Ohjelmistotestauksen perusteita I

Luento 1

Antti-Pekka Tuovinen

Luennon oppimistavoitteet

- Mitä ohjelmistojen testaaminen on?
- Mitä tarkoittavat virhetoiminto, vika ja erehdys?
- Mikä on testauksen suhde ohjelmiston laatuun?
- Mitkä ovat testattavuuden rajat?

Testi

”***testi*** jnk. kunnon, kelvollisuuden, soveltuvuuden tms. selville saamiseksi suoritettu koe;”

”***testata*** tutkia, koetella testejä käyttäen.”

Nykysuomen sanakirja, WSOY, 1990

”test*:*

1. *The means by which the presence, quality, or genuiness of anything is determined.*
2. *The trial of the quality of something.”*

Webster’s Encyclopedic Unabridged Dictionary of the English Language

Ohjelmistot

- Ohjelmisto on *ei-materiaalinen tuote*, jonka ominaisuudet eivät ole fysikaalisten mittausten kautta määritettävissä
  - Staattinen ja dynaaminen muoto
- Ohjelma*koodin* automaattisella (algoritmisella) analysoinnilla ja ominaisuuksien päättelyllä on teoreettiset ja käytännölliset rajoitteensa
- Ohjelma on siis *suoritettava*, jotta sen ominaisuuksia ja todellista ajonaikaista käyttäytymistä voidaan tutkia ja kokeilla

Ohjelmiston testaaminen on

- *Ohjelman suorittamista tietokoneella ja sen käyttäytymisen vertaamista ohjelman vaatimuksiin*

IEEE:n määritelmiä

“testing[2] : *An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component.”*

[2] IEEE Standard for *Software and System Test Documentation*. IEEE Std 829-2008, http://ieeexplore.ieee.org/servlet/opac?punumber=4578271

IEEE:n määritelmiä

“testing process[2]: *provides objective evidence that the software-based system and its associated products:*

a) *Satisfy the allocated system requirements*

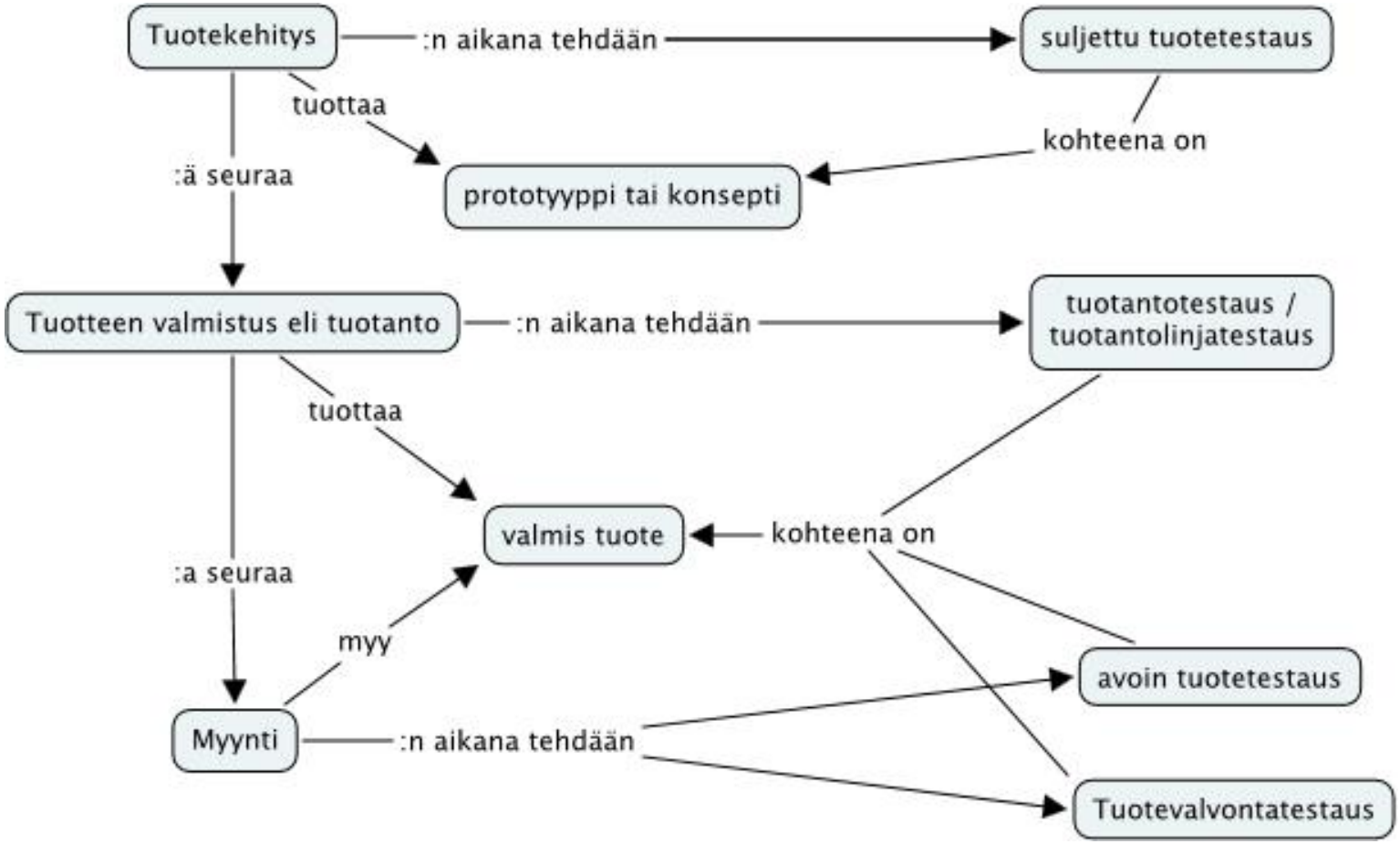
b) *Solve the right problem (e.g., correctly model physical laws, implement business rules, and use the proper system assumptions)*

