

Ohjelmistotuotanto

Matti Luukkainen ja ohjaajat Antti, Pooki, Riku, Sini, Taneli

syksy 2024

Luento 8

19.11.2024

Kurssipalaute

- ▶ Kurssipalaute
 - ▶ Kurssilla lopussa kerättävän palautteen lisäksi ns. jatkuva palaute <https://norppa.helsinki.fi>

Ohjelmiston elinkaaren vaiheet

- ▶ Riippumatta tyylistä ja tavasta jolla ohjelmisto tehdään, ohjelmistojen tekemiseen kuuluu
 - ▶ vaatimusten analysointi ja määrittely
 - ▶ **suunnittelu**
 - ▶ **toteutus**
 - ▶ testaus/laadunhallinta
 - ▶ ohjelmiston ylläpito

Ohjelmiston elinkaaren vaiheet

- ▶ Riippumatta tyylistä ja tavasta jolla ohjelmisto tehdään, ohjelmistojen tekemiseen kuuluu
 - ▶ vaatimusten analysointi ja määrittely
 - ▶ **suunnittelu**
 - ▶ **toteutus**
 - ▶ testaus/laadunhallinta
 - ▶ ohjelmiston ylläpito
- ▶ Jakautuu kahteen vaiheeseen:
 - ▶ arkkitehtuurisuunnittelu
 - ▶ olio/komponenttisuunnittelu

Ohjelmiston elinkaaren vaiheet

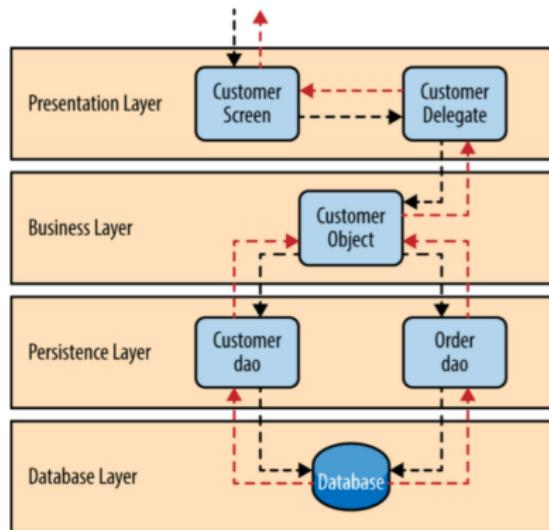
- ▶ Riippumatta tyylistä ja tavasta jolla ohjelmisto tehdään, ohjelmistojen tekemiseen kuuluu
 - ▶ vaatimusten analysointi ja määrittely
 - ▶ **suunnittelu**
 - ▶ **toteutus**
 - ▶ testaus/laadunhallinta
 - ▶ ohjelmiston ylläpito
- ▶ Jakautuu kahteen vaiheeseen:
 - ▶ arkkitehtuurisuunnittelu
 - ▶ olio/komponenttisuunnittelu
- ▶ Näiden lisäksi UI/UX-suunnittelu

Arkkitehtuurityyli

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

Kerrosarkkitehtuuri

- ▶ Kerros on kokoelma toisiinsa liittyviä olioita, jotka muodostavat toiminnallisuuden suhteen loogisen kokonaisuuden

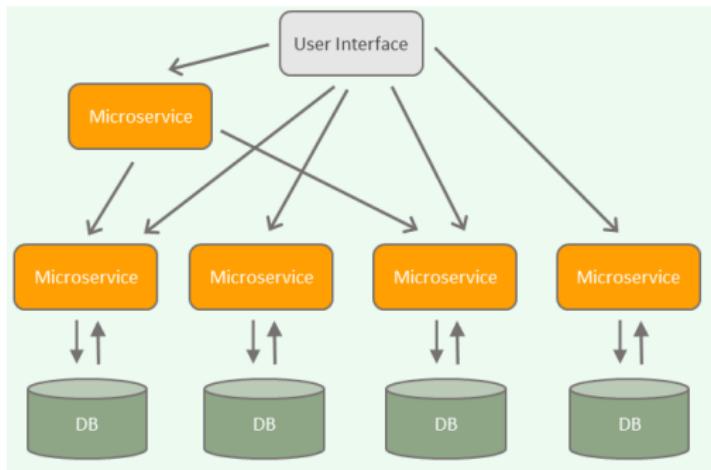


Mikropalveluarkkitehtuuri

- ▶ Mikropalveluarkkitehtuuri (microservice) pyrkii vastaamaan näihin haasteisiin

Mikropalveluarkkitehtuuri

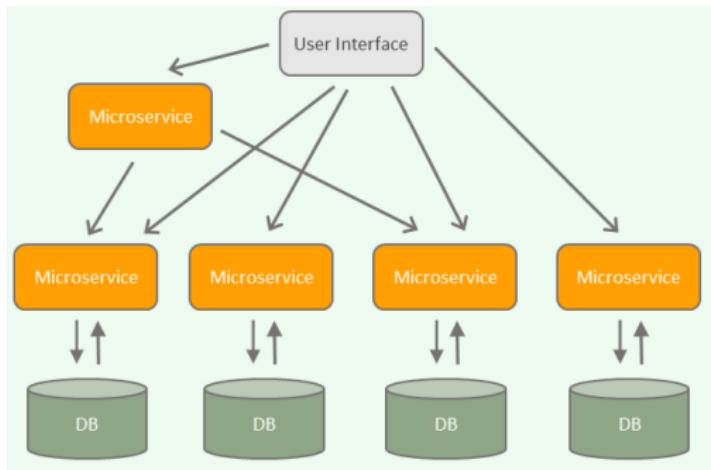
- ▶ Mikropalveluarkkitehtuuri (microservice) pyrkii vastaamaan näihin haasteisiin



- ▶ sovellus koostataan useista (jopa sadoista) pienistä verkossa toimivista autonomisista palveluista

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
- ▶ Mikropalvelut ovat pieniä ja huolehtivat vain ”yhdestä asiasta”

Mikropalveluarkkitehtuuri

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

Mikropalveluiden etuja

- ▶ Kun lisätään toiminnallisuutta: toteutetaan uusi palvelu tai laajennetaan ainoastaan *jotain* palvelua
 - ▶ Sovelluksen laajentaminen voi olla helpompaa kuin kerrosarkkitehtuurissa

Mikropalveluiden etuja

- ▶ Kun lisätään toiminnallisuutta: toteutetaan uusi palvelu tai laajennetaan ainoastaan *jotain* palvelua
 - ▶ Sovelluksen laajentaminen voi olla helpompaa kuin kerrosarkkitehtuurissa
- ▶ Skaalaaminen helpompaa kuin monoliittisten sovellusten
 - ▶ suorituskyvyn pullonkaulan aiheuttavia mikropalveluja voidaan suorittaa useita rinnakkain

Mikropalveluiden etuja

- ▶ Kun lisätään toiminnallisuutta: toteutetaan uusi palvelu tai laajennetaan ainoastaan *jotain* palvelua
 - ▶ Sovelluksen laajentaminen voi olla helpompaa kuin kerrosarkkitehtuurissa
- ▶ Skaalaaminen helpompaa kuin monoliittisten sovellusten
 - ▶ suorituskyvyn pullonkaulan aiheuttavia mikropalveluja voidaan suorittaa useita rinnakkain
- ▶ Sovellus voidaan helposti koodata monella ohjelmointikielellä ja sovelluskehysillä, toisin kuin monoliittisissä projekteissa

Mikropalveluiden etuja

- ▶ Kun lisätään toiminnallisuutta: toteutetaan uusi palvelu tai laajennetaan ainoastaan *jotain* palvelua
 - ▶ Sovelluksen laajentaminen voi olla helpompaa kuin kerrosarkkitehtuurissa
- ▶ Skaalaaminen helpompaa kuin monoliittisten sovellusten
 - ▶ suorituskyvyn pullonkaulan aiheuttavia mikropalveluja voidaan suorittaa useita rinnakkain
- ▶ Sovellus voidaan helposti koodata monella ohjelmointikielellä ja sovelluskehysillä, toisin kuin monoliittisissä projekteissa
- ▶ Työn jakaminen isolle kehittäjämääärälle helpompaa

Lähtökohta: The Bezos mandate

FROM: Jeff Bezos
TO: All Development
SUBJECT: Bezos Mandate



All teams will henceforth expose their data and functionality through service interfaces. Teams must communicate with each other through these interfaces.

