

Ohjelmistotuotanto

Matti Luukkainen ja ohjaajat Valtteri Kantanen, Hannah
Leinson, Riku Rauhala, Ville Saastamoinen

syksy 2023

Luento 7

20.11.2023

Kurssipalaute

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

Ketterien menetelmien testauskäytänteitä

Ketterien menetelmien testauskäytänteitä

- ▶ Sprintissä toteutettavat storyt integroidaan ja testataan sprintin aikana
 - ▶ Sykli ominaisuuden määrittelystä siihen, että se on valmis ja testattu on erittäin lyhyt

Ketterien menetelmien testauskäytänteitä

- ▶ Sprintissä toteutettavat storyt integroidaan ja testataan sprintin aikana
 - ▶ Sykli ominaisuuden määrittelystä siihen, että se on valmis ja testattu on erittäin lyhyt
- ▶ Automatisointi erittäin tärkeässä roolissa, sillä testejä suoritetaan usein

Ketterien menetelmien testauskäytänteitä

- ▶ Sprintissä toteutettavat storyt integroidaan ja testataan sprintin aikana
 - ▶ Sykli ominaisuuden määrittelystä siihen, että se on valmis ja testattu on erittäin lyhyt
- ▶ Automatisointi erittäin tärkeässä roolissa, sillä testejä suoritetaan usein
- ▶ Ideaalilanteessa testaajia sijoitettu kehittäjätiimiin, myös ohjelmoijat kirjoittavat testejä
 - ▶ tiimit *cross functional*

Ketterien menetelmien testauskäytänteitä

Ketterien menetelmien testauskäytänteitä

- ▶ Test driven development (TDD)
 - ▶ Nimestä huolimatta kyseessä toteutus- ja suunnitteluteknikka
 - ▶ Sivutuotteena paljon automaattisesti suoritettavia testejä

Ketterien menetelmien testauskäytänteitä

- ▶ Test driven development (TDD)
 - ▶ Nimestä huolimatta kyseessä toteutus- ja suunnitteluteknikka
 - ▶ Sivutuotteena paljon automaattisesti suoritettavia testejä
- ▶ Acceptance Test Driven Development / Behavior Driven Development
 - ▶ User storyjen tasolla tapahtuva automatisoitu testaus
 - ▶ Robot

Ketterien menetelmien testauskäytänteitä

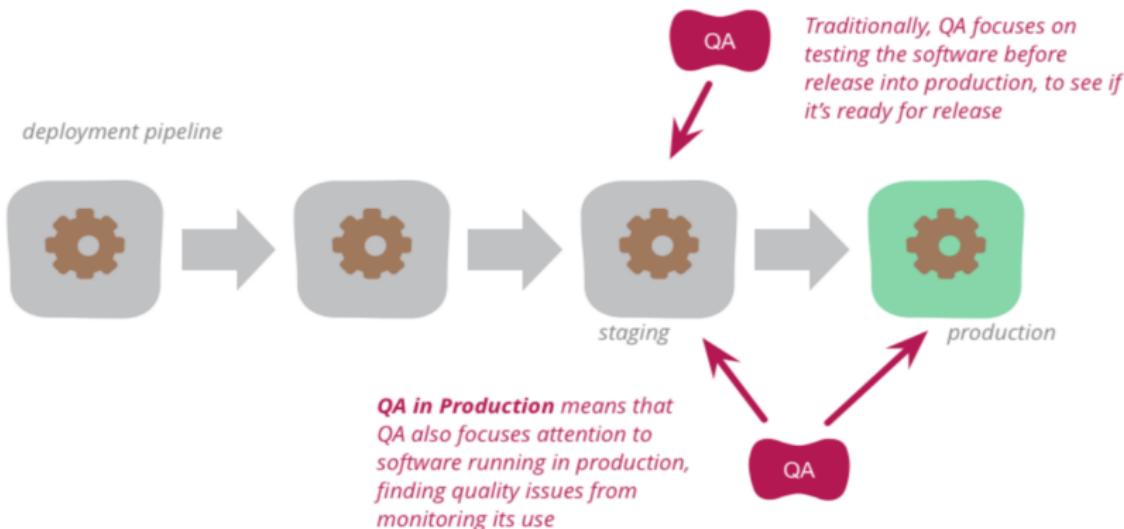
- ▶ Test driven development (TDD)
 - ▶ Nimestä huolimatta kyseessä toteutus- ja suunnitteluteknikka
 - ▶ Sivutuotteena paljon automaattisesti suoritettavia testejä
- ▶ Acceptance Test Driven Development / Behavior Driven Development
 - ▶ User storyjen tasolla tapahtuva automatisoitu testaus
 - ▶ Robot
- ▶ Exploratory testing, suomeksi tutkiva testaus
 - ▶ Järjestelmätestauksen tekniikka, jossa testaaminen tapahtuu ilman formaalia testaussuunnitelmaa

Ketterien menetelmien testauskäytänteitä

- ▶ Test driven development (TDD)
 - ▶ Nimestä huolimatta kyseessä toteutus- ja suunnitteluteknikka
 - ▶ Sivutuotteena paljon automaattisesti suoritettavia testejä
- ▶ Acceptance Test Driven Development / Behavior Driven Development
 - ▶ User storyjen tasolla tapahtuva automatisoitu testaus
 - ▶ Robot
- ▶ Exploratory testing, suomeksi tutkiva testaus
 - ▶ Järjestelmätestauksen tekniikka, jossa testaaminen tapahtuu ilman formaalia testaussuunnitelmaa
- ▶ Jatkuva integraatio (CI) ja jatkuva toimittaminen (CD)
 - ▶ Perinteisen integraatio- ja integraatiotestausvaiheen korvaava työskentelytapa
 - ▶ Ulottuu jopa sovelluksen tuotantoonviemiseen

Tuotannossa tapahtuva testaaminen ja laadunhallinta

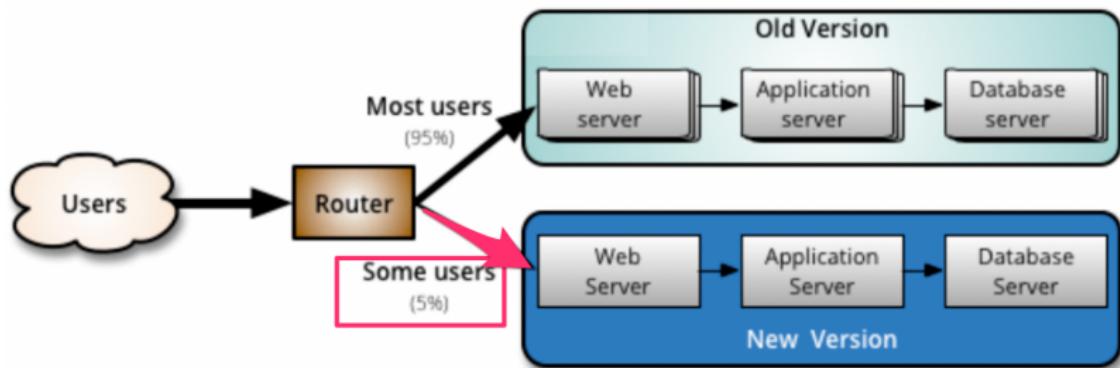
- ▶ Perinteisesti ajateltu: kaikki laadunhallintaan tehdään ennen kuin uudet toiminnallisuudet otetaan käyttöön



- ▶ Viime aikainen trendi on tehdä osa laadunhallinnasta *monitoroimalla* tuotannossa olevaa ohjelmistoa

Canary release

- ▶ Kaksi rinnakkaista tuotanto-ympäristöä, joista uudet ominaisuudet viedään toiseen



- ▶ Uuden ominaisuuden sisältävään ympäristöön ohjataan osa järjestelmän käyttäjistä
- ▶ Uuden ominaisuuden sisältämää versiota *monitoroidaan*
 - ▶ jos ei ongelmia ohjataan kaikki liikenne uuteen versioon

Feature toggle

- ▶ *Feature togglejen* avulla voidaan canary releaset toteuttaa käyttämällä yhtä tuotantopalvelinta

Feature toggle

- ▶ *Feature togglejen* avulla voidaan canary releaset toteuttaa käyttämällä yhtä tuotantopalvelinta
- ▶ Koodiin *ehtolauseita*: osa liikenteestä ohjataan vanhan toteutuksen sijaan testauksen alla olevaan toteutukseen
- ▶ Esim. some-palvelussa feature toggle: *osalle käytetään näytetään uuden algoritmin perusteella generoitu lista uutisia*

```
def recommended_news_generator(user):  
    if is_in_canary_release(user):  
        return experimental_recommendation_algorithm(user)  
    else:  
        return recommendation_algorithm(user)
```

Feature togglejen soveltaminen

- ▶ Aluksi piilotetaan uusi ominaisuus käyttäjiltä feature toggleilla
 - ▶ eli toggle palauttaa vanhan version normaaleille käyttäjille

Feature togglejen soveltaminen

- ▶ Aluksi piilotetaan uusi ominaisuus käyttäjiltä feature toggleilla
 - ▶ eli toggle palauttaa vanhan version normaaleille käyttäjille
- ▶ Sovellus kehittäjien mahdollista valita kumman version toggle palauttaa

