Ohjelmistotuotanto

Matti Luukkainen ja ohjaajat Kalle Ilves, Antti Kantola, Riikka Korolainen, Touko Puro

syksy 2021

Luento 8

23.11.2021

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
- Vaatimusmäärittelyä ja testausta sekä laadunhallintaa käsitelty

Ohjelmiston elinkaaren vaiheet

- Riippumatta tyylistä ja tavasta jolla ohjelmisto tehdään, ohjelmistojen tekemiseen kuuluu
  - vaatimusten analysointi ja määrittely
  - suunnittelu
  - toteutus
  - testaus/laadunhallinta
  - ohjelmiston ylläpito
- Vaatimusmäärittelyä ja testausta sekä laadunhallintaa käsitelty
- Siirrymme käsittelemään ohjelmiston suunnittelua ja toteuttamista
  - osa suunnittelusta tapahtuu vasta toteutusvaiheessa, joten suunnittelun ja toteuttamisen käsittelyä ei voi eriyttää

Ohjelmiston elinkaaren vaiheet

- Riippumatta tyylistä ja tavasta jolla ohjelmisto tehdään, ohjelmistojen tekemiseen kuuluu
  - vaatimusten analysointi ja määrittely
  - suunnittelu
  - toteutus
  - testaus/laadunhallinta
  - ohjelmiston ylläpito
- Vaatimusmäärittelyä ja testausta sekä laadunhallintaa 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

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

- IEEE: 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

- **Ei-toiminnallisilla vaatimuksilla** suuri vaikutus arkkitehtuuriin
  - käytettävyys, saavutettavuus
  - suorituskyky, skaalautuvuus
  - vikasietoisuus, tiedon ajantasaisuus
  - tietoturva
  - ylläpidettävyys, laajennettavuus
  - hinta, time-to-market, ...

Ohjelmiston arkkitehtuuri

- **Ei-toiminnallisilla vaatimuksilla** suuri vaikutus arkkitehtuuriin
  - käytettävyys, saavutettavuus
  - suorituskyky, skaalautuvuus
  - vikasietoisuus, tiedon ajantasaisuus
  - tietoturva
  - ylläpidettävyys, laajennettavuus
  - hinta, time-to-market, ...
- Myös **toimintaympäristö** vaikuttavaa arkkitehtuuriin
  - integraatiot muihin järjestelmiin
  - käytettävät sovelluskehykset ja tietokannat
  - lainsäädäntö

Ohjelmiston arkkitehtuuri

- **Ei-toiminnallisilla vaatimuksilla** suuri vaikutus arkkitehtuuriin
  - käytettävyys, saavutettavuus
  - 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*
  - tradeoff

Arkkitehtuurityyli

- Ohjelmiston arkkitehtuuri perustuu yleensä yhteen tai useampaan **arkkitehtuurityyliin** (architectural style)
  - hyväksi havaittua tapaa strukturoida tietyntyyppisiä sovelluksia

Arkkitehtuurityyli

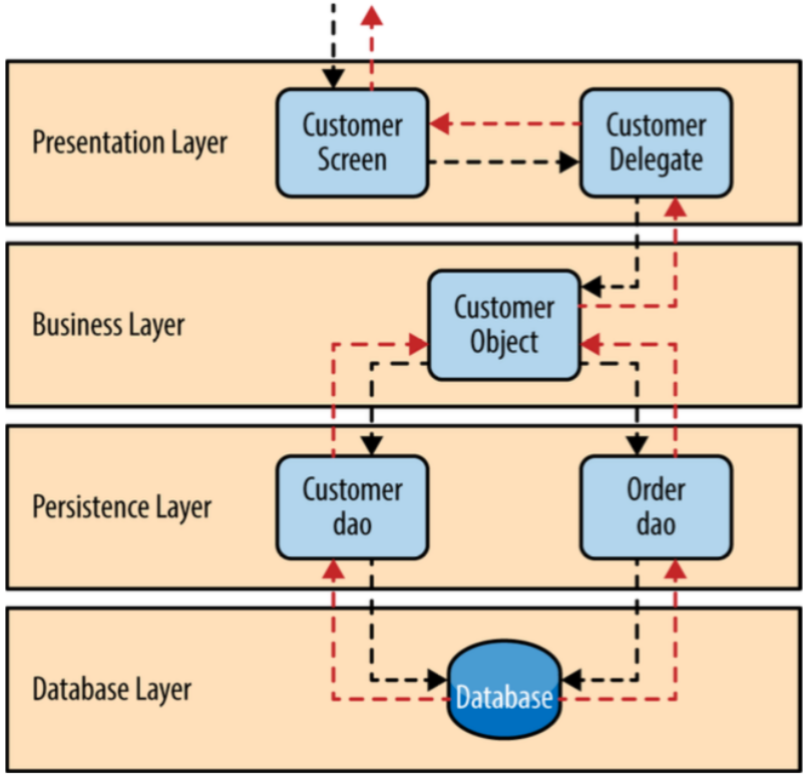
- Ohjelmiston arkkitehtuuri perustuu yleensä yhteen tai useampaan **arkkitehtuurityyliin** (architectural style)
  - hyväksi havaittua tapaa strukturoida tietyntyyppisiä sovelluksia
- Tyylejä suuri määrä
  - Kerrosarkkitehtuuri
  - Mikropalveluarkkitehtuuri
  - MVC
  - Pipes-and-filters
  - Repository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

