

Ohjelmistotekniikka - Luento 6

Ilkka Tervonen

Luku 10: Komponenttitason suunnittelu

komponentin määrittely

luokkapohjaisten komponenttien suunnittelu

komponenttipohjainen ohjelmistotekniikka (CBSE)

CORBA ja SOA

tuoterunkoon perustuva ohjelmistokehitysprosessi

Luku 11: Käyttöliittymän suunnittelu

kultaiset säännöt

käyttöliittymän suunnitteluprosessi

käyttöliittymän suunnitteluperiaatteet

Soveltuvat lait ja *pohdiskelun aiheita*

1. COTS perustainen ohjelmistokehitys ei poista kehitysprosessin riskejä / hyp_no 7, Basili - Boehm 2001
 - *Mitä tarkoitetaan ohjelmistokomponentilla? Anna muutamia esimerkkejä.*
 - *tai*
Mitkä ovat CBSE:n peruskäsitteet ja periaatteet Crnkovic et al. mukaan?
 - *Li et al. esittävät 10 faktaa, jotka liittyvät OTS (off-the-self) perustaisen kehittämisen teollisiin käytänteisiin. Esittele niistä viisi tarkemmin, eli esittele myös perustelut näille viidelle faktalle.*
 - *Mitä tarkoittavat pilvilaskenta-arkkitehtuurin kolme tasoa; SaaS, PaaS ja IaaS (Yau & An mukaan)?*
 - *Miten ohjelmistokehitys tapahtuu palvelupohjaisessa ohjelmistotekniikassa (SOSE) (Yau & An mukaan)?*
 - *Mitä haasteita Yau & An ovat havainneet ohjelmistokehityksessä, kun käytetään palvelupohjaista ohjelmistotekniikkaa (SOSE)?*

Mikä on komponentti?

- *OMG Unified Modeling Language Specification [OMG01]* määrittelee komponentin
 - “... a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces.”
- **oliopuolella:** komponentti koostuu joukosta yhteistyössä toimivia luokkia
- **perinteisesti:** ohjelmiston osa, joka sisältää toimintalogiikan kuvauksen, sen tarvitsemat sisäiset tietorakenteet ja rajapinnan, jonka kautta se voidaan käynnistää ja jonka kautta tietoa siirretään
- **yleisesti:** itsenäinen ohjelmayksikkö, joka tarjoaa palveluja rajapinnan kautta

- **Komponentit ovat ohjelmistoarkkitehtuurissa käsiteltäviä pienimpiä yksiköitä: raja arkkitehtuurisuunnittelun ja yksityiskohtaisen suunnittelun välillä.**
- **Suuri komponentti voi kuitenkin koostua useista pienemmistä komponenteista, jolloin komponentilla on oma sisäinen arkkitehtuurinsa**
 - **esim. Eclipsen kaltainen ohjelmointiympäristö ja siihen integroidut työkalut**
- **Komponentin sisäisen rakenteen ei pitäisi näkyä ulospäin eikä toiminnan riippua käyttöympäristöstä (muuten kuin vaadittujen rajapintojen kautta)**

- **Komponentti muodostaa toiminnallisen kokonaisuuden, joka tarjoaa joukon loogisesti toisiinsa liittyviä palveluja**
 - palvelut käyttävät samoja tietorakenteita
 - palveluja käytetään tyypillisesti samassa yhteydessä
- **Toiminnallisuus on peruste komponentin olemassaololle.**
- **Komponenteilla hallitaan järjestelmän muunneltavuutta: eristetään järjestelmän osat, jotka halutaan helposti päivitettäväksi**
- **Komponenttityyppejä: esim. plugin-komponentit ja COTS (Commercial Off-The-Shelf)-komponentit**

- **Komponentti on työnjaon perusyksikkö kaikissa kehitystyön vaiheissa**
 - esim. suunnittelu, toteutus, testaus, konfiguraation hallinta, ylläpito
- **Yksi komponentti yhden henkilön vastuulle.**
- **Työryhmä kehittää toisiinsa liittyviä komponentteja (esim. tiettyä alijärjestelmää).**
- **Järjestelmää kehittävän organisaation rakenne alkaa muistuttaa järjestelmän rakennetta (tai päinvastoin: vrt. Conway:n laki)**

Komponentin ominaispiirteet

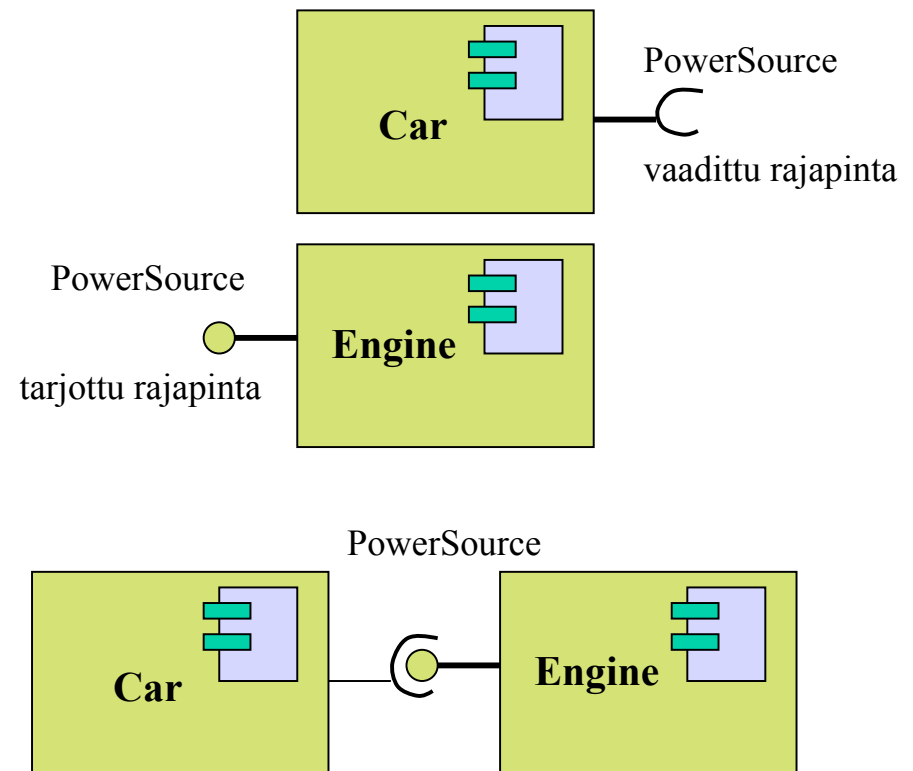
- **Riippuvuudet**
 - komponentti voi riippua palveluista, joita se olettaa saavansa muilta komponenteilta vaadittujen rajapintojen kautta
- **Käyttöönotto**
 - komponentti voidaan ottaa käyttöön lähdekoodisena (käännösaikana) tai käännettynä binäärimuodossa (linkkausaikana)
- **Koko**
 - komponentin tulisi olla yhden henkilön hallittavissa
- **Standardointi**
 - esimerkkinä Sunin EJB-komponenttistandardi
 - » *Koskimies, Mikkonen 2005*

