TIES546 Ohjelmistotestaus

Kevät 2010

Sami Äyrämö
Tietotekniikan laitos
Jyväskylän yliopisto

Kurssin tavoitteet

- Perehtyminen käytännön ohjelmistotestauksen menetelmiin, tekniikoihin ja haasteisiin
- Saada kokonaiskuva erilaisten sovellusten tämän hetkisistä testausmenetelmistä yrityksissä
- Luennoitsijoina mm. useita yritysvieraita
- Käytännön harjoittelua

Luennot

- 17x2h
  - http://users.jyu.fi/~samiayr/testaus2010/
- 2 johdantoluentoa (ei luentopäiväkirjaa)
- 15 vierailuluentoa
  - 85% läsnäolopakko vierailuluennoilla (max 2 poissaoloa)
  - Vierailuluennoista kirjoitetaan luentopäiväkirja joka pitää palauttaa viimeistään 31.3.2010
  - Sairastapaukset ym. pakottavista syistä johtuvat ylimääräiset poissaolot katsotaan tapaus kerrallaan
  - Tehkää myös kysymyksiä luennoitsijoille!

Luentopäiväkirja

- Kurssilla on 15 luentoa joista vähintään 13:sta tulee pitää luentopäiväkirjaa
- Yhdestä luennosta noin yhden sivun verran tekstiä
- Luentopäiväkirja arvostellaan asteikolla 0-2 pistettä
  - Täydet pisteet edellyttävät omaa pohdintaa

Demot

- 5 x 2h
- Demomestarina toimii Panu Suominen
- Demoista 4 kertaa arvosteltavia tehtäviä
  - Demovastaukset tulee palauttaa seuraaviin demoihin mennessä paitsi viimeiset sähköpostilla 31.3.2010 mennessä
  - Demot arvostellaan asteikolla 0-2 ja pisteet jaetaan demojen määrällä. Lopulliseen arvosanaan pisteistä lasketaan keskiarvo.
  - Kurssin hyväksytty suorittaminen edellyttää demoista vähintään 0,5 pistettä

"Bonustutkielma"

- Valitse itseäsi kiinnostavalta luennolta kolme avainsanaa
- Etsi kolme aiheeseen liittyvää tieteellistä julkaisua kirjaston elektronisten lehtien tietokannasta
- Kirjoita lyhyt 6-8 sivun (laitoksen gradupohjan fonteilla ja riviväleillä) tutkielma aiheesta tutkimusten näkökulmasta

Suorittaminen

- Luentopäiväkirja vierailuluennoista
  - Arvostellaan 0-2p (1p minimi)
- Demot
  - Arvostellaan 0-2p (0.5p minimi)
- Lyhyt tutkielma
  - 0-1p
  - Vapaaehtoinen
  - Vaikuttaa vain arvosanaan
- Kurssin arvosana muodostuu kolmen yllä mainitun osan summasta
  - Kurssilla ei ole tenttiä

Mitä on ohjelmistotestaus?

- Ohjelmistovirheiden etsimistä:
- Myers 2004
  - ”*Testing is a process of executing a program with the intent of finding an error*”
  - *”A good test case is one that has a high probability of finding an as-yet undiscovered error”*
  - *”A successful test is one that uncovers an as-yet undiscovered error”*

Mitä on ohjelmistotestaus?

- ”*Testing is the process of executing a program with the intent of finding errors*”…
- Tavoiteasettelu on olennaista, sillä…
- Dijkstra aforismi:
  - “Program testing can be used to show the presence of bugs, but never to show their absence.”

Määritelmiä

- Virhe (error, mistake)
  - Ohjelmiston poikkeama määrittelystä
- Vika (fault, defect)
  - Aiheutuu suoritettaessa virheellistä tai puutteellista kohtaa koodista
- Häiriö (failure)
  - Viasta johtuva ulkoisesti havaittava poikkeama ohjelmiston toiminnassa
  - Kaikki virheet tai viat eivät johda häiriöihin

- IEEE610.12-1990 (IEEE Standard Glossary of Software Engineering Terminology)

V & V

- Validation:
  - ”*Are we building the right product*”
  - Ohjelmisto vastaa käyttäjän odotuksia
- Verification:
  - ”*Are we building the product right*”
  - Ohjelmisto toimii spesifikaatioiden mukaisesti

- Staattinen tapa (koodi vs. määrittelyt):
  - Katselmoinnit, tarkastukset, automaattinen koodianalyysi, formaalit menetelmät,…
- Dynaaminen tapa (toiminnallisuus):
  - Koodin testaaminen testidatalla

Testaamisen on haastavaa

- On helppo suunnitella joukko testitapauksia, jotka ”osoittavat” että ohjelma toimii oikein
- Vaikeampi tehtävä on suunnitella joukko testitapauksia, jotka osoittavat että ohjelmassa on virhe
- Virheen löytäminen (tai siis korjaaminen…) parantaa ohjelmiston laatua
- Virheettömyyden takaaminen testaamalla on mahdotonta
  - Sekä polkujen kattava läpikäyminen että syötekombinaatioiden määrä epäkäytännöllisiä jo pienilläkin ohjelmilla
- Kattavuuden maksimointi + kustannusten minimointi