c) *Satisfy its intended use and user needs”*

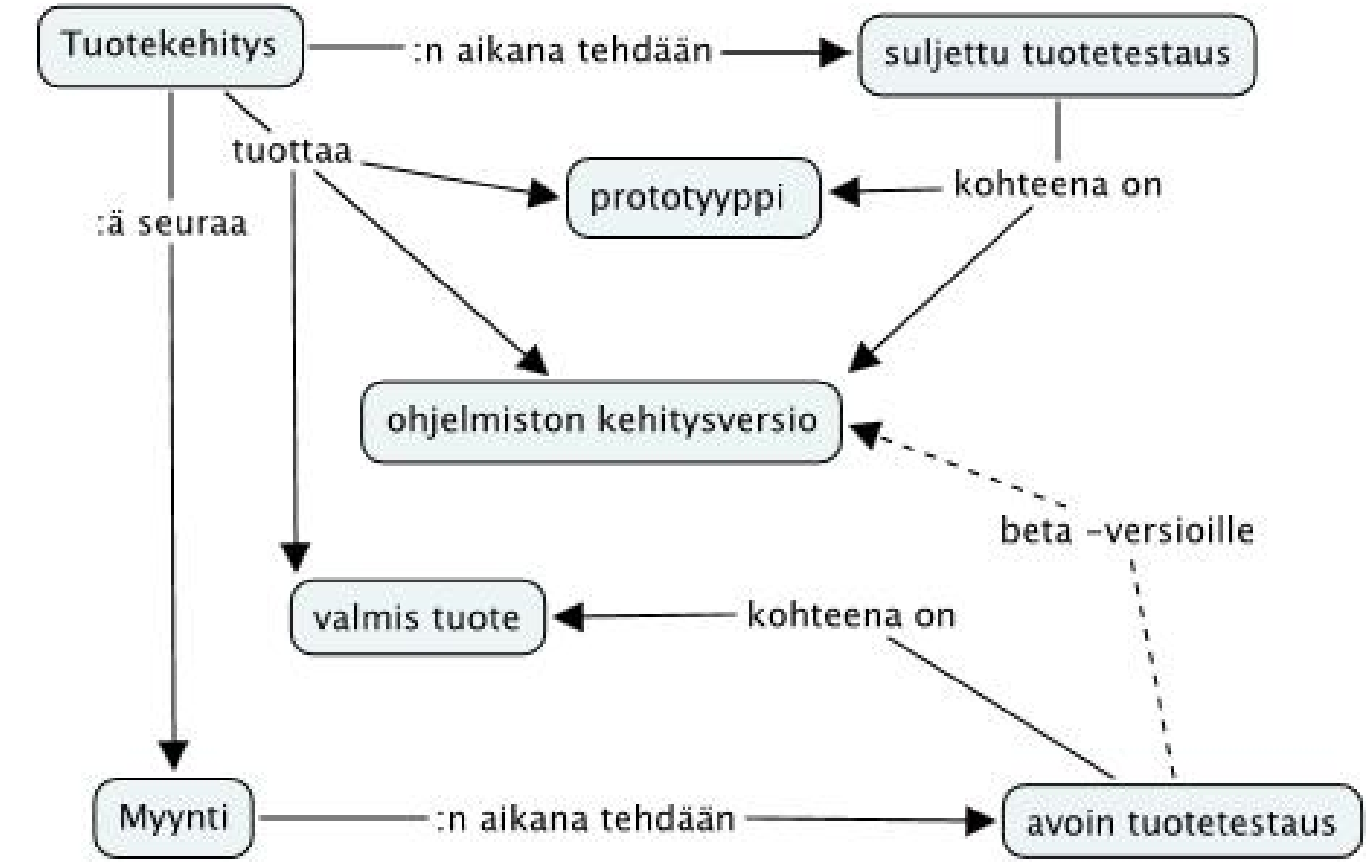
Ohjelmistojen testauksen erityispiirteitä

