JURNAL INFORMATIKA, Vol.5 No.1 April 2018, pp. 112~118
ISSN: 2355-6579
E-ISSN**:** 2528-2247

Strategi Perbaikan Uji Coba Struktural Perangkat Lunak Pada Metode *White-Box*

Agus Pamuji

Universitas Indraprasta PGRI
e-mail: agus.pamuji@gmail.com

Abstrak

Uji coba perangkat lunak merupakan aktifitas yang sangat menentukan sebelum diterima oleh pengguna akhir. Pada siklus pengembangan perangkat lunak, aktiftas uji coba menghabiskan 50% biaya, usaha, dan waktu. Hal ini terutama pada teknik pengujian dengan menggunakan metode *white-box* yang memerlukan waktu yang lama. Dalam penelitian ini, diusulkan sebuah strategi untuk memperbaiki uji coba struktural menggunakan 4 tahap uji coba yaitu uji alur kontrol, uji alur data, uji coba berbasis *slice*, uji coba mutasi melalui penerapan parameter metrik uji. Adapun metrik uji antara lain perancangan jumlah kasus uji, jumlah kasus uji dieksekusi, jumlah kasus uji lolos, jumlah kasus uji gagal, waktu ekseskusi kasus uji, dan waktu yang digunakan selama proses pengembangan. Metode ini untuk mengurangi rawan kesalahan dan mempercepat proses uji coba. Hasil akhir menunjukan bahwa dengan strategi uji coba yang diterapkan dapat menurunkan tingkat dan rawan jumlah kesalahan walapun pada awalnya mengalami peningkatan pada tahap 1 dan 2.

Kata Kunci: uji coba, metrik uji, strategi, struktural, white-box

Abstract

Software testing is a crucial activity that have the goal to determine before it are accepted by end-users. In the software development life cycle, testing activity has spent about 50% on cost, effort, and the time. This is especially on the testing techniques when the using white-box method that have takes a long time. In this study, a strategy was proposed to improve the structural testing through four phases, i.e, control flow testing, data flow testing, slice based testing, and the mutation through the implementation of testing metrics parameter. The testing metric include designing the number of test case, a number of test cases executed, a number of test cases passed, a number of test cases failed, a test case execution time, and the time spent during the development process. This method are reduced the error prone and to increas during testing process. As a result show that with the experimental strategy was applied could decrease the level and prone to the number of errors even though initially increased on the 1 and 2 phases.

Keywords: *testing, testing metrics, strategy, structural, white-box*

1. Pendahuluan

Pengujian merupakan aspek penting yang pada dasarnya program dieksekusi dengan masukan dan keluaran yang dinginkan (Sommerville, 2011). Suatu sumber kode (*source code*) perangkat lunak yang telah dihasilkan harus diuji (Myers, Badgett, & Sandler, 2012)coba untuk menemukan kesalahan sebelum diterima oleh pengguna (*end-user*) (Pressman, 2001). Sekumpulan kasus uji dirancang untuk diuji baik secara logika internal (Laird & Brennan, 2006) dan kebutuhan eksternal (Pressman, 2001). Hakikat uji coba perangkat lunak dapat dibagi menjadi 2 perspektif yaitu logika internal diuji dengan menggunakan teknik atau metode perancangan uji kasus *white-box* (Agarwal, Tayal, & Gupta, 2010), dan kebutuhan perangkat lunak diuji coba menggunakan metode perancangan uji kasus balck-box (Henard, Papadakis, & Harman, 2016). Terkait dengan penelitian yang dilakukan adalah menggunakan metode pengujian

Diterima 13 Desember 2017; Revisi 29 Januari 2018; Disetujui 8 Maret 2018

kotak putih (*white-box*). Uji coba dengan metode *white-box* mengacu pada teknik pengujian sistem dengan pengetahuan sistem internal (Beydeda, Gruhn, & Stachorski, 2014). Masalah utama industri perangkat lunak adalah tidak mampu untuk mengembangkan perangkat lunak bebas kesalahan (Singh, 2012).