ISO 9126-1 laatuominaisuudet

- Toiminnallisuus (Functionality)
  - suitability, accuracy, interoperability, security
- Luotettavuus (Reliability)
  - maturity, fault tolerance, recoverability
- Käytettävyys (Usability)
  - understandability, learnability, operability, attractiveness
- Tehokkuus (Efficiency)
  - time behaviour, resource utilisation
- Ylläpidettävyys (Maintainability)
  - analysability, changeability, stability, testability
- Siirrettävyys (Portability)
  - adaptability, installability, co-existence, replaceability

Testausperiaatteita

- Pressman 2001
  - *”All tests should be traceable to customer requirements”*
  - *”Tests should be planned long before testing begins”*
  - *”The Pareto principle (80-20) applies to software testing”*
    - *Kuinka kriittinen 20% tunnistetaan…*
  - *”Testing should begin ”in the small” and progress toward testing ”in the large”*
  - *”Exhaustive testing is not possible”*
  - *”To be most effective, testing should be conducted by an independent third party”*

Miksi ohjelmistoon jää virheitä ja vikoja

- Kaikkia osia koodista ei testata
- Koodi suoritetaan eri järjestyksessä kuin testissä
- Käyttäjä antaa ohjelmalle testaamattoman syötekombinaation
- Koodia ei ole testattu käyttöympäristössä
- …
- *”It’s written by people – so it’s not perfect”*, Patton 2006

Yleiset testausstrategiat

- White-box testing - lasilaatikkotestaus
- Black-box testing - mustalaatikkotestaus

White-box testing – lasilaatikkotestaus

- Aka Logic-driven testing

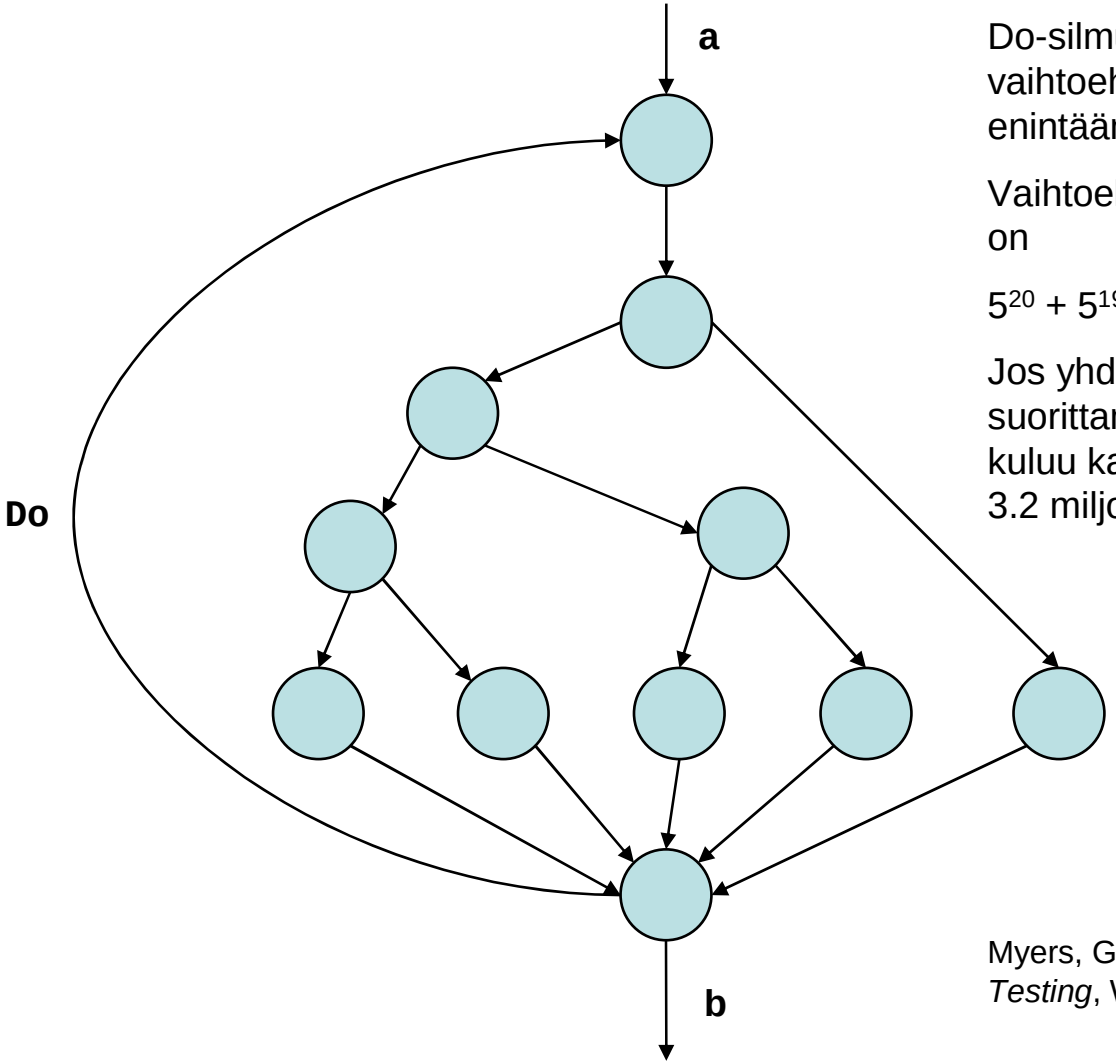
- Testataan ohjelman sisäistä rakennetta
- Edellyttää tietoa siitä miten ohjelma on suunniteltu
  - Testitapaukset johdetaan ohjelmalogiikasta
- Kattava polkujen testaaminen mahdotonta