Feature togglejen soveltaminen

- ▶ Aluksi piilotetaan uusi ominaisuus käyttäjiltä feature toggleilla
 - ▶ eli toggle palauttaa vanhan version normaaleille käyttäjille
- ▶ Sovellus kehittäjien mahdollista valita kumman version toggle palauttaa
- ▶ Kun valmiina laajempaan testiin, julkaistaan esim.
 - ▶ ensin kehittäjärytyksen omaan käyttöön
 - ▶ sitten osalle käyttäjistä canary releasena

Feature togglejen soveltaminen

- ▶ Aluksi piilotetaan uusi ominaisuus käyttäjiltä feature toggleilla
 - ▶ eli toggle palauttaa vanhan version normaaleille käyttäjille
- ▶ Sovellus kehittäjien mahdollista valita kumman version toggle palauttaa
- ▶ Kun valmiina laajempaan testiin, julkaistaan esim.
 - ▶ ensin kehittäjärytyksen omaan käyttöön
 - ▶ sitten osalle käyttäjistä canary releasena
- ▶ Käytetään paljon A/B-testaamiseen

Feature togglejen soveltaminen

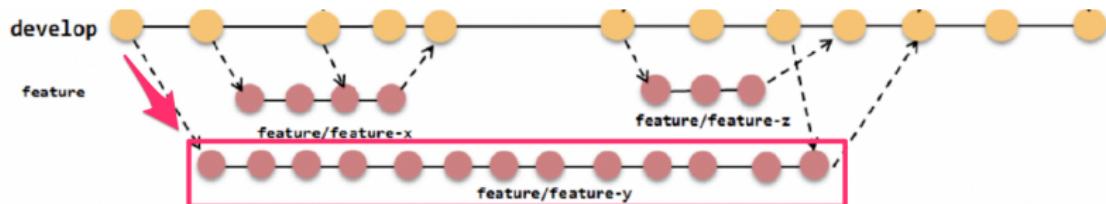
- ▶ Aluksi piilotetaan uusi ominaisuus käyttäjiltä feature toggleilla
 - ▶ eli toggle palauttaa vanhan version normaaleille käyttäjille
- ▶ Sovellus kehittäjien mahdollista valita kumman version toggle palauttaa
- ▶ Kun valmiina laajempaan testiin, julkaistaan esim.
 - ▶ ensin kehittäjärytyksen omaan käyttöön
 - ▶ sitten osalle käyttäjistä canary releasena
- ▶ Käytetään paljon A/B-testaamiseen
- ▶ Lopulta feature toggle ja vanha toteutus voidaan poistaa

Versionhallinnan käytötavoista

Feature branchit

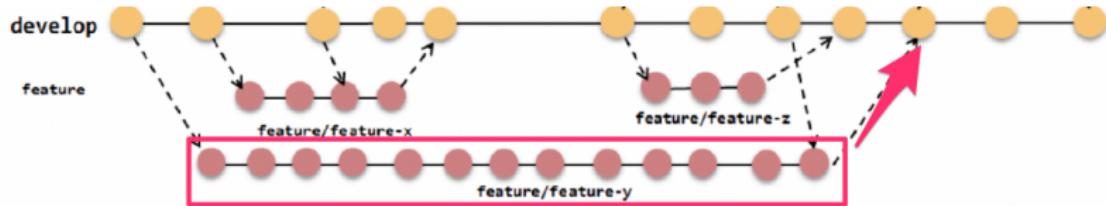
Feature branchit

- ▶ Uusi ominaisuus, esim. user story toteutetaan ensin omaan versionhallinnan haaraansa



Feature branchit

- ▶ Uusi ominaisuus, esim. user story toteutetaan ensin omaan versionhallinnan haaraansa



- ▶ ja ominaisuuden valmistuttua haara mergetään pääkehityshaaraan

Feature branchit ja merge hell

- ▶ Monet pitävät feature brancheja versionhallinnan *parhaana käytänteenä*

Feature branchit ja merge hell

- ▶ Monet pitävät feature brancheja versionhallinnan *parhaana käytänteenä*
- ▶ Viime aikoina huomattu, että feature branchit aiheuttavat helposti pahoja merge-konflikteja sprintin lopussa

Feature branchit ja merge hell

- ▶ Monet pitävät feature brancheja versionhallinnan *parhaana käytänteenä*
- ▶ Viime aikoina huomattu, että feature branchit aiheuttavat helposti pahoja merge-konflikteja sprintin lopussa
- ▶ Seurausena pienimuotoinen integraatiohelvetti: *merge hell*

Feature branchit ja merge hell

- ▶ Monet pitävät feature brancheja versionhallinnan *parhaana käytänteenä*
- ▶ Viime aikoina huomattu, että feature branchit aiheuttavat helposti pahoja merge-konflikteja sprintin lopussa
- ▶ Seurausena pienimuotoinen integraatiohelvetti: *merge hell*
- ▶ Martin Fowler kuuluisassa artikkelissa Continuous integration: *Everyone Commits To the Mainline Every Day*

Feature branchit ja merge hell

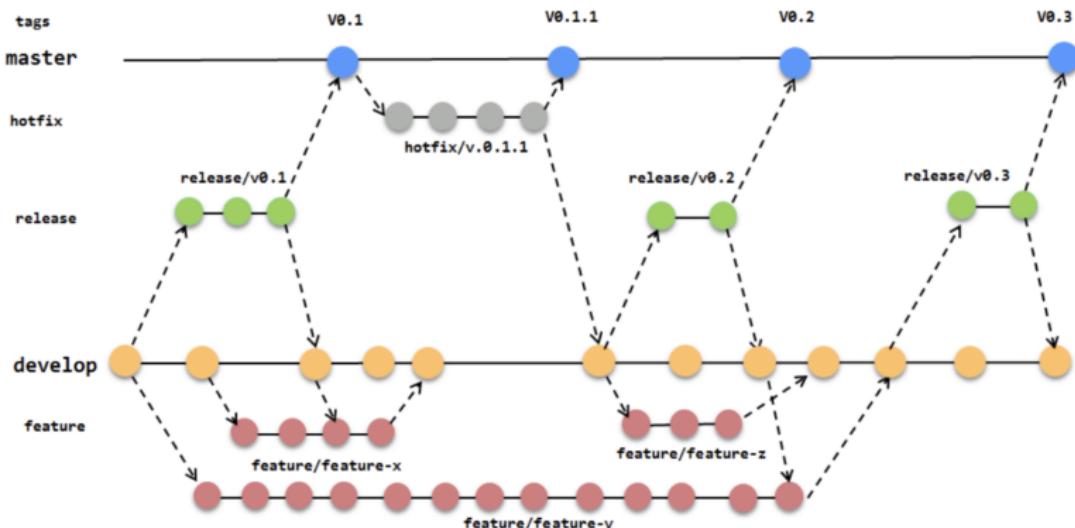
- ▶ Monet pitävät feature brancheja versionhallinnan *parhaana käytänteenä*
- ▶ Viime aikoina huomattu, että feature branchit aiheuttavat helposti pahoja merge-konflikteja sprintin lopussa
- ▶ Seurausena pienimuotoinen integraatiohelvetti: *merge hell*
- ▶ Martin Fowler kuuluisassa artikkelissa Continuous integration: *Everyone Commits To the Mainline Every Day*
- ▶ Voidaanko edes puhua jatkuvasta integraatiosta jos feature branchit ovat käytössä?

Branchayskäytänteet

- ▶ Toisin kuin aiemmissa versionhallintajärjestelmissä, Gitissä brancien tekoon on erittäin helppoa

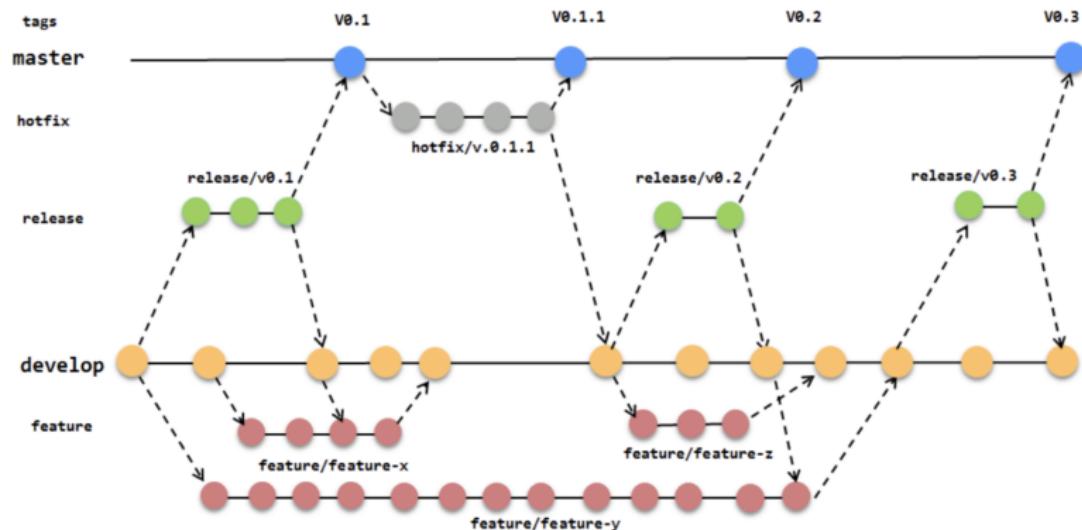
Branchayskäytänteet

- ▶ Toisin kuin aiemmissa versionhallintajärjestelmissä, Gitissä branchien teko on erittäin helppoa
- ▶ Tämä on johtanut monimutkaisiin branchayskäytänteisiin



