

Ohjelmistotuotanto

Matti Luukkainen ja ohjaajat Kalle Ilves, Silva Perander, Topias Pyykönen, Jussi Laisi, Petrus Peltola, Kristian Krok

syksy 2020

Luento 7

16.11.2020

Miniprojektien aloitustilaisuudet



Ketterien menetelmien testauskäytänteet

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

Ketterien menetelmien testauskäytänteet

- ▶ 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 testauskäytänteet

- ▶ 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 testauskäytänteet

- ▶ Test driven development (TDD)
 - ▶ Nimistä huolimatta kyseessä toteutus- ja suunnitteluteknikka
 - ▶ Sivutuotteena paljon automaattisesti suoritettavia testejä
- ▶ Acceptance Test Driven Development / Behavior Driven Development
 - ▶ User storyjen tasolla tapahtuva automatisoitu testaus
- ▶ Continuous Integration (CI) eli jatkuva integraatio
 - ▶ Perinteisen integraatio- ja integraatiotestausvaiheen korvaava työskentelytapa
- ▶ Exploratory testing, suomeksi tutkiva testaus
 - ▶ Järjestelmätestauksen tekniikka, jossa testaaminen tapahtuu ilman formaalia testaussuunnitelmaa

Ketterien menetelmien testauskäytänteet

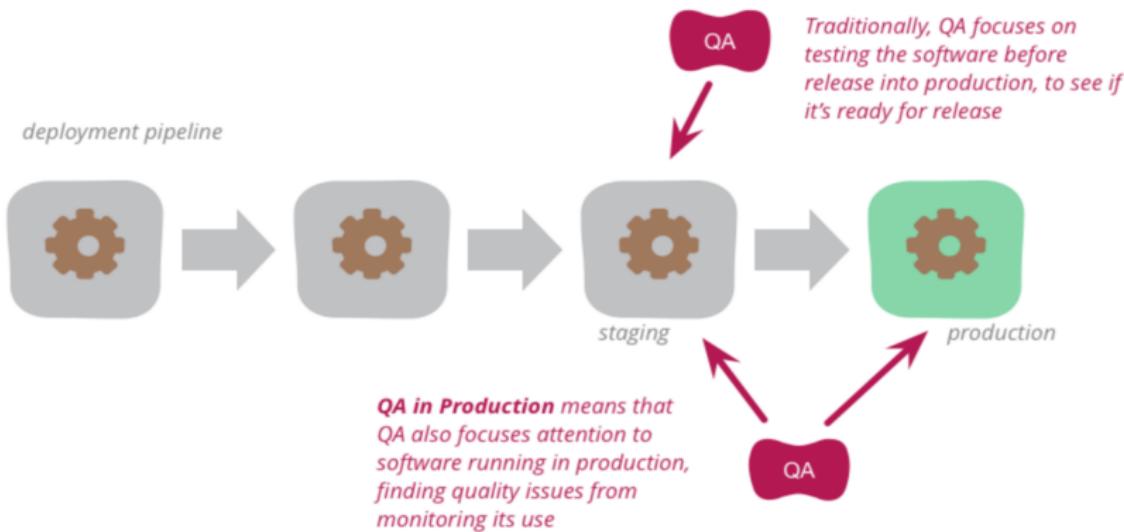
- ▶ Test driven development (TDD)
 - ▶ Nimistä huolimatta kyseessä toteutus- ja suunnitteluteknikka
 - ▶ Sivutuotteena paljon automaattisesti suoritettavia testejä
- ▶ Acceptance Test Driven Development / Behavior Driven Development
 - ▶ User storyjen tasolla tapahtuva automatisoitu testaus
- ▶ Continuous Integration (CI) eli jatkuva integraatio
 - ▶ Perinteisen integraatio- ja integraatiotestausvaiheen korvaava työskentelytapa
- ▶ Exploratory testing, suomeksi tutkiva testaus
 - ▶ Järjestelmätestauksen tekniikka, jossa testaaminen tapahtuu ilman formaalia testaussuunnitelmaa
- ▶ Tuotannossa tapahtuva testaus

Tuotannossa tapahtuva testaaminen ja laadunhallinta

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

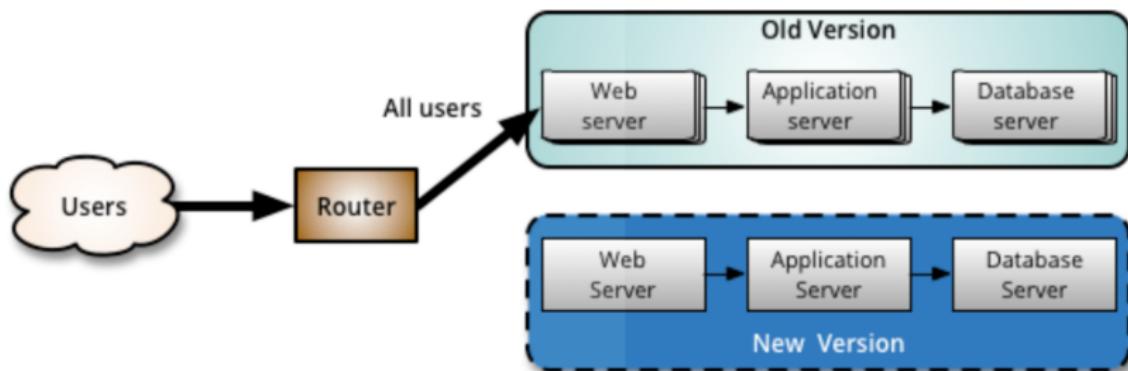
Tuotannossa tapahtuva testaaminen ja laadunhallinta

- ▶ Perinteisesti ajateltu: kaikki laadunhallintaan tehdään ennen kuin ohjelmisto / uudet toiminnallisuudet otetaan käyttöön
- ▶ Viime aikainen trendi on tehdä osa laadunhallinnasta monitoroimalla tuotannossa olevaa ohjelmistoa



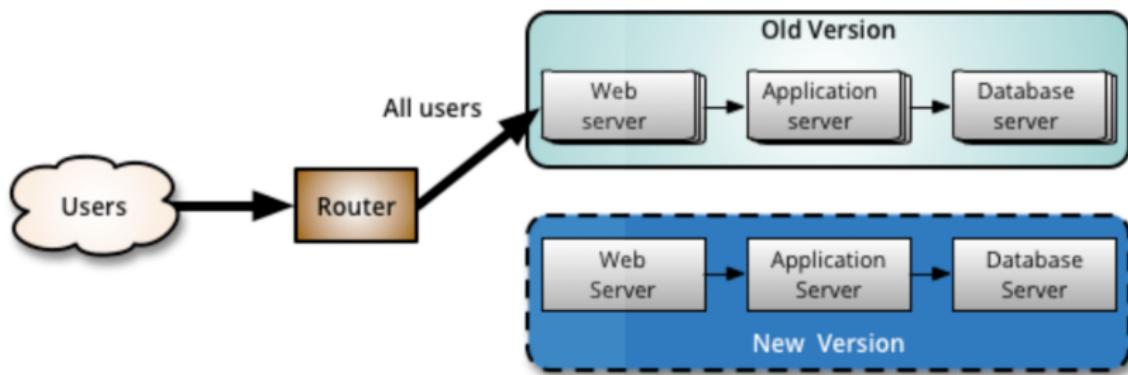
Blue-green-deployment

- ▶ Kaksi rinnakkaista tuotantoympäristöä: *blue* ja *green*
- ▶ Vain toinen on ohjelmiston käyttäjien aktiivisessa käytössä
 - ▶ edustopalvelin ohjaa käyttäjien liikenteen aktiiviseen ympäristöön



