9 786025 052507
ISBN : 978-602-50525-

PROSIDING
SEMINAR NASIONAL
INFORMATIKA DAN APLIKASINYA
SNIA - TAHUN 2017

Rabu, 27 September 2017
Hotel Mason Pine Kota Baru Parahyangan Bandung Barat

SNIA 17
SEMINAR NASIONAL INFORMATIKA DAN APLIKASINYA

TEMA

Computer Crime & Digital Evidence

JURUSAN INFORMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS JENDERAL ACHMAD YANI

Pengujian Perangkat Lunak

Berbasis Flow Graph, Cyclomatic Complexity dan Graph Matrix

Rudi Setiawan
Jurusan Sistem Informasi
Fakultas Industri Kreatif dan Telematika
Universitas Trilogi
Jl. TMP. Kalibata No.1 Pancoran, Jakarta Selatan
rudi@trilogi.ac.id

***Abstrak*—Untuk mengetahui kualitas dari perangkat lunak maka perlu dilakukan suatu pengujian. Pengujian perangkat lunak merupakan proses validasi dan verifikasi serta merupakan elemen kritis dalam proses pengembangan perangkat lunak. Terdapat beberapa metode dalam pengujiannya, di antaranya *blackbox* dan *whitebox*. Kajian penelitian ini mengarah pada metode *whitebox* dengan teknik pengujian berbasis *path* yang berupa *flow graph* yang mana menggambarkan jalur eksekusi dari perangkat lunak sedangkan *cyclomatic complexity* memberikan pengukuran kuantitatif terhadap kompleksitas logis suatu program dan *graph matrix* merupakan prosedur untuk mendapatkan *flow graph* dan menentukan serangkaian basis *path*. Hasil kajian penelitian menunjukkan metode *whitebox* dapat menggambarkan struktur program secara utuh dan dapat menunjukkan kesalahan yang ada pada kode program, akan tetapi butuh penguji yang paham akan kode program yang sedang diuji. Pada kode program dengan jumlah *path* < 5 cenderung struktur program tergolong sederhana, sedangkan kode program dengan *path* > 5 menunjukkan struktur program tergolong berada ditingkat yang lebih kompleks sedangkan pada kode program hasil pengujian dengan jumlah *Path* > 50 maka tergolong rumit dan sulit untuk dilakukan pengujian.**

Kata kunci*— *pengujian perangkat lunak; whitebox.

I. PENDAHULUAN

Pengujian perangkat lunak merupakan tahapan yang penting dalam pengembangan perangkat lunak guna menghasilkan perangkat lunak yang berkualitas baik secara perancangan maupun struktur kontrol pemrogramannya. Pengujian perangkat lunak juga merupakan bagian yang tidak dapat dipisahkan dari rekayasa perangkat lunak[2], sedangkan menurut[4] pengujian perangkat lunak merupakan proses validasi dan verifikasi dari suatu perangkat lunak. Pengujian perangkat lunak merupakan elemen kritis dari jaminan kualitas perangkat lunak dan bagian yang tidak dapat terpisahkan dalam siklus hidup pengembangan perangkat lunak[9].

Tahap pengujian perangkat lunak memerlukan biaya yang tidak sedikit bahkan tergolong membutuhkan biaya yang mahal. Tinjauan dari beberapa literatur mengenai pengujian perangkat lunak, 50% biaya pengembangan perangkat lunak dipergunakan untuk keperluan pengujian [2]. Untuk mengatasi besarnya biaya pengujian perangkat lunak, pengembangan perangkat lunak secara otomatis dapat dilakukan untuk mengurangi biaya[2][3][5].

Setidaknya terdapat beberapa teknik yang dapat diterapkan untuk melakukan pengujian perangkat lunak, di antaranya menggunakan metode *blackbox* dan *whitebox*. Pada metode *blackbox*, yang diuji adalah fungsionalitas dari perangkat lunak[3], sedangkan pada metode *whitebox* yang diuji adalah struktur program[8]. Masing-masing dari metode tersebut memiliki kelemahan seperti kelemahan yang dimiliki metode *blackbox* yang berfokus terhadap fungsionalitas dan masukan(*input*), apabila spesifikasi perangkat lunak yang dibuat kurang jelas maka akan sulit membuat dokumentasi hasil dari pengujian *blackbox*, sedangkan pada metode *whitebox* yang berfokus pada struktur program, apabila diujikan pada perangkat lunak yang jenisnya besar, metode *whitebox* testing membutuhkan banyak sumber daya untuk melakukannya[6]. Metode *whitebox* memerlukan penguji yang paham akan kode program yang sedang diuji[7][8]. Dibandingkan pengujian fungsional, pengujian struktural memegang peranan penting karena lebih efektif dalam mengurangi biaya pada proses pengujian perangkat lunak.