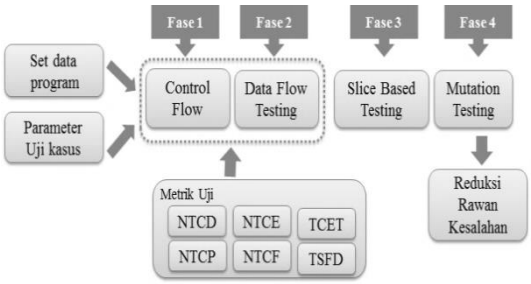
Penelitian mengenai metode white-box sudah banyak dilakukan. Hal ini terutama saat ini banyak peneliti melakukan kolaborasi (Henard, Papadakis, & Harman, 2016) antara metode *white-box* dengan metode *black-box* sebagai upaya memperbaiki kualitas (Beydeda, Gruhn, & Stachorski, 2014). Salah satu penelitian tentang kombinasi kedua metode tersebut adalah Sandra Kulkoj et al. ditahun 2013 yaitu uji coba dengan gabungan metode untuk seleksi dan menentuka skala prioritas untuk kasus uji. Tujuannya adalah menghasilkan kasus uji baru untuk fungsional (Kukolj, Marinkov'c, & Popovic, 2013). Hasil penelitiannya metode tersebut divalidasi dengan studi kasus pada perangkat android.

Penelitian pengembangan metode *white-box* pernah dilakukan oleh Arlinta Christy Barus dkk di tahun 2015. Penelitiannya membangun pengembangan prototipe yang akan diterapkan di metode white-box (Barus, Hutasoit, Siringoringo, & Siahaan, 2015). Hasil akhirnya adalah bahwa prototipe tersebut mampu menghitung semua cakupan p-*use* menggunakan perangkat lunak yang diuji. Walaupun demikian, metode *white-box* digunakan sebagai alat untuk menguji secara struktutral dari program yang diuji dan dieksekusi. Akibatnya adalah para penguji sistem atau program tidak mengetahui ukuran uji coba sebagai landasan kuat apakah perangkat lunak tersebut bisa dikatakan bebas dari kesalahan.

Uji coba struktural dianggap melebihi bahkan lebih baik dibandingkan dengan uji coba fungsional (*functional testing*) (Singh, 2012) . Hal ini dilakukan berasal dari kode program bukan kebutuhan perangkat lunak . Kode program diuji untuk memahami kinerja logika internal dan penerapan yang lain (Malhotra, 2016). Pada penelitian ini, untuk memperbaiki kinerja metode *white-box*, maka diusulkan metrik pengujian yang terintegrasi dengan komponen pengujian struktural, diantaranya adalah jumlah kasus uji yang dirancang, jumlah kasus uji yang dieksekusi, jumlah kasus uji yang lolos, jumlah kasus uji yang gagal, waktu eksekusi uji kasus, jumlah waktu ekseskusi, waktu yang digunakan untuk pengembangan kasus uji . Metrik pengujian merupakan bagian untuk mengambil kemajuan dan tingkatan uji coba pada perangkat lunak (Hambling & Goethem, 2013).

2. Metode Penelitian

Proses uji coba perangkat lunak pada studi kasus ini dilakukan melalui 4 tahap yaitu Alur kondisi (*control flow*), uji coba alur data (*data flow testing*), uji coba berbasis irisan (*slice based testing*) dan uji coba berbasis mutasi (*mutation testing*) *(*Malhotra*, Empirical Research In Software Engineering*, 2016). Data atau program yang digunakan selama proses uji coba menggunakan bahasa pemrograman PHP sedangkan parameter yang terlibat meliputi variabel, nilai yang dimiliki variabel serta efek perubahan apabila ada beberapa baris program yang dirubah.



