Ohjelmistotekniikan menetelmät

syksy 2013

Matti Luukkainen

Kurssin aihepiiri: ohjelmistotuotannon alkeita

- [wikipedia]:
  - **Ohjelmistotuotanto** on yhteisnimitys niille työnteon ja työnjohdon menetelmille, joita käytetään, kun tuotetaan tietokoneohjelmia sekä monista tietokoneohjelmista koostuvia tietokoneohjelmistoja.
  - Laajasti ymmärrettynä ohjelmistotuotanto kattaa kaiken tietokone-ohjelmistojen valmistukseen liittyvän prosessinhallinnan sekä kaikki erilaiset **tietokoneohjelmien valmistamisen menetelmät.**
  - Ohjelmistotuotantoon kuuluu siis periaatteessa mikä tahansa aktiviteetti, joka tähtää tietokoneohjelmien tai -ohjelmistojen valmistukseen.
- Näihin **tietokoneohjelman valmistamisen menetelmiin liittyy mallintaminen,** eli kyky tuottaa erilaisia kuvauksia, joita tarvitaan ohjelmiston kehittämisen yhteydessä
  - Mallit toimivat kommunikoinnin välineinä:
    - *Mitä ollaan tekemässä, miten ollaan tekemässä, mitä tehtiin?*

Hallinnolliset asiat

- Aikataulu ja kurssimateriaali:

  https://github.com/mluukkai/OTM2013/wiki/Ohjelmistotekniikan-menetelmat

- Laskarit
  - Aloitetaan jo tällä viikolla
  - Joka viikko sekä etukäteen tehtäviä että paikanpällä tehtäviä tehtäviä
- Kurssin arvostelu
    - Kurssikoe 24 pistettä
    - Laskarit 6+6 pistettä
    - 18p => 1, ..., noin 32p => 5
    - Läpipääsyyn vaaditaan lisäksi **puolet kurssikokeen pisteistä JA puolet laskaripisteistä**
    - Laskarieista lisää seuraavalla sivulla

Laskareista ja laskaripisteistä

- Laskareista yhteensä siis maksimissaan 12 pistettä (36:sta kurssipisteestä)
    - Läpipääsyyn vaaditaan puolet laskaripisteistä
- 6 pisteistä tulee etukäteen tehtävistä tehtävistä, täydet saa jos tekee 90% tehtävistä
    - Etukäteen tehtävät tehtävät käsitellään laskareissa
- Paikanpäällä tehtävistä tehtävistä saa loput 6 pistettä
    - 1 piste per viikko
    - Viikon pisteen saa paikanpäällä olemalla ja tekemällä kaikki viikon paikanpäällä tehtävät tehtävät
- Laskarit eivät ole paja, paikalla on oltava alusta loppuun
- Laskariajat **keskiviikko 9.30-12 ja 14.15-16.45**, **torstai 9.30-12 ja 14.15-16.45, perjantai 9.30-12 B221, paitsi viikolla 1 BK107**
    - Vierailu on tarvittaessa sallittua

Kurssimateriaalista

- Kurssin kotisivulta
  https://github.com/mluukkai/OTM2013/wiki/Ohjelmistotekniikan-menetelmat

  löytyvät luentokalvot ja luentomoniste
- Moniste ei kata ihan koko kurssin sisältöä, osa asioista vain luentokalvoilla
- Laskarit ja niihin liittyvä lisämateriaali löytyy sivulta

  https://github.com/mluukkai/OTM2013/wiki/Laskarit
- Edes luentokalvoilla ei kaikkia asioita käsitellä, osa löytyy laskareiden lisämateriaalista

Ohjelmistotuotantoprosessi

Ohjelmistotuotantoprosessi

- Jos ohjelmistoja tehdään häkkeröimällä noudattamatta mitään systematiikkaa, ei lopputulos yleensä vastaa odotuksia
  - Ok jos kyseessä pieni, itselle tehtävä ohjelma
  - Ei toimi isommille, monen hengen projekteissa asiakasta varten tuotetuille ohjelmille
    - Toimiiko niin kuin haluttiin?
    - Rakenne epämääräinen => Ylläpidettävyys huono
- Kehitelty erilaisia menetelmiä ohjelmistotuotantoprosessin systematisoimiseksi
  - Menetelmiä paljon, ks. http://en.wikipedia.org/wiki/List_of_software_development_philosophies
  - Mitä menetelmää tulisi käyttää? Hyvä kysymys!

Ohjelmistotuotantoprosessi

- Käytetystä menetelmästä riippumatta löytyy ohjelmistotuotantoprosessista lähes aina seuraavat **vaiheet**
  - **Vaatimusanalyysi- ja määrittely**
    - Mitä halutaan?
  - **Suunnittelu**
    - Miten tehdään?
  - **Toteutus**
    - Ohjelmointi
  - **Testaus**
    - Varmistetaan että toimii niin kuin halutaan
  - **Ylläpito**
    - Korjataan bugit ja laajennetaan

Vaatimusanalyysi ja -määrittely

- Kartoitetaan ja dokumentoidaan mitä asiakas haluaa

- **Toiminnalliset vaatimukset**
  - Miten ohjelman tulisi toimia?
- Toimintaympäristön asettamat **rajoitteet**
  - Toteutusympäristö
  - Suorituskykyvaatimukset
  - Luotettavuusvaatimukset
  - Käytettävyys

- Ei vielä puututa siihen miten järjestelmä tulisi toteuttaa
  - Ei oteta kantaa ohjelman sisäisiin teknisiin ratkaisuihin, ainoastaan siihen miten toiminta näkyy käyttäjälle

Esim. Yliopiston kurssinhallintajärjestelmä

- toiminnallisia vaatimuksia esim.:
  - Opetushallinto voi syöttää kurssin tiedot järjestelmään
  - Opiskelija voi ilmoittautua valitsemalleen kurssille
  - Opettaja voi syöttää opiskelijan suoritustiedot
  - Opettaja voi tulostaa kurssin tulokset
- Toimintaympäristön rajoitteita esim.:
  - Kurssien tiedot talletetaan jo olemassa olevaan tietokantaan
  - Järjestelmää käytetään www-selaimella
  - Toteutus Javalla
  - Kyettävä käsittelemään vähintään 100 ilmoittautumista minuutissa

Vaatimusanalyysi ja -määrittely

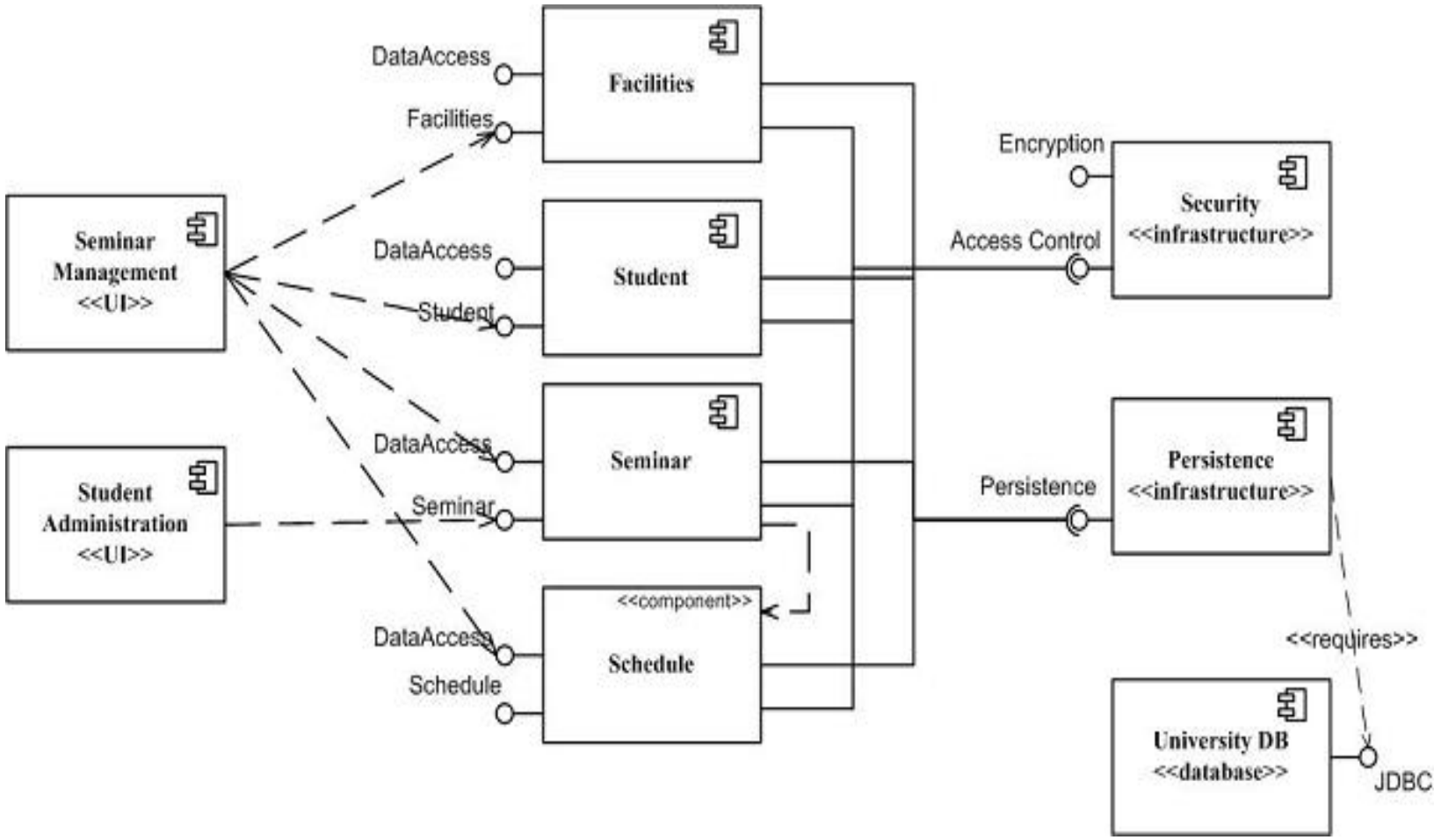
- Jotta toteuttajat ymmärtäisivät mitä pitää tehdä, joudutaan ongelma-aluetta **analysoimaan**
  - Esim. Jäsennetään ongelma-alueen käsitteistöä
  - Tehdään ongelma-alueesta **malli** eli yksinkertaistettu kuvaus

- Vaatimusmäärittelyn päätteeksi yleensä tuotetaan **määrittelydokumentti**
  - Kirjaa sen mitä ohjelmalta halutaan
    - Ohjeena suunnitteluun ja toteutukseen
    - Testaus perustuu dokumentissa asetettuihin ohjelman vaatimuksiin
- Määrittelydokumentin sijaan määrittely (tai ainakin sen osa) voidaan myös ilmaista ns. hyväksymistesteinä. Tällöin ohjelma toimii ”määritelmänsä mukaisesti” jos se läpäisee kaikki määritellyt hyväksymistestit

Ohjelmiston suunnittelu

- Miten saadaan toteutettua määrittelydokumentissa vaaditulla tavalla toimiva ohjelma
- Kaksi vaihetta:
  - **Arkkitehtuurisuunnittelu**
    - Ohjelman rakenne karkealla tasolla
    - Mistä suuremmista rakennekomponenteista ohjelma koostuu?
    - Miten komponentit yhdistetään, eli komponenttien väliset rajapinnat
  - **Oliosuunnittelu**
    - yksittäisten komponenttien suunnittelu
- Lopputuloksena **suunnitteludokumentti**
  - Ohje toteuttajille
- Joskus suunnittelu- ja ohjelmointivaihe ovat niin kiinteästi sidottuna toisiinsa että tarkkaa suunnitteludokumenttia ei tehdä
    - Koodi toimii dokumenttina