Branchayskäytänteet

- ▶ Toisin kuin aiemmissa versionhallintajärjestelmissä, Gitissä branchien teko on erittäin helppoa
- ▶ Tämä on johtanut monimutkaisiin branchayskäytänteisiin



- ▶ Tilanne on alkanut osin jo lähteä lapasesta

Trunk based development

Trunk based development

- ▶ Uusi trendi *trunk based development*: pitkäikäisiä feature brancheja ei käytetä ollenkaan
 - ▶ Kaikki koodi suoraan pääkehityshaaraan
 - ▶ ... josta käytetään nimitystä *trunk*

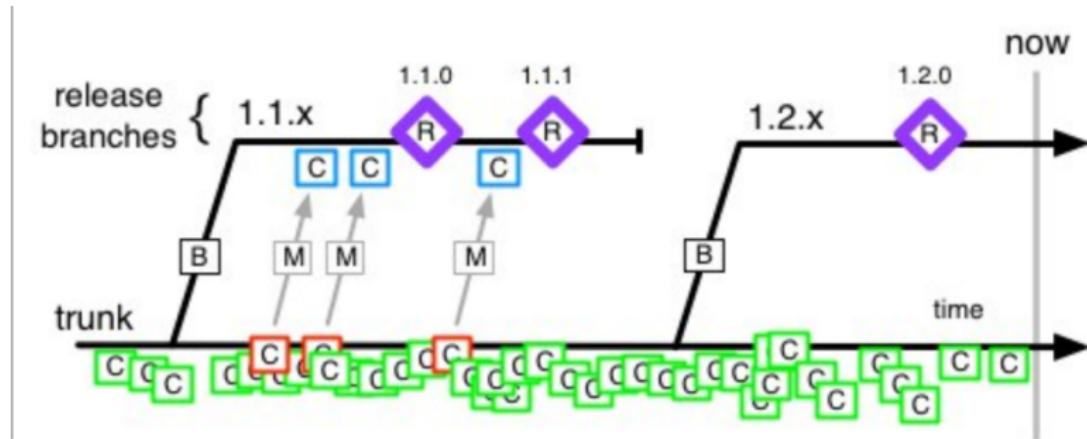
Trunk based development

- ▶ Uusi trendi *trunk based development*: pitkäikäisiä feature brancheja ei käytetä ollenkaan
 - ▶ Kaikki koodi suoraan pääkehityshaaraan
 - ▶ ... josta käytetään nimitystä *trunk*



Trunk based development

- ▶ Uusi trendi *trunk based development*: pitkäikäisiä feature brancheja ei käytetä ollenkaan
 - ▶ Kaikki koodi suoraan pääkehityshaaraan
 - ▶ ... josta käytetään nimitystä *trunk*



- ▶ Ohjelmiston kustakin julkaistusta versiosta saatetaan tehdä oma *release branch*

Trunk based development

- ▶ Pakottaa sovelluskehittäjät tekemään pieniä, nopeasti päähaaraan mergettäviä muutoksia

Trunk based development

- ▶ Pakottaa sovelluskehittäjät tekemään pieniä, nopeasti päähaaraan mergettäviä muutoksia
- ▶ Käytetään feature toggleja
 - ▶ puolivalmiitakin ominaisuuksia voidaan helposti ohjelmoida päähaaraan ilman toiminnallisuuden rikkomista

Trunk based development

- ▶ Pakottaa sovelluskehittäjät tekemään pieniä, nopeasti päähaaraan mergettäviä muutoksia
- ▶ Käytetään feature toggleja
 - ▶ puolivalmiitakin ominaisuuksia voidaan helposti ohjelmoida päähaaraan ilman toiminnallisuuden rikkomista
- ▶ Edellyttää sovelluskehittäjiltä *todella hyvää* kuria ja systemaattisuutta

Trunk based development

- ▶ Pakottaa sovelluskehittäjät tekemään pieniä, nopeasti päähaaraan mergettäviä muutoksia
- ▶ Käytetään feature toggleja
 - ▶ puolivalmiitakin ominaisuuksia voidaan helposti ohjelmoida päähaaraan ilman toiminnallisuuden rikkomista
- ▶ Edellyttää sovelluskehittäjiltä *todella hyvää* kuria ja systemaattisuutta
- ▶ Kehitysmallia noudattavat esim. Google, Facebook ja Netflix

Trunk based development

How GitHub uses GitHub to build GitHub 2012

Trunk based development

How GitHub uses GitHub to build GitHub 2012

*Build features fast. Ship them. That's what we try to do at GitHub. Our process is the anti-process: **what's the minimum overhead we can put up with to keep our code quality high**, all while building features as quickly as possible? It's not just features, either: faster development means happier*

Dev vs Ops

- ▶ Jatkuva toimittaminen ja toimitusvalmius (CD) sekä tuotannossa testaaminen on haastavaa

Dev vs Ops

- ▶ Jatkuva toimittaminen ja toimitusvalmius (CD) sekä tuotannossa testaaminen on haastavaa
- ▶ Perinteisesti tarkka erottelu *sovelluskehittäjien* (developers, dev) ja *ylläpitäjien* (operations, ops) välillä
 - ▶ yleistä että sovelluskehittäjät eivät pääse kirjautumaan tuotantopalvelimille
 - ▶ tuotantoon vieminen ja tietokantaan skeeman päivitykset tapahtuvat ylläpitäjien toimesta

Dev vs Ops

- ▶ Jatkuva toimittaminen ja toimitusvalmius (CD) sekä tuotannossa testaaminen on haastavaa
- ▶ Perinteisesti tarkka erottelu *sovelluskehittäjien* (developers, dev) ja *ylläpitäjien* (operations, ops) välillä
 - ▶ yleistä että sovelluskehittäjät eivät pääse kirjautumaan tuotantopalvelimille
 - ▶ tuotantoon vieminen ja tietokantaan skeeman päivitykset tapahtuvat ylläpitäjien toimesta
- ▶ Jos näin on, tuotantopalvelimelle pystytään viemään uusia versioita vain harvoin, esim 4 kertaa vuodessa

Dev vs Ops

- ▶ Jatkuva toimittaminen ja toimitusvalmius (CD) sekä tuotannossa testaaminen on haastavaa
- ▶ Perinteisesti tarkka erottelu *sovelluskehittäjien* (developers, dev) ja *ylläpitäjien* (operations, ops) välillä
 - ▶ yleistä että sovelluskehittäjät eivät pääse kirjautumaan tuotantopalvelimille
 - ▶ tuotantoon vieminen ja tietokantaan skeeman päivitykset tapahtuvat ylläpitäjien toimesta
- ▶ Jos näin on, tuotantopalvelimelle pystytään viemään uusia versioita vain harvoin, esim 4 kertaa vuodessa
- ▶ Joustavammat toimintamallit vaativat kulttuurinmuutoksen

Dev vs Ops

- ▶ Jatkuva toimittaminen ja toimitusvalmius (CD) sekä tuotannossa testaaminen on haastavaa
- ▶ Perinteisesti tarkka erottelu *sovelluskehittäjien* (developers, dev) ja *ylläpitäjien* (operations, ops) välillä
 - ▶ yleistä että sovelluskehittäjät eivät pääse kirjautumaan tuotantopalvelimille
 - ▶ tuotantoon vieminen ja tietokantaan skeeman päivitykset tapahtuvat ylläpitäjien toimesta
- ▶ Jos näin on, tuotantopalvelimelle pystytään viemään uusia versioita vain harvoin, esim 4 kertaa vuodessa
- ▶ Joustavammat toimintamallit vaativat kulttuurinmuutoksen
- ▶ Kehittäjien (dev) ja ylläpidon (ops) työskenneltävä yhdessä
 - ▶ Sovelluskehittäjille tulee antaa tarvittava pääsy tuotantopalvelimelle
 - ▶ Scrum-tiimiin sijoitetaan ylläpitovastuulla olevia ihmisiä

DevOps

- ▶ *DevOps*: toimintamalli missä dev ja ops työskentelevät tiiviisti yhdessä

DevOps

- ▶ *DevOps*: toimintamalli missä dev ja ops työskentelevät tiiviisti yhdessä
- ▶ DevOps on hypetermi, jonka merkitys osin epäselvä
 - ▶ työpaikkailmoituksissa voidaan arvostaa DevOps-taitoja
 - ▶ tai etsiä ihmistä DevOps-tiimiin
 - ▶ myynnissä mitä erilaisempia DevOps-työkaluja

DevOps

- ▶ *DevOps*: toimintamalli missä dev ja ops työskentelevät tiiviisti yhdessä
- ▶ DevOps on hypetermi, jonka merkitys osin epäselvä
 - ▶ työpaikkailmoituksissa voidaan arvostaa DevOps-taitoja
 - ▶ tai etsiä ihmistä DevOps-tiimiin
 - ▶ myynnissä mitä erilaisempia DevOps-työkaluja
- ▶ Suurin osa järkevistä määritelmistä tarkoittaa DevOpsilla *kehittäjien ja järjestelmäylläpidon yhteistä työntekon tapaa*, ja sen takia onkin hyvä puhua DevOps-kulttuurista