Komponenttien välinen vuorovaikutus

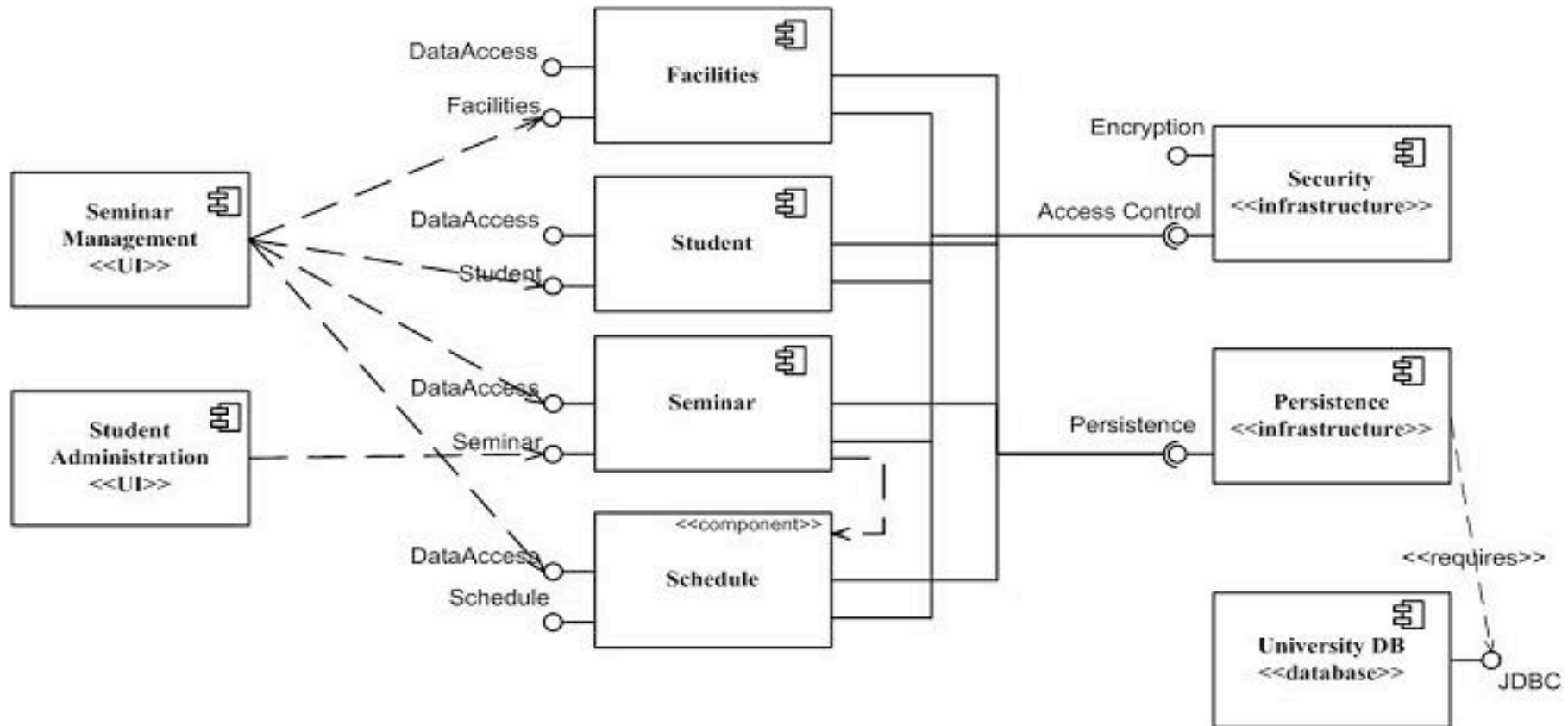
- **Rajapinta**

- kertoo, miten palvelu otetaan käyttöön
- palvelun nimi, parametrit ja niiden tyypit ja mahdollisen tuloksen tyyppi = kutsumuoto (signature)
- palvelun merkitys kuvataan tulo- ja jättöehtojen (pre/post condition) avulla

» *Koskimies, Mikkonen 2005*



Esimerkki komponenttimallista: osa yliopisto-järjestelmästä

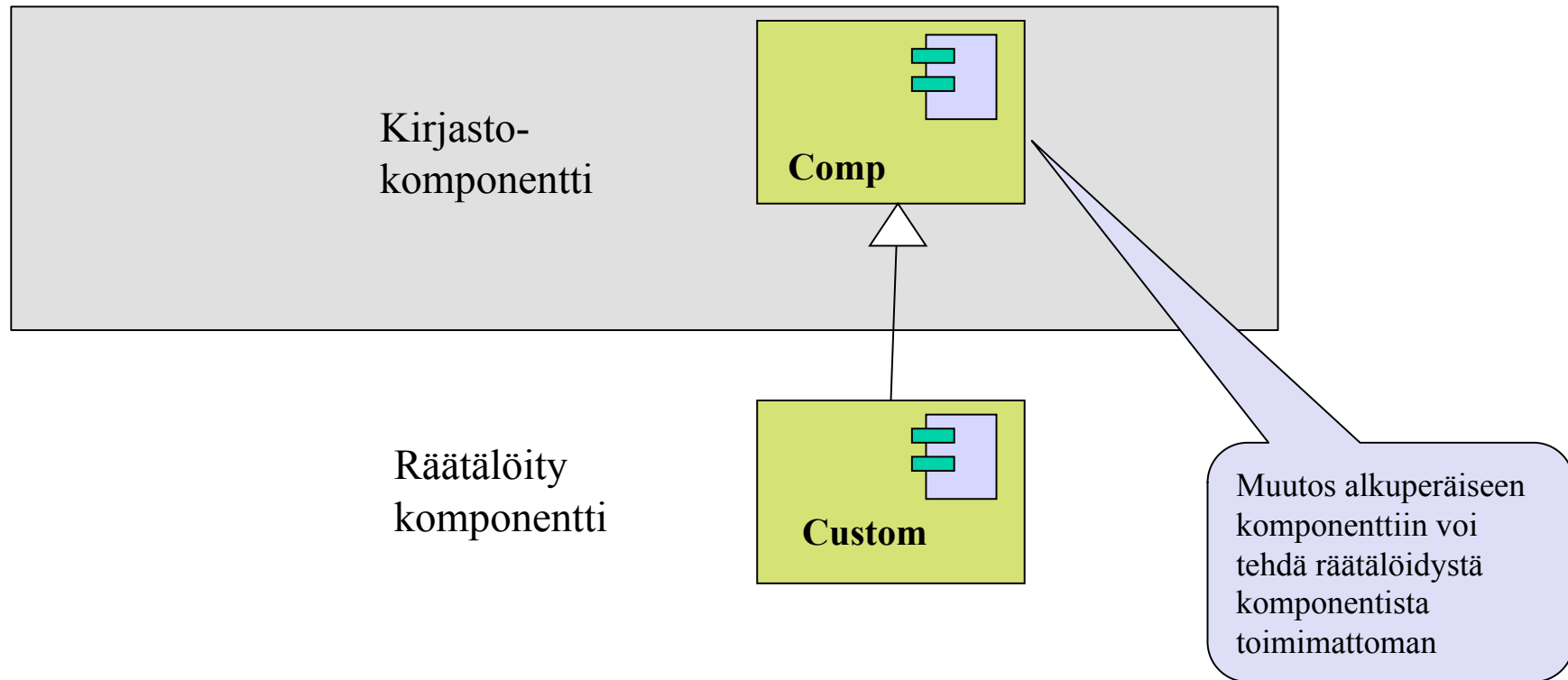


Komponentin räätälöinti

- **Komponentin uudelleenkäytettävyys edellyttää komponentin soveltumista erilaisiin yhteyksiin.**
 - **Komponentin voitava muunnella tarjoamaansa palvelua tarpeen mukaan.**
 - **Komponentin suunnittelussa huomioitava varianssin hallinta (samaan tapaan kuin tuoterungoissa tai sovelluskehysissä).**

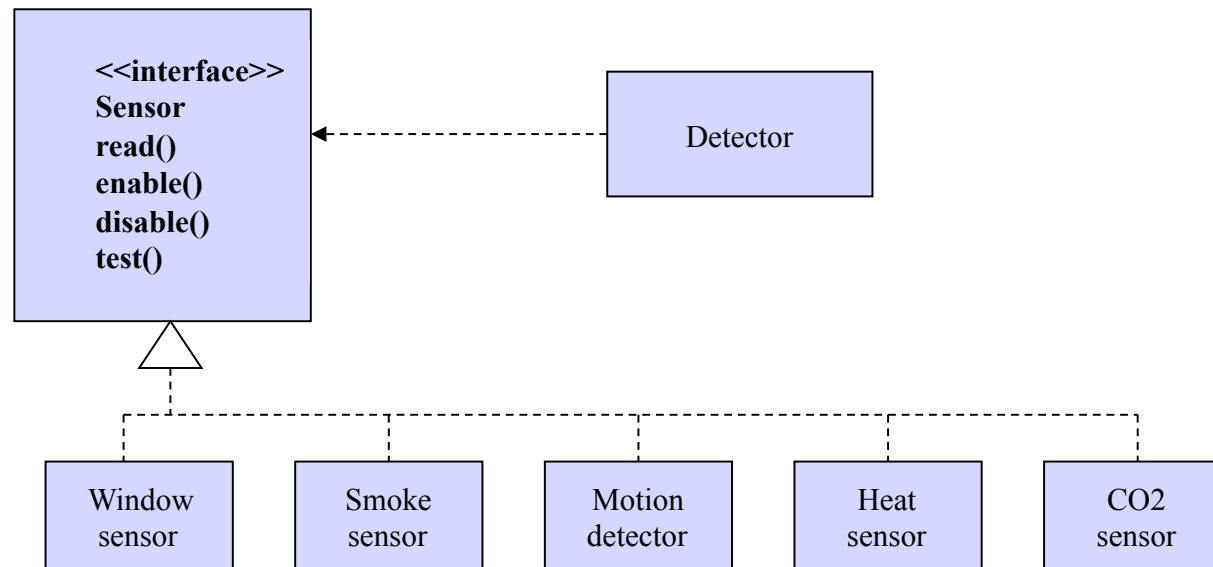
- **Komponentin räätälöinti periytymisen avulla**
 - **Määritellään halutut operaatiot aliluokassa uudelleen**
 - **dynaamisen sidonnan ansiosta suoritusajana tulee kutsutuksi olion dynaamisen tyyppin mukainen operaatio**
 - **Huono puoli: periytyminen rikkoo kapseloinnin -> semanttinen särkyvän ylliluokan ongelma (semantic fragile base class problem)**