Blue-green-deployment

- ▶ Kaksi rinnakkaisista tuotantoympäristöä: *blue* ja *green*
- ▶ Vain toinen on ohjelmiston käyttäijien aktiivisessa käytössä
 - ▶ edustopalvelin ohjaa käyttäijien liikenteen aktiiviseen ympäristöön



- ▶ Uusi ominaisuus deployataan ensin passiiviseen ympäristöön
- ▶ ja sitä testataan
 - ▶ osa liikenteestä ohjataan aktiivisen lisäksi passiiviseen ympäristöön ja varmistetaan, että toiminta odotettua

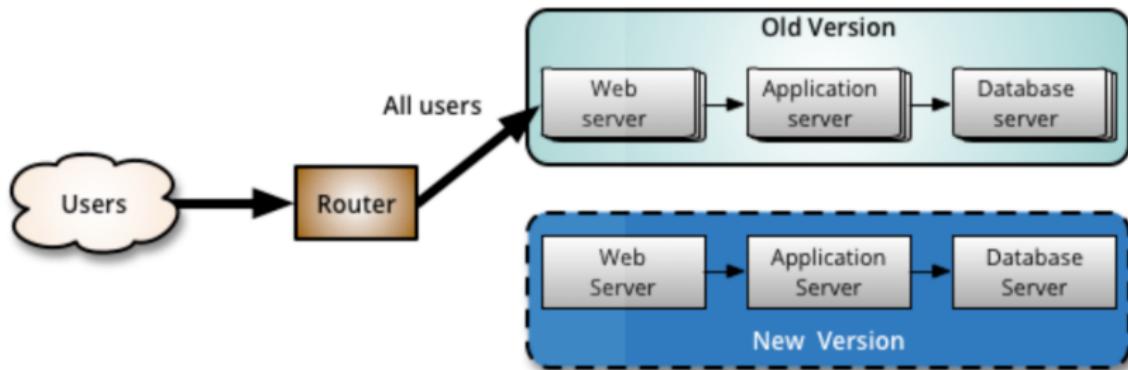
- ▶ Kun uuden ominaisuuden todetaan toimivan, vaihdetaan palvelinten rooli
 - ▶ määritellään edustapalvelin ohjaamaan liikenne uudelle palvelimelle

- ▶ Kun uuden ominaisuuden todetaan toimivan, vaihdetaan palvelinten rooli
 - ▶ määritellään edustopalvelin ohjaamaan liikenne uudelle palvelimelle
- ▶ Jos vaihdon jälkeen havaitaan ongelmia, tehdään *rollback*
 - ▶ vanha versio jälleen aktiiviseksi

- ▶ Kun uuden ominaisuuden todetaan toimivan, vaihdetaan palvelinten rooli
 - ▶ määritellään edustopalvelin ohjaamaan liikenne uudelle palvelimelle
- ▶ Jos vaihdon jälkeen havaitaan ongelmia, tehdään *rollback*
 - ▶ vanha versio jälleen aktiiviseksi
- ▶ testit, tulosten varmistaminen, tuotantoymäristön vaihto ja mahdollinen rollback *tulee automatisoida*

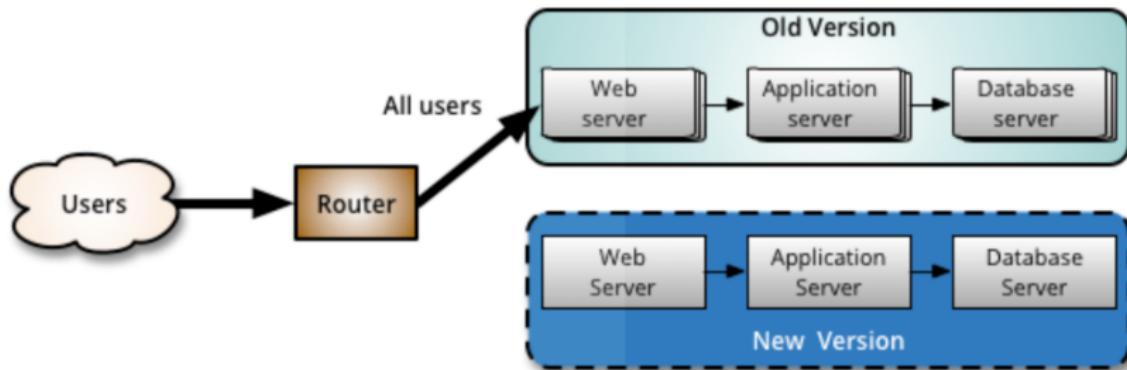
Canary release

- ▶ Canary-releasessa uuden ominaisuuden sisältävään ympäristöön ohjataan osa järjestelmän käyttäjistä



Canary release

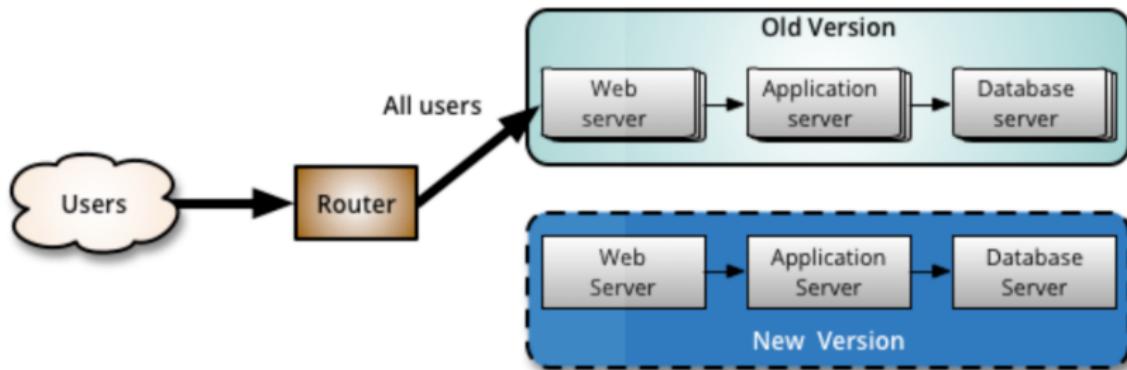
- ▶ Canary-releasessa uuden ominaisuuden sisältävään ympäristöön ohjataan osa järjestelmän käyttäjistä



