tekninen velka

- Koodi ei ole aina laadultaan optimaalista, joskus on jopa asiakkaan kannalta tarkoituksenmukaista tehdä vähemmän laadukasta koodia
- Huonoa suunnittelua tai/ja ohjelmointia kuvaa käsite *tekninen velka* (engl. technical debt).
- Piittaamattomalla ja laiskalla ohjelmoinnilla/suunnittelulla saadaan ehkä nopeasti aikaan jotain
  - mutta hätäinen ratkaisu tullaan maksamaan korkoineen takaisin *jos* ohjelmaa on tarkoitus laajentaa

Tekninen velka

- Koodi ei ole aina laadultaan optimaalista, joskus on jopa asiakkaan kannalta tarkoituksenmukaista tehdä vähemmän laadukasta koodia
- Huonoa suunnittelua tai/ja ohjelmointia kuvaa käsite *tekninen velka* (engl. technical debt).
- Piittaamattomalla ja laiskalla ohjelmoinnilla/suunnittelulla saadaan ehkä nopeasti aikaan jotain
  - mutta hätäinen ratkaisu tullaan maksamaan korkoineen takaisin *jos* ohjelmaa on tarkoitus laajentaa
- Jos korkojen maksun aikaa ei koskaan tule, voi “huono koodi” olla asiakkaan etu
  - Esim. minimal viable product (MVP)

Tekninen velka

- Koodi ei ole aina laadultaan optimaalista, joskus on jopa asiakkaan kannalta tarkoituksenmukaista tehdä vähemmän laadukasta koodia
- Huonoa suunnittelua tai/ja ohjelmointia kuvaa käsite *tekninen velka* (engl. technical debt).
- Piittaamattomalla ja laiskalla ohjelmoinnilla/suunnittelulla saadaan ehkä nopeasti aikaan jotain
  - mutta hätäinen ratkaisu tullaan maksamaan korkoineen takaisin *jos* ohjelmaa on tarkoitus laajentaa
- Jos korkojen maksun aikaa ei koskaan tule, voi "huono koodi" olla asiakkaan etu
  - Esim. minimal viable product (MVP)
- Lyhytaikainen tekninen velka voi olla järkevää tai jopa välttämätöntä
  - Esim. voidaan saada tuote nopeammin markkinoille tekemällä tietoisesti huonoa designia, joka korjataan myöhemmin

- Teknisen velan taustalla: holtittomuus, osaamattomuus, tietämättömyys tai tarkoituksella tehty päätös

- Teknisen velan taustalla: holtittomuus, osaamattomuus, tietämättömyys tai tarkoituksella tehty päätös
- Kaikki tekninen velka ei ole samanlaista, Martin Fowler jaottelee teknisen velan neljään eri luokkaan:
  - Reckless and deliberate: “we do not have time for design”
    Reckless and inadverent: “what is layering”?
  - Prudent and inadverent: “now we know how we should have done it”
  - Prudent and deliberate: “we must ship now and will deal with consequences”