Särkyvän yluokan ongelma



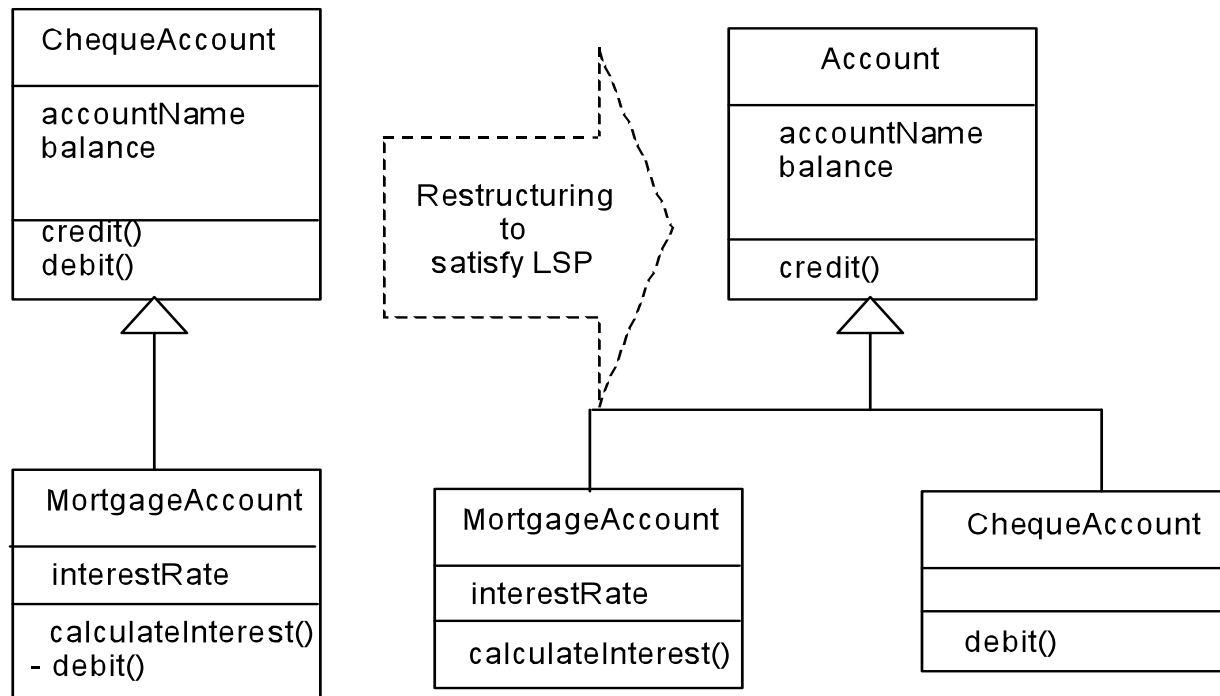
Luokkapohjaisten komponenttien suunnittelu (oliosuunnittelun periaatteet)

- **The Open-Closed Principle (OCP).** *“A module [component] should be open for extension but closed for modification.”*
 - *muutokset eivät saa aiheuttaa muutoksia useisiin paikkoihin*
 - *esim. abstrakteihin rajapintoihin perustuvissa malleissa uuden rajapinnan toteuttavan aliluokan lisääminen ei saisi aiheuttaa muutoksia mallin muissa luokissa*



- **The Liskov Substitution Principle (LSP).** *“Subclasses should be substitutable for their base classes.*
 - *Ohjelman toiminta säilyy samana, jos luokan T toiminnallisuus korvataan luokalla S, joka on T:n aliluokka*
 - *Lapsiluokan tulee olla sijoitettavissa kantaluokkiensa tilalle. Ongelma on, että tätä periaatetta ei voi varmistaa tarkastelemalla vain luokkaa itseään, vaan pitää myös tietää miten sitä käytetään.*

Vasemman puoleinen kuvaus ei noudata LSP periaatetta, koska **debit()** metodin toteutus aliluokassa ei vastaa yluokan toteutusta



Oliosuunnittelun periaatteet

- **Dependency Inversion Principle (DIP).** *“Depend on abstractions. Do not depend on concretions.”*
 - *Korkeamman tason modulit eivät saa riippua alemman tason moduleista. Molempien tulisi riippua abstraktioista. Abstraktioiden ei tulisi riippua yksityiskohdista, vaan päinvastoin.*
- **The Interface Segregation Principle (ISP).** *“Many client-specific interfaces are better than one general purpose interface.”*
 - *Lihava palvelinrajapinta tulisi erottaa erillisiksi asiakaskohtaisiksi rajapinnoiksi. Rajapinta kuuluu asiakkaalle, eikä palvelimelle.*

Source: Martin, R., “Design Principles and Design Patterns,” downloaded from <http://www.objectmentor.com>, 2000.

Oliosunnittelun periaatteet

- **The Release Reuse Equivalency Principle (REP).** *“The granule of reuse is the granule of release.”*
 - *Paketoinnissa käytetään samaa rakeisuutta kuin uudelleenkäytettävyydessä.*
- **The Common Closure Principle (CCP).** *“Classes that change together belong together.”*
 - *Luokat joihin kohdistuu samanlaisia muutospaineita kuuluvat yhteen (laitetaan samaan pakettiin).*
- **The Common Reuse Principle (CRP).** *“Classes that aren’t reused together should not be grouped together.”*
 - *Jos luokkia ei ole uudelleenkäytetty yhdessä, ei niitä myöskään pitäisi käsitellä yhtenä ryhmänä (laittaa samaan pakettiin).*

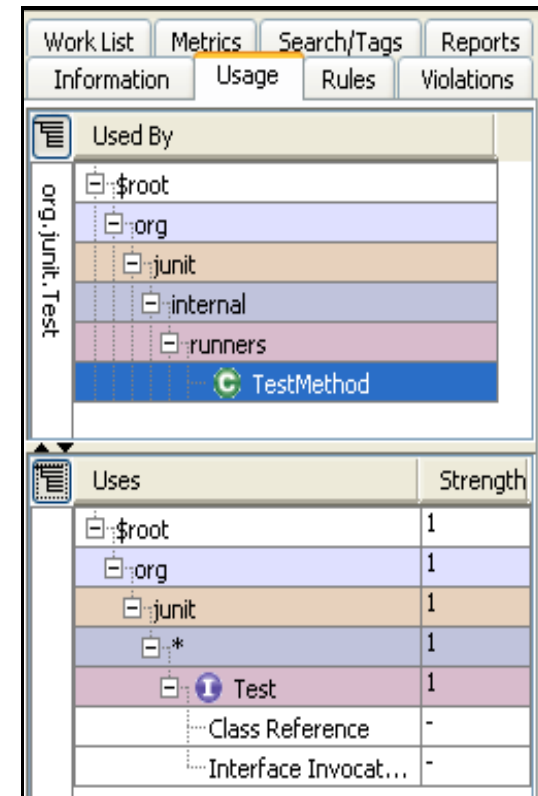
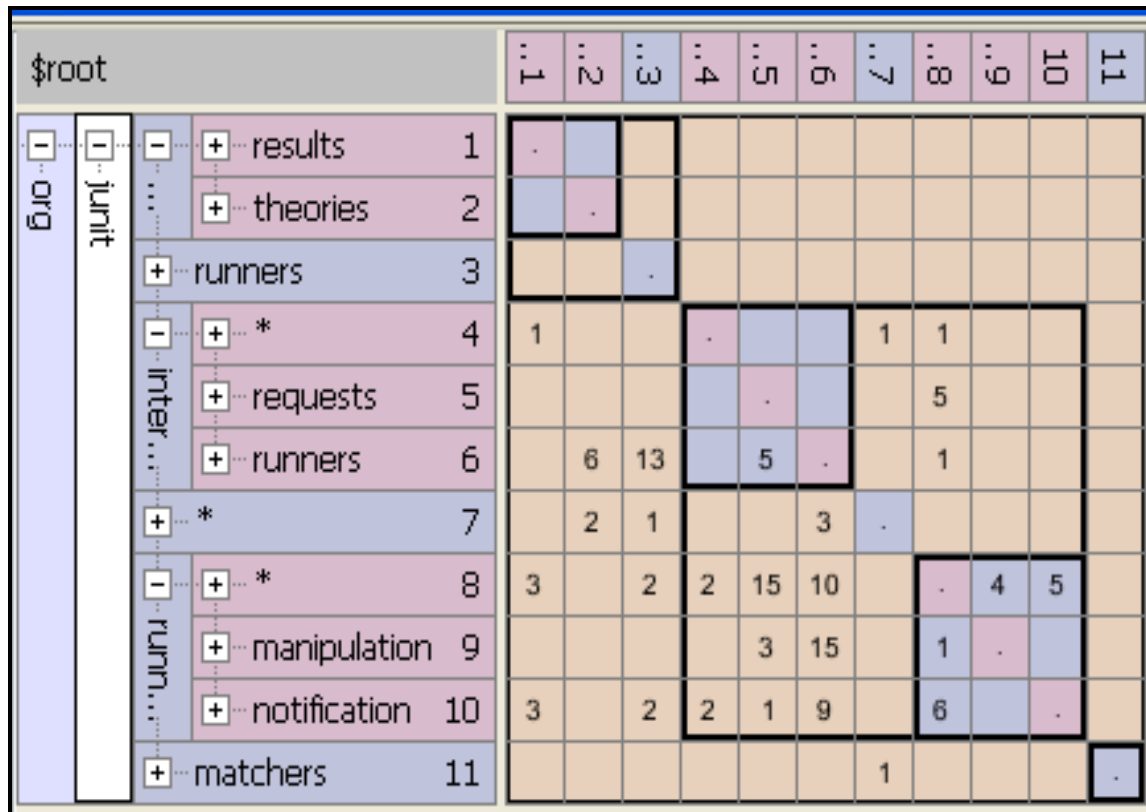
Source: Martin, R., “Design Principles and Design Patterns,” downloaded from <http://www.objectmentor.com>, 2000.

