Ohjelmistojen mallintaminen

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 luentomateriaali:
  - http://www.cs.helsinki.fi/u/mluukkai/ohmas10/
- Laskarit
  - Aloitetaan jo tällä viikolla
  - Viikossa noin 5 kotona tehtävää tehtävää ja muutama paikalla yhdessä tehtävä tehtävä
- Kurssin arvostelu
  - Kurssikoe 28 pistettä
  - Laskarit 8 pistettä
  - 18p => 1, ..., noin 30p => 5
  - Läpipääsyyn vaaditaan lisäksi **puolet kurssikokeen pisteitä**

Muuta

- English speaking students, please contact me immediately
- Kommunikaatio
  - Kurssin kotisivu
  - IRC #ohma10
  - Kurssiblogi (ehkä tulee, ehkä ei)
- Apua laskareihin:
    - Neuvontapaja, ks. http://www.cs.helsinki.fi/opiskelu/neuvontapaja
  - Tällä kurssilla ei ole pakollisia opintopiirejä
  - Vapaaehtoisia opintopiirejä kannattaa muodostaa esim. ensimmäisessä laskarissa
  - Myös tehtävien tekeminen kaveriporukassa on erittäin suositeltavaa

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

- 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

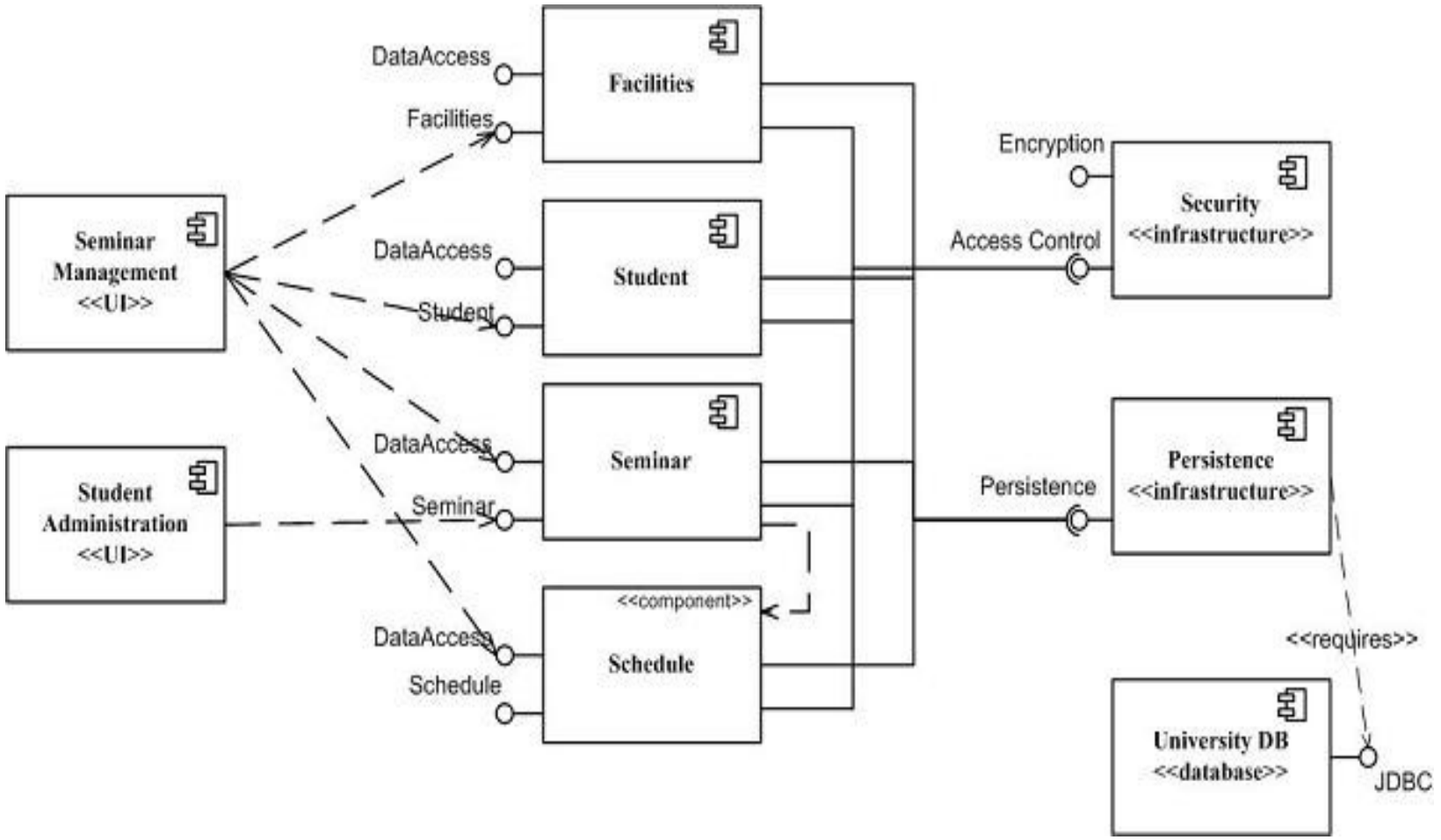
- Jotta toteuttajat ymmärtäisivät mitä pitää tehdä, joudutaan ongelma-aluetta **analysoimaan**
  - Esim. Jäsennetään ongelma-alueen käsitteistöä
  - Tehdään ongelma-alueesta **malli** eli yksinkertaistettu kuvaus

