Aineopintojen harjoitustyö: Ohjelmistotekniikka (4 op)

Aloitusluento 9.3.2026

Kurssista

Opettajat: Kai Korpimies ja joukko assareita

Pieni harjoitustyö Python-kielellä

Alussa muutama harjoitus, ei tenttiä

Esitietoina Ohjelmoinnin jatkokurssi, Tietokantojen perusteet, Tietokannat ja web-ohjelmointi tai vastaavat tiedot sekä Tietokone työvälineenä

Kurssimateriaali: https://ohjelmistotekniikka-hy.github.io/

Osaamistavoitteita

"Kurssin suoritettuasi

- tunnet **ohjelmistotuotantoprosessin vaiheet**,
- olet tietoinen **vesiputousmallin ja ketterän ohjelmistotuotannon** luonteesta,
- osaat soveltaa **versionhallintaa** osana ohjelmistokehitystä,
- osaat soveltaa **UML-mallinnustekniikkaa** vaatimusmäärittelyssä ja ohjelmiston suunnittelussa tarkoituksenmukaisella tavalla
- tunnet ohjelmiston **testauksen** eri vaiheet
- osaat soveltaa **automatisoitua testausta** yksinkertaisissa ohjelmistoprojekteissa
- tunnet tärkeimpiä **ohjelmiston suunnitteluperiaatteita** ja osaat soveltaa niitä yksinkertaisissa projekteissa"

Kurssipalautteesta

"Syksyn palautteessa toteutustapa sai arvosanakseen 4.2/5, materiaalit 4.0/5, arviointimenetelmät 4.2/5 ja osaamistavoitteiden selkeys 4.5/5. Työmäärä nähtiin yleensä sopivaksi (76 %) tai melko raskaaksi (18 %).

Sanallisissa palautteissa oli monia hyviä huomioita, joskaan mikään yksittäinen asia ei noussut erityisen korostetusti esille. Huomioista oli suurta apua kun kurssimateriaalia päivitettiin kevään kurssia varten."

Ohjelmistotuotanto

Tarvitaanko seuraavissa tilanteissa erilaisia käytäntöjä?

1. kirjoitat pienen ohjelman ohjelmointikurssin harjoituksissa
2. toteutat itse vähän isomman projektin
3. osallistut avoimen lähdekoodin projektiin, jossa on monta ohjelmoijaa tai
4. osallistut ohjelmistotuotantoon jossain ohjelmistotalossa

Monellako kokemusta tilanteista 2 - 4?

Ohjelmistotuotanto

Pohdittavaa mm.:

- järjestelmällisyys
- versioiden ja riippuvuuksien hallinta, koodin laadunvarmistus, testaus, automatisointi….
- yhteistyö, työnjako, toimivat käytännöt
- ohjelmiston ei-toiminnalliset vaatimukset (tehokkuus, turvallisuus, käytettävyys ym.)
- asiakkaan/käyttäjän näkökulma

Ohjelmistotuotannon vaiheet

Vaatimusmäärittely

Suunnittelu

Toteutus

Testaus

Ylläpito

Vaatimusmäärittely (requirements analysis)

Mitä ohjelmiston tai järjestelmän tulee tehdä

- esim. tarjottavat palvelut, suorituskyky, kustannukset
- ei niinkään vastauksia kysymykseen "miten"

Käyttäjän/asiakkaan näkökulma

Huomioidaan toimintaympäristö ja toteutusteknologia

Dokumentoidaan

Suunnittelu (system design)

Miten onnistuu? Järjestelmän rakenne, toiminta, tietosisältö?

Arkkitehtuurisuunnittelu keskittyy ohjelmiston yleisrakenteeseen:

- minkälaisista "suuremmista osista" ohjelmisto koostuu ja minkälaisia keskinäisiä suhteita näiden osien välillä on
- rajapinnat, riippuvuudet

Oliosuunnittelu tarkentaa:

- mitä luokkia, miten toimivat yhdessä?

Dokumentoidaan

Toteutus

Ohjelmointi etusijalla

Ei yleensä ole pelkkää mekaanista koodausta määrittelyn ja suunnittelun mukaisesti

Yleensä hyödynnetään monenlaisia välineitä, valmiita kirjastoja, sovelluskehyksiä…

Testaus

Tavoitteena on löytää virheet jotka estävät ohjelmistoa toimimasta määritysten mukaisesti.

Yksikkötestaus (unit testing): täyttävätkö yksittäiset luokat vaatimuksensa?