II. KAJIAN PUSTAKA

A. Flow Graph

Notasi yang digunakan untuk menggambarkan jalur eksekusi adalah notasi diagram alir, yang menggunakan simbol lingkaran atau dapat disebut sebagai *node* dan tanda anak panah atau dapat disebut sebagai *edge*. Notasi ini menggambarkan aliran kontrol logika yang digunakan dalam suatu bahasa pemrograman seperti yang dituliskan pada Tabel 1.

B. Cyclomatic Complexity

Merupakan pengujian *perangkat lunak* yang memberikan pengukuran terhadap kuantitatif dan kompleksitas logika program untuk mencari jumlah *path* dalam satu *flow graph*. Pengukuran dilakukan berdasarkan teori *graph* dan perhitungannya berdasarkan dari struktur program yang terlihat pada *flow graph*. *Cyclomatic complexity* diekspresikan dengan persamaan 1.

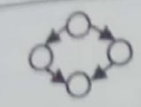
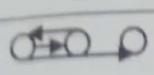
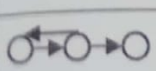
$$\textit{Cyclomatic complexity}\ V(G) = E - N + 2 \qquad (1)$$

Di mana :

E = jumlah *edge* (anak panah) pada *flow graph*
N = jumlah *node* (titik) pada *flow graph*

ISBN: 978-602-50525-0-7

TABEL 1. NOTASI FLOW GRAPH

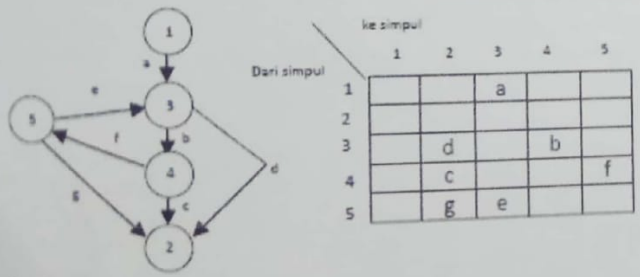
Simbol	Notasi
	Skema Sequence
	Skema Percabangan menggunakan If
	Skema perulangan menggunakan While …. Do……
	Skema perulangan menggunakan Repeat …. Until ……
	Skema Case ……. of

C. Graph Matrix

Merupakan matriks berbentuk segi empat sama sisi, di mana jumlah baris dan kolom sama dengan jumlah *node* serta isi data adalah keberadaan penghubung antar *node*. *Graph matrix* merupakan *software* yang dikembangkan untuk membantu pengujian perangkat lunak berbasis *path* atau struktur data.

Contoh sederhana dari *flow graph* dan *graph matrix* digambarkan pada Gambar 1 dan Tabel 2.

TABEL 2. GRAPH MATRIX



Gambar 1. *Flow Graph*

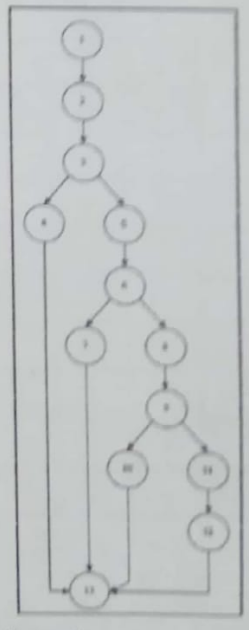
III. PEMBAHASAN

Pada penelitian ini, dilakukan contoh pengujian kode program PHP yang dituliskan pada Gambar 2.

Gambar 2. Kode Program PHP

A. Flow Graph

Flow graph menggambarkan struktur kontrol dari kode program yang diuji. Lingkaran pada *flow graph* merepresentasikan suatu *statemen* prosedural, sedangkan anak panah pada gambar merepresentasikan aliran kontrol. *Flow graph* dari contoh kode program yang diuji pada Gambar 2 digambarkan pada Gambar 3.



Gambar 3. *Flow Graph* hasil pengujian

B. Cyclomatic complexity

Cyclomatic complexity memberikan nilai pengukuran kuantitatif terhadap kompleksitas logis suatu program.