Arkkitehtuurityyli

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

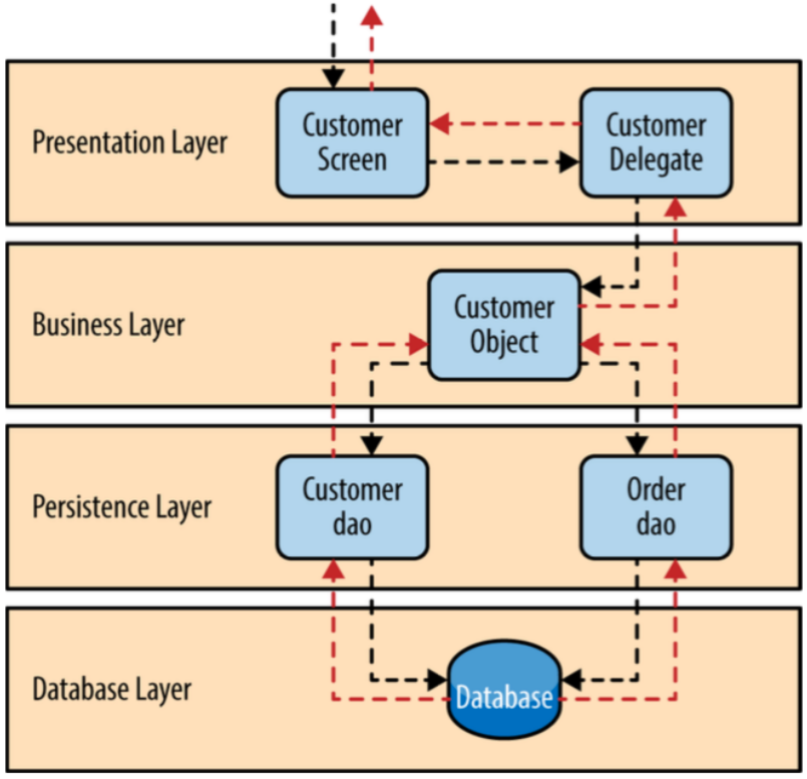
Kerrosarkkitehtuuri

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



Kerrosarkkitehtuuri

- *Kerros* on kokoelma toisiinsa liittyviä olioita, 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