DevOps

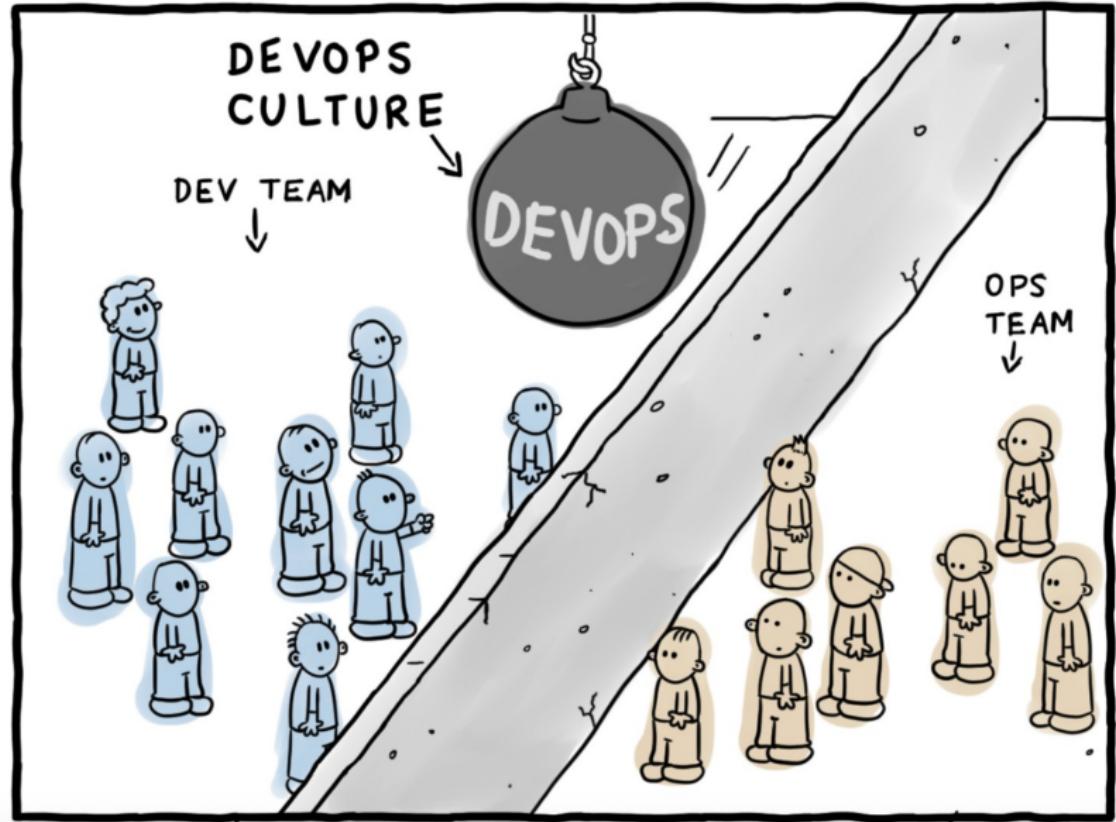
- ▶ *DevOps*: toimintamalli missä dev ja ops työskentelevät tiiviisti yhdessä
- ▶ DevOps on hypetermi, jonka merkitys osin epäselvä
 - ▶ työpaikkailmoituksissa voidaan arvostaa DevOps-taitoja
 - ▶ tai etsiä ihmistä DevOps-tiimiin
 - ▶ myynnissä mitä erilaisempia DevOps-työkaluja
- ▶ Suurin osa järkevistä määritelmistä tarkoittaa DevOpsilla *kehittäjien ja järjestelmäylläpidon yhteistä työntekon tapaa*, ja sen takia onkin hyvä puhua DevOps-kulttuurista
- ▶ Työkaluja/asioita jotka liittyvät DevOpsiin:
 - ▶ automatisoitu testaus
 - ▶ jatkuva integraatio ja toimittaminen (CI/CD)
 - ▶ virtualisointi ja kontainerisointi (Docker)
 - ▶ infrastructure as code
 - ▶ pilvipalveluna toimivat palvelimet ja sovellusympäristöt (PaaS, IaaS, SaaS)

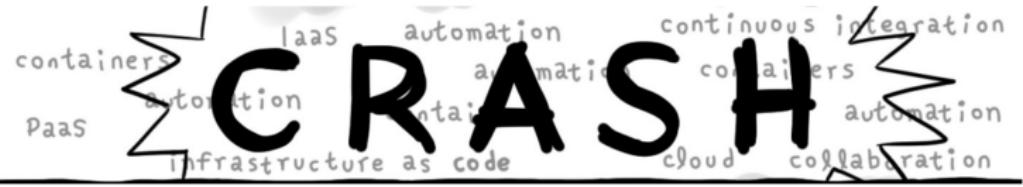
- ▶ Monet listatuista kehittyneet viimeisen 5-10 vuoden aikana ja mahdollistaneet DevOpsin helpomman soveltamisen

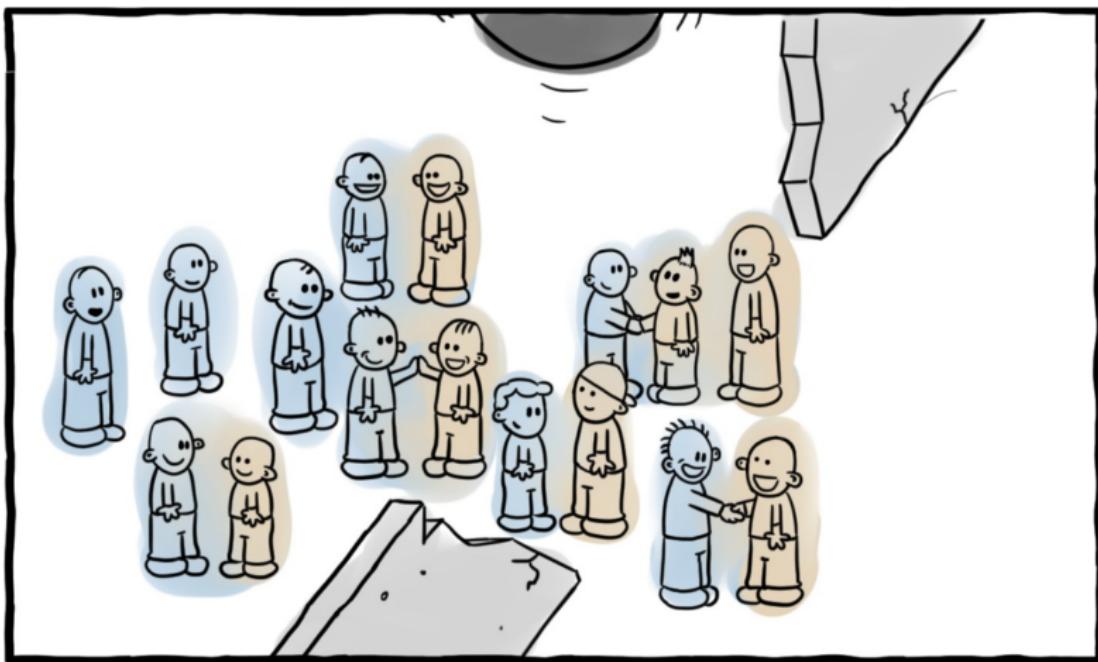
- ▶ Monet listatuista kehittyneet viimeisen 5-10 vuoden aikana ja mahdollistaneet DevOpsin helpomman soveltamisen
- ▶ Eräs tärkeimmistä DevOpsia mahdollistavista asioista *infrastructure as code*
 - ▶ fyysisen palvelinten sijaan virtuaalisia ja pilvessä toimivia palvelimia, joita voi konfiguroida ohjelmallisesti

- ▶ Monet listatuista kehittyneet viimeisen 5-10 vuoden aikana ja mahdollistaneet DevOpsin helpomman soveltamisen
- ▶ Eräs tärkeimmistä DevOpsia mahdollistavista asioista *infrastructure as code*
 - ▶ fyysisen palvelinten sijaan virtuaalisia ja pilvessä toimivia palvelimia, joita voi konfiguroida ohjelmallisesti
- ▶ Raudastakin on tullut “koodia”
 - ▶ palvelinten konfiguraatioita voidaan tallettaa versionhallintaan ja jopa testata
 - ▶ sovelluskehitys ja ylläpito ovat alkaneet muistuttaa toisiaan

- ▶ Monet listatuista kehittyneet viimeisen 5-10 vuoden aikana ja mahdollistaneet DevOpsin helpomman soveltamisen
- ▶ Eräs tärkeimmistä DevOpsia mahdollistavista asioista *infrastructure as code*
 - ▶ fyysisen palvelinten sijaan virtuaalisia ja pilvessä toimivia palvelimia, joita voi konfiguroida ohjelmallisesti
- ▶ Raudastakin on tullut “koodia”
 - ▶ palvelinten konfiguraatioita voidaan tallettaa versionhallintaan ja jopa testata
 - ▶ sovelluskehitys ja ylläpito ovat alkaneet muistuttaa toisiaan
- ▶ Työkalujen käyttöönotto ei riitä, DevOpsin “tekeminen” lähtee kulttuurisista tekijöistä, tiimirakenteista, sekä asioiden sallimisesta







