

# Ohjelmistotuotanto

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

syksy 2019

Luento 7

18.11.2019

# Miniprojektien aloitustilaisuudet

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

# 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
- ▶ Ideaalitilanteessa testaajia sijoitettu kehittäjätiimiin, myös ohjelmoijat kirjoittavat testejä
  - ▶ tiimit *cross functional*

# Ketterien menetelmien testauskäytänteet

- ▶ Test driven development (TDD)
  - ▶ Nimestä huolimatta kyseessä toteutus- ja suunnittelutekniikka
  - ▶ 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)
  - ▶ Nimestä huolimatta kyseessä toteutus- ja suunnittelutekniikka
  - ▶ 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

# Testaaminen tuotannossa

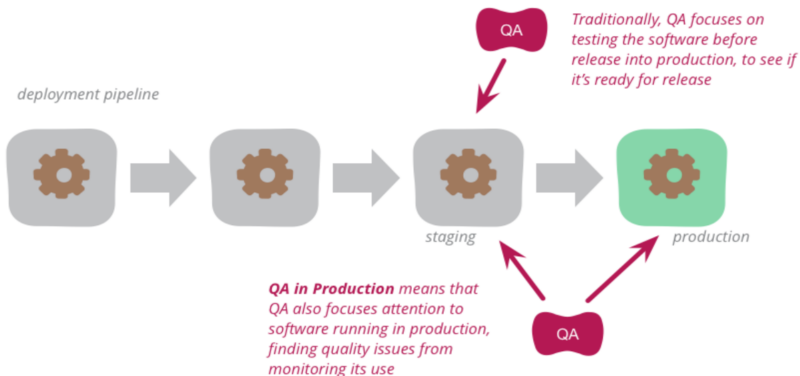


# Tuotannossa tapahtuva testaaminen ja laadunhallinta

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

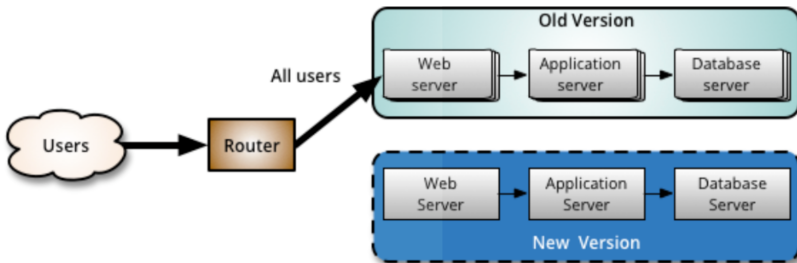
# Tuotannossa tapahtuva testaaminen ja laadunhallinta

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



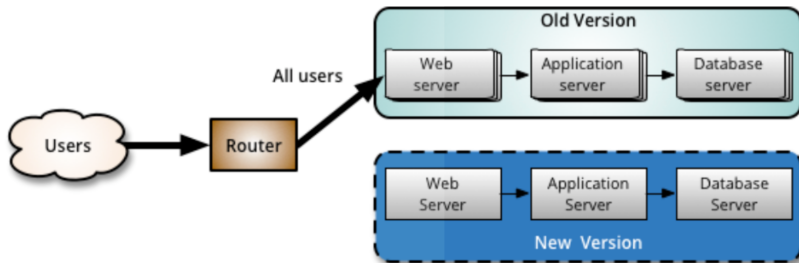
# Blue-green-deployment

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



# Blue-green-deployment

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



- ▶ Uusi ominaisuus deployataan ensin passiiviseen ympäristöön
- ▶ ja sitä testataan
  - ▶ esim. osa käyttäjien liikenteestä ohjata aktiivisen lisäksi passiiviseen ympäristöön ja varmistaa, että se toimii odotetulla tavalla

