

# Ohjelmistotuotanto

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

syksy 2019

Luento 5

11.11.2019

# Miniprojektien aloitustilaisuudet

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

Jos taskien työmäärä on arvioitu tunteina, tulee taskiin käytetty työaika päivän päätteeksi aina vähentää työmääräarviosta

☐ Kyllä

☒ Ei

# Ohjelmistojen laadunhallinta

- ▶ Laadunhallinnan kaksi oleellista kysymystä
  - ▶ verifiointi: *are we building the product right*
  - ▶ validointi: *are we building the right product*

# Ohjelmistojen laadunhallinta

- ▶ Laadunhallinnan kaksi oleellista kysymystä
  - ▶ verifiointi: *are we building the product right*
  - ▶ validointi: *are we building the right product*
- ▶ Verifiointissa varmistetaan, että ohjelmisto toteuttaa vaatimusmäärittelyn aikana asetetut vaatimukset
  - ▶ Testataan toiminnalliset ja ei-toiminnalliset vaatimukset

# Ohjelmistojen laadunhallinta

- ▶ Laadunhallinnan kaksi oleellista kysymystä
  - ▶ verifiointi: *are we building the product right*
  - ▶ validointi: *are we building the right product*
- ▶ Verifiointissa varmistetaan, että ohjelmisto toteuttaa vaatimusmäärittelyn aikana asetetut vaatimukset
  - ▶ Testataan toiminnalliset ja ei-toiminnalliset vaatimukset
- ▶ Validointi varmistetaan, että ohjelmisto täyttää käyttäjän odotukset
  - ▶ Vaatimusmäärittelyn aikana kirjatut vaatimukset eivät ole aina se mitä käyttäjä todella tarvitsee

# Verifiointi ja Validointi

- ▶ Tavoitteena on varmistaa että ohjelma on “riittävän hyvä” käyttötarkoitukseensa
  - ▶ Hyvyys on suhteellista ja riippuu ohjelman käyttötarkoituksesta
  - ▶ Ohjelman ei yleensä tarvitse olla virheetön ollakseen riittävän hyvä käytettäväksi



# Verifiointi ja Validointi

- ▶ Tavoitteena on varmistaa että ohjelma on “riittävän hyvä” käyttötarkoitukseensa
  - ▶ Hyvyys on suhteellista ja riippuu ohjelman käyttötarkoituksesta
  - ▶ Ohjelman ei yleensä tarvitse olla virheetön ollakseen riittävän hyvä käytettäväksi
- ▶ Verifiointin ja validoinnin suorittamista käytetään yleisesti nimitystä *laadunhallinta* (engl. quality assurance, QA)
- ▶ Jos laadunhallinta on erillisen tiimin vastuulla, käytetään tästä usein nimitystä *QA-tiimi*

# Verifioinnin ja Validoinnin tekniikat

- ▶ Verifioinnissa käytetään kahta erilaista tekniikkaa
  - ▶ Katselmointeja
  - ▶ Testausta

# Verifioinnin ja Validoinnin tekniikat

- ▶ Verifioinnissa käytetään kahta erilaista tekniikkaa
  - ▶ Katselmointeja
  - ▶ Testausta
- ▶ Katselmoinneissa (review) käydään läpi tuotantoprosessin aikana tehtyjä dokumentteja ja ohjelmakoodia, ja etsitään näistä ongelmia

# Verifioinnin ja Validoinnin tekniikat

- ▶ Verifioinnissa käytetään kahta erilaista tekniikkaa
  - ▶ Katselmointeja
  - ▶ Testausta
- ▶ Katselmoinneissa (review) käydään läpi tuotantoprosessin aikana tehtyjä dokumentteja ja ohjelmakoodia, ja etsitään näistä ongelmia
- ▶ Katselmointi on *staattinen tekniikka*, suorituskelpoista ohjelmakoodia ei tarvita
  - ▶ Jos katselmoinnin kohteena on ohjelmakoodi, ei sitä katselmoinnissa suoriteta

# Verifioinnin ja Validoinnin tekniikat

- ▶ Verifioinnissa käytetään kahta erilaista tekniikkaa
  - ▶ Katselmointeja
  - ▶ Testausta
- ▶ Katselmoinneissa (review) käydään läpi tuotantoprosessin aikana tehtyjä dokumentteja ja ohjelmakoodia, ja etsitään näistä ongelmia
- ▶ Katselmointi on *staattinen tekniikka*, suorituskelpoista ohjelmakoodia ei tarvita
  - ▶ Jos katselmoinnin kohteena on ohjelmakoodi, ei sitä katselmoinnissa suoriteta
- ▶ Testaus on *dynaaminen tekniikka*, joka edellyttää ohjelmakoodin suorittamista
  - ▶ tarkkaillaan miten ohjelma reagoi annettuihin testisyötteisiin

# Vaatimusten validointi

- ▶ Ohjelmistolle määritellyt vaatimukset on validoitava:
  - ▶ varmistettava, että määrittelydokumentti määrittelee oikeanlaisen ohjelmiston

# Vaatimusten validointi

- ▶ Ohjelmistolle määritellyt vaatimukset on validoitava:
  - ▶ varmistettava, että määrittelydokumentti määrittelee oikeanlaisen ohjelmiston
- ▶ Vesiputousmallissa määrittelydokumentin kirjattujen vaatimusten validointi suoritetaan *katselmoimalla*
  - ▶ Vaatimusmäärittelyn lopuksi asiakas tarkastaa määrittelydokumenttiin kirjattujen vaatimuksien vastaavan asiakkaan kuvaa tilattavasta järjestelmästä
  - ▶ Katselmoinnin jälkeen määrittelydokumentti jäädytetään ja sen muuttaminen vaatii yleensä monimutkaista prosessia