DANIEL STORI {TURNOFF.US}

DevOps: ketteryyys laajennettuna

DevOps: ketteryyys laajennettuna

- ▶ Scrumin ja ketterän eräs tärkeimmistä periaatteista on tehdä kehitystiimeistä itseorganisoituvia ja “cross functional”

DevOps: ketteryyys laajennettuna

- ▶ Scrumin ja ketterän eräs tärkeimmistä periaatteista on tehdä kehitystiimeistä itseorganisoituvia ja “cross functional”
- ▶ DevOps on keino viedä ketteryyttä askeleen pitemmälle
 - ▶ Mahdollistaa että kehitystiimi pystyy viemään vaivattomasti uudet toiminnallisuudet tuotantoalustaan
 - ▶ ja jopa testaamaan sekä operoimaan niitä tuotannossa

DevOps: ketteryys laajennettuna

- ▶ Scrumin ja ketterän eräs tärkeimmistä periaatteista on tehdä kehitystiimeistä itseorganisoituvia ja “cross functional”
- ▶ DevOps on keino viedä ketteryyttä askeleen pitemmälle
 - ▶ Mahdollistaa että kehitystiimi pystyy viemään vaivattomasti uudet toiminnallisuudet tuotantoalustaan
 - ▶ ja jopa testaamaan sekä operoimaan niitä tuotannossa
- ▶ DevOps siis laajentaa ketteryiden koskemaan myös järjestelmälläpitoa

DevOps: ketteryys laajennettuna

- ▶ Scrumin ja ketterän eräs tärkeimmistä periaatteista on tehdä kehitystiimeistä itseorganisoituvia ja “cross functional”
- ▶ DevOps on keino viedä ketteryyttä askeleen pitemmälle
 - ▶ Mahdollistaa että kehitystiimi pystyy viemään vaivattomasti uudet toiminnallisuudet tuotantoalustaan
 - ▶ ja jopa testaamaan sekä operoimaan niitä tuotannossa
- ▶ DevOps siis laajentaa ketteryiden koskemaan myös järjestelmälläpitoa
- ▶ DevOps-ajattelutapa asettaa sovelluskehittäjille lisää osaamisvaatimuksia
 - ▶ kehittäjien pitää hallita enenevissä määrin ylläpitoasioita

13.3.2024-

The screenshot shows a web browser window with the URL devopswithdocker.com in the address bar. The page has a red header with the text "DevOps with Docker". On the left, there's a sidebar with navigation links: "About this course" (which is highlighted in grey), "Getting started", "Links", "Frequently asked questions", and "DevOps with Docker" (with "Part 1", "Part 2", and "Part 3" listed below it). The main content area features a red background with white line-art icons of laptops, monitors, keyboards, and coffee cups. Overlaid on this is a white rectangular box containing the text "DevOps with Docker" in large red letters and "Containers for Beginners" in smaller red letters. Below this section, the heading "About this course" is displayed in a large, bold, black font. A paragraph of text follows, explaining the course content and requirements.

About this course

This course is an introductory course to Docker and docker-compose. The course will also look into what different parts web services consist of, such as reverse proxies, databases, etc. Docker can not be installed on faculty computers, so students will need to use their computers to follow the examples outlined in this course material and to complete the exercises.

► <https://devopswithdocker.com/> by Jami Kousa

TAUKO 10 min

Loppupäätelmiä testauksesta

Loppupäätelmiä testauksesta

- ▶ Ketterissä menetelmissä kantavana teemana on *arvon tuottaminen asiakkaalle*
 - ▶ Sopii ohjeeksi myös arvioitaessa testauksen laajuutta
 - ▶ Testauksella ei ole itseisarvoista merkitystä
 - ▶ Testaamattomuus alkaa pian heikentää tuotteen laatua liikaa

Loppupäätelmiä testauksesta

- ▶ Ketterissä menetelmissä kantavana teemana on *arvon tuottaminen asiakkaalle*
 - ▶ Sopii ohjeeksi myös arvioitaessa testauksen laajuutta
 - ▶ Testauksella ei ole itseisarvoista merkitystä
 - ▶ Testaamattomuus alkaa pian heikentää tuotteen laatua liikaa
- ▶ Testausta ja laadunhallintaa on tehtävä paljon ja toistuvasti eli automatisointi on yleensä pidemmällä tähtäimellä kannattavaa

Loppupäätelmiä testauksesta

- ▶ Ketterissä menetelmissä kantavana teemana on *arvon tuottaminen asiakkaalle*
 - ▶ Sopii ohjeeksi myös arvioitaessa testauksen laajuutta
 - ▶ Testauksella ei ole itseisarvoista merkitystä
 - ▶ Testaamattomuus alkaa pian heikentää tuotteen laatua liikaa
- ▶ Testausta ja laadunhallintaa on tehtävä paljon ja toistuvasti eli automatisointi on yleensä pidemmällä tähtäimellä kannattavaa
- ▶ Automatisointi ei ole halpaa eikä helppoa
 - ▶ Väärin, väärään aikaan tai väärälle tasolle tehdyt automatisoidut testit voivat tuottaa enemmän harmia ja kustannuksia kuin hyötyä

- ▶ Jos ohjelmistossa komponentteja, jotka tullaan ehkä poistamaan tai korvaamaan pian, ei niiden testejä kannata automatisoida
 - ▶ esim. jos kyseessä *minimal viable product*

- ▶ Jos ohjelmistossa komponentteja, jotka tullaan ehkä poistamaan tai korvaamaan pian, ei niiden testejä kannata automatisoida
 - ▶ esim. jos kyseessä *minimal viable product*
- ▶ Väliaikaiseksi tarkoitettu komponentti voi jäädä järjestelmään vuosiksi...

- ▶ Jos ohjelmistossa komponentteja, jotka tullaan ehkä poistamaan tai korvaamaan pian, ei niiden testejä kannata automatisoida
 - ▶ esim. jos kyseessä *minimal viable product*
- ▶ Väliaikaiseksi tarkoitettu komponentti voi jäädä järjestelmään vuosiksi...
- ▶ Aluksi kannattaa ohjelman rakenteen ensin antaa stabiloitua, kattavammat testit vasta myöhemmin

- ▶ Jos ohjelmistossa komponentteja, jotka tullaan ehkä poistamaan tai korvaamaan pian, ei niiden testejä kannata automatisoida
 - ▶ esim. jos kyseessä *minimal viable product*
- ▶ Väliaikaiseksi tarkoitettu komponentti voi jäädä järjestelmään vuosiksi...
- ▶ Aluksi kannattaa ohjelman rakenteen ensin antaa stabiloitua, kattavammat testit vasta myöhemmin
- ▶ *Testattavuus* tulee pitää koko ajan mielessä

- ▶ Oppikirjamääritelmän mukaista TDD:tä sovelletaan harvoin
 - ▶ Välillä TDD hyödyllinen, esim. testattaessa rajapintoja, joita käyttäviä komponentteja ei ole vielä olemassa
 - ▶ Testit tekee samalla vaivalla kuin “pääohjelman”

- ▶ Oppikirjamääritelmän mukaista TDD:tä sovelletaan harvoin
 - ▶ Välillä TDD hyödyllinen, esim. testattaessa rajapintoja, joita käyttäviä komponentteja ei ole vielä olemassa
 - ▶ Testit tekee samalla vaivalla kuin “pääohjelman”
- ▶ Kattavien yksikkötestien tekeminen ei yleensä ole mielekästä ohjelman kaikille luokille

- ▶ Oppikirjamääritelmän mukaista TDD:tä sovelletaan harvoin
 - ▶ Välillä TDD hyödyllinen, esim. testattaessa rajapintoja, joita käyttäviä komponentteja ei ole vielä olemassa
 - ▶ Testit tekee samalla vaivalla kuin “pääohjelman”
- ▶ Kattavien yksikkötestien tekeminen ei yleensä ole mielekästä ohjelman kaikille luokille
- ▶ Yksikkötestaus hyödyllisimmillään kompleksia logiikkaa sisältäviä luokkia testattaessa

- ▶ Oppikirjamääritelmän mukaista TDD:tä sovelletaan harvoin
 - ▶ Välillä TDD hyödyllinen, esim. testattaessa rajapintoja, joita käyttäviä komponentteja ei ole vielä olemassa
 - ▶ Testit tekee samalla vaivalla kuin “pääohjelman”
- ▶ Kattavien yksikkötestien tekeminen ei yleensä ole mielekästä ohjelman kaikille luokille
- ▶ Yksikkötestaus hyödyllisimmillään kompleksia logiikkaa sisältäviä luokkia testattaessa
- ▶ Mielummin integraatiotason testejä ohjelman isompien komponenttien rajapintoja vasten
 - ▶ Pysyvät todennäköisemmin valideina komponenttien sisäisen rakenteen muuttuessa

- ▶ Automaattisia testejä kannattaa tehdä etenkin niiden komponenttien rajapointoihin, joita muokataan usein

- ▶ Automaattisia testejä kannattaa tehdä etenkin niiden komponenttien rajapointoihin, joita muokataan usein
- ▶ Käyttöliittymän läpi suoritettavat, käyttäjän interaktioita simuloivat testit usein hyödyllisimpää
 - ▶ Liian aikaisin tehtynä ne saattavat aiheuttaa kohtuuttoman paljon ylläpitovaivaa

- ▶ Testitapauksista kannattaa aina tehdä todellisia käyttöskenaarioita vastaavia
 - ▶ Pelkkiä testauskattavuutta kasvattavia testejä on turha tehdä