- ▶ Uuden ominaisuuden sisältämää versiota *monitoroidaan*
 - ▶ jos ei ongelmia ohjataan kaikki liikenne uuteen versioon

Canary release

- ▶ Canary-releasessa uuden ominaisuuden sisältävään ympäristöön ohjataan osa järjestelmän käyttäjistä



- ▶ Uuden ominaisuuden sisältämää versiota *monitoroidaan*
 - ▶ jos ei ongelmia ohjataan kaikki liikenne uuteen versioon
- ▶ Ongelmatilanteissa palautetaan käyttäjät aiempaan, toimivaksi todettuun versioon

- ▶ Uuden version toimivaksi varmistaminen perustuu järjestelmän *monitorointiin*

- ▶ Uuden version toimivaksi varmistaminen perustuu järjestelmän *monitorointiin*
- ▶ Esim. sosiaalisen median palvelussa
 - ▶ palvelun muistin ja prosessoriajan kulutusta
 - ▶ verkkoliikenteen määrää
 - ▶ sovelluksen eri sivujen vasteaikoja
 - ▶ kirjautuneiden käyttäjien määrää
 - ▶ luettujen ja lähetettyjen viestien märiä per käyttäjä
 - ▶ kirjautuneen käyttäjän sovelluksessa viettämää aikaa

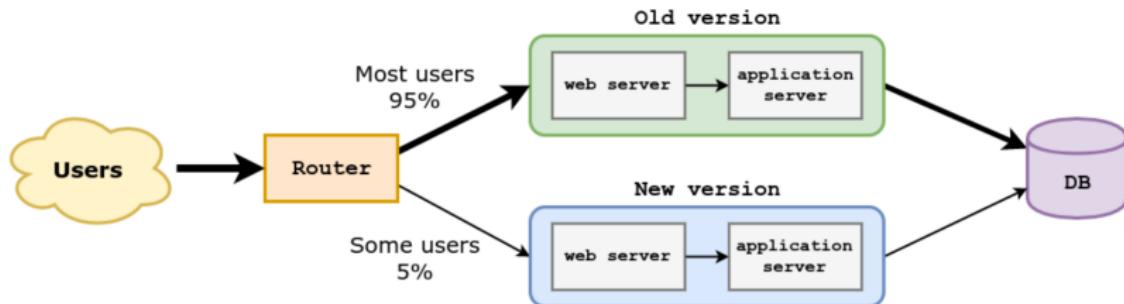
- ▶ Uuden version toimivaksi varmistaminen perustuu järjestelmän *monitorointiin*
- ▶ Esim. sosiaalisen median palvelussa
 - ▶ palvelun muistin ja prosessoriajan kulutusta
 - ▶ verkkoliikenteen määrää
 - ▶ sovelluksen eri sivujen vasteaikoja
 - ▶ kirjautuneiden käyttäjien määrää
 - ▶ luettujen ja lähetettyjen viestien märiä per käyttäjä
 - ▶ kirjautuneen käyttäjän sovelluksessa viettämää aikaa
- ▶ Monitoroidaan palvelimen yleisen toimivuuden lisäksi *käyttäjätason metriikoita* (engl. business level metrics)

- ▶ Uuden version toimivaksi varmistaminen perustuu järjestelmän *monitorointiin*
- ▶ Esim. sosiaalisen median palvelussa
 - ▶ palvelun muistin ja prosessoriajan kulutusta
 - ▶ verkkoliikenteen määrää
 - ▶ sovelluksen eri sivujen vasteaikojen
 - ▶ kirjautuneiden käyttäjien määrää
 - ▶ luettujen ja lähetetyjen viestien määrä per käyttäjä
 - ▶ kirjautuneen käyttäjän sovelluksessa viettämää aikaa
- ▶ Monitoroidaan palvelimen yleisen toimivuuden lisäksi *käyttäjätason metriikoita* (engl. business level metrics)
- ▶ Jos suuria eroja aiempaan, tehdään rollback aiempaan versioon
 - ▶ esim. kirjautuneet käyttäjät eivät lähetä viestejä samaa määrää kuin keskimäärin normaalista

- ▶ Uuden version toimivaksi varmistaminen perustuu järjestelmän *monitorointiin*
- ▶ Esim. sosiaalisen median palvelussa
 - ▶ palvelun muistin ja prosessoriajan kulutusta
 - ▶ verkkoliikenteen määrää
 - ▶ sovelluksen eri sivujen vasteaikoja
 - ▶ kirjautuneiden käyttäjien määrää
 - ▶ luettujen ja lähetetyjen viestien märiä per käyttäjä
 - ▶ kirjautuneen käyttäjän sovelluksessa viettämää aikaa
- ▶ Monitoroidaan palvelimen yleisen toimivuuden lisäksi *käyttäjätason metriikoita* (engl. business level metrics)
- ▶ Jos suuria eroja aiempaan, tehdään rollback aiempaan versioon
 - ▶ esim. kirjautuneet käyttäjät eivät lähetä viestejä samaa määrää kuin keskimäärin normaalisti
- ▶ Testauksen ja kaikkien tuotantoon vientiin liittyvän on syytä tapahtua automatisoidusti

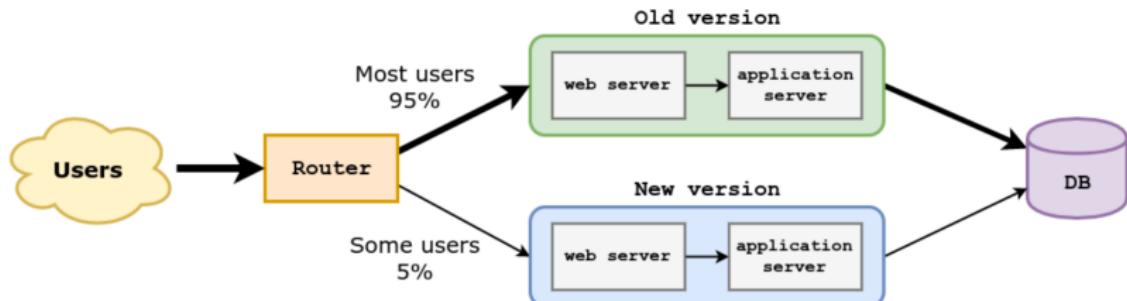
Tuotannossa testaaminen ja tietokanta

- ▶ Erityisesti canary releasejen yhteydessä järjestelmän molemmat versiot käyttävät yleensä samaa tietokantaa