- Vaatimusmäärittelyn päätteeksi yleensä tuotetaan **määrittelydokumentti**
  - Kirjaa sen mitä ohjelmalta halutaan
    - Ohjeena suunnitteluun ja toteutukseen
    - Testaus perustuu dokumentissa asetettuihin ohjelman vaatimuksiin

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

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

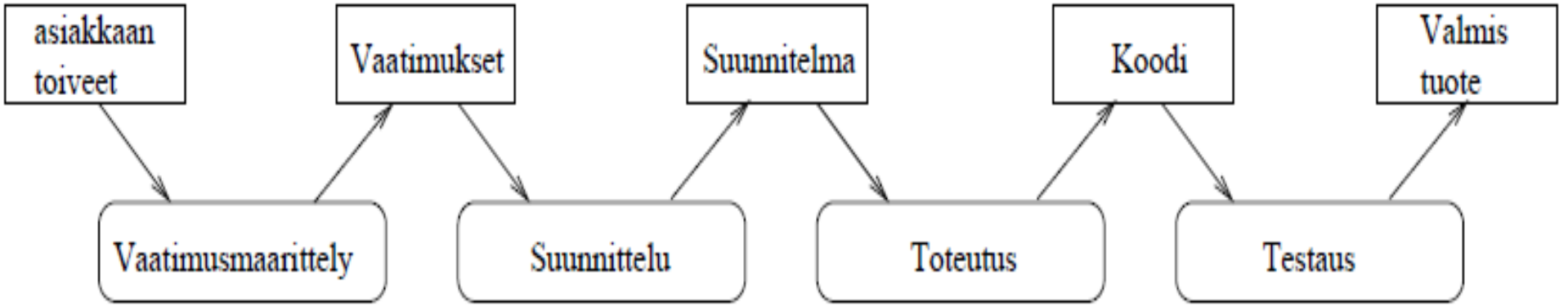


Toteutus, testaus ja ylläpito

- Suunnitteludokumentin mukainen järjestelmä toteutetaan valittuja tekniikoita käyttäen
- Toteutuksen yhteydessä testataan:
  - Yksikkötestaus
    - Toimivatko yksittäiset metodit ja luokat?
  - Integrointitestaus
    - 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**

Vesiputousmalli



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

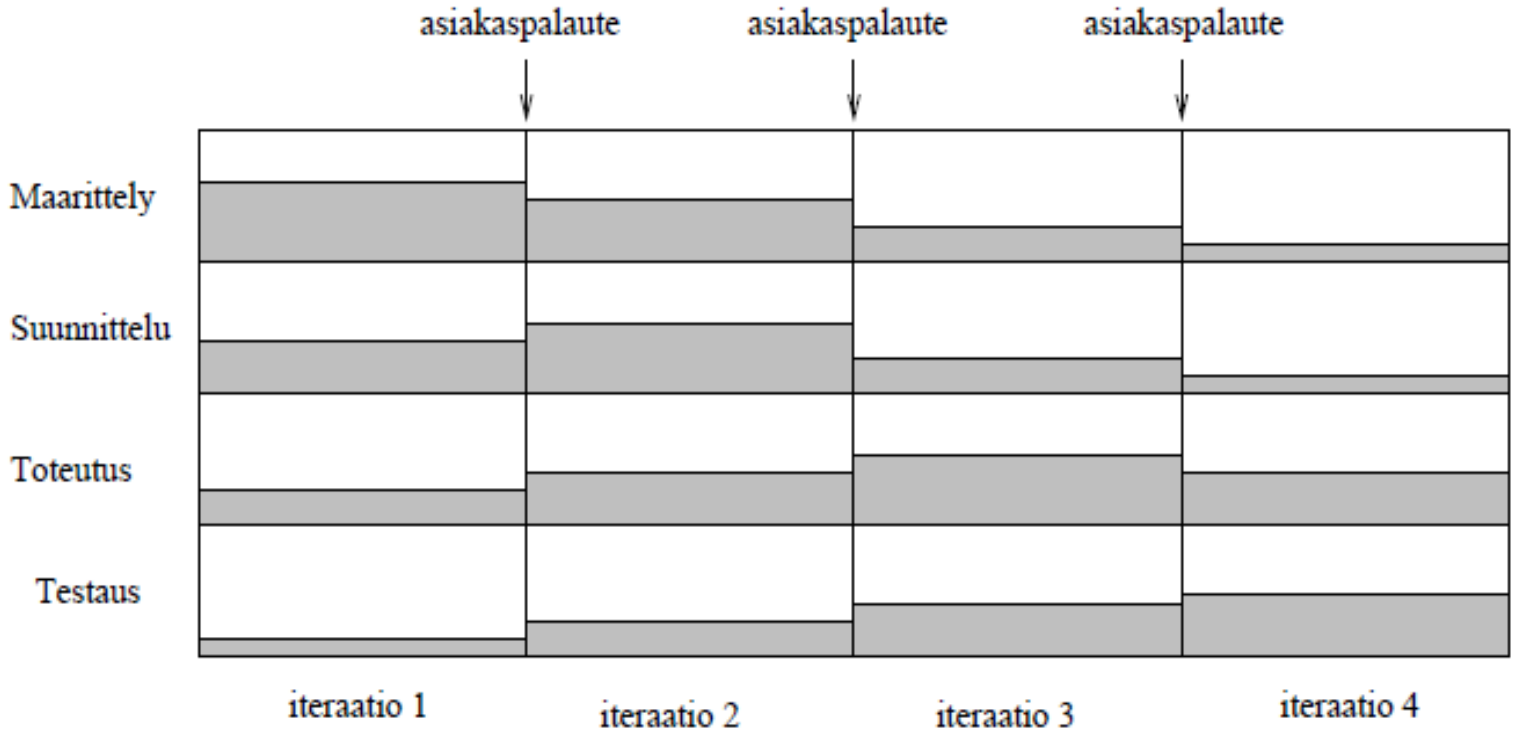
Vesiputousmallin ongelmia

- Järjestelmää testataan 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 toteuttajan välinen kommunikointi
  - kärjistyy vesiputousmallissa koska palautetta hankala saada kesken projektin

Ketterä ohjelmistokehitys

- *Lähdetään olettamuksesta, että asiakkaan vaatimukset muuttuvat 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