There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.

It doesn't matter what technology they use.

All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.

Anyone who doesn't do this will be fired. Thank you; have a nice day!

Lähtökohta: The Bezos mandate

FROM: Jeff Bezos
TO: All Development
SUBJECT: Bezos Mandate



All teams will henceforth expose their data and functionality through service interfaces. Teams must communicate with each other through these interfaces.

There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.

It doesn't matter what technology they use.

All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.

Anyone who doesn't do this will be fired. Thank you; have a nice day!

FROM: Jeff Bezos
TO: All Development
SUBJECT: Bezos Mandate



All teams will henceforth expose their data and functionality through service interfaces. Teams must communicate with each other through these interfaces.

There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.

It doesn't matter what technology they use.

All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.

Mikropalveluiden haasteita

- ▶ Sovelluksen jakaminen järkeviin mikropalveluihin on vaikeaa

Mikropalveluiden haasteita

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

Mikropalveluiden 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 on haastavaa
 - ▶ vaatii pitkälle menevää automatisointia

Mikropalveluiden haasteita

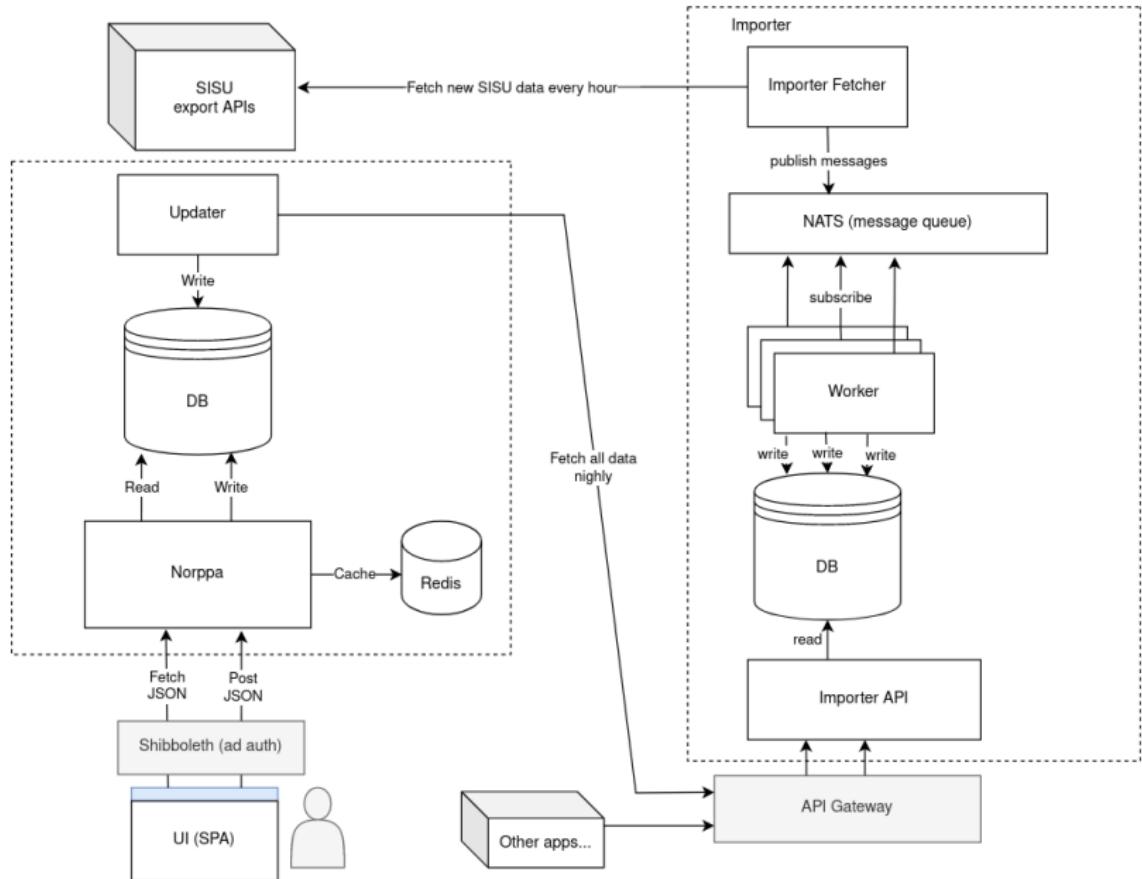
- ▶ Sovelluksen jakaminen järkeviin mikropalveluihin on vaikeaa
- ▶ Testaaminen ja debuggaus voi olla vaikeaa koska asioita tapahtuu niin monessa paikassa
- ▶ Kymmenistä tai jopa sadoista mikropalveluista koostuvan ohjelmiston operoiminen on haastavaa
 - ▶ vaatii pitkälle menevää automatisointia
- ▶ Mikropalveluiden menestyksekäs soveltaminen edellyttää vahvaa DevOps-kulttuuria

Mikropalveluiden haasteita

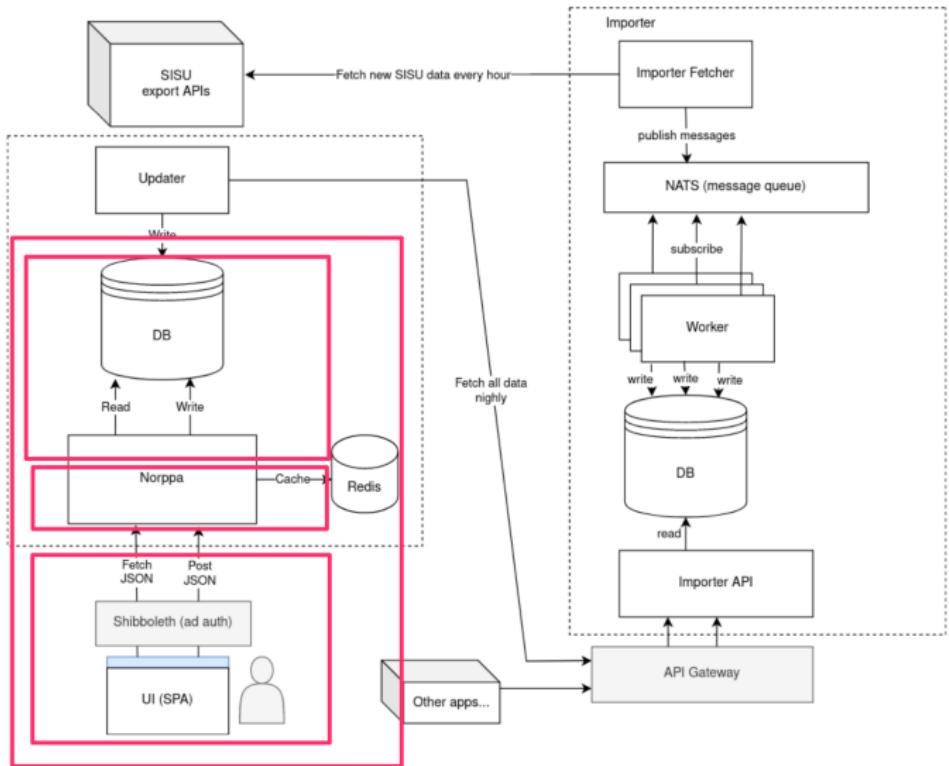
- ▶ Sovelluksen jakaminen järkeviin mikropalveluihin on vaikeaa
- ▶ Testaaminen ja debuggaus voi olla vaikeaa koska asioita tapahtuu niin monessa paikassa
- ▶ Kymmenistä tai jopa sadoista mikropalveluista koostuvan ohjelmiston operoiminen on haastavaa
 - ▶ vaatii pitkälle menevää automatisointia
- ▶ Mikropalveluiden menestyksekäs soveltaminen edellyttää vahvaa DevOps-kulttuuria
- ▶ Kehitetty massiivisiin järjestelmiin (mm Amazon, Netflix)
 - ▶ onko järkevä kaikkialla?