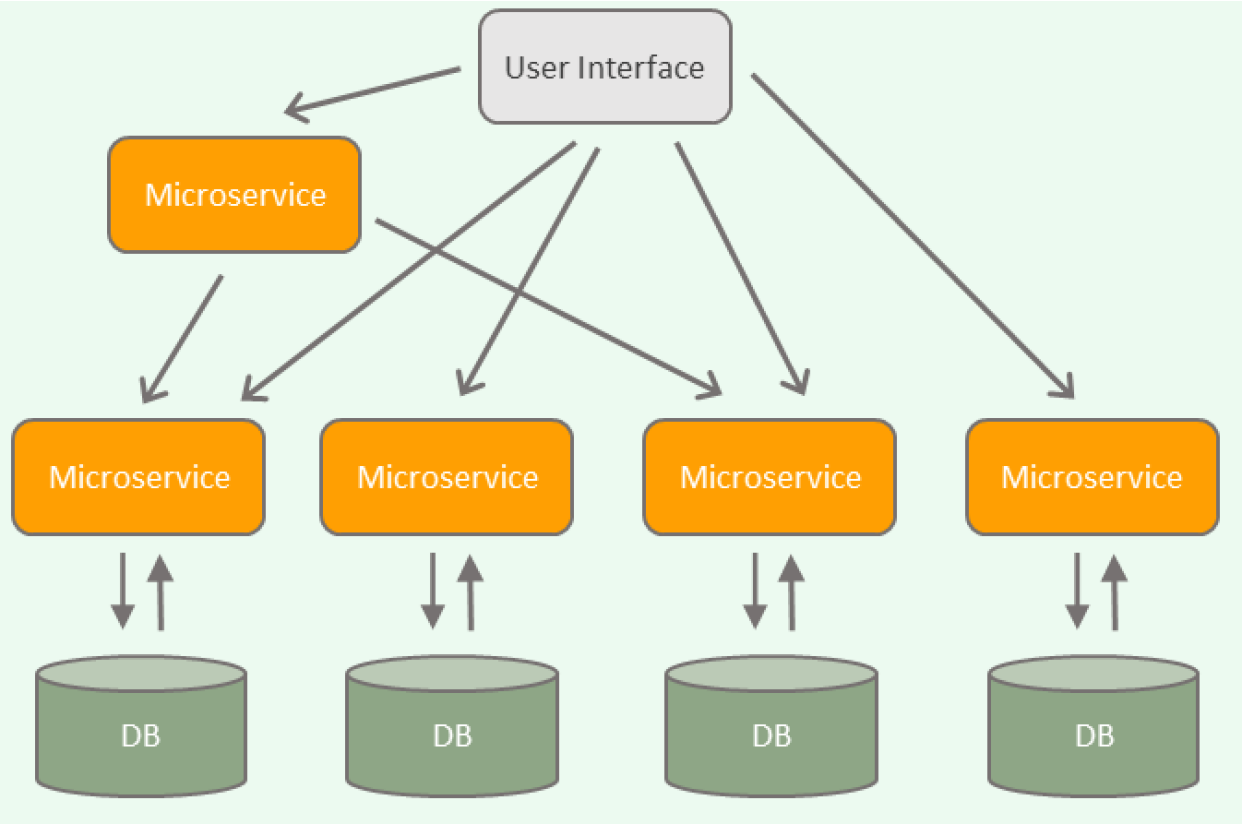
- Kerrokset omalla abstraktiotasollaan
  - Ylimmät kerrokset ovat lähellä käyttäjää: UI ja sovelluslogiikka
  - Alimmat kerrokset taas keskittyvät koneläheisiin asioihin: esim. tiedon tallennus
- Kerroksittaisuus helpottaa ylläpitoa
  - yhden kerroksen muutokset vaikuttavat korkeintaan yläpuolella olevaan kerrokseen
- Sovelluslogiikan riippumattomuus käyttöliittymästä helpottaa ohjelman siirtämistä uusille alustoille
  - esim. web-sovelluksesta voidaan tehdä mobiiliversio

- 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
  - esim. web-sovelluksesta voidaan tehdä mobiiliversio
- 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
  - esim. web-sovelluksesta voidaan tehdä mobiiliversio
- Alimpien kerroksien palveluja, voidaan osin uusiokäyttää myös muissa sovelluksissa
- Kerroksittaisuus 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”

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 hakutoiminnot
  - tuotteiden suosittelu
  - ostoskorin toiminnallisuus
  - ostosten maksusta huolehtiva toiminnallisuus

Mikropalveluiden etuja

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

Mikropalveluiden etuja

- Kun järjestelmään lisätään toiminnallisuutta, se yleensä tarkoittaa uusien palveluiden toteuttamista tai ainoastaan joidenkin palveluiden laajentamista
  - Sovelluksen laajentaminen voi olla helpompaa kuin kerrosarkkitehtuurissa
- Skaalaaminen helpompaa kuin monoliittisten sovellusten
  - suorituskyvyn pullonkaulan aiheuttavia mikropalveluja voidaan suorittaa useita rinnakkain