Kattava lasilaatikkotestaus

- Exhaustive path testing
- Kaikkien mahdollisten lauseiden testaaminen ei riitä
- Kaikkien mahdollisten kontrollivuograafin polkujen testaaminen ei välttämättä riitä (Myers 2004)
  - Kattava polkujen testaaminen
    - ei takaa että ohjelma vastaa määritelmiä
    - ei tunnista puuttuvia polkuja
    - ei tunnista data-sensitiivisyys virheitä
      - Esim. konvergenssitesti

```
if (a – b < c)
   System.out.println(”a – b < c”)
```

Kattava lasilaatikkotestaus



Do-silmukka jonka sisällä viisi vaihtoehtoista polkua suoritetaan enintään 20 kertaa

Vaihtoehtoisia polkuja a:sta b:hen on

$5^{20} + 5^{19} + \ldots + 5^{1} \sim 10^{14}$

Jos yhden testitapauksen suorittaminen kestää sekunnin, kuluu kattavaan testaamiseen aikaa 3.2 miljoona vuotta

Myers, G.J, *The Art of Software Testing*, Wiley, 2004

Black box testing – mustalaatikkotestaus

- Aka Data-driven, input/output driven testing

- Toimiiko ohjelma määritysten mukaisesti
  - Edellyttää tietoa syöte-/vastekombinaatioista
  - Koodin rakenteesta ja logiikasta ei välitetä

- Testitapaukset johdetaan spesifikaatioista
- Kattava syötekombinaatioiden-parien testaaminen mahdotonta
- Testitapaukset pitää valita siten että maksimoidaan virheiden löytymisen todennäköisyys

Kattava mustalaatikko testaaminen – esimerkkejä (Myers 2004)

- `Olkoot meillä ohjelma joka lukee syötteenä kolme kokonaislukua ja tulostaa näytölle onko kyseessä tasasivuinen, tasakylkinen vai epäsäännöllinen kolmio.`
  - Kattava mustalaatikkotestaus edellyttää ei pelkästään kaikkien validien (integer range) vaan kaikkien mahdollisten syötteiden testaamista (negatiiviset luvut (-3,-3,4), kirjaimet (5,A,5)) joita on käytännössä ääretön määrä

Kattava mustalaatikko testaaminen – esimerkkejä (Myers 2004)

- Esim. C++ kääntäjä
  - Testattava ei pelkästään kaikki mahdolliset validit ohjelmat vaan myös kaikki ei-validit ohjelmat (joita on ääretön määrä)
- Tietokantasovellus: Lentolippujen varausjärjestelmä
  - Ei riitä pelkkä validien ja ei-validien tietokantatransaktioiden testaaminen vaan kaikki mahdolliset sekvenssit on testattava
  - Tietokantasovellusten, käyttöjärjestelmien ym. muisti tekee kattavan testaamisen entistä vaativammaksi

Testitapausten suunnittelutekniikat

- Koska kattava testaaminen on mahdotonta
  - Tarvitaan tehokkaita tekniikoita kattavuuden ja tehokkuuden maksimointiin käytettävissä olevan ajan ja resurssien rajoissa
  - *”What subset of all possible test cases has the highest probability of detecting the most errors”*, Myers 2004
- Suunnittelutekniikoiden jako yleisesti:
  - White-box testing – lasilaatikkotestaus
  - Black box testing – mustalaatikkotestaus
  - Yhdistelemällä näiden elementtejä voitaneen suunnitella ei kaiken kattavia mutta järjellisiä testitapauksia

Lasilaatikkotekniikoita 1

- **Path coverage (polkukattavuus)**
  - **Testataan vuograafin kaikki entry-to-exit polut**
  - **Polkujen määrä kasvaa helposti suureksi (esim. silmukkarakenteet)**
  - **Vahvin tekniikka mutta ei käytännöllinen**
- **Statement coverage (lausekattavuus)**
  - **Jokainen lause suoritetaan vähintään kerran**
  - **Lausekattavuus voidaan ilmaista prosentteina**
  - **100% polkukattavuus → 100% lausekattavuus**
- **Decision/branch coverage (päätös-/haarautumiskattavuus)**
  - **Jokainen päätös saa true ja false arvon vähintään kerran**
  - **Kontrollivuon jokainen kaari kuljetaan vähintään kerran**
  - **Vahvempi kuin lausekattavuus (poikkeuksia)**

Lasilaatikkotekniikoita 2

- **Condition coverage (ehtokattavuus)**
  - **Päätöksen jokainen ehto on tosi ja epätosi vähintään kerran**
  - **Vahvempi kuin päätöskattavuus (poikkeuksia), mutta ehtokattavuudesta ei välttämättä seuraa päätöskattavuutta**
- **Decision-condition coverage (päätös-ehtokattavuus)**
  - **Toteuttaa sekä päätös- että ehtokattavuuden**
  - **Ei välttämättä käy läpi kaikkia ehtoja**
- **Multiple-condition coverage (moniehtokattavuus)**
  - **Kaikkien päätösten kaikki mahdolliset ehtokombinaatiot käydään läpi**
  - **100% moniehtokattavuudesta seuraa**
    - **100% päätöskattavuus**
    - **100% ehtokattavuus**
    - **100% päätös-ehtokattavuus**