- 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

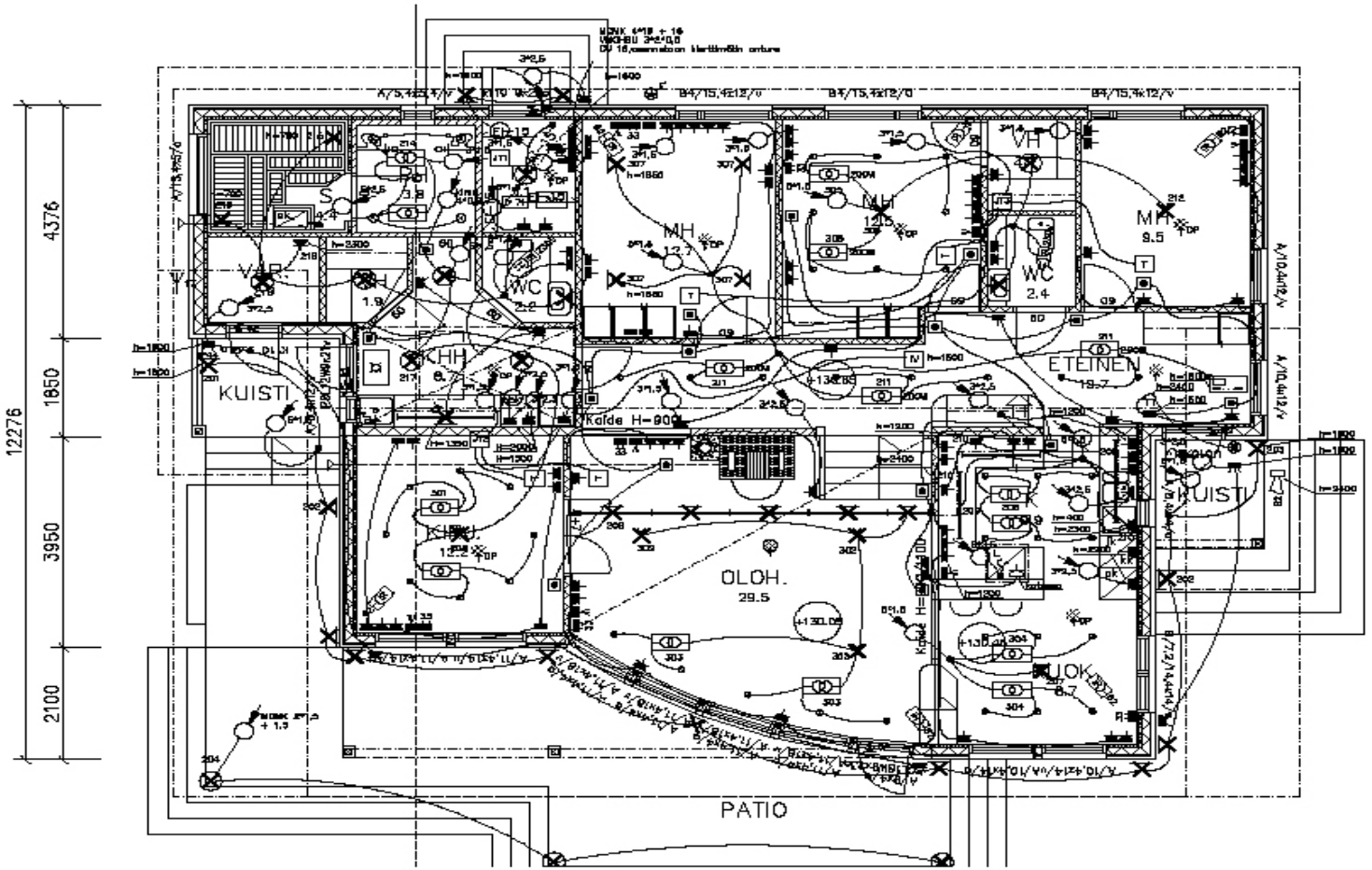
- Samaa kohdetta voidaan mallintaa monesta eri **näkökulmasta**
  - Jos kaikki yritetään mahduttaa samaan malliin, ei lopputulos ole selkeä
  - Malli kuvaa usein korostetusti tiettyä näkökulmaa
  - Eri näkökulmat yhdistämällä saadaan idea kokonaisuudesta

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 suunnitelma
- 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