Tuotannossa testaaminen ja tietokanta

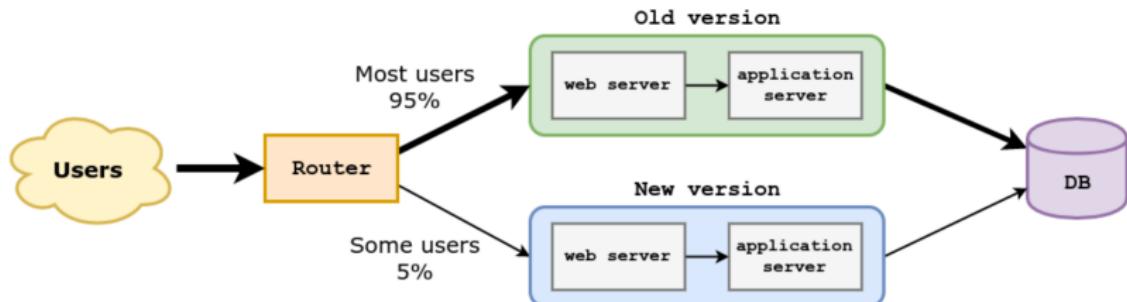
- ▶ Erityisesti canary releasejen yhteydessä järjestelmän molemmat versiot käyttävät yleensä samaa tietokantaa



- ▶ Asettaa haasteita, jos järjestelmään toteutetut uudet ominaisuudet edellyttävät muutoksia tietokannan skeemaan
 - ▶ Tarvitaan yhtä aikaa sekä uutta että vanhaa versiota kannasta

Tuotannossa testaaminen ja tietokanta

- ▶ Erityisesti canary releasejen yhteydessä järjestelmän molemmat versiot käyttävät yleensä samaa tietokantaa



- ▶ Asettaa haasteita, jos järjestelmään toteutetut uudet ominaisuudet edellyttävät muutoksia tietokannan skeemaan
 - ▶ Tarvitaan yhtä aikaa sekä uutta että vanhaa versiota kannasta
- ▶ Jos järjestelmän versioilla käytössä eri tietokannat, täytyy kantojen tila synkronoida, jotta järjestelmien vaihtaminen onnistuu saumattomasti

Feature toggle

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

Feature toggle

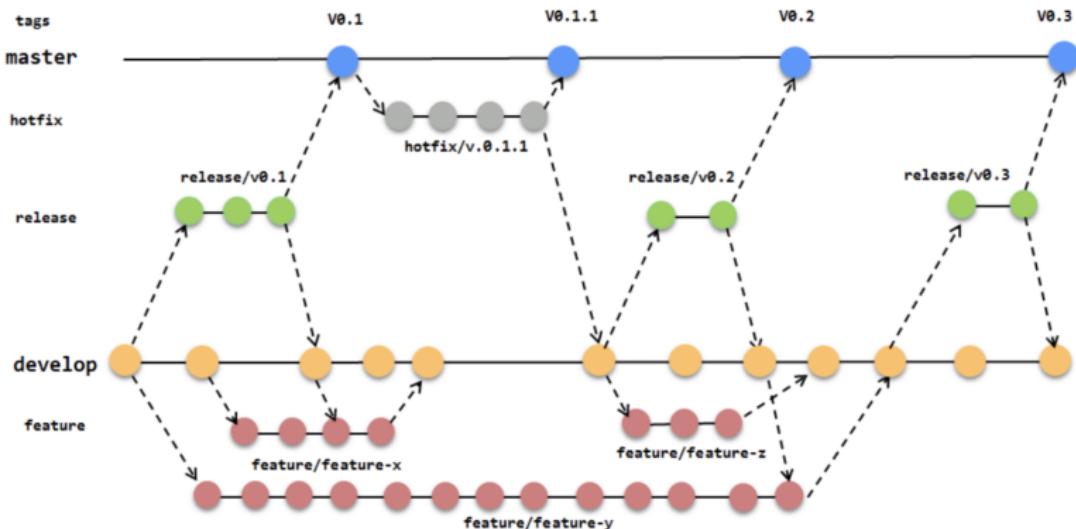
- ▶ *Feature togglejen* avulla voidaan canary releaset toteuttaa käyttämällä yhtä tuotantopalvelinta
- ▶ Koodiin laitetaan *ehtolauseita*, joiden avulla osa liikenteestä ohjataan vanhan toteutuksen sijaan testauksen alla olevaan toteutukseen

```
List<News> recommendedNews(User user) {  
    if ( isIn CanaryRelease(user) ) {  
        return experimentalRecommendationAlgorithm(user)  
    } else {  
        return recommendationAlgorithm(user)  
    }  
}
```

- ▶ Esim. some-palvelussa feature toggle: osalle käytetään näytetään uuden algoritmin perusteella generoitu lista uutisia

Vaihtoehto feature brancheille

- ▶ Feature toggleja käytetään myös eliminoimaan tarve pitkäikäisille feature brancheille



Vaihtoehto feature brancheille

- ▶ Ei erillistä versionhallinnan haaraa uuden ominaisuuden toteuttamiseen, koodataan suoraan päähaaraan
 - ▶ piilotetaan uusi ominaisuus käyttäjiltä feature toggleilla

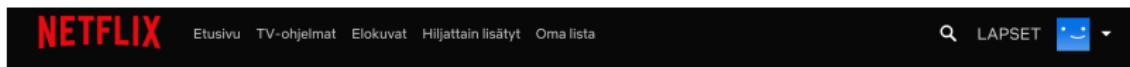
Vaihtoehto feature brancheille

- ▶ Ei erillistä versionhallinnan haaraa uuden ominaisuuden toteuttamiseen, koodataan suoraan päähaaraan
 - ▶ piilotetaan uusi ominaisuus käyttäjiltä feature toggleilla
- ▶ Feature toggle palauttaa vanhan version normaaleille käyttäjille
 - ▶ kehittäjien mahdollista valita kumman version toggle palauttaa

Vaihtoehto feature brancheille

- ▶ Ei erillistä versionhallinnan haaraa uuden ominaisuuden toteuttamiseen, kodataan suoraan päähaaraan
 - ▶ piilotetaan uusi ominaisuus käyttäjiltä feature toggleilla
- ▶ Feature toggle palauttaa vanhan version normaaleille käyttäjille
 - ▶ kehittäjien mahdollista valita kumman version toggle palauttaa
- ▶ Kun valmiina laajempaan testiin, julkaistaan esim.
 - ▶ ensin kehittäjärytyksen omaan käyttöön
 - ▶ ja lopulta osalle käyttäjistä canary releasena
- ▶ Lopulta feature toggle ja vanha toteutus voidaan poistaa

- ▶ Suuret internetpalvelut kuten Facebook, Netflix, Google ja Flickr soveltavat laajalti canary releaseihin ja feature flageihin perustuvaa kehitysmallia



Osallistuminen testeihin

Osallistun testeihin ja esikatseluihin
Poista valinta siirtyäksesi normaaliin käyttäjäkokemukseen.