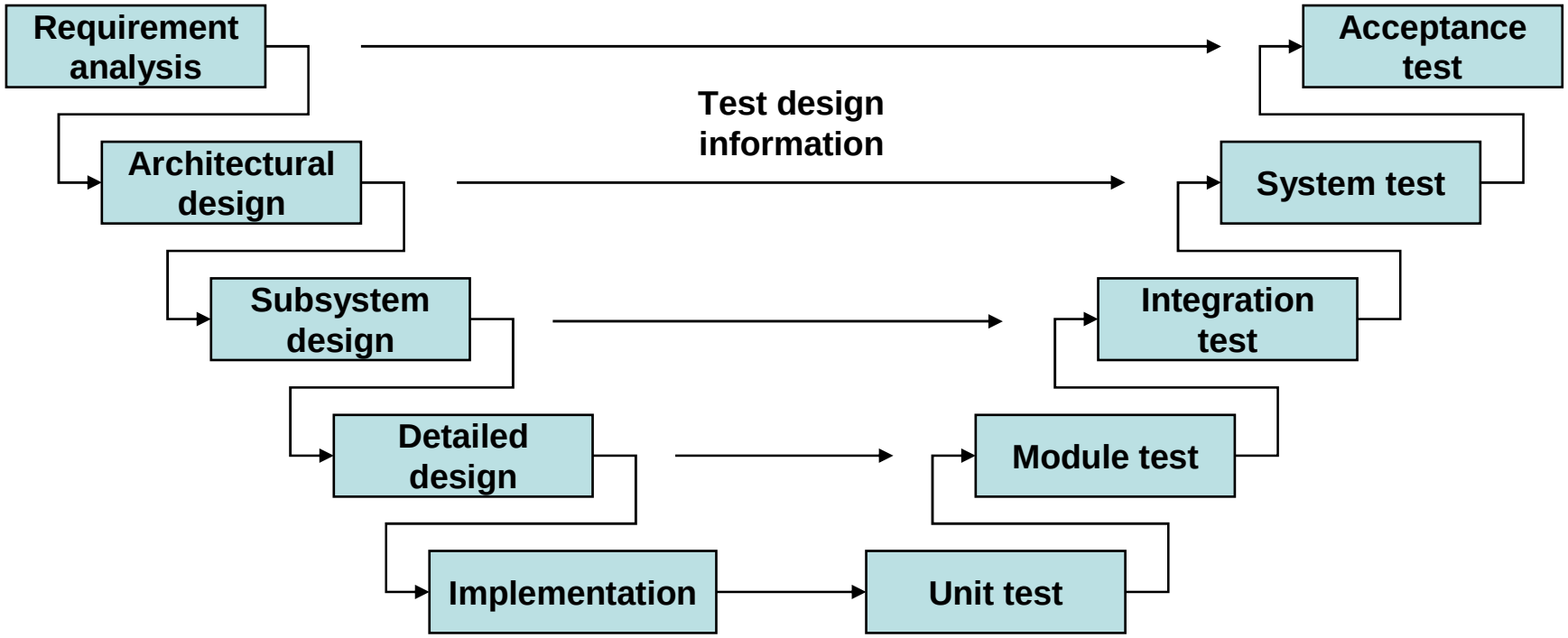
Mustalaatikkotekniikoita

- Ekvivalenssiluokat (equivalence classes)
  - Syöteavaruus jaetaan äärelliseen määrään samanarvoisia luokkia
  - Yhtä ekvivalenssiluokkaa kohti suoritetaan vain edustava testitapaus
  - Luotetaan että jos ekvivalenssiluokasta valittu testitapaus aiheuttaa häiriön niin muutkin luokan tapaukset tekevät samoin (jos ei häiriötä niin oletetaan että muutkaan luokan tapaukset eivät sitä aiheuta)
  - Kaksi vaihetta:
    - 1) ekvivalenssiluokkien tunnistaminen, 2) testitapausten valinta
- Raja-arvo analyysi (boundary value analysis)
  - Valitaan testitapauksia syöteparametrien arvoalueiden rajoilta (esim. silmukan max iteraatiomäärä)
  - Täydentää ekvivalenssiluokkatestausta
    - Valitaan testitapauksia luokkien reuna-arvojen läheltä
- Robustisuus testaus (robustness testing)
  - Valitaan testitapauksia parametrien arvoalueiden ulkopuolelta
- Syy-seuraus-kaavio testaus (cause-effect graph testing)
- Error guessing
  - Intuitiivinen ad-hoc prosessi
  - Testataan virhealttiilta vaikuttavia tapauksia (nolla, tyhjä syötejoukko, yhden alkion syötejoukko, sama arvo kaikilla syötejoukon alkioilla, valmiiksi sortattu syötejoukko)

Ohjelmistotestauksen tasoja

- Yksikkötestaus (unit testing)
- Integrointitestaus (integration testing)
- Järjestelmätestaus (system testing)
- Hyväksymistestaus (acceptance testing)

Esimerkki:
Testauksen tasot vs. V-malli



Ammann and Offutt, *Introduction to software testing,* Cambridge University Press, 2008.

Yksikkötestaus

- **Unit testing:** ”Testing of individual hardware or software units or groups of related units”
  - (IEEE standard glossary of software engineering terminology 610.12-1990)

- **Test unit:** A set of one or more computer program modules together with associated control data, (for example, tables), usage procedures, and operating procedures that satisfy the following conditions (IEEE Standard for Software Unit Testing 1008-1987):
  - (a) All modules are from a single computer program;
  - (b) At least one of the new or changed modules in the set has not completed the unit test;
  - (c) The set of modules together with its associated data and procedures are the sole object of a testing process.