Koheesio

- **Perinteisesti: yhden tarkoituksen moduuli**
- **OO näkökulma:**
 - koheesio tarkoittaa, että komponentti tai luokka kapseloi vain sellaiset attribuutit ja operaatiot, jotka liittyvät läheisesti toisiinsa ja itse luokkaan tai komponenttiin
- **Koheesion tasot (lueteltu koheesion korkeusjärjestyksessä)**
 - **Toiminnallinen**
 - **Peräkkäinen**
 - toisen osan tuloste on toisen osa syöte
 - **Kommunkationaalinen**
 - saman datan käsittelyyn liittyvät operaatiot samassa luokassa
 - **Proseduraalinen**
 - tarkastetaan lupa käyttää tiedostoa ja sitten avataan se
 - **Ajallinen**
 - virheen sattuessa tietyt operaatiot suoritetaan (suljetaan avoimet tiedostot, kirjataan virhelokiin ja ilmoitetaan käyttäjälle)
 - **Looginen**
 - samankaltaiseen tehtävään liittyvät, esim I/O käsittelyyn
 - **Satunnainen**

Kytkentä

- **Perinteisesti:**
 - kytkenän aste komponenttien välillä
 - pyritään alhaiseen kytkentään
- **OO näkökulma:**
 - kytkenän aste luokkien välillä
- **Kytkenän tasot (korkeasta matalaan)**
 - Sisältökytkentä (haetaan toisen moduulin paikallista dataa)
 - Yleinen kytkentä (globaalin muuttujan käyttö)
 - Ulkoinen kytkentä (kaksi moduulia jakavat ulkopuolisen määrittelemän tietformaatin, protokollan tai laiterajapinnan)
 - Ohjauskytkentä (operaation A herättää operaation B)
 - Leimakytkentä (moduulit jakavat tietorakennekoosteen ja käyttävät vain osaa siitä)
 - Datakytkentä (tietoa siirretään argumenttien avulla)
 - Viestikytchentä (moduulit eivät ole riippuvia toisistaan, käyttävät julkista rajapintaa viestien vaihdolle)



Luokkien ja pakkausten välisiä riippuvuuksia voidaan tutkia esim. riippuvuutta kuvaavilla matriiseilla. Ohessa Lattix LDM (<http://www.lattix.com/products/LDM.php>)

Komponenttipohjainen kehitys

“Komponentti on itsenäinen ohjelmistoyksikkö, jota yhdessä muiden komponenttien kanssa voidaan käyttää ohjelmiston rakentamisessa”

- **Jos uudelleenkäyttö on mahdollista ohjelmistoprojektissa, seuraavia kysymyksiä tulisi kysyä:**
 - **Onko vaatimukseen sopivia valmiita (commercial off-the-shelf (COTS)) komponentteja saatavilla?**
 - **Onko vaatimukseen sopivia itse kehitettyjä komponentteja saatavilla?**
 - **Ovatko saatavilla olevien komponenttien rajapinnat sopivia järjestelmän arkkitehtuuriin?**

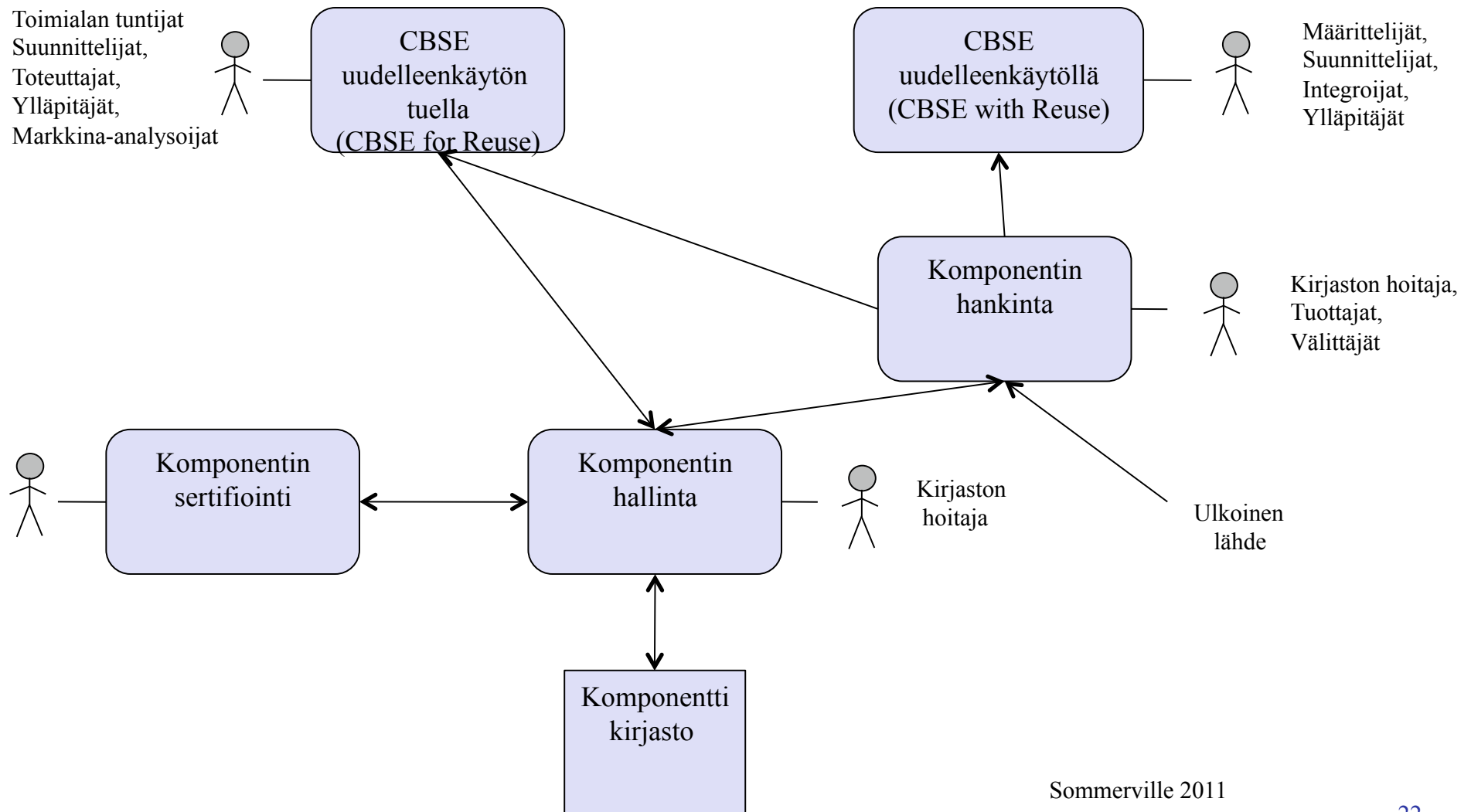
Komponentin ominaisuudet

- **Yhdistettävä**
 - rajapinnat noudattavat julkista rajapintamäärittelyä
- **Itsenäinen**
 - toimitetaan itsenäisenä, ilman tarvetta käyttää muita komponentteja
- **Toimitettava**
 - pystyy toimimaan yksin alustalla, joka on komponenttimallin mukainen
- **Dokumentoitu**
 - riittävä kuvaus, jotta käyttäjät pystyvät päättelemään, sopiiko komponentti heidän tarpeeseen
- **Standardoitu**
 - noudattaa jotain standardoitua komponenttimallia
- **Riippuvuudet**
 - komponentti voi riippua palveluista, joita se olettaa saavansa muilta komponenteilta vaadittujen rajapintojen kautta
- **Käyttöönotto**
 - komponentti voidaan ottaa käyttöön lähdekoodisena (käännösaikana) tai käännettynä binäärimuodossa (linkkausaikana)
- **Koko**
 - komponentin tulisi olla yhden henkilön hallittavissa
- **Standardointi**
 - esimerkkinä Sunin EJB-komponenttistandardi