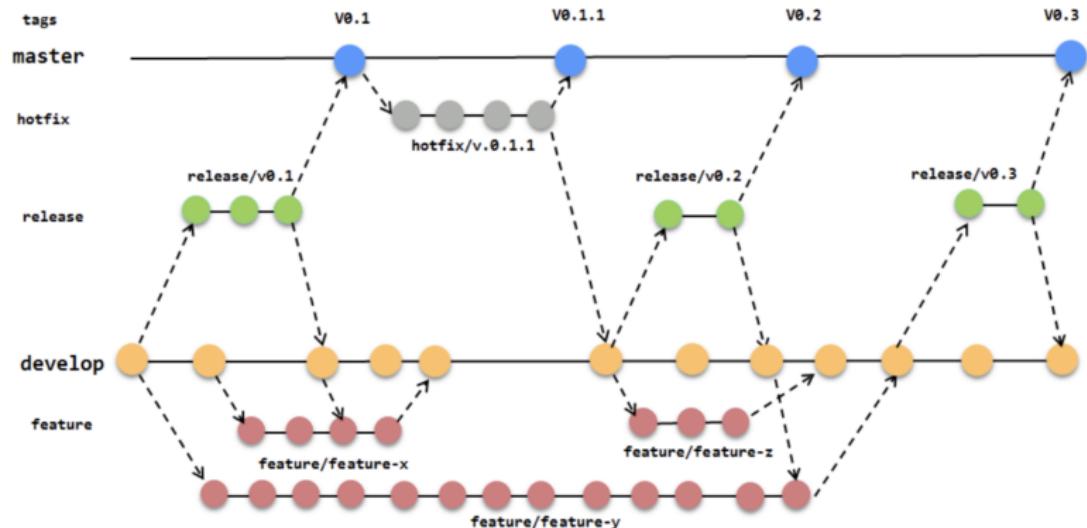
KÄYTÖSSÄ

Osallistu testeihin, joilla kehitetään Netflix-käyttäjäkokemusta. Saat nähdäksesi mahdollisia uutuuksia ennen muita.

Valmis

Feature branchit

- ▶ Uusi ominaisuus, esim. user story toteutetaan ensin omaan versionhallinnan haaraansa
 - ▶ ja ominaisuuden valmistuttua haara mergetään päähaaraan (masteriin)



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*
- ▶ Arkipäivää ohjelmistotuimissä



[REDACTED] 12:55 PM

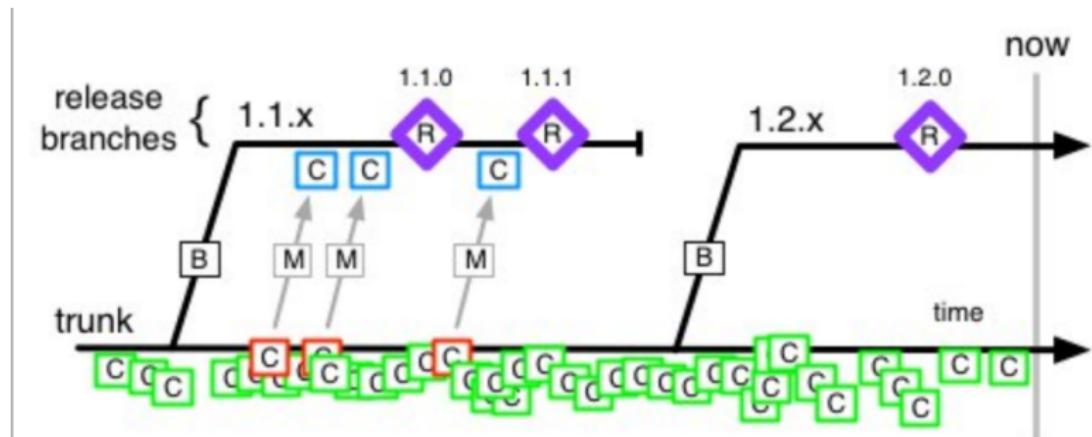
yritän huomenna mergee ton mun fixStatusCode branchin trunkkiin. se on
sen verran hajalla nyt et pakko fixailla

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

- ▶ Pääkehityshaara voi olla master tai joku erillinen branch
- ▶ Ohjelmiston kustakin julkaistusta versiosta saatetaan tehdä oma *release branch*

Trunk based development

- ▶ Pääkehityshaara voi olla master tai joku erillinen branch
- ▶ Ohjelmiston kustakin julkaistusta versiosta saatetaan tehdä oma *release branch*
- ▶ Pakottaa sovelluskehittäjät tekemään pieniä, nopeasti päähaaraan mergettäviä muutoksia

Trunk based development

- ▶ Pääkehityshaara voi olla master tai joku erillinen branch
- ▶ Ohjelmiston kustakin julkaistusta versiosta saatetaan tehdä oma *release branch*
- ▶ Pakottaa sovelluskehittäjät tekemään pieniä, nopeasti päähaaraan mergettäviä muutoksia
- ▶ Käytetään feature toggleja
 - ▶ puolivalmiitakin ominaisuuksia voidaan helposti ohjelmoida päähaaraan ilman toiminnallisuuden rikkomista

Trunk based development

- ▶ Pääkehityshaara voi olla master tai joku erillinen branch
- ▶ Ohjelmiston kustakin julkaistusta versiosta saatetaan tehdä oma *release branch*
- ▶ 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

- ▶ Pääkehityshaara voi olla master tai joku erillinen branch
- ▶ Ohjelmiston kustakin julkaistusta versiosta saatetaan tehdä oma *release branch*
- ▶ 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 toimitusvalmius ja käyttöönotto (CD) sekä tuotannossa testaaminen on haastavaa

Dev vs Ops

- ▶ Jatkuva toimitusvalmius ja käyttöönotto (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 toimitusvalmius ja käyttöönotto (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 toimitusvalmius ja käyttöönotto (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 kiittivät DevOpsiin:
 - ▶ automatisoitu testaus
 - ▶ continuous deployment
 - ▶ 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