Yksikkötestaus

- Helpottaa suurempien kokonaisuuksien testaamista
- Helpottaa virheiden paikantamista (debuggaus)
- Mahdollistaa rinnakkaiset testausprosessit
  - Testataan useita moduuleja yhtä aikaa
- Yleensä moduulin toteuttaja suorittaa

Yksikkötestaus

- Vastaavatko yksikön/moduulin toiminta ja rajapinnat määritelmiä
  - Moduulin spesifikaatio
  - Lähdekoodi
- Lasilaatikkokeskeistä
  - Eri kattavuuskriteerit käytössä – tehokkaat testitapaukset
  - Voidaan täydentää mustalaatikkotesteillä
- Keskeistä ketterässä ohjelmistokehityksessä
  - Jokaisella moduulilla oltava yksikkötestit ennen toteutusta
  - Jokaisen moduulin pitää läpäistä yksikkötestit ennen julkaisemista

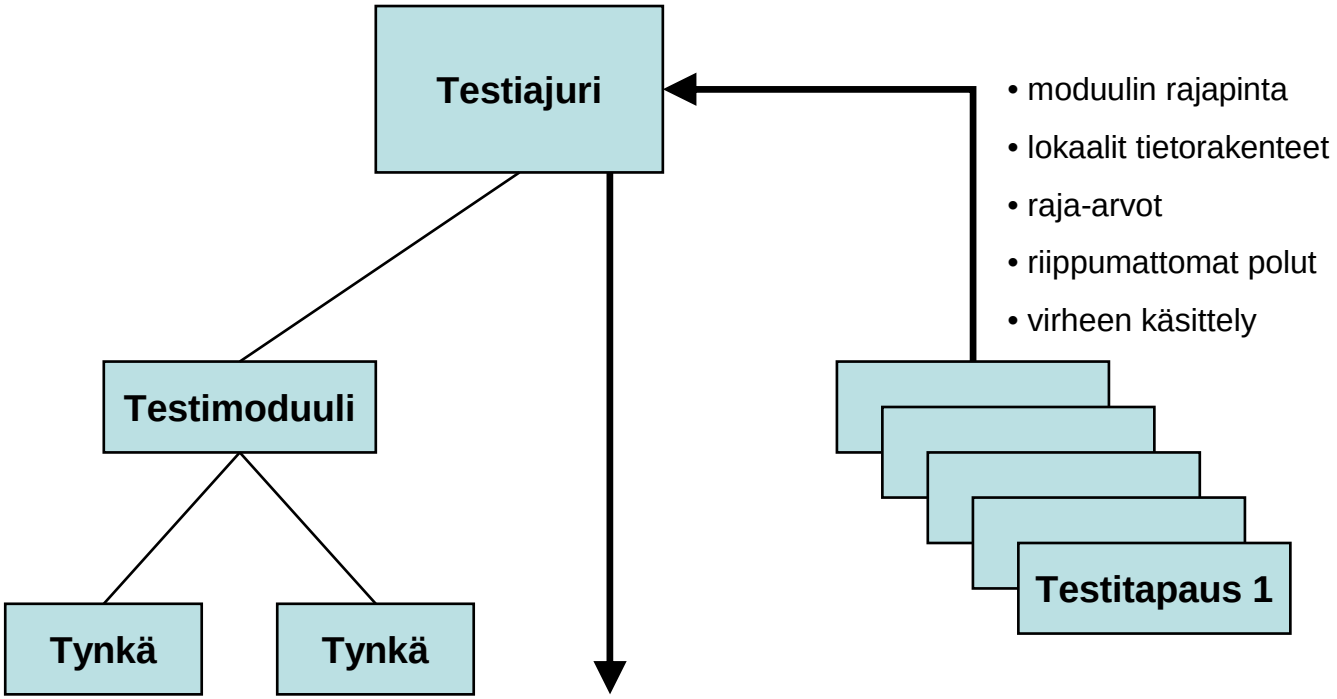
Yksikkötestaus

- Mitä testataan:
  - Moduulin rajapintojen toiminta
    - Tehdään yleensä ensimmäiseksi
    - Data välitys täytyy toimia jotta muut testit voidaan järkevästi suorittaa
  - Lokaalit tietorakenteet
    - Myös lokaalivaikutus globaaleihin tietorakenteisiin
  - Riippumattomat polut
  - Virheiden käsittely!
    - Hallittu virheentilanteen käsittely tai ohjelman lopetus
  - Raja-arvotapaukset
    - Yksikkötestien viimeinen vaihe

Yksikkötestaus

- Moduulit eivät yleensä ole itsenäisiä ohjelmia, joten tarvitaan
  - Testiajurit (test driver)
    - ”Pääohjelma” joka välittää testidata moduulille ja tulostaa asiaankuuluvat tulokset
  - Tyngät (stub, dummy subprogram)
    - Korvaavat moduulit joita kutsutaan testattavasta moduulista
      - Toteuttaa rajapinnat, minimaalista datan manipulointia, palauttaa kontrollin testattavaan moduuliin
- Tynkien ja ajureiden toteuttaminen vaatii aikaa ja resursseja
  - Vaikeissa tapauksissa osa yksikkötesteistä saatetaan joutua tekemään integraatiotestauksen yhteydessä