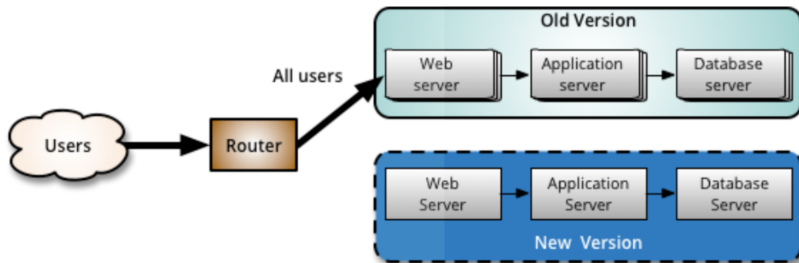
- ▶ Kun uuden ominaisuuden todetaan toimivan, vaihdetaan palvelinten rooli
  - ▶ määritellään edustapalvelin ohjaamaan liikenne uudelle palvelimelle

- ▶ Kun uuden ominaisuuden todetaan toimivan, vaihdetaan palvelinten rooli
  - ▶ määritellään edustapalvelin ohjaamaan liikenne uudelle palvelimelle
- ▶ Jos ympäristön vaihdon jälkeen havaitaan ongelmia, on mahdollista suorittaa nopea rollback-operaatio: vanha versio jälleen aktiiviseksi

- ▶ Kun uuden ominaisuuden todetaan toimivan, vaihdetaan palvelinten rooli
  - ▶ määritellään edustapalvelin ohjaamaan liikenne uudelle palvelimelle
- ▶ Jos ympäristön vaihdon jälkeen havaitaan ongelmia, on mahdollista suorittaa nopea rollback-operaatio: vanha versio jälleen aktiiviseksi
- ▶ blue-green-deploymentiin liittyvät testit, tulosten varmistaminen, tuotantoympäristön vaihto ja mahdollinen rollback *tulee automatisoida*

# Canary release

- *canary-releasessa* uuden ominaisuuden sisältävään ympäristöön ohjataan osa järjestelmän käyttäjistä



- Uuden ominaisuuden sisältämää versiota monitoroidaan aktiivisesti ja jos ongelmia ei ilmene, vähitellen kaikki liikenne ohjataan uuteen versioon
- Ongelmatilanteissa palautetaan käyttäjät aiempaan, toimivaksi todettuun versioon



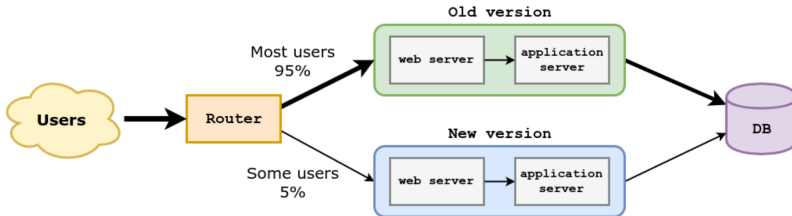
- ▶ Uuden version toimivaksi varmistaminen perustuu järjestelmän *monitorointiin*
- ▶ Esim. sosiaalisen median palvelussa
  - ▶ Palvelun muistin ja prosessoriajan kulutusta sekä verkkoliikenteen määrää
  - ▶ Sovelluksen eri sivujen vasteaikoja eli latautumiseen menevää aikaa
  - ▶ Kirjautuneiden käyttäjien määrää
  - ▶ Luettujen ja lähetettyjen viestien määriä per käyttäjä
  - ▶ Kirjautuneen käyttäjän sovelluksessa viettämää aikaa

- ▶ Uuden version toimivaksi varmistaminen perustuu järjestelmän *monitorointiin*
- ▶ Esim. sosiaalisen median palvelussa
  - ▶ Palvelun muistin ja prosessoriajan kulutusta sekä verkkoliikenteen määrää
  - ▶ Sovelluksen eri sivujen vasteaikoja eli latautumiseen menevää aikaa
  - ▶ Kirjautuneiden käyttäjien määrää
  - ▶ Luettujen ja lähetettyjen viestien määriä per käyttäjä
  - ▶ Kirjautuneen käyttäjän sovelluksessa viettämää aikaa
- ▶ Monitoroidaan palvelimen yleisen toimivuuden lisäksi seurata *käyttäjätason metriikoita* (engl. bussiness level metrics)

