Ohjelmistotuotanto

Ohjelmistojen testaus 1

Laadunvarmistustekniikoita

- Testaus (testing)
  - ohjelman suorittamista tarkoituksena löytää virheitä
- Tarkastukset, katselmukset (inspections, reviews)
  - asiantuntijoiden suorittamia dokumentin (voi olla myös koodi) läpikäyntejä tarkoituksena löytää virheitä
- Osittain samoja virheitä - osittain erilaisia

Testaus – periaatteita

- Testiajo on onnistunut, jos se on paljastanut ohjelmistosta aikaisemmin havaitsemattoman virheen
- Hyvä testitapaus on sellainen, joka suurella todennäköisyydellä paljastaa aiemmin havaitsemattoman virheen
- Hyvä testaus on tuhoavaa toimintaa, toisin kuin esimerkiksi luova ohjelmointi

Testaus – termejä

- virhe (error, mistake, bug):
  - ohjelmiston poikkeaminen sen spesifikaatiosta (määrittelystä), virhe esimerkiksi ohjelmakoodissa
- vika (fault)
  - virheellisen ohjelmakohdan suoritus - vika esiintyy siis vain, jos virheellinen ohjelmakohta suoritetaan
- häiriö (failure)
  - viasta aiheutuva ulkoisesti havaittava poikkeama ohjelmiston spesifikaatiosta
  - kaikki virheet ja viat eivät välttämättä johda häiriöön

Testaus – havaintoja

- Havaintoja:
  - ammattiohjelmoija tuottaa ~1,2 virhettä / 200 riviä ohjelmakoodia
  - uudessa 200000 rivin ohjelmatuotteessa on ~1200 virhettä
  - pitkäänkin käytössä olleissa ohjelmistoissa on noin 1 virhe / 1000 riviä koodia
  - yhden virheen poistamiseen kuluu noin 12 henkilötyötuntia
- Testaus on erittäin tehoton tapa varmistua ohjelman toimivuudesta
- Testaus voi osoittaa ohjelmiston virheellisyyden mutta ei sen oikeellisuutta (Dijkstra)

Testaus – havaintoja

- Tyypillinen virhejakauma
  - Beizer: Software Testing Techniques, 2nd ed, van Nostrand Reinhold, 1990
  - ohjelmakoodin lauseissa 25%
  - ohjelmakoodin tietorakenteissa 22%
  - spesifikaation toteutustavassa 16%
  - moduulien/komponenttien integroinnissa 9%
  - huolimattomuusvirheitä 9%
  - ...
  - spesifikaatioissa 8%

Testaus

- Testitapaus:
  - tietty ohjelmiston syötearvojen joukko

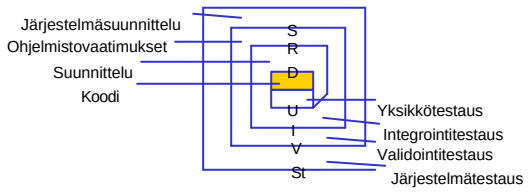
1. Rakenteellinen eli sisäinen eli lasilaatikkotestaus (structural / glass-box / white-box testing)
   - perustuu ohjelmakoodin mukaiseen läpikäyntiin, eli testattava komponentti on 'läpinäkyvä laatikko', joka paljastaa sisäisen rakenteensa
   - testitapaukset valitaan siten, että ne aiheuttavat ohjelmakoodin mahdollisten suoritusreittien läpikäyntiä
   - ohjelmasilmukoista johtuen kaikkien mahdollisten suoritusreittien läpikäynti ei ole yleensä mahdollista järkevässä ajassa

Testaus

2. Toiminnallinen eli funktionaalinen eli ulkoinen eli mustalaatikkotestaus (functional / black-box testing)
   - perustuu ohjelmiston ulkoisesti havaittavan toiminnan tarkasteluun
   - testattava komponentti on 'musta laatikko', jonka sisään ei nähdä
   - testitapaus -> komponentti -> tulokset (ok?)
   - suuresta syöteavaruudesta johtuen (esim. R: real) kaikkien mahdollisten syötteiden testaus ei ole mahdollista järkevässä ajassa