Yksikkötestaus



Pressman, R.S., *Sofware Engineering*, McGraw-Hill, 2001

Testiajuri ja tynkä
IEEE 610.12-1990

- Testiajuri: Ohjelmistomoduuli jolla kutsutaan testattavaa moduulia, ja usein lisäksi välitetään testisyötteet, kontrolloidaan ja monitoroidaan suoritusta, ja raportoidaan testitulokset
- Tynkä: luurankomainen tai erityistä tarkoitusta varten tehty ohjelmistomuudulin toteutus, jota käytetään kehitettäessä tai testattaessa komponenttia, joka kutsuu sitä tai on muuten riippuvainen siitä

Integraatiotestaus

- Miksi:
  - Itsenäisesti toimiviksi testatut yksiköt eivät välttämättä toimi kokonaisuutena
    - Datan välitys moduulien rajapintojen yli
    - Moduulien keskinäiset riippuvuudet ja vaikutukset
    - Osakokonaisuuksien yhdistäminen kokonaisvaltaisiksi toiminnoiksi
    - Globaalit tietorakenteet
    - …

Integraatiotestaus

- Ei-inkrementaalinen (big-bang)
  - Yksikkötestatut moduulit yhdistetään kokonaisuudeksi kertarysäyksellä
  - Koko ohjelma testataan kerralla
  - Nopea mutta johtaa helposti kaaokseen
    - Virheiden paikallistaminen
    - Virheiden kertautuminen korjattaessa

Integraatiotestaus

- Inkrementaalinen
  - Liitetään testattava yksikkö valmiiksi testatuista yksiköistä muodostuvaan kokonaisuuteen
  - Top-down ja bottom-up strategiat
    - Sandwich-strategia näiden välimuoto

Top-down integraatiotestaus

- Aloitetaan pääohjelmasta (main program)
- Kutsuttavat moduulit korvataan tyngillä
- Moduuleiden lisäys joko
  - syvyys-ensin (depth-first)
    - Mahdollistaa täydellisten toimintojen implementaation ja testaamisen
  - leveys-ensin (breadth-first)
- Ohjelman keskeiset ohjaus- ja päätöspisteet testataan heti prosessin alussa
- Ylätason moduulien voivat olla liian riippuvaisia alatason moduuleista
  1. Osa testeistä voidaan jättää odottamaan alatason moduuleita (testauksen hallinta ja virheiden jäljitys vaikeutuu)
  2. Toteutetaan riittävän täydelliset tyngät (kallista)
  3. Valita bottom-up lähestymistapa

Bottom-up integraatiotestaus

- Aloitetaan integrointi alimman tason komponenteista (atomic modules)
  - Muodostetaan klustereita jotka toteuttavat tietyn alitoiminnallisuuden
  - Testiajureilla kutsutaan testimoduuleita
- Tynkiä ei tarvita
- Ajureiden tarve vähenee edettäessä koodirakenteessa ylöspäin

Regressiotestaus

- Pyritään osoittamaan että ohjelma läpäisee ne testit jotka se on läpäissyt aiemminkin
- Suoritetaan ohjelmaan tehtyjen parannusten tai korjausten jälkeen
  - Esim. integraatiotestauksessa havaitut virheet ja niiden korjaus
- Muutokset voivat aiheuttaa koodiin uusia virheitä
  - Ovat jopa alttiimpia virheille kuin alkuperäinen koodi
- Regressiotestit suoritetaan ajamalla valittu osajoukko testitapauksia
  - Edustava joukko ohjelman toiminnan varmistavia tapauksia
  - Muutosten vaikutuksille alttiit ohjelman osat
  - Muutetut/korjatut moduulit
- Kallista
  - Automatisointi
- XP testing

Järjestelmätestaus

- Testataan kokonaisuuden käyttäytymistä
  - laitteisto+ohjelmisto
- Testataan myös ei-toiminnallisia ominaisuuksia
- Virheiden löytäminen kallista tässä vaiheessa
  - Suurin osa virheistä pitäisi olla havaittuna aiemmissa vaiheissa

Järjestelmätestaus

- Toiminallisuustestaus (functional testing)
  - Toteuttaako järjestelmä vaaditut toiminnallisuudet
- Volyymitestaus (volume testing)
  - Pystyykö ohjelma käsittelemään määritysten mukaisen määrän dataa (esim. suuren lähdekoodin kääntäminen)
- Kuormitustestaus (stress testing)
  - Kuormitetaan järjestelmää datavirralla, keskeytyksillä,...
  - Esim. yhtäaikaiset kirjautumiset www-järjestelmään
- Suorituskykytestaus (performance testing)
  - Vaste- ja läpimenoajat tietyillä kuormilla
  - Etenkin sulautettujen reaali-aikajärjestelmien ominaisuudet
- Toipumistestaus (recovery testing)
  - Käynnistyykö järjestelmä uudelleen vakavan häiriön jälkeen
- Turvallisuustestaus (security testing)
  - Suojamekanismit järjestelmää kohtaan tehtyjä hyökkäyksiä kohtaan