Kurssipalautejärjestelmä Norppa

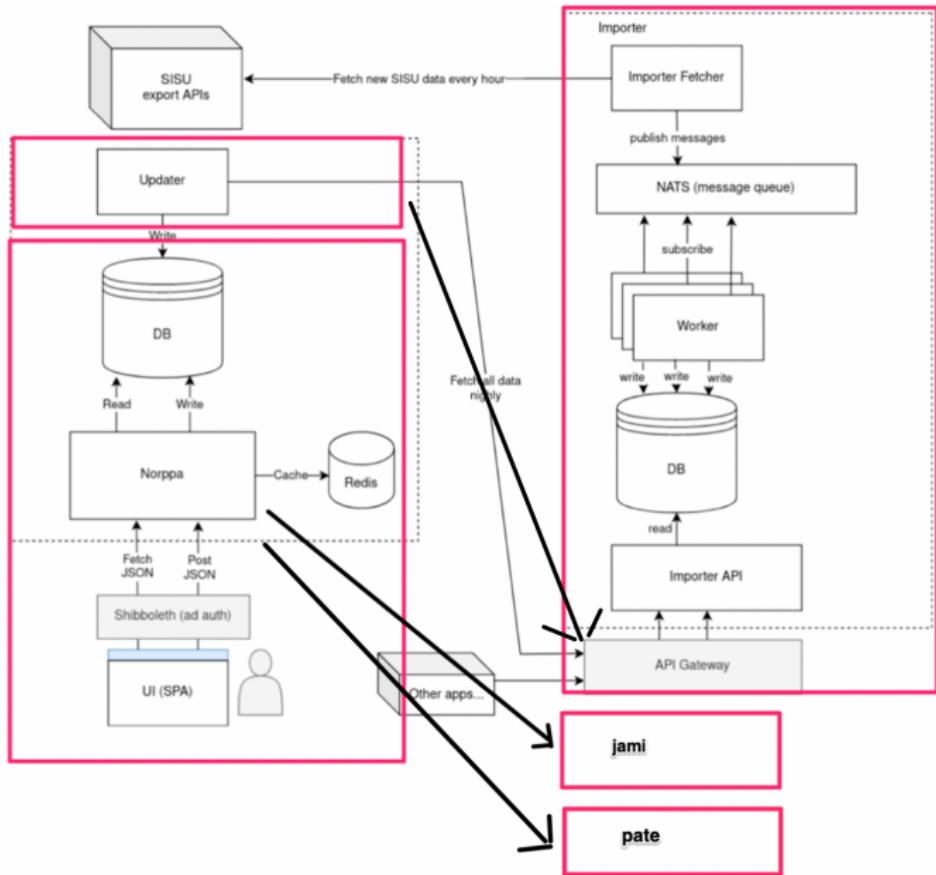
- ▶ Kerrosarkkitehtuuri
- ▶ Mikropalvelu
- ▶ Publish subscribe / Event driven