- ▶ Uuden version toimivaksi varmistaminen perustuu järjestelmän *monitorointiin*
- ▶ Esim. sosiaalisen median palvelussa
  - ▶ Palvelun muistin ja prosessoriajan kulutusta sekä verkkoliikenteen määrää
  - ▶ Sovelluksen eri sivujen vasteaikoja eli latautumiseen menevää aikaa
  - ▶ Kirjautuneiden käyttäjien määrää
  - ▶ Luettujen ja lähetettyjen viestien määriä per käyttäjä
  - ▶ Kirjautuneen käyttäjän sovelluksessa viettämää aikaa
- ▶ Monitoroidaan palvelimen yleisen toimivuuden lisäksi seurata *käyttäjätason metriikoita* (engl. bussiness 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

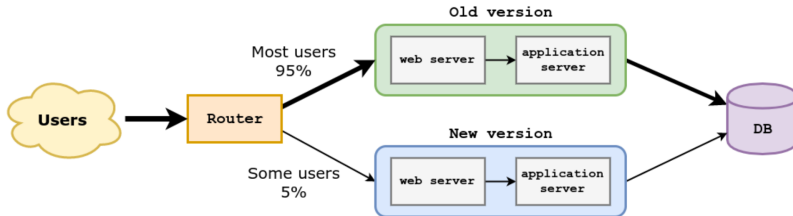
- ▶ Erityisesti canary releasejen yhteydessä järjestelmän molemmat versiot käyttävät yleensä samaa tietokantaa