Integraatiotestaus: toimivatko luokat yhdessä oikein?

Järjestelmätestaus: toimiiko järjestelmä kokonaisuudessaan (ml. käyttöliittymä) vaatimusten mukaisesti?

Vesiputousmalli (waterfall model)

Edetään "top down" -tyyliin em. vaiheet yksi kerrallaan

- eri vaiheet ehkä eri tahojen vastuulla

Periaatteessa suoraviivaista, mutta:

- yleensä aiempiin vaiheisiin on pakko vielä palata -> lisätyötä!
  - käyttäjien toiveet eivät usein selviä heti alussa
  - muutenkaan ei ehkä vielä tiedetä tarpeeksi
  - toimintaympäristö voi muuttua
- testaus alkaa kovin myöhään

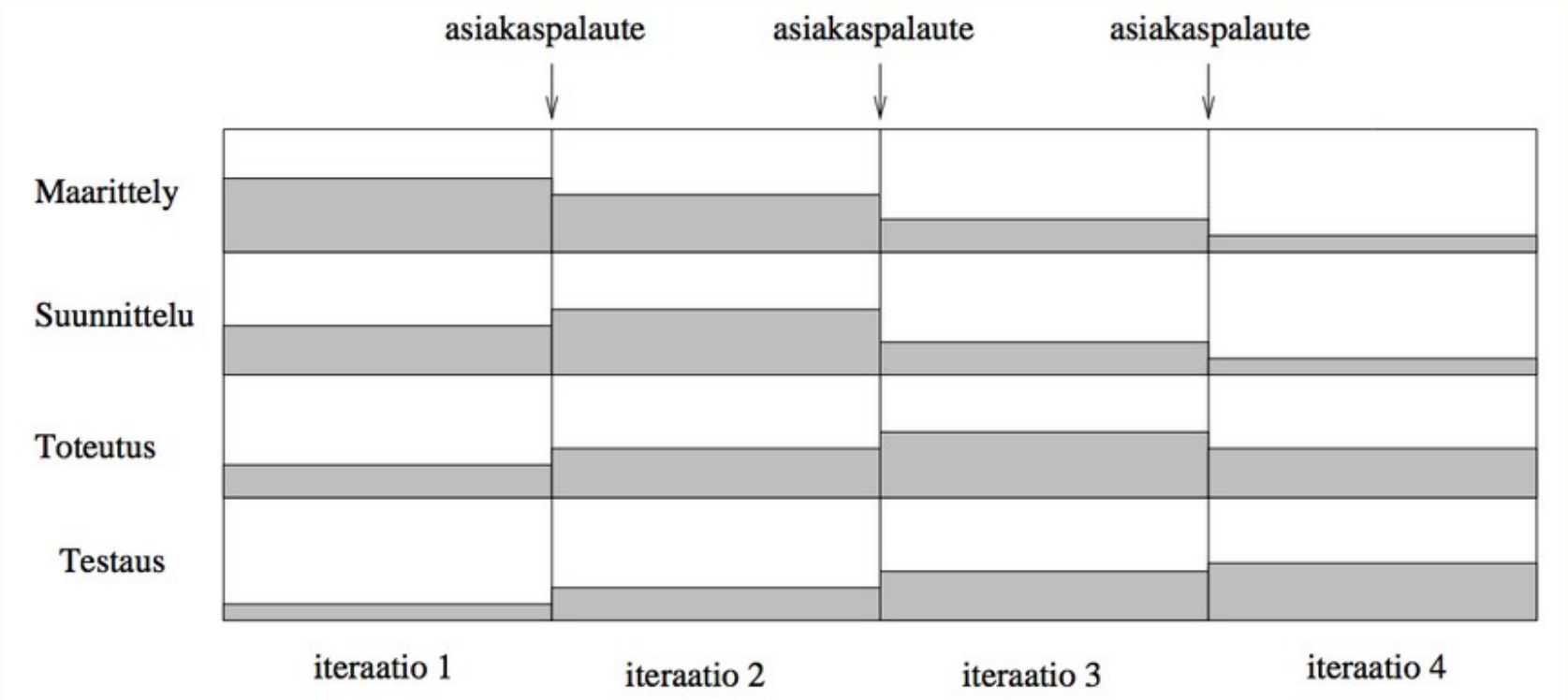
Miten näitä ongelmia voisi ratkaista?