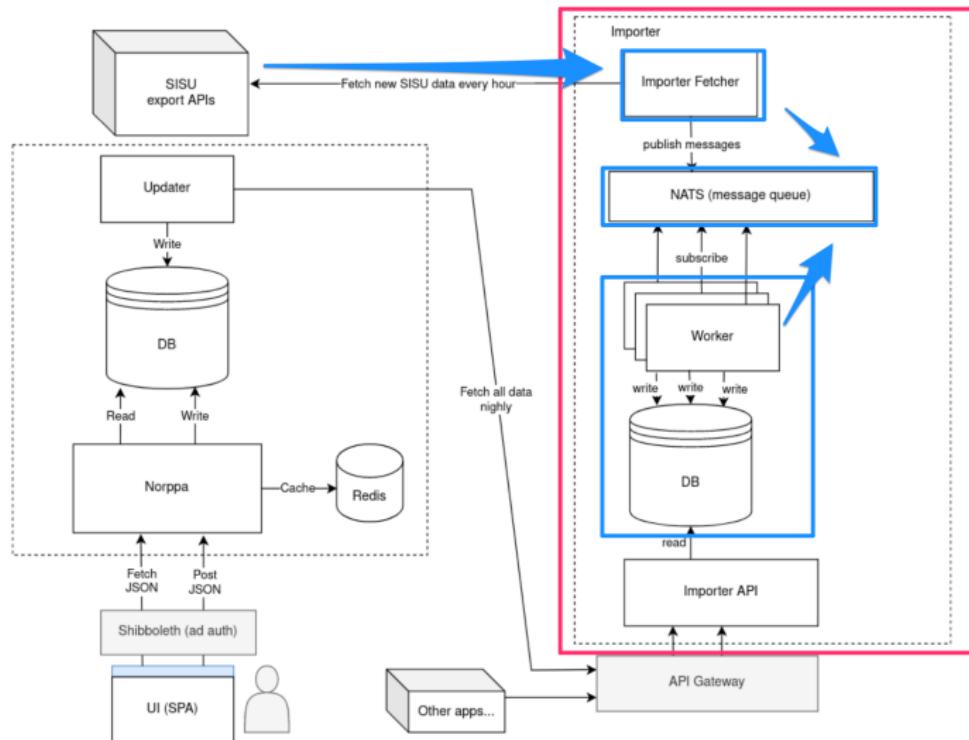
kerrosarkkitehtuuri



mikropalvelut



event driven / messaging



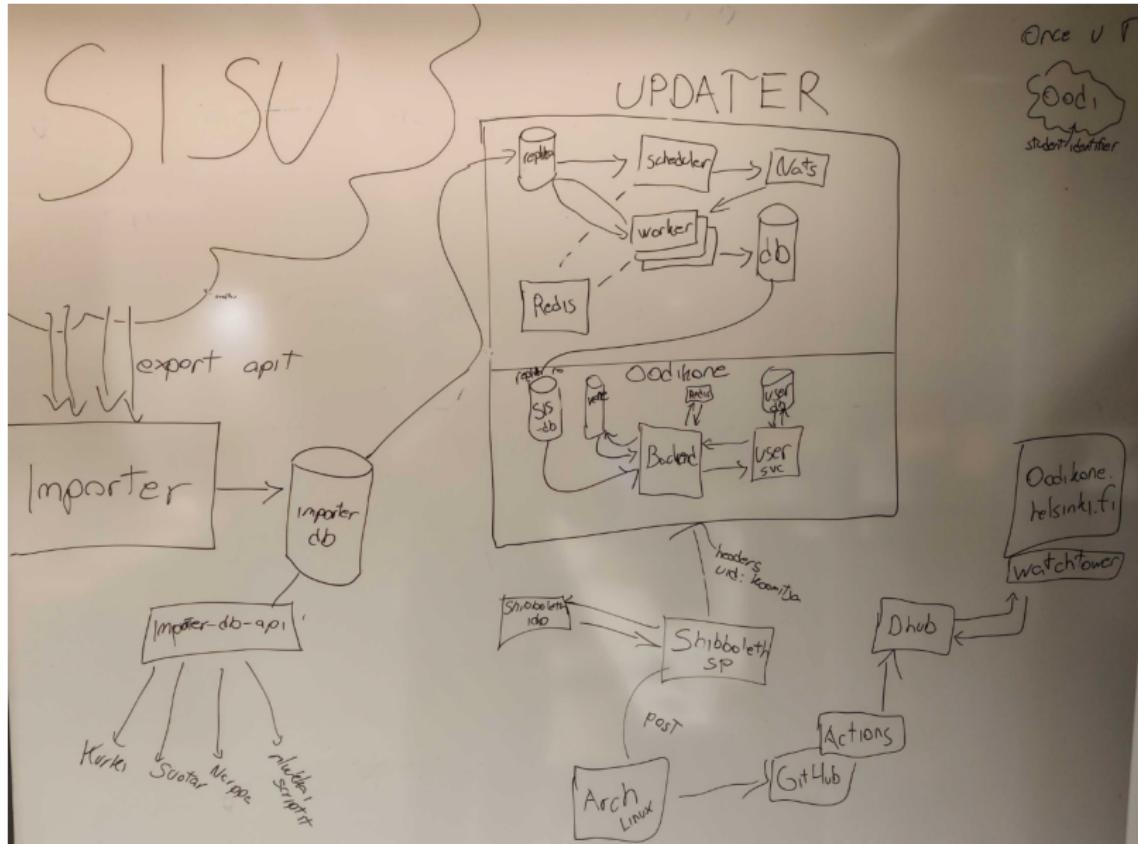
Arkkitehtuurin kuvaamisesta

- ▶ On tilanteita, missä sovelluksen arkkitehtuuri on dokumentoitava jollain tavalla

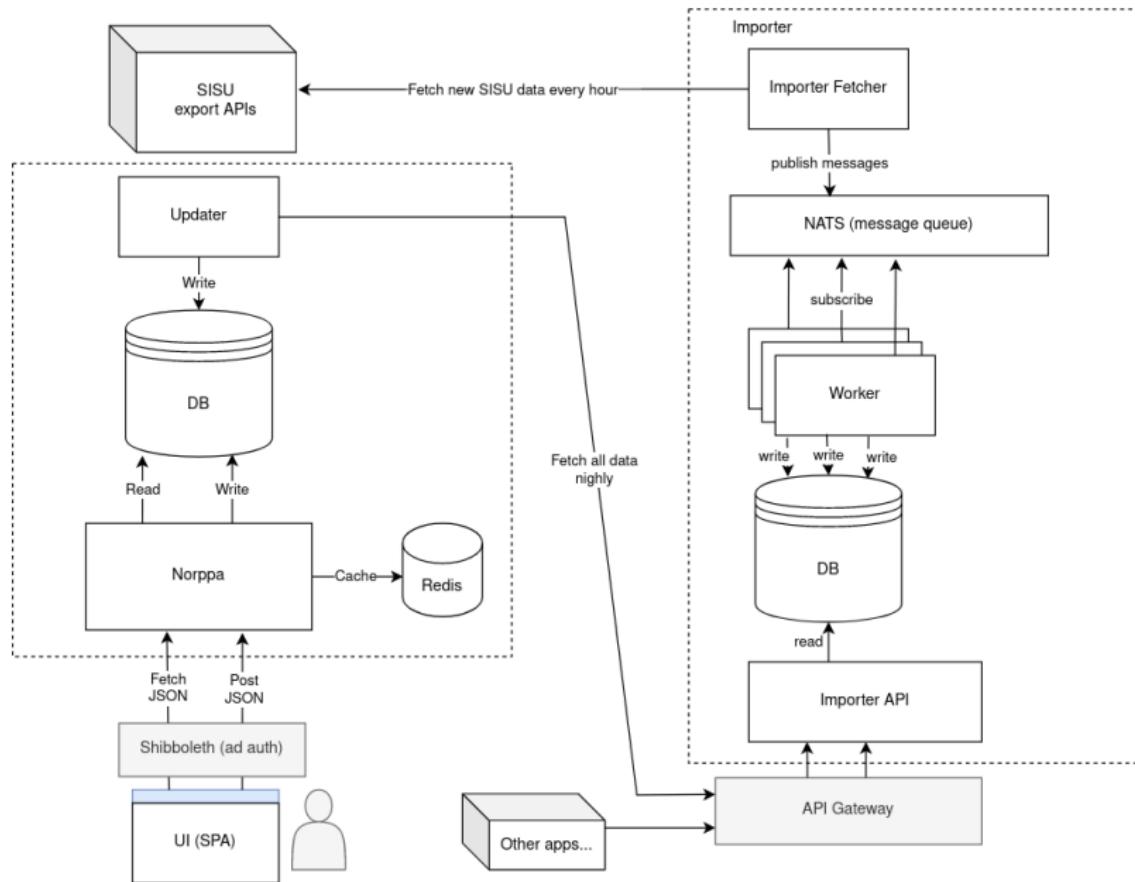
Arkkitehtuurin kuvaamisesta

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

Laatikko ja nuoli -kaavio



Hienompi laatikko ja nuoli -kaavio



Arkkitehtuurin kuvaamisesta

- ▶ Arkkitehtuurikuvaus voi olla tarpeen tehdä useasta eri tarpeita palvelevasta *näkökulmasta*
 - ▶ korkean tason kuvaus voi olla hyödyksi esim. vaatimusmäärittelyssä
 - ▶ tarkemmat kuvaukset toimivat ohjeena tarkemmassa suunnittelussa ja ylläpitovaiheen aikaisessa laajentamisessa

Arkkitehtuurin kuvaamisesta

- ▶ Arkkitehtuurikuvaus voi olla tarpeen tehdä useasta eri tarpeita palvelevasta *näkökulmasta*
 - ▶ korkean tason kuvaus 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

Arkkitehtuuri ketterissä menetelmissä

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

Arkkitehtuuri ketterissä menetelmissä

- ▶ Ketterien menetelmien kantava teema on toimivan, asiakkaalle arvoa tuottavan ohjelmiston nopea toimittaminen
- ▶ Periaatteita
 - ▶ *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software*
 - ▶ *Deliver working software frequently...*

Arkkitehtuuri ketterissä menetelmissä

- ▶ Ketterien menetelmien kantava teema on toimivan, asiakkaalle arvoa tuottavan ohjelmiston nopea toimittaminen
- ▶ Periaatteita
 - ▶ *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software*
 - ▶ *Deliver working software frequently...*
- ▶ Ketterät menetelmät suosivat yksinkertaisuutta
 - ▶ *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
- ▶ Periaatteita
 - ▶ *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software*
 - ▶ *Deliver working software frequently...*
- ▶ Ketterät menetelmät suosivat yksinkertaisuutta
 - ▶ *Simplicity, the art of maximizing the amount of work not done, is essential*
- ▶ Arkkitehtuuriin suunnittelu ja dokumentointi on perinteisesti pitkäkestoinen, ohjelmoinnin aloittamista edeltävä vaihe

Arkkitehtuuri ketterissä menetelmissä

- ▶ Ketterien menetelmien kantava teema on toimivan, asiakkaalle arvoa tuottavan ohjelmiston nopea toimittaminen
- ▶ Periaatteita
 - ▶ *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software*
 - ▶ *Deliver working software frequently...*
- ▶ Ketterät menetelmät suosivat yksinkertaisuutta
 - ▶ *Simplicity, the art of maximizing the amount of work not done, is essential*
- ▶ Arkkitehtuurin suunnittelu ja dokumentointi on perinteisesti pitkäkestoinen, ohjelmoinnin aloittamista edeltävä vaihe
- ▶ Ketterät menetelmät ja “arkkitehtuurivetoinen” ohjelmistotuotanto siis jossain määrin ristiriidassa

Arkkitehtuuri ketterissä menetelmissä

- ▶ Ketterien menetelmien yhteydessä puhutaan *inkrementaalisesta suunnittelusta ja arkkitehtuurista*

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 uutta toiminnallisuutta toteutetaan