- Tarkastellaan vertailun vuoksi fyysisten eli *materiaalisten* tuotteiden ja toisaalta digitaalisten, *ei-materiaalisten* ohjelmistojen tuotekehitystä ja valmistusta

Materiaalisen tuotteen kehitys ja (massa-) tuotanto



Ohjelmistotuotteen kehittäminen



Huomioita

- Ohjelmiston ”valmistus” (engl. manufacturing, mass production) on pelkkää mallikappaleen digitaalista kopiointia, jossa virheiden mahdollisuus on käytännössä hyvin pieni
  - Valmistusprosesseista ja –materiaaleista johtuvat viat jäävät pois
- Tuotevalvontaa ei tehdä
  - [Ehkä olisi syytä – esimerkiksi tietoturvan vuoksi?]
  - Applikaatiokauppapaikkojen tekemät testit (Google Play, App Store)

Huomioita

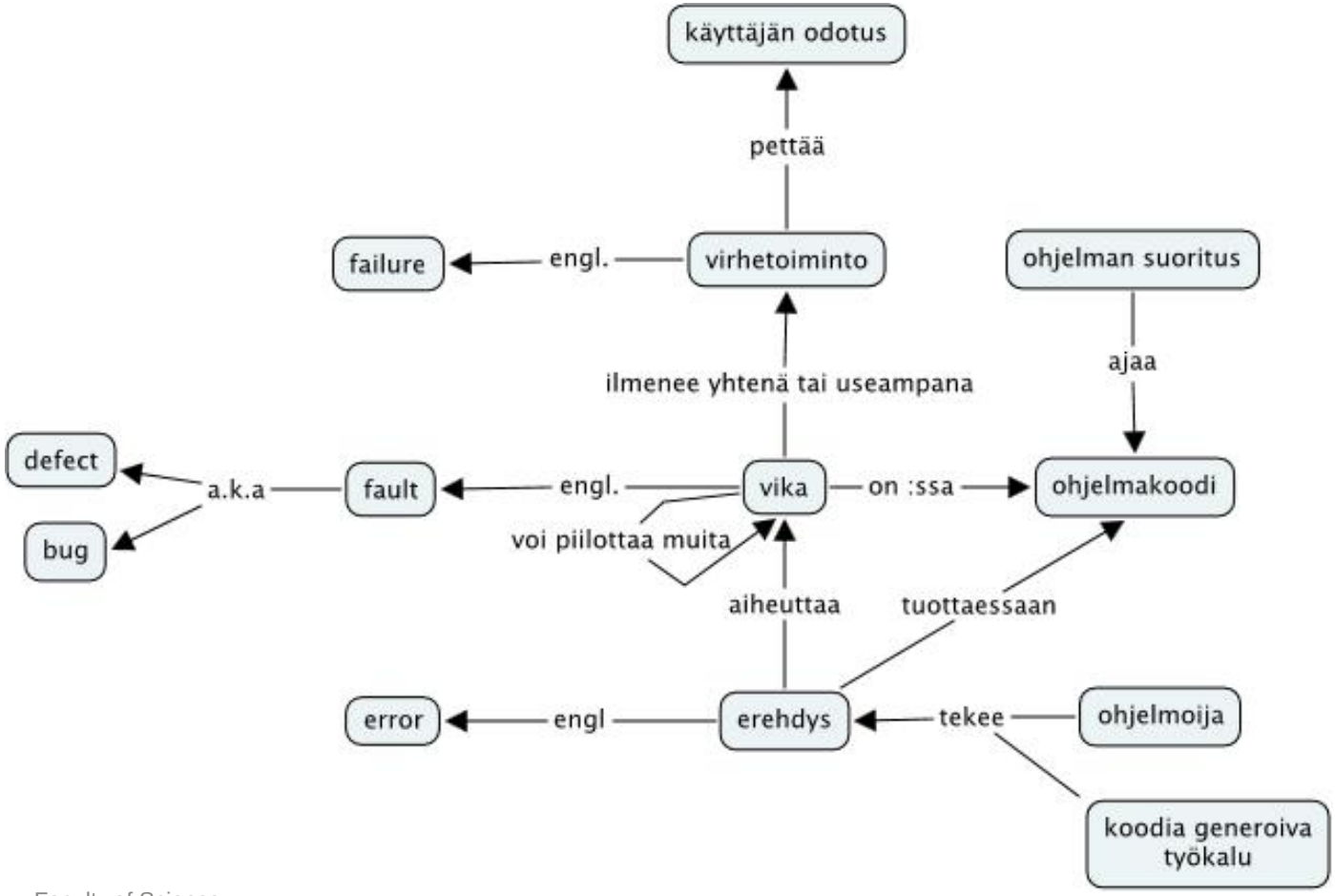
- **Kaikki ohjelmiston viat syntyvät tuotekehityksen aikana**

- …tosin myös ohjelmiston käyttö-ympäristössä tapahtuvat muutokset voivat aiheuttaa ohjelmiston ”vikaantumisen” sen käyttöaikana (toiminnallisten riippuvuuksien kautta)

Huomioita

- Ohjelmiston kehitysversioiden testaaminen voi olla vaikeaa, koska tuote on vain osittain implementoitu
  - Ohjelmiston kehitys- ja kokoonpanojärjestys määrää milloin mitäkin toimintoja päästään testaamaan
- Ohjelmiston arkkitehtuuri voi tukea komponenttien ja toimintojen osittaista testausta tai jopa estää sen
- Yksittäin testatuista osista ei välttämättä synny toimivaa kokonaisuutta

Mistä viat tulevat?



Entä vikojen korjaaminen eli debuggaus?

- Testaus *ei ole sama asia* kuin debuggaus
- Debuggaus tarkoittaa vikojen *jäljitystä* ja *korjaamista* ohjelmakoodissa
- Testaajan tehtävä on *löytää virheitä* ohjelman toiminnassa ja raportoida virheet sekä testiolosuhteet niin selkeästi, että varsinaisia vikoja voidaan alkaa etsiä ja korjata
- Usein kehittäjä itse sekä testaa että korjaa

Ohjelmistojen testaamisella on monia tavoitteita (goal)