- ▶ Testitapauksista kannattaa aina tehdä todellisia käyttöskenaarioita vastaavia
 - ▶ Pelkkiä testauskattavuutta kasvattavia testejä on turha tehdä
- ▶ Erityisesti järjestelmätason testeissä kannattaa käyttää mahdollisimman oikeanlaista dataa
 - ▶ Koodissa hajoaa aina jotain kun käytetään oikeaa dataa riippumatta siitä miten hyvin testaus on suoritettu

- ▶ Testitapauksista kannattaa aina tehdä todellisia käyttöskenaarioita vastaavia
 - ▶ Pelkkiä testauskattavuutta kasvattavia testejä on turha tehdä
- ▶ Erityisesti järjestelmätason testeissä kannattaa käyttää mahdollisimman oikeanlaista dataa
 - ▶ Koodissa hajoaa aina jotain kun käytetään oikeaa dataa riippumatta siitä miten hyvin testaus on suoritettu
- ▶ Parasta on jos staging-ympäristössä on käytössä sama *data kuin* tuotantoymäristössä

- ▶ Ehdottomasti kaikkein tärkein laadunhallinnan kannalta on **mahdollisimman usein tapahtuva käyttöönotto**
 - ▶ edellyttää hyvin rakennettua deployment pipelineä, kohtuullista testauksen automatisointia

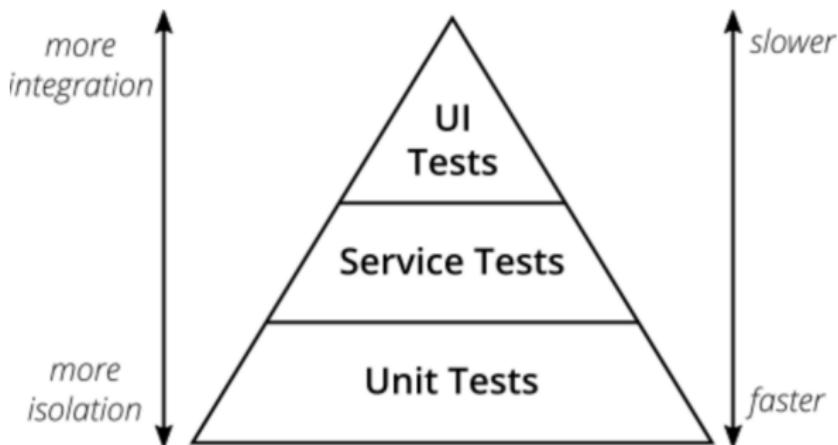
- ▶ Ehdottomasti kaikkein tärkein laadunhallinnan kannalta on **mahdollisimman usein tapahtuva käyttöönotto**
 - ▶ edellyttää hyvin rakennettua deployment pipelineä, kohtuullista testauksen automatisointia
- ▶ Trunk based development auttaa nopeaa käyttöönottoa feature brancheihin verrattuna

- ▶ Ehdottomasti kaikkein tärkein laadunhallinnan kannalta on **mahdollisimman usein tapahtuva käyttöönotto**
 - ▶ edellyttää hyvin rakennettua deployment pipelineä, kohtuullista testauksen automatisointia
- ▶ Trunk based development auttaa nopeaa käyttöönottoa feature brancheihin verrattuna
- ▶ Suosittelen että käyttöönotto tapahtuu niin usein kuin mahdollista, jopa useita kertoja päivässä
 - ▶ takaa sen, että pahoja integrointiongelmia ei synny
 - ▶ sovellukseen syntyvät regressiot havaitaan ja pystytään korjaamaan mahdollisimman nopeasti

- ▶ Ehdottomasti kaikkein tärkein laadunhallinnan kannalta on **mahdollisimman usein tapahtuva käyttöönotto**
 - ▶ edellyttää hyvin rakennettua deployment pipelineä, kohtuullista testauksen automatisointia
- ▶ Trunk based development auttaa nopeaa käyttöönottoa feature brancheihin verrattuna
- ▶ Suosittelen että käyttöönotto tapahtuu niin usein kuin mahdollista, jopa useita kertoja päivässä
 - ▶ takaa sen, että pahoja integrointiongelmia ei synny
 - ▶ sovellukseen syntyvät regressiot havaitaan ja pystytään korjaamaan mahdollisimman nopeasti
- ▶ Nopea käyttöönotto **pakottaa** laatuun

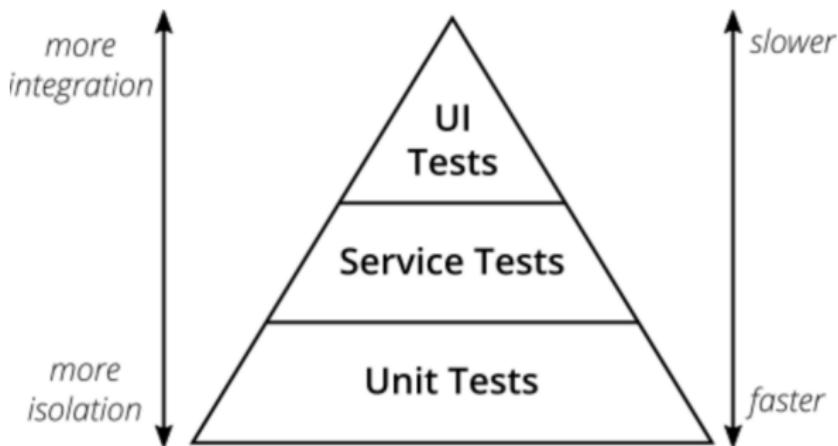
Testauspyramiidi

- ▶ Oma näkemykseni poikkeaa jossain määrin ns *testauspyramidista*



Testauspyramidi

- ▶ Oma näkemykseni poikkeaa jossain määrin ns *testauspyramidista*



- ▶ DISA: 570 yksikkötestiä, ja kaikki vihreällä. Softa ei edes käynnisty...

Guillermo Rauch



Guillermo Rauch
@rauchhg

Write tests. Not too many. Mostly integration.

4:43 PM · December 10th, 2016

24

473

1,360



Kent Dodds: testauspokaali

THE FOUR TYPES OF TESTS

End to End

A helper robot that behaves like a user to click around the app and verify that it functions correctly.

Sometimes called "functional testing" or e2e.

Integration

Verify that several units work together in harmony.

Unit

Verify that individual, isolated parts work as expected.

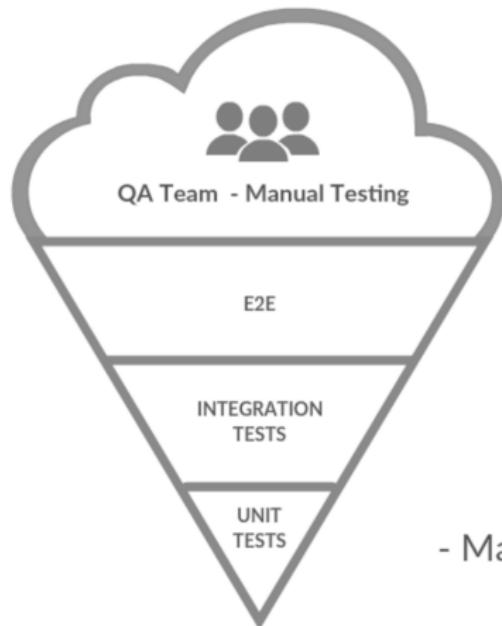
Static

Catch typos and type errors as you write the code.



Testausjäätelö

- ▶ On oltava varuillaan että ei synnytetä *testausjäätelöä*

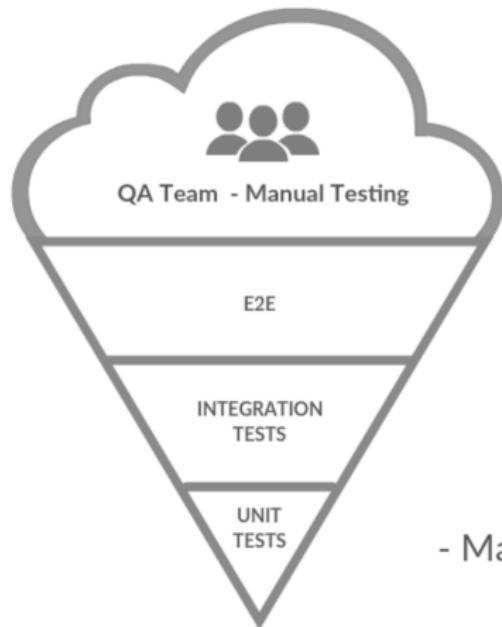


ICE CREAM CONE

- Very slow to execute
- Bugs are hard to find
- Manual Testers are swamped

Testausjäätelö

- ▶ On oltava varuillaan että ei synnytetä *testausjäätelöä*



ICE CREAM CONE

- Very slow to execute
- Bugs are hard to find
- Manual Testers are swamped

- ▶ Jälkikäteen yleensä helppo sanoa miten olisi kannattanut testata...