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 uutta toiminnallisuutta toteutetaan
- ▶ Esim. kerrosarkkitehtuurin mukaista sovellusta ei rakenneta “kerros kerrallaan”
 - ▶ Jokaisessa iteraatiossa tehdään pieni pala jokaista kerrostaa, sen verran kuin iteraation tavoitteiden toteuttaminen edellyttää

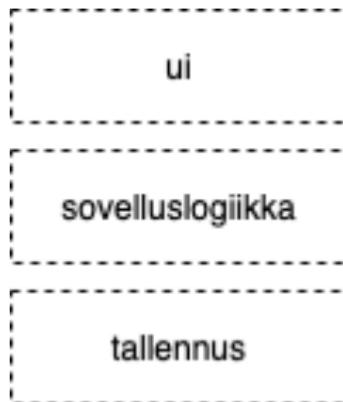
Ankrementaalinen arkkitehtuuri

- ▶ Alussa ns. *walking skeleton*
 - ▶ sisältää tynkäversiot ohjelmiston komponenttirakenteesta



Ankrementaalinen arkkitehtuuri

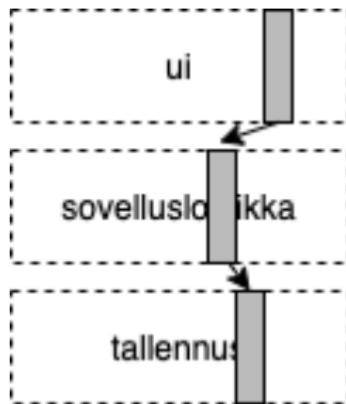
- ▶ Alussa ns. *walking skeleton*
 - ▶ sisältää tynkäversiot ohjelmiston komponenttirakenteesta



- ▶ Rakennetaan skeletonin varaan tuotetta story storyltä

Ominaisuuksiin perustuva integraatio

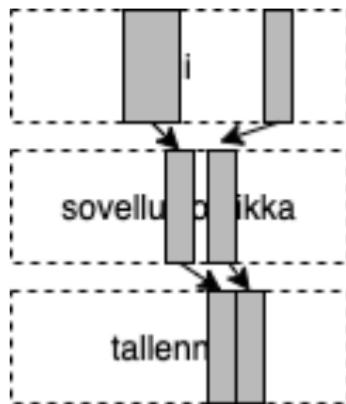
- ▶ Alussa ns. *walking skeleton*
 - ▶ sisältää tynkäversiot ohjelmiston komponenttirakenteesta



- ▶ Rakennetaan skeletonin varaan tuotetta story storyltä

Ominaisuuksiin perustuva integraatio

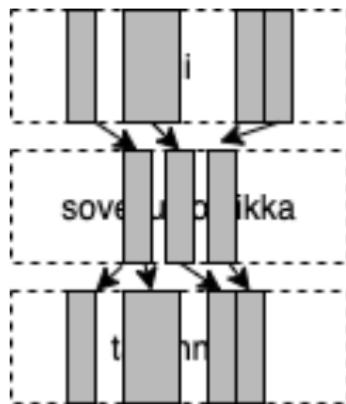
- ▶ Alussa ns. *walking skeleton*
 - ▶ sisältää tynkäversiot ohjelmiston komponenttirakenteesta



- ▶ Rakennetaan skeletonin varaan tuotetta story storyltä

Ominaisuuksiin perustuva integraatio

- ▶ Alussa ns. *walking skeleton*
 - ▶ sisältää tynkäversiot ohjelmiston komponenttirakenteesta



- ▶ Rakennetaan skeletonin varaan tuotetta story storyltä

Arkkitehtuuri ketterissä menetelmissä

- ▶ Perinteisesti arkkitehtuurin luonut *ohjelmistoarkkitehti*
 - ▶ ohjelmoijat velvoitettuja noudattamaan arkkitehtuuria

Arkkitehtuuri ketterissä menetelmissä

- ▶ Perinteisesti arkkitehtuurin luonut *ohjelmistoarkkitehti*
 - ▶ ohjelmoijat velvoitettuja noudattamaan arkkitehtuuria
- ▶ Ketterissä menetelmissä ei suosita erillistä arkkitehdin roolia
 - ▶ Scrumissa kaikista tiimiläisistä käytetään nimikettä developer

Arkkitehtuuri ketterissä menetelmissä

- ▶ Perinteisesti arkkitehtuurin luonut *ohjelmistoarkkitehti*
 - ▶ ohjelmoijat velvoitettuja noudattamaan arkkitehtuuria
- ▶ Ketterissä menetelmissä ei suosita erillistä arkkitehdin roolia
 - ▶ Scrumissa kaikista tiimiläisistä käytetään nimikettä developer
- ▶ Ketterä idealli: kehitystiimi luo arkkitehtuurin yhdessä
 - ▶ *The best architectures, requirements, and designs emerge from self-organizing teams*

Arkkitehtuuri ketterissä menetelmissä

- ▶ Perinteisesti arkkitehtuurin luonut *ohjelmistoarkkitehti*
 - ▶ ohjelmoijat velvoitettuja noudattamaan arkkitehtuuria
- ▶ Ketterissä menetelmissä ei suosita erillistä arkkitehdin roolia
 - ▶ Scrumissa kaikista tiimiläisistä käytetään nimikettä developer
- ▶ Ketterä idealli: kehitystiimi luo arkkitehtuurin yhdessä
 - ▶ *The best architectures, requirements, and designs emerge from self-organizing teams*
- ▶ **Arkkitehtuuri koodin tapaan tiimin yhteisomistama**

Arkkitehtuuri ketterissä menetelmissä

- ▶ Perinteisesti arkkitehtuurin luonut *ohjelmistoarkkitehti*
 - ▶ ohjelmoijat velvoitettuja noudattamaan arkkitehtuuria
- ▶ Ketterissä menetelmissä ei suosita erillistä arkkitehdin roolia
 - ▶ Scrumissa kaikista tiimiläisistä käytetään nimikettä developer
- ▶ Ketterä idealli: kehitystiimi luo arkkitehtuurin yhdessä
 - ▶ *The best architectures, requirements, and designs emerge from self-organizing teams*
- ▶ **Arkkitehtuuri koodin tapaan tiimin yhteisomistama**
- ▶ Etuja:
 - ▶ kehittäjät sitoutuvat paremmin arkkitehtuurin noudattamiseen kuin "norsunluutornissa" olevan arkkitehdin määrittelemään
 - ▶ dokumentaatio voi olla kevyt, tiimi tuntee arkkitehtuurin hengen ja pystyy sitä noudattamaan

Inkrementaalinen arkkitehtuuri: edut ja riskit

- ▶ Oletus: optimaalista arkkitehtuuria ei pystytä suunnittelemaan projektin aluss
 - ▶ Jo tehtyjä arkkitehtuuriratkaisuja muutetaan tarvittaessa

Inkrementaalinen arkkitehtuuri: edut ja riskit

- ▶ Oletus: optimaalista arkkitehtuuria ei pystytä suunnittelemaan projektin aluss
 - ▶ Jo tehtyjä arkkitehtuuriratkaisuja muutetaan tarvittaessa
- ▶ Kuten vaatimusmäärittelyssä, myös arkkitehtuurin suunnittelussa ketterä pyrkii välittämään *liian aikaisin tehtävää, ehkä turhaksi osoittautuvaa työtä*

Inkrementaalinen arkkitehtuuri: edut ja riskit

- ▶ Oletus: optimaalista arkkitehtuuria ei pystytä suunnittelemaan projektin aluss
 - ▶ Jo tehtyjä arkkitehtuuriratkaisuja muutetaan tarvittaessa
- ▶ Kuten vaatimusmäärittelyssä, myös arkkitehtuurin suunnittelussa ketterä pyrkii välittämään *liian aikaisin tehtävää, ehkä turhaksi osoittautuvaa työtä*
- ▶ Inkrementaalinen arkkitehtuuri edellyttää koodilta hyvää sisäistä laatua ja kehittäjiltä kurinalaisuutta
 - ▶ muuten seurauksena on kaaos