Testaus

Spiraalikuva ohjelmistotuotantoprosessista



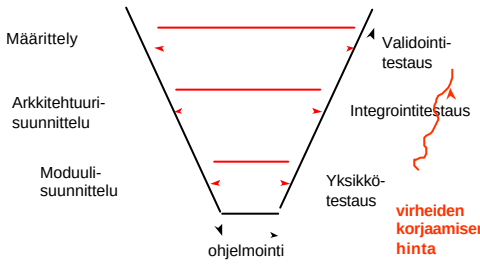
tuotanto sisäänpäin - testaus ulospäin

Testaus – vaiheet

- Yksikkötestaus (unit test):
  - testataan erikseen ohjelmiston jokainen komponentti (moduuli, aliohjelma, luokka, ...)
  - tyypillisesti white-box
- Integrointitestaus (integration test)
  - testataan komponenttien yhteistoiminta
- Validointitestaus (validation test)
  - testataan ohjelmiston ja vaatimusmäärittelyn vastaavuus
- Järjestelmätestaus (system test)
  - testataan järjestelmä kokonaisuudessaan (mukana laitteistot, ympäristö)

tyypillisesti black-box

Testausstrategia: V-malli



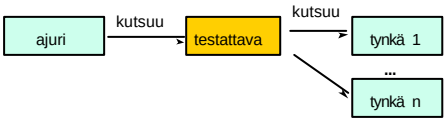
virheiden korjaamisen hinta

Testausstrategia: V-malli

- V-malli kytkee yhteen ohjelmiston rakentamisvaiheen ja testausvaiheen
- Testaussuunnitelma laaditaan kutakin testausvaihetta vastaavassa rakentamisvaiheessa
- Kussakin testausvaiheessa ohjelmiston toimintaa verrataan vastaavan rakentamisvaiheen tuottamaan spesifikaatioon

Testaus – vaiheet

- Yksikkötestaus
  - toimiiko moduuli irrallisena kokonaisuutena?
  - toimivatko paikalliset tietorakenteet ja sisäiset suoritusreitit?
  - voidaan tarvita erityisiä 'tynkiä' (stub) ja 'ajureita' (driver) korvaamaan testattavan kutsumia ja sitä kutsuvia moduuleja



Testaus – vaiheet

- Integrointitestaus
  - toimiiko moduulijoukko integroituna kokonaisuutena (kun yksittäiset moduulit on ensin testattu erikseen)?
  - toimiiko tiedonvälitys moduulien rajapinnoilla?
  - toimiiko moduulien kytkentä?
- Tapa integroida vaikuttaa testaukseen
  - big bang - kaikki valmiiksi, sitten kasaan ja testataan
  - poluittain (threads) - toteutetaan ja testataan suorituspolku kerrallaan
  - hierarkian suhteen top down (tarvitaan tynkiä) tai bottom up (tarvitaan ajureita)
  - useimmiten bottom up tehokkain

Testaus – vaiheet

- Validointitestaus / järjestelmätestaus (usein yhdistetty)
  - toimiiko kokonaisjärjestelmä niin kuin tuotantoprosessin alussa on määritelty?
  - millainen on ulkoisesti havaittava toiminta, suorituskyky, varmuus, siirrettävyys, virheensietokyky, ylläpidettävyys,..?
  - toimivatko ohjelmisto ja laitteisto yhdessä?
  - testilajeja:
    - **alfa-testaus**
      - asiakkaan suorittama tuottajan tiloissa ja laitteilla
    - **beta-testaus**
      - asiakkaan suorittama omassa todellisessa käyttöympäristössään
    - **hyväksymistestaus**, jonka osana voi olla **käytettävyystestaus**

1. Toiminnallinen testaus (black-box)

- testaus perustuu useimmiten syötemuuttujien arvoalueen (domain) osittamiseen => arvoaluetestaus (domain testing)
- parametrin tyyppi (jos tiedossa) kertoo arvoalueen

  var I: integer; I kokonaisluku (välillä $-\infty < I < +\infty$*)*
- ohjelma synnyttää arvoalueita

  if I<0 then ... else ....