Ketterät (agile) menetelmät

Alussa vaatimukset pääpiirteissään ja arkkitehtuurin alustavaa hahmottelua, sitten ohjelmallista iterointia ja inkrementaalista etenemistä

- jokaisessa iteraatiossa suunnitellaan, toteutetaan ja testataan pieni osa ohjelmistoa
- asiakas voi kokeilla ohjelmistoa jokaisen iteraation jälkeen

Vaatimuksia ja arkkitehtuuria voidaan tarkentaa ja muuttaakin



Edetään kurssilla ketterästi…

Esimerkki: TodoApp

Kurssin "referenssisovellus"

Havainnollistaa, toimii mallina

https://github.com/ohjelmistotekniikka-hy/python-todo-app

TodoApp: vaatimusmäärittelyä

Sovelluksen avulla voi pitää kirjaa tekemättömistä töistä (todoista)

Kaksi käyttäjäroolia:

- tavalliset käyttäjät
- ylläpitäjät

Mitä toiminnallisuutta nämä tarvitsevat?

Tavalliset käyttäjät voivat:

- luoda järjestelmään käyttäjätunnuksen
- kirjautua järjestelmään, jonka jälkeen he voivat:
- nähdä omat tekemättömät työnsä (todot)
- luoda uusia todoja
- merkitä todoja tehdyiksi, jolloin ne häviävät listalta

Ylläpitäjät voivat:

- tarkastella tilastoja
- poistaa käyttäjätunnuksia

Toimintaympäristö:

- toteutetaan Pythonilla ja käyttöliittymä Tkinterillä
- käyttäjä- ja todotiedot tallennetaan paikallisen koneen levylle
- ohjelmiston tulee toimia osaston Linux-työasemilla

Käyttöliittymä:

- vapaamuotoinen luonnos (piirros) valikoista ja niiden välisistä siirtymistä

Tehokkuuteen, kustannuksiin ym. ei oteta kantaa..

TodoApp: suunnittelusta

Kurssimateriaalissa esitellään muutamia ohjelmistotuotannossa hyödyllisiä periaatteita, joita TodoApp:issakin on noudatettu. Esimerkiksi:

- *käyttöliittymä on erotettu sovelluslogiikasta* (vrt. kerrosarkkitehtuurit)
- luokat, metodit ja funktiot toteuttavat ns. *single responsibility* -periaatetta: yhden komponentin ei tule tehdä liian montaa asiaa

TodoApp: yksikkötestaus

Yksikkötestauksessa luokille tehdään **testiluokkia**, jotka sisältävät testimetodeja

- tyypillisesti näitä metodeita kertyy paljon

TodoAPP:in testiluokat hakemistossa python-todo-app/src/tests/

- esim. luokka todo_service.py ja vast. testiluokka todo_service_test.py

Miksi yksikkötestausta testiluokkineen?

Graafista käyttöliittymää ei kurssilla tarvitse yksikkötestata

Kurssin aikataulu

viikko 1: palautuksen deadline ti 17.3.

viikko 2: palautuksen deadline ti 24.3.

viikko 3: palautuksen deadline ti 31.3.

viikkoharjoitukset ("laskarit")

Pääsiäistauko

viikko 4: palautuksen deadline ti 14.4.

viikko 5: palautuksen deadline ti 21.4.

viikko 6: palautuksen deadline ti 28.4.

viikko 7: loppupalautuksen deadline su 10.5.

harjoitustyö

Deadlinet aina klo 23:59. *Myöhästyneitä palautuksia ei huomioida*.

Pisteytys ja hieman palautetta aina vast. viikon loppuun mennessä (paitsi viimeisen deadlinen kohdalla)

Välineitä

Github: kaikki palautukset tänne. Myös kurssimateriaali täällä.

Labtool: pisteet ja palaute näkyvät täällä viikoittain

Moodle: Suuria kielimalleja koskeva kysely, palautepisteen haku. Täällä näet lopulliset pisteesi tammikuussa. Täällä voit myös mm. osallistua ylimääräiseen koodikatselmointiin.

Unittest ja Coverage yksikkötestaukseen

Docstring koodin kommentointiin

Poetry ja Invoke riippuvuuksien hallinta, virtuaaliympäristöt, koodin ja testien suorittaminen

Pylint koodaustyylin laadunvarmistukseen

1. viikko

Kurssimateriaalissa harjoituksia, aiheina **komentorivi, Git ja GitHub**

Tämän tulisi olla tuttua aiemmilta kursseilta (mm. Tietokone työvälineenä); jos näin, voit sivuuttaa

Lue komentorivi- ja versionhallintaharjoitusten tehtävänannot

Jos tuntuu yhtään vieraalta, tee harjoitukset käytännössä!