- Mallit liittyvät vahvasti myös suunnitteluun
- Alla esimerkki arkkitehtuurikuvauksesta
  - järjestelmän jako alikomponentteihin
  - komponenttien väliset rajapinnat

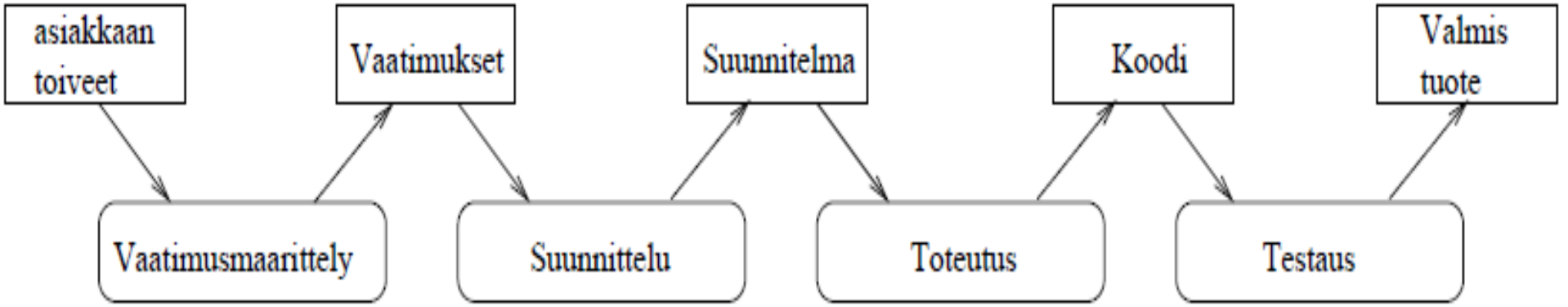


Toteutus, testaus ja ylläpito

- Suunnittelun mukainen järjestelmä toteutetaan valittuja tekniikoita käyttäen
- Toteutuksen yhteydessä/jälkeen testataan:
  - Yksikkötestaus
    - Toimivatko yksittäiset metodit ja luokat?
  - Integraatiotitestaus
    - Varmistetaan komponenttien yhteentoimivuus
  - Järjestelmätestaus
    - Toimiiko kokonaisuus niin kuin vaatimusdokumentissa sanotaan?
- Valmiissakin järjestelmässä virheitä ja tarvetta laajennuksiin
  - Ohjelmiston ylläpitoa

- **Tämän kurssin asiat liittyvät lähinnä vaatimusmäärittelyyn ja suunnitteluun, osin myös testaamiseen**

Vesiputousmalli



- Perinteinen tapa tehdä ohjelmistoja
  - Tuotantoprosessin vaiheet etenevät peräkkäin
  - Eli ensin vaatimusmäärittely kokonaisuudessaan
  - Sitten suunnittelu, jne..
- Jokainen vaihe lopullisesti valmiiksi ennen kuin seuraavaan vaiheeseen
- Jokaisen vaiheen lopputulos dokumentoidaan tyypillisesti erittäin tarkasti

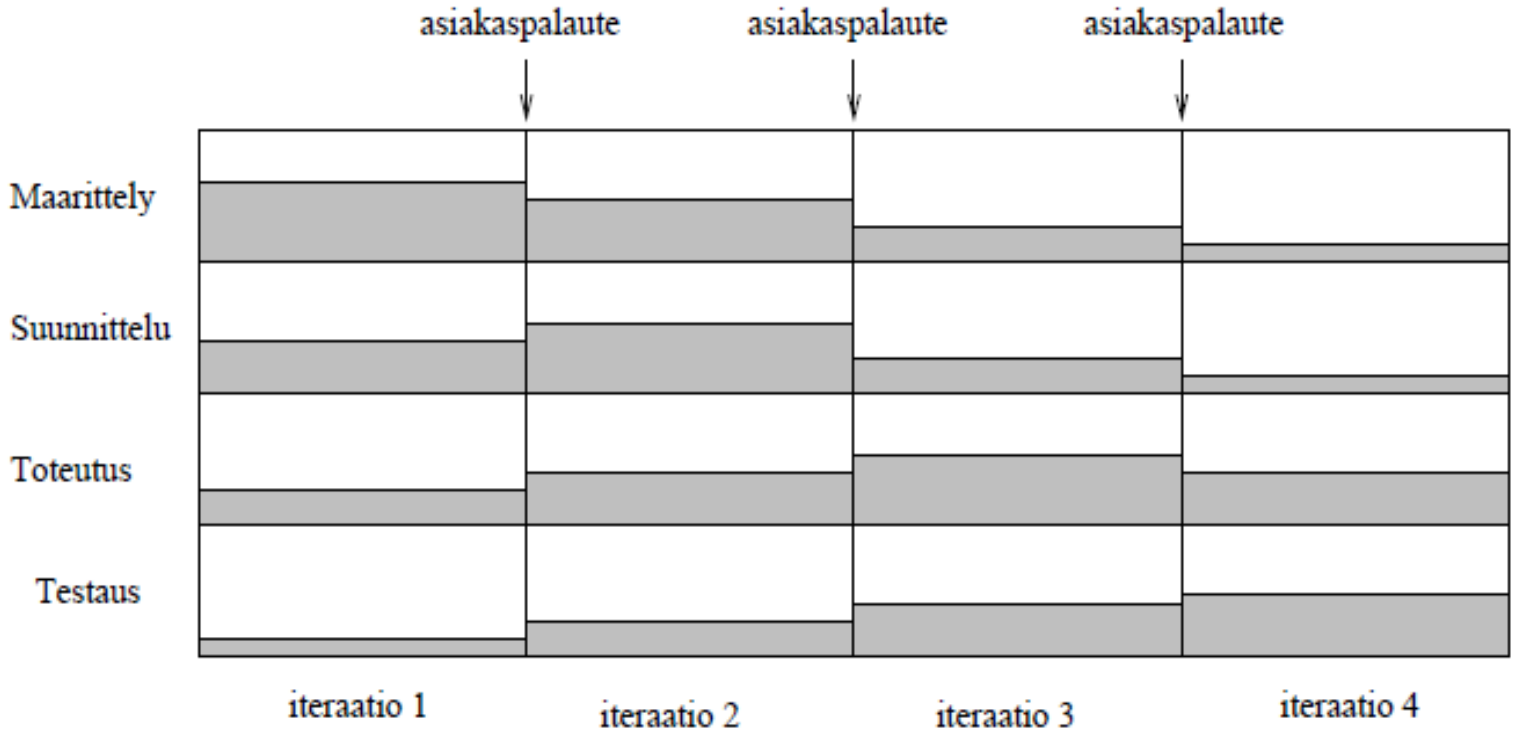
Vesiputousmallin ongelmia

- Järjestelmää testataan kokonaisuudessaan vasta kun “kaikki” on valmiina
  - Suunnitteluvaiheen virheet saattavat paljastua vasta testauksessa
  - Eli ongelmat selviävät myöhään
- *Perustuu oletukselle, että käyttäjä pystyy määrittelemään ohjelmalta halutun toiminnallisuuden heti projektin alussa*
  - Näin ei useinkaan ole
  - Vasta nähdessään lopputuloksen käyttäjä tajuaa mitä oikeastaan halusikaan => Seurauksena raskas ylläpitovaihe
  - Jos projekti on pitkäkestoinen, voi olla että käyttäjän tarve muuttuu projektin kuluessa (esim. yritysfuusion seurauksena)
    - Eli se ohjelma mitä haluttiin tilausvaiheessa ei olekaan enää tarpeellinen ohjelman valmistuessa
- **Ohjelmistotuotannon yksi perustavanlaatuisimmista ongelmista on asiakkaan ja toteuttajien välinen kommunikointi**
  - kärjistyy vesiputousmallissa koska palautetta hankala saada kesken projektin

Ketterä ohjelmistokehitys

- *Lähdetään olettamuksesta, että asiakkaan vaatimukset muuttuvat ja tarkentuvat projektin kuluessa*
  - Ei siis yritetäkään kirjoittaa alussa määrittelydokumenttia, jossa kirjattuna tyhjentävästi järjestelmältä haluttu toiminnallisuus
- Tuotetaan järjestelmä ***iteratiivisesti***, eli pienissä paloissa
  - Ensimmäisen iteraation aikana tuotetaan pieni osa järjestelmän toiminnallisuutta
    - määritellään vähän, suunnitellaan vähän ja toteutetaan ne
    - Lopputuloksena siis jo ohjelmisto, jossa mukana osa toiminnallisuutta
  - Iteraatio kestää tyypillisesti muutaman viikon
  - Asiakas antaa palautteen iteraation päätteeksi
    - Jos huomataan, että järjestelmä ei ole sellainen kuin haluttiin, voidaan tehdä heti korjausliike
  - Seuraavassa iteraatiossa toteutetaan taas hiukan uutta toiminnallisuutta asiakkaan toiveiden mukaan

Ketterä ohjelmistokehitys



- Eli jokainen iteraatio tuottaa toimivan järjestelmän
- Asiakkaan palaute välitön
  - Vaatimuksia voidaan tarkentaa ja muuttaa
- Asiakas valitsee jokaisen iteraation aikana toteutettavat lisäominaisuudet
- *Kommunikaatio asiakkaan kanssa jatkuvaa*
  - todennäköisempää että aikaansaannos toiveiden mukainen

Ketterä ohjelmistokehitys

- Iteraation sisällä määrittely, suunnittelu, toteutus ja testaus eivät välttämättä etene peräkkäin
  - Määritellään, suunnitellaan, toteutetaan ja testataan jatkuvasti
- Ketterissä menetelmissä dokumentoinnin rooli kevyempi kun vesiputousmallissa

- **Virheellinen johtopäätös on ajatella, että kaikki ei-perinteinen tapa tuottaa ohjelmistoja on ketterien menetelmien mukainen**
  - Häkkerointi siis ei ole ketterä menetelmä!
- Monissa ketterissä menetelmissä (kuten XP eli eXtreme Programming) on päinvastoin erittäin tarkasti määritelty miten ohjelmien laatua hallitaan
  - Pariohjelmointi, jatkuva integraatio, automatisoitu testaus, Testaus ensin -lähestymistapa (TDD), ...
- Myös ketteryys siis vaatii kurinalaisuutta, joskus jopa enemmän kuin perinteinen vesiputousmalli

Mallintaminen

Mallintaminen

- Perinteiset insinöörialat perustuvat malleihin
  - Esim. siltaa rakentaessa tarkat lujuuslaskelmat (=malli)
  - Näihin perustuen tehdään piirustukset, eli malli siitä miten silta pitää toteuttaa (=edellistä hieman tarkempi malli)
- **Malli on abstrakti kuvaus mielenkiinnon alla olevasta kohteesta**
  - pyrkii kuvaamaan vaan olennaisen
    - Käyttötarkoitusta varten liian tarkat tai liian ylimalkaiset mallit epäoptimaalisia
  - Mitä on olennaista, riippuu mallin käyttötarkoituksesta
    - Esim. Metron linjakartta on hyvä malli julkisen liikenteen käyttäjälle
    - Autoilija taas tarvitsee tarkemman mallin eli kartan
    - Pelkkä maantiekartta riittää tarkasteltaessa esim. miten päästään Helsingistä Rovaniemelle
    - Helsingin keskustassa taas tarvitaan tarkempi kartta