ISBN: 978-602-50525-0-7

Cyclomatic complexity yang dihasilkan dari proses pengujian kode program pada Gambar 2 menjadi persamaan 2:

$$V(G) = 15 \text{ edge} - 13 \text{ node} + 2 = 4 \quad (2)$$

Cyclomatic complexity yang didapatkan berdasarkan hasil perhitungan sebanyak 4 *path*. Tabel 3 menunjukkan *path* yang didapat dari urutan *flow graph*.

TABEL 3. *PATH* URUTAN PROGRAM

Path	*Node*
Path 1	1, 2, 3, 4, 13
Path 2	1, 2, 3, 5, 6, 7, 13
Path 3	1, 2, 3, 5, 6, 8, 9, 10, 13
Path 4	1, 2, 3, 5, 6, 8, 9, 11, 12, 13

C. *Graph Matrix*

Graph matrix dibuat berdasarkan jumlah *node* dan *edge* dari *flow graph*. Masing-masing baris dan kolom sesuai dengan jumlah *node* yang diidentifikasikan pada *flow graph*, dan nilai dari matriks sesuai dengan *edge* yang berada di antara *node*. hasil pengujian pada kode program yang ada di Gambar 2 dituliskan pada Tabel 4.

TABEL 4. GRAPH MATRIX HASIL PENGUJIAN

	1	2	3	4	5	6	7	8	9	10	11	12	13
1		1											
2			1										
3				1	1								
4													1
5						1							
6							1	1					
7													1
8									1				
9										1	1		
10													1
11												1	
12													1
13													

IV. KESIMPULAN

Metode *whitebox* dengan teknik pengujian berbasis *path* yang berupa *flow graph* yang mana menggambarkan jalur eksekusi dari perangkat lunak, sedangkan *cyclomatic complexity* memberikan pengukuran kuantitatif terhadap kompleksitas logis suatu program dan *graph matrix* merupakan prosedur untuk mendapatkan *flow graph* dan menentukan serangkaian basis *path*.

Pada kode program dengan jumlah *Path* < 5 menunjukkan struktur program tergolong sederhana, sedangkan kode program dengan *path* > 5 menunjukkan struktur program tergolong berada ditingkat yang lebih kompleks sedangkan pada kode program hasil pengujian dengan jumlah *path* > 50 maka tergolong rumit dan sulit untuk dilakukan pengujian.

Daftar Pustaka

[1] A.M. Alakeel, "Using Fuzzy Logic Techniques for Assertion-Based Software Testing Metrics". *The Scientific World Journal*. Vol.2015, Article ID 629430.

[2] S. Anand, E.K. Bure, T. Yuesh, J. Clark. The Journal of Systems and Software, "An orchestrated survey of methodologies for automated software test case generation", 2013, Journal of Systems and Software, p.1978-2001.

[3] J. Ferrer, P.M Kruse, F. Chicano, E. Alba. "Search Based Algorithms for Test Sequence Generation in Functional Testing". *Informatioan and Software Tehcnology*, 2015, Vol.58, p.419.432.

[4] N. Garimella, P. Khrisna, S.D Kumar. "Software Testing Techniques and Documentation". *International journal of application or innovation in engineering and management*, 2013. Vol.2.

[5] I. Hermandi, C. Lokan, and R. Sarker. "Dynamic Stopping Criteria for Search Based Test Data Generation for Path Testing", Journal Information and software technology, 2014, Vol.56 p395-407.

[6] Nidhra, Srinivas and Dondetti, Jagruthi. "Blackbox and Whitebox Testing Techniques" – A Literature Review, *International Journal of Embedded Systems and Applications* (IJESA), 2012, Vol.2, No.2.

[7] C. Mao. "Generating Test Data for Software Structural Testing Based On Particle Swarm Optimizasion", 2014, *Arabian Journal for science and engineering*. Vol.39, p.4593-4607.

[8] C. Mao, L. Xiao, X. Yu, J. Chen. "Adapting Ant Colony Optimization to Generate Test Data for Software Structural Testing", 2015, *Swarm and Evolutionary Computation*, Vol.20, p.23-36.

[9] M. Shi. "Software Functional Testing from The Perspective of Business Practice", 2010, *Computer and information science*. Vol.3, No.4.

ISBN: 978-602-50525-0-7