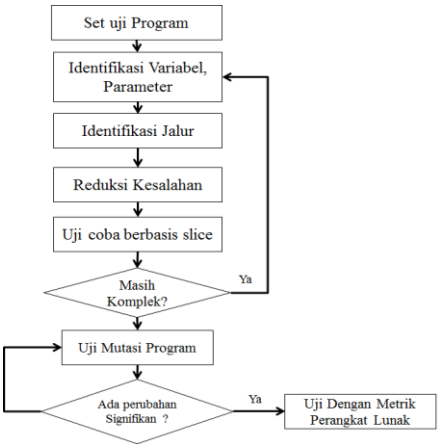
Gambar 1.
Kerangka Kerja Usulan

Proses uji coba yang mengeksekusi baris kode program akan memasuki tahap awal yaitu setiap baris program direpresantasikan melalui graf sehingga terbentuk jalur independen bersamaan parameter input untuk melihat respon keluaran yang dihasilkan (Ahmed, 2010). Jalur independen yang sudah terbentuk maka langkah selanjutnya dilakukan uji cakupan percabangan yang terbentuk pada perintah (*statement*) logika IF. Hasil eksekusi pernyataan logika ini akan diukur nilai kelompok kebenaran nilai sesuai dengan kondisi. Tahap kedua yaitu uji coba alur data meliputi mendefinisikan bentuk anomali. Tujuan tahap ini adalah untuk mereduksi bentuk keslahan yang terjadi selama uji coba. Uji coba tahap ini

hakikatnya melibatkan parameter variabel serta pembangkitan data uji kasus. Program yang berukuran besar memberikan kesulitan untuk menganalisa variasi variabel sehingga perlu dibuat program kecil (*subset*) untuk mempermudah identifikas yang dilakukan pada tahap uji coba berbasis slice. Tahap terakhir adalah uji coba mutasi. Tahap ini dilakukan dengan tujuan untuk melihat dampak perubahan yang dilakukan selama uji coba. Perubahan ini memiliki kemungkinan, perubahan kode program akan mengakibatkan dampak yang signifikan atau tidak. Uji coba mutasi ini mempunyai cakupan dimulai dari perubahan operator, operand, akses pengenal, perubahan pengenal statis, dan perubahan argumen. Perubahan yang terjadi selama uji coba akan akan diberikan skor sebagaimana persamaan dibawah ini :

$$skor\ mutasi = \frac{Jumlah\ mutan\ berpengaruh}{jumlah\ mutan}$$

Berikut ini adalah teknik dan langkah – langkah penelitian terkait uji coba berdasarkan metode yang diusulkan



Gambar 2. Alur Penelitian

Semua fase ini akan diterapkan dan diukur melalui metrik kualitas uji coba perangkat lunak. Metrik uji yang digunakan antara lain jumlah uji kasus yang dirancang (*Number of test case designed* – NTCD), jumlah uji kasus yang dieksekusi (*Number of test case execute* – NTCE), jumlah uji kasus yang lolos (*Number of test case passed* – NTCP), jumlah uji kasus gagal (*Number of test case failed* – NTCF), waktu eksekusi uji kasus (*Test case execution time* – TCET) dan waktu yang dipakai untuk pengembangan (*Time Spent for Development* - TSFD). Disamping itu ada penambahan untuk metrik uji sebagai uji pelacakan dan efisiensi

$$PTCP = \frac{(NOPT)}{TNOTE}\ x\ 100$$

$$FTCP = \frac{NOFT}{TNOTE}\ x\ 100$$

$$Executed\ test = \frac{NOTR}{TNOTBR}\ x\ 100$$

Keterangan :
PTCP = *Passed Test Case Percentage*
NOPT = *Number of passed test*
TNOTE = *Total Number of tests executed*
FTCP = *Failed test cases percentage*
NOFT = *Number of failed test*
NOTR = *Number of tests run*
TNOR = *Total number of tests to be run*

3. Hasil dan Pembahasan

Dalam upaya memperbaiki proses uji coba perangkat lunak pada metode white-box, ada metode yag diusulkan yaitu menggunakan proses metrik uji (*Testing Metric*). Walaupun dapat memperbaiki, proses metrik uji digunakan untuk mempercepat dan memonitoring selama uji coba agar dapat mengurangi kerawanan kesalahan.