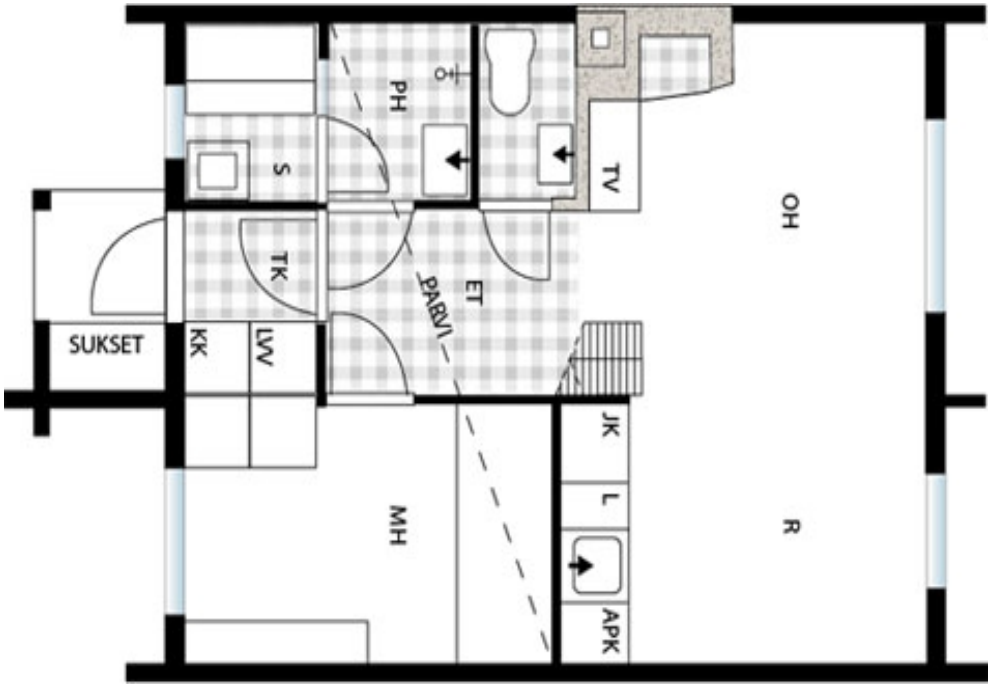
Mallin näkökulma ja abstraktiotaso

- Mallien **abstraktiotaso** vaihtelee
  - Abstraktimpi malli käyttää korkeamman tason käsitteitä
  - Konkreettisempi malli taas on yksityiskohtaisempi ja käyttää “matalamman” tason käsitteitä ja kuvaa kohdetta tarkemmin

- Samaa kohdetta voidaan mallintaa monesta eri **näkökulmasta**
  - Jos kaikki yritetään mahduttaa samaan malliin, ei lopputulos ole selkeä
  - Malli kuvaa usein korostetusti tiettyä näkökulmaa
  - Eri näkökulmat yhdistämällä saadaan idea kokonaisuudesta

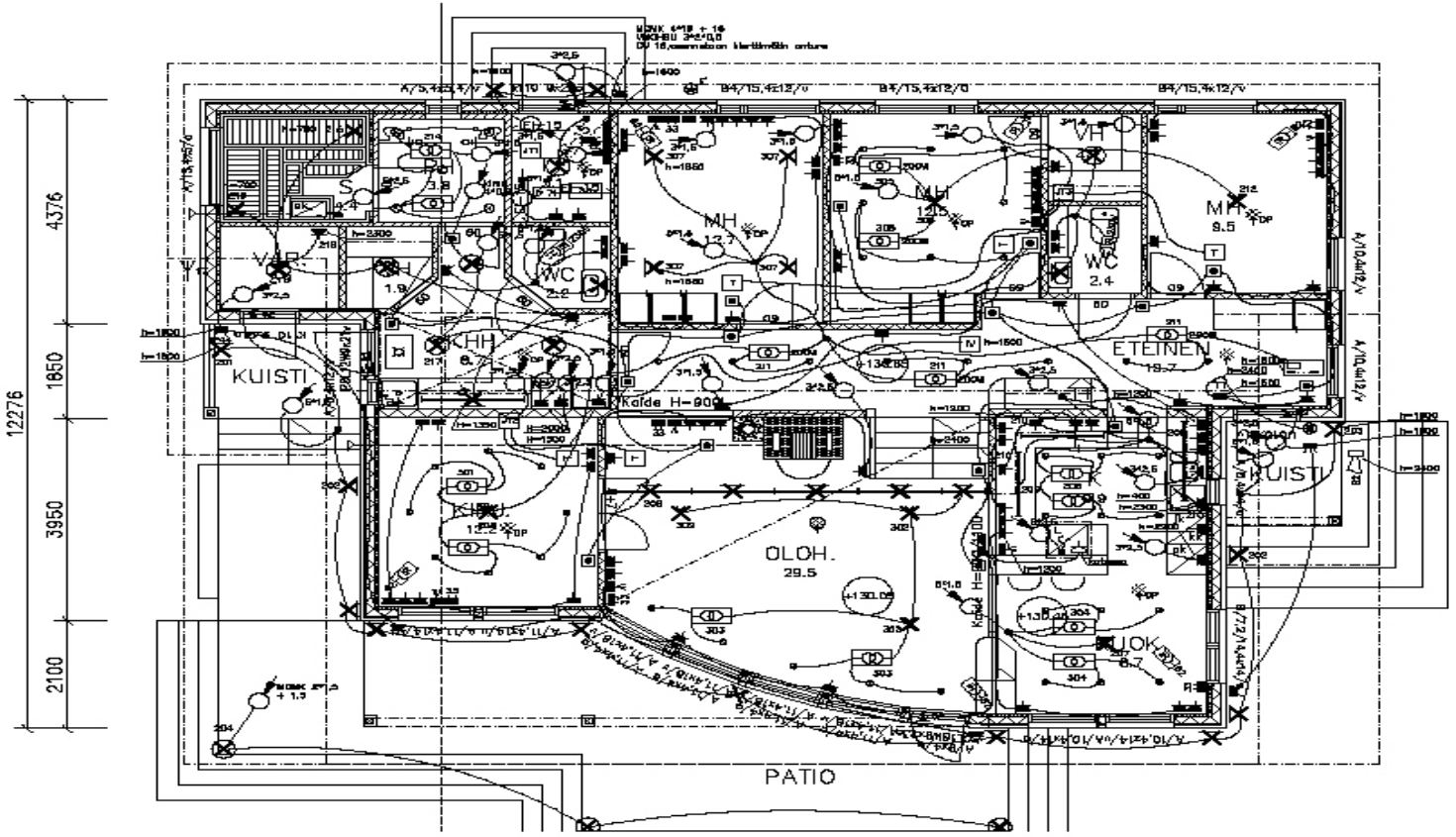
Eri abstraktiotason mallit

- Hyvin abstrakti kuvaus talosta:
  - 78m2, 2h+keittiö+sauna
- Hieman konkreettisempi:



Tarkentuva malli ja näkökulman valinta

- Vielä konkreettisempi malli
  - Näkökulmana sähkösuunnitelma:



Ohjelmistojen mallintaminen

- Vaatimusdokumentissa *mallinnetaan* mitä järjestelmän toiminnallisuudelta halutaan
- Suunnitteludokumentissa *mallinnetaan*
  - Järjestelmän arkkitehtuuri eli jakautuminen tarkempiin komponentteihin
  - Yksittäisten komponenttien toiminta
- Toteuttaja käyttää näitä malleja ja luo konkreettisen tuotteen

- Vaatimuksien mallit yleensä korkeammalla abstraktiotasolla kuin suunnitelman mallit
  - Vaatimus ei puhu ohjelman sisäisestä rakenteesta toisin kuin suunnitelma

Ohjelmistojen mallintaminen

- Myös ohjelmistojen malleilla on erilaisia *näkökulmia*
- Jotkut mallit kuvaavat rakennetta
  - Mitä komponentteja järjestelmässä on
- Jotkut taas toimintaa
  - Miten komponentit kommunikoivat
- Eri näkökulmat yhdistämällä saadaan idea kokonaisuudesta

Mallinnuksen kaksi suuntaa

- Usein mallit toimivat apuna kun ollaan rakentamassa jotain uutta, eli
  - Ensin tehdään malli, sitten rakennetaan esim. silta
  - Eli rakennetaan mallin mukaan
- Toisaalta esim. fyysikot tutkivat erilaisia fyysisen maailman ilmiöitä rakentamalla niistä malleja ymmärryksen helpottamiseksi
  - Ensin on olemassa jotain todellista josta sitten luodaan malli
  - Eli mallinnetaan olemassa olevaa
- Ohjelmistojen mallinnuksessa myös olemassa nämä kaksi mallinnussuuntaa
  - Ohje toteuttamiselle: malli => ohjelma
  - Apuna asiakkaan ymmärtämiseen: todellisuus => malli
  - ns. takaisinmallinnus: Ohjelma => malli

Ohjelmistojen mallinnuskäytännöt

- Esim. talonrakennuksessa noudatetaan vakiintuneita mallinnuskäytäntöjä
- Miten on asia ohjelmistojen mallinnuksen suhteen?

Oliomallinnus

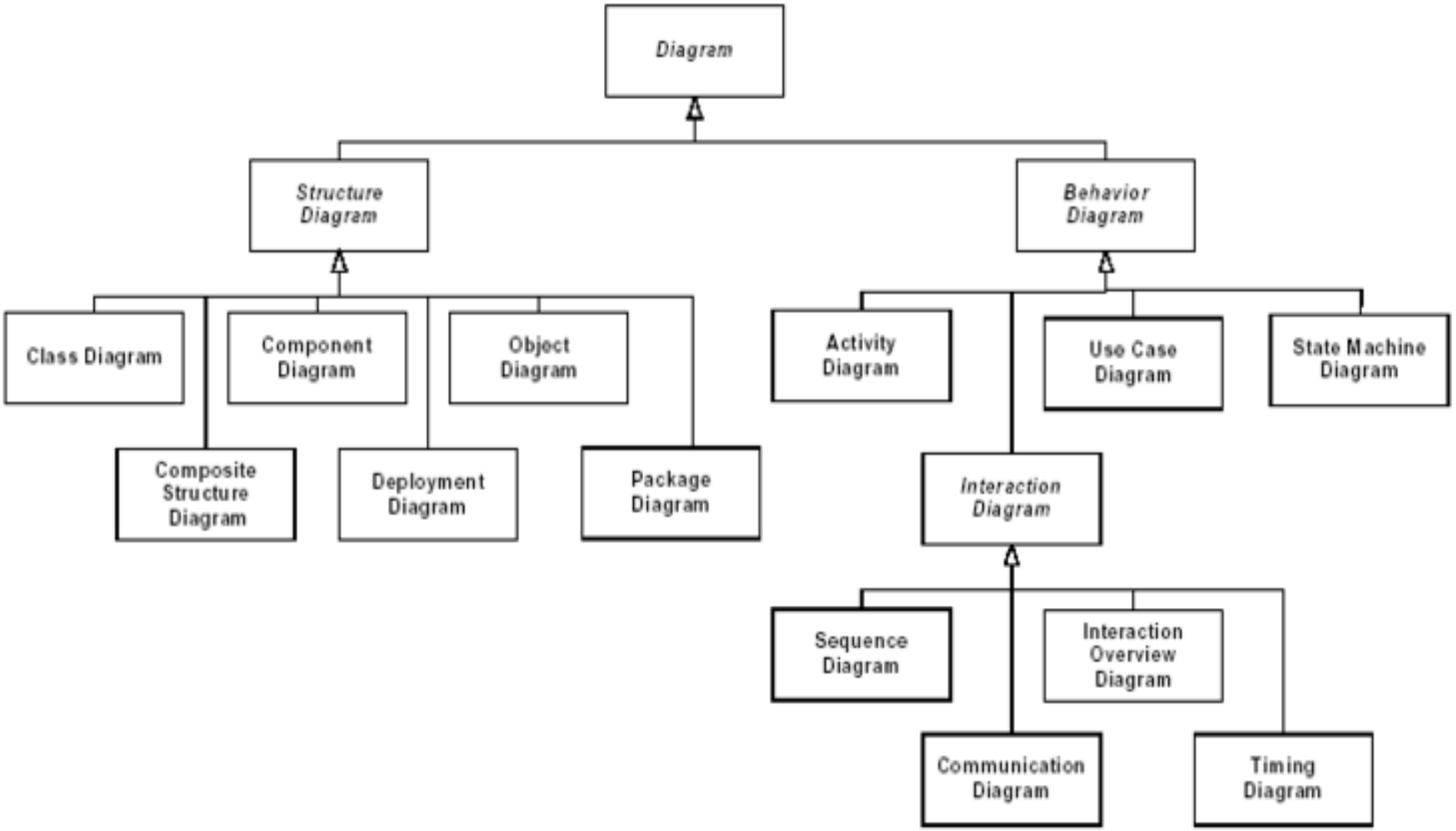
- Pitkään tilanne oli sekava ja on sitä osin edelleen
- Suosituimmaksi tavaksi on noussut **oliomallinnus**
  - Perustuu seuraavaan oletukseen:
  - *Minkä tahansa järjestelmän katsotaan voivan muodostua olioista, jotka yhteistyössä toimien ja toistensa palveluja hyödyntäen tuottavat järjestelmän tarjoamat palvelut*