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

# 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 tietokannan uutta että vanhaa versiota
- ▶ Jos versiot käyttävät eri tietokantaa, 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 ( isInCanaryRelease(user) ) {  
        return experimentalRecommendationAlgorithm(user)  
    } else {  
        return recommendationAlgorithm(user)  
    }  
}
```

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

- ▶ Feature toggleja käytetään myös eliminoimaan tarve pitkäikäisille feature brancheille




- ▶ Feature toggleja käytetään myös eliminoimaan tarve pitkäikäisille feature brancheille
- ▶ Ei erillistä versionhallinnan haaraa uuden ominaisuuden toteuttamiseen, koodataan suoraan päähaaraan
  - ▶ piilotetaan uusi ominaisuus käyttäjiltä feature toggleilla

- ▶ Feature toggleja käytetään myös eliminoimaan tarve pitkäikäisille 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

- ▶ Feature toggleja käytetään myös eliminoimaan tarve pitkäikäisille 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
- ▶ Kun valmiina laajempaan testattavaksi, julkaistaan esim. ensin kehittäjäyrityksen 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


**NETFLIX** Etusivu TV-ohjelmat Elokuvat Hiljattain lisätyt Oma lista

Q LAPSET 

## 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 ja merge hell

- ▶ Edellisellä kalvolla mainittiin feature branchit
- ▶ Uusi ominaisuus, esim. user story toteutetaan ensin omaan versionhallinnan haaraan
  - ▶ ja ominaisuuden valmistuttua haara mergetään päähaaraan (masteriin)

# Feature branchit ja merge hell

- ▶ Edellisellä kalvolla mainittiin feature branchit
- ▶ Uusi ominaisuus, esim. user story toteutetaan ensin omaan versionhallinnan haaraansa
  - ▶ ja ominaisuuden valmistuttua haara mergetään päähaaraan (masteriin)
- ▶ Monet pitävät feature brancheja versionhallinnan best practicena
- ▶ Viime aikoina huomattu, että feature branchit aiheuttavat helposti pahoja merge-konflikteja sprintin lopussa

# Feature branchit ja merge hell

- ▶ Edellisellä kalvolla mainittiin feature branchit
- ▶ Uusi ominaisuus, esim. user story toteutetaan ensin omaan versionhallinnan haaraan
  - ▶ ja ominaisuuden valmistuttua haara mergetään päähaaraan (masteriin)
- ▶ Monet pitävät feature brancheja versionhallinnan best practicena
- ▶ Viime aikoina huomattu, että feature branchit aiheuttavat helposti pahoja merge-konflikteja sprintin lopussa
- ▶ Seurauksena pienimuotoinen integraatiohelvetti: merge hell
- ▶ Arkipäivää ohjelmistotiimissä



 12:55 PM

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

# Trunk based development

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

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

- ▶ Uusi trendi *trunk based development*: pitkäikäisiä feature brancheja ei käytetä ollenkaan
- ▶ Kaikki koodi suoraan pääkehityshaaraan, josta käytetään nimitystä *trunk*
- ▶ Pääkehityshaara voi olla master tai joku erillinen branch