Mikropalveluiden etuja

- Kun järjestelmään lisätään toiminnallisuutta, se yleensä tarkoittaa uusien palveluiden toteuttamista tai ainoastaan joidenkin palveluiden laajentamista
  - 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 sovelluskehyksillä, toisin kuin monoliittisissa projekteissa

Haasteita

- Sovelluksen jakaminen järkeviin mikropalveluihin on vaikeaa

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

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
  - Useimmiten käytetään epäformaaleja laatikko/nuoli-kaavioita
  - UML:n luokka- ja pakkauskaaviot sekä komponentti- ja sijoittelukaaviot joskus käyttökelpoisia

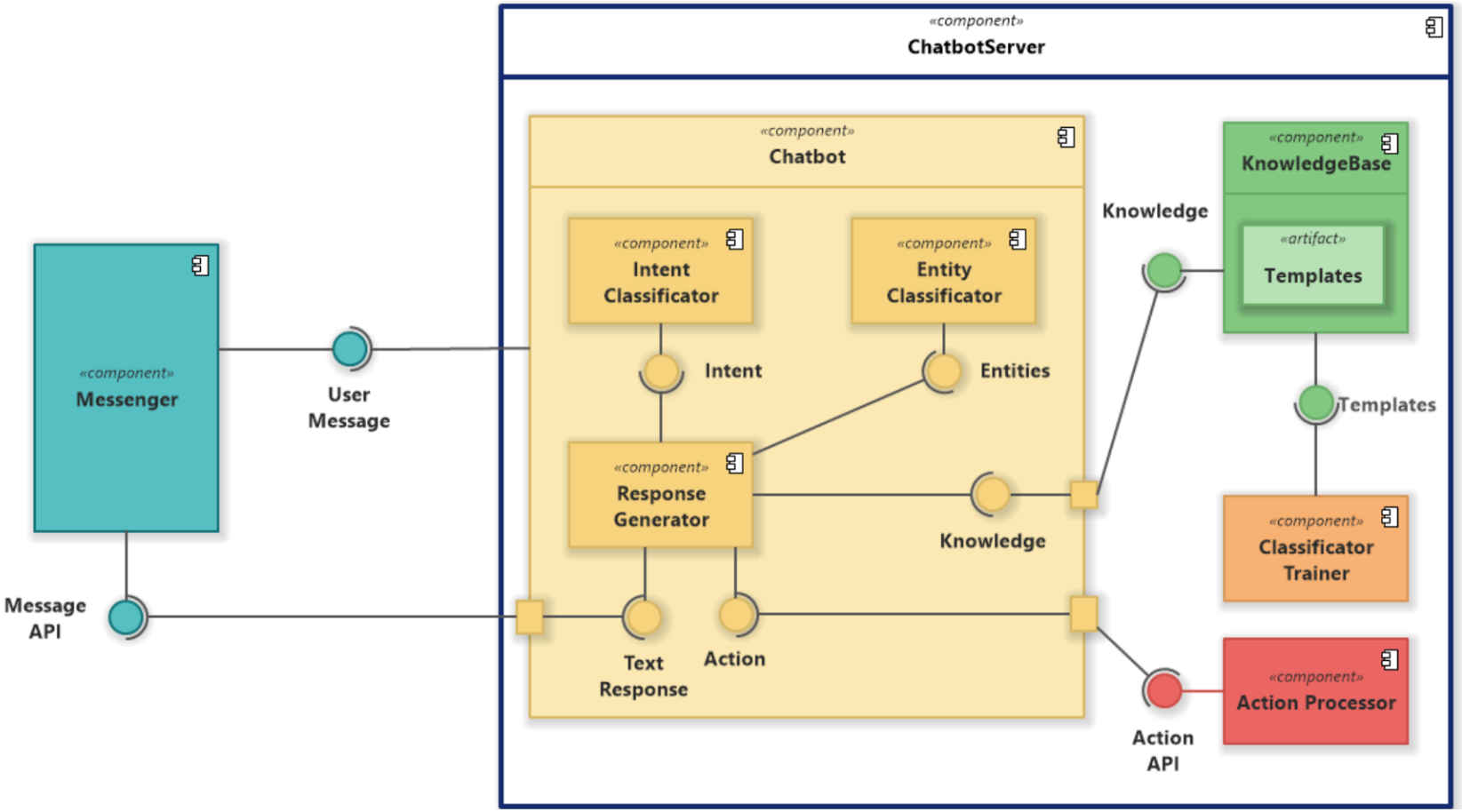
Arkkitehtuurin kuvaamisesta

- On tilanteita, missä sovelluksen arkkitehtuuri on 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

