

Ohjelmistotuotanto

Syksy 2025

Luento 5

10.11.2025

Kurssipalaute

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

Paja

- ▶ BK107
 - ▶ ma 14.30-16.30
 - ▶ ti 12-16
 - ▶ to 12-16
 - ▶ pe 12-14

Miniprojektit

- ▶ Käynnistyvä tänään!
- ▶ Aloitustilaisuudet
 - ▶ ma 14-16
 - ▶ ti 14-16
 - ▶ ke 10-12
 - ▶ ke 12-14
 - ▶ to 14-16
 - ▶ to 16-18
- ▶ Seuraavat viikot: sprinttien katselmus ja suunnittelu samassa aikaikkunassa
- ▶ Loppudemot
 - ▶ ke 10.12. klo 10-12 B123
 - ▶ to 11.12. klo 10-12 A111

CurreChat

- ▶ Käytettävissä HY:n GPT-chat
 - ▶ <https://curre.helsinki.fi/chat>

- ▶ Käytettävissä HY:n GPT-chat
 - ▶ <https://curre.helsinki.fi/chat>

TKT20006
Ohjelmistotuotanto

27.10.–10.01.2026
[kurssin asetukset](#)

🕒 Tyhjennä keskustelu
🖨️ Kielimalli: gpt-4o
📄 Kurssimateriaali 📄 Kurssimateriaali ✉️ Tallenna sähköpostina
📝 Keskustelun asetukset
❓ Käyttöohje

milloin kurssikoe pidetään

» Kurssimateriaali (gpt-4o)

Kurssikoe pidetään tiistaina 16.12. klo 13–16 luokkahuoneissa A111 ja CHE A110. Muistathan ilmoittautua kokeeseen!

Lähteet hakusanalla 'kurssikoe ajankohta': etusivu.txt,...

Hakutulokset

Hakusanat: kurssikoe ajankohta

1 Kurssilla käsitellään ohjelmistotuotantoprojektienv hallinnan, työvaiheiden ja työvälaineiden perusteita. Erityinen painotus ketterisää...
2 Kokeessa suurin paino tulee olemaan teoriassa ja sen soveltamisessa käytäntöön. Laskareiden teknisimpia asioita, kuten versionhallintaa ei kokees...
Lue lisää

- ▶ Käytettävissä HY:n GPT-chat
 - ▶ <https://curre.helsinki.fi/chat>

TKT20006
Ohjelmistotuotanto
27.10.–10.01.2026
[kurssin asetukset](#)

milloin kurssikoe pidetään

Tyhjennä keskustelu

Kielimalli: gpt-4o

Kurssimateriaali (highlighted)

Tallenna sähköpostina

Keskustelun asetukset

Käyttöohje

Ota yhteyttä tukeen:
opetusteknologia@helsinki.fi

» Kurssimateriaali (gpt-4o)
Kurssikoe pidetään tiistaina 16.12. klo 13–16
luokkahuoneissa A111 ja CHE A110. Muistathan ilmoittautua
kokeseen!

Lähteet hakusanalla 'kurssikoe ajankohta': etusivu.txt,...

1 Kurssilla käsitellään
ohjelmistotuotantoprojektien
hallinnan, työvaiheiden ja
työvälaineiden perusteita. Erityinen
painotus ketterisää...
2 Kokeessa suurin paino tulee
olemaan teoriassa ja sen
soveltamisessa käytäntöön.
Laskareiden teknisimpia asioita,
kuten versionhallintaa ei kokees...

Lue lisää

- ▶ Ei (ehkä) hallusinoi
- ▶ Osin herkkä kysymysten sanamuodon suhteen

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 ei aina ole kirjattu sitä 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
- ▶ Vesiputousmallissa 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.

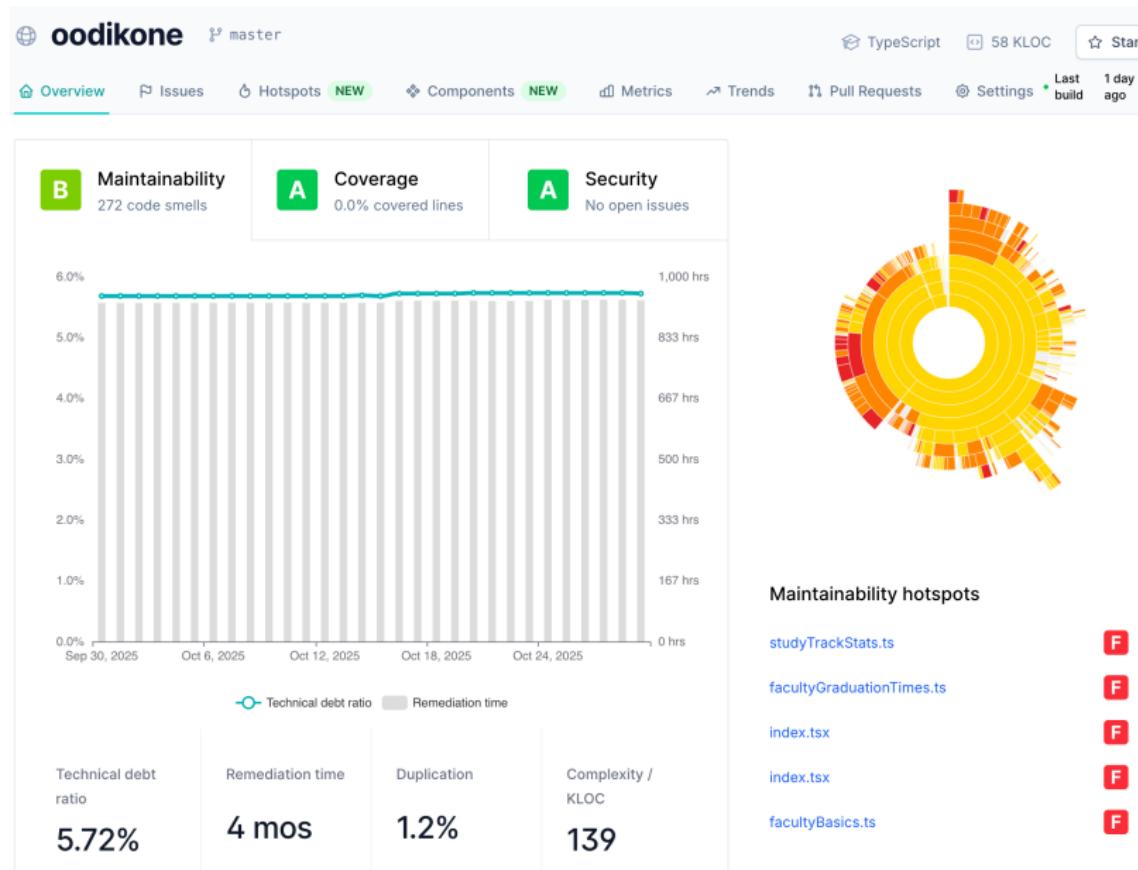
Staattinen analyysi

- ▶ Nykyään on tarjolla katselmointia automatisoivia *staattista analyysiä* tekeviä työkaluja
 - ▶ Pythonilla Pylint
 - ▶ JavaScriptilla ESLint

Staattinen analyysi

- ▶ Nykyään on tarjolla katselointia automatisoivia *staattista analyysiä* tekeviä työkaluja
 - ▶ Pythonilla Pylint
 - ▶ JavaScriptilla ESLint
- ▶ Myös pilvipalveluna toimivia työkaluja (esim. Qlty)
 - ▶ Suorittavat tarkastukset aina kun uutta koodia pushataan GitHubiin
 - ▶ Huomaavat koodin laadun muutoksista, esim. jos koodin kompleksisuus kasvaa muutosten yhteydessä

Esimerkki qlty.sh



Type	File ▾	Size	Loc	Issues	Maintainability	Complexity	Duplication
📁	..						
📁	config						
📁	database	S	1,928	3	A 4 hrs	32	2.7%
📁	environment						
📁	middleware	XS	190	1	A 35 mins	39	0.0%
📁	models	S	1,020	5	A 1 hr	1	6.1%
📁	rapodiff	XS	198	2	A 1 hr	38	0.0%
📁	routes	S	2,205	7	C 4 days	523	2.3%
📁	services	S	8,862	61	D 2 mos	1,892	1.8%
📁	types	XS	42	0	A 0 mins	0	0.0%
📁	util	XS	325	0	A 0 mins	80	0.0%
📁	worker	XS	143	0	A 0 mins	7	0.0%

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 hyväksytäviä, 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](#).

GitHub copilotin tekemä koodin katselointi

► Pull requestin tekemien muutosten yleiskatsaus

Copilot AI left a comment ...

Pull Request Overview

This pull request adds a UI management system for warehouse (Varasto) operations and integrates pre-commit hooks for code quality. The main changes include:

- New interactive CLI interface for managing multiple warehouses
- Integration of pre-commit with pylint for automated code quality checks
- Minor indentation fix in the core warehouse module

Reviewed Changes

Copilot reviewed 3 out of 5 changed files in this pull request and generated 3 comments.

▼ Show a summary per file

File	Description
src/varasto.py	Fixed indentation on line 29 in the <code>lisaa_varastoon</code> method
src/ui.py	Added new VarastoUI class providing a complete CLI interface for warehouse management operations
pyproject.toml	Added pre-commit as a project dependency

Copilot antaa myös detaljoidumpia kommentteja sekä parannusehdotuksia

src/ui.py

Comment on lines +137 to +138

```
137 +     print(f"Täytöaste: {(varasto saldo / varasto.tilavu
138 +
```



Copilot AI 13 minutes ago

...

Division by zero error will occur if `varasto.tilavuus` is 0. Although the `Varasto` class sets `tilavuus` to 0 when negative values are provided, this would cause a runtime error when displaying warehouse details for such cases.

Suggested change

```
137 -     print(f"Täytöaste: {(varasto saldo /
    varasto.tilavuus * 100):.1f}%)"
137 +     if varasto.tilavuus == 0:
138 +         print("Täytöaste: Ei määritetty (tilavuus
    0)")
139 +     else:
140 +         print(f"Täytöaste: {(varasto saldo /
    varasto.tilavuus * 100):.1f}%)"
```

Commit suggestion ▾

Add suggestion to batch

Extreme programmingin katselmoinnin menetelmiä

Extreme programmingin katselmoinnin menetelmiä

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

Extreme programmingin katselmoinnin menetelmiä

- ▶ Toisin kuin Scrum, eXtreme Programming eli XP määrittelee useita sovelluskehityksen käytänteitä
 - ▶ Suuri osa näistä on hyvin tunnettuja parhaita käytänteitä, vietyän 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 pariohjelointia
tehdään aika harvassa paikassa aamusta iltaan

Pariohjelmoinnin yleisyys

- ▶ 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
ryhmätyymistä tapahtuu
 - ▶ erityisesti teknisesti haasteellisissa koodin osissa
 - ▶ tai jos kyse itselle tuntemattomasta osasta koodia

Yhteisomistajuus

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

Yhteisomistajuus

- ▶ *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

Yhteisomistajuus

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

Yhteisomistajuus

- ▶ *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
 - ▶ Tyyllillä tarkoitetaan nimeämiskäytäntöä, koodin muotoilua ja myös tiettyjä ohjelman rakenteeseen liittyviä seikkoja

Yhteisomistajuus

- ▶ *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
 - ▶ 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

- ▶ Ohjelmiston osoittaminen virheettömäksi on mahdotonta

Testaus

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

Testaus

- ▶ Ohjelmiston 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

- ▶ Ohjelmiston 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
 - ▶ toteuttaako halutun toiminnallisuuden
 - ▶ onko riittävän bugiton käytettäväksi
 - ▶ sopiiko sovellus sen käyttötarkoitukseen

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

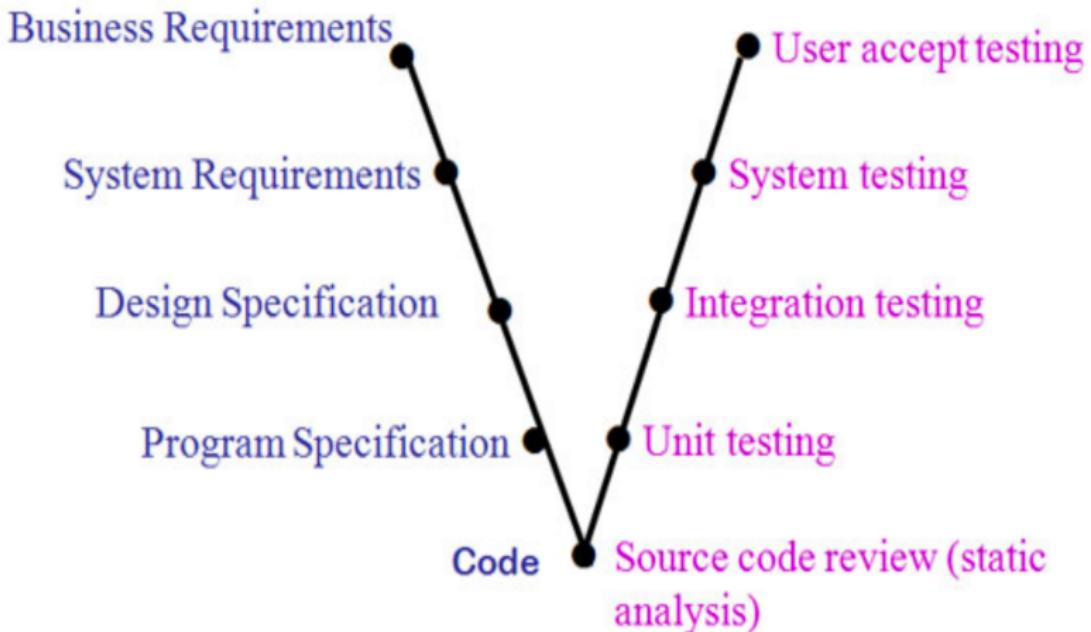
Testauksen tasot

- ▶ *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

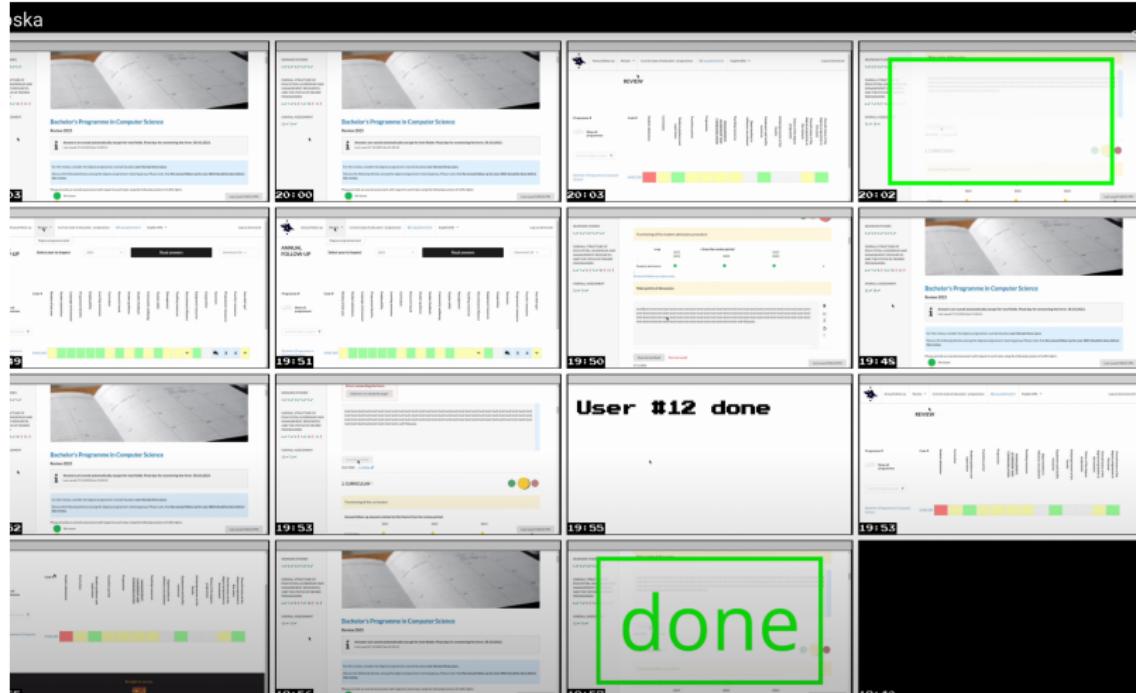
- ▶ Perustuu järjestelmän potentiaalisiin käyttöskenaarioihin
 - ▶ user storyinä olevista vaatimuksista helppo muotoilla testejä
 - ▶ jos storyillä hyväksymiskriteeriot, tilanne on vieläkin parempi
- ▶ Kutsutaan myös *toiminnallisiksi testeiksi* (functional test)

Järjestelmätestaus

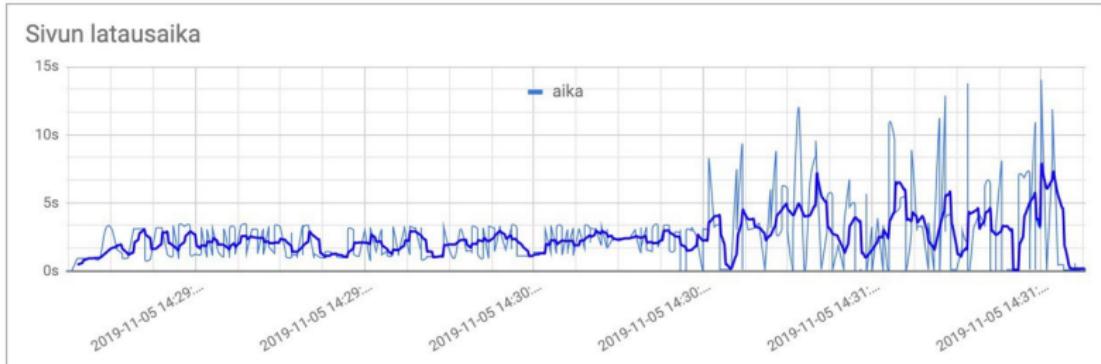
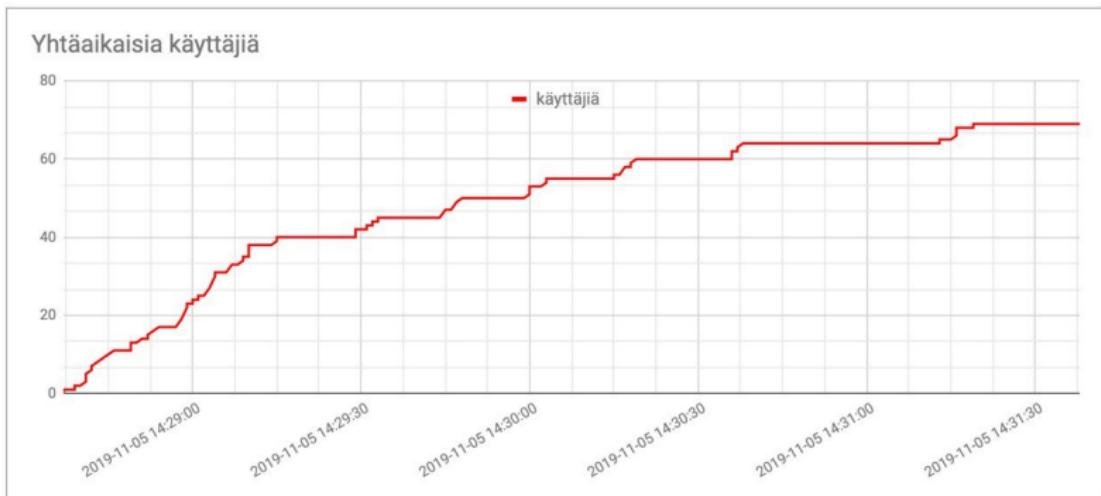
- ▶ Perustuu järjestelmän potentiaalisiin käyttöskenaarioihin
 - ▶ user storyinä olevista vaatimuksista helppo muotoilla testejä
 - ▶ jos storyillä hyväksymiskriteeriot, tilanne on vieläkin parempi
- ▶ Kutsutaan myös *toiminnallisiksi testeiksi* (functional test)
- ▶ Toiminnallisen testauksen lisäksi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

Kuormitustestaus: esimerkki

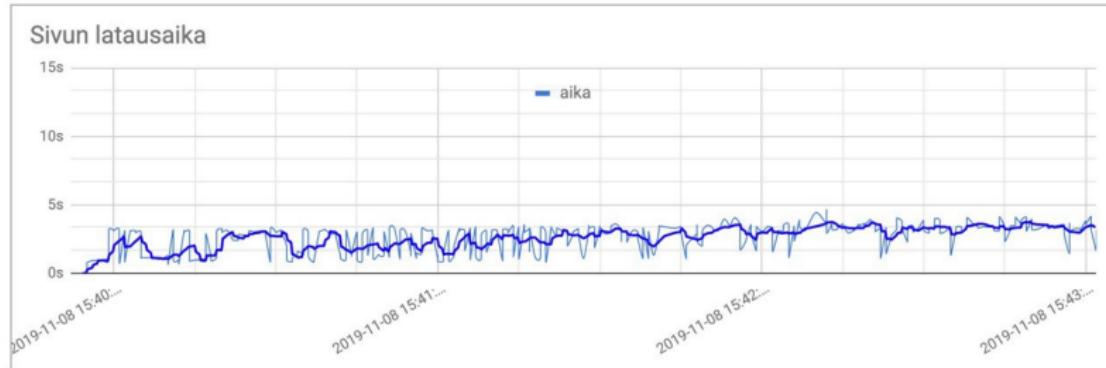
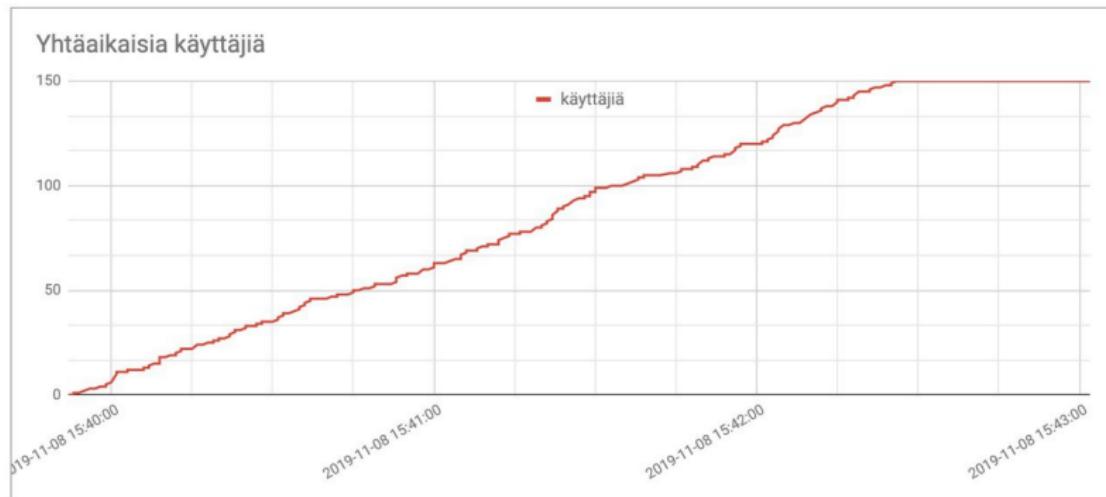
Kuormitustestaus: esimerkki



Kuormitustestaus: ennen



Kuormitustestaus: jälkeen



Saavutettavuuden testaaminen: palautusjärjestelmä

Saavutettavuuden testaaminen: palautusjärjestelmä

https://study.cs.helsinki.fi/stats/courses/ohtu2025/submissions

The following apply to the entire page:

powered by WebAIM

Styles: OFF ON

all courses my submissions aria *aria-current="page" admin fsadmin suotar Matti Luukkainen - mluukkai

Details

Errors	16	Contrast Errors	2
Features	1	Structure	22
ARIA	2		

AIM Score: 4.9 out of 10

3 Errors

- 3 X Empty table header

16 Contrast Errors

16 X Very low contrast

2 Alerts

- 1 X No page regions

My submissions for course Ohjelmistotuotanto 2025

Miniprojektiin ilmoittautuminen käynnissä, deadline 8.11. Lue lisää [täältä](#)

Create submission for part 3

part	exercises	hours	GitHub	comment	solutions
1	17	12	show https://github.com/mluukkai/ohtuvarasto	show	show
2	13	1	show https://github.com/mluukkai/ohtu	show	show
total	30	13	show	show	show

Code

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*

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

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

Testauskattavuus

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

Testauskattavuus

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

Testauskattavuus

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

Coverage for `src/varasto.py`: 79%

33 statements 27 run 6 missing 0 excluded 4 partial

« prev ^ index » next coverage.py v7.10.7, created at 2025-11-01 17:23 +0200

```
1 class Varasto:
2     def __init__(self, tilavuus, alku_saldo = 0):
3         self.tilavuus = self._tilavuus_alussa(tilavuus)
4         self.saldo = self._saldo_alussa(alku_saldo)
5
6     def paljonko_mahtuu(self):
7         return self.tilavuus - self.saldo
8
9     def lisaa_varastoon(self, maara):
10        if maara < 0:
11            return
12        if maara <= self.paljonko_mahtuu():
13            self.saldo = self.saldo + maara
14        else:
15            self.saldo = self.tilavuus
16
17    def ota_varastosta(self, maara):
18        if maara < 0:
19            return 0.0
20        if maara > self.saldo:
21            kaikki_mita_voidaan = self.saldo
22            self.saldo = 0.0
23
24        return kaikki_mita_voidaan
25
26        self.saldo = self.saldo - maara
27
28    return maara
29
```

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 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:
 - ▶ rakenteeseen perustuva
 - ▶ toiminnallisuksiin perustuva

Rakenteeseen perustuva integraatio

- ▶ Integroidaan arkkitehtuurikomponentti kerrallaan

Rakenteeseen perustuva integraatio

- ▶ Integroidaan arkkitehtuurikomponentti kerrallaan
- ▶ Verkkokaupassa: sovelluslogiikan luokat, käyttöliittymän luokat ja tietokantarajapinta ensin omina kokonaisuuksinaan

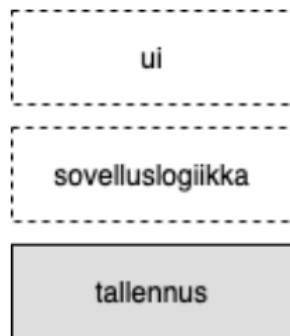
Rakenteeseen perustuva integraatio

- ▶ Integroidaan arkkitehtuurikomponentti kerrallaan
- ▶ Verkkokaupassa: sovelluslogiikan luokat, käyttöliittymän luokat ja tietokantarajapinta ensin omina kokonaisuuksinaan



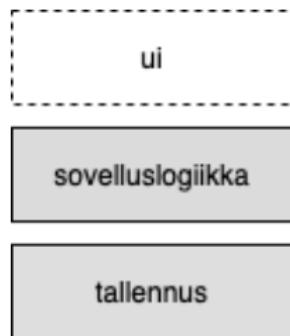
Rakenteeseen perustuva integraatio

- ▶ Integroidaan arkkitehtuurikomponentti kerrallaan
- ▶ Verkkokaupassa: sovelluslogiikan luokat, käyttöliittymän luokat ja tietokantarajapinta ensin omina kokonaisuuksinaan



Rakenteeseen perustuva integraatio

- ▶ Integroidaan arkkitehtuurikomponentti kerrallaan
- ▶ Verkkokaupassa: sovelluslogiikan luokat, käyttöliittymän luokat ja tietokantarajapinta ensin omina kokonaisuuksinaan



Rakenteeseen perustuva integraatio

- ▶ Integroidaan arkkitehtuurikomponentti kerrallaan
- ▶ Verkkokaupassa: sovelluslogiikan luokat, käyttöliittymän luokat ja tietokantarajapinta ensin omina kokonaisuuksinaan