- Eli järjestelmä tarjoaa joukon palveluja
  - Nämä palvelut toteuttavat asiakkaan vaatimukset
  - Järjestelmä rakentuu oliosta, jotka toteuttavat järjestelmän palvelut
  - Oliot ovat itsessään osajärjestelmiä, jotka tarjoavat palveluja toisille olioille ja käyttävät toisten olioiden palveluja

Unified Modelling Language eli UML

- Oliomallinnusta varten kehitetty standardoitu **kuvaustekniikka**
  - Taustalla joukko 90-luvun alussa kehitettyjä kuvaustekniikoita
  - Nykyinen versio 2.2
  - Vanhojen standardien mukaisia kaavioita näkyy yhä
- UML:ssä nykyään 13 esityyppistä kaaviota
  - Eri näkökulmille omat kaavionsa
- UML standardi ei määrittele **miten ja missä tilanteissa** kaavioita tulisi käyttää
  - Tätä varten olemassa useita **oliomenetelmiä**
    - Menetelmät antavat ohjeistoa UML:n soveltamiselle määrittelyssä ja suunnittelussa

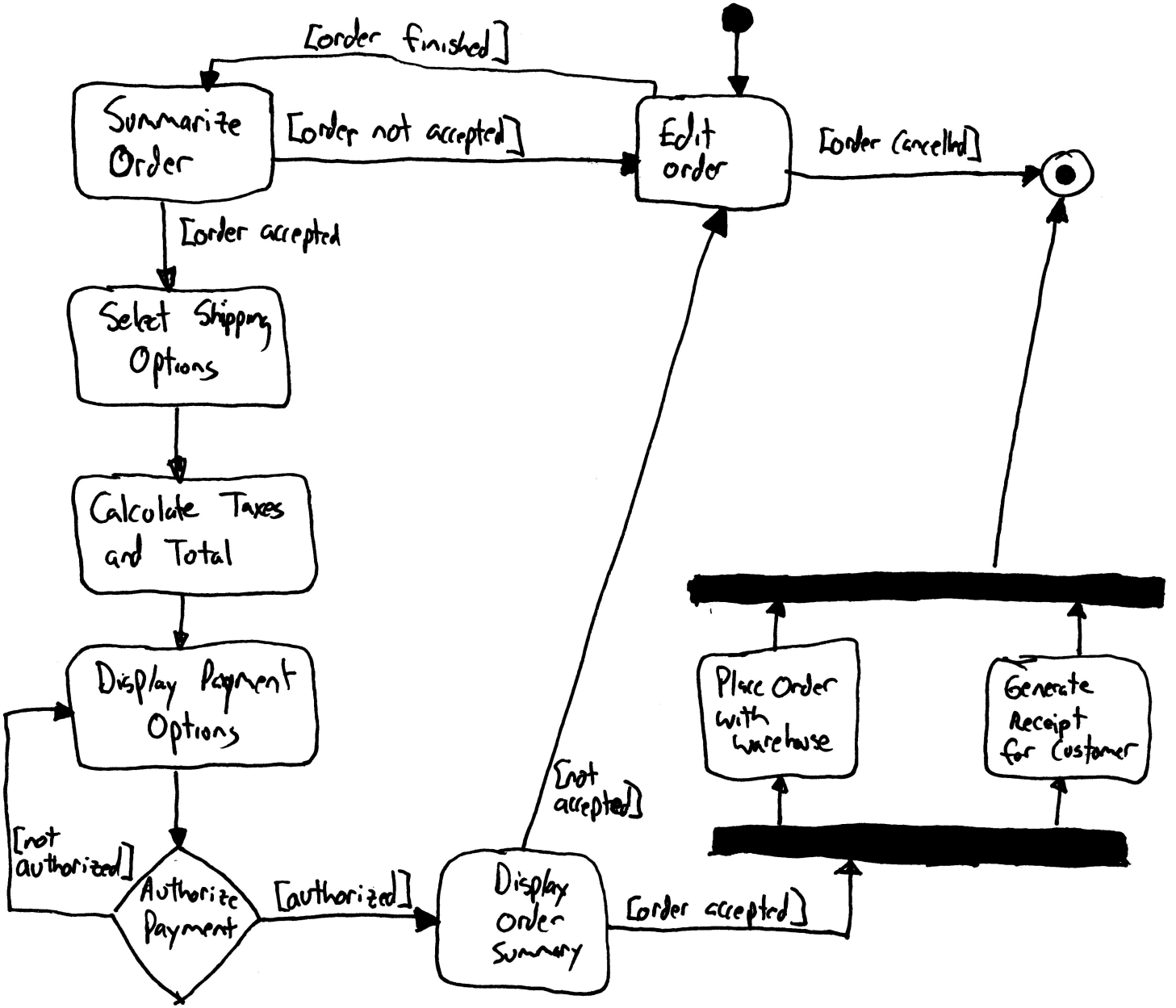
UML:n kaaviotyypit



UML:n käyttötapa

- UML-standardi määrittelee kaavioiden syntaksin eli oikeaoppisen piirtotavan suhteellisen tarkasti
  - Eri versioiden välillä pieniä muutoksia
- Jotkut suosivat UML:n käyttöä tarkasti syntaksia noudattaen
  - Kaaviot piirretään tällöin usein tietokoneavusteisella suunnitteluvälineellä
- On myös UML:n luonnosmaisemman käytön puolestapuhujia
  - Kuvia piirretään usein valkotaululle tai paperille
  - ns. ketterä mallinnus
  - Kaaviot ennenkaikkia kommunikoinnin apuväline
  - Tärkeimmät kuvat ehkä siirretään sähköiseen muotoon
    - Digikuva tai uudelleenpiirto editorilla

Käsin piirretty luonnosmainen kaavio



Käyttötapausmalli

Kertausta

- Ohjelmistotuotantoprosessin vaiheet:
  - Vaatimusanalyysi- ja määrittely
    - Mitä halutaan?
  - Suunnittelu
    - Miten tehdään?
  - Toteutus
    - Ohjelmointi
  - Testaus
    - Varmistetaan että toimii niin kuin halutaan
  - Ylläpito
    - Korjataan bugit ja tehdään laajennuksia

Nyt tarkastelun alla vaatimusanalyysi ja -määrittely

- Vaatimuksia siis kahdenlaisia
- *Toiminnalliset vaatimukset*
  - Mitä toimintoja ohjelmassa on?
  - Esim. kurssihallintojärjestelmä:
    - Opetushallinto voi syöttää kurssin tiedot järjestelmään
    - Opiskelija voi ilmoittautua valitsemalleen kurssille
    - Opettaja voi syöttää opiskelijan suoritustiedot
    - Opettaja voi tulostaa kurssin tulokset
- *Ei-toiminnalliset vaatimukset (eli ympäristön rajoitteet)*
  - Toteutusympäristö, suorituskykyvaatimukset, ...

- Vaatimusmäärittelyssä ei oteta kantaa ohjelman sisäisiin teknisiin ratkaisuihin, ainoastaan siihen miten toiminta näkyy käyttäjälle

- *Miten toiminnalliset vaatimukset tulisi ilmaista?*

Käyttötapausmalli

- Nyt esiteltävä **käyttötapausmalli** (engl. use case model) on yksi tapa ohjelman *toiminnallisten vaatimusten* ilmaisemiseen
  - Ei-toiminnallisten vaatimusten ilmaisemiseen käyttötapausmalli ei juuri ota kantaa, vaan ne on ilmaistava muuten
  - On olemassa muitakin tapoja toiminnallisten vaatimusten ilmaisuun

- Ohjelmisto tarjoaa käyttäjälleen palveluita
  - **Ohjelmiston toiminta voidaan kuvata määrittelemällä sen tarjoamat palvelut**

- Palveluilla on **käyttäjä**
  - Henkilö, toinen järjestelmä, laite yms. taho, joka on järjestelmän ulkopuolella, mutta tekemisissä järjestelmän kanssa

- Käyttäjä voi olla
  - Järjestelmän tiedon hyväksikäyttäjä
  - Järjestelmän tietojen lähde

Käyttäjien tunnistaminen

- Hyvä tapa aloittaa vaatimusmäärittely on tunnistaa/etsiä rakennettavan järjestelmän käyttäjät
- Kysymyksiä jotka auttavat:
  - Kuka/mikä saa tulosteita järjestelmästä?
  - Kuka/mikä toimittaa tietoa järjestelmään?
  - Kuka käyttää järjestelmää?
  - Mihin muihin järjestelmiin kehitettävä järjestelmä on yhteydessä?

- Käyttäjä on oikeastaan **rooli**
  - Missä roolissa toimitaan järjestelmän suhteen
  - Yksi ihminen voi toimia monessa roolissa...

TKTL:n kurssi-ilmoittautumisjärjestelmä

- Käyttäjärooleja
  - Opiskelija
  - Opettaja
  - Opetushallinto
  - Suunnittelija
  - Laitoksen johtoryhmä
  - Tilahallintojärjestelmä
  - Henkilöstöhallintajärjestelmä

- Osa käyttäjistä yhteydessä järjestelmään vain epäsuorasti (esim. Johtoryhmä)
- Osa “käyttäjistä” on muita järjestelmiä
  - Sana käyttäjä ei ole terminä tässä tilanteessa paras mahdollinen
  - Englanninkielinen termi *actor* onkin hieman suomenkielistä termiä kuvaavampi

Käyttötapaus, engl. use case

- **Käyttötapaus** kuvaa käyttäjän ohjelman avulla suorittaman tehtävän
  - *miten käyttäjä kommunikoi järjestelmän kanssa* tietyssä käyttötilanteessa
  - Käyttötilanteet liittyvät käyttäjän *tarpeeseen* tehdä järjestelmällä jotain
  - Esim.
    - Kurssi-ilmoittautumisjärjestelmä: *Opiskelijan ilmoittautuminen*
    - Mitä vuorovaikutusta käyttäjän ja järjestelmän välillä tapahtuu kun opiskelija ilmoittautuu kurssille?