Järjestelmätestaus

- Luotettavuustestaus (reliability testing)
  - Pyritään arvioimaan järjestelmän vikataajuutta (Mean time between failure), toiminta-aikaa,…
  - Huom. ohjelmistot eivät kulu käytössä eivätkä ikäänny samalla tavalla kuin mekaaniset osat ts. käyttämätön ohjelmisto ei vikaannu
- Käytettävyystestaus (usability)
  - ”Onko ohjelman käyttö helppoa?”
  - ”Onko ohjelman käyttö helppo oppia?”
- Konfiguraatiotestaus
  - HW-konfiguraatiot: I/O-laitteet, tietoliikenneyhteydet, muistikomponentit,…
  - Käyttöjärjestelmät, Web-selaimet,…
- Yhteensopivuus-/konfiguraatio-/muutostestaus
  - Ohjelmistoa vaihdettaessa uuteen testataan yhteensopivuus siirrettävien elementtien suhteen
  - Eteenpäin ja taaksepäin yhteensopivuus
    - Esim. text.txt kirjoitettu Win98:ssa Notepadilla
  - Esim: datakapasiteetin riittävyys DBMS muutoksessa
- Asennustestaus (Installability testing)
  - Ohjelmiston asennusohjeiden ja –ohjelmien toimivuus
  - Tärkeää koska voi estää varsinaisen ja hyvin testatun ohjelmiston käytön

Järjestelmätestaus

- Kuka suorittaa, Myers 2004:
  - ”*Programmers should not perform a system test*”
  - ”*Of all the phases, this is the one that the organization responsible for developing the programs definitely should not perform*”
  - Loppukäyttäjä?
  - Järjestelmätestauksen ammattilainen?
- Kuinka simuloida loppukäyttäjän käyttötapoja ja –ympäristöä
- Ideaali järjestelmätestaustiimi?
  - Järjestemätestauksen ammattilaisia, 1-2 loppukäyttäjien edustajaa, käytettävyysasiantuntija, keskeiset järjestelmäkehittäjät,…
- Riippumaton testaustiimi
  - Ulkopuolinen yritys
  - Psykologinen etu
  - Taloudellinen hyöty (löydettyjen vikojen määrä vs. kustannukset

Hyväksymistestaus

- Acceptance testing
- Verrataan ohjelman toimintaa sopimuksen mukaisiin toiminnallisiin ja laadullisiin vaatimuksiin
- Yleensä asiakas suorittaa
- Jos kyseessä on markkinoille julkaistava ohjelmistotuote testaus voidaan antaa muutaman koekäyttäjän vastuulle
  - Alpha-testaus
    - Kehittäjäorganisaation sisäinen koekäyttö
    - Kehittäjä seuraa testausta
  - Beta-testaus
    - Asiakkaiden/asiakas organisaation suorittama koekäyttö
    - Häiriöt raportoidaan kehittäjälle