- ▶ Ohjelmiston kustakin julkaistusta versiosta saatetaan tehdä oma *release branch*
- ▶ Pakottaa sovelluskehittäjät tekemään pieniä, nopeasti päähaaraan mergetäviä muutoksia
- ▶ Käytetään feature toggleja: puolivalmiitakin ominaisuuksia voidaan helposti ohjelmoida päähaaraan ilman toiminnallisuuden rikkomista
- ▶ Edellyttää sovelluskehittäjiltä kuria ja systemaattisuutta

# Trunk based development

- ▶ Uusi trendi *trunk based development*: pitkäikäisiä feature brancheja ei käytetä ollenkaan
- ▶ Kaikki koodi suoraan pääkehityshaaraan, josta käytetään nimitystä *trunk*
- ▶ 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 kuria ja systemaattisuutta
- ▶ Kehitysmallia noudattavat esim. Google, Facebook ja Netflix

# 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 ylläpitovastuilla olevia ihmisiä

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

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

- ▶ *DevOps*: toimintamalli missä dev ja ops työskentelevät tiiviisti yhdessä
- ▶ DevOps on termi hypetermi, jonka merkitys osin epäselvä
  - ▶ työpaikkailmoituksissa voidaan arvostaa DevOps-taitoja
  - ▶ tai etsiä ihmistä DevOps-tiimiin
  - ▶ myynnissä mitä erilaisempia DevOps-työkaluja
- ▶ Suurin osa määritelmistä tarkoittaa DevOpsilla *kehittäjien ja järjestelmäylläpidon yhteistä työnteon tapaa*, ja sen takia onkin hyvä puhua DevOps-kulttuurista
- ▶ Työkaluja/asioita jotka kiittyvät 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 devops:ia mahdollistavista asioista *infrastructure as code*
  - ▶ Fyysisten 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 devops:ia mahdollistavista asioista *infrastructure as code*
  - ▶ Fyysisten 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

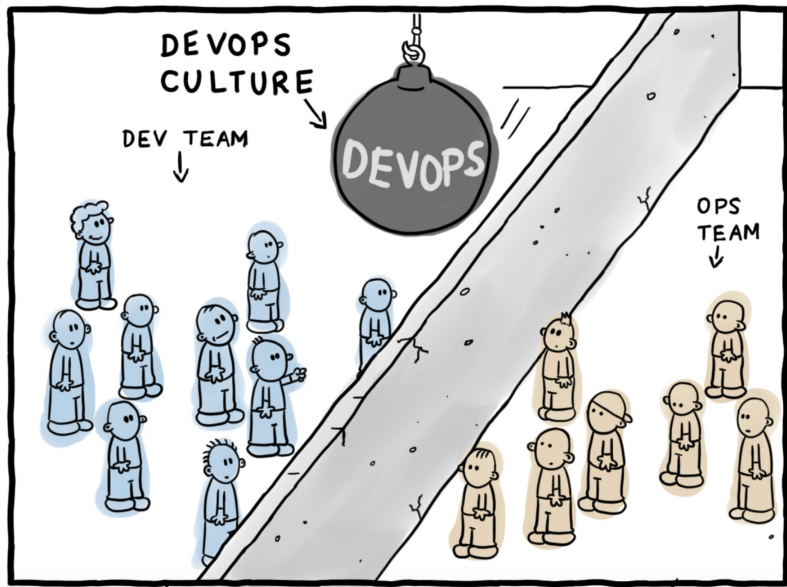


- ▶ Monet listatuista kehittyneet viimeisen 5-10 vuoden aikana ja mahdollistaneet DevOpsin helpomman soveltamisen
- ▶ Eräs tärkeimmistä devops:ia mahdollistavista asioista *infrastructure as code*
  - ▶ Fyysisten palvelinten sijaan virtuaalisia ja pilvessä toimivia palvelimia, joita voi konfiguroida 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 ketteryttä askeleen pitemmälle
- ▶ Mahdollistaa että kehitystiimi pystyy viemään vaivattomasti uudet toiminnallisuudet tuotantoympäristöön ja jopa testaamaan sekä operoimaan niitä tuotannossa

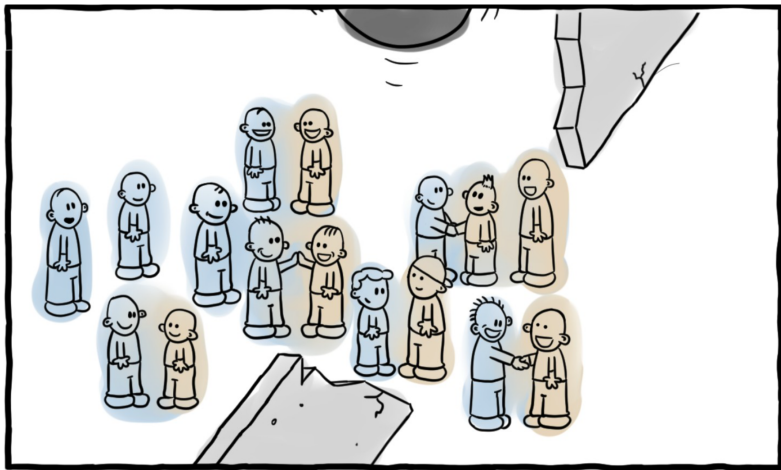
- ▶ Scrumin ja agilen eräs tärkeimmistä periaatteista on tehdä kehitystiimeistä itseorganisoituvia ja “cross functional”
- ▶ DevOps on keino viedä ketteryttä askeleen pitemmälle
- ▶ Mahdollistaa että kehitystiimi pystyy viemään vaivattomasti uudet toiminnallisuudet tuotantoympäristöön ja jopa testaamaan sekä operoimaan niitä tuotannossa
- ▶ DevOps siis laajentaa ketteryyden koskemaan myös järjestelmäylläpitoa



containers IaaS automation continuous integration  
PaaS automation containers automation  
infrastructure as code cloud collaboration

# CRASH

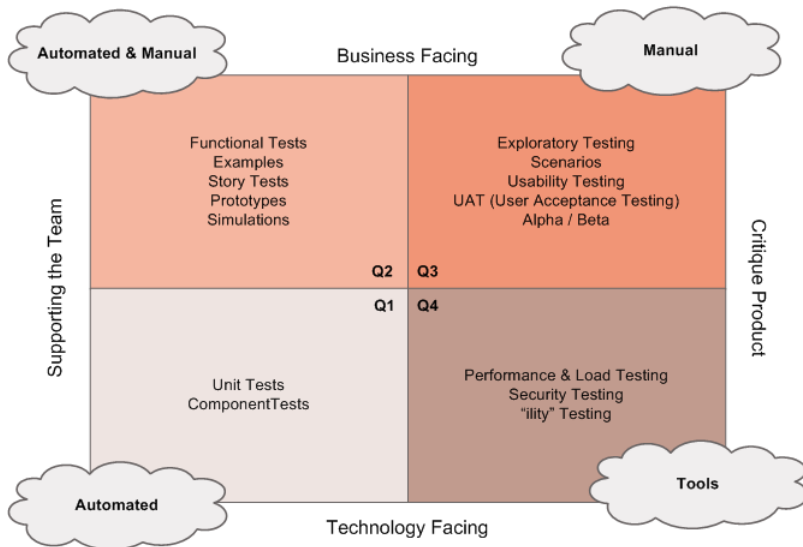
---



DANIEL STORİ {TURNOFF.US}