# Vaatimusten validointi

- ▶ Ohjelmistolle määritellyt vaatimukset on validoitava:
  - ▶ varmistettava, että määrittelydokumentti määrittelee oikeanlaisen ohjelmiston
- ▶ Vesiputousmallissa määrittelydokumentin kirjattujen vaatimusten validointi suoritetaan *katselmoimalla*
  - ▶ Vaatimusmäärittelyn lopuksi asiakas tarkastaa määrittelydokumenttiin kirjattujen vaatimuksien vastaavan asiakkaan kuvaa tilattavasta järjestelmästä
  - ▶ Katselmoinnin jälkeen määrittelydokumentti jäädytetään ja sen muuttaminen vaatii yleensä monimutkaista prosessia
- ▶ Ketterässä ohjelmistotuotannossa vaatimusten validointi tapahtuu sprinttien päättävien demonstraatioiden yhteydessä
  - ▶ Asiakkaalle näytetään ohjelman toimivaa versiota
  - ▶ Asiakas voi itse verrata onko lopputulos haluttu
  - ▶ Jos ei, on seuraavassa sprintissä mahdollista ottaa korjausliike



- ▶ Koodin katselmointi eli lukeminen jonkun muun kuin ohjelmoijan toimesta on havaittu tehokkaaksi keinoksi laadun parantamisessa

# Koodin katselmointi

- ▶ Koodin katselmointi eli lukeminen jonkun muun kuin ohjelmoijan toimesta on havaittu tehokkaaksi keinoksi laadun parantamisessa
- ▶ Katselmoinnin avulla voidaan havaita koodista ongelmia, joita testauksella ei välttämättä havaita, esim.
  - ▶ noudattaako koodi sovittua tyyliä
  - ▶ onko koodi ylläpidettävää
  - ▶ onko koodissa tietoturvan kannalta vaarallisia piirteitä

# Koodin katselmointi

- ▶ Koodin katselmointi eli lukeminen jonkun muun kuin ohjelmoijan toimesta on havaittu tehokkaaksi keinoksi laadun parantamisessa
- ▶ Katselmoinnin avulla voidaan havaita koodista ongelmia, joita testauksella ei välttämättä havaita, esim.
  - ▶ noudattaako koodi sovittua tyyliä
  - ▶ onko koodi ylläpidettävää
  - ▶ onko koodissa tietoturvan kannalta vaarallisia piirteitä
- ▶ Perinteisesti käyty läpi onko koodissa tiettyjä checklisteissä listattuja riskialttiita piirteitä

|    |   |
|----|---|
| V1 | Variables are lower case words separated by underscores ( <code>this_is_a_var</code> )  |
| V2 | Constants are upper case words separated by underscores. ( <code>THIS_IS_A_CONST</code> )   |
| V3 | No <b>int</b> declarations. (Use <b>uint8</b> , <b>int8</b> , <b>uint16</b> , <b>int16</b> , <b>uint32</b> , <b>int32</b> instead.) |
| V4 | One declaration per line. (Exception: Highly coupled variables, i.e. <code>width</code> and <code>height</code> .)                  |
| V5 | All declarations have a comment after them explaining the variable.   |
| V6 | Units are declared when appropriate.  |
| V7 | No hidden variables. That is, a variable defined in an inner block may not have the same name as variable in an outer block.        |
| V8 | Never use "O" (Capital O) or "l" (lower case "l") for variable or constant names.   |

- Joissakin kielissä, esim. Javassa kääntäjän tuki tekee osan näistä tarkistuksista turhaksi

- Nykyään on tarjolla katselmointia automatisoivia *staattista analyysiä* tekeviä työkaluja esim. Javalla Checkstyle, Javascriptilla ESLint

# Staattinen analyysi

- ▶ Nykyään on tarjolla katselmointia automatisoivia *staattista analyysiä* tekeviä työkaluja esim. Javalla Checkstyle, Javascriptilla ESLint
- ▶ Myös pilvipalveluna toimivia staattisen analyysin työkaluja kuten Codeclimate
  - ▶ Suorittavat tarkastukset aina kun uutta koodia pushataan GitHubiin
  - ▶ Huomaavat koodin laadun muutoksista, esim. jos koodin kompleksisuus kasvaa muutosten yhteydessä

# Koodin katselmointi: GitHub ja pull requestit

- ▶ GitHubin *pull requestit* tarjoavat hyvän työkalun koodikatselmointiin
- ▶ Työn kulku on seuraava
  - ▶ Sovelluskehittäjä forkkaa repositorin itselleen, tekee muutokset omaan repositorioon ja tekee pull requestin
  - ▶ Joku, esim. *senior developer* tekee katselmoinnin pull requestille
  - ▶ Jos koodi ei ole riittävän hyvää, annetaan pull requestin tekijälle parannusehdotuksia
  - ▶ Muutosten ollessa hyväksyttäviä, pull request mergetään päärepositorioon

# Pullrequest TMC:hen

 testmycode / tmc-server

 Unwatch ▾ 8

## Course participants #201

 **Open** kennyhei wants to merge 9 commits into `testmycode:master` from `rage:course-participants`

 Conversation 24

 Commits 9

 Files changed 13



kennyhei commented on Oct 27, 2014

Implementing [#185](#)



kennyhei added some commits on Oct 21, 2014



 Course JSON with participants

9287e10



 Course knows its students through submissions and vice versa

e3e7c03



 Prettier JSON

b1b5dd7



lib/course\_info.rb

[View full changes](#)



```
@@ -31,6 +32,17 @@ def course_data(course)
```

31  
32  
33

32  
33  
34

```
})  
end
```

```
+ # Course JSON with participants  
+ def course_participants_data(course)  
+   participants = course.users  
+  
+   data = {  
+     :id => course.id,  
+     :name => course.name,  
+     :participants => participants.map {|participant| participant_data(participant)}
```



mpartel added a note on Oct 29, 2014

Owner



On my desktop, with the mooc production DB dump, this takes around 30 seconds for the k2014-mooc course. I'd really like to avoid adding more really slow queries to TMC.

Would the following make sense?