  $-\infty < I < 0$ $\qquad$ $0 \le I < +\infty$

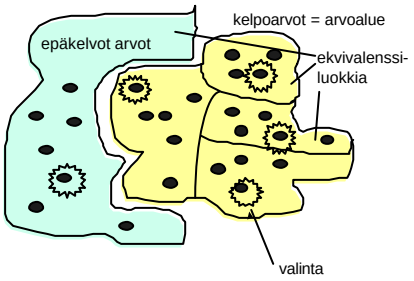
Toiminnallinen testaus (black-box)

- Arvoaluetestauksen periaate:
  - valitaan yksi testitapaus arvoalueen sisältä
  - valitaan arvoalueen rajat
  - valitaan yksi testitapaus välittömästi arvoalueen rajojen ulkopuolelta
  - esim: spesifikaatiossa arvoalue: $0 \le I \le 100$, testiaineisto:
    - I=50 (sisällä, voisi olla yhtä hyvin vaikkapa 78)
    - I=0 (alarajalla)
    - I=100 (ylärajalla)
    - I= -1 (alarajan ulkopuolella)
    - I=101 (ylärajan ulkopuolella)

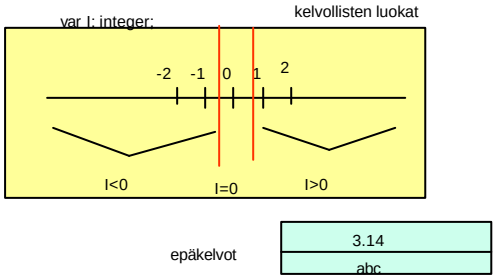
Toiminnallinen testaus (black-box)

- Ekvivalenssiluokat
  - T.J.Ostrand, M.J.Balcer: The Category-Partition Method for Specifying and Generating Functional Tests, CACM, 31, 6, 1988, pp 676-686
  - jokainen arvoalue jaetaan **ekvivalenssiluokkiin,** joille pätee:
    - jokainen tiettyyn ekvivalenssiluokkaan kuuluva arvo käyttäytyy testauksen kannalta 'samalla tavalla'
    - jokainen luokan arvo paljastaa virheen yhtä suurella todennäköisyydellä
    - jos toiminta on virheellistä arvolla α, se on virheellistä muillakin ko. luokan arvoilla
    - jos toiminta on oikein arvolla α, se on oikein myös muilla ko. luokan arvoilla
  - testitapauksiksi kustakin luokasta 1 (tai muutama) arvo

Toiminnallinen testaus (black-box)



Toiminnallinen testaus (black-box)

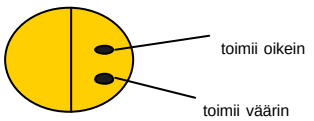


epäkelvot

3.14
abc

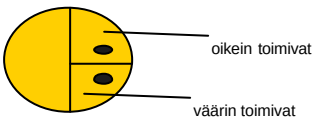
Toiminnallinen testaus (black-box)

- Testauksen aikana ekvivalenssihypoteesi saattaa osoittautua pätemättömäksi



Toiminnallinen testaus (black-box)

- Luokkarajoja on tällöin muutettava



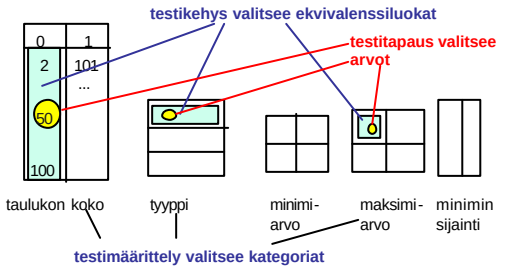
Tekijäperustainen testaus (category partitioning)

- Jaa ohjelmisto spesifikaation perusteella erikseen testattaviksi osajärjestelmiksi
- Tunnista jokaisen osajärjestelmän syötemuuttujat
  - testitapaus on syötemuuttujien alustusarvojen joukko
- Määrittele jokaiselle syötemuuttujalle sen tekijät
  - tekijä, kategoria (category) on muuttujan keskeinen ominaisuus
  - esim. järjestettävä taulukko
    - tekijöitä: taulukon koko, alkioiden tyyppi, alkioiden minimiarvo, alkioiden maksimiarvo, minimiarvon sijainti, maksimiarvon sijainti - asiat jotka ovat oleellisia järjestämisen toimimisen kannalta