3.1 Uji Coba Alur Kontrol

Pertama, analisa dilakukan dengan uji alur kontrol (*control flow testing*) yaitu dengan mengidentifikasi jalur independen dari setiap baris kode program. Adapun jalur independen adalah sebagai berikut :

1-11,12,18,24,33,34,35
1-11,12,17,23,30,32,34,35
1-11,12,18,25,26,28,31,32,34,35
1-11,12,28,25,27,28,31,32,34,35
1-11,12,28,25,27,29,30,32,34,35
1-11,13,16,21,22,23,30,32,34,35
1-11,14,19,21,,22,23,30,32,34,35
1-11,15,20,21,22,23,30,32,34,35

Berdasarkan jalur independen diatas dapat diketahui bahwa cakupan percabangan untuk mengguji nilai benar dan salah setiap percabangan dapat diidentifikasi terdapat 5 yaitu 1-11,12,18,25,27. Dalam menentukan cakupan jalur terdapat kriteria ujia coba sebagaimana pad atabel dibawah ini.

Cakupan jalur digunakan untuk menyederhanakan alur uji coba diantara baris program yang terlalu banyak.

Tabel 1. Tabel Uji Alur Kontrol

Item Field	Expected Ouput	Paths
Path1	FIRST NAME	1-11,20-22,26,27,31,35
Path2	ADDRESS	1-15,20-22,23,30-24
Path2	EMAIL	1-12,20,25,32
Path3	HANDPHONE	1-12,20,25,28
Path4	DATE OF BIRTH	1-12,20,25

3.2 Uji Coba Alur Data

Langkah selanjutnya adalah uji coba alur data yang didalamnya pendefinisian serta identifikasi anomali terdapat pada variabel. Ada 4 kondisi anomali diantaranya adalah variabel sudah didefinisikan tetapi tidak pernah digunakan, variabel digunakan tetapi belum didefinisikan, variabel didefinisikan sebanyak dua kali sebelum digunakan, variabel telah digunakan bahkan didefinisikan pertama. Setiap baris program (*statement*) direpresentasikan dengan simpul (*node*) sedangkan alur kontrol direpresentasikan dengan ruas (*edge*) yang didefinisikan menjadi dua bagian pertama definition use path (du) dan definition clear path (dc). *Definition use path* menjelaskan alur diantara 2 simpul (*node*) sedangkan *definition clear path* menjelaskan penggunaan jalur dengan simpul awal (*node*) dan simpul akhir (*final node*). Berikut ini *definition use path* (du) pada tabel dibawah ini:

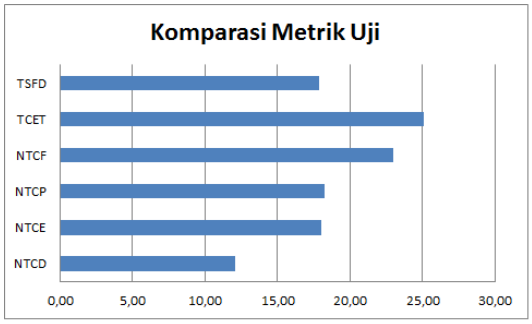
Tabel2. Definition use path (du)

VARIABLE	DEFINE AT NODE	USED AT NODE
VFIRST_NAME	6	11,12,13
VLAST_NAME	8	11,20,24
VAGE	10	12,16,20,21
VADDRESS	13	6,23,10
VEMAIL	4	33,26,14

Tabel 3. Definition Clear Path (DC)

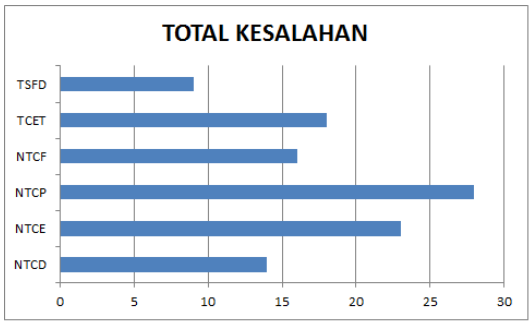
ALL DU-PATHS	DEFINITION CLEAR ?
7-12	Valid
1-7-12, 15-16	Valid
1-9-12,	Valid
1-9-12, 15,16	Valid
1-16-17,18	Valid
1-9,12,23	Valid
1-6,34,22	Valid
1-8,17,1	Valid
1-32,22,34	Valid
1-29,29,15	Valid

Tabel diatas menunjukan semua jalur independent sudah diuji melalui jalur 7-12, dan 1-7-12, 15-16 yang paling dominan, artinya dijalur tersebut merupakan jalur kritis.



Gambar 3. Komparasi Metrik Uji

Berdasarkan gambar diketahui selama proses uji coba, ditemukan bahwa 1 dari 6 parameter metrik uji yang memiliki nilai signifikan adalah waktu yang dihabiskan untuk uji kasus terutama disebabkan pada fase alur kontrol dan Uji alur data yang didalamnya terdapat logika. Pada alur data banyak yang harus diuji dimulai dari penggunaan percabangan dengan kondisi benar dan salah untuk setiap simpul (*node*).



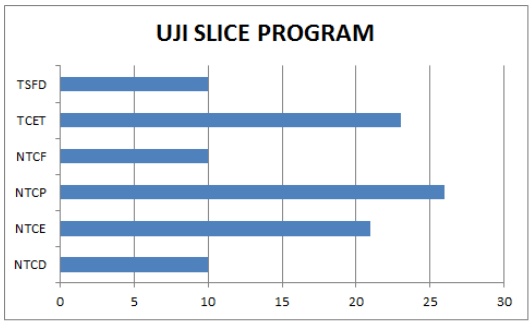
Gambar 4. Total Kesalahan dilaporkan

Setiap set program yang diuji akan terlihat hasil berupa pesan atau informasi kesalahan. Kesalahan ini muncul di setiap parameter metrik uji. Jumlah kesalahan terbesar pada NTCP (*Number of Test Case Passed*) Jumlah kasus uji yang diloloskan, artinya didalam kesalahan dalam set program bervariasi mulai dari identifikasi variabel, pergantian penggunaan variabel , dan baris program yang tidak dikenal.

3.3 Uji Coba Berbasis *Slice*

Dalam upaya mempercepat proses uji coba maka perlu dilakukan slicing pada program untuk menyederhanakan program

tidak semakin kompleks dan menimbulkan kerawanan kesalahan. Pada sebuah program sering ditemukan banyak variabel yang tersebar didalam baris kode program bahkan variabel sering berganti dengan variabel lain walaupun dengan tujuan yang sama dan mempunyai nilai item yang sama. Akibat jumlah variabel yang begitu banyak maka perlu dilakukan langkah slicing untuk mereduksinya. Uji coba berbasis slicing ini merupakan teknik untuk merubah program menjadi sederhana yang disebabkan jumlah variabel yang kompleks. Berikut adalah hasil slicing.



Gambar 5. Hasil Uji Berbasis Slice

Program yang diuji pada fase 1 dan 2 banyak ditemukan kesalahan yang disebabkan anomali variabel, perintah program yang tidak dikenal dan lain sebagainya. kondisi ini akan di lakukan uji coba dengan *slice program*. Namun demikian, dengan *slice program* dapat membantu menurunkan kesalahan walapun tidak signifikan.

Tabel 4. Hasil Slicing Program

Slice	Line Covered	Expected output
A(X, 12)	1-11, 33,34	FIRST NAME
B(X, 23)	1-11,22,27	ADDRESS
C(X, 28)	1-11,21,28	EMAIL
D(X, 17)	1-11,13,17	HANDPHONE

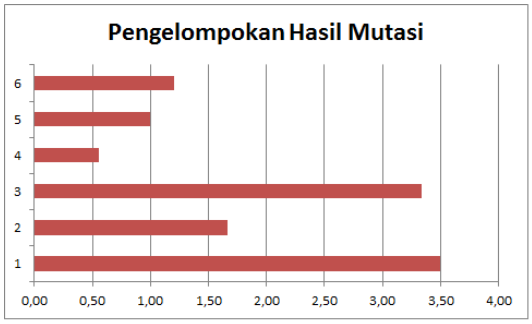
Hasil *slicing* diatas masih perlu dilakukan pemeriksaaan dan identifikasi terhadap perubahan yang dilakukan selama membuat program. Perubahan terhadap variabel merupakan suatu hal yang lazim, selain variabel yaitu perubahan operator yang digunakan, akan tetapi perubahan tersebut hakikatnya apakah memberikan dampak terhadap program itu sendiri apa tidak. Proses perubahan yang terdapat program ini disebut sebagai mutasi. Proses perubahan ini kemudian akan diberikan nilai (*score*) untuk menganalisa seberapa kuat pengaruh perubahannya diantara perubahan yang dilakukan selama proses uji coba.

3.4. Uji Coba Mutasi

Salah satu cara untuk mengurangi rawan kesalahan maka dilakukan dengan fase uji mutasi . Dalam penelitian ini, terdapat 30 mutan artinya terdapat 30 percobaan, namun yang ditampilkan hanya 10 mutan sebagai sample sebagaimana ditunjukan pada tabel dibawah ini.

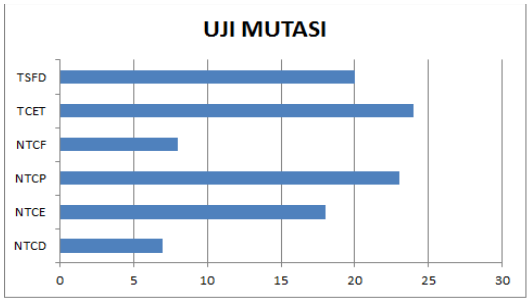
Tabel 5. Hasil Uji Mutasi

Mutan No	Number of mutan	Mutation Kiled	Mutation Score
M1	5	5	1,00
M2	4	3	0,75
M3	8	5	0,63
M4	2	8	4,00
M5	6	7	1,17
M6	8	9	1,13
M7	6	8	1,33
M8	6	4	0,67
M9	3	4	1,33
M10	8	5	0,63



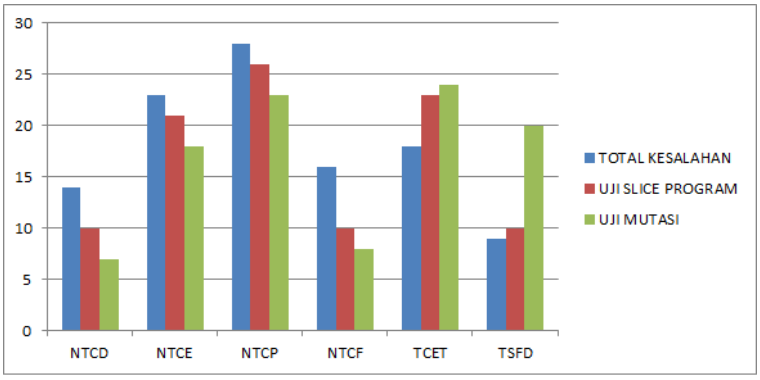
Gambar 6. Pengelompokan Mutasi

Berdasarkan pengelompokan mutasi pada gambar ditas bahwa kelompok pertama sangat signifikan. Mutasi ini terdapat pada perubahan tipe data pada variabel, penggunaan variabel yang konsisten, serta penghapusan terhadap variabel yang tidak pernah digunakan. Hampir 63% seluruhnya didominasi adanya penggunaan variabel yang terindikasi anomali.



Gambar 7 Hasil Uji Mutasi

Hasil mutasi sangat membantu untuk menguragi kesalahan terutama pada tahap 1 dan 2. Tahap ke 3 kesalahan semakin berkurang walaupun tidak signifikan seperti pada hasil uji mutasi pada gambar diatas.



Gambar 8. Hasil Akhir Pengujian

Berdasarkan gambar diatas menunjukan tingkat kesalahan mengalami peningkatan terutama pada tahap 1 dan tahap 2. Berdasarkan 4 fase yang diterapkan dengan bantuan metrik uji, tingkat kesalahan dapat berkurang hal ini dipengaruhi oleh Uji berbasis *slice* program dan uji mutasi sebagaimana tertera pada gambar dibawah diatas.

4. Kesimpulan

Uji coba perangkat lunak merupakan aktifitas yang menentukan kualitas perangkat lunak (*software*) sebelum diterima oleh pengguna akhir. Pada penelitian ini hakikatnya melakukan strategi uji coba pada level struktutal untuk memperbaiki pada metode *white-box*. Awalnya perpaduan metrik uji dengan uji coba struktural mengalami peningkatan penemuan kesalahan, namun tingkat kesalahan ini dapat diatasi dengan uji slice program dan uji mutasi, sehingga jumlah kesalahan dapat berkurang secara signifikan. Pengembangan selanjutnya tingkat kesalahan yang sudah diprediksi dapat ditentukan dengan pendekatan logika *fuzzy* untuk menentukan resiko terhadap program yang memiliki tingkat *slice* dan mutasi yang tinggi terutama pada program yang kompleks.

Referensi

Agarwal, B. B., Tayal, S. P., & Gupta, M. (2010). *Software Engineering and Testing.* Ontario: Jones and Bartlet Publishers.

Ahmed, A. (2010). *SoftwareTesting as Service.* Boca Raton: CRC Press.

Barus, A. C., Hutasoit, D. I., Siringoringo, J. H., & Siahaan, Y. A. (2015). White Box Testing Tool Prototype Development. *The 5th International Conference on Electrical Engineering and Informatics* (hal. 417-422). Bali: IEEE.

Beydeda, S., Gruhn, V., & Stachorski, M. (2014). A Graphical Class Representation for Integrated Black- and White-Box Testing. *IJSE*, 12-13.

Hambling, B., & Goethem, P. v. (2013). *User Acceptance Testing.* Swindon: British Computer Society.

Henard, C., Papadakis, M., & Harman, M. (2016). Comparing White-box and Black-box Test Prioritization. *IEEE International Conference on Software Engineering*, 523-534.

Kukolj, S., Marinkov'c, V., & Popovic, M. (2013). Selection and Prioritization of Test Cases by Combining White-Box and Black-Box Testing Methods. *Eastern European Regional Conference on the Engineering of Computer Based Systems* (hal. 153-156). Moscow: IEEE.

Laird, L. M., & Brennan, M. C. (2006). *Software Measurement and Estimation : A Practical Approach.* New Jersey: Wiley Interscience.

Malhotra, R. (2016). *Empirical Research In Software Engineering.* Boca Raton: CRC Press.

Malhotra, R., Pritam, N., & Singh, Y. (2014). On the Applicability of Evolutionary Computation for Software Defect Prediction. *International Conference on Advances in Computing,Communications and Informatics (ICACCI)* (hal. 2249-2257). New Delhi: IEEE.

Myers, G. J., Badgett, T., & Sandler, C. (2012). *The Art of Software Testing.* New Jersey: Willey.

Pressman, R. S. (2001). *Software Engineering.* New York: Mc Graw Hill.

Singh, Y. (2012). *Software Testing.* New York: Cambridge University Press.

Sommerville. (2011). *Software Engineering.* New York: Addison-Wesley.