» *Koskimies, Mikkonen 2005*

» *Sommerville 2004*

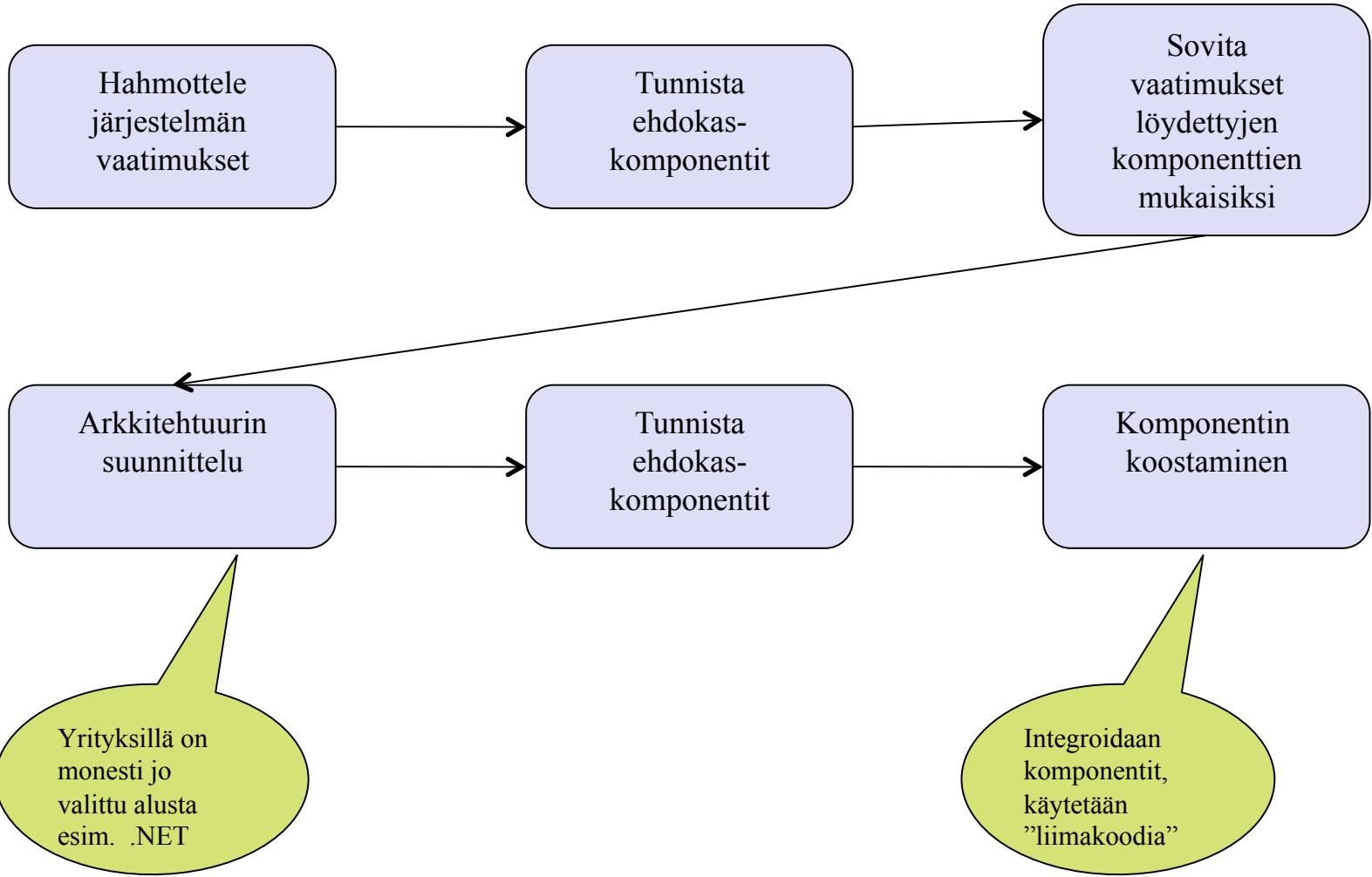
CBSE (component based software engineering) prosessit



CBSE
uudelleenkäytön
tuella
(CBSE for Reuse)

- **Miten uudelleenkäyttöä voidaan edistää**
 - Poistetaan sovelluskohtaiset metodit (operaatiot)
 - Muutetaan nimet helpommin ymmärrettäviksi
 - Lisätään metodeja, jotta saadaan toiminnot kattavammiksi
 - Tehdään poikkeusten käsittely yhdenmukaiseksi kaikille metodeille
 - Lisätään ”konfigurointi” rajapinta, jotta komponentti saadaan sovitettua helpommin käyttöön eri tilanteissa
- **Uudelleenkäytön ja käytettävyyden ristiriita**
 - Komponentin uudelleenkäytön vahvistaminen tekee komponentista mutkikkaamman ja vaikeuttaa sen ymmärtämistä
- **Mistä komponentteja**
 - Vanhoista (legacy) järjestelmistä, lisätään kääre (wrapper), jonka avulla komponentin saa helpommin käyttöön

CBSE
uudelleenkäytöllä
(CBSE with Reuse)



Tunnista
ehdokas-
komponentit

Komponentin
etsintä

Komponentin
valinta

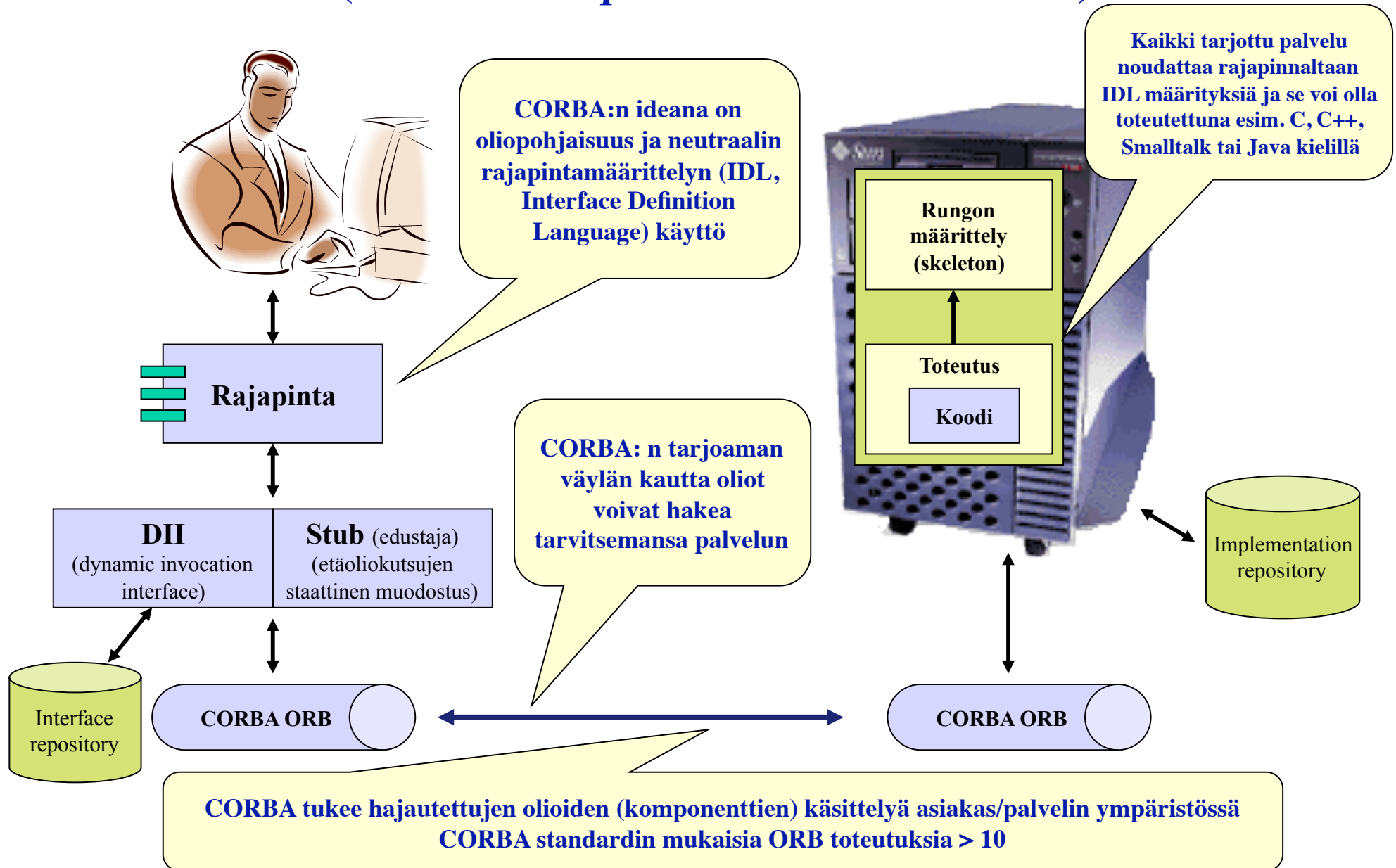
Komponentin
validointi

Luotettavilta
toimittajilta,
omista varastoista,
koodikirjastoista
(Sourceforce,
Google Code)

Voidaan tarvita useampia
komponentteja/vaatus,
mikä komponenttijoukko
parhaiten kattaa
vaatimukset