- tästä ei saa pisteitä, mutta ***tarvitset näitä asioita jatkossa***

Jos et tee komentorivi- ja versionhallintaharjoituksia:

Kurssimateriaalissa on tehtävänanto, jossa pyydetään mm. luomaan GitHub-repositorio ja lisäämään sinne pari tiedostoa

- jos et osaa suorittaa tehtävänannossa pyydettyjä asioita, toimi siis komentorivi- ja versionhallintaharjoitusten ohjeiden mukaisesti

Tutustu jo alustavasti seuraavan viikon kurssimateriaaliin

- sivun loppupuolelta löydät ohjeita harjoitustyön aiheenvalintaan ja alustavan määrittelydokumentin kirjoittamiseen
- mikään ei estä jo käytännössä tekemästä 2. viikon harjoituksia ja/tai harjoitustyösi määrittelydokumenttia. Ne kuitenkin arvioidaan vasta kurssin toisen viikon jälkeen.

Teethän myös **lähtötasotestin** ensimmäisen viikon aikana

- Moodlessa, muutama monivalintakysymys, varsin nopeasti tehtävissä
- tarvitaan suurten kielimallien käyttöä koskevasta kyselystä tarjolla olevaan pisteeseen
- tulos ei vaikuta mitenkään kurssin arvosanaan

Harjoitustyön vaatimusmäärittelystä

Millainen aihe kiinnostaa?

Vältä kovin "eeppistä" vaatimusmäärittelyä

- *perustoiminnallisuuden* tulisi olla toteutettavissa nopeasti
- perustoiminnallisuutta tulisi voida *laajentaa* helposti
- voit sijoittaa joitain toimintoja kategoriaan "jos aikaa jää"

Liian suppeakaan aihe ei saisi olla

- TodoApp:ia vastaava laajuus riittää

Harjoitustyön vaatimusmäärittelystä

Mieti, minkälaisia mahdollisia ongelmia työssä saattaa tulla vastaan

- osaatko jo toteuttaa kaiken vai joudutko opettelemaan paljon uutta? (Esim. TkInter?)

Työn ei siis tarvitse välttämättä olla kovin laaja tai mutkikas

Arvioinnissa:

- kurssista max. 62 p.; työn toteutuksesta voi saada max. 24 p.
- työn laajuus vaikuttaa 4 p. verran; TodoAppin laajuus on 2 p.
- kompleksisuudesta (onko monimutkaistakin laskentaa, epäsuoraviivaista logiikkaa tms.) tarjolla 3 p. TodoApp: 1 p.
- myös mm. käyttöliittymän ja tietojen tallennuksen laajuus vaikuttaa (onko esim. montakin näyttöä ja käyttöliittymäelementtiä tai taulua ja minkälaisia operaatioita)

Harjoitustyön vaatimusmäärittelystä

Toisaalta painotamme seuraavaa:

- toimivuus ja varautuminen virhetilanteisiin
- ohjelman selkeä ja tarkoituksenmukainen rakenne ja toteutus, ml. luokkien vastuut
- laajennettavuus ja ylläpidettävyys

Esim. seuraavat eivät ole tällä kurssilla tärkeitä:

tehokkuus, grafiikka, tietoturva, tekoäly…

Harjoitustyöltä odotetaan mm.

Selkeää arkkitehtuuria

Sovelluslogiikasta erotettua käyttöliittymää

- mieluiten graafista

Tietojen tallennusta

- tiedostoihin tai vielä paremmin tietokantaan

*Valitse aihe, jossa on **riittävästi sovelluslogiikkaa**: älä keskity liikaa käyttöliittymään tai tietojen tallennukseen*

Rajauksia

Ohjelmointikieli on **Python**, GUI-toteutukseen suositellaan Tkinteriä tai Pygamea

Työn tulee toimia ja olla testattavissa osaston Linux-koneilla ja yliopiston **Linux-virtuaalityöasemalla**, Cubbli Linux -etätyöpöydällä

- ohjelmointiympäristöksi *suositellaankin* Linuxia, vaikkakin myös Windowsilla sinänsä tulisi onnistua

Ei web-sovelluksia

Esimerkkejä aiheista

Kortistointi, kirjanpito, suunnittelu:

kirjakortisto, budjetointi, opinnot, kuntoilu, laihdutus, sää, sähkönkulutus..

Opetus- ja harjoitteluohjelmat: kielet, matematiikka...

Simulointi: soittimet, erilaiset laskimet...

