Instrumentasi *Source Code* Secara Otomatis untuk *Basis Path Testing*

Raden Asri Ramadhina Fitriani(G64154007)*, Irman Hermadi

Abstrak/*Abstract*

Pengujian adalah serangkaian proses yang dirancang untuk memastikan sebuah perangkat lunak melakukan apa yang seharusnya dilakukan dan bertujuan untuk menemukan kesalahan pada perangkat lunak. *Basis Path testing* merupakan salah satu metode pengujian struktural yang menggunakan *source code* dari program untuk menemukan semua jalur yang mungkin dapat dilalui program dan dapat digunakan untuk merancang data uji. Untuk menguji perangkat lunak yang kompleks secara keseluruhan akan memakan waktu yang lama dan membutuhkan sumber daya manusia yang banyak. Idealnya, pengujian dilakukan untuk semua kemungkinan dari perangkat lunak. Kumar dan Mishra (2016) mengatakan bahwa pengujian perangkat lunak menggunakan hampir 60% dari total biaya pengembangan perangkat lunak. Sehingga mengotomasi bagian dari pengujian akan membuat proses ini menjadi lebih cepat dan mengurangi kerawