

hyväksymispäivä

arvosana

arvostelija

Mutaatiotestaus oliojärjestelmissä

Eveliina Pakarinen

Aine

HELSINGIN YLIOPISTO

Tietojenkäsittelytieteen laitos

Helsinki, 7. lokakuuta 2015

Tiedekunta — Fakultet — Faculty		Laitos — Institution — Department	
Matemaattis-luonnontieteellinen		Tietojenkäsittelytieteen laitos	
Tekijä — Författare — Author			
Eveliina Pakarinen			
Työn nimi — Arbetets titel — Title			
Mutaatitotestaus oliojärjestelmissä			
Oppiaine — Läroämne — Subject			
Tietojenkäsittelytiede			
Työn laji — Arbetets art — Level		Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Aine		7. lokakuuta 2015	7
Tiivistelmä — Referat — Abstract			
Aineen tiivistelmä			
Avainsanat — Nyckelord — Keywords			
mutaatitotestaus, oliojärjestelmät, Java			
Säilytyspaikka — Förvaringsställe — Where deposited			
Muita tietoja — Övriga uppgifter — Additional information			

Sisältö

1 Johdanto [1s]	1
2 Nämä luvut ovat ihan väärin	1
3 Testaus oliojärjestelmissä [2s]	2
3.1 Testauksen rooli oliojärjestelmissä	2
3.2 Testauksen tasot	2
3.3 Testien suunnittelu	3
3.4 Testauksen rajoitukset	4
4 Mutaatiotestaus oliojärjestelmissä [3s]	4
4.1 Mutaatiotestauksen yleisesittely	4
4.2 Mutaatiotestauksen piirteet oliojärjestelmissä (erot muihin järjestelmiin?)	5
5 Mutaatiotestauksen haasteet [3s]	5
5.1 Ekvivalentit mutantit	5
5.2 Tehokkuusongelmat	5
5.3 Muita ongelmia?	6
5.4 Käytännöntoteutus (miten käytännössä on yritetty kiertää haasteet)	6
6 Miten mutaatiotestaus auttaa testien laadun parantamisessa?	6
6.1 Miksi mutaatiotestausta tulisi käyttää, vaikka se on raskasta ja työlästä?	6
6.2 Miltä tulevaisuus näyttää, tuleeko käyttö lisääntymään vai jääkö mutaatiotestaus unohduksiin/pienen piirin harrastukseksi?	6
7 Yhteenveto [1s]	6
Lähteet	7

1 Johdanto [1s]

Tutkimuskysymys: On tutkittu, että mutaatiotestauksesta on hyötyä testien laadun parantamisessa ja testikattavuuden tarkastelussa/kasvattamisessa. Miksi mutaatiotestausta ei kuitenkaan käytetä laajasti näiden asioiden tekemiseen?

Vastaus: Koska mutaatiotestaus on tällä hetkellä vielä työlästä ja tehotonta sen tuomaan hyötyyn nähden. Koska mutaatiotestauksessa on ratkaisemattomia ongelmia (esim. ekvivalentit mutantit), joita ei voi automatisoida -> vaativat ihmisten panostusta -> lisää työläyttä.

Tai toisinpäin: Miksi mutaatiotestausta käytettäisiin/pitäisi käyttää enemmän testaamiseen, vaikka mutaatiotestaus on tehotonta/työlästä?

Vastaus: Koska mutaatiotestausta voidaan käyttää testien laadun parantamiseen tai testikattavuuden kasvattamiseen. Koska mutaatiotestauksen avulla voi varmistaa, että testien laatu on hyvä ja näin ollen parantaa samalla ohjelmiston laatua -> ohjelmistosta saadaan suurin osa ongelmista ja virheistä pois testien avulla. Koska mutaatiotestauksella voi paikata normaaliin testaukseen jääneitä puutteita.

Kuvaus aineesta ja sen osioista (esim. luvussa 2 käsitellään tätä ja tätä ja luvussa 3 tätä) niin kuin artikkeleissa?

Johdanto jäsennelty asianmukaisesti (kenelle, miksi, millaisessa ympäristössä; ratkaisun lähestymistapa; tutkimuskysymys, tulokset ja impakti; loppulukujen roolitus), pituus 1,5 - 2 s.

MIKÄ ON MINUN TUTKIMUSKYSYMYKSENI?

2 Nämä luvut ovat ihan väärin

Luku 2 purkaa auki teknisemmälle yleisölle samalla käsitteitä määrittellen tutkimuskysymyksen, ratkaisuille ympäristöstä esiintulevat vaatimukset ja keskeiset käsitteet näiden asioiden esittelemiseksi. Antaa riittävästi termistöä ja kysymyksenasetteluja, jotta myöhemmänä teksissä on mahdollisuus analyysiin ja evaluointiin.

MITÄ MÄ LAITAN TÄHÄN LUKUUN?

3 Testaus oliojärjestelmissä [2s]

Olioperustaisen ohjelmoinnin kehityksen myötä klassisia ohjelmistojen testausmenetelmiä on sopeutettu mahdollistamaan *oliojärjestelmien* (*object oriented systems*) kattava ja laadukas testaaminen. Vaikka olioperustainen ohjelmointi ratkaisee joitakin proseduraalisen ohjelmoinnin suunnittelu- ja toteutusongelmia, olio-ohjelmoinnin mukana tulevat uudet haasteet vaativat uusien testaus- ja analysointimenetelmien kehittämistä.

3.1 Testauksen rooli oliojärjestelmissä

Testausta käytetään ohjelmistokehityksessä ohjelmiston laadun varmistamiseen ja auttamaan virheiden havaitsemisessa jo kehitysvaiheen aikana. (Binderin mukaan: väite melkein suora käännös kirjan lauseesta) Ohjelmistojen testaamisen ensisijainen tavoite on siis paljastaa virheitä, joiden havaitseminen muiden laadunvarmistusmenetelmien avulla olisi työlästä tai mahdotonta [Bin99, s. 59]. Testauksen avulla pyritään lisäksi varmistamaan, että ohjelma toimii sille asetettujen vaatimusten mukaisesti.

Olio-ohjelmoinnissa testaukseen tuovat haasteita olio-ohjelmien erityispiirteet, joita ovat muun muassa kapselointi, perintä, dynaaminen sidonta ja polymorfismi [MP08, s. 86]. **Lisää tähän? Binderin kirjasta löyty tietoa.**

3.2 Testauksen tasot

Ohjelmistoja voidaan testata usealla tasolla. Tasoja ovat yksikkö-, integraatio- ja järjestelmätasot ja ne muodostuvat joukosta ohjelman komponentteja, joita tason testit testaavat [Bin99, s. 45]. Komponentteja ovat olio-ohjelmissa esimerkiksi yksittäiset metodit ja luokat, ohjelman luokkien väliset rajapinnat tai jo valmis ohjelmisto.

Alimmalla testauksen tasolla *yksikkötestauksessa* (*unit testing*) [Bin99, s. 45] testataan ohjelman pienimpiä suoritettavissa olevia komponentteja. Olio-ohjelmissa komponentteja ovat yksittäiset metodit ja oliot. **Lisää tietoa yksikkötestauksesta.**

Yksikkötestauksesta seuraava taso ylöspäin on *integraatiotestaus* (*inte-*

gration testing) [Bin99, s. 45], jossa tarkastellaan järjestelmän tai sen osien yhteistoimintaa. Integraatiotestauksessa testataan siis järjestelmän osien välistä rajapintoja ja osien keskinäistä kommunikointia. Olio-ohjelmissa luokkien muodostuminen perinnän avulla ja luokkien koostuminen toisten luokkien olioista aiheuttaa, että integraatiotestaukselle on olio-ohjelmoinnissa tarvetta jo ohjelmoinnin alkuvaiheessa. **Lisää ehkä vähän myös tästä.**

Valmista integroitua sovellusta testataan *järjestelmätestauksen* (*system testing*) [Bin99, s. 45] avulla. Tällä testauksen tasolla keskitytään vain valmiissa sovelluksessa esiintyvien piirteiden testaamiseen. Testauksen kohteena voi olla esimerkiksi sovelluksen toiminnallisuus, suorituskyky tai sovelluksen kestävä kuormitus [Bin99, s. 45]. **Tästä jonkin verran lisää.**

3.3 Testien suunnittelu

Testien suunnitteluun ja kehittämiseen voidaan käyttää erilaisia menetelmiä. Testausmenetelmän avulla kuvataan näkökulmaa, josta ohjelman lähdekoodia tarkastellaan testejä kehitettäessä [Bin99, s. 51]. Testausmenetelmiä ovat esimerkiksi white box - ja black box -testaus sekä niitä yhdistävä hybriditestaus. Lisäksi testien laadun parantamiseen voidaan käyttää virheisiin perustuvaa testausmenetelmää [?].

Ohjelman sisäisen rakenteen eli lähdekoodin tuntemukseen perustuvaa testausmenetelmää kutsutaan *white box -testaukseksi* [Bin99, s. 52]. **Lisää tästä. Seuraava omaa keksintöä:** White box -testausta voidaan käyttää esimerkiksi yksikkötestauksessa apuna testien suunnittelussa, sillä lähdekoodin tuntemus auttaa kehittämään testejä yksittäisille metodeille ja olioille [?].

Black box -testaukseksi tai *funktionaaliseksi testaukseksi* [Bin99, s. 52] kutsutussa testausmenetelmässä testejä suunnitellaan analysoimalla ohjelmiston ulkoista toiminnallisuutta. **Lisää tuosta. Oma keksintöä:** Koska valmiin sovelluksen piirteitä testatessa tutkitaan myös sovelluksen ulkoista toiminnallisuutta, on black box -testauksesta apua esimerkiksi suunniteltaessa testejä järjestelmätestaukseen [?].

Gray box - eli *hybriditestauksessa* [Bin99, s. 52] sekoitetaan white box - ja black box -testausmenetelmien piirteitä. **Omaa keksintöä:** White box - ja black box -testausta voidaan siis käyttää testien suunnittelussa erikseen

tai molempien piirteitä yhdistäen useilla testauksen tasoilla [?].

Testausmenetelmää, jossa ohjelman lähdekoodiin lisätään virheitä, kutsutaan *virheperustaiseksi testausmenetelmäksi (fault-based testing)* [Bin99, s. 52]. Esimerkkinä virheperustaisesta testausmenetelmästä on mutaatiotestaus, jonka avulla tutkitaan testien kykyä havaita ohjelmiston virheitä [?, s. X].

3.4 Testauksen rajoitukset

Kuinka paljon rajoituksista on kerrottava? Niitä on aika paljon erilaisia. Mitkä niistä ovat relevantteja tämän tutkielman kannalta?

Testaukseen kuuluu kuitenkin myös rajoituksia. Testauksen avulla ei voi esimerkiksi todeta ohjelmiston oikeellisuutta. Ohjelmistojen oikeaksi todistamiseen ja testauksen rajoituksiin liittyen Dijkstra kirjoitti seuraavan korollaarin: *”Program testing can be used to show the presence of bugs, but never to show their absence!”* [DDH72, s. 6].

4 Mutaatiotestaus oliojärjestelmissä [3s]

Haasteisiin voidaan vastata mutaatiotestauksen avulla.

Vaikka testauksen avulla ei voikaan varmistua ohjelmiston oikeellisuudesta, voidaan testausta käyttää välineenä ohjelmiston laadun parantamisessa. Testaukseen sisältyvien rajoitusten lisäksi siihen liittyy myös epävarmuutta käytettävän testausjärjestelmän oikeellisuudesta ja oikeellisuuden varmistamisesta [MW78, s. 209]. Tämä herättää kysymyksen siitä, kuka voi ”valvoa valvojia” eli kuinka varmistetaan ohjelmiston testien laadukkuus.

4.1 Mutaatiotestauksen yleisesittely

Yksi strategia testien laadun varmistamiseen on *mutaatiotestaus*. Mutaatiotestauksessa ohjelmiston alkuperäistä lähdekoodia käsitellään *mutaatio-operaattoreilla (mutation operators)*, jotka muuntavat koodia muodostaen siitä virheellisiä versioita. Näitä virheellisiä ohjelmakoodin versioita kutsutaan *mutanteiksi* [MHK06, s. 869].

Mutanttien generoinnin jälkeen ohjelmiston alkuperäiset testit suoritetaan jokaisen mutantin kohdalla. Tavoitteena on, että testien avulla havaitaan lähdekoodiin tehdyt muutokset. Jos alkuperäiset testit eivät mene läpi, se tarkoittaa, että mutantti on tapettu eli lähdekoodiin tehdyt muutokset on havaittu [KCM00, s. 9].

Mutaatiotestausprosessi tuottaa lopputuloksena *mutaatiopistemäärän* (*mutation adequacy score*), jonka avulla voidaan arvioida ohjelmiston testien laadukkuutta ja kykyä havaita lähdekoodissa olevia vikoja.

Mutaatiotestaus on *virheperustainen testausmenetelmä* (*fault based jotakin*), jonka periaatteena on ohjelmoijien tekemien ohjelmointivirheiden simulointi [JH11, s. 649]. Tavoitteena mutaatiotestauksessa on tutkia, ovatko ohjelmistoa varten tehdyt testit laadukkaita ja havaitaanko niillä kattavasti ohjelmistossa mahdollisesti esiintyvät virheet ja ongelmat.

4.2 Mutaatiotestauksen piirteet oliojärjestelmissä (erot muihin järjestelmiin?)

Erityisesti juuri nuo mutaatio-operaattorit ja minkälaisia erilaisia niitä on. Esittelyä tulee mutaatio-operaattoreista.

5 Mutaatiotestauksen haasteet [3s]

Misi mutaatiotestaus ei ole päätenyt suureen suosioon/käyttöön? Valitaan muutama haaste ja esitellään niitä? Kerrotaan että myös muita haasteita, mutta ei esitellä niitä niin tarkasti?

Haasteet sisältäen esimerkin, miten se yksittäinen haaste on yritetty ratkaista (eli joko ratkaisu tai ratkaisuehdotus)??

5.1 Ekvivalentit mutantit

Ratkaisematon ongelma. Pitäisi avata ja kehittää esimerkki.

5.2 Tehokkuusongelmat

Laitteisto, testien määrä, ihmisten aika tulee vastaan. Esimerkkejä.

5.3 Muita ongelmia?

Varmasti on muita, mutta kuinka paljon niitä mahtuu tähän esitelmään?

5.4 Käytännöntoteutus (miten käytännössä on yritetty kiertää haasteet)

Onko tämä relevantti aihealue käsitellä?

6 Miten mutaatiotestaus auttaa testien laadun parantamisessa?

6.1 Miksi mutaatiotestausta tulisi käyttää, vaikka se on raskasta ja työlästä?

6.2 Miltä tulevaisuus näyttää, tuleeko käyttö lisääntymään vai jääkö mutaatiotestaus unohduksiin/pienen piirin harastukseksi?

7 Yhteenveto [1s]

Conclusion.

Lähteet

- Bin99 Binder, Robert V.: *Testing Object-oriented Systems: Models, Patterns, and Tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999, ISBN 0-201-80938-9.
- DDH72 Dahl, O. J., Dijkstra, E. W. ja Hoare, C. A. R. (toimittajat): *Structured Programming*. Academic Press Ltd., London, UK, UK, 1972, ISBN 0-12-200550-3.
- JH11 Jia, Yue ja Harman, Mark: *An Analysis and Survey of the Development of Mutation Testing*. IEEE Trans. Softw. Eng., 37(5):649–678, syyskuu 2011, ISSN 0098-5589. <http://dx.doi.org/10.1109/TSE.2010.62>.
- KCM00 Kim, Sunwoo, Clark, John A. ja McDermid, John A.: *Class Mutation: Mutation Testing for Object-oriented Programs*. Teoksessa *Proceedings of the Net.ObjectDays Conference on Object-Oriented Software Systems*, sivut 9–12, lokakuu 2000.
- MHK06 Ma, Yu Seung, Harrold, Mary Jean ja Kwon, Yong Rae: *Evaluation of Mutation Testing for Object-oriented Programs*. Teoksessa *Proceedings of the 28th International Conference on Software Engineering, ICSE '06*, sivut 869–872, New York, NY, USA, 2006. ACM, ISBN 1-59593-375-1. <http://doi.acm.org/10.1145/1134285.1134437>.
- MP08 Mariani, Leonardo ja Pezze, Mauro: *Testing Object-Oriented Software*, luku Emerging Methods, Technologies and Process Management in Software Engineering, sivut 85–108. Wiley-IEEE Computer Society Press, 2008.
- MW78 Manna, Zohar ja Waldinger, Richard J.: *The Logic of Computer Programming*. IEEE Trans. Software Eng., 4(3):199–229, 1978. <http://doi.ieeecomputersociety.org/10.1109/TSE.1978.231499>.