Editorit: tekstieditorit, HTML-editorit...

Omaan tieteenalaan ja opintoihin liittyvät ohjelmat

Esimerkkejä aiheista

Pelit: reaaliaikaisia tai vuoropohjaisia

- monenlaiset "arcade-tyyppiset" pelit
- lautapelit: tammi, kaksin pelattava shakki ym.
- korttipelit pasianssista pokeriin
- ristinolla, sudoku, muistipelit, tietovisat...

Tai sitten jotain ihan muuta.

Dokumentaatio

Seuraa materiaalin ohjeita

Voit myös katsoa TodoApp:in dokumentaatiota: (vähintään) sen tasoista odotetaan

- ei suoraa kopiointia…

Koodin muuttujat, luokat ja metodit kirjoitetaan englanniksi, dokumentaatio suomeksi tai englanniksi

2. viikko

Harjoituksia: **Poetry, testaus, testikattavuus**

- varaa aikaa näihin!

Harjoitustyön **määrittelydokumentti**

Aloita harjoitustyön tuntikirjanpito viimeistään nyt

3. viikko

Harjoituksia **luokka- ja sekvenssikaavioista**

Harjoitustyössä mm.

- toteutetaan toiminnallisuutta
- otetaan Poetry käyttöön
- aloitetaan testaus ja generoidaan testikattavuusraportti
- varmistutaan, että toimii osaston koneilla (virtuaalityöasemassa)

UML-kaavioista

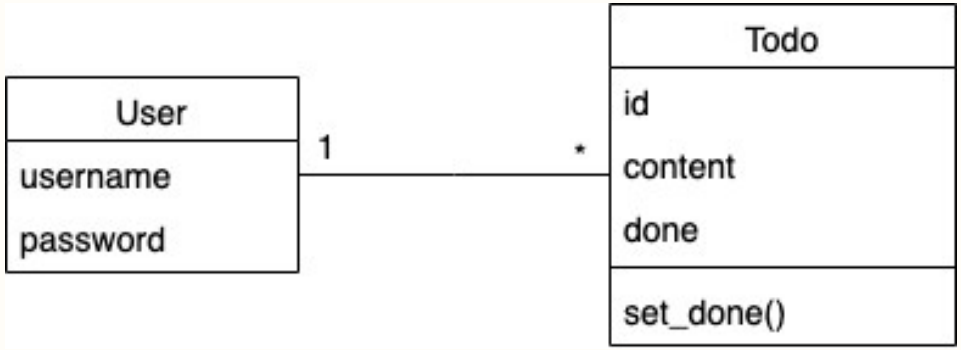
Unified Modeling Language (1996): joukko ohjelmistojen rakennetta ja käyttäytymistä kuvaavia kaavioita

- näistä tällä kurssilla **luokka-, pakkaus- ja sekvenssikaaviot**
- havainnollistamiseen suunnittelussa ja myöhemmin
- laajalti ymmärrettyjä
- UML ei ole sidoksissa ohjelmointikieliin tai prosessimalleihin

Kaavioita voi piirtää monin eri tavoin; yksi kätevä tapa on *Mermaid-syntaksi*

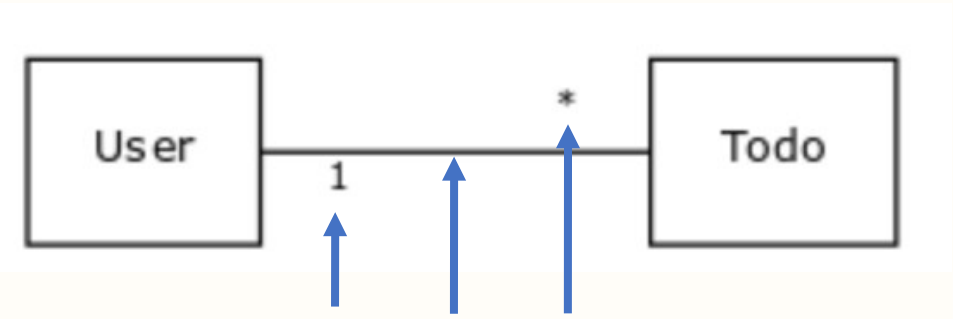
Luokkakaaviot kuvaavat ohjelmiston luokkia ja niiden välisiä yhteyksiä

```
class User:
    def __init__(self, username, password):
        self.username = username
        self.password = password
import uuid
```




```
class Todo:
    def __init__(self, content, done=False, user=None, todo_id=None):
        self.content = content
        self.done = done
        self.user = user
        self.id = todo_id

    def set_done(self):
        self.done = True
```