- Ohjelman suorittaminen *virheiden löytämiseksi*
- Ohjelman suorittaminen *laadun mittaamiseksi*
- Ohjelman suorittaminen *luottamuksen lisäämiseksi*
- Ohjelman tai sen dokumentaation analysointi *vikojen ennaltaehkäisemiseksi*
  - Myös ohjelmiston rakenteiden ja koodin staattinen analyysi voi olla osa testausta
  - Ohjelman ”suoritus” tapahtuu analysoijan mielessä

Ohjelmistojen laatu ja testaus

- Ohjelmiston *laatu* on monisyinen asia
  - Sisäinen laatu
  - Ulkoinen laatu
  - Käytössä ilmenevä laatu
- *Laatumalli* määrittelee joukon laatupiirteitä (characteristics, quality attributes) joiden suhteen laatua mitataan
  - ISO 9126 laatumalli

Ohjelmistojen laatu ja testaus

- Tietyn ohjelmiston (vaadittu) laatu määritellään
  - Valitsemalla joukko ohjelman tarkoituksen ja käytön kannalta aiheellisia *laatupiirteitä*
  - Antamalla *mitta-asteikot* ja *tavoitearvot* laatupiirteille
  - Määrittelemällä miten laatupiirteiden arvojen *mittaus* tehdään

Ohjelmistojen laatu ja testaus

- Virheiden määrällä on suora yhteys ohjelmiston laatuun
- Käyttäjän kokema laatu on huono, jos ohjelma
  - Ei tee kaikkea mitä sen pitäisi
  - Toimii liian hitaasti
  - On liian vaikea käyttää
- Virheiden löytämiseksi tehtävä testaus onkin oleellinen osa ohjelmiston kokonaistavaltaista *laadunvarmistusta* (Quality Assurance)

Ohjelmistojen laatu ja testaus

- Ohjelmiston laatutavoitteiden edellä kuvattu määritteleminen mahdollistaa myös laadun suoran testaamisen (non-functional testing)
  - Jos testissä saavutetaan testattavalle piirteelle annettu tavoitearvo, tulos on ”pass” muuten ”fail”
  - Vakioidut testiaineistot (test data), testiolosuhteet (test conditions) ja testin suoritustapa (test execution) tekevät mittaustuloksista ajallisesti vertailukelpoisia
- Esimerkki – Smartphone UI performance testing (case Nokia)

Testattavuuden rajat – formaalit menetelmät

- Ohjelmien virheettömyyden täydellinen automaattinen (algoritminen) varmistaminen eli verifiointi on teoreettisestikin mahdotonta
  - Kts. ”halting problem” (pysähtymisongelma, tuttu Laskennan mallit -kurssilta)
- Voidaan tehdä joidenkin ominaisuuksien suhteen
  - Vaatii yleensä ohjelman formaalin spesifikaation erikoiskielellä
  - Soveltuu työläytensä vuoksi vain erikoistapauksiin
- Lopullinen implementaatio pitää joka tapauksessa testata suorittamalla
  - Tosin kevyemmin kuin ilman formaalia verifiointia

Testattavuuden rajat – testaus suorittamalla

- Entä sitten ohjelman testaaminen suorittamalla – voidaanko siinä päästä täydelliseen kattavuuteen?
  - Halutaan testata ohjelman/ohjelman osan toiminta *kaikilla* teknisesti mahdollisilla (myös odottamattomilla) *syötteillä* (black-box testing)
  - Halutaan testata *kaikki* mahdolliset *eri suorituspolut* ohjelmarakenteen läpi (white-box testing)
- Täydellinen testaus löytäisi kaikki virheet

Black-box -esimerkki

Testattava metodi:

```
public class BMICalculator {

    public static double computeBMI(
            Integer  body_weight_in_kg,
            Integer  height_in_cm)  {
    // formula:
    //    1.3 * body_weight / pow( height, 2.5 )
       ...
    }
}
```

Black-box -esimerkki

Silmukka kaikkien syötearvoparien läpikäymiseksi:

```
for (int w=Integer.MIN_VALUE;w<Integer.MAX_VALUE;w++) {
   for (int h=Integer.MIN_VALUE;h<Integer.MAX_VALUE;h++) {
      BMICalculator.computeBMI(w,h);
   }
}
```

(Huom: tämä *ei* ole testi – miksi?)