- Yksi käyttötapaus on looginen, “isompi” kokonaisuus
  - Käyttötapauksella lähtökohta
  - Ja merkityksen omaava lopputulos (goal)
- Eli pienet asiat, kuten “syötä salasana” eivät ole käyttötapauksia
  - Kyseessä pikemminkin yksittäinen operaatio, joka voi sisältyä käyttötapaukseen

Esim. käyttötapaus opiskelija ilmoittautuu kurssille

- *Käyttäjä:* opiskelija
- *Tavoite:* saada kurssipaikka
- *Laukaisija:* opiskelijan tarve
- *Käyttötapauksen kulku:* Opiskelija tutkii kurssitarjontaa ja valitsee ohjelmiston esittämästä tarjonnasta kurssin ja ryhmän, tunnistautuu ja aktivoi ilmoittautumistoiminnon. Opiskelija saa kuittauksen ilmoittautumisen onnistumisesta.
- *Poikkeuksellinen toiminta:* Opiskelija ei voi ilmoittautua täynnä olevaan ryhmään. Opiskelija ei voi ilmoittautua, jos hänelle on kirjattu osallistumiseste.
- *Lisähuomioita:* 4 ruuhkahuippua vuodessa, noin 400 ilmoittautumista ensimmäisen 10 minuutin aikana ilmoittautumisen alkamisesta. Muulloin tapahtumia on vähän

Käyttötapauksen kuvaaminen

- **Kuvataan tekstinä**
- Ei ole olemassa täysin vakiintunutta tapaa kuvaukseen (esim. UML ei ota asiaan kantaa), mutta edellinen ja seuraava sivu sekä muut lähteet näyttävät mallia
- Kuvauksessa mukana usein tietyt osat
  - Käyttötapauksen nimi
  - Käyttäjät
  - Laukaisija
  - Esiehto
  - Jälkiehto
  - Käyttötapauksen kulku
  - Poikkeuksellinen toiminta
- Seuraavalla sivulla käyttötapaus *opiskelija ilmoittautuu kurssille* hieman tarkemmalla tasolla kuvattuna

- *Käyttäjä:* opiskelija
- *Tavoite:* saada kurssipaikka
- *Laukaisija:* opiskelijan tarve
- *Esiehto:* opiskelija on ilmoittautunut kuluvalla lukukaudella läsnäolevaksi
- *Jälkiehto:* opiskelija on lisätty haluamansa ryhmän ilmoittautujien listalle
- *Käyttötapauksen kulku:*

    1. Opiskelija aloittaa kurssi-ilmoittautumistoiminnon
    2. Järjestelmä näyttää kurssitarjonnan
    3. Opiskelija tutkii kurssitarjontaa
    4. Opiskeljia valitsee ohjelmiston esittämästä tarjonnasta kurssin ja ryhmän
    5. Järjestelmä pyytää opiskelijaa tunnistautumaan
    6. Opiskelija tunnistautuu ja aktivoi ilmoittautumistoiminnon
    7. Järjestelmä ilmoittaa opiskelijalle ilmoittautumisen onnistumisesta.

- *Poikkeuksellinen toiminta:*

    4a. Opiskelija ei voi valita ryhmää, joka on täynnä

    6a. Opiskelija ei voi ilmoittautua, jos hänelle on kirjattu osallistumiseste.

huomioita

- Esiehto
  - Asioiden tila joka on vallittava, jotta käyttötapaus pystyy käynnistymään
- Jälkiehto
  - Kuvaa mikä on tilanne käyttötapauksen onnistuneen suorituksen jälkeen
- Laukuaisija
  - Mikä aiheuttaa käyttötapauksen käynnistymisen, voi olla myös ajan kuluminen
- Käyttötapauksen kulku
  - Kuvaa onnistuneen suorituksen, usein edellisen sivun tapaan käyttäjän ja koneen välisenä dialogina
- Poikkeuksellinen toimita
  - Mitä tapahtuu jos tapahtumat eivät etene onnistuneen suorituksen kuvauksen mukaan
  - Viittaa onnistuneen suorituksen dialogin numeroihin, esim. jos kohdassa 4 voi tapahtua poikkeus normaaliin kulkuun, kuvataan se askeleena 4a

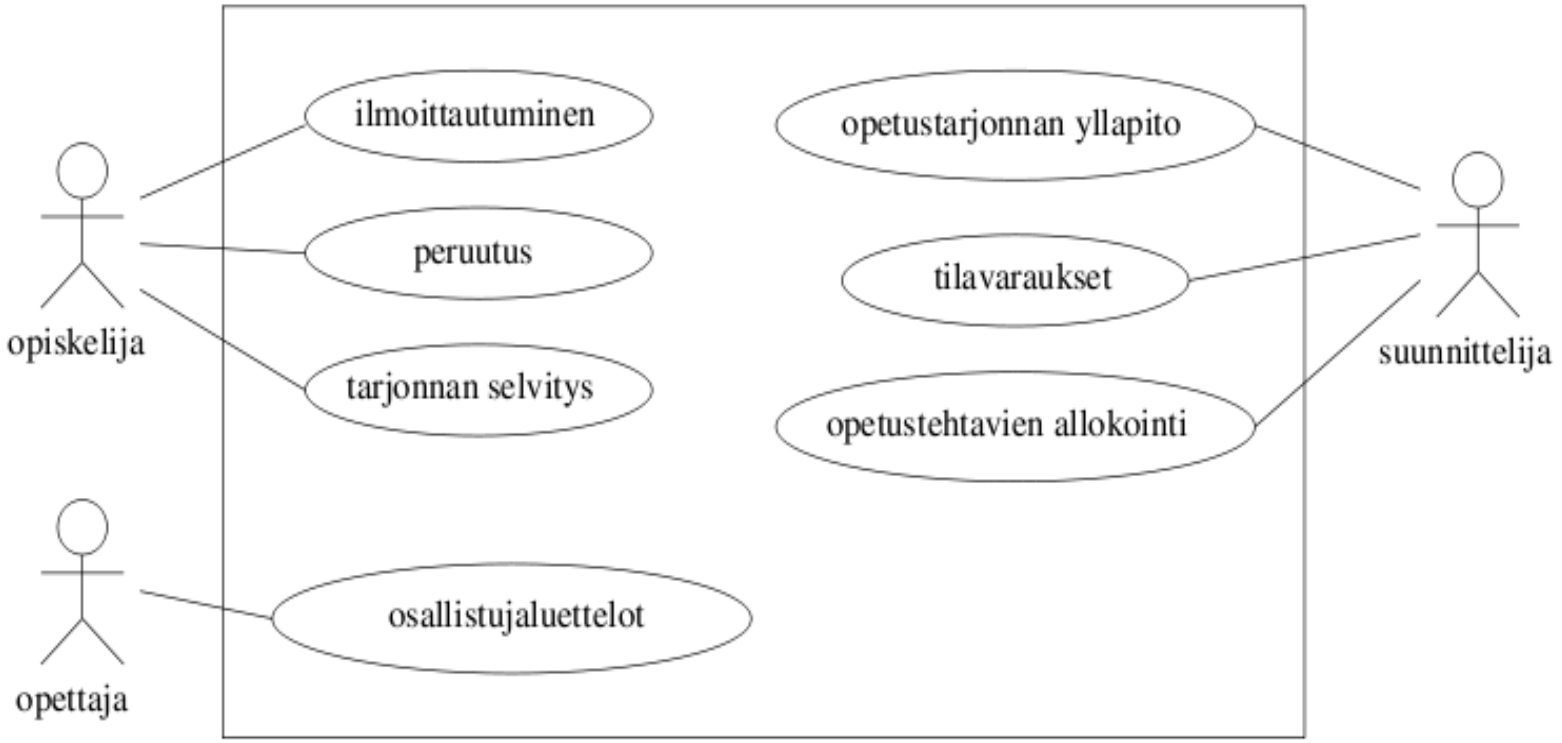
Käyttötapaus *ilmoittautumisen peruminen*

- *Käyttäjä:* opiskelija
- *Tavoite:* perua ilmoittautuminen, välttää sanktiot
- *Laukaisija:* opiskelijan tarve poistaa ilmoittautuminen
- *Esiehto:* opiskelija on ilmoittautunut tietylle kurssille
- *Jälkiehto:* opiskelijan ilmoittautuminen kurssille on poistettu
- *Käyttötapauksen kulku:*

1. Opiskelija valitsee toiminnon "omat ilmoittautumiset"
2. Järjestelmä pyytää opiskelijaa tunnistautumaan
3. Opiskelija tunnistautuu
4. Järjestelmä näyttää opiskelijan ilmoittautumiset
5. Opiskeljia valitsee tietyn ilmoittatumisensa ja peruu sen
6. Järjestelmä ilmoittaa opiskelijalle ilmoittautumisen peruuntumisesta

Käyttötapauskaavio

- UML:n **käyttötapauskaavion** avulla voidaan kuvata käyttötapausten ja käyttäjien (englanniksi termi on actor) keskinäisiä suhteita
- Kurssi-ilmoittautumisjärjestelmän “korkean tason” käyttötapauskaavio



Käyttötapauskaavio

- Käyttäjät kuvataan tikku-ukkoina
  - Olemassa myös vaihtoehtoinen symboli, joka esitellään pian
- Käyttötapaukset järjestelmää kuvaavan nelilön sisällä olevina ellipseinä
  - Ellipsin sisällä käyttötapauksen nimi
- Käyttötapausellipsiin yhdistetään viivalla kaikki sen käyttäjät
  - Kuvaan ei siis piirretä nuolia!

- **HUOM: Käyttötapauskaaviossa ei kuvata mitään järjestelmän sisäisestä rakenteesta**
  - Esim. vaikka tiedettäisiin että järjestelmä sisältää tietokannan, ei sitä tule kuvata käyttötapausmallissa

Käyttötapauskaavion käyttö

- Kaaviossa siis käyttötapauksista ainoastaan nimi
  - **Käyttötapauksen sisältö kuvataan aina tekstuaalisena esityksenä**
- Kaavio tarjoaa hyvän yleiskuvan järjestelmän käyttäjistä ja palveluista
- **Määrittelydokumentin** alussa kannattaakin olla käyttötapauskaavio “sisällysluettelona”
  - **Jokainen käyttötapaus tulee sitten kirjata tekstuaalisesti tarvittavalla tarkkuudella**

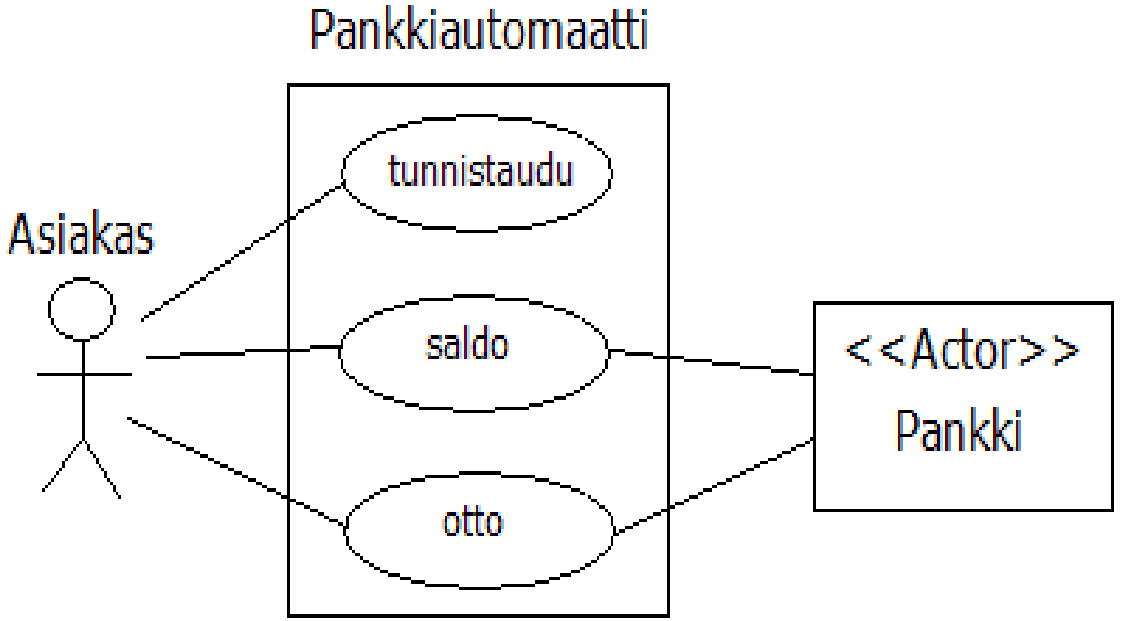
Käyttötapausten dokumentointi

- Ei siis ole olemassa standardoitua tapaa käyttötapauksen kirjaamiseen
- Ohjelmistoprojektissa tulee kuitenkin sopia joku yhteinen muoto, jota kaikkien käyttötapausten dokumentoinnissa noudatetaan
  - *Käyttötapauspohja*