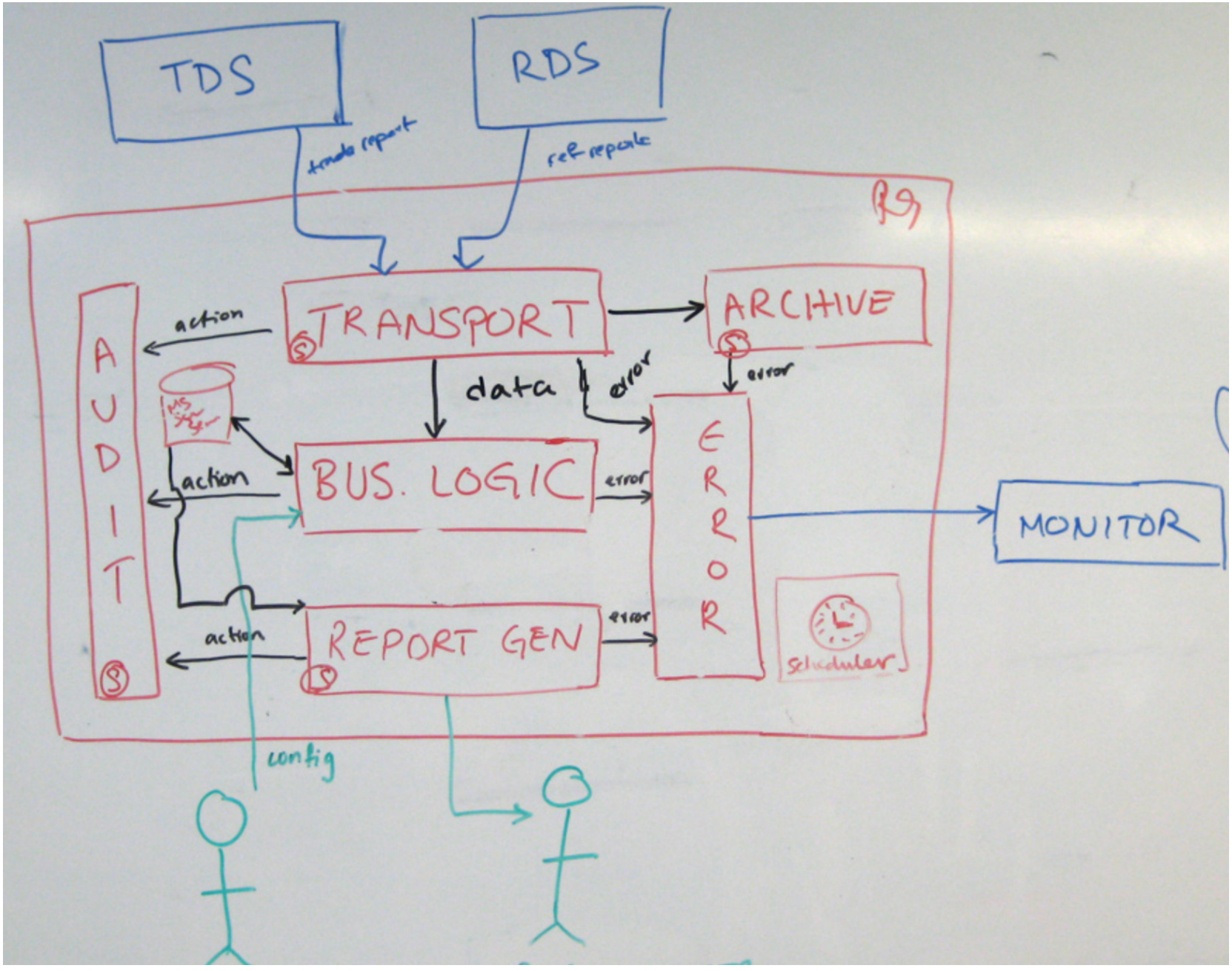
Arkkitehtuurin kuvaamisesta

- On tilanteita, missä sovelluksen arkkitehtuuri on 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*