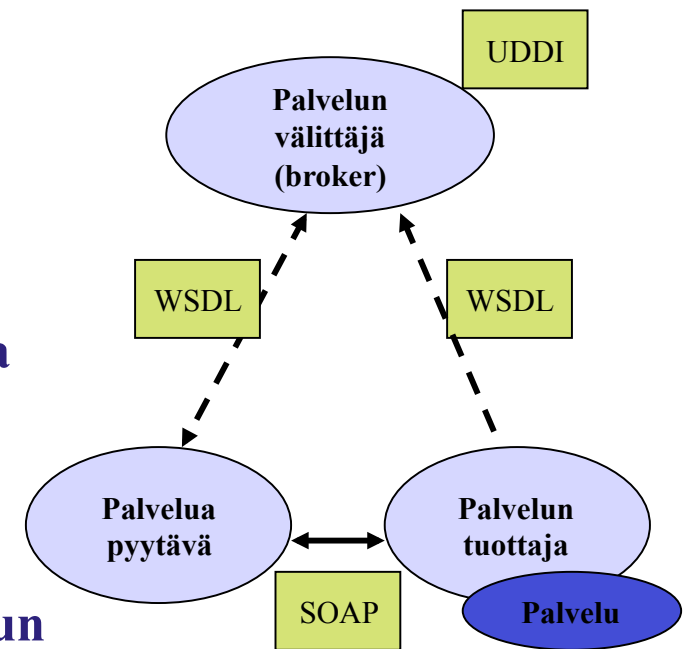
Testaus voi olla
haastavaa, jos
komponentin
määrittely ei ole
riittävän tarkka

CORBA (Common Request Broker Architecture)



Palvelupohjainen arkkitehtuuri (SOA)

- löysästi kytkeytynyt arkkitehtuuri
- kolme protokollaa (XML pohjaisia)
 - SOAP (Simple Object Access Protocol)
 - kieli ja protokolla, jonka avulla rakenteellista tietoa siirretään
 - WSDL (Web Services Description Language)
 - kieli, jonka avulla SOAP protokollalla tarjotun palvelun rajapinta (pyyntö- ja vastausmuoto) voidaan määritellä
 - UDDI (Universal Description, Discovery and Integration)
 - standardoitu tapa julkaista ja etsiä eri palvelujen metadataa

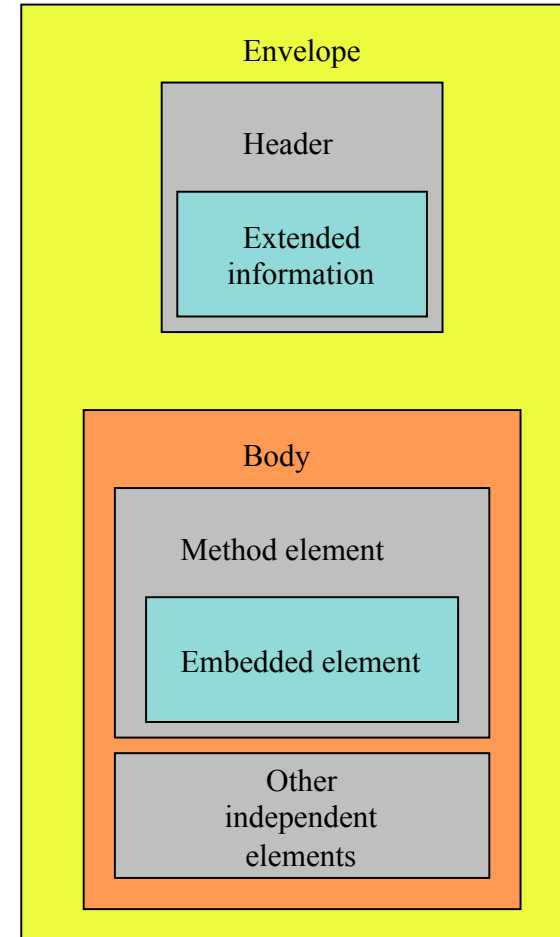


Web palvelujen arkkitehtuuri

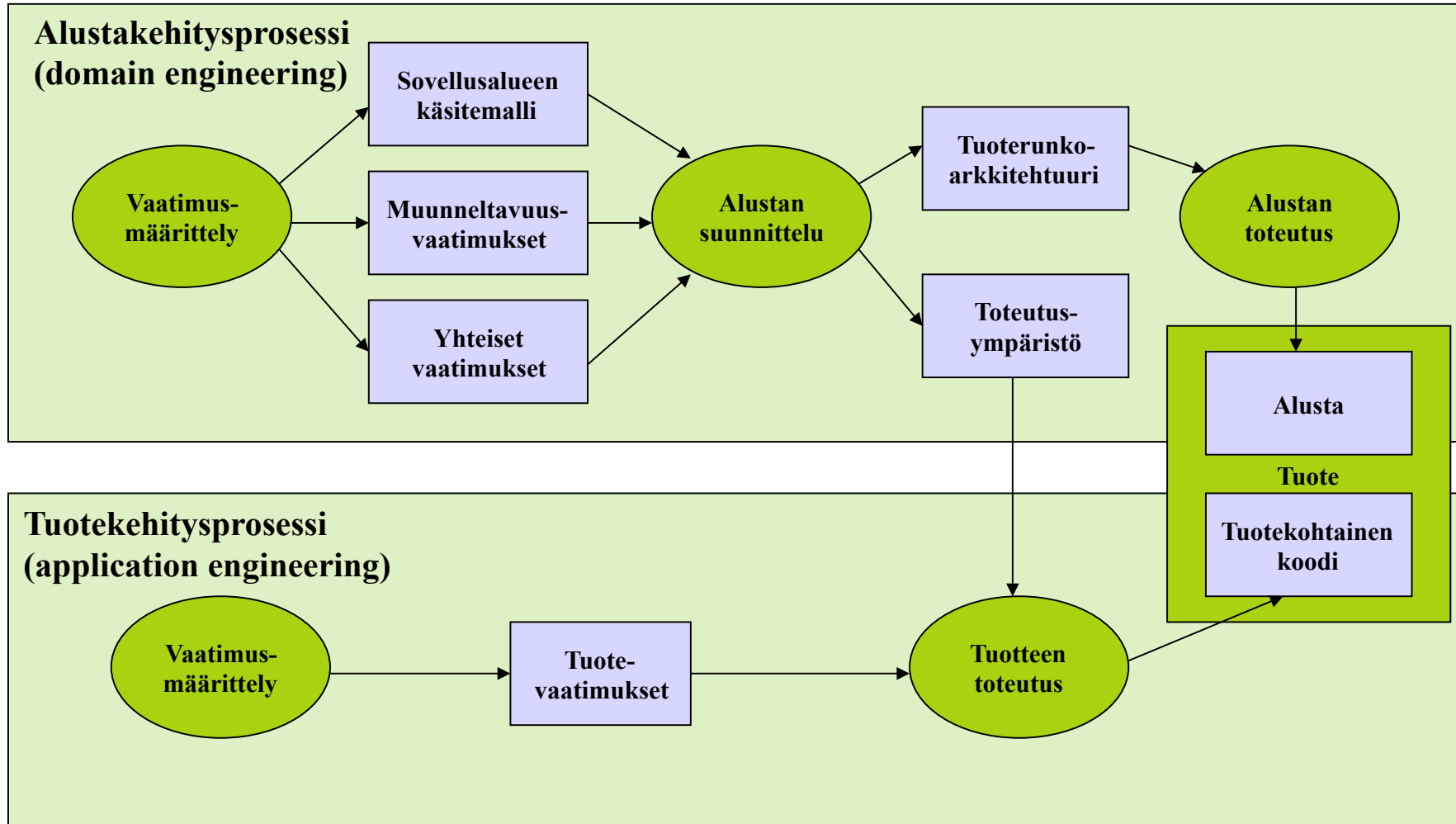
SOAP

- SOAP on kieliriippumaton
- Käyttää XML (Extensible Markup Language) kieltä kuvauskielenä rakennettaessa viestipakettia, joka tyypillisesti siirretään verkon yli HTTP protokollaa käyttäen.
- SOAP viestin peruosana on kantaelementti `<SOAP:Envelope>` joka sisältää vaihtoehtoisen otsikkoelementin `<SOAP:Header>`
 - Oikeuksien tarkistamista
 - Tapahtumien hallintatietoa
- ja pakollisen runkoelementin `<SOAP:Body>`
`<SOAP-ENV:Body>`