*Kaaviossa tärkeitä **luokkien suhteet, ml. osallistumisrajoitteet***
Metodeita voi kuvata koodin yhteydessä (Docstring)

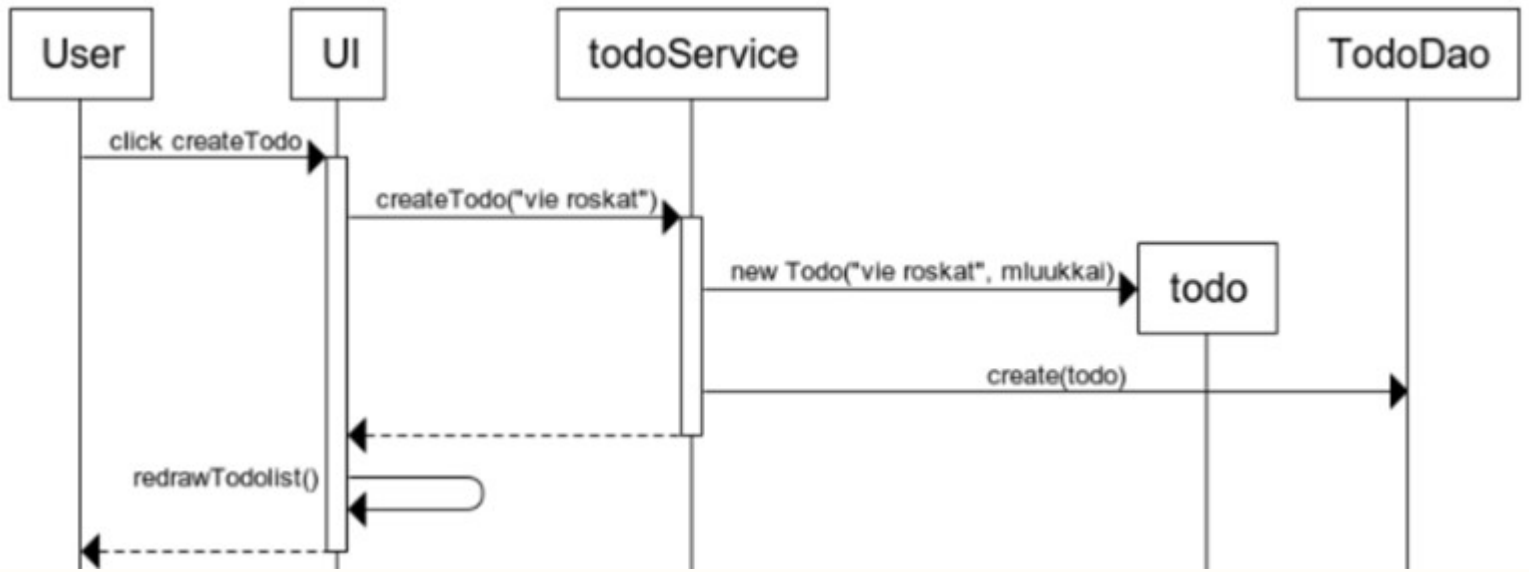
Kurssimateriaalissa vielä mm. riippuvuudet, perintä, pakkauskaaviot…

Näillä voidaan kuvata ohjelmiston rakennetta

Miten kuvaisit sen toimintaa?

Sekvenssikaaviot: toimenpidesarjan suoritus ajassa

Mikä olio kutsuu minkäkin mitäkin palvelua, missä järjestyksessä kutsutaan ja millä ehdoilla



- aika etenee ylhäältä alas
- metodikutsut kuvataan nuolilla
- metodien suoritusta kuvataan yleensä "aktivaatiopalkeilla"
- paluuarvot merkitään katkoviivoilla

(UML tarjoaa muitakin käyttäytymiskaavioita)

Viikot 4 - 7

Edistetään harjoitustyötä ohjeiden mukaisesti joka viikko…

Viikolla 6 on ohjelmassa koodikatselmointi, jossa annetaan palautetta jonkun toisen harjoitustyöstä

Mahdollista osallistua myös *ylimääräiseen ja vapaaehtoiseen* koodikatselmointiin viikoilla 4 ja 5

Arviointi

Viikkodeadlinet 16 p. (osa viikkopisteistä tulee laskareista)

Koodikatselmointi 2 p.

Dokumentaatio 13 p. ***loppupalautus***

Automatisoitu testaus 5 p.