- Edellä esitetyt esimerkkitapaukset tarjoavat mallin käyttötapaukselle
- Internetistä ja kirjoista löytyy myös paljon käyttötapauspohjia
- Ehkä suurimman käyttötapausgurun Alistair Cockburnin käyttötapauspohja löytyy osoitteesta:
  - www.cs.helsinki.fi/u/mluukkai/ohmas10/usecase.pdf

Toinen esimerkki: pankkiautomaatin käyttötapaukset

- Pankkiautomaatin käyttötapaukset ovat *tunnistaudu, saldo* ja *otto*
- Käyttötapausten *käyttäjät* eli toimintaan osallistuvat tahot ovat *Asiakas* ja *Pankki*
  - Alla on esitelty tikku-ukolle vaihtoehtoinen tapa merkitä käyttäjä eli laatikko, jossa merkintä <<actor>>
  - Tämä lienee luontevampi jos käyttäjä ei ole ihminen
- Seuraavalla kalvolla käyttötapauksen *Otto* tekstuaalinen kuvaus
  - Huomaa, että esiehto edellyttää, että käyttötapaus tunnistaudu on suoritettu



Käyttötapaus 1: otto

Tavoite: asiakas nostaa tililtään haluamansa määrän rahaa

Käyttäjät: asiakas, pankki

Esiehto: kortti syötetty ja asiakas tunnistautunut

Jälkiehto: käyttäjä saa tililtään haluamansa määrän rahaa

Jos saldo ei riitä, tiliä ei veloiteta

Käyttötapauksen kulku:

1 asiakas valitsee otto-toiminnon

2 automaatti kysyy nostettavaa summaa

3 asiakas syöttää haluamansa summan

4 pankilta tarkistetaan riittääkö asiakkaan saldo

5 summa veloitetaan asiakkaan tililtä

6 kuitti tulostetaan ja annetaan asiakkaalle

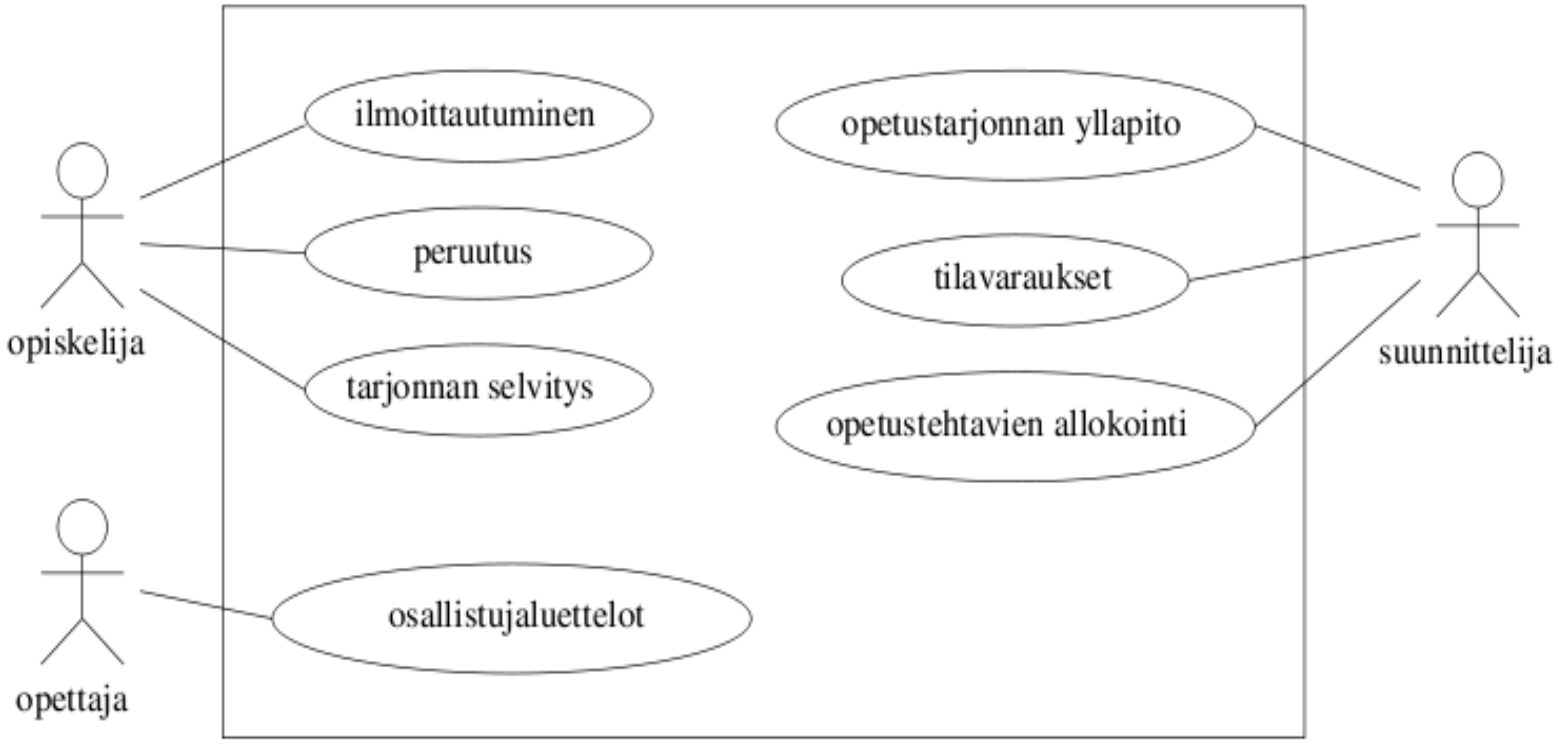
7 rahat annetaan asiakkaalle

8 pankkikortti palautetaan asiakkaalle

Poikkeuksellinen toiminta:

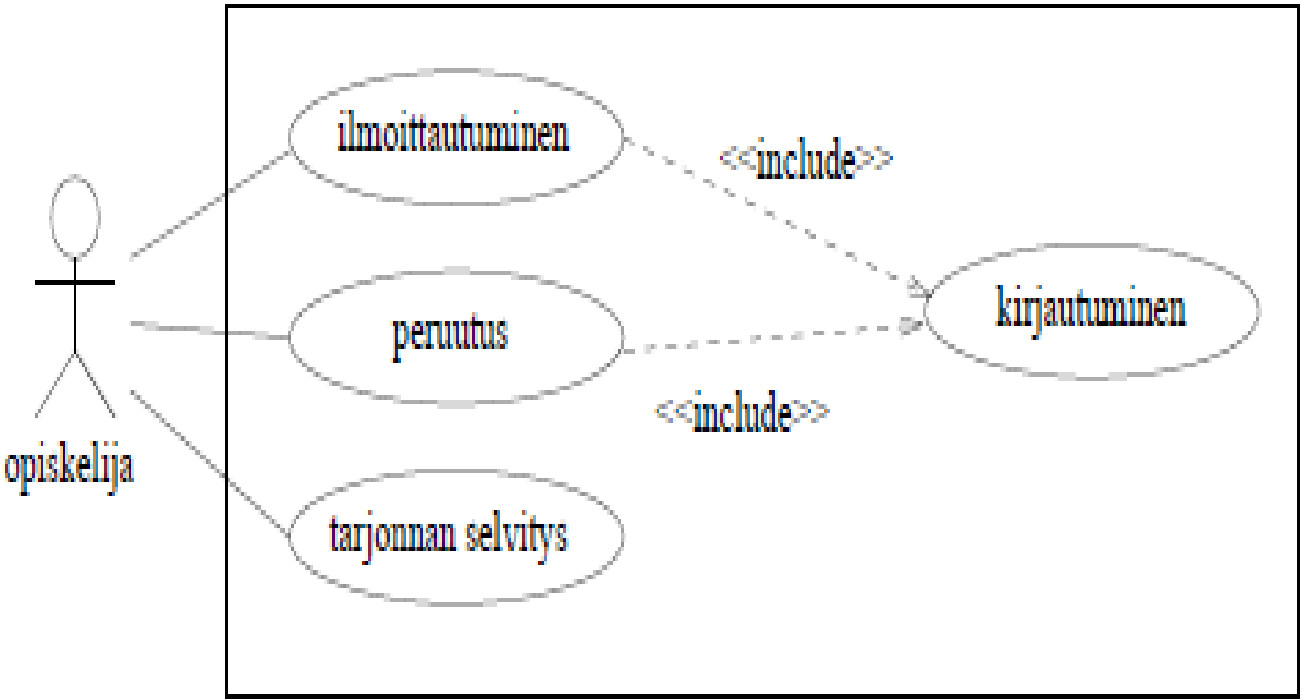
4a asiakkaan tilillä ei tarpeeksi rahaa, palautetaan kortti asiakkaalle

Palataan takaisin kurssi-ilmoittautumisjärjestelmään



Yhteiset osat

- Moneen käyttötapaukseen saattaa liittyä yhteinen osa
- Yhteisestä osasta voidaan tehdä “alikäyttötapaus”, joka *sisällytetään* (include) pääkäyttötapaukseen
- Käyttötapauskaaviossa tätä varten merkintä <<include>>
  - katkoviivanuoli pääkäyttötapauksesta apukäyttötapaukseen
- Esim. käyttötapaus *kirjautuminen* suoritetaan aina kun tehdään *ilmoittautuminen* tai *peruutus*



Yhteiset osat ja include

- Apukäyttötapauksen sisällytys on tärkeä ilmaista käyttötapauksen tekstuaalisessa kuvauksessa
  - Muuten ei tietoa missä kohtaan sisällytys tapahtuu

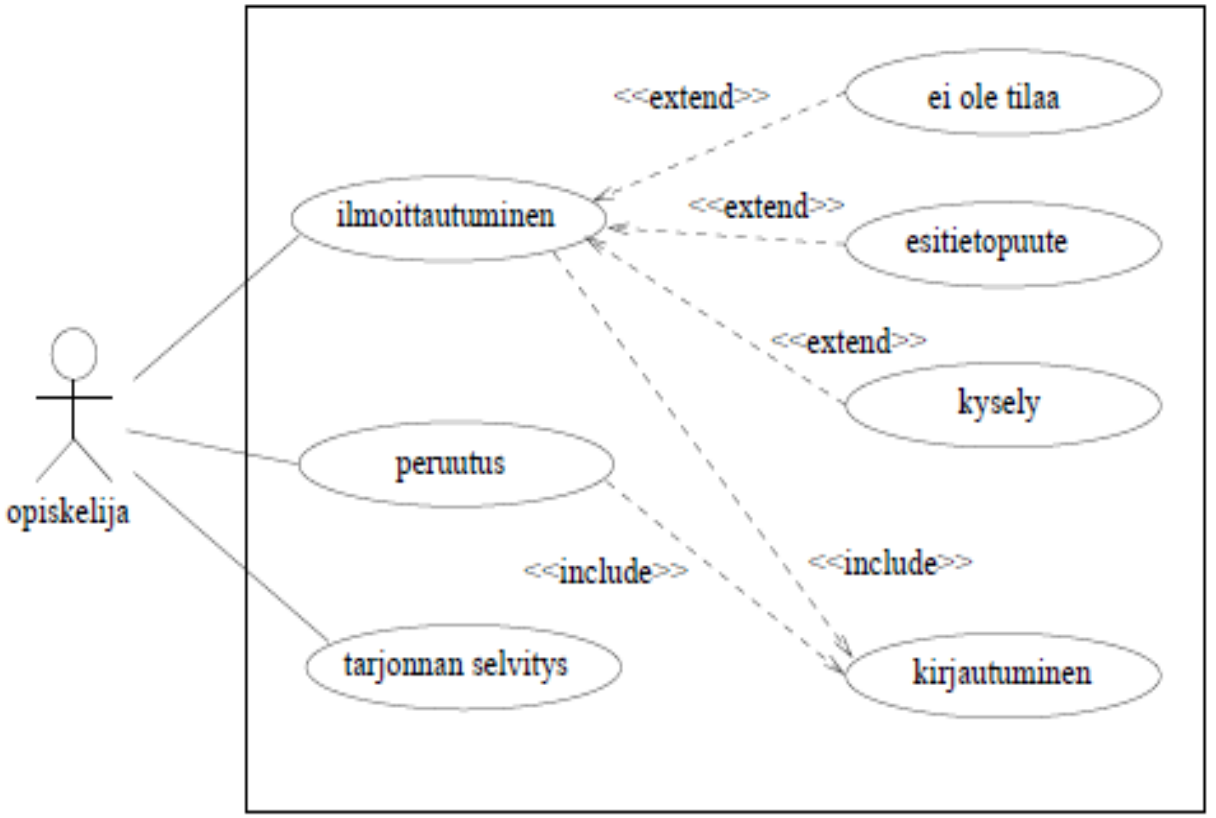
- *Käyttäjä:* opiskelija
- *Tavoite:* saada kurssipaikka
- *Laukaisija:* opiskelijan tarve
- *Esiehto:* opiskelija on ilmoittautunut kuluvalla lukukaudella läsnäolevaksi
- *Jälkiehto:* opiskelija on lisätty haluamansa ryhmän ilmoittautujien listalle
- *Käyttötapauksen kulku:*