# Yhteenveto - ketterän testauksen nelikettä

Agile Testing Quadrants





# Yhteenveto - 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

# Yhteenveto - 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* edellyttävät manuaalista työtä

# Yhteenveto - 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* edellyttävät manuaalista 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
- ▶ *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 koodia käyttävän “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 koodia käyttävän “pääohjelman”
- ▶ Kattavien yksikkötestien tekeminen ei yleensä ole mielekästä ohjelman kaikille luokille
- ▶ Mielummin integraatiotason testejä ohjelman isompien komponenttien rajapintoja vasten
  - ▶ Pysyvät todennäköisemmin valideina komponenttien sisäisen rakenteen muuttuessa

- ▶ Oppikirjamääritelmän mukaista TDD:tä sovelletaan harvoin
  - ▶ Välillä TDD hyödyllinen, esim. testattaessa rajapintoja, joita käyttäviä komponentteja ei ole vielä olemassa
  - ▶ Testit tekee samalla vaivalla kuin koodia käyttävän “pääohjelman”
- ▶ Kattavien yksikkötestien tekeminen ei yleensä ole mielekästä ohjelman kaikille luokille
- ▶ Mielummin integraatiotason testejä ohjelman isompien komponenttien rajapintoja vasten
  - ▶ Pysyvät todennäköisemmin valideina komponenttien sisäisen rakenteen muuttuessa
- ▶ 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 koodia käyttävän “pääohjelman”
- ▶ Kattavien yksikkötestien tekeminen ei yleensä ole mielekästä ohjelman kaikille luokille
- ▶ Mielummin integraatiotason testejä ohjelman isompien komponenttien rajapintoja vasten
  - ▶ Pysyvät todennäköisemmin valideina komponenttien sisäisen rakenteen muuttuessa
- ▶ Yksikkötestaus hyödyllisimmillään kompleksia logiikkaa sisältäviä luokkia testattaessa
- ▶ Käyttöliittymän läpi suoritettavat, käyttäjän interaktiota simuloivat testit usein hyödyllisimpiä
  - ▶ Liian aikaisin tehtynä ne saattavat aiheuttaa kohtuuttoman paljon ylläpitovaivaa

- ▶ Oppikirjamääritelmän mukaista TDD:tä sovelletaan harvoin
  - ▶ Välillä TDD hyödyllinen, esim. testattaessa rajapintoja, joita käyttäviä komponentteja ei ole vielä olemassa
  - ▶ Testit tekee samalla vaivalla kuin koodia käyttävän “pääohjelman”
- ▶ Kattavien yksikkötestien tekeminen ei yleensä ole mielekästä ohjelman kaikille luokille
- ▶ Mielummin integraatiotason testejä ohjelman isompien komponenttien rajapintoja vasten
  - ▶ Pysyvät todennäköisemmin valideina komponenttien sisäisen rakenteen muuttuessa
- ▶ Yksikkötestaus hyödyllisimmillään kompleksia logiikkaa sisältäviä luokkia testattaessa
- ▶ Käyttöliittymän läpi suoritettavat, käyttäjän interaktiota simuloivat testit usein hyödyllisimpiä
  - ▶ Liian aikaisin tehtynä ne saattavat aiheuttaa kohtuuttoman paljon ylläpitovaivaa
- ▶ Automaattisia testejä kannattaa tehdä etenkin niiden komponenttien rajapintoihin, joita muokataan usein

- ▶ 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* tuotantoympäristössä

- ▶ 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