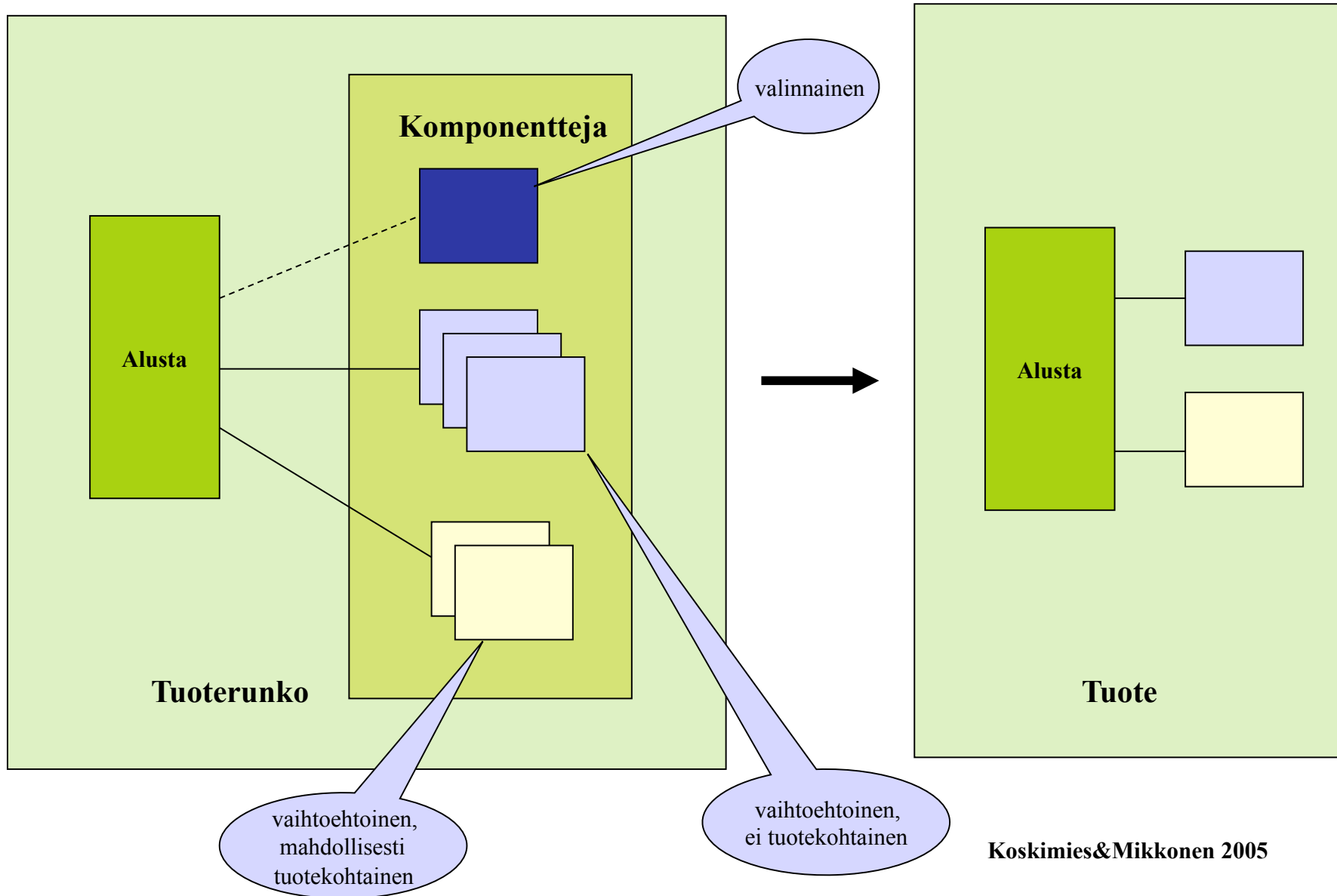
```
<mFindPart xmlns:m"the-method-uri">  
  <PartNo>12345 </PartNo>  
  <GroupID>7 </GroupID>  
</mFindPart>  
</SOAP-ENV:Body>
```



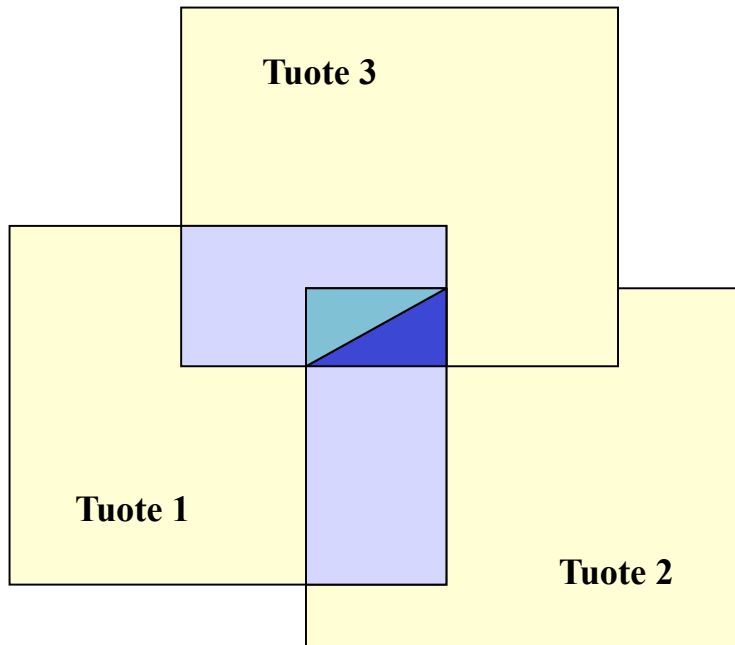
Tuoterunkoon perustuva ohjelmistokehitysprosessi



Tuoterunko ja yksittäinen tuote



Yhdistävät ja erottavat piirteet



Yhteiset piirteet

Toiminnallisuus, joka on kaikissa tuotteissa samassa muodossa

Eroavat piirteet

Sama käsitteellinen toiminnallisuus, jossa on pieniä tuotekohtaisia eroja

Toiminnallisuus esiintyy kahdessa tai useammassa tuotteessa (mutta ei kaikissa)

Tuotekohtainen toiminnallisuus

11. Käyttöliittymän suunnittelu

Helppo oppia?

Helppo käyttää?

Helppo ymmärtää?



Käyttöliittymän suunnittelu

Tyypillisiä suunnitteluvirheitä

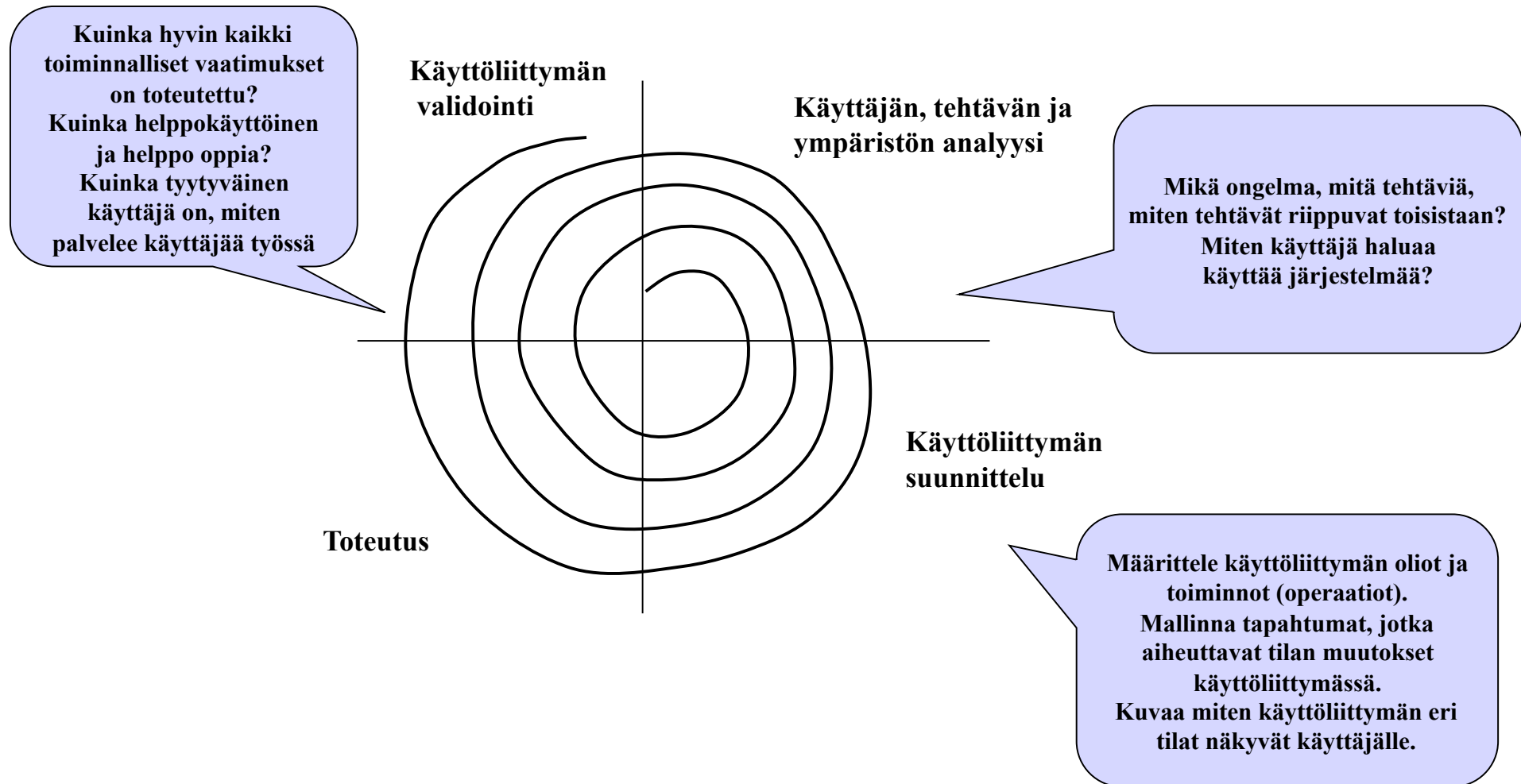
yhdenmukaisuuden puute
liikaa muistamista
ei ohjausta/apua
ei poistumistietä
outo kieli (tietokonetermit)
ei oikopolkuja
ei riittävästi palautetta
vaikea dialogi
vaikeat virheilmoitukset