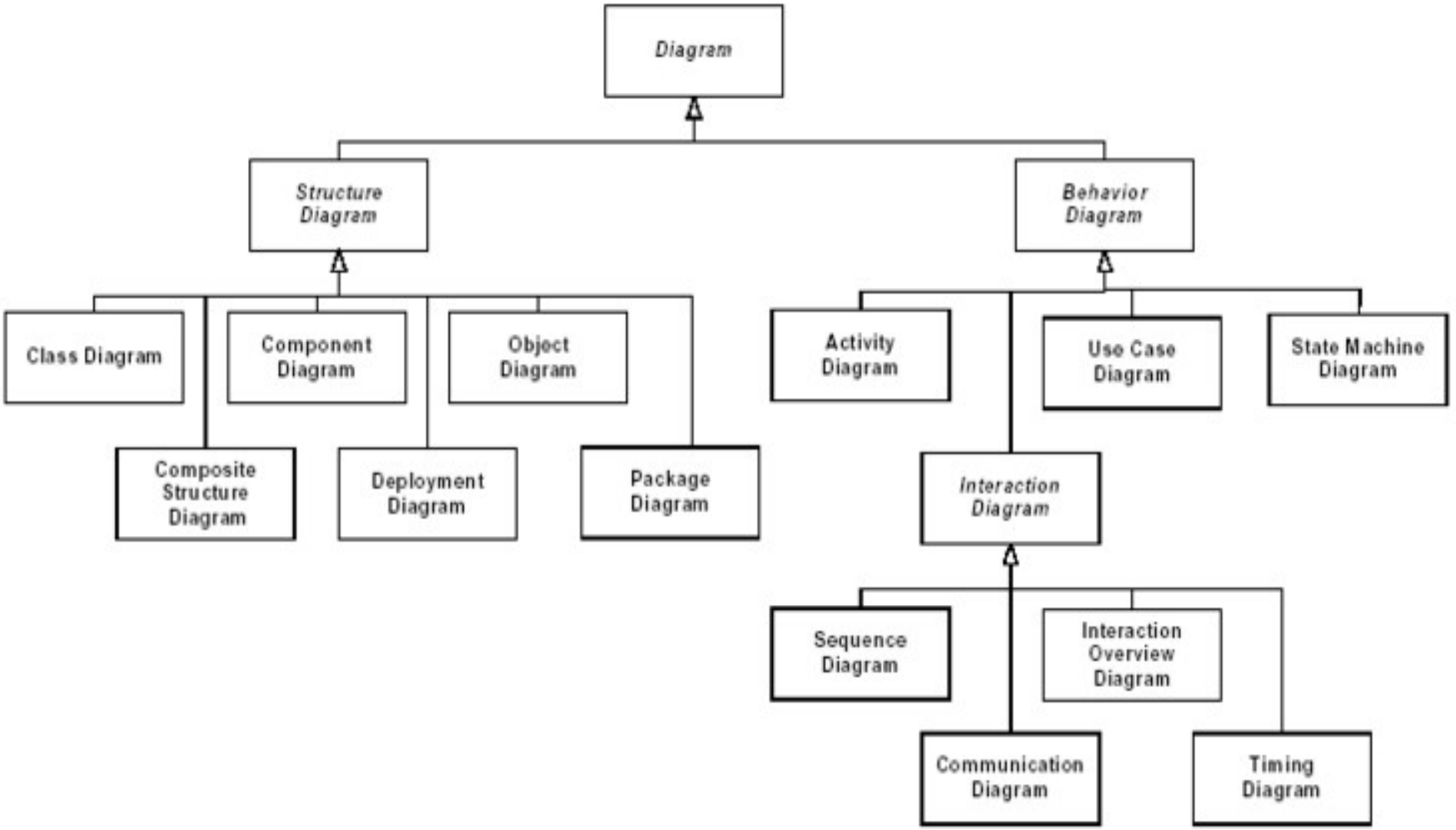
- Pitkään tilanne oli sekava ja on sitä osin edelleen
- Suosituimmaksi tavaksi on noussut **oliomallinnus**
  - Perustuu seuraavaan oletukseen:
  - *Minkä tahansa järjestelmän katsotaan voivan muodostua olioista, jotka yhteistyössä toimien ja toistensa palveluja hyödyntäen tuottavat järjestelmän tarjoamat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