  1. Opiskelija aloittaa kurssi-ilmoittautumistoiminnon
  2. Järjestelmä näyttää kurssitarjonnan
  3. Opiskelija tutkii kurssitarjontaa
  4. Opiskeljia valitsee ohjelmiston esittämästä tarjonnasta kurssin ja ryhmän
  5. **Suoritetaan käyttötapaus kirjautuminen**
  6. Järjestelmä ilmoittaa opiskelijalle ilmoittautumisen onnistumisesta.

- *Poikkeuksellinen toiminta:*

Poikkeustilanteet ja laajennukset

- Sisällytettävä (eli include) *käyttötapaus suoritetaan aina* pääkäyttötapauksen suorituksen yhteydessä
- Myös tarvittaessa suoritettava *laajennus tai poikkeustilanne* voidaan kuvata apukäyttötapauksena, joka *laajentaa* (extend) pääkäyttötapausta
  - Laajennus suoritetaan siis vaan *tarvittaessa*
- Esim. Ilmoittautuessa saatetaan huomata esitietopuute, jonka käsittely on oma käyttötapauksensa



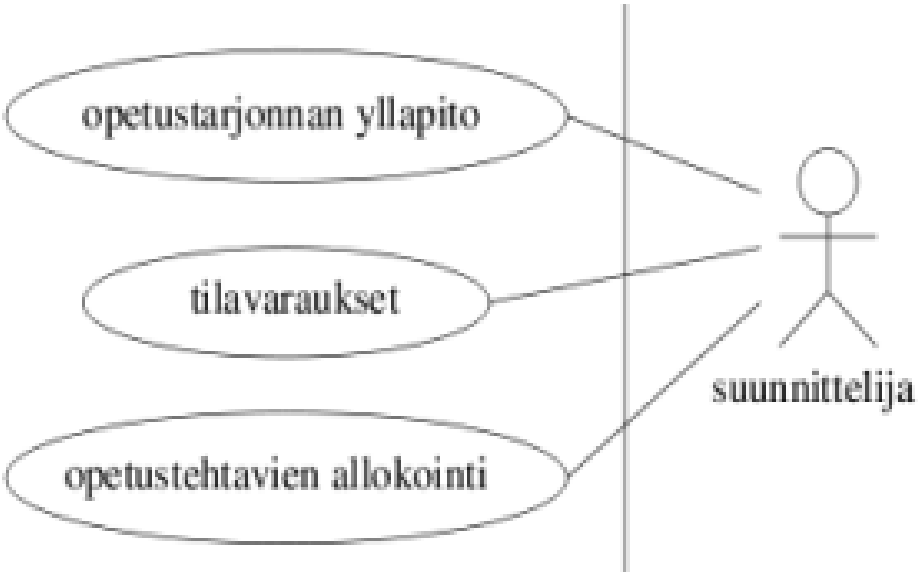
Poikkeustilanteet ja laajennukset

- Huomaa, että laajennuksessa nuolensuunta on apukäyttötapauksesta pääkäyttötapaukseen päin (toisin kuin sisällytyksessä)
- Myös laajennus tulee ehdottomasti merkitä käyttötapauksen tekstuaaliseen kuvaukseen
- Edellisen sivun laajennusesimerkki ei ole erityisen onnistunut
  - Laajennuksienkin pitäisi olla kunnollisia käyttötapauksia (eli asioita joilla on selkeä tavoite), ei metodikutsumaisia kyselyjä tai ilmoituksia (kuten *ei tilaa* - tai *esitietopuute*-ilmoitus)
  - Poikkeustilanteet on parempi kuvata tekstuaalisessa esityksessä ja jättää ne kokonaan pois käyttötapauskaavioista

- Koko laajennuskäsitteen tarve käyttötapauskaavioissa on hieman kyseenalainen

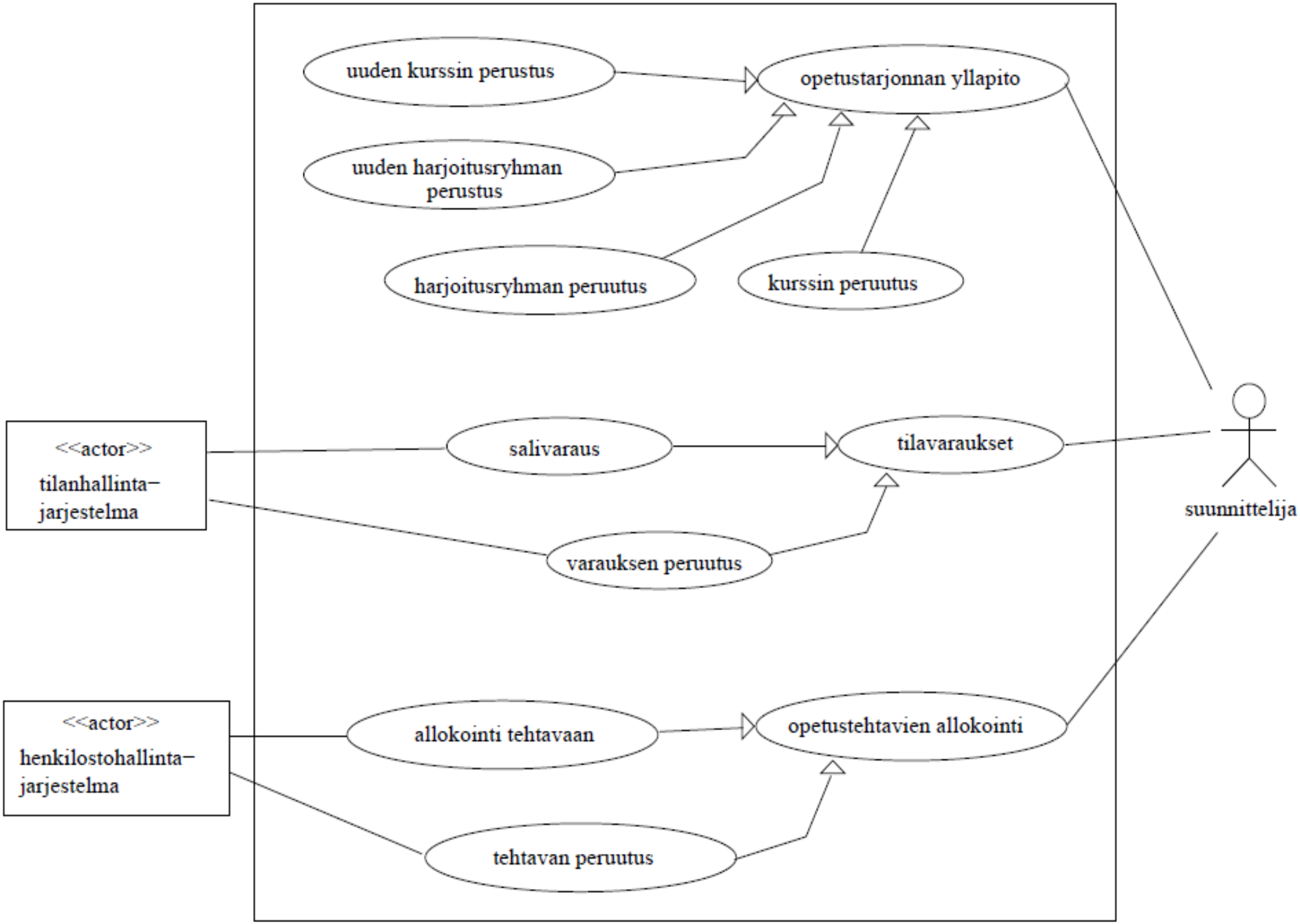
Yleistetty ja erikoistettu käyttötapaus

- Suunnittelijan käyttötapauksista erityisesi opetustarjonnan ylläpito on hyvin laaja tehtäväkokonaisuus
- Voidaankin ajatella, että kyseessä on *yleistetty käyttötapaus,* joka oikeasti pitääkin sisällään useita konkreettisia käyttötapauksia, kuten
  - Uuden kurssin perustus
  - Uuden harjoitusryhmän perustus
  - Harjoitusryhmän peruutus
  - Kurssin peruutus



- Seuraavan sivun kaaviossa suunnittelijan käyttötapaukset tarkemmalla tasolla (huomioi miten yleistys merkitään)
- Mukana myös ulkoiset järjestelmät, *tilanhallintajärjestelmä* ja *henkilöstöhallintajärjestelmä,* jotka osallistuvat käyttötapauksiin

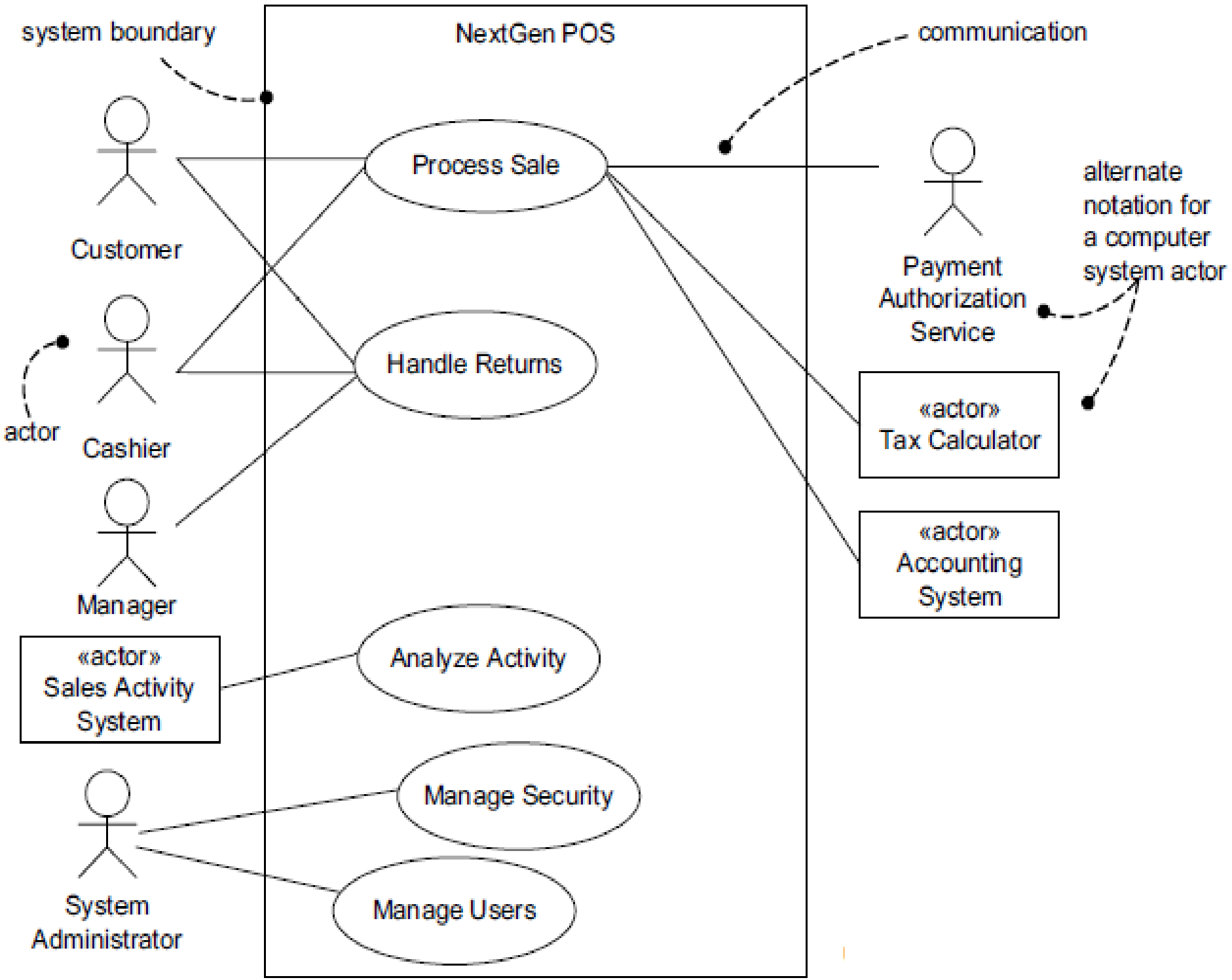
Yleistetyt käyttötapaukset käyttötapauskaaviona