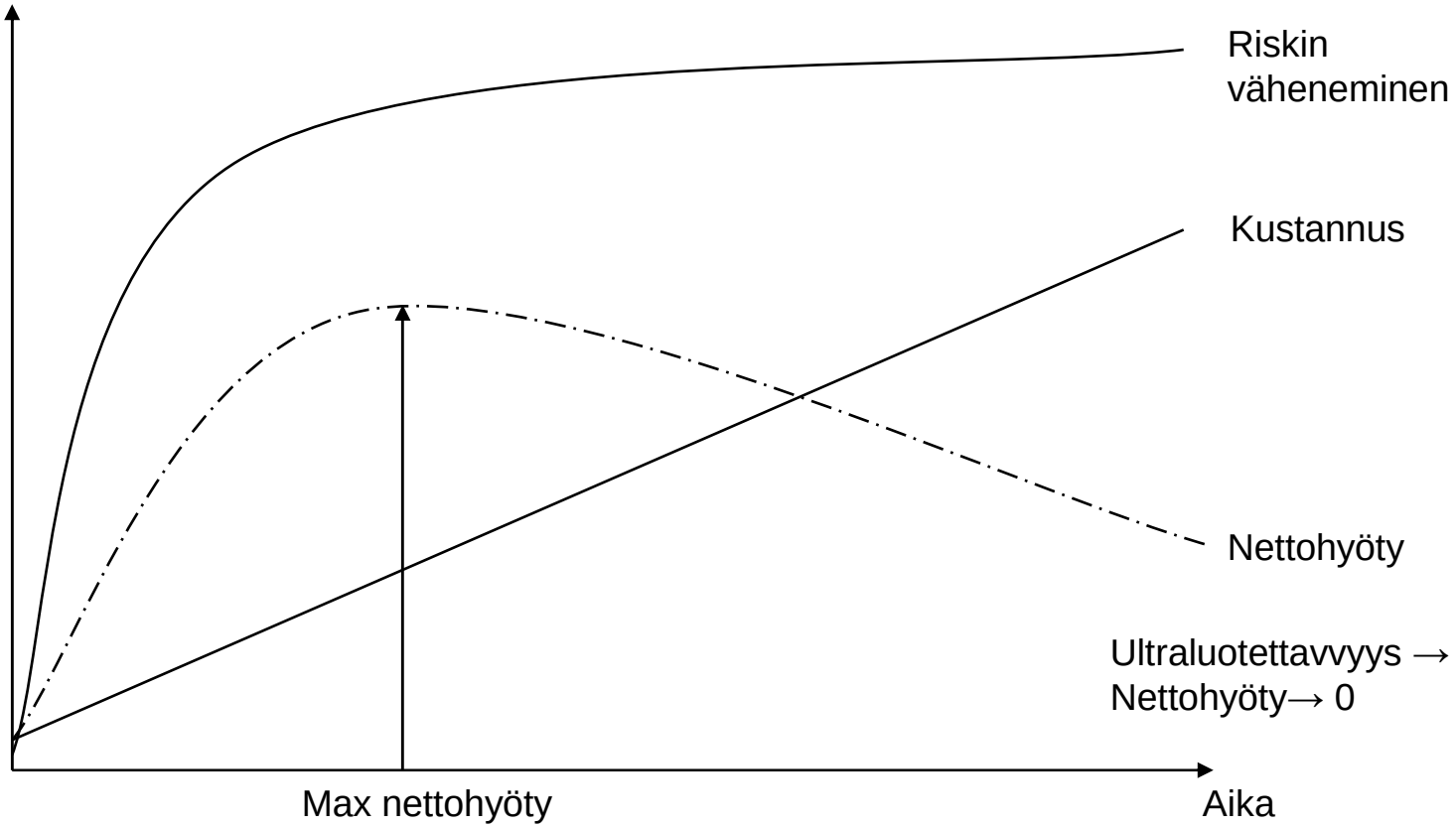
Testioraakkeli

- Oraakkelin tehtävä on ratkaista:
  - Toimiiko ohjelma oikein annetulla syötteellä
- Ihmisen käyttö tehokasta mutta kallista
  → Automatisointi
- Oraakkelien koodaaminen on vaativaa
- Redundantit implementaatiot
  - Verrataan kahden riippumattoman toteutuksen tuloksia
  - Regressiotestauksessa edellinen versio
  - Kaksi eri algoritmia samalle tehtävälle
  - Jotkut syötteet ovat “vaikeampia” hallita
    - Sama virhe toistuu redundantissa koodissa

Testioraakkeli

- Johdonmukaisuus - Consistency checks
  - Ulkoinen (external)
    - Tarkastetaan vaste (esim. todennäköisyys täytyy olla välillä [0,1])
  - Sisäinen (internal)
    - Invariantit
      - Esim. oliosäiliö: dublikaatti olioiden rajoittaminen
      - Tarkastus saattaa olla osa olion toteutusta
      - Suorituskykysyistä voidaan käyttää vain testausvaiheessa
- Redundantti data
  - $\sin(x)^2+\cos(x)^2 = 1$
    - sin ja cos virheet voivat kompensoida toisensa
    - Entäs jos cos toteutettu kutsumalla sin-funktiota?
  - $\sin(a+b) = \sin(a)\cos(b)+\cos(a)\sin(b)$
    - x = a+b – varioidaan a:ta ja tarkistetaan tulos

Testaamisen lopetuskriteeri



Sherer's malli kustannus-tehokkaasta testaamisesta (kuva mukailtu lähteestä): J.C. Huang, *Software error detection: through testing and analysis*, Wiley, 2009

Testaamisen lopetuskriteeri

- Aikarajoite
  - Täyttyy myös tekemättä mitään
- Kaikki testitapaukset menevät läpi
  - Rohkaisee kirjoittamaan testejä jotka eivät löydä virheitä
- Testitapausten suunnitelumenetelmät:
  - Yksikkötestit:
    - Johdetaan testitapaukset moniehtokattavuus kriteereistä ja määrittelydokumentin raja-arvo analyysista ja testataan kunnes kaikki hyväksytysti läpi
  - Toiminnallinen testaus:
    - Johdetaan testitapaukset syy-seuraus-kaaviosta, raja-arvo-analyysista ja virheen arvaustekniikalla ja testataan kunnes kunnes kaikki hyväksytysti läpi
  - Ei toimi välttämättä kaikille järjestelmätesteille
  - Esim. raja-arvo-analyysin kattavuus on vaikea arvioida
- Löydettyjen virheiden minimilukumäärä
  - Esim. kolme virhettä/moduuli
  - Tai yhdistelmä: 50 havaittua ja korjattua virhettä tai 3 kk
  - Kuinka asetetaan rajat
    - Estimoidaan virheiden lukumäärä koko ohjelmassa (esim. edellisen version tai ohjelmistoalan keskiarvojen perusteella)
    - Estimoidaan löydettävissä olevien virheiden %-osuus
    - Estimoidaan missä vaiheessa virheet ovat syntyneet ja missä vaiheessa ne todennäköisimmin löytyvät
      - 40% virheistä ohjelmalogiikan suunnitelu- tai koodausvaiheessa, loput aiemmin (Myers 2004)
- Paras vaihtoehto kokonaisuudelle riippunee sovelluksesta, kehitysorganisaatiosta ym. Ja löytynee edellisten yhdistelmistä

Muuta

- Ketterät menetelmät
- Testilähtöinen ohjelmistokehitys
- Katselmoinnit, läpikäynnit, debuggaus, staattinen analyysi, todistamistekniikat
- Eri sovellustyypit
  - Reaaliaikajärjestelmät, tietokantasovellukset, Internet-sovellukset
- Olio-ohjelmien testaaminen
- Testauksen suunnitteminen
- Testitapausten/virheiden raportointi (tyyppi, luokka, vakavuus,…)
- Ja paljon muuta…

Standardeja, lehtiä,…

- IEEE 829 Testauksen dokumentoinnin standardi
- IEEE610.12-1990 (IEEE Standard Glossary of Software Engineering Terminology)
- ISO 9126-1 laatuominaisuudet
- Software Quality Journal, Springer
- Software Testing, Verification & Reliability, Wiley
- IEEE Transactions on Software