UML kompomnemnttikaavio



Laatikko ja nuoli -kaavio



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
  - *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
  - *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:
  - *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
  - *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 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 kerrosta, sen verran kuin iteraation tavoitteiden toteuttaminen edellyttää

Ankrementaalinen arkkitehtuuri

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

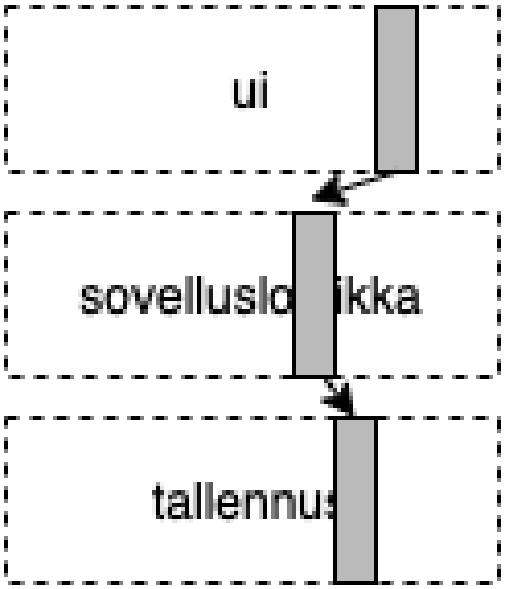
Ankrementaalinen arkkitehtuuri

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

- Rakennetaan skeletonin varaan tuotetta story storyltä

Ominaisuuksiin perustuva integraatio

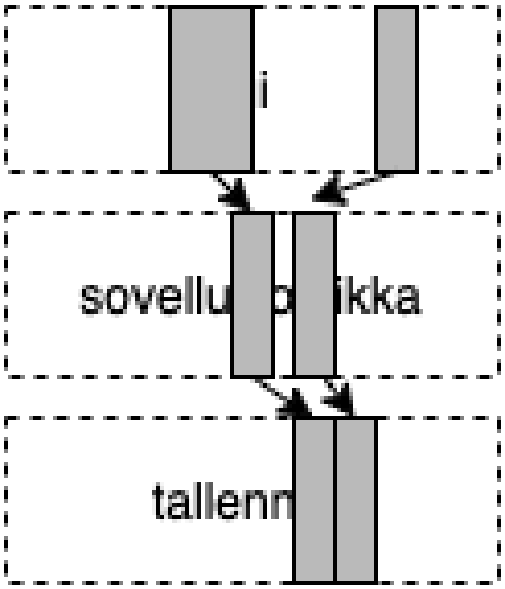
- Alussa ns. *walking skeleton*
  - sisältää tynkäversiot ohjelmiston komponenttirakenteesta



- Rakennetaan skeletonin varaan tuotetta story storyltä

Ominaisuuksiin perustuva integraatio

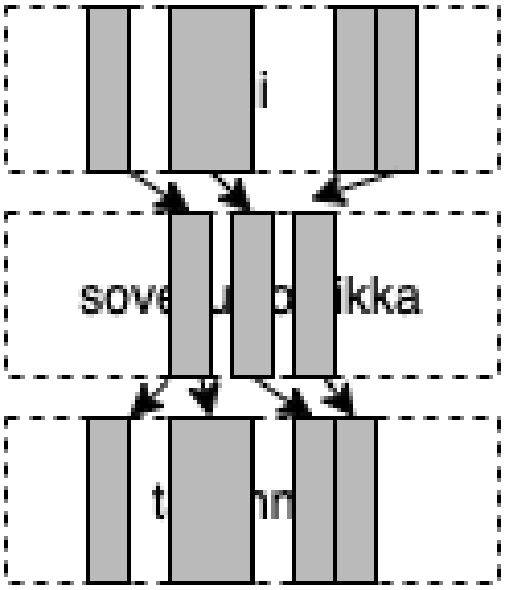
- Alussa ns. *walking skeleton*
  - sisältää tynkäversiot ohjelmiston komponenttirakenteesta



- 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 arkkitehtuurista on vastannut ohjelmistoarkkitehti ja ohjelmoijat ovat olleet velvoitettuja noudattamaan arkkitehtuuria

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 tiimiläisistä nimikettä developer

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 tiimiläisistä nimikettä developer
- 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 tiimiläisistä nimikettä developer
- 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 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