Realistisempi esimerkki: kassapäätejärjestelmä

- Craig Larmanin kirjasta *Applying UML and Patterns*
- Kirjan käyttötapausluku löytyy verkosta
  - http://www.craiglarman.com/wiki/index.php?title=Articles

- Aluksi etsitään järjestelmän käyttäjät
- Mietitään käyttäjien tavoitteita: *mitä käyttäjä haluaa saada järjestelmällä tehtyä*
  - Käyttäjän tavoitteellisista toiminnoista (esim. käsittele ostos) tulee tyypillisesti käyttötapauksia
  - Samalla saatetaan löytää uusia käyttäjiä (erityisesti ulkoisia järjestelmiä joihin järjestelmä yhteydessä)
- Hahmotellaan alustava käyttötapausdiagrammi
  - ks. seuraava sivu


system boundary
NextGen POS
communication
Process Sale
Customer
Payment
Authorization
Service
alternate
notation for
a computer
system actor
Handle Returns
actor
Cashier
«actor»
Tax Calculator
«actor»
Accounting
System
Manager
«actor»
Sales Activity
System
Analyze Activity
Manage Security
System
Administrator
Manage Users

Käyttötapauksen tarkentaminen

- Otetaan aluksi tarkasteluun järjestelmän toiminnan kannalta kriittisimmät käyttötapaukset
- Ensin kannattanee tehdä vapaamuotoinen kuvaus käyttötapauksista (“brief use case”)
  - POS, point of sales terminal eli kassapääte

Process Sale: A customer arrives at a checkout with items to purchase. The cashier uses the POS system to record each purchased item. The system presents a running total and line-item details. The customer enters payment information, which the system validates and records. The system updates inventory. The customer receives a receipt from the system and then leaves with the items.

- Tarkempi käyttötapaus kirjoitetaan projektin sopiman käyttötapauspohjan määräämässä muodossa

Use Case UC1: Process Sale

Primary Actor: Cashier

Preconditions: Cashier is identified and authenticated.

Success Guarantee (Postconditions): Sale is saved. Tax is correctly calculated. Accounting and Inventory are updated. Commissions recorded. Receipt is generated. Payment authorization approvals are recorded.

Main Success Scenario (or Basic Flow):

1. Customer arrives at POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total. Price calculated from a set of price rules.

Cashier repeats steps 3-4 until indicates done.

5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any).

- Kuvaus jatkuu: Laajennukset, tarkennukset ja poikkeukset

Extensions (or Alternative Flows):

*a. At any time, System fails:

To support recovery and correct accounting, ensure all transaction sensitive state and events can be recovered from any step of the scenario.

1. Cashier restarts System, logs in, and requests recovery of prior state.
2. System reconstructs prior state.
   2a. System detects anomalies preventing recovery:
      1. System signals error to the Cashier, records the error, and enters a clean state.
      2. Cashier starts a new sale.

3a. Invalid identifier:

1. System signals error and rejects entry.

3b. There are multiple of same item category and tracking unique item identity not important (e.g., 5 packages of veggie-burgers):

1. Cashier can enter item category identifier and the quantity.

3-6a: Customer asks Cashier to remove an item from the purchase:

1. Cashier enters item identifier for removal from sale.
2. System displays updated running total.

3-6b. Customer tells Cashier to cancel sale:

1. Cashier cancels sale on System.

3-6c. Cashier suspends the sale:

1. System records sale so that it is available for retrieval on any POS terminal.

- Tarkennuksia:

7a. Paying by cash:
1. Cashier enters the cash amount tendered.
2. System presents the balance due, and releases the cash drawer.
3. Cashier deposits cash tendered and returns balance in cash to Customer.
4. System records the cash payment.

7b. Paying by credit:
1. Customer enters their credit account information.
2. System sends payment authorization request to an external Payment Authorization Service System, and requests payment approval.
   2a. System detects failure to collaborate with external system:
      1. System signals error to Cashier.
      2. Cashier asks Customer for alternate payment.
3. System receives payment approval and signals approval to Cashier.
   3a. System receives payment denial:
      1. System signals denial to Cashier.
      2. Cashier asks Customer for alternate payment.
4. System records the credit payment, which includes the payment approval.
5. System presents credit payment signature input mechanism.
6. Cashier asks Customer for a credit payment signature. Customer enters signature.

Tarkkaan kuvattu käyttötapaus

- Esimerkin mallin mukaan käyttötapauksen *pääkulku kannattaa kuvata tiiviisti*
  - Eri askeleiden sisältöä voi tarvittaessa tarkentaa
    - Kuten edellisellä sivulla tarkennettu askel 7 “customer pays...”
- Huomioi tapa, miten poikkeusten ja laajennusten sijainti pääkulussa merkitään
    - 7a => laajentaa/tarkentaa pääkulun kohtaa 7
- Osa jossa laajennukset, tarkennukset ja poikkeukset dokumentoidaan, on usein paljon pidempi kuin normaali kulku
- Koska kyse *vaatimusmäärittelystä*, kuvaus on abstraktilla tasolla
    - *Ei oteta kantaa toteutusyksityiskohtiin*
    - eikä käyttöliittymään
    - Esim. tunnistetaanko ostos viivakoodin perusteella...

Yhteenveto

- Käyttötapaukset ovat yksi tapa kuvata ohjelmiston toiminnallisia vaatimuksia
- **Käyttötapauksen tekstuaalinen esitys oleellinen**
- Ohjelmistoprojektissa pitää sopia yhteinen tapa (*käyttötapauspohja*) käyttötapausten tekstuaaliseen esitykseen
- Käyttötapauskaavion merkitys lähinnä yleiskuvan antaja

- Jos käytät huomaat käyttäväsi paljon aikaa “oikeaoppisen” käyttötapauskaavion piirtämiseen, ryhdy välittömästi tekemään jotakin hyödyllisempää (esim. käyttötapausten tekstuaalisia esityksiä)

Hieman yksikkötestauksesta

Testaustasot

- Ohjelmiston elinkaareen kuuluvat vaiheet ovat siis
    - Määrittely
    - Suunnittelu
    - Toteutus
    - Testaus
    - Ylläpito
- Testauskin jakautuu vaiheisiin joita kutsutaan myös testaustasoiksi:
    - **Yksikkötestaus**
        - Toimivatko yksittäiset metodit ja luokat kuten halutaan?
    - **Integraatiotestaus**
        - Varmistetaan komponenttien yhteentoimivuus
    - **Järjestelmä/hyväksymistestaus**
        - Toimiiko kokonaisuus niin kuin vaatimusdokumentissa sanotaan?
- **Regressiotestauksella** tarkoitetaan järjestelmälle muutosten ja bugikorjausten jälkeen ajettavia testejä jotka varmistavat että muutokset eivät riko mitään ehjää
    - Regressiotestit koostuvat yleensä yksikkö-, integraatio- ja järjestelmä/hyväksymätesteistä

Yksikkötestit

- Yksikkötestit ovat alimman tason testejä ja ne kohdistuvat useimmiten yhden luokan yhteen metodiin
- Yksittäistä testiä sanotaan testitapaukseksi (test case)
- Yksikkötesteissä on yleensä jokaista testattavan luokan metodia kohti *vähintään* yksi testitapaus, yleensä useampia
- Yksikkötestauksessa pyritään mahdollisimman hyvään *kattavuuteen*
- Kattavuutta voidaan mitata esimerkiksi sillä kuinka suurta osaa koodiriveistä testit tutkivat
- Normaalien syötteiden lisäksi on testeissä erityisen tärkeää tutkia testattavien metodien toimintaa virheellisillä syötteillä ja erilaisilla raja-arvoilla
- Esim. laskareissa testataan Ohpe-kurssin luokkaa Lyyrakortti
  - virheellinen syöte voisi olla esim. yritys ladata kortille negatiivinen määrä rahaa
  - Raja-arvo taas olisi edullisen lounaan ostaminen kun kortilla oleva rahamäärä on tasan edullisen lounaan hinta

Testauksen automatisointi

- Manuaalisesti tapahtuva testaus on toivottoman työlästä
  - Erityisesti sen takia, että ei riitä että ohjelma testataan kerran, jokaisen muutoksen jälkeen on tehtävä regressiotestaus joka varmistaa että muutos ei riko mitään
- Testit kannattaa siis tehdä koodiksi joka voidaan ajaa siten, että testikoodi varmistaa automaattisesti testattavan koodin toiminnan
  - Testikoodin ajamisen täytyy olla helppoa, ”nappia painamalla” tapahtuvaa
- xUnit-testauskehys on automatisoidun yksikkötestauksen defacto-standardi
  - JUnit on Javalle tarkoitettu xUnitin versio
- JUnitin peruskäyttö on todella helppoa erityisesti modernien kehitysympäristöjen (Netbeans, Eclipse, IntelliJ) yhteydessä, ks.
  - https://github.com/mluukkai/OTM2013/wiki/Ohje-JUnit:in-k%C3%A4ytt%C
- Ohpessa käytetyn TMC:n testit ovat JUnit-testejä
  - Opettelemme laskareissa tekemään JUnit-testejä

Debuggeri

Ohjelmien debuggaus

- Debuggaus eli virheiden löytäminen ohjemista ei ole aina helppoa
- Yksi perinteisesti käytetty debuggauskeino on aputulostusten lisäily ohjelmakoodiin
  - Aputulosten avulla pyritään varmistumaan, että ohjelman tila on se mikä ohjelmoija sen olettaa olevan
- Aputulostuksia kehittyneempi tapa virheenjäljittämiseen on debuggeri.
- Debuggerin avulla on mahdollista suorittaa ohjelmaa askeltaen, komento kerrallaan ja samalla tarkastella ohjelman muuttujien arvoja
- Moderneissa ohjelmointiympäristöissä (Netbeans, Eclipse, IntelliJ) on integroidut suhteleellisen helppokäyttöiset debuggerit
- Tutustumme Netbeansin debuggeriin laskareissa
  - https://github.com/mluukkai/OTM2013/wiki/Ohje-debuggerin-k%C