Tieteellinen evidenssi

Tieteellinen evidenssi

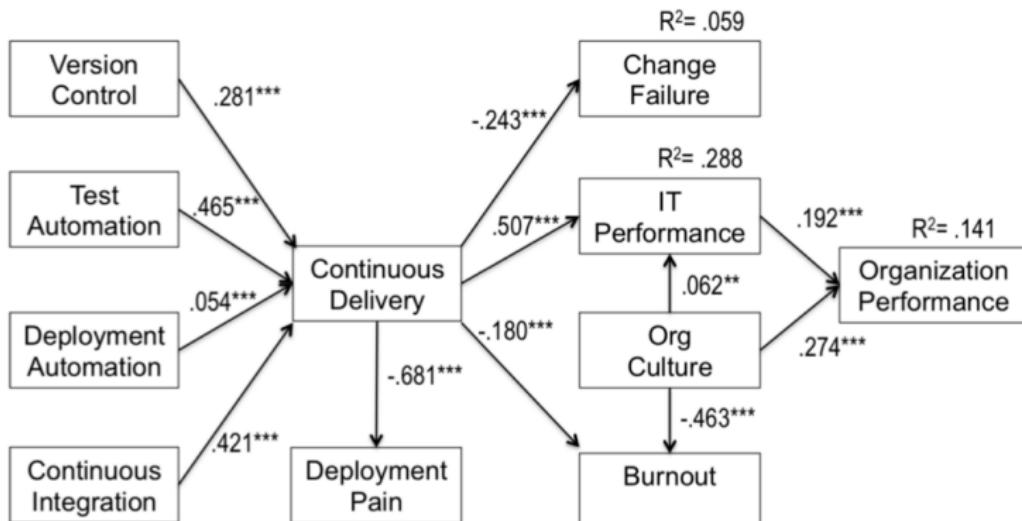
- ▶ Edellä esitellyistä ohjelmistojen käyttöönnoton ja laadunhallinnan käytenteiden toimivuudesta on runsaasti anekdotalista evidenssiä

Tieteellinen evidenssi

- ▶ Edellä esitellyistä ohjelmistojen käyttöönnoton ja laadunhallinnan käytenteiden toimivuudesta on runsaasti anekdotaalista evidenssiä
- ▶ Accelerate: *The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations* 2018

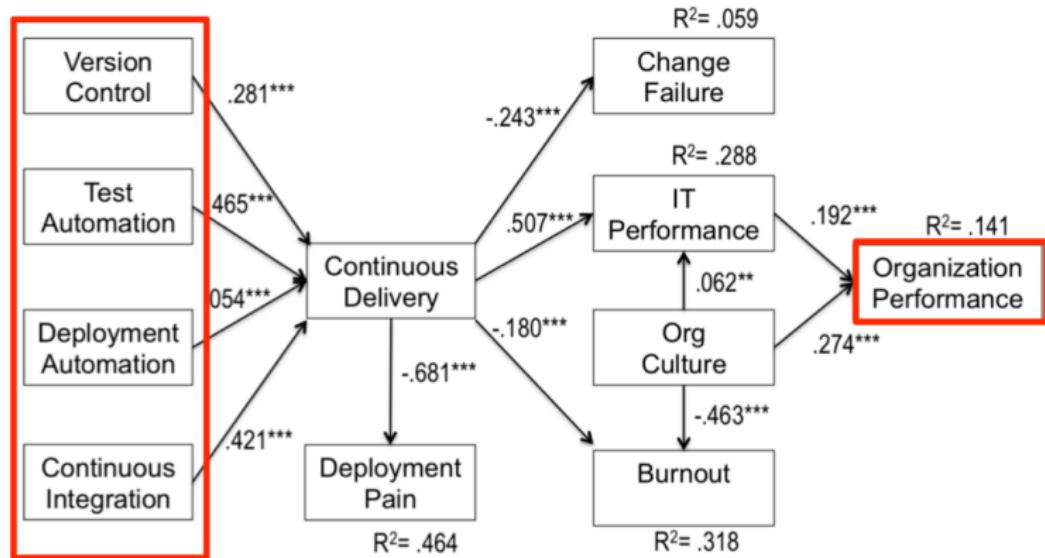
Tieteellinen evidenssi

- ▶ Edellä esitellyistä ohjelmistojen käyttöönnoton ja laadunhallinnan käytenteiden toimivuudesta on runsaasti anekdotaalista evidenssiä
- ▶ Accelerate: *The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations* 2018



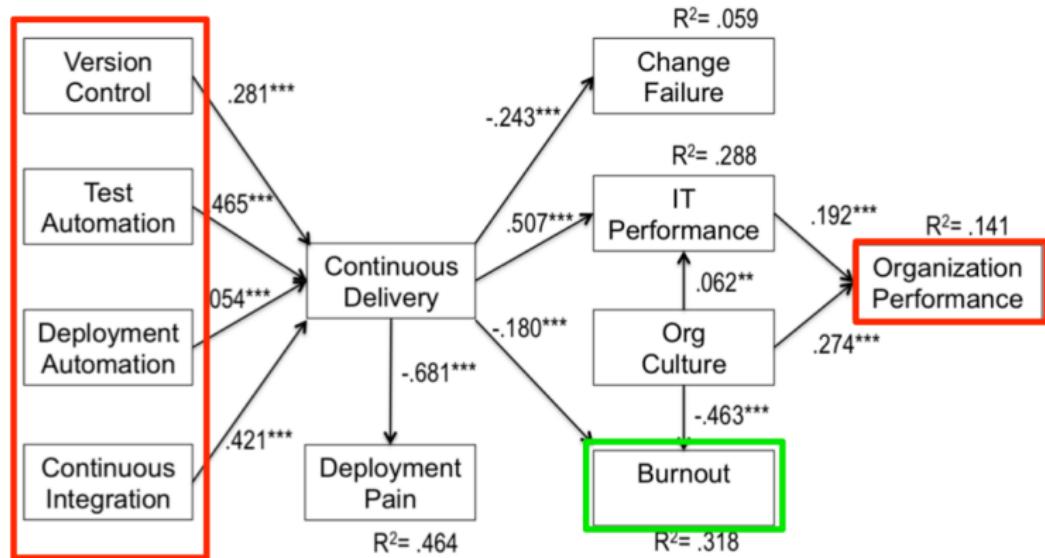
Tieteellinen evidenssi

- ▶ Edellä esitellyistä julkaisun ja laadunhallinnan käytenteiden toimivuudesta on paljon runsaasti anekdotalista evidenssiä
- ▶ Accelerate: *The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations* 2018



Tieteellinen evidenssi

- ▶ Edellä esitellyistä julkaisun ja laadunhallinnan käytenteiden toimivuudesta on paljon runsaasti anekdotalista evidenssiä
- ▶ Accelerate: *The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations* 2018



Laadunhallinta käsitelty, seuraavana...

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
- ▶ Siirrymme käsittelemään ohjelmiston suunnittelua ja toteuttamista
 - ▶ osa suunnittelusta tapahtuu vasta toteutusvaiheessa, joten suunnittelun ja toteuttamisen käsitteilyä 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
- ▶ Siirrymme käsittelemään ohjelmiston suunnittelua ja toteuttamista
 - ▶ osa suunnittelusta tapahtuu vasta toteutusvaiheessa, joten suunnittelun ja toteuttamisen käsitteilyä 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:

Ohjelmiston suunnittelu

- ▶ Jakautuu kahteen vaiheeseen:
 - ▶ arkkitehtuurisuunnittelu
 - ▶ olio/komponenttisuunnittelu
- ▶ Ajoittuminen riippuu käytettävästä tuotantoprosessista:
- ▶ Vesiputoosmallissa vaatimusmäärittelyn jälkeen, ennen toteutuksen aloittamista, tarkasti dokumentoitu

Ohjelmiston suunnittelu

- ▶ Jakautuu kahteen vaiheeseen:
 - ▶ arkkitehtuurisuunnittelu
 - ▶ olio/komponenttisuunnittelu
- ▶ Ajoittuminen riippuu käytettävästä tuotantoprosessista:
- ▶ Vesiputoosmallissa vaatimusmäärittelyn jälkeen, ennen toteutuksen aloittamista, tarkasti dokumentoitu
- ▶ Ketterissä menetelmissä suunnittelua tehdään tarvittava määärä jokaisessa iteraatiossa
 - ▶ ei yleensä tarkkaa suunnitteludokumenttia

Ohjelmiston suunnittelu

- ▶ Jakautuu kahteen vaiheeseen:
 - ▶ arkkitehtuurisuunnittelu
 - ▶ olio/komponenttisuunnittelu
- ▶ Ajoittuminen riippuu käytettävästä tuotantoprosessista:
- ▶ Vesiputoousmallissa vaatimusmäärittelyn jälkeen, ennen toteutuksen aloittamista, tarkasti dokumentoitu
- ▶ Ketterissä menetelmissä suunnittelua tehdään tarvittava määärä jokaisessa iteraatiossa
 - ▶ ei yleensä tarkkaa suunnitteludokumenttia
- ▶ Vesiputoousmallin suunnitteluprosessi tuskin on enää käytössä
 - ▶ “jäykimmissäkin” prosesseissa ainakin vaatimusmäärittely ja arkkitehtuurisuunnittelu limittivät