- Oletetaan, että optimaalista arkkitehtuuria ei pystytä suunnittelemaan projektin alussa, kun vaatimuksia, toimintaympäristöä ja toteutusteknologioita ei tunneta
  - Jo tehtyjä arkkitehtuuriratkaisuja muutetaan tarvittaessa

Inkrementaalinen arkkitehtuuri

- Oletetaan, että optimaalista arkkitehtuuria ei pystytä suunnittelemaan projektin alussa, kun vaatimuksia, toimintaympäristöä ja toteutusteknologioita ei tunneta
  - Jo tehtyjä arkkitehtuuriratkaisuja muutetaan tarvittaessa
- Kuten vaatimusmäärittelyssä, myös arkkitehtuurin suunnittelussa agile pyrkii *välttämään liian aikaisin tehtävää, myöhemmin ehkä turhaksi osoittautuvaa työtä*

Inkrementaalinen arkkitehtuuri

- Oletetaan, että optimaalista arkkitehtuuria ei pystytä suunnittelemaan projektin alussa, kun vaatimuksia, toimintaympäristöä ja toteutusteknologioita ei tunneta
  - Jo tehtyjä arkkitehtuuriratkaisuja muutetaan tarvittaessa
- Kuten vaatimusmäärittelyssä, myös arkkitehtuurin suunnittelussa agile pyrkii *välttämään liian aikaisin tehtävää, myöhemmin ehkä turhaksi osoittautuvaa työtä*
- Inkrementaalinen lähestymistapa arkkitehtuurin muodostamiseen edellyttää koodilta hyvää sisäistä laatua ja kehittäjiltä kurinalaisuutta
  - muuten seurauksena on kaaos

Olio-komponenttisuunnittelu

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 tehty ennen ohjelmointia ja dokumentoitu tarkkaan esim. UML:ää käyttäen

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 tehty ennen ohjelmointia ja 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 tehty ennen ohjelmointia ja 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 tehty ennen ohjelmointia ja 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 tuntemus auttaa
  - kehitetty monia suunnittelumenetelmiä, mikään niistä ei ole vakiintunut

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

- 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*, *singleton*, *repository*
  - lisää kurssimateriaalissa ja laskareissa

Koodin laatuattribuutti: kapselointi

- *Kapselointi* ohjelmoinnin peruskursseilla:
  - *oliomuuttujat tulee määritellä piilotetuksi ja niille tulee tehdä tarvittaessa setterit ja getterit*

Koodin laatuattribuutti: kapselointi

- *Kapselointi* ohjelmoinnin peruskursseilla:
  - *oliomuuttujat tulee määritellä piilotetuksi 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

- *Kapselointi* ohjelmoinnin peruskursseilla:
  - *oliomuuttujat tulee määritellä piilotetuksi 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
  - mikropalvelut: yksittäinen palvelu kapseloi sisäisen logiikan, tiedon säilytystavan ja tarjoaa ainoastaan verkon välityksellä käytettävän rajapinnan

Koodin laatuattribuutti: koheesio

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

Koodin laatuattribuutti: koheesio

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

Koodin laatuattribuutti: koheesio

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

Metoditason koheesio

```
def populate(self):
    connection = sqlite3.connect(DATABASE_FILE_PATH)
    connection.row_factory = sqlite3.Row

    cursor = connection.cursor()
    cursor.execute(SQL_SELECT_PARTS)
    rows = cursor.fetchall()

    parts = []

    for row in rows:
        parts.append(Part(row["name"], row["brand"], row["retail_price"]))

    connection.close()

    return parts
```

Metoditason koheesio

```
def populate(self):
    connection = sqlite3.connect(DATABASE_FILE_PATH)
    connection.row_factory = sqlite3.Row

    cursor = connection.cursor()
    cursor.execute(SQL_SELECT_PARTS)
    rows = cursor.fetchall()

    parts = []

    for row in rows:
        parts.append(Part(row["name"], row["brand"], row["retail_price"]))

    connection.close()

    return parts
```