TAUKO 10 min

Olio/komponenttisuunnittelu

Olio/komponenttisuunnittelu

- ▶ Sovelluksen arkkitehtuuri antaa raamit, jotka ohjaavat sovelluksen tarkempaa suunnittelua ja toteuttamista

Olio/komponenttisuunnittelu

- ▶ Sovelluksen arkkitehtuuri antaa raamatit, 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 raamatit, jotka ohjaavat sovelluksen tarkempaa suunnittelua ja toteuttamista
- ▶ *Olio- tai komponenttisuunnittelu*
 - ▶ tarkentaa arkkitehtuuristen komponenttien väliset rajapinnat sekä hahmottelee ohjelman luokka- tai moduulirakenteen
- ▶ Vesiputousmallissa komponenttisuunnittelu tehty ennen ohjelmointia ja dokumentoituu tarkkaan esim. UML:lä

Olio/komponenttisuunnittelu

- ▶ Sovelluksen arkkitehtuuri antaa raamatit, jotka ohjaavat sovelluksen tarkempaa suunnittelua ja toteuttamista
- ▶ *Olio- tai komponenttisuunnittelu*
 - ▶ tarkentaa arkkitehtuuristen komponenttien väliset rajapinnat sekä hahmottelee ohjelman luokka- tai moduulirakenteen
- ▶ Vesiputousmallissa komponenttisuunnittelu tehty ennen ohjelmointia ja dokumentoitu tarkkaan esim. UML:lä
- ▶ Ketterässä tarkka suunnittelu tehdään vasta ohjelmoitaessa

Olio/komponenttisuunnittelu

- ▶ Sovelluksen arkkitehtuuri antaa raamatit, jotka ohjaavat sovelluksen tarkempaa suunnittelua ja toteuttamista
- ▶ *Olio- tai komponenttisuunnittelu*
 - ▶ tarkentaa arkkitehtuuristen komponenttien väliset rajapinnat sekä hahmottelee ohjelman luokka- tai moduulirakenteen
- ▶ Vesiputoosmallissa komponenttisuunnittelu tehty ennen ohjelmointia ja dokumentoitu tarkkaan esim. UML:lä
- ▶ Ketterässä tarkka suunnittelu tehdään vasta ohjelmoitaessa
- ▶ Suunnittelussa pyritään maksimoimaan *koodin sisäinen laatu*
 - ▶ helppo ylläpidettävyys ja laajennettavuus

Olio/komponenttisuunnittelu

- ▶ Sovelluksen arkkitehtuuri antaa raamatit, jotka ohjaavat sovelluksen tarkempaa suunnittelua ja toteuttamista
- ▶ *Olio- tai komponenttisuunnittelu*
 - ▶ tarkentaa arkkitehtuuristen komponenttien väliset rajapinnat sekä hahmottelee ohjelman luokka- tai moduulirakenteen
- ▶ Vesiputoosmallissa komponenttisuunnittelu tehty ennen ohjelmointia ja dokumentoitu tarkkaan esim. UML:lä
- ▶ 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 tuntemus auttaa
 - ▶ kehitetty monia suunnittelumenetelmiä, mikään niistä ei ole vakiintunut

Laadukas koodi

- ▶ Tavoitteena siis **sisäiseltä laadultaan** hyvä koodi

Laadukas koodi

- ▶ Tavoitteena siis **sisäiseltä laadultaan** hyvä koodi
- ▶ *Sisäinen laatu* (internal quality)
 - ▶ onko virheiden jäljitys ja korjaaminen helppoa
 - ▶ onko koodia helppo laajentaa ja jatkokehittää
 - ▶ pystytäänkö koodin toiminnallisuuden oikeellisuus varmistamaan muutoksia tehtäessä

Laadukas koodi

- ▶ Tavoitteena siis **sisäiseltä laadultaan** hyvä koodi
- ▶ *Sisäinen laatu* (internal quality)
 - ▶ onko virheiden jäljitys ja korjaaminen helppoa
 - ▶ onko koodia helppo laajentaa ja jatkokehittää
 - ▶ pystytäänkö koodin toiminnallisuuden oikeellisuus varmistamaan muutoksia tehtäessä
- ▶ Jos sisäinen laatu rapistuu
 - ▶ alkaa vaikuttamaan myös ulkoiseen eli käyttäjän kokemaan laatuun
 - ▶ kehitystiimin velositeetti alkaa tippua

Laadukkaan koodin tuntomerkejä

Laadukkaan koodin tuntomerkejä

- ▶ Laadukkaalla koodilla joukko yhteneviä ominaisuuksia, tai *laatuattribuutteja*, esim. seuraavat:
 - ▶ kapselointi
 - ▶ korkea koheesion aste
 - ▶ riippuvuuksien vähäisyys
 - ▶ toisteettomuus
 - ▶ testattavuus
 - ▶ selkeys

Laadukkaan koodin tuntomerkkejä

- ▶ Laadukkaalla 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, repository*
 - ▶ lisää kurssimateriaalissa ja laskareissa

Koodin laatuattribuutti: kapselointi

Koodin laatuattribuutti: kapselointi

- ▶ *Kapselointi ohjelmoinnin peruskursseilla:*
 - ▶ *oliomuuttujat tulee määritellä piilotetuksi ja niille tulee tehdä tarvittaessa aksessorimetodit*

Koodin laatuattribuutti: kapselointi

- ▶ *Kapselointi ohjelmoinnin peruskursseilla:*
 - ▶ *oliomuuttujat tulee määritellä piilotetuksi ja niille tulee tehdä tarvittaessa aksessorimetodit*
- ▶ *Olion sisäisen tilan lisäksi kapseloinnin kohde voi olla mm. käytettävän olion tyyppi, käytetty algoritmi, olioiden luomisen tapa, käytettävän komponentin rakenne*

Koodin laatuatribuutti: kapselointi

- ▶ *Kapselointi ohjelmoinnin peruskursseilla:*
 - ▶ *oliomuuttujat tulee määritellä piilotetuksi ja niille tulee tehdä tarvittaessa aksessorimetodit*
- ▶ Olion sisäisen tilan lisäksi kapseloinnin kohde voi olla mm.
käytettävä olion tyyppi, käytetty algoritmi, olioiden luomisen tapa, käytettävä komponentin rakenne
- ▶ Näkyy myös arkkitehtuurin tasolla
 - ▶ kerrosarkkitehtuuri: ylempi kerros käyttää ainoastaan alemman kerroksen ulospäin tarjoamaa rajapintaa, muu kapseloitu
 - ▶ mikropalvelut: yksittäinen palvelu kapseloi sisäisen logiikan, tiedon säilytystavan ja tarjoaa ainoastaan verkon välityksellä käytettävän rajapinnan

Koodin laatuattribuutti: koheesio

Koodin laatuatribuutti: koheesio

- ▶ *Koheesio:*

- ▶ kuinka pitkälle metodin, luokan tai komponentin koodi keskittyy tietyn yksittäisen toiminnallisuuden toteuttamiseen
- ▶ hyvänä pidetään mahdollisimman korkeaa koheesion astetta

Koodin laatuatribuutti: koheesio

- ▶ *Koheesio:*
 - ▶ kuinka pitkälle metodin, luokan tai komponentin koodi keskittyy tietyn yksittäisen toiminnallisuuden toteuttamiseen
 - ▶ hyvänä pidetään mahdollisimman korkeaa koheesion astetta
- ▶ Luokkataslon koheesio
 - ▶ luokan *vastuulla* vain yksi asia, tunnetaan myös nimellä *single responsibility principle*