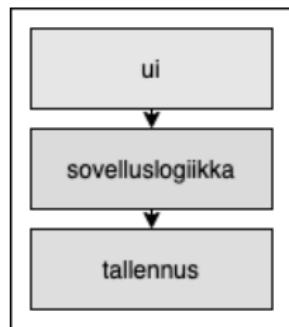
ui

sovelluslogiikka

tallennus

Rakenteeseen perustuva integraatio

- ▶ Integroidaan arkkitehtuurikomponentti kerrallaan
- ▶ Verkkokaupassa: sovelluslogiikan luokat, käyttöliittymän luokat ja tietokantarajapinta ensin omina kokonaisuuksinaan



Ohjelmiston integraatio

- ▶ Rakenteeseen perustuva integraatio on vesiputuoksen toimintatapa

Ohjelmiston integraatio

- ▶ Rakenteeseen perustuva integraatio on vesiputuksen toimintatapa
- ▶ Tämän tyylinen *big bang* -integraatio on osoittautunut todella riskialttiaksi
 - ▶ Seurauksena usein ns. **integraatiohelvetti**

Ohjelmiston integraatio

- ▶ Rakenteeseen perustuva integraatio on vesiputuoksen toimintatapa
- ▶ Tämän tyylinen *big bang* -integraatio on osoittautunut todella riskialttiaksi
 - ▶ Seurauksena usein ns. **integraatiohelvetti**
- ▶ Moderni ohjelmistotuotanto suosii ns. *jatkuvaa integraatiota*
 - ▶ Hyvin tiheässä tähdissä tapahtuvaa ominaisuuksiin perustuva integroitinta

Ominaisuuksiin perustuva integraatio

- ▶ Toiminnallisuus kerrallaan

Ominaisuuksiin perustuva integraatio

- ▶ Toiminnallisuus kerrallaan
- ▶ Esim. integroidaan kerrallaan kaikki storyn *lisää tuote ostoskoriin* toteuttava koodi
 - ▶ Tämän jälkeen integroidaan seuraavan storyn koodi

Ominaisuuksiin perustuva integraatio

- ▶ Toiminnallisuus kerrallaan
- ▶ Esim. integroidaan kerrallaan kaikki storyn *lisää tuote ostoskoriiin* toteuttava koodi
 - ▶ Tämän jälkeen integroidaan seuraavan storyn koodi



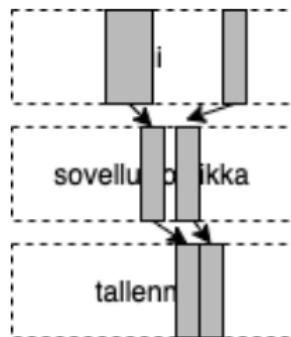
Ominaisuuksiin perustuva integraatio

- ▶ Toiminnallisuus kerrallaan
- ▶ Esim. integroidaan kerrallaan kaikki storyn *lisäää tuote ostoskoriin* toteuttava koodi
 - ▶ Tämän jälkeen integroidaan seuraavan storyn koodi



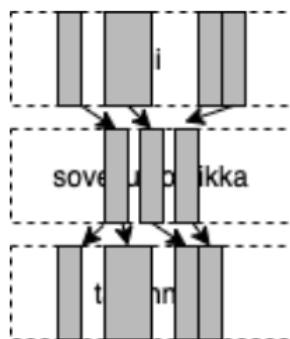
Ominaisuuksiin perustuva integraatio

- ▶ Toiminnallisuus kerrallaan
- ▶ Esim. integroidaan kerrallaan kaikki storyn *lisäää tuote ostoskoriin* toteuttava koodi
 - ▶ Tämän jälkeen integroidaan seuraavan storyn koodi



Ominaisuuksiin perustuva integraatio

- ▶ Toiminnallisuus kerrallaan
- ▶ Esim. integroidaan kerrallaan kaikki storyn *lisäää tuote ostoskoriin* toteuttava koodi
 - ▶ Tämän jälkeen integroidaan seuraavan storyn koodi



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