Ohjelmiston suunnittelu

- ▶ Jakautuu kahteen vaiheeseen:
 - ▶ arkkitehtuurisuunnittelu
 - ▶ olio/komponenttisuunnittelu
- ▶ Ajoittuminen riippuu käytettävästä tuotantoprosessista:
- ▶ Vesiputoousmallissa vaatimusmäärittelyn jälkeen, ennen toteutuksen aloittamista, tarkasti dokumentoitu
- ▶ Ketterissä menetelmissä suunnittelua tehdään tarvittava määrä jokaisessa iteraatiossa
 - ▶ ei yleensä tarkkaa suunnitteludokumenttia
- ▶ Vesiputoousmallin suunnitteluprosessi tuskin on enää käytössä
 - ▶ "jäykimmissäkin" prosesseissa ainakin vaatimusmäärittely ja arkkitehtuurisuunnittelu limittyytä
- ▶ Tarkkaa ennen ohjelointia tapahtuvaa suunnittelua toki edelleen tapahtuu ja joihinkin tilanteisiin se sopiikin

Ohjelmiston suunnittelu

- ▶ Jakautuu kahteen vaiheeseen:
 - ▶ arkkitehtuurisuunnittelu
 - ▶ olio/komponenttisuunnittelu
- ▶ Ajoittuminen riippuu käytettävästä tuotantoprosessista:
- ▶ Vesiputoousmallissa vaatimusmäärittelyn jälkeen, ennen toteutuksen aloittamista, tarkasti dokumentoitu
- ▶ Ketterissä menetelmissä suunnittelua tehdään tarvittava määrä jokaisessa iteraatiossa
 - ▶ ei yleensä tarkkaa suunnitteludokumenttia
- ▶ Vesiputoousmallin suunnitteluprosessi tuskin on enää käytössä
 - ▶ "jäykimmissäkin" prosesseissa ainakin vaatimusmäärittely ja arkkitehtuurisuunnittelu limittyytä
- ▶ Tarkkaa ennen ohjelointia tapahtuvaa suunnittelua toki edelleen tapahtuu ja joihinkin tilanteisiin se sopiikin
- ▶ Näiden lisäksi UI/UX-suunnittelu

Ohjelmiston arkkitehtuuri

Ohjelmiston arkkitehtuuri

- ▶ Abstraktimpi kuvaus joka määrittelee ohjelmiston suuret linjat

Ohjelmiston arkkitehtuuri

- ▶ Abstraktimpi kuvaus joka määrittelee ohjelmiston suuret linjat
- ▶ 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
 - ▶ käytettävyyss, saavutettavuus
 - ▶ suorituskyky, skaalautuvuus
 - ▶ vikasietoisuus, tiedon ajantasaisuus
 - ▶ tietoturva
 - ▶ ylläpidettävyyss, 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
 - ▶ lainsääädäntö

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

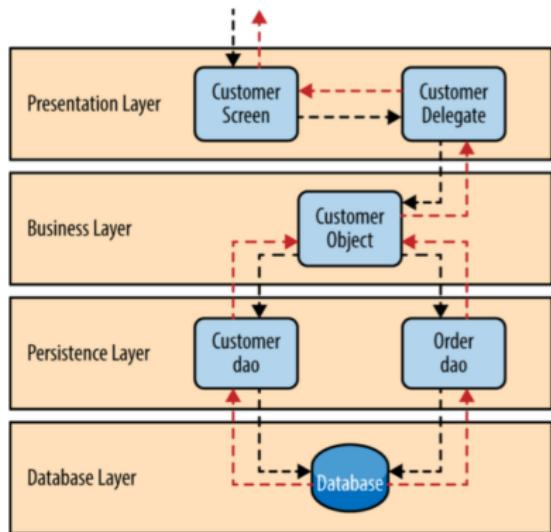
- ▶ Ohjelmiston arkkitehtuuri perustuu yleensä yhteen tai useampaan **arkkitehtuurityyliin** (architectural style)
 - ▶ hyväksi havaittua tapaa strukturoida tietyytyyppisiä sovelluksia

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
 - ▶ ...

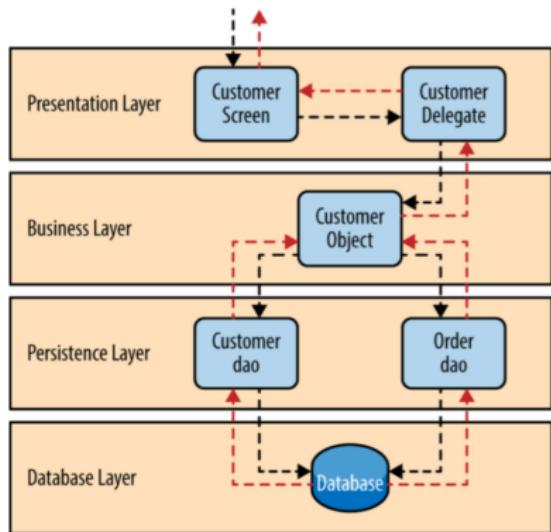
Kerrosarkkitehtuuri

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



Kerrosarkkitehtuuri

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



- ▶ Kerros käyttää ainoastaan alempaan 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

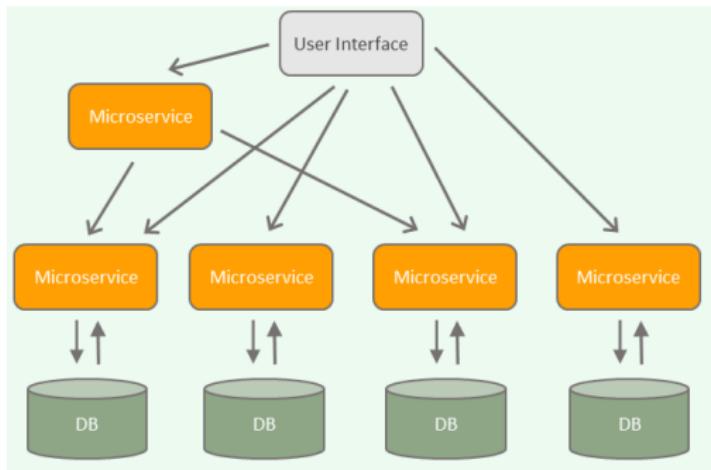
- ▶ Kerrokset omalla abstraktiotasollaan
 - ▶ Ylimmät kerrokset ovat lähellä käyttäjää: UI ja sovelluslogiikka
 - ▶ Alimmat kerrokset taas keskittyvät koneläheisiin asioihin: esim. tiedon tallennus
- ▶ Kerroksittaisuus helpottaa ylläpitoa
 - ▶ yhden kerroksen muutokset vaikuttavat korkeintaan yläpuolella olevaan kerrokseen
- ▶ Sovelluslogiikan riippumattomuus käyttöliittymästä helpottaa ohjelman siirtämistä uusille alustoiille
 - ▶ esim. web-sovelluksesta voidaan tehdä mobiiliversio
- ▶ Alimpien kerroksien palveluja, voidaan osin uusiokäyttää myös muissa sovelluksissa
- ▶ Kerroksittaisuus saattaa johtaa massiivisiin monoliittisiin sovelluksiin
 - ▶ vaikea laajentaa ja skaalaata suurille käyttäjämääritteille
 - ▶ haastavaa kehittää jos sovelluskehittäjiä suuri määrä

Mikropalveluarkkitehtuuri

- ▶ Mikropalveluarkkitehtuuri (microservice) pyrkii vastaamaan näihin haasteisiin

Mikropalveluarkkitehtuuri

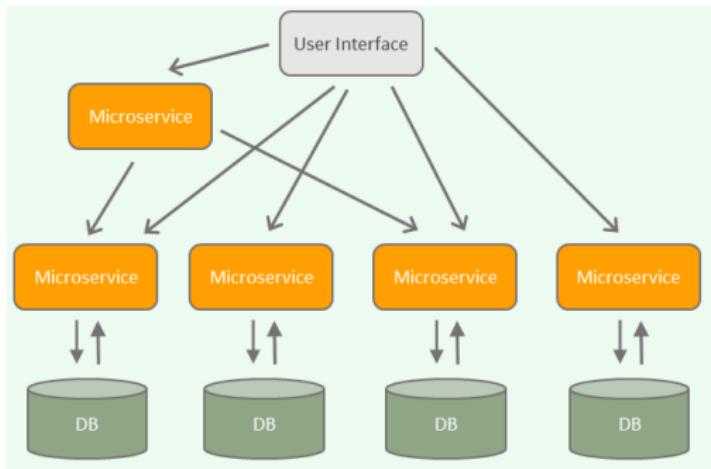
- ▶ Mikropalveluarkkitehtuuri (microservice) pyrkii vastaamaan näihin haasteisiin



- ▶ 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

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 tuotantopalvelimilla on haastavaa ja vaatii pitkälle menevää automatisointia
 - ▶ Sama koskee sovelluskehitysympäristöä ja jatkuvaan integraatiota

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 tuotantopalvelimilla on haastavaa ja vaatii pitkälle menevää automatisointia
 - ▶ Sama koskee sovelluskehitysympäristöä ja jatkuvaan integraatiota
- ▶ 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 tuotantopalvelimilla on haastavaa ja vaatii pitkälle menevää automatisointia
 - ▶ Sama koskee sovelluskehitysympäristöä ja jatkuvaa integraatiota
- ▶ Mikropalveluiden menestyksekäs soveltaminen edellyttää vahvaa DevOps-kulttuuria
- ▶ Kehitetty massiivisiin järjestelmiin (mm Amazon, Netflix)
 - ▶ onko järkevä kaikkialla?