Koodin laatuatribuutti: koheesio

- ▶ *Koheesio:*
 - ▶ kuinka pitkälle metodin, luokan tai komponentin koodi keskittyy tietyn yksittäisen toiminnallisuuden toteuttamiseen
 - ▶ hyvänä pidetään mahdollisimman korkeaa koheesion astetta
- ▶ Luokkataslon koheesio
 - ▶ luokan *vastuulla* vain yksi asia, tunnetaan myös nimellä *single responsibility principle*
- ▶ Arkkitehtuurin tasolla
 - ▶ kerrosarkkitehtuurin kerrokset samalla abstraktiotasolla, esim. käyttöliittymä tai tietokanttarajapinta
 - ▶ mikropalvelu toteuttaa tiettyyn liiketoiminnan tason toiminnallisuuden, esim. suosittelualgoritmin tai käyttäjien hallinnan

Koheesio Flask-sovelluksessa

```
@app.route("/create_todo", methods=["POST"])
def todo_creation():
    content = request.form.get("content")

    try:
        if len(content) < 5:
            raise UserInputError("Todo content length must be greater than 4")

        if len(content) > 100:
            raise UserInputError("Todo content length must be smaller than 100")

        sql = text("INSERT INTO todos (content) VALUES (:content)")
        db.session.execute(sql, { "content": content })
        db.session.commit()

        return redirect("/")
    except Exception as error:
        flash(str(error))
        return redirect("/new_todo")
```

Koheesio Flask-sovelluksessa

```
@app.route("/create_todo", methods=["POST"])
def todo_creation():
    content = request.form.get("content")

    try:
        validate_todo(content)
        create_todo(content)
        return redirect("/")
    except Exception as error:
        flash(str(error))
        return redirect("/new_todo")

# util.py

def validate_todo(content):
    if len(content) < 5:
        raise UserInputError("Todo content length must be greater than 4")

    if len(content) > 100:
        raise UserInputError("Todo content length must be smaller than 100")

# todo_repository.py

def create_todo(content):
    sql = text("INSERT INTO todos (content) VALUES (:content)")
    db.session.execute(sql, { "content": content })
    db.session.commit()
```

Koheesio Flask-sovelluksessa

```
@app.route("/create_todo", methods=["POST"])
def todo_creation():
    content = request.form.get("content")

    try:
        validate_todo(content)
        create_todo(content)
        return redirect("/")
    except Exception as error:
        flash(str(error))
        return redirect("/new_todo")

# util.py

def validate_todo(content):
    if len(content) < 5:
        raise UserInputError("Todo content length must be greater than 4")

    if len(content) > 100:
        raise UserInputError("Todo content length must be smaller than 100")

# todo_repository.py

def create_todo(content):
    sql = text("INSERT INTO todos (content) VALUES (:content)")
    db.session.execute(sql, { "content": content })
    db.session.commit()
```

- ▶ delegoidaan osa vastuista eri komponenteille

Koodin laatuattribuutti: riippuvuuksien vähäisyys

Koodin laatuattribuutti: riippuvuuksien vähäisyys

- ▶ Pyrkimys korkeaan koheesioon johtaa ohjelmiin, joissa suuri määrä olioita/komponentteja

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 laatuatribuutti: 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
- ▶ Hyödynnetään *dependence injection* -suunnittelumallia

Koodin laatuatribuutti: riippuvuuksien vähäisyys

```
def main():
    stats = StatisticsService()

class StatisticsService:
    def __init__(self):
        reader = PlayerReader()
```

VS

```
def main():
    reader = PlayerReader("https://studies.cs.helsinki.fi/nhlstats/2021-22/players.txt")
    stats = StatisticsService(reader)

class StatisticsService:
    def __init__(self, reader):
        self._players = reader.get_players()
```

Koodin laatuattribuutti: toisteettomuus

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
 - ▶ 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
 - ▶ tietokantaskeemaa, testejä, build-skriptejä
- ▶ Suoraviivainen copypaste helppo eliminoida metodien avulla
 - ▶ kaikki toisteisuus ei ole yhtä ilmeistä, monissa suunnittelumalleissa kyse hienovaraisempien toisteisuuden muotojen eliminoinnista

Koodin laatuatribuutti: 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
 - ▶ tietokantaskeemaa, testejä, build-skriptejä
- ▶ Suoraviivainen copypaste helppo eliminoida 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

Koodin laatuattribuutti: testattavuus

- ▶ Laadukas koodi on helppo testata kattavasti yksikkö- ja integraatiotestein
 - ▶ seuraa yleensä siitä, että koodi koostuu löyhästi kytketyistä, selkeän vastuun omaavista komponenteista

Koodin laatuattribuutti: testattavuus

- ▶ Laadukas koodi on helppo testata kattavasti yksikkö- ja integraatiotestein
 - ▶ seuraa yleensä siitä, että koodi koostuu löyhästi kytketyistä, selkeän vastuun omaavista komponenteista
- ▶ Hyvää testattavuutta auttaa turhien riippuvuuksien eliminointi dependency injection -periaatteen avulla

Koodin laatuattribuutti: testattavuus

- ▶ Laadukas koodi on helppo testata kattavasti yksikkö- ja integraatiotestein
 - ▶ seuraa yleensä siitä, että koodi koostuu löyhästi kytketyistä, selkeän vastuun omaavista komponenteista
- ▶ Hyvää testattavuutta auttaa turhien riippuvuuksien eliminointi dependency injection -periaatteen avulla
- ▶ Test driven development tuottaa varmuudella hyvin testattavissa olevaa koodia

Koodin laatuatribuutti: selkeys ja luettavuus

- ▶ Perinteisesti ajateltu että koodi kryptistä ja vaikeasti luettavaa
 - ▶ yleistä C-kielessä, pyritty esim. optimoimaan tehokkuutta ja muistinkäyttöä

Koodin laatuatribuutti: selkeys ja luettavuus

- ▶ Perinteisesti ajateltu että koodi kryptistä ja vaikeasti luettavaa
 - ▶ yleistä C-kielessä, pyritty esim. optimoimaan tehokkuutta ja muistinkäyttöä
- ▶ Nykytrendi: tehdän koodia, joka nimeämisen sekä rakenteen kautta ilmaisee hyvin sen, mitä koodi tekee

Koodin laatuatribuutti: selkeys ja luettavuus

- ▶ Perinteisesti ajateltu että koodi kryptistä ja vaikeasti luettavaa
 - ▶ yleistä C-kielessä, pyritty esim. optimoimaan tehokkuutta ja muistinkäyttöä
- ▶ Nykytrendi: tehdän koodia, joka nimeämisen sekä rakenteen kautta ilmaisee hyvin sen, mitä koodi tekee
- ▶ Miksi selkeää koodi on tärkeää?
 - ▶ joidenkin arvioiden mukaan jopa 90% "ohjelointiin" kuluvasta ajasta menee olemassa olevan koodin lukemiseen
 - ▶ oma aikoinaan niin selkeää koodi, ei enää olekaan yhtä selkeää parin kuukauden kuluttua

Code smell

- ▶ Koodi ei ole aina hyvää...