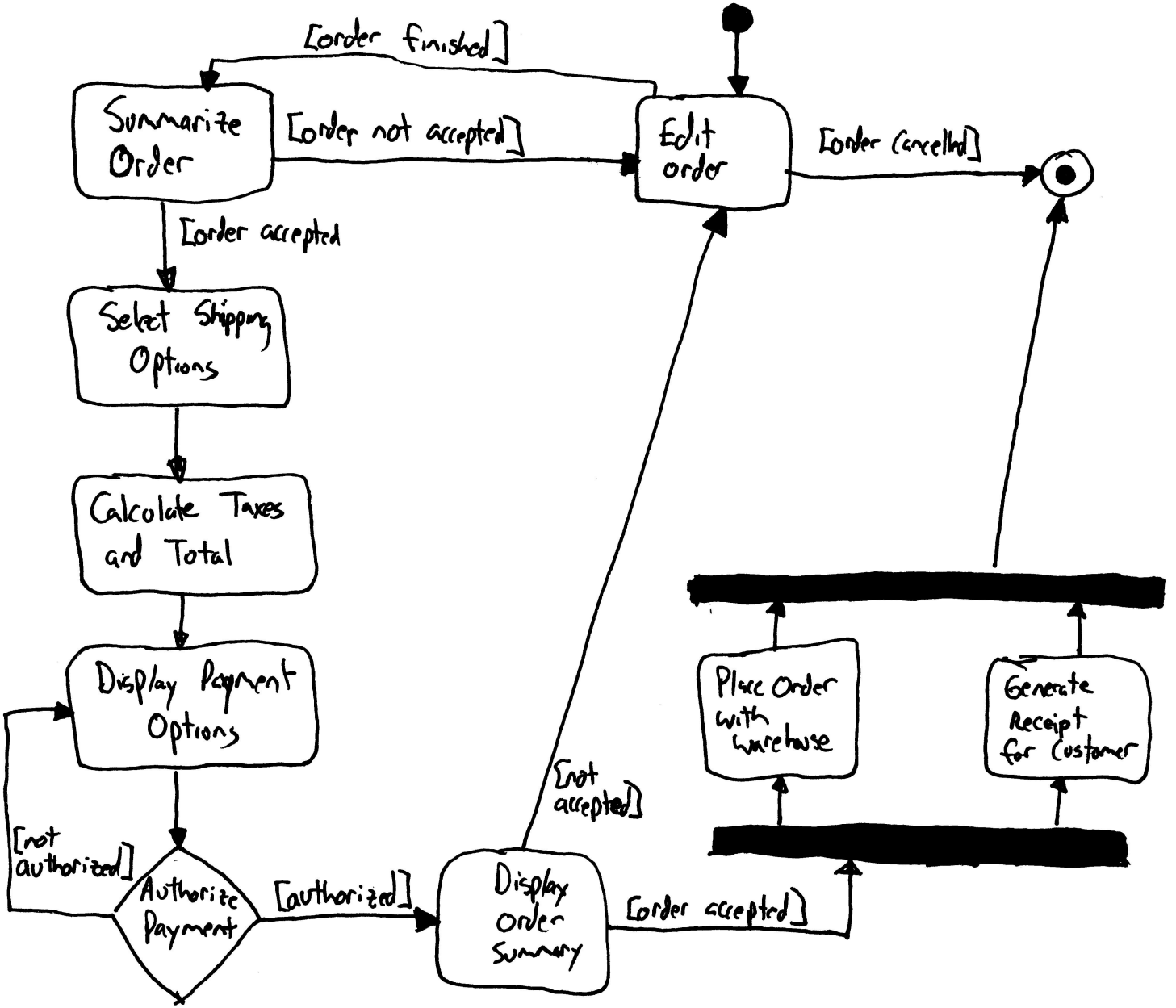
UML:n kaaviotyypit



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



Kurssin sisältö

- Käyttötapausmalli: UML:n käyttötapauskaaviot
  - Käyttäjien vaatimusten dokumentointi
- Luokkamalli: UML:n luokkakaaviot
  - Tapa määritellä luokkia ja niiden välisiä suhteita
- Olioiden yhteistoiminnan kuvaaminen
  - UML:n sekvenssikaaviot ja kommunikaatiokaaviot
- UML:n soveltaminen ohjelmiston kehittämisessä
  - Vaatimusmäärittely
  - Suunnittelu
    - Arkkitehtuurisuunnittelu ja oliosuunnittelu
  - Toteutus
  - Miten UML liittyy kaikkeen tähän?
- UML:n tila- ja aktiviteettikaaviot
  - Hieman vähemmän käytettyjä mutta joskus hyödyllisiä kaavioita