- metodi tekee kolmea eri asiaa

Metoditason koheesio

```
def populate(self):
    connection = self.get_database_connection()
    rows = self.get_rows(connection)
    parts = self.get_parts_by_rows(rows)
    connection.close()
    return parts

def get_database_connection(self):
    connection = sqlite3.connect(DATABASE_FILE_PATH)
    connection.row_factory = sqlite3.Row
    return connection

def get_rows(self, connection):
    cursor = connection.cursor()
    cursor.execute(SQL_SELECT_PARTS)
    return cursor.fetchall()

def get_parts_by_rows(self, rows):
    parts = []
    for row in rows:
        parts.append(Part(row["name"], row["brand"], row["retail_price"]))
    return parts
```

Luokkatason koheesio

```
class Laskin:
    def __init__(self):
        self.lue = input
        self.kirjoita = print

    def suorita(self):
        while True:
            luku1 = int(self.lue("Luku 1:"))

            if luku1 == -9999:
                return

            luku2 = int(self.lue("Luku 2:"))

            if luku2 == -9999:
                return

            vastaus = self.laske_summa(luku1, luku2)

            self.kirjoita(f"Summa: {vastaus}")

    def laske_summa(self, luku1, luku2):
        return luku1 + luku2
```

- Single responisibility: *yksi syy muuttua*

Luokkatason koheesio

```
class Laskin:
    def __init__(self, io):
        self.io = io

    def suorita(self):
        while True:
            luku1 = int(self.io.lue("Luku 1:"))

            if luku1 == -9999:
                return

            luku2 = int(self.io.lue("Luku 2:"))

            if luku2 == -9999:
                return

            vastaus = self.laske_summa(luku1, luku2)

            self.io.kirjoita(f"Summa: {vastaus}")

    def laske_summa(self, luku1, luku2):
        return luku1 + luku2
```

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
- Hyödynnetään rajapintoja ja *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 helppo eliminoida 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 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

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

Koodin laatuattribuutti: testattavuus

- Koodi on helppo testata kattavasti yksikkö- ja integraatiotestein
  - seuraa yleensä siitä, että koodi koostuu löyhästi kytketyistä, selkeän vastuun omaavista komponenteista
- Kurssin alusta asti pyritty hyvään testattavuuteen esim. purkamalla turhia riippuvuuksia dependency injection -periaatteen avulla

Koodin laatuattribuutti: selkeys ja luettavuus

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

Koodin laatuattribuutti: selkeys ja luettavuus

- Perinteisesti ajateltu että koodi kryptistä ja vaikeasti luettavaa
  - yleistä C-kielessä, pyritty esim. optimoimaan tehokkuutta ja muistinkäyttöä
- Nykytrendin mukaan tulee tehdä koodia, joka nimeämisen sekä rakenteen kautta ilmaisee mahdollisimman hyvin sen, mitä koodi tekee

Koodin laatuattribuutti: selkeys ja luettavuus

- Perinteisesti ajateltu että koodi kryptistä ja vaikeasti luettavaa
  - yleistä C-kielessä, pyritty esim. optimoimaan tehokkuutta ja muistinkäyttöä
- Nykytrendin mukaan tulee tehdä 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

- 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:
  - 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
- Koodin rakennetta parantavia refaktorointeja on lukuisia, mm.
  - *rename variable/method/class*
  - *extract method*
  - *move field/method*
  - *extract interface*
  - *extract superclass*

Refaktorointi

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

Miten refaktorointi kannattaa tehdä

- Refaktoroinnin edellytys on kattavien testien olemassaolo

Miten refaktorointi kannattaa tehdä

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

Miten refaktorointi kannattaa tehdä

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

Miten refaktorointi kannattaa tehdä

- Refaktoroinnin edellytys on kattavien testien olemassaolo
- Kannattaa ehdottomasti edetä pienin askelin
  - yksi hallittu muutos kerrallaan
  - testit suoritettava mahdollisimman usein
- Refaktorointia kannattaa suorittaa lähes jatkuvasti
  - 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* (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* (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
  - joskus on jopa asiakkaan kannalta tarkoituksenmukaista tehdä vähemmän laadukasta koodia
- Huonoa suunnittelua tai/ja ohjelmointia 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
  - joskus on jopa asiakkaan kannalta tarkoituksenmukaista tehdä vähemmän laadukasta koodia
- Huonoa suunnittelua tai/ja ohjelmointia 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 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”

- 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 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”
- Joskus tekninen velka pakottaa koodaamaan koko järjestelmän uudelleen