Code smell

- ▶ Koodi ei ole aina hyvää...
- ▶ 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

- ▶ Koodi ei ole aina hyvää...
- ▶ 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:
 - ▶ toisteenen koodi
 - ▶ liian pitkät metodit
 - ▶ luokat joissa on liikaa oliomuuttujia
 - ▶ luokat joissa on liikaa koodia
 - ▶ metodien liian pitkät parametristat
 - ▶ epäselkeät muuttujien, metodien tai luokkien nimet
 - ▶ kommentit

Koodihajuja

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

Refaktoriointi

Refaktoriointi

- ▶ Lääke koodin sisäisen laadun ongelmiin on *refaktoriointi*
 - ▶ koodin toiminnalisuuden ennallaan pitävä sisäisen rakenteen muutos

Refaktoriointi

- ▶ Lääke koodin sisäisen laadun ongelmiin on *refaktoriointi*
 - ▶ koodin toiminnalisuuden ennallaan pitävä sisäisen rakenteen muutos
- ▶ Koodin rakennetta parantavia refaktorointeja on lukuisia, mm.
 - ▶ *rename variable/method/class*
 - ▶ *extract method*
 - ▶ *move field/method*
 - ▶ *extract superclass*

Refaktoriointi

- ▶ Lääke koodin sisäisen laadun ongelmiin on *refaktoriointi*
 - ▶ koodin toiminnalisuuden ennallaan pitävä sisäisen rakenteen muutos
- ▶ Koodin rakennetta parantavia refaktorointeja on lukuisia, mm.
 - ▶ *rename variable/method/class*
 - ▶ *extract method*
 - ▶ *move field/method*
 - ▶ *extract superclass*
- ▶ Osa pystytään tekemään sovelluskehitysympäristön avustamana
 - ▶ helpompaa staattisesti tyypitetyillä kielillä kuten Java

Miten refaktoriointi kannattaa tehdä

- ▶ Refaktorioidun edellytys on kattavien testien olemassaolo

Miten refaktoriointi kannattaa tehdä

- ▶ Refaktorioiden edellytys on kattavien testien olemassaolo
- ▶ Kannattaa ehdottomasti edetä pienin askelin
 - ▶ yksi hallittu muutos kerrallaan
 - ▶ testit suoritettava mahdollisimman usein

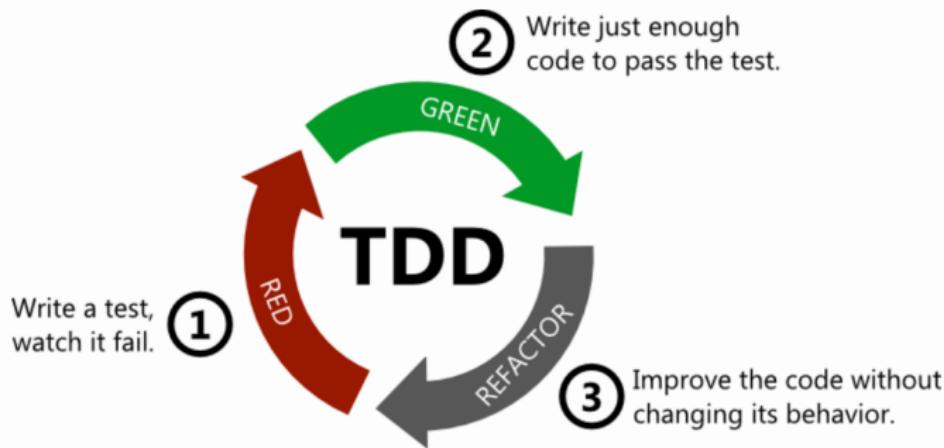
Miten refaktoriointi kannattaa tehdä

- ▶ Refaktorioiden edellytys on kattavien testien olemassaolo
- ▶ Kannattaa ehdottomasti edetä pienin askelin
 - ▶ yksi hallittu muutos kerrallaan
 - ▶ testit suoritettava mahdollisimman usein
- ▶ Refaktoriointia kannattaa suorittaa lähes jatkuvasti
 - ▶ pitää koodin rakenteen selkeänä ja helpottaa sekä nopeuttaa koodin laajentamista

Miten refaktoriointi kannattaa tehdä

- ▶ Refaktorioidin edellytys on kattavien testien olemassaolo
- ▶ Kannattaa ehdottomasti edetä pienin askelin
 - ▶ yksi hallittu muutos kerrallaan
 - ▶ testit suoritettava mahdollisimman usein
- ▶ Refaktoriointia kannattaa suorittaa lähes jatkuvasti
 - ▶ pitää koodin rakenteen selkeänä ja helpottaa sekä nopeuttaa koodin laajentamista
- ▶ Osa refaktorioidista on helppoa ja suoraviivaista, aina ei näin ole
 - ▶ joskus tarve tehdä isoja, jopa viikkojen kestoisia refaktointeja joissa ohjelman rakenne muuttuu paljon

Refaktoriointi tärkeä osa Test driven development -menetelmää



1. Kirjoitetaan sen verran testiä että testi ei mene läpi
2. Kirjoitetaan koodia sen verran, että testi menee läpi
3. **Jos huomataan koodin rakenteen menneen huonoksi refaktoroidaan koodin rakenne paremmaksi**
4. Jatketaan askeleesta 1

15.1.2025-

Avoim yliopisto: Test-Driven Development 4 + 1 cr

← → ⌂ tdd.mooc.fi ☆ ⓘ

Test-Driven Development

Course overview

Practicalities (2024 Spring)

Exercises and schedule

Chapter 1: What is TDD

Chapter 2: Refactoring and design

Chapter 3: The Untestables

Chapter 4: Legacy code

Chapter 5: Advanced techniques

Chapter 6: To infinity and beyond

About the material

The banner features a large circular graphic divided into three segments: red at the top left, green at the top right, and blue at the bottom. Red arrows point from the top towards the center and from the center towards the bottom. Green arrows point from the top towards the center and from the center towards the bottom. Blue arrows point from the bottom towards the center and from the center towards the top. Overlaid on the graphic is the text "Test-Driven Development" in a large, bold, red font, and below it, "a plunge into the TDD programming technique" in a smaller, red font. At the bottom of the graphic, the word "Refactor" is written in a large, blue font.

Test-Driven Development

a plunge into the TDD programming technique

Refactor

- ▶ Esko Luontola Nitor (Suomen johtava TDD-asiantuntija)

Tekninen velka

Tekninen velka

- ▶ Koodi ei ole aina laadultaan optimaalista

Tekninen velka

- ▶ Koodi ei ole aina laadultaan optimaalista
- ▶ Huonoa suunnittelua tai/ja ohjelointia kuvaavat käsitykset *tekninen velka* (technical debt)

Tekninen velka

- ▶ Koodi ei ole aina laadultaan optimaalista
- ▶ Huonoa suunnittelua tai/ja ohjelointia kuvaa käsite *tekninen velka* (technical debt)
- ▶ Piittaamattomalla ja laiskalla ohjelmoinnilla/suunnittelulla saadaan ehkä nopeasti aikaan jotain
 - ▶ hätäinen ratkaisu tullaan maksamaan korkoineen takaisin *jos* ohjelmaa on tarkoitus laajentaa

Tekninen velka

- ▶ Koodi ei ole aina laadultaan optimaalista
- ▶ Huonoa suunnittelua tai/ja ohjelointia kuvaa käsite *tekninen velka* (technical debt)
- ▶ Piittaamattomalla ja laiskalla ohjelmoinnilla/suunnittelulla saadaan ehkä nopeasti aikaan jotain
 - ▶ 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
- ▶ Huonoa suunnittelua tai/ja ohjelointia kuvaa käsite *tekninen velka* (technical debt)
- ▶ Piittaamattomalla ja laiskalla ohjelmoinnilla/suunnittelulla saadaan ehkä nopeasti aikaan jotain
 - ▶ 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 voi olla järkevää tai jopa välttämätöntä
 - ▶ voidaan saada tuote nopeammin markkinoille tekemällä tietoisesti huonoa designia, joka korjataan myöhemmin

- ▶ Kaikki tekninen velka ei samanlaista, taustalla voi olla
 - ▶ holtittomuus, osaamattomuus, tietämättömyys tai tarkoituksella tehty päätös

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

- ▶ Kaikki tekninen velka ei samanlaista, taustalla voi olla
 - ▶ holtittomuus, osaamattomuus, tietämättömyys tai tarkoituksella tehty päätös
- ▶ Martin Fowler jaottelee teknisen velan neljään eri luokkaan:
 - ▶ Reckless and deliberate: “we do not have time for design”
 - ▶ Reckless and inadvertent: “what is layering”?
 - ▶ Prudent and inadvertent: “now we know how we should have done it”
 - ▶ **Prudent and deliberate: “we must ship now and will deal with consequences”**
- ▶ Joskus tekninen velka pakottaa koodaamaan koko järjestelmän uudelleen