Lopullinen ohjelma 24 p. (laajuus, ominaisuudet ja koodin laatu)

Yht. max. 60 p.

+1 lisäpiste palautteesta

+1 lisäpiste vapaaehtoisesta koodikatselmoinnista

Arvosana 1: 30 p., arvosana 5: n. 55 p.

Läpipääsyyn vaaditaan lisäksi vähintään 10 p. lopullisesta ohjelmasta.

Lopullinen ohjelma tarkemmin:

Käyttöliittymä 4 p.

- hyvin yksinkertainen tekstikäyttöliittymä 0 p. … laaja GUI 4 p.

Tiedon pysyväistallennus 4 p.

- tiedosto 1 - 2 p.; tietokanta/Internet 3 - 4 p.

Ohjelman lajuus 4 p.

- TodoApp siis laajuudeltaan 2 p.

Sovelluslogiikan kompleksisuus 3 p.

Ulkoisten kirjastojen hyödyntäminen 1 p.

Release 1 p.

Koodin laatu 5 p.

Virheiden käsittely 2 p.

Muutama varoitus…

1. Muista, että harjoitustyösi täytyy toimia osaston Linux-koneilla
2. Muista, että aika kuluu nopeasti
3. Älä plagioi
4. Älä riko tekijänoikeuksia
5. Ole varovainen kelimallien kanssa

Osaston Linux-koneet

Ympäristöissä on eroja; ei riitä, että työsi toimii omalla koneellasi!

Harjoitustyösi pitää pystyä **joka viikko** suorittamaan, kääntämään ja testaamaan komentoriviltä käsin osaston Linux-koneilla

- muuten työtäsi ei tarkasteta ja menetät viikon/loppupalautuksen pisteet

Voit testata ohjelmistosi toimintaa **yliopiston Linux-virtuaalityöasemalla**

- (tai jopa toteuttaakin työtäsi sillä?)

Aika kuluu nopeasti

Seitsemän viikkoa ei ole paljoa

Kaikkea ei kannata jättää deadlineja edeltäviin iltoihin!

Mieti jo etukäteen, missä kohtaa harjoitustyötä saattaa tulla ongelmia

Työ on saatava valmiiksi kurssin aikana ja sitä on toteutettava tasaisesti, muuten kurssi katsotaan keskeytetyksi

- samaa ohjelmaa ei voi jatkaa seuraavalla kurssilla, vaan työ on aloitettava uudella aiheella alusta

Älä plagioi

Verkosta tai kavereilta löytyneiden vastausten kopiointi ja niiden palauttaminen omina aikaansaannoksina on kiellettyä

Yliopistolla plagiointiin suhtaudutaan varsin ankarasti:

https://studies.helsinki.fi/ohjeet/artikkeli/mita-ovat-vilppi-ja-plagiointi

Vilpillä voi olla vakavia seurauksia

Älä riko tekijänoikeuksia

…tai muitakaan immateriaalioikeuksia

Muista, että mitä tahansa verkosta löytynyttä ei saa käyttää omassa työssä miten tahansa

Tarkista lisenssit. Sallivatko ne käytön?

Koskee monenlaista materiaalia: koodia, kuvia, tekstiä…

Harjoitustyöt ovat julkisia GitHubissa. On omalla vastuullasi, ettet riko tekijänoikeuksia!

ChatGPT ja muut tekoälypohjaiset välineet

Microsoft Copilot, Google Gemini, GitHub Copilot yms.

Sopivat **varauksin** ja omalla vastuulla mm.

- virheidenjäljitykseen
- itse kirjoitetun koodin paranteluun
- dokumentaation luonnosteluun ja hiomiseen
- tiedonhakuun ja neuvojen kysymiseen
- ohjelmakoodin selittämiseen…

*Entä harjoitustyön koodin ja dokumentaation **generointi**?*

Kurssisivulla: "**sallittu myös ohjelmakoodin generointiin**"

…mutta muista, ettei ole mitään takeita, että generoitu koodi todella tekee mitä odotetaan tai on muuten sopivaa. (Kielimalleihin liittyy muitakin ongelmia, esim. avoimia tekijänoikeuskysymyksiä.)

Monilla kursseilla generoinnin täyskielto

Kurssisivulla: "**sallittu myös ohjelmakoodin generointiin**"

…mutta muista, ettei ole mitään takeita, että generoitu koodi todella tekee mitä odotetaan tai on muuten sopivaa. (Kielimalleihin liittyy muitakin ongelmia, esim. avoimia tekijänoikeuskysymyksiä.)