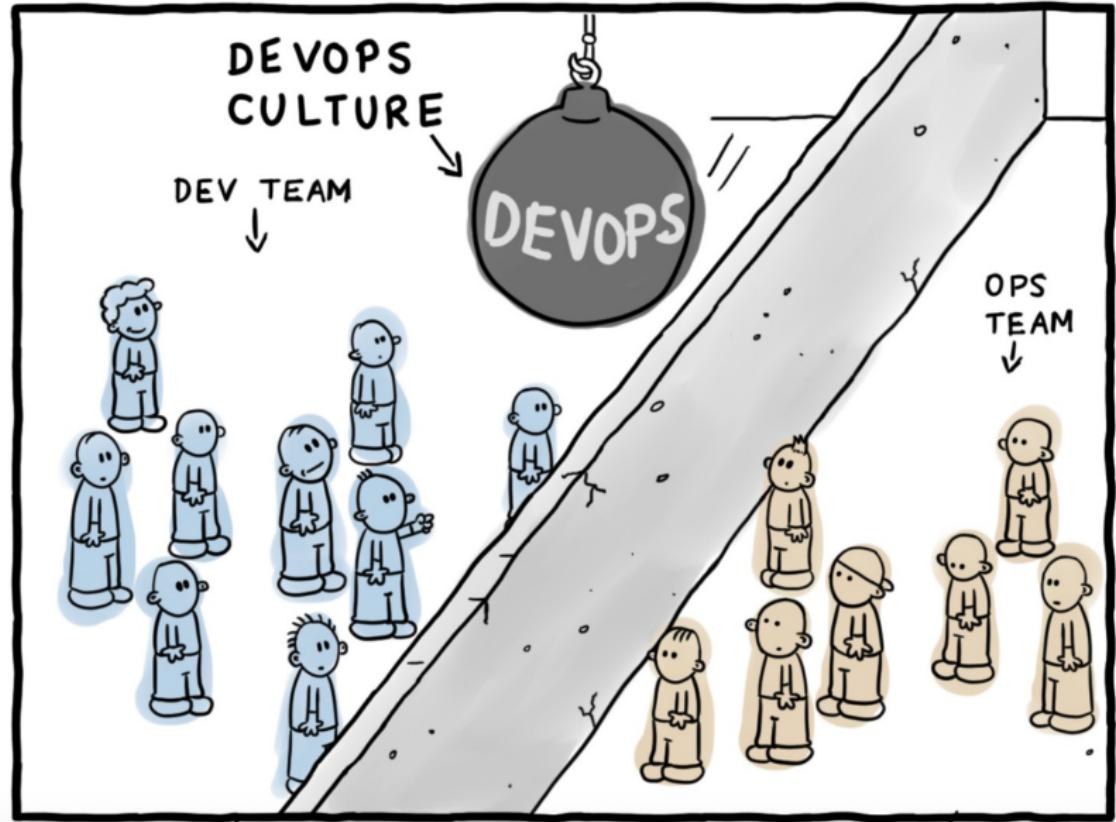
- ▶ Monet listatuista kehittyneet viimeisen 5-10 vuoden aikana ja mahdollistaneet DevOpsin helpomman soveltamisen
- ▶ Eräs tärkeimmistä DevOpsia mahdollistavista asioista *infrastructure as code*
 - ▶ fyysisen palvelinten sijaan virtuaalisia ja pilvessä toimivia palvelimia, joita voi konfiguroida ohjelmallisesti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

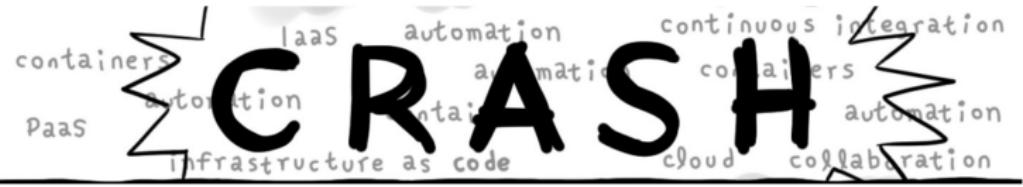
- ▶ Scrumin ja agilen eräs tärkeimmistä periaatteista on tehdä kehitystiimeistä itseorganisoituvia ja “cross functional”

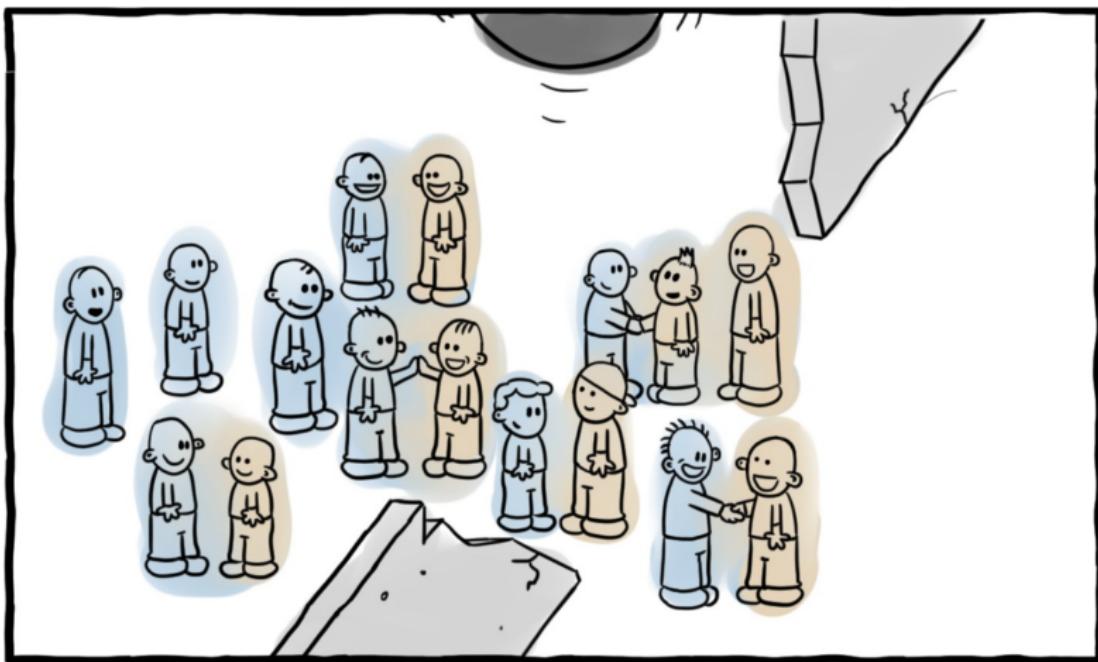
- ▶ Scrumin ja agilen eräs tärkeimmistä periaatteista on tehdä kehitystiimeistä itseorganisoituvia ja “cross functional”
- ▶ DevOps on keino viedä ketteryyttä askeleen pitemmälle
- ▶ Mahdollistaa että kehitystiimi pystyy viemään vaivattomasti uudet toiminnallisuudet tuotantoalustaan ja jopa testaamaan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

- ▶ Scrumin ja agilen 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 tuotantoymäristöön ja jopa testaamaan sekä operoimaan niitä tuotannossa
- ▶ DevOps siis laajentaa ketteryyden koskemaan myös järjestelmälläpitoa

- ▶ Scrumin ja agilen 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 tuotantoymäristöön ja jopa testaamaan sekä operoimaan niitä tuotannossa
- ▶ DevOps siis laajentaa ketteryyden koskemaan myös järjestelmälläpitoa
- ▶ DevOps-ajattelutapa asettaa sovelluskehittäjille lisää osaamisvaatimuksia
 - ▶ kehittäjien pitää hallita enenevissä määrin ylläpitoasioita