- Let this only return a list of participants and their newest submission IDs.
- Load a user's exercise statuses on demand, and cache them either on your side or maybe in TMC until the submission ID changes.
- Consider having the per-user URL support [ETags](#).

# Koodin katselmointi ketterissä menetelmissä

- ▶ Toisin kuin Scrum, eXtreme Programming eli XP määrittelee useita sovelluskehityksen käytänteitä
- ▶ Suuri osa XP:n käytänteistä on hyvin tunnettuja *best practiseja*, vietyinä äärimmäiseen (extreme) muotoon

# Koodin katselmointi ketterissä menetelmissä

- ▶ Toisin kuin Scrum, eXtreme Programming eli XP määrittelee useita sovelluskehityksen käytänteitä
- ▶ Suuri osa XP:n käytänteistä on hyvin tunnettuja *best practiseja*, vietyinä äärimmäiseen (extreme) muotoon
- ▶ Osa käytänteistä tähtää laadun maksimoimiseen ja kolmen voidaan ajatella olevan katselmoinnin äärimmilleen vietyjä muotoja

# Pariohjelmointi

- ▶ *Pariohjelmoinnissa* (pair programming) kaksi ohjelmoijaa työskentelee yhdessä samalla koneella
- ▶ Koodia kirjoittava osapuoli toimii *ohjaajana* (driver) ja toinen *navigoijana* (navigator), roolia vaihdetaan sopivin väliajoin
  - ▶ Navigoija tekee koodiin jatkuvaa katselmointia

# Pariohjelmointi

- ▶ *Pariohjelmoinnissa* (pair programming) kaksi ohjelmoijaa työskentelee yhdessä samalla koneella
- ▶ Koodia kirjoittava osapuoli toimii *ohjaajana* (driver) ja toinen *navigoijana* (navigator), roolia vaihdetaan sopivin väliajoin
  - ▶ Navigoija tekee koodiin jatkuvaa katselmointia
- ▶ Etuja:
  - ▶ Parantaa ohjelmoijien kuria ja työhön keskittymistä
  - ▶ Hyvä oppimisen väline: ohjelmoijat oppivat toisiltaan erityisesti noviisit kokeneimmilta, järjestelmän tietyn osan tuntee aina useampi ohjelmoija
  - ▶ Todettu vähentävän bugien määrää 15-50%, kokonaisresurssin kulutus nousee hieman

- ▶ Pariohjelmointi tai useamman ihmisen versio siitä, *mob-programming* on melko yleistä
- ▶ “Määritelmän” mukaista systemaattista pariohjelmointia tehdään aika harvassa paikassa aamusta iltaan

- ▶ Pariohjelmointi tai useamman ihmisen versio siitä, *mob-programming* on melko yleistä
- ▶ “Määritelmän” mukaista systemaattista pariohjelmointia tehdään aika harvassa paikassa aamusta iltaan
- ▶ Yleensä ohjelmoidaan yksin, mutta spontaania pariutumista ja ryhmäytymistä tapahtuu
  - ▶ erityisesti teknisesti haasteellisissa koodin osissa
  - ▶ tai jos kyse itselle tuntemattomasta osasta koodia

# Koodin katselmointi ketterissä menetelmissä

- ▶ *Koodin yhteisomistajuus* (collective code ownership): kukaan yksittäinen ohjelmoija ei hallitse yksin mitään kohtaa koodista
  - ▶ Kaikilla lupa tehdä muutoksia mihin tahansa kohtaan koodia



# Koodin katselmointi ketterissä menetelmissä

- ▶ *Koodin yhteisomistajuus* (collective code ownership): kukaan yksittäinen ohjelmoija ei hallitse yksin mitään kohtaa koodista
  - ▶ Kaikilla lupa tehdä muutoksia mihin tahansa kohtaan koodia
- ▶ Pariohjelmointi tukee yhteisomistajuutta
- ▶ Yhteisomistajuudessa on riskinsä: koodia tuntematon voi saada pahaa jälkeä aikaan
  - ▶ XP eliminoi riskejä testauksiin liittyvillä käytänteillä

# Koodin katselmointi ketterissä menetelmissä

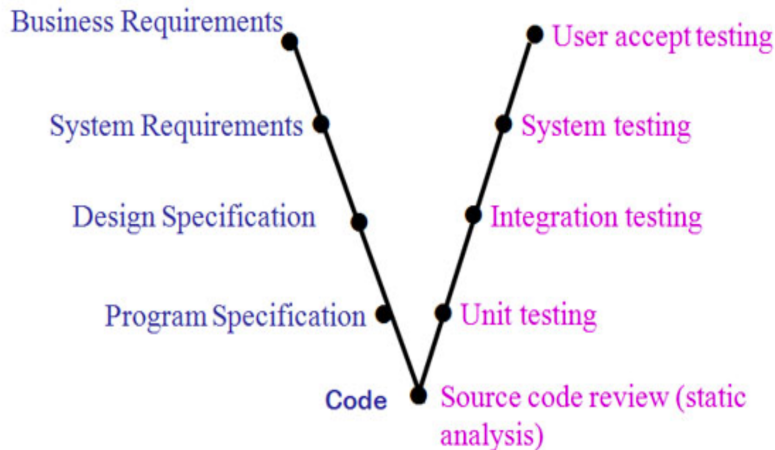
- ▶ *Koodin yhteisomistajuus* (collective code ownership): kukaan yksittäinen ohjelmoija ei hallitse yksin mitään kohtaa koodista
  - ▶ Kaikilla lupa tehdä muutoksia mihin tahansa kohtaan koodia
- ▶ Pariohjelmointi tukee yhteisomistajuutta
- ▶ Yhteisomistajuudessa on riskinsä: koodia tuntematon voi saada pahaa jälkeä aikaan
  - ▶ XP eliminoi riskejä testauksiin liittyvillä käytänteillä
- ▶ *Ohjelmointistandardi* (coding standards): tiimi määrittelee koodityylin, johon kaikki ohjelmoijat sitoutuvat
  - ▶ Tyylillä tarkoitetaan nimeämiskäytäntöä, koodin muotoilua ja myös tiettyjä ohjelman rakenteeseen liittyviä seikkoja
- ▶ Noudattamista kontrolloidaan osittain automaattisesti staattisen analyysin työkaluilla kuten checkstylellä

- ▶ Ohjelmien osoittaminen virheettömäksi on mahdotonta
- ▶ Testauksen tarkoituksena vakuuttaa asiakas ja kehittäjät siitä, että ohjelmisto on *tarpeeksi hyvä* käytettäväksi

- ▶ Ohjelmien osoittaminen virheettömäksi on mahdotonta
- ▶ Testauksen tarkoituksena vakuuttaa asiakas ja kehittäjät siitä, että ohjelmisto on *tarpeeksi hyvä* käytettäväksi
- ▶ Testauksella on kaksi eriävää tavoitetta
  - ▶ osoittaa, että ohjelmisto täyttää sille asetetut vaatimukset
  - ▶ löytää ohjelmistosta virheitä

- ▶ Ohjelmien osoittaminen virheettömäksi on mahdotonta
- ▶ Testauksen tarkoituksena vakuuttaa asiakas ja kehittäjät siitä, että ohjelmisto on *tarpeeksi hyvä* käytettäväksi
- ▶ Testauksella on kaksi eriävää tavoitetta
  - ▶ osoittaa, että ohjelmisto täyttää sille asetetut vaatimukset
  - ▶ löytää ohjelmistosta virheitä
- ▶ Tähtää ohjelman *ulkoisen laadun* (external quality) eli käyttäjän kokemuksen laadun parantamiseen
  - ▶ External Quality is the fitness for purpose of the software. It's most obvious measure is the Functional Tests, and some measure of the bugs that are still loose when the product is released.

# Testauksen tasot



- ▶ *Yksikkötestaus* (unit testing)
  - ▶ Yksittäisten luokkien, metodien ja moduulien testaus erillään muusta kokonaisuudesta

- ▶ *Yksikkötestaus* (unit testing)
  - ▶ Yksittäisten luokkien, metodien ja moduulien testaus erillään muusta kokonaisuudesta
- ▶ *Integraatiotestaus* (integration testing)
  - ▶ Yksittäin testattujen komponenttien liittäminen yhteen eli integrointi ja kokonaisuuden testaus



- ▶ *Yksikkötestaus* (unit testing)
  - ▶ Yksittäisten luokkien, metodien ja moduulien testaus erillään muusta kokonaisuudesta
- ▶ *Integraatiotestaus* (integration testing)
  - ▶ Yksittäin testattujen komponenttien liittäminen yhteen eli integrointi ja kokonaisuuden testaus
- ▶ *Järjestelmätestaus* (system testing)
  - ▶ Toimiiko ohjelmisto vaatimuksiin kirjatulla tavalla?
  - ▶ Jakautuu useisiin alalajeihin

- ▶ *Yksikkötestaus* (unit testing)
  - ▶ Yksittäisten luokkien, metodien ja moduulien testaus erillään muusta kokonaisuudesta
- ▶ *Integraatiotestaus* (integration testing)
  - ▶ Yksittäin testattujen komponenttien liittäminen yhteen eli integrointi ja kokonaisuuden testaus
- ▶ *Järjestelmätestaus* (system testing)
  - ▶ Toimiiko ohjelmisto vaatimuksiin kirjatulla tavalla?
  - ▶ Jakautuu useisiin alalajeihin
- ▶ *Käyttäjän hyväksymistestaus* (user acceptance testing)
  - ▶ Loppukäyttäjän tuotteelle suorittama testaus

# Järjestelmätestaus

- ▶ Tarkoitus varmistaa, että järjestelmä toimii vaatimuksiin kirjatulla tavalla
- ▶ Kehittäjäorganisaatio suorittaa järjestelmätestauksen

# Järjestelmätestaus

- ▶ Tarkoitus varmistaa, että järjestelmä toimii vaatimuksiin kirjatulla tavalla
- ▶ Kehittäjäorganisaatio suorittaa järjestelmätestauksen
- ▶ Testaus tapahtuu ilman tietoa järjestelmän sisäisestä rakenteesta eli kyseessä *black box* -testaus
- ▶ Testataan järjestelmää saman rajapinnan kautta, jonka kautta järjestelmää käytetään

# Järjestelmätestaus

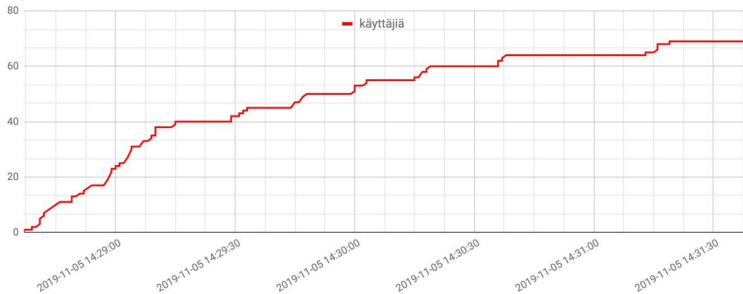
- ▶ Tarkoitus varmistaa, että järjestelmä toimii vaatimuksiin kirjatulla tavalla
- ▶ Kehittäjäorganisaatio suorittaa järjestelmätestauksen
- ▶ Testaus tapahtuu ilman tietoa järjestelmän sisäisestä rakenteesta eli kyseessä *black box* -testaus
- ▶ Testataan järjestelmää saman rajapinnan kautta, jonka kautta järjestelmää käytetään
- ▶ Perustuu järjestelmän potentiaalisiin käyttöskenaarioihin
  - ▶ jos vaatimukset ilmaistu user storyina, on niistä helppo muotoilla testejä
  - ▶ jos storyillä hyväksymiskriteeriot, tilanne on vieläkin parempi
- ▶ Kutsutaan myös *toiminnallisiksi testeiksi* (functional test)

# Järjestelmätestaus

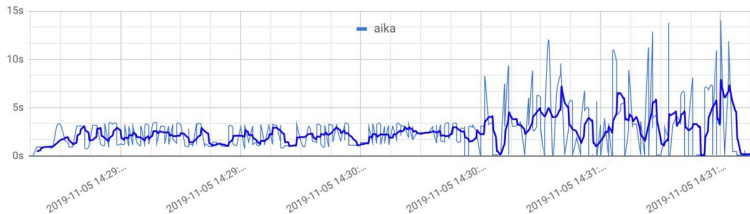
- ▶ Tarkoitus varmistaa, että järjestelmä toimii vaatimuksiin kirjatulla tavalla
- ▶ Kehittäjäorganisaatio suorittaa järjestelmätestauksen
- ▶ Testaus tapahtuu ilman tietoa järjestelmän sisäisestä rakenteesta eli kyseessä *black box* -testaus
- ▶ Testataan järjestelmää saman rajapinnan kautta, jonka kautta järjestelmää käytetään
- ▶ Perustuu järjestelmän potentiaalisiin käyttöskenaarioihin
  - ▶ jos vaatimukset ilmaistu user storyina, on niistä helppo muotoilla testejä
  - ▶ jos storyillä hyväksymiskriteeriot, tilanne on vieläkin parempi
- ▶ Kutsutaan myös *toiminnallisiksi testeiksi* (functional test)
- ▶ Toiminnallisen testauksen lisäksi järjestelmätestaukseen kuuluu
  - ▶ Käytettävyystestaus
  - ▶ Suorituskykytestaus
  - ▶ Kuormitustestaus
  - ▶ Tietoturvan testaus

# Kuormitustestaus: ennen

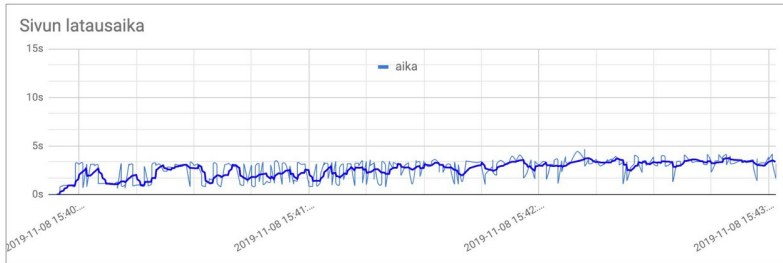
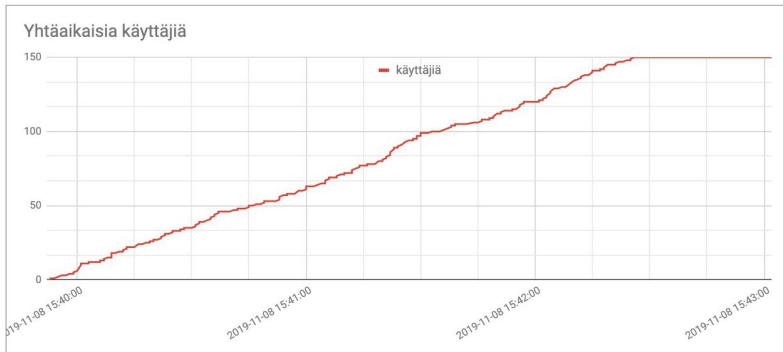
Yhtäaikaisia käyttäjiä



Sivun latausaika



# Kuormitustestaus: jälkeen





# Testitapausten valinta

- ▶ Kattava testaaminen on mahdotonta ja testaus työlästä
- ▶ Oleellista löytää kohtuullisen kokoinen *testitapausten joukko*, jonka avulla löytyy mahdollisimman suuri määrä virheitä

# Testitapausten valinta

- ▶ Kattava testaaminen on mahdotonta ja testaus työlästä
- ▶ Oleellista löytää kohtuullisen kokoinen *testitapausten joukko*, jonka avulla löytyy mahdollisimman suuri määrä virheitä
- ▶ *Testitapaus* testaa ohjelmiston toiminnallisuutta joillakin syötteillä
- ▶ Useat syötteen ohjelmiston toiminnan kannalta samanlaisia
  - ▶ nämä muodostavat *ekvivalenssiluokan*

# Testitapausten valinta

- ▶ Kattava testaaminen on mahdotonta ja testaus työlästä
- ▶ Oleellista löytää kohtuullisen kokoinen *testitapausten joukko*, jonka avulla löytyy mahdollisimman suuri määrä virheitä
- ▶ *Testitapaus* testaa ohjelmiston toiminnallisuutta joillakin syötteillä
- ▶ Useat syötteen ohjelmiston toiminnan kannalta samanlaisia
  - ▶ nämä muodostavat *ekvivalenssiluokan*
- ▶ Jaetaan syötteen ekvivalenssiluokkiin ja tehdään yksi testitapaus kutakin ekvivalenssiluokkaa tai syötteiden ekvivalenssiluokkien kombinaatiota kohti
- ▶ Erityisen kiinnostavia syötearvoja ovat ekvivalenssiluokkien väliset *raja-arvot*

# Testisyötteiden valinta

- ▶ Mitä testitapauksia kannattaisi valita *tekstiv:n sivun valintaikkunan* testaamiseen?
  - ▶ Tekstiv:n sivu vastaa lukua väliltä 100-899
  - ▶ Osaa välin luvuista vastaavaa sivua ei ole olemassa

# Testisyötteiden valinta

- ▶ Mitä testitapauksia kannattaisi valita *tekstiv:n sivun valintaikkunan* testaamiseen?
  - ▶ Tekstiv:n sivu vastaa lukua väliltä 100-899
  - ▶ Osaa välin luvuista vastaavaa sivua ei ole olemassa
- ▶ Testisyötteiden ekvivalenssiluokkia olisivat ainakin seuraavat
  - ▶ Olemassa olevaa sivua vastaavat luvut: 235
  - ▶ Validit luvut jotka eivät vastaa mitään sivua: ???
  - ▶ Liian pienet ja liian suuret luvut: 99, 900, 1000
  - ▶ Syötteet jotka sisältävät kiellettyjä merkkejä: 10X, 100X
  - ▶ Tyhjä syöte

# Tekstiv:n testitapaukset

[Edellinen sivu](#) | [Edellinen alasivu](#) | [Seuraava alasivu](#) | [Seuraava sivu](#)

Teksti-TV

[yle.fi/tekstiv](#)    [199 PÄÄHAKEMISTO](#)

[106](#) Taksisääntelyn purkaminen etenee

[107](#) Tieto irtisanoo lähes [180](#)

[162](#) Reuters: Kreikan kasvu yllätti

[651](#) MM-karsinta: Turkki nujersi Suomen

[221](#) Janne Keränen jälleen KalPa-sankari

|                      |          |                     |        |                         |           |
|----------------------|----------|---------------------|--------|-------------------------|-----------|
| <a href="#">101</a>  | UUTiset  | <a href="#">160</a> | TALOUS | <a href="#">190</a>     | ENGLISH   |
| <a href="#">201</a>  | URHEILU  | <a href="#">350</a> | RADIOT | <a href="#">470</a>     | VEIKKAUS  |
| <a href="#">300</a>  | OHJELMAT | <a href="#">400</a> | SÄÄ    | <a href="#">575</a>     | TEKSTI-TV |
| <a href="#">799</a>  | SVENSKA  | <a href="#">500</a> | ALUEET | <a href="#">890</a>     | KALENTERI |
| Sää paikkakunnittain |          |                     |        | <a href="#">406-408</a> |           |
| Nopea piiras         |          |                     |        | <a href="#">811</a>     |           |

[Edellinen sivu](#) | [Edellinen alasivu](#) | [Seuraava alasivu](#) | [Seuraava sivu](#)

[Kotimaa](#) | [Ulkomaat](#) | [Talous](#) | [Urheilu](#) | [Svenska sidor](#) | [Teksti-TV](#) [Yle.fi](#) [YLE Uutiset](#)

Sivun valinta:  OK

- ▶ Kohteena siis yksittäiset metodit ja luokat
- ▶ Testattavan koodin rakenne otetaan huomioon testejä laatiessa, *lasilaatikkotestausta* (white box testing)
- ▶ *Developer testing*: sovelluskehittäjien vastuulla

- ▶ Kohteena siis yksittäiset metodit ja luokat
- ▶ Testattavan koodin rakenne otetaan huomioon testejä laatiessa, *lasilaatikkotestausta* (white box testing)
- ▶ *Developer testing*: sovelluskehittäjien vastuulla
- ▶ Päättarkoitus *sisäisen laadun* (internal quality) kontrollointi
  - ▶ onko virheiden jäljitys ja korjaaminen helppoa
  - ▶ onko koodia helppo laajentaa ja jatkokehittää
  - ▶ pystytäänkö koodin toiminnallisuuden oikeellisuus varmistamaan muutoksia tehtäessä



- ▶ Iteratiivisessa kehityksessä sisäisellä laadulla suuri merkitys (ohjelmistoa laajennetaan koko ajan)

# Yksikkötestaus

- ▶ Iteratiivisessa kehityksessä sisäisellä laadulla suuri merkitys (ohjelmistoa laajennetaan koko ajan)
- ▶ Edesauttaa myös ulkoista eli asiakkaan kokemaa laatua
  - ▶ Yksikkötestit voivat eliminoida joitain asiakkaalle näkyviä virheitä, joita järjestelmätestauksen testitapaukset eivät löydä

# Yksikkötestaus

- ▶ Iteratiivisessa kehityksessä sisäisellä laadulla suuri merkitys (ohjelmistoa laajennetaan koko ajan)
- ▶ Edesauttaa myös ulkoista eli asiakkaan kokemaa laatua
  - ▶ Yksikkötestit voivat eliminoida joitain asiakkaalle näkyviä virheitä, joita järjestelmätestauksen testitapaukset eivät löydä
- ▶ Bugit on taloudellisesti edullista paikallistaa mahdollisimman aikaisessa vaiheessa
  - ▶ yksikkötestauksessa löydetty bugi on halvempi ja nopeampi korjata kuin järjestelmä- tai integraatiotestauksessa löytyvä

- ▶ Iteratiivisessa kehityksessä sisäisellä laadulla suuri merkitys (ohjelmistoa laajennetaan koko ajan)
- ▶ Edesauttaa myös ulkoista eli asiakkaan kokemaa laatua
  - ▶ Yksikkötestit voivat eliminoida joitain asiakkaalle näkyviä virheitä, joita järjestelmätestauksen testitapaukset eivät löydä
- ▶ Bugit on taloudellisesti edullista paikallistaa mahdollisimman aikaisessa vaiheessa
  - ▶ yksikkötestauksessa löydetty bugi on halvempi ja nopeampi korjata kuin järjestelmä- tai integraatiotestauksessa löytyvä
- ▶ Koska yksikkötestejä joudutaan ajamaan moneen kertaan, tulee niiden suorittaminen ja testien tulosten raportointi *automatisoida*

# Mitä tulisi testata yksikkötestein?

- ▶ JUnitin kehittäjän Kent Beckin vastaus:
  - ▶ “Do I have to write a test for everything?”
  - ▶ “No, just test everything that could reasonably break”

# Mitä tulisi testata yksikkötestein?

- ▶ JUnitin kehittäjän Kent Beckin vastaus:
  - ▶ “Do I have to write a test for everything?”
  - ▶ “No, just test everything that could reasonably break”
- ▶ Optimitilanteessa testitapaukset
  - ▶ kaikkien metodit ja niiden kutsukombinaatiot hyväksyttävillä parametrien arvoilla
  - ▶ ja virheellisillä parametrien arvoilla

# Mitä tulisi testata yksikkötestein?

- ▶ JUnitin kehittäjän Kent Beckin vastaus:
  - ▶ “Do I have to write a test for everything?”
  - ▶ “No, just test everything that could reasonably break”
- ▶ Optimitilanteessa testitapaukset
  - ▶ kaikkien metodit ja niiden kutsukombinaatiot hyväksyttävillä parametrien arvoilla
  - ▶ ja virheellisillä parametrien arvoilla
- ▶ Ekvivalenssiluokat ja niiden raja-arvot kannattaa huomioida
- ▶ Koska yksikkötestejä tehtäessä ohjelmakoodi on nähtävillä, testien parametrien ekvivalenssiluokat ja raja-arvot pääteltävissä koodista

## Ohtuvarasto, ekvivalenssiluokat: tyhjä, puolitäysi, täysi

```
public class Varasto {  
    private double tilavuus;  
    private double saldo;  
  
    public double otaVarastosta(double maara) {  
        if (maara < 0) return 0.0;  
  
        if(maara > saldo) {  
            double kaikkiMitaVoidaan = saldo;  
            saldo = 0.0;  
            return kaikkiMitaVoidaan;  
        }  
  
        saldo = saldo - maara;  
        return maara;  
    }  
}
```



- Yksikkötestien (ja toki myös muunkinlaisten testien) hyvyyttä voidaan mitata *testauskattavuuden* (test coverage) käsitteellä

- ▶ Yksikkötestien (ja toki myös muunkinlaisten testien) hyvyttä voidaan mitata *testauskattavuuden* (test coverage) käsitteellä
- ▶ Muutamaa eri tyyppiä
  - ▶ rivikattavuus
  - ▶ haarautumakattavuus
  - ▶ ehtokattavuus
  - ▶ polkukattavuus

# Testauskattavuus

- ▶ Yksikkötestien (ja toki myös muunkinlaisten testien) hyvyttä voidaan mitata *testauskattavuuden* (test coverage) käsitteellä
- ▶ Muutamaa eri tyyppiä
  - ▶ rivikattavuus
  - ▶ haarautumakattavuus
  - ▶ ehtokattavuus
  - ▶ polkukattavuus
- ▶ Rivi- ja haarautumakattavuudelle hyvä työkalutuki, esim JaCoCo

| Element                 | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|-------------------------|---------------------|------|-----------------|------|--------|------|--------|-------|--------|---------|
| Varasto(double, double) |                     | 0%   |                 | 0%   | 4      | 4    | 10     | 10    | 1      | 1       |
| toString()              |                     | 0%   |                 | n/a  | 1      | 1    | 1      | 1     | 1      | 1       |
| otaVarastosta(double)   |                     | 62%  |                 | 50%  | 2      | 3    | 4      | 8     | 0      | 1       |
| lisaaVarastoon(double)  |                     | 77%  |                 | 50%  | 2      | 3    | 2      | 6     | 0      | 1       |
| Varasto(double)         |                     | 82%  |                 | 50%  | 1      | 2    | 1      | 6     | 0      | 1       |
| paljonkoMahtuu()        |                     | 100% |                 | n/a  | 0      | 1    | 0      | 1     | 0      | 1       |
| getSaldo()              |                     | 100% |                 | n/a  | 0      | 1    | 0      | 1     | 0      | 1       |
| getTilavuus()           |                     | 100% |                 | n/a  | 0      | 1    | 0      | 1     | 0      | 1       |
| Total                   | 66 of 126           | 47%  | 11 of 16        | 31%  | 10     | 16   | 18     | 34    | 2      | 8       |

- Epäkattavasti testattu haarautumiskohta esim. if ilmaistaan keltaisella

```

51.     public void lisaaVarastoon(double maara) {
52.         ♦ if (maara < 0) // virhetilanteessa voidaan tehdä
53.         {
54.             return; // tällainen pikapoistuminenkin!
55.         }
56.         ♦ if (maara <= paljonkoMahtuu()) // omia aksessoreita voi kutsua
57.         {
58.             saldo 1 of 2 branches missed.++; // ihan suoraan sellaisinaan
59.         } else {
60.             saldo = tilavuus; // täyteen ja ylimäärä hukkaan!
61.         }
62.     }
63.

```

# Mutaatiotestaus

- ▶ Pelkkä kattavuus ei kerro paljoa testien laadusta
- ▶ Hyvien testien pitäisi huomata (ts. mennä rikki) jos ohjelmaan tulee bugi

# Mutaatiotestaus

- ▶ Pelkkä kattavuus ei kerro paljoa testien laadusta
- ▶ Hyvien testien pitäisi huomata (ts. mennä rikki) jos ohjelmaan tulee bugi
- ▶ Mutaatiotestaus (engl. mutation testing) tutkii testien hyvyyttä generoimalla koodiin *mutantteja* eli pieniä “bugeja” ja tarkastamalla hajoavatko testit
- ▶ Erilaisia mutanttityyppejä:

|                                |   |
|--------------------------------|---|
| <code>if ( x&lt;0 )</code>     | <code>if (x &lt;= 0) tai if (true)</code> |
| <code>x += 1</code>            | <code>x-= 1</code>                        |
| <code>return x</code>          | <code>return true</code>                  |
| <code>olio = new Olio()</code> | <code>olio = null</code>                  |

# Mutaatiotestaus

- ▶ Pelkkä kattavuus ei kerro paljoa testien laadusta
- ▶ Hyvien testien pitäisi huomata (ts. mennä rikki) jos ohjelmaan tulee bugi
- ▶ Mutaatiotestaus (engl. mutation testing) tutkii testien hyvyttä generoimalla koodiin *mutantteja* eli pieniä “bugeja” ja tarkastamalla hajoavatko testit
- ▶ Erilaisia mutanttityyppejä:

|                                |   |
|--------------------------------|---|
| <code>if ( x&lt;0 )</code>     | <code>if (x &lt;= 0) tai if (true)</code> |
| <code>x += 1</code>            | <code>x-= 1</code>                        |
| <code>return x</code>          | <code>return true</code>                  |
| <code>olio = new Olio()</code> | <code>olio = null</code>                  |

- ▶ Ongelmana on mutaatioiden suuri määrä ja ns. ekvivalentit mutantit
  - ▶ tulos vaatii aina ihmisen tulkintaa

- ▶ Järjestelmän yksittäiset, erillään yksikkötestatut luokat tulee integroida toimivaksi kokonaisuudeksi
- ▶ Integroinnin yhteydessä tai sen jälkeen suoritetaan integrointitestaus
  - ▶ Toimivatko komponentit yhdessä?



# Ohjelmiston integraatio

- ▶ Järjestelmän yksittäiset, erillään yksikkötestatut luokat tulee integroida toimivaksi kokonaisuudeksi
- ▶ Integroinnin yhteydessä tai sen jälkeen suoritetaan integrointitestaus
  - ▶ Toimivatko komponentit yhdessä?
- ▶ Kaksi lähestymistapaa: rakenteisiin perustuva ja toiminnallisuuksiin perustuva

# Ohjelmiston integraatio

- ▶ *Rakenteeseen perustuvassa* integraatiossa keskitytään kerrallaan sovelluksen yksittäisten arkkitehtuurillisten komponenttien integrointiin
  - ▶ Verkkokaupassa integroitaisiin sovelluslogiikan luokat, käyttöliittymän luokat ja tietokantarajapinta ensin omina kokonaisuuksinaan

# Ohjelmiston integraatio

- ▶ *Rakenteeseen perustuvassa* integraatiossa keskitytään kerrallaan sovelluksen yksittäisten arkkitehtuurillisten komponenttien integrointiin
  - ▶ Verkkokaupassa integroitaisiin sovelluslogiikan luokat, käyttöliittymän luokat ja tietokantarajapinta ensin omina kokonaisuuksinaan
- ▶ *Ominaisuuksiin perustuvassa* integroinnissa liitetään yhteen alikomponentit, jotka toteuttavat järjestelmän loogisen toimintakokonaisuuden
  - ▶ Esim. integroidaan kerrallaan kaikki storyn *lisää tuote ostokoriin* toteutukseen liittyvä koodi
  - ▶ Tämän jälkeen integroidaan seuraavan storyn koodi

# Ohjelmiston integraatio

- ▶ Vesiputuoksen toimintatapa
  - ▶ Yksittäiset komponentit ohjelmoidaan ja yhsikkötestataan erikseen
  - ▶ Tämän jälkeen ne integroidaan, yleensä rakenteeseen perustuen, kerralla yhteen

# Ohjelmiston integraatio

- ▶ Vesiputuoksen toimintatapa
  - ▶ Yksittäiset komponentit ohjelmoidaan ja yksikkötestataan erikseen
  - ▶ Tämän jälkeen ne integroidaan, yleensä rakenteeseen perustuen, kerralla yhteen
- ▶ Tämän tyylinen *big bang* -integraatio on osoittautunut todella riskialttiiksi
- ▶ Seurauksena usein ns. integraatiohelvetti

# Ohjelmiston integraatio

- ▶ Vesiputuoksen toimintatapa
  - ▶ Yksittäiset komponentit ohjelmoidaan ja yksikkötestataan erikseen
  - ▶ Tämän jälkeen ne integroidaan, yleensä rakenteeseen perustuen, kerralla yhteen
- ▶ Tämän tyylinen *big bang* -integraatio on osoittautunut todella riskialttiiksi
- ▶ Seurauksena usein ns. integraatiohelvetti
- ▶ Moderni ohjelmistotuotanto suosii ns. *jatkuvaa integraatiota*
  - ▶ Hyvin tiheässä tahdissa tapahtuvaa ominaisuuksiin perustuvaa integrointia

# Regressiotestaus

- ▶ Iteratiivisessa ohjelmistotuotannossa, jokainen iteraatio tuottaa ohjelmistoon uusia ominaisuuksia
- ▶ Tulee huolehtia, että ei rikota jo toimivia osia

# Regressiotestaus

- ▶ Iteratiivisessa ohjelmistotuotannossa, jokainen iteraatio tuottaa ohjelmistoon uusia ominaisuuksia
- ▶ Tulee huolehtia, että ei rikota jo toimivia osia
- ▶ Testit siis on suoritettava uudelleen aina kun ohjelmistoon tehdään muutoksia
- ▶ Tätä käytäntöä sanotaan *regressiotestaukseksi*



# Regressiotestaus

- ▶ Iteratiivisessa ohjelmistotuotannossa, jokainen iteraatio tuottaa ohjelmistoon uusia ominaisuuksia
- ▶ Tulee huolehtia, että ei rikota jo toimivia osia
- ▶ Testit siis on suoritettava uudelleen aina kun ohjelmistoon tehdään muutoksia
- ▶ Tätä käytäntöä sanotaan *regressiotestaukseksi*
- ▶ Regressiotestijoukko koostuu kaikista ohjelmistolle tehdyistä testeistä
  - ▶ sisältää yksikkö-, integraatio- ja järjestelmätesteistä

# Regressiotestaus

- ▶ Iteratiivisessa ohjelmistotuotannossa, jokainen iteraatio tuottaa ohjelmistoon uusia ominaisuuksia
- ▶ Tulee huolehtia, että ei rikota jo toimivia osia
- ▶ Testit siis on suoritettava uudelleen aina kun ohjelmistoon tehdään muutoksia
- ▶ Tätä käytäntöä sanotaan *regressiotestaukseksi*
- ▶ Regressiotestijoukko koostuu kaikista ohjelmistolle tehdyistä testeistä
  - ▶ sisältää yksikkö-, integraatio- ja järjestelmätesteistä
- ▶ Testaus on työlästä ja regressiotestauksen tarve tekee siitä entistä työläämpää
- ▶ *Testaus kannattaa automatisoida* mahdollisimman suurissa määrin