Monilla kursseilla generoinnin täyskielto

POIKKEUS: *yksikkötestejä* ei saa generoida

Lisäksi:

- dokumentaatio tulee kirjoittaa pääosin itse
- viikkoharjoituksissa generoitu vastaus = 0 p.

(*Todella ahkeran* ohjelmakoodin generoinnin täytynee jossain kohtaa vaikuttaa meilläkin jonkin verran pisteytykseen… Tämä raja ei kuitenkaan tule ihan pian vastaan.)

Kurssisivulla: **"kielimallien käytöstä pitää raportoida"**

Jos liität työhösi generoitua koodia sellaisenaan tai vähäisin muutoksin, lisää **kommentit**

- kommentit generoidun koodinpätkän alkuun ja loppuun. Jos tämä ei käyttötavasta johtuen tunnu mielekkäältä, voit kertoa dokumentaation liitteessä miten ja kuinka paljon käytit tekoälyä minkäkin ohjelman osan toteutuksessa.
- muista, että myös generoidun koodin esittäminen omana on plagiointia

Lisäksi lopussa pieni **selvitys** tekoälypohjaisten välineiden käytöstä

- kirjaattehan matkan varrella muistiin, että millä tavoin mahdollisesti olette näitä käyttäneet ja millaisin kokemuksin
- osa dokumentaatiota, mutta toteutetaan kyselylomakkeella
- tuloksia saatetaan käyttää pienimuotoisessa tutkimuksessa

Keväällä -24:

- 63 % vastaajista oli käyttänyt kielimalleja koodin paranteluun
- 59 % koodin generointiin
- 12 % arvioi generoineensa yli neljänneksen koodistaan

Yli 70 % näki kielimallien *edistäneen oppimista*. Todettiin mm., että aikaa jäi enemmän kurssin kannalta oleellisten asioiden, kuten suunnittelun, testaamisen ja dokumentoinnin harjoitteluun.

Hyvä kurssimenestys liittyi koodin *maltilliseen* generointiin silloin kun se yhdistyi yleiseen tiedonhakuun kurssin aihepiiristä, ahkeraan koodin paranteluun tai ideointiin

Myöhempiin kyselyihin on tullut pitkälti samansuuntaisia vastauksia

- Kielimalleja käyttäneiden osuus ja käytettyjen kielimallien suosio ovat jonkin verran vaihdelleet
- ChatGPT on ollut suosituin väline, GitHub Copilotin käyttö on lisääntynyt

Jos kielimalleja käyttää, kannattaa pyrkiä todella *ymmärtämään* kurssilla käsiteltyjä asioita niiden avulla

- kysy myös “miksi” ja “mitä”
- älä yritä käyttää niitä vain kurssin suorittamista nopeuttavana oikotienä

Muista, että vastaukset **eivät** aina ole parhaita mahdollisia ja generoidussa koodissa voi olla erinäisiä ongelmia ja suoranaisia virheitäkin. Kyselyssä tai kurssilla muuten on tullut vastaan mm. seuraavia ongelmia:

- kielimallit eivät välttämättä osaa hahmottaa kokonaisuutta tai kontekstia riittävän hyvin
- kokonaisuudesta saattaa helposti tulla jossain määrin sekava ja koodiin voi tulla liikaa toisteisuutta. Ongelma mm. virheenjäljityksen, tietoturvan, ylläpidettävyyden ja laajennettavuuden kannalta.
- toteutettavan asian mutkikkuus saattaa (odotetusti) lisätä ongelmia
- kielimalli saattaa joskus tarjota sellaisia ratkaisutapoja, joita itse ei olisi käyttänyt
- älä edes yritä generoida laajasti dokumentaatiota: lopputulos ei todennäköisesti ole lähelläkään sitä, mitä kurssilla haetaan.

…Toisaalta kielimallien generoima koodi on saanut enemmän kehuja kuin moitteita ja ongelmat on todettu havaitun useimmiten jo ennen varsinaista testausta.

Ongelmia?

Viikkopalautuksista tulee *lyhyttä* palautetta, mutta *varsinaista tukea* on tarjolla:

1. **pajassa**	kävele sisään paja-aikoina
2. **Discord-kanavalla**	kirjoita kysymys
3. **Zoomissa**	pyydä tuutorilta ohjausaika

Lisäksi Labtooliin voi kirjoittaa kommentteja, jos viikoittaiset pisteet askarruttavat

Pajassa saat parhaiten apua!