Black-box -esimerkki

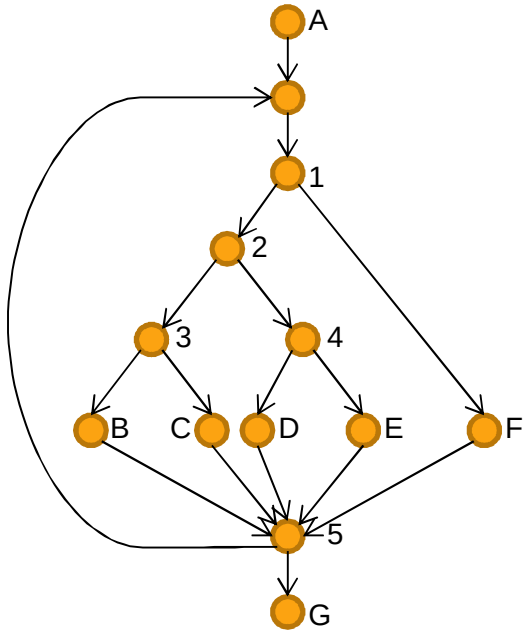
Silmukan suoritusaika tehokkaalla PC:llä

~ 122 000 vuotta

Käytännössä eri testitapauksiakin voi olla ääretön määrä (esim. käyttäjän antamat erilaiset syötteet interaktiivisessa sovelluksessa)

White-box -esimerkki

```
<stmt A>
Do {
  if (<cond 1>) {
    if (<cond 2>) {
       if (<cond 3>){
         <stmt B>
       } else {
         <stmt C>
       }
    } else {
       if (<cond 4>) {
         <stmt D>
       } else {
         <stmt E>
       }
    }
  } else {
    <stmt F>
  }
} While (<cond 5>)
<stmt G>
```



White-box -esimerkki

- Olkoon niin, että
  - Do-While -silmukka suoritetaan vähintään kerran mutta enintään 20 kertaa
  - Silmukan suoritukset ovat toisistaan riippumattomia
    - ➢ Silmukan edellinen suoritus ei vaikuta siihen, mikä kontrollivuoverkon haara käydään läpi seuraavalla kerralla

White-box -esimerkki

- Halutaan käydä läpi jokainen erilainen silmukan suorituksen sisältävä suorituspolku (= eri pituinen ja eri suoritushaarat sisältävä polku)
  - Suorituskertojen yhteenlaskettu lukumäärä on:

$$5 + 5^2 + \ldots + 5^{19} + 5^{20} \sim 10^{14}$$

- Jos testaus voidaan automatisoida niin, että yhden polun suoritukseen kuluva aika on 5 µs, koko testin suoritukseen kuluu n. **19 vuotta**

Testattavuuden rajat

- Täydellinen (exhaustive) eli kaikki mahdolliset (järkevät ja järjettömät) syötearvot ja suorituspolut kattava testaaminen ei ole siis yleensä mahdollista
  - Käytännössä voidaan testata vain *murto-osa kaikista mahdollisuuksista*
- Hyvin valituilla testitapauksilla voidaan kuitenkin paljastaa merkittävä määrä vikoja ja saavuttaa tyydyttävä testauksen kattavuus

Testauksen suunnittelu

- Jokaisen ohjelmiston testauksen kattavuus ja intensiteetti täytyy päättää erikseen
- Päätöksenteon perusteet
  - **Riskianalyysi**
  - Testaukseen käytettävissä olevat resurssit (henkilöstö, työkalut, osaaminen)
  - Testaukseen käytettävissä oleva aika
- Tavoitteena hyvä hyötysuhde
  - Testaamisen kustannukset vs. saavutettu hyöty
  - Testaukseen 20 – 50% kokonaisbudjetista

Luennon oppimistavoitteet

- Mitä ohjelmistojen testaaminen on?
- Mitä tarkoittavat virhetoiminto, vika ja erehdys?
- Mikä on testauksen suhde ohjelmiston laatuun?
- Mitkä ovat testattavuuden rajat?