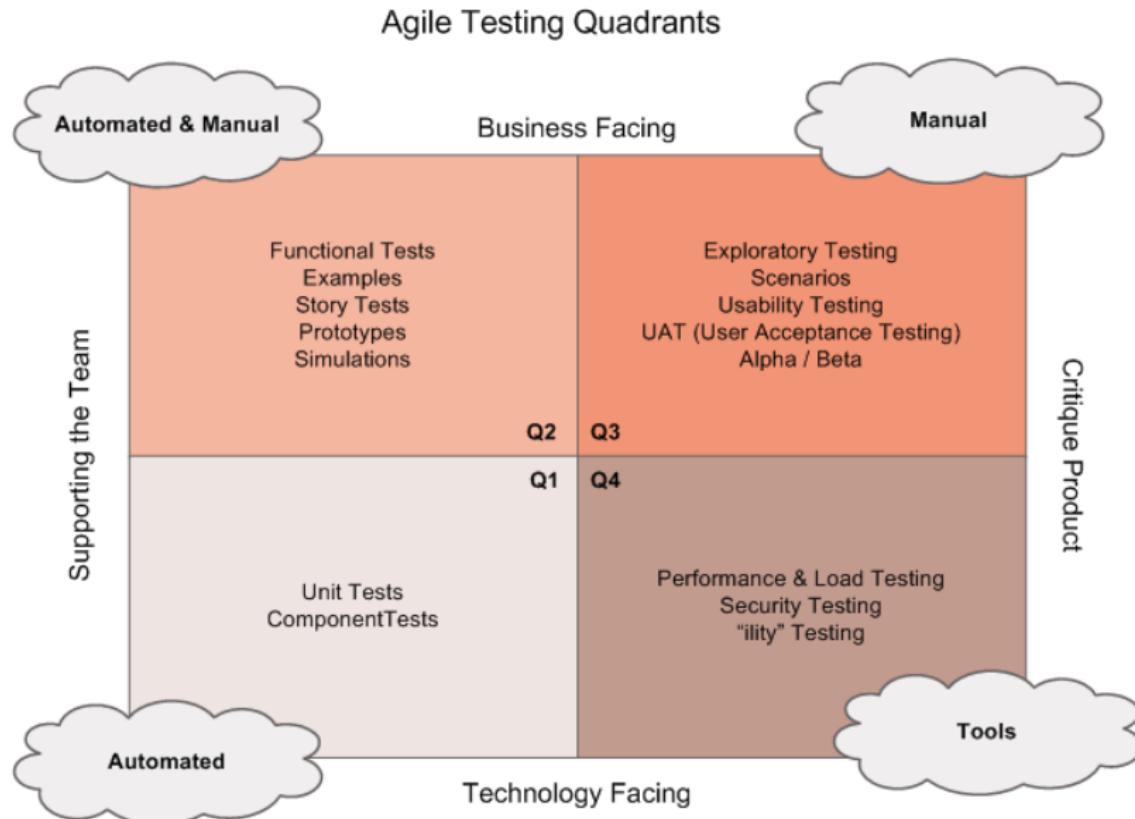






DANIEL STORI {TURNOFF.US}

Yhteenvetö - ketterän testauksen nelikettä



Yhteenvetö - ketterän testauksen nelikettä

- ▶ business facing vs. technology facing
 - ▶ testataanko käyttäjän kokemaa toiminnallisuutta vai ohjelmiston teknisiä ominaisuuksia
- ▶ supporting team vs. critique to the product
 - ▶ onko testaus sovelluskehittäjien tukena vai ulkoista laatua varmistamassa

Yhteenvetö - ketterän testauksen nelikettä

- ▶ business facing vs. technology facing
 - ▶ testataanko käyttäjän kokemaa toiminnallisuutta vai ohjelmiston teknisiä ominaisuuksia
- ▶ supporting team vs. critique to the product
 - ▶ onko testaus sovelluskehittäjien tukena vai ulkoista laatua varmistamassa
- ▶ Eri tyypiset testit suurelta osin automatisoitavissa
 - ▶ poikkeuksena *tutkiva testaaminen ja käyttäjän hyväksymätestaus* edellyttää manialista työtä

Yhteenvetö - ketterän testauksen nelikettä

- ▶ business facing vs. technology facing
 - ▶ testataanko käyttäjän kokemaa toiminnallisuutta vai ohjelmiston teknisiä ominaisuuksia
- ▶ supporting team vs. critique to the product
 - ▶ onko testaus sovelluskehittäjien tukena vai ulkoista laatua varmistamassa
- ▶ Eri tyypiset testit suurelta osin automatisoitavissa
 - ▶ poikkeuksena *tutkiva testaaminen ja käyttäjän hyväksymätestaus* edellyttää manialista työtä
- ▶ Kaikilla neljänneksillä on oma roolinsa
 - ▶ tilanteesta riippuu missä suhteessa laadunhallinnan resurssit kannattaa kuhunkin neljännekseen kohdentaa

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...
- ▶ Kokonaan uutta ohjelmistoa tai komponenttia tehtäessä 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...
- ▶ Kokonaan uutta ohjelmistoa tai komponenttia tehtäessä 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 hyödyllisimpia
 - ▶ 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

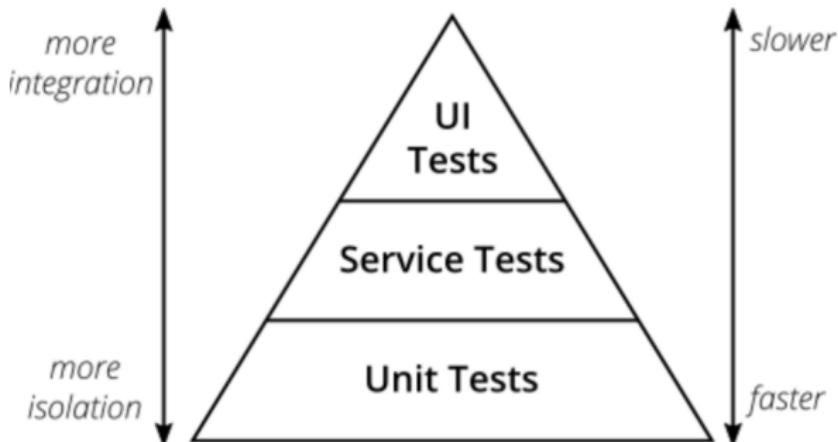
- ▶ Ehdottomasti kaikkein tärkein laadunhallinnan kannalta on mahdollisimman usein tapahtuva tuotantoonvienti
 - ▶ edellyttää hyvin rakennettua deployment pipelineä, kohtuullista testauksen automatisointia