Kultaiset säännöt / Mandel 1997

- **Käyttäjä ohjaa (place the user in control)**
 - joustavat vuorovaikutustavat
 - käyttäjä voi keskeyttää halutessaan
 - ei teknisiä järjestelmätason ilmoituksia
- **Vähennä muistikuormaa**
 - ei tarvetta muistaa komentoja, vaihtoehdot esille
 - loogiset oikopolkukomennot
- **Pidä käyttöliittymät yhdenmukaisina**
 - samat sanat, tilanteet tai toimenpiteet tarkoittavat aina samaa asiaa
 - useissa ikkunoissa esiintyvät osaset pitäisi sijoitella kaikissa ikkunoissa samoihin paikkoihin

Käyttöliittymän suunnitteluprosessi



Käyttöliittymän suunnittelun periaatteet / Tognozzi 2001

- **Ennakointi**
 - sovelluksen tulisi ennakoida käyttäjän tarpeita
 - esim. tulostimen valmistajan sivulla on “driver” infon lisäksi linkki, mistä eri käyttöjärjestelmien “driver” voidaan ladata
- **Kommunikointi**
 - käyttäjää tulisi informoida aktivoituista toiminnoista
- **Yhdenmukaisuus**
 - toimintojen ja esitystavan tulisi olla yhtenäinen läpi sovelluksen
- **Kontrolloitu itsenäisyys**
 - käyttäjän navigointia sovelluksessa tulisi tukea, mutta tarvittaessa voidaan myös rajata käyttöä käyttäjätunnuksen ja salasanan avulla
- **Tehokkuus**
 - sovelluksen käyttöliittymä tulisi suunnitella sellaiseksi, että tavoitteena on käyttäjän tehokkuuden paraneminen, ei suunnittelijan

Käyttöliittymän suunnittelun periaatteet / Tognozzi 2001

- **Joustavuus**
 - käyttöliittymän tulisi tukea sekä sovellusta tuntevia, että satunnaisia käyttäjiä (“eiku” kokeilua)
- **Fokus**
 - käyttöliittymän tulisi “estää” käyttäjää eksymästä
- **Fittin laki**
 - kohteen löytämiseen kuluva aika on etäisyyden ja kohteen koon funktio /Fitt 1954
- **Käyttöliittymäoliot**
 - suunnaton määrä käyttöliittymäolioita löytyy kirjastoista, käytä niitä
- **Odotusajan minimointi**
 - sovelluksen tulisi käyttää “multitasking” ominaisuutta ja suorittaa raskaita tehtäviä taustalla

Käyttöliittymän suunnittelun periaatteet / Tognozzi 2001

- **Opittavuus**
 - sovelluksen oppimiseen kuluva aika pitäisi minimoida
- **Metaforat**
 - tuttujen periaatteiden (ja nimien) käyttö helpottaa oppimista
- **Varmista tulosten säilyvyys**
 - varmista täytettävän lomakkeen talletus
- **Luettavuus**
 - esitettävän tiedon tulisi olla helppolukuista
- **Tilatiedon jäljitys**
 - tilatietoa tulisi tallettaa, jotta siihen voidaan palata
- **Näkyvä (visible) navigointi**
 - “the illusion that users are in the same place, with the work brought to them” (käyttäjä valitsee toiminnot käyttöliittymässä esitellyistä)

Soveltuvat lait ja *pohdiskelun aiheita*

1. **COTS perustainen ohjelmistokehitys ei poista kehitysprosessin riskejä / hyp_no 7, Basili - Boehm 2001**
 - **COTS (Commercial Off-The-Shelf) perustaista ohjelmistokehitystä on tehty aina. Nykyisinkin lähes kaikissa järjestelmissä on osa koottu valmiista komponenteista/ pakkauksista.**
 - **COTS pakkausten käyttö lyhentää ohjelmistokehitysaikaa ja pienentää kehitysprosessin riskejä, mutta hypoteesin tueksi ei ole vielä riittävästi tieteellistä näyttöä.**

Soveltuvat lait ja *pohdiskelun* aiheita

- *Mitä tarkoitetaan ohjelmistokomponentilla? Anna muutamia esimerkkejä.*
- *tai*
- *Mitkä ovat CBSE:n peruskäsitteet ja periaatteet Crnkovic et al. mukaan?*
- *Li et al. esittävät 10 faktaa, jotka liittyvät OTS (off-the-self) perustaisen kehittämisen teollisiin käytänteisiin. Esitele niistä viisi tarkemmin, eli esitele myös perustelut näille viidelle faktalle.*
- *Mitä tarkoittavat pilvilaskenta-arkkitehtuurin kolme tasoa; SaaS, PaaS ja IaaS (Yau & An mukaan)?*
- *Miten ohjelmistokehitys tapahtuu palvelupohjaisessa ohjelmistotekniikassa (SOSE) (Yau & An mukaan)?*
- *Mitä haasteita Yau & An ovat havainneet ohjelmistokehityksessä, kun käytetään palvelupohjaista ohjelmistotekniikkaa (SOSE)?*

Soveltuvat lait ja *pohdiskelun* aiheita

- *Aiheeseen liittyvää luettavaa:*
- *Crnkovic I., Stafford J. and Szyperski C., Software Components beyond Programming: From Routines to Services, IEEE Software, vol 28, no 3, 2011, pp. 22-26*
- *Li J., Conradi R., and Slyngstad O., Bunse C., Torchiano M. and Morisio M., Development with Off-the-Shelf Components: 10 Facts, IEEE Software, vol 26, no 2, 2009, pp. 80-87*
- *Yau S.S., An H.G., Software Engineering Meets Services and Cloud Computing, Computer, vol 44, no 10, 2011, pp. 47-53*

Harjoitustehtävät: Viikko 5

- 1. Kirjoita toiminnalliset (järjestelmä)vaatimukset, joissa tarkennetaan käyttäjävaatimuksia ja otetaan huomioon toteutusympäristön aiheuttamat rajoitukset (jokaisesta vaatimuksesta yksilöllinen tunniste, kuvaus, rajoitukset ja liitântä käyttäjävaatimukseen). Voit käyttää samaa dokumenttipohjaa, kuin käyttäjävaatimuksissa.**
- 2. Kirjoita ei-toiminnalliset (laatu)vaatimukset, joissa kiinnitetään tuotteen käytettävyyteen, tehokkuus, tilan tarpeeseen, luotettavuuteen, siirrettävyyteen ja turvallisuuteen liittyvät tavoitteet (jokaisesta vaatimuksesta yksilöllinen tunniste, liitântä laatutekijään ja kuvaus). Voit käyttää samaa dokumenttipohjaa, kuin käyttäjävaatimuksissa.**
- 3. Tee käyttäjävaatimusten ja toiminnallisten vaatimusten jäljitettävyydsmatriisi.**

Harjoitustehtävät: Viikko 5

4. Tee jokin tehtävistä a – e.

Sinun pitäisi rakentaa jokin seuraavista järjestelmistä

- a. Verkkopohjainen kurssille ilmoittautumisjärjestelmä yliopiston tarpeisiin.
- b. Web-perustainen tilausten käsittelyjärjestelmä tietokonekaupalle.
- c. Yksinkertainen laskutusjärjestelmä pienelle yritykselle.
- d. Internet-pohjainen keittokirja, joka on rakennettu osaksi sähköhella.

Valitse näistä yksi ja kuvaa se luokkakaaviolla. Kaavoissa tulisi näkyä tieto-oliot, yhteydet ja tärkeimmät attribuutit.

e. Tee luokkakaavio, jossa on vähintään 10 suhdetta (assosiaatio, kooste, yleistys/erikoistus) seuraavien luokkien kesken. Esitä kertautuminen assosiaatioissa ja nimi, jos assosiaatio ei ole tarkoitus ei ole ilmeinen. Voit tarvittaessa lisätä muita luokkia. Luokat: koulu, pelikenttä, rehtori, johtokunta, luokkahuone, kirja, oppilas, opettaja, kahvila, tietokone, pöytä, tuoli, ovi, luokanvalvoja