Tekijäperustainen testaus (category partitioning)

- Määrää jokaiselle kategorialle arvojen ekvivalenssiluokat (choice)
  - esim. järjestettävä taulukko / taulukon koko:
    {koko=0} {koko=1} {2<=koko<=100} {koko>100}
- Tuota testimäärittely jokaiselle osajärjestelmälle
  - kategorialista
  - kunkin kategorian ekvivalenssiluokat
  - näiden perusteella voidaan määrätä **testikehykset** (test frame)
    - testikehys muodostetaan valitsemalla joukko kategorioita
    - tarkoituksena on muodostaa testitapaus valitsemalla ensin yksi ekvivalenssiluokka kuhunkin testikehyksen kategoriaan liittyen ja sitten yksi arvo kustakin valitusta luokasta

Tekijäperustainen testaus (category partitioning)



Tekijäperustainen testaus (category partitioning)

- Lisää testimäärittelyihin rajoitukset, jotka koskevat kategorioiden ja ekvivalenssiluokkien mukaanottamista testikehyksiin
  - esim. jotkut ekvivalenssiluokat eivät voi olla samaan aikaan voimassa, kuten {taulukon koko=0} ja {minimin sijainti>100}
  - rajoituksilla vähennetään testikehysten ja -tapausten lukumäärää
- Tuota testikehykset automaattisesti (testaustyökalulla)
- Tuota testikehyksiin liittyvät testitapaukset (testaustyökalulla)
- Yhdistä loogisesti yhteenkuuluvat testitapaukset testijonoiksi (test script)

Tekijäperustainen testaus (category partitioning)

- Testaa osajärjestelmät suorittamalla testijonot
- Ylläpidä testimäärittelyjä ja -tuloksia testitietokannassa
  - jos saman ekvivalenssiluokan testitapaukset aiheuttavat ristiriitaista toimintaa, muuta luokkajakoa

Ekvivalenssiluokat

- Luokittelu on pyrittävä tekemään niiden ominaisuuksien mukaan, jotka todennäköisimmin aiheuttavat virheitä
- Testitapauksia on valittava luokan koon mukaan:
  - iso luokka - enemmän tapauksia luokituksen epäonnistumisen huomaamiseksi
- Huono luokittelu paljastaa (yleisrasitteen takia) virheitä tehottomammin kuin puhtaasti satunnainen testaus, jossa koko arvoalue tulkitaan yhdeksi ekvivalenssiluokaksi ja testitapaukset valitaan satunnaisesti

Toiminnallinen testaus – esimerkki

***Find* (document, text, direction, match case)**

- *document* : työn alla oleva tekstidokumentti
- *text* : dokumentista etsittävä merkkijono
- *direction* (*down*, *up*): etsintäsuunta suhteessa kohdistimen (kursorin) nykyiseen sijaintiin
- *match case* (*yes*, *no*): vaikuttaako kirjainten koko hakuun vai ei (so., tulkitaanko iso ja pieni kirjain samoiksi)

➢ myös *kohdistimen sijainti* vaikuttaa operaation toimintaan; jätetään tässä yksinkertaisuuden vuoksi pois

Toiminnallinen testaus – esimerkki

Ekvivalenssiluokat **:**

- *text* :
  {merkkijonot, joissa pieniä kirjaimia muttei isoja}
  {merkkijonot, joissa isoja kirjaimia muttei pieniä}
  {merkkijonot, joissa sekä pieniä että isoja kirjaimia}
  {merkkijonot, joissa ei lainkaan kirjaimia}
  {tyhjät (laittomat) merkkijonot}
- *direction* : {*down*}, {*up*}
- *match case* : {*yes*}, {*no*}
- *document* : {*found*}, {*not found*} – löytyikö vaiko ei

Menetelmä : ekvivalenssiluokkien yhdistely

text					dir.		match		doc.	
pk	ik	pik	ei-pik	e	d	u	y	n	f	n-f
x					x		x		x	
x					x		x			x
x						x	x		x	
x						x	x			x
	...									
	x					x	x			x
	x				x			x	x	
	...									
				x		x		x		x

Toiminnallinen testaus – esimerkki

(Riippumattomien) yhdistelmien eli kombinaatioiden määrä = suoritettavien testien määrä:

$$E_1 * E_2 * \ldots * E_k$$

E_i = parametrille i määriteltyjen ekvivalenssiluokkien määrä

Tässä: $5 * 2 * 2 * 2 = 40$

Huom: Joitakin (laittomia syötearvoja sisältäviä) yhdistelmiä ei ehkä voi suorittaa, mutta sekin on testattava!