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

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

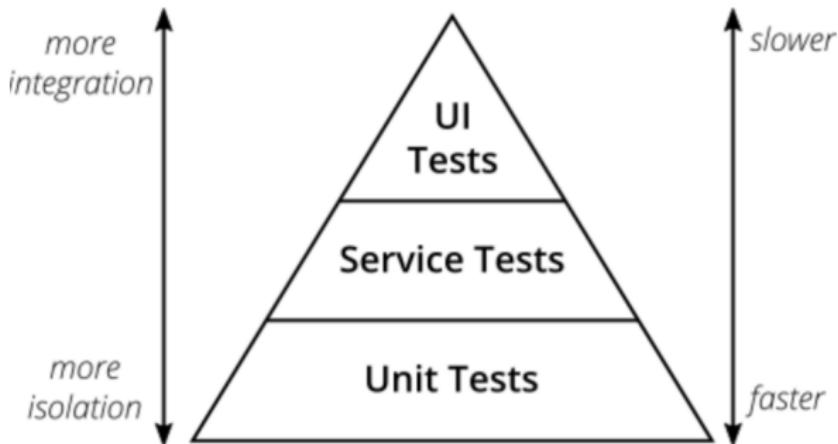
Testauspyramiidi

- ▶ Oma näkemykseni poikkeaa jossain ns *testauspyramidista*



Testauspyramiidi

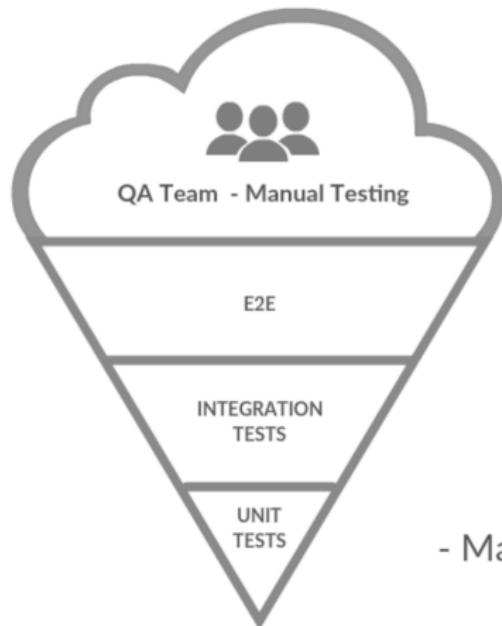
- ▶ Oma näkemykseni poikkeaa jossain ns *testauspyramidista*



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

Testausjäätelö

- ▶ Onkin oltava varuillaan että ei synnytetä *testausjäätelöä*

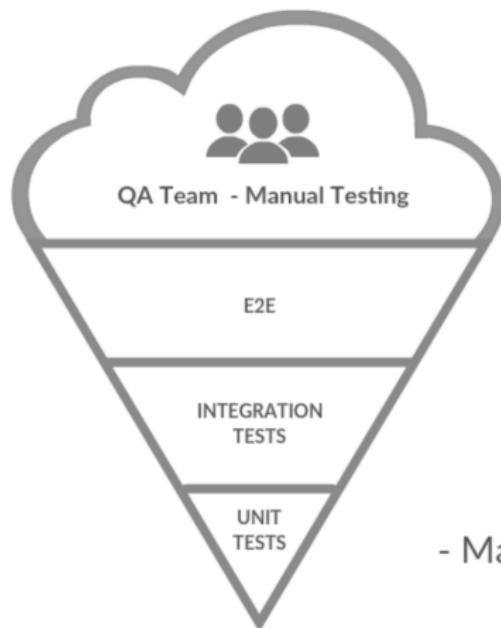


ICE CREAM CONE

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

Testausjäätelö

- ▶ Onkin 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

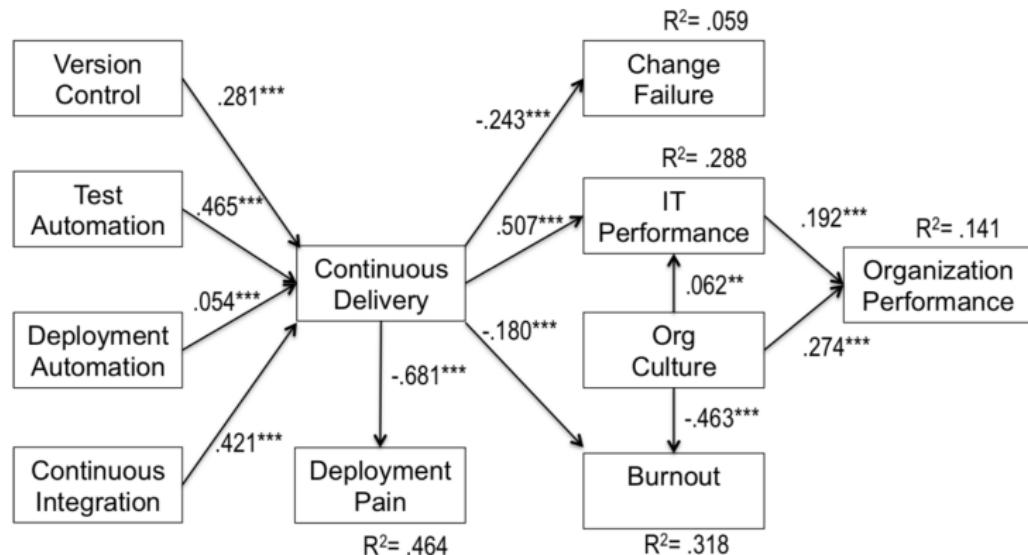
- ▶ Edellä esitellyistä julkaisun ja laadunhallinnan käytenteiden toimivuudesta on paljon runsaasti anekdotaalista evidenssiä

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* 2017

Tieteellinen evidenssi

- ▶ Edellä esitellyistä julkaisun ja laadunhallinnan käytenteiden toimivuudesta on paljon runsaasti anekdotaalista evidenssiä
- ▶ Accelerate: *The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations* 2017



The Science of Lean Software and DevOps

- ▶ Tutkimus tehty 2013-2017, perustuu yli 20000 tuhanteen kyselytutkimuksen vastaukseen
- ▶ Julkaistu kirjan lisäksi vertaisarvioituina tieteellisänä julkaisuinta

The Science of Lean Software and DevOps

- ▶ Tutkimus tehty 2013-2017, perustuu yli 20000 tuhanteen kyselytutkimuksen vastaukseen
- ▶ Julkaistu kirjan lisäksi vertaisarvioituina tieteellisänä julkaisuinta

Organizational Performance

Select the number that best indicates degree of conformance to your organization's goals over the last year (1 = Below expectations, 7 = Performed well above)

Overall organizational performance

Overall organizational profitability

Relative market share for primary products

Overall productivity of the delivery system

Increased number of customers

Time to market

Quality and performance of applications