

# Ohjelmistotuotanto

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

syksy 2023

Luento 5

13.11.2022

## Kurssipalaute

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

# Kurssipalaute

- ▶ Kurssipalaute
  - ▶ Kurssilla lopussa kerättävän palautteen lisäksi ns. jatkuva palaute <https://norppa.helsinki.fi>
- ▶ Materiaalissa on paljon linkkejä, ja on epäselvää missä määrin niiden takana oleviin sivuihin kuuluu perehtyä
  - ▶ linkit jotka johtaa pois kurssisivulta on nice to know

# Kurssipalaute

- ▶ Kurssipalaute
  - ▶ Kurssilla lopussa kerättävän palautteen lisäksi ns. jatkuva palaute <https://norppa.helsinki.fi>
- ▶ Materiaalissa on paljon linkkejä, ja on epäselvää missä määrin niiden takana oleviin sivuihin kuuluu perehtyä
  - ▶ linkit jotka johtaa pois kurssisivulta on nice to know
- ▶ Kurssille on lisätty kaksi paja-aikaa
  - ▶ ma 10-12 ja ke 10-12, molemmat salissa BK107

# Miniprojektit lähestyvät

- ▶ Kohta alkaa...

## Ohjelmiston elinkaari (software lifecycle)

- ▶ Vaatimusten analysointi ja määrittely
- ▶ Suunnittelu
- ▶ Toteutus
- ▶ **Testaus ja laadunhallinta**
- ▶ Ohjelmiston ylläpito ja evoluutio

# Ohjelmistojen laadunhallinta

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

# Ohjelmistojen laadunhallinta

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

# Ohjelmistojen laadunhallinta

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

## Verifointi 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

# Verifointi 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
- ▶ Verifioinnin 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
  - ▶ Katselointeja
  - ▶ Testausta
- ▶ **Katselmoinneissa** (review) käydään läpi tuotantoprosessin aikana tehtyjä dokumentteja ja ohjelmakoodia, ja etsitään näistä ongelmia

# Verifioinnin ja Validoinnin tekniikat

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

Katselmointi

Vaatimusten validointi katselmoimalla dokumentaatiota

## Vaatimusten validointi katselmoimalla dokumentaatiota

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

## Vaatimusten validointi katselmoimalla dokumentaatiota

- ▶ Ohjelmistolle määritellyt vaatimukset on validoitava:
  - ▶ varmistettava, että määrittelydokumentti määrittelee oikeanlaisen ohjelmiston
- ▶ Vesiputousmallissa määrittelydokumentin kirjattujen vaatimusten validointi suoritetaan *katselmoimalla*

## Vaatimusten validointi katselmoimalla dokumentaatiota

- ▶ Ohjelmistolle määritellyt vaatimukset on validoitava:
  - ▶ varmistettava, että määrittelydokumentti määrittelee oikeanlaisen ohjelmiston
- ▶ Vesiputoousmallissa määrittelydokumentin kirjattujen vaatimusten validointi suoritetaan *katselmoimalla*
- ▶ Vaatimusmäärittelyn lopuksi asiakas tarkastaa vastaako määrittelydokumentti mielikuvaa tilattavasta järjestelmästä

## Vaatimusten validointi katselmoimalla dokumentaatiota

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

## Vaatimusten ketterä validointi

- ▶ Ketterässä ohjelmistotuotannossa vaatimusten validointi tapahtuu sprinttien päättävien demonstraatioiden yhteydessä

## Vaatimusten ketterä validointi

- ▶ Ketterässä ohjelmistotuotannossa vaatimusten validointi tapahtuu sprinttien päättävien demonstraatioiden yhteydessä
- ▶ Asiakkaalle näytetään ohjelman toimivaa versiota

## Vaatimusten ketterä validointi

- ▶ 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

## Koodin katselointi

- ▶ Koodin katselointi eli lukeminen jonkun muun kuin ohjelmoijan toimesta on tehokas keino laadun parantamisessa

## Koodin katselointi

- ▶ Koodin katselointi eli lukeminen jonkun muun kuin ohjelmoijan toimesta on tehokas keino laadun parantamisessa
- ▶ Voidaan havaita koodista ongelmia, joita testaus ei havaitse
  - ▶ noudattaako koodi sovittua tyyliä
  - ▶ onko koodi ylläpidettävää
  - ▶ onko koodissa tietoturvan kannalta vaarallisia piirteitä

## Koodin katselointi

- ▶ Koodin katselointi eli lukeminen jonkin muun kuin ohjelmoijan toimesta on tehokas keino laadun parantamisessa
- ▶ Voidaan havaita koodista ongelmia, joita testaus ei havaitse
  - ▶ 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 <code>int</code> declarations. (Use <code>uint8</code> , <code>int8</code> , <code>uint16</code> , <code>int16</code> , <code>uint32</code> , <code>int32</code> 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 käänitäjän tuki tekee osan näistä tarkistuksista turhaksi

# Staattinen analyysi

- ▶ Nykyään on tarjolla katselmointia automatisoivia *staattista analyysiä* tekeviä työkaluja
  - ▶ Pythonilla Pylint
  - ▶ JavaScriptilla ESlint
  - ▶ Javalla Checkstyle

# Staattinen analyysi

- ▶ Nykyään on tarjolla katselmointia automatisoivia *staattista analyysiä* tekeviä työkaluja
  - ▶ Pythonilla Pylint
  - ▶ JavaScriptilla ESLint
  - ▶ Javalla Checkstyle
- ▶ Myös pilvipalveluna toimivia työkaluja (esim. 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

## Koodin katselointi: GitHub ja pull requestit

- ▶ GitHubin *pull requestit* tarjoavat hyvän työkalun koodikatselointiin
- ▶ 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

31	32	33	34	35	36	37	38	39	40	41	42
@@ -31,6 +32,17 @@ def course_data(course)											
})											
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

## Koodin katselmointi ketterissä menetelmissä

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

## Koodin katselointi ketterissä menetelmissä

- ▶ Toisin kuin Scrum, eXtreme Programming eli XP määrittelee useita sovelluskehityksen käytänteitä
  - ▶ Suuri osa näistä on hyvin tunnettuja *best practiseja*, vietyvä äärimmäiseen (extreme) muotoon
- ▶ Osa käytänteistä tähtää laadun maksimoimiseen, kolmen voidaan ajatella olevan katselmoinnin äärimmäinen muoto

## Pariohjelmointi

- ▶ *Pariohjelmoinnissa* (pair programming) kaksi ohjelmoijaa työskentelee yhdessä samalla koneella

## Pariohjelmointi

- ▶ *Pariohjelmoinnissa* (pair programming) kaksi ohjelmoijaa työskentelee yhdessä samalla koneella
- ▶ Koodia kirjoittava osapuoli toimii *ohjaajana* (driver) ja toinen *navigoijana* (navigator)
  - ▶ Roolia vaihdetaan sopivin väliajoin

## 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 katselmountia

## Pariohjelmoinnin etuja

- ▶ Parantaa ohjelmoijien kuria ja työhön keskittymistä

## Pariohjelmoinnin etuja

- ▶ Parantaa ohjelmoijien kuria ja työhön keskittymistä
- ▶ Hyvä oppimisen väline
  - ▶ ohjelmoijat oppivat toisiltaan erityisesti noviisit kokeneimmilta, järjestelmän tietyn osan tuntee aina useampi ohjelmoija

## Pariohjelmoinnin 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

## Pariohjelmoinnin yleisyys

- ▶ Pariohjelmointi tai useamman ihmisen versio siitä,  
*mob-programming* on melko yleistä

## Pariohjelmoinnin yleisyys

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

## Pariohjelmoinnin yleisyys

- ▶ Pariohjelmointi tai useamman ihmisen versio siitä,  
*mob-programming* on melko yleistä
- ▶ "Määritelmän" mukaista systemaattista pariohjelmosta  
tehdään aika harvassa paikassa aamusta iltaan
- ▶ Yleensä ohjelmoidaan yksin, mutta spontaania pariutumista ja  
ryhmätyymistä 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 mihiin 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

## 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
- ▶ Pariohjelointi tukee yhteisomistajuutta
- ▶ Yhteisomistajuudessa on riskinsä: koodia tuntematon voi saada pahaa jälkeä aikaan
  - ▶ XP eliminoi riskejä testauksiin liittyvillä käytänteillä
- ▶ *Ohjelointistandardi* (coding standards): tiimi määrittelee koodityylin, johon kaikki ohjelmoijat sitoutuvat
  - ▶ Tyyllillä tarkoitetaan nimeämiskäytäntöä, koodin muotoilua ja myös tiettyjä ohjelman rakenteeseen liittyviä seikkoja

## Koodin katselointi ketterissä menetelmissä

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

TAUKO 10 min

Testaus

## Testaus

- ▶ Ohjelmien osoittaminen virheettömäksi on mahdotonta

# Testaus

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

# Testaus

- ▶ 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

# Testaus

- ▶ 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 kokeman laadun parantamiseen
  - ▶ sopiiko sovellus sen käyttötarkoitukseen
  - ▶ toteuttaako halutun toiminnallisuuden
  - ▶ onko riittävän bugiton käytettäväksi

# Testauksien tasot

## Testauksen tasot

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

## Testauksen tasot

- ▶ *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

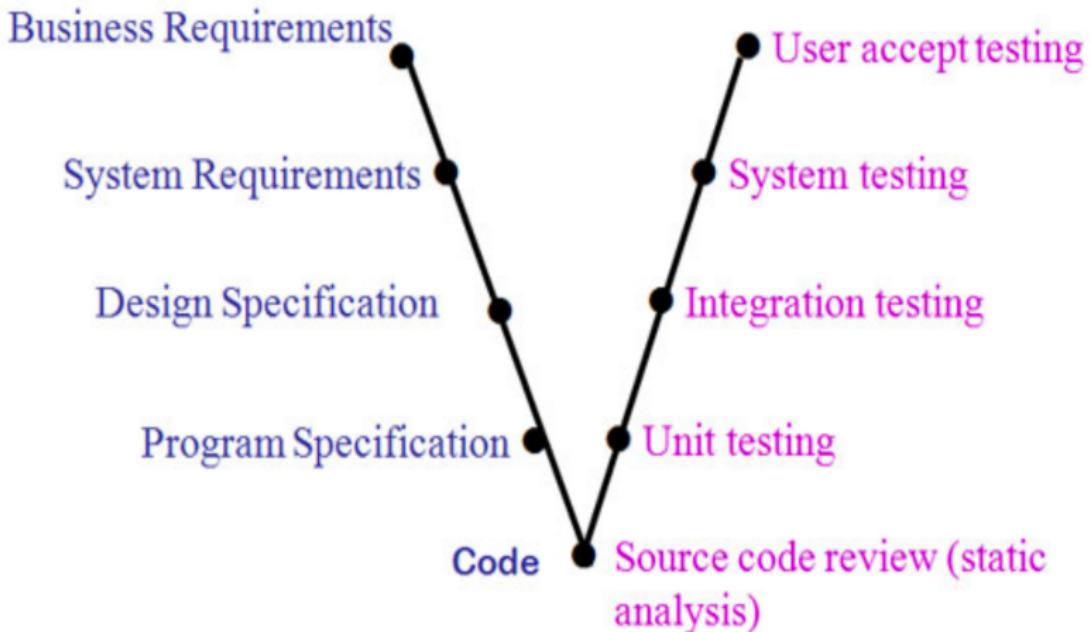
# Testauksen tasot

- ▶ *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?
  - ▶ Tutki järjestelmää kokonaisuudessaan: *end to end -testaus*
  - ▶ Jakautuu useisiin alalajeihin

# Testauksen tasot

- ▶ *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?
  - ▶ Tutkii järjestelmää kokonaisuudessaan: *end to end -testaus*
  - ▶ Jakautuu useisiin alalajeihin
- ▶ *Käyttäjän hyväksymistestaus* (user acceptance testing)
  - ▶ Loppukäyttäjän tuotteelle suorittama testaus

# “V-malli”



## 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 yleensä ilman tietoa järjestelmän sisäisestä rakenteesta eli kyseessä *black box* -testaus

## Järjestelmätestaus

- ▶ Tarkoitus varmistaa, että järjestelmä toimii vaatimuksiin kirjatulla tavalla
  - ▶ Kehittäjäorganisaatio suorittaa järjestelmätestauksen
- ▶ Testaus tapahtuu yleensä 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 yleensä ilman tietoa järjestelmän sisäisestä rakenteesta eli kyseessä *black box* -testaus
- ▶ Testataan järjestelmää saman rajapinnan kautta, jonka kautta järjestelmää käytetään
- ▶ Tarkastelevat sovelluksen toiminnalisuutta kaikilla tasolla käyttöliittymästä sovelluslogiikkaan ja tietokantaan
  - ▶ Käytetään nimitystä *End to End -testaus*

## Järjestelmätestaus

- ▶ Perustuu järjestelmän potentiaaliin käyttöskenaarioihin
  - ▶ user storyinä olevista vaatimuksista helppo muotoilla testejä
  - ▶ jos storyillä hyväksymiskriteeriot, tilanne on vieläkin parempi

## Järjestelmätestaus

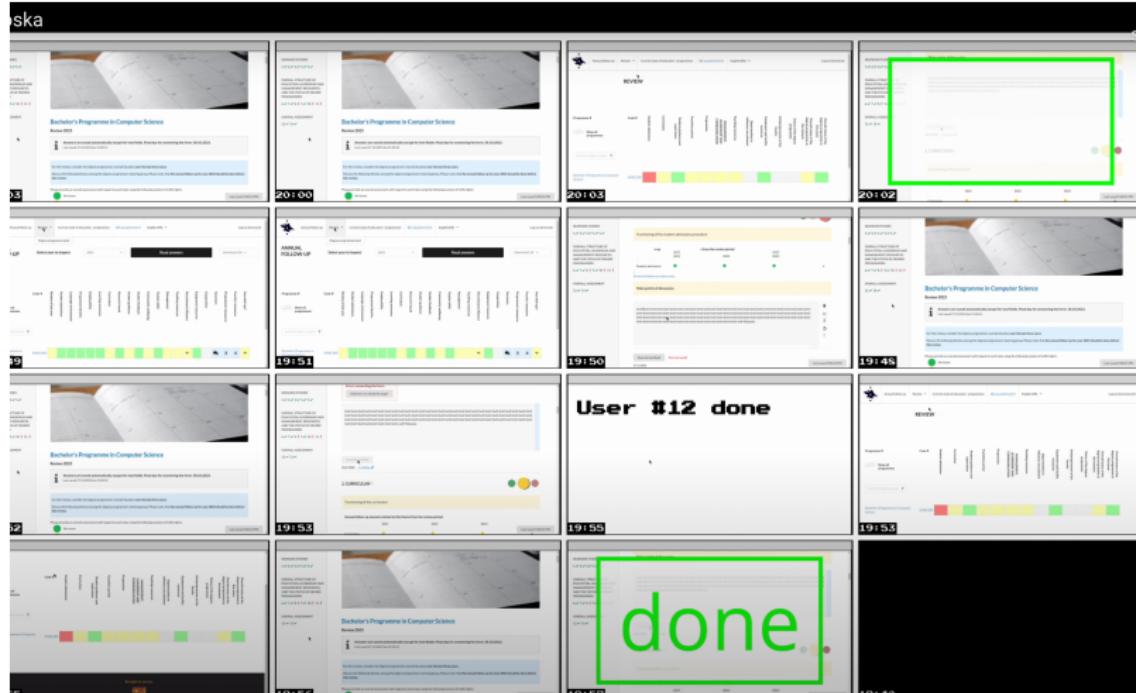
- ▶ Perustuu järjestelmän potentiaalisiin käyttöskenaarioihin
  - ▶ user storyinä olevista vaatimuksista helppo muotoilla testejä
  - ▶ jos storyillä hyväksymiskriteeriot, tilanne on vieläkin parempi
- ▶ Kutsutaan myös *toiminnallisiksi testeiksi* (functional test)

# Järjestelmätestaus

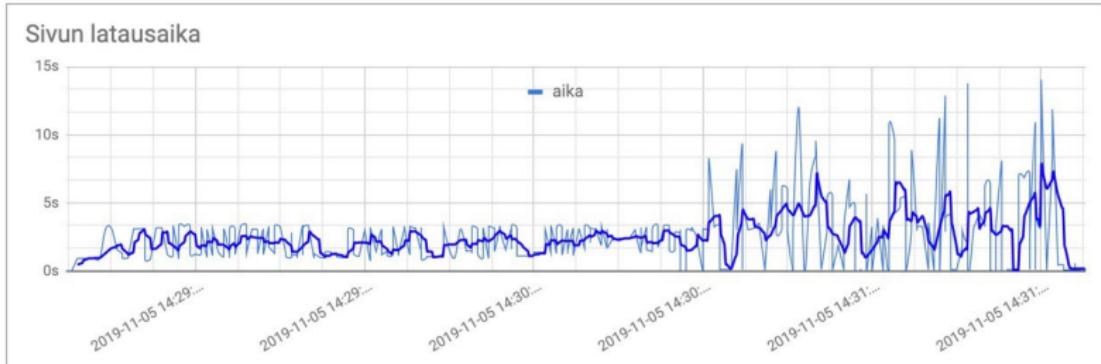
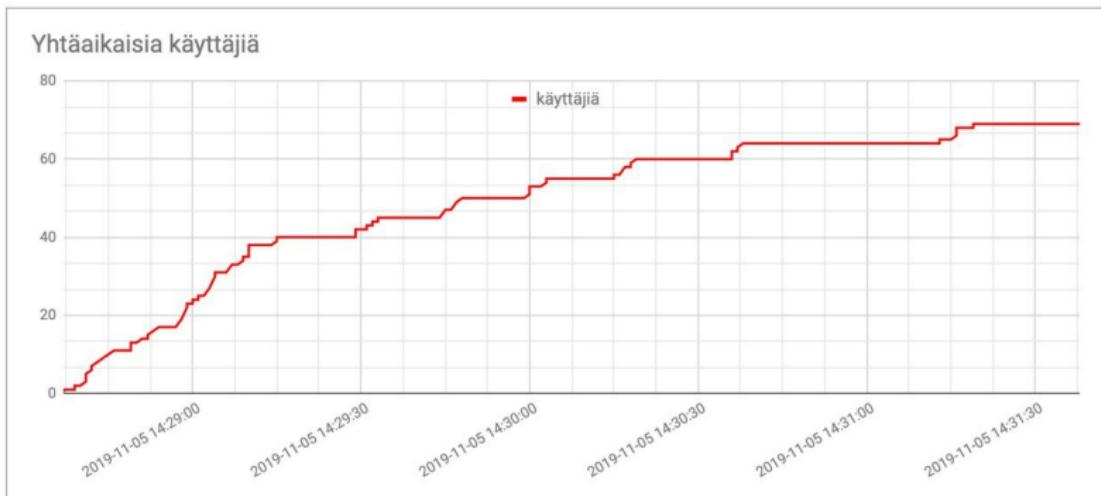
- ▶ Perustuu järjestelmän potentiaalisiin käyttöskenaarioihin
  - ▶ user storyinä olevista vaatimuksista helppo muotoilla testejä
  - ▶ jos storyillä hyväksymiskriteeriot, tilanne on vieläkin parempi
- ▶ Kutsutaan myös *toiminnallisiksi testeiksi* (functional test)
- ▶ Toiminnallisen testauksen lisäksi järjestelmätestaukseen kuuluu mm.
  - ▶ Käytettävyyystestaus
  - ▶ Suorituskykytestaus
  - ▶ Kuormitustestaus
  - ▶ Tietoturvan testaus
  - ▶ Saavutettavuuden testaus

## Kuormitustestaus: esimerkki

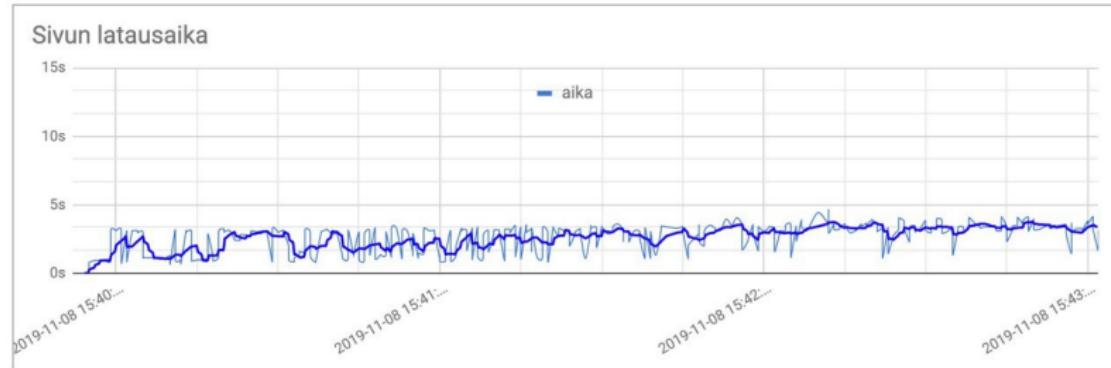
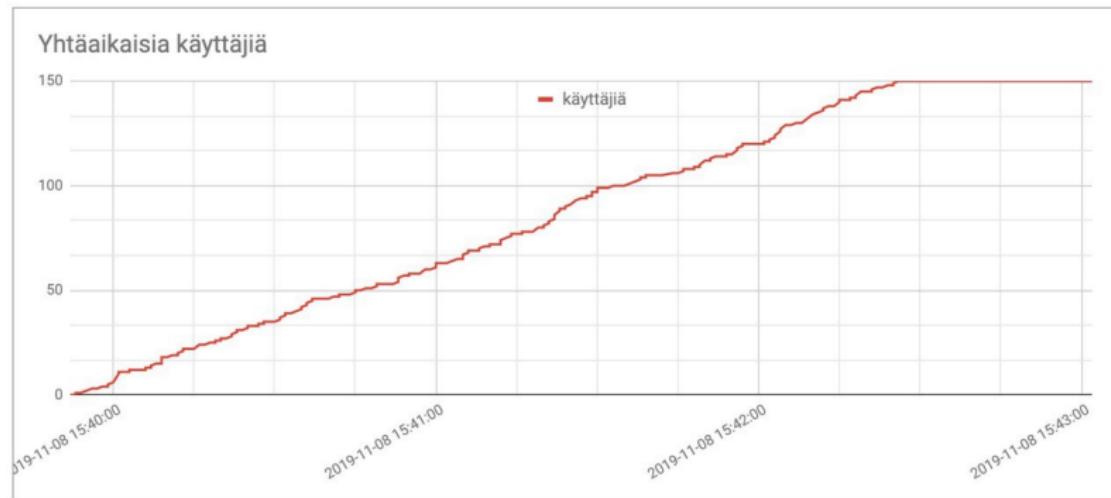
# Kuormitustestaus: esimerkki



# Kuormitustestaus: ennen



# Kuormitustestaus: jälkeen



## Testitapausten valinta

- ▶ Yksi *Testitapaus* testaa toiminnallisuutta joillakin syötteillä

## Testitapausten valinta

- ▶ Yksi *Testitapaus* testaa toiminnallisuutta joillakin syötteillä
- ▶ Kattava testaaminen on mahdotonta ja testaus työlästä
  - ▶ Oleellista löytää kohtuullisen kokoinen *testitapausten joukko*, jonka avulla löytyy suuri määrä virheitä

## Testitapausten valinta

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

## Testitapausten valinta

- ▶ Yksi *Testitapaus* testaa toiminnallisuutta joillakin syötteillä
- ▶ Kattava testaaminen on mahdotonta ja testaus työlästä
  - ▶ Oleellista löytää kohtuullisen kokoinen *testitapausten joukko*, jonka avulla löytyy suuri määrä virheitä
- ▶ Useat syötteet ohjelmiston toiminnan kannalta samanlaisia
  - ▶ nämä muodostavat *ekvivalenssiluokan*
- ▶ Jaetaan syötteet *ekvivalenssiluokkiin* ja tehdään yksi *testitapaus* jokaisesta *ekvivalenssiluokasta*

## Testitapausten valinta

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

## Testitapausten valinta

- ▶ Yksi *Testitapaus* testaa toiminnallisuutta joillakin syötteillä
- ▶ Kattava testaaminen on mahdotonta ja testaus työlästä
  - ▶ Oleellista löytää kohtuullisen kokoinen *testitapausten joukko*, jonka avulla löytyy suuri määrä virheitä
- ▶ Useat syötteet ohjelmiston toiminnan kannalta samanlaisia
  - ▶ nämä muodostavat *ekvivalenssiluokan*
- ▶ Jaetaan syötteet *ekvivalenssiluokkiin* ja tehdään yksi *testitapaus* jokaisesta *ekvivalenssiluokasta*
- ▶ Erityisen kiinnostavia syötearvoja ovat *ekvivalenssiluokkien* väliset *raja-arvot*
- ▶ Henkilötietoja käsittelevä järjestelmä: henkilön iän *ekvivalenssiluokat*?
  - ▶ 0-6, 7-17, 18-65, 66-

# Testisyötteiden valinta: palautussovellus

- Mitä testitapauksia kannattaisi valita palautussovelluksen testaamiseen?

## Create a submission for part2

Mark all exercises you have done (check the box if the exercise is done)

1  2  3  4  5  6  7  8  9  10  11  12

Mark all

Clear all

Used hours (reading the material and completing exercises)

GitHub repository

[https://github.com/mluukkai/put\\_your\\_repository\\_name\\_here](https://github.com/mluukkai/put_your_repository_name_here)

Comments

Pressing send will submit this whole part. Any exercises you have not marked done above for this part can not be marked done later. If you by accident submit the wrong number of exercises contact the course teacher or Discord admins.

Send

Cancel

## Testisyötteiden valinta: palautussovellus

- ▶ Tunnit
  - ▶ tyhjä
  - ▶ negatiivinen
  - ▶ nolla
  - ▶ positiivinen järkevä arvo (esim. 5)
  - ▶ positiivinen mutta epärealistinen (esim. 1000)
  - ▶ merkkejä sisältävä syöte

# Testisyötteiden valinta: palautussovellus

- ▶ Tunnit
  - ▶ tyhjä
  - ▶ negatiivinen
  - ▶ nolla
  - ▶ positiivinen järkevä arvo (esim. 5)
  - ▶ positiivinen mutta epärealistinen (esim. 1000)
  - ▶ merkkejä sisältävä syöte
- ▶ Repositorio
  - ▶ tyhjä
  - ▶ validi repositoriolinkki
  - ▶ epävalidi merkkijono

# Yksikkötestaus

## Yksikkötestaus

- ▶ Kohteena siis yksittäiset metodit ja luokat
  - ▶ *Developer testing*: sovelluskehittäjien vastuulla

## Yksikkötestaus

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

## Yksikkötestaus

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

## Yksikkötestaus

- ▶ 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 laattua
  - ▶ 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

## 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
- ▶ Koska yksikkötestejä joudutaan suorittamaan moneen kertaan, tulee niiden suorittaminen *automatisoida*

## Mitä tulisi (yksikkö)testata?

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

# Mitä tulisi (yksikkö)testata?

- ▶ Automatisoidun testauksen esi-isän 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 (yksikkö)testata?

- ▶ Automatisoidun testauksen esi-isän 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
  - ▶ testien parametrien ekvivalenssiluokat ja raja-arvot päättelävissä koodista

## Ohtuvarasto: tyhjä, puolitäysi, täysi

```
class Varasto:
    def __init__(self, tilavuus, alkusalto = 0):
        self.tilavuus = tilavuus
        self.saldo = alkusalto

    def ota_varastosta(self, maara):
        if maara < 0:
            return 0.0

        if maara > self.saldo:
            kaikki_mita_voidaan = self.saldo
            self.saldo = 0.0
            return kaikki_mita_voidaan

        self.saldo = self.saldo - maara
        return maara
```

## Ohtuvarasto: tyhjä, puolitäysi, täysi

- ▶ Täysi varasto:
  - ▶ otetaan liikaa
  - ▶ otetaan kaikki
  - ▶ otetaan osa
  - ▶ otetaan negatiivinen määärä

## Ohtuvarasto: tyhjä, puolitäysi, täysi

- ▶ Täysi varasto:
  - ▶ otetaan liikaa
  - ▶ otetaan kaikki
  - ▶ otetaan osa
  - ▶ otetaan negatiivinen määrä
- ▶ Puolitäysi varasto
  - ▶ otetaan liikaa
  - ▶ otetaan kaikki
  - ▶ otetaan osa
  - ▶ otetaan negatiivinen määrä

## Ohtuvarasto: tyhjä, puolitäysi, täysi

- ▶ Täysi varasto:
  - ▶ otetaan liikaa
  - ▶ otetaan kaikki
  - ▶ otetaan osa
  - ▶ otetaan negatiivinen määrä
- ▶ Puolitäysi varasto
  - ▶ otetaan liikaa
  - ▶ otetaan kaikki
  - ▶ otetaan osa
  - ▶ otetaan negatiivinen määrä
- ▶ Tyhjä varasto:
  - ▶ ...

## Testauskattavuus

- ▶ Testien hyvyttä voidaan mitata *testauskattavuuden* (test coverage) käsitteellä

# Testauskattavuus

- ▶ Testien hyvyttä voidaan mitata *testauskattavuuden* (test coverage) käsitteellä
- ▶ Muutamaa eri tyyppiä
  - ▶ **rivikattavuus**
  - ▶ **haarautumakattavuus**
  - ▶ ehtokattavuus
  - ▶ polkukattavuus

# Testauskattavuus

- ▶ Testien hyvyttä voidaan mitata *testauskattavuuden* (test coverage) käsitteellä
- ▶ Muutamaa eri tyyppiä
  - ▶ **rivikattavuus**
  - ▶ **haarautumakattavuus**
  - ▶ ehtokattavuus
  - ▶ polkukattavuus
- ▶ Rivi- ja haarautumakattavuudelle hyvä työkalutuki, esim. coverage Pythonille

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
• <a href="#">Varasto(double, double)</a>		0%		0%	4	4	10	10	1	1
• <a href="#">toString()</a>		0%		n/a	1	1	1	1	1	1
• <a href="#">otaVarastosta(double)</a>		62%		50%	2	3	4	8	0	1
• <a href="#">lisaaVarastoon(double)</a>		77%		50%	2	3	2	6	0	1
• <a href="#">Varasto(double)</a>		82%		50%	1	2	1	6	0	1
• <a href="#">paljonkoMahtuu()</a>		100%		n/a	0	1	0	1	0	1
• <a href="#">getSaldo()</a>		100%		n/a	0	1	0	1	0	1
• <a href="#">getTilavuus()</a>		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 haarautumiskohda esim. if ilmaistaan keltaisella

```

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

```

# Ohjelmiston integraatio

## Ohjelmiston integraatio

- ▶ Järjestelmän yksittäiset, erillään yksikkötestatut luokat tulee integroida toimivaksi kokonaisuudeksi

## Ohjelmiston integraatio

- ▶ Järjestelmän yksittäiset, erillään yksikkötestatut luokat tulee integroida toimivaksi kokonaisuudeksi
- ▶ Integroinnin yhteydessä tai sen jälkeen suoritetaan integroititestaus
  - ▶ 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:
  - ▶ rakenteeseen perustuva
  - ▶ toiminnallisuksiin perustuva

## Rakenteeseen perustuva integraatio

- ▶ *Rakenteeseen perustuvassa integraatiossa keskitytään kerrallaan sovelluksen yksittäisten arkkitehtuurillisten komponenttien integrointiin*

## Rakenteeseen perustuva integraatio

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

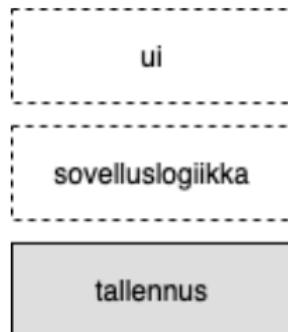
# Rakenteeseen perustuva integraatio

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



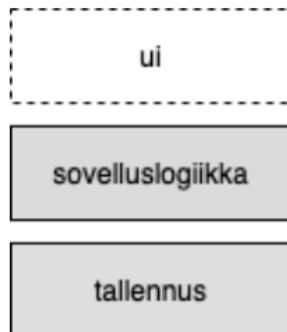
# Rakenteeseen perustuva integraatio

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



# Rakenteeseen perustuva integraatio

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



# Rakenteeseen perustuva integraatio

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

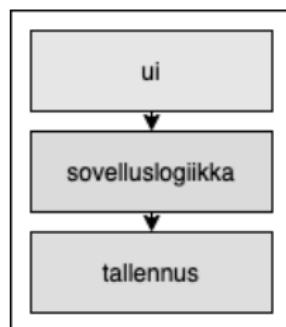
ui

sovelluslogiikka

tallennus

# Rakenteeseen perustuva integraatio

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



## Ominaisuuksiin perustuva integraatio

- ▶ *Ominaisuuksiin perustuvassa integroinnissa liitetään yhteen alikomponentit, jotka toteuttavat järjestelmän loogisen toimintakokonaisuuden*

## Ominaisuuksiin perustuva integraatio

- ▶ *Ominaisuuksiin perustuvassa integroinnissa liitetään yhteen alikomponentit, jotka toteuttavat järjestelmän loogisen toimintakokonaisuuden*
- ▶ Esim. integroidaan kerrallaan kaikki storyn *lisää tuote ostoskoriin* toteutukseen liittyvä koodi
  - ▶ Tämän jälkeen integroidaan seuraavan storyn koodi

# Ominaisuuksiin perustuva integraatio

- ▶ *Ominaisuuksiin perustuvassa integroinnissa liitetään yhteen alikomponentit, jotka toteuttavat järjestelmän loogisen toimintakokonaisuuden*
- ▶ Esim. integroidaan kerrallaan kaikki storyn *lisää tuote ostoskoriiin* toteutukseen liittyvä koodi
  - ▶ Tämän jälkeen integroidaan seuraavan storyn koodi



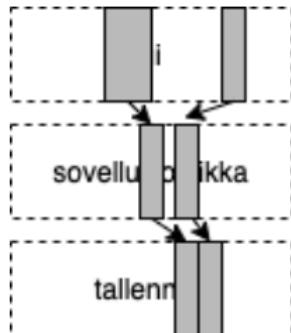
# Ominaisuuksiin perustuva integraatio

- ▶ *Ominaisuuksiin perustuvassa integroinnissa liitetään yhteen alikomponentit, jotka toteuttavat järjestelmän loogisen toimintakokonaisuuden*
- ▶ Esim. integroidaan kerrallaan kaikki storyn *lisäää tuote ostoskoriin* toteutukseen liittyvä koodi
  - ▶ Tämän jälkeen integroidaan seuraavan storyn koodi



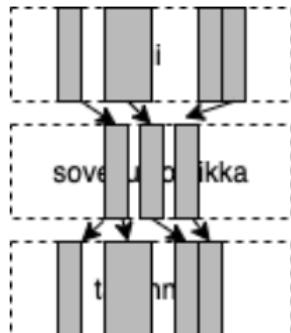
# Ominaisuksiin perustuva integraatio

- ▶ *Ominaisuksiin perustuvassa integroinnissa liitetään yhteen alikomponentit, jotka toteuttavat järjestelmän loogisen toimintakokonaisuuden*
- ▶ Esim. integroidaan kerrallaan kaikki storyn *lisäää tuote ostoskoriin* toteutukseen liittyvä koodi
  - ▶ Tämän jälkeen integroidaan seuraavan storyn koodi



## Ominaisuksiin perustuva integraatio

- ▶ *Ominaisuksiin perustuvassa integroinnissa liitetään yhteen alikomponentit, jotka toteuttavat järjestelmän loogisen toimintakokonaisuuden*
- ▶ Esim. integroidaan kerrallaan kaikki storyn *lisää tuote ostoskoriin* toteutukseen liittyvä koodi
  - ▶ Tämän jälkeen integroidaan seuraavan storyn koodi



# Ohjelmiston integraatio

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

# Ohjelmiston integraatio

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

# Ohjelmiston integraatio

- ▶ Vesipituoksen toimintatapa
  - ▶ Yksittäiset komponentit ohjelmoidaan ja yksikkötestataan erikseen
  - ▶ Tämän jälkeen ne integroidaan, yleensä rakenteeseen perustuen, kerralla yhteen
- ▶ Tämän tyylinen *big bang*-integraatio on osoittautunut todella riskialttiaksi
  - ▶ 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
  - ▶ Samalla tulee huolehtia, että ei rikota jo toimivia osia

## Regressiotestaus

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

# Regressiotestaus

- ▶ Iteratiivisessa ohjelmistotuotannossa, jokainen iteraatio tuottaa ohjelmistoon uusia ominaisuuksia
  - ▶ Samalla tulee huolehtia, että ei rikota jo toimivia osia
- ▶ Testit 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
  - ▶ Samalla tulee huolehtia, että ei rikota jo toimivia osia
- ▶ Testit 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*

Testauksen tasot: kertaus

# Testauksen tasot: kertaus

- ▶ *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?
  - ▶ Tutkii järjestelmää kokonaisuudessaan: *end to end -testaus*
  - ▶ Jakautuu useisiin alalajeihin
- ▶ *Käyttäjän hyväksymistestaus* (user acceptance testing)
  - ▶ Loppukäyttäjän tuotteelle suorittama testaus

# Huomenna

## Huomenna

- ▶ Ketterien menetelmien suosimia testauksen ja laadunhallinnan käytänteitä