Toiminnallinen testaus – esimerkki

Testitapausten hahmot (40 kpl):

1 *text* : pieni-kirjain, *direction* : down, *match case* : yes, *document* : found
2 *text* : pieni-kirjain, *direction* : down, *match case* : yes, *document* : ***not found***
3 *text* : pieni-kirjain, *direction* : ***up***, *match case* : yes, *document* : found
4 *text* : pieni-kirjain, *direction* : up, *match case* : yes, *document* : ***not found***
...
40 *text* : ***empty***, *direction* : up, *match case* : no, *document* : not found

Toiminnallinen testaus – esimerkki

- Jokaiselle hahmossa esiintyvälle ekvivalenssiluokalle valitaan sitä edustava parametrin arvo
- Samasta ekvivalenssiluokasta valitaan eri testitapauksissa eri arvoja
- Valitaan raja-arvoja, mikäli mahdollista
  - tekstille sekä lyhyitä että pitkiä merkkijonoja
  - tekstille koko merkistöä
  - kohdistin sekä dokumentin alussa että lopussa

document		text	direction	match case
This **bea**utiful text	(1)	bea	down	yes
This beautiful text	(2)	beatles	down	yes
This **1bea**utiful text	(3)	1bea	up	yes
This 1Beautiful text	(4)	1bea	up	yes
This & **%1bEAu**tiful text	(5)	%1beau	down	no
This &%2beautiful text	(6)	%1beau	down	no
This **B**E utiful text	(7)	b	up	no
This BE utiful text	(8)	beauti	up	no
This **BEA**UTIFUL text	(9)	BEA	down	yes
This BEAUTIFUL text	(10)	BEAT	down	yes
THIS beautiFUL text	(11)	THIS	up	yes
THIS beatiful text	(12)	TH2S	up	yes
T**his** Beautiful Text	(13)	HIS	down	no
this %#& beautiful text	(14)	S	down	no
this %#& beautiful text	(15)	HIS %#&	up	no
This %#&beautiful text	(16)	#& BE	up	no

document		text	direction	match case
This Beautiful **Text**	(17)	Text	down	yes
This Beautiful Text		Text	down	yes
T**HIS is** beautiful text		IS is	up	yes
This is beautiful text		IS is	up	yes
T**his text** 1-99		ExT 1	down	no
This text 1 and text 2		eXt 1	down	no
This was beautiful text		His Was Beauti	up	no
(This) (Was) (123text)	(24)	aS( )	up	no
123 one-two-three	(25)	123	down	yes
One-two-three 1-2-3		12-3	down	yes
This &007&mess		&	up	yes
This Bloody Mess		#%	up	yes
(This) (was1) (was[**2**])		2]	down	no
Q987654321!"#%&/*///		7654321#	down	no
1!2"3**#4$5%6&7/8(9)**0=oops		#4$5%6&7/8(9)	up	no
This %#&beautiful text	(32)	22	up	no
This is beautiful texT	(33)		down	yes
1 or two			down	yes
1 or two			up	yes
QK1+(BThose			up	yes
1 & 2			down	no
_			down	no
This %#&beautiful text			up	no
_	(40)		up	no

Jos kombinaatioita on liikaa, pitää niitä karsia.
"Laiskan miehen menetelmä", jossa vaaditaan vain, että jokaisen ekvivalenssiluokan on oltava mukana *jossakin* testissä (vähintään yhdessä), saattaa johtaa harhateille:

text					*dir.*		*match*		*doc.*	
pk	*ik*	*pik*	*ei-pik*	*e*	*d*	*u*	*y*	*n*	*f*	*n-f*
x					x		x		x	
	x				x		x		x	
		x			x		x		x	
			x		x		x		x	
				x		x		x		x

Mukana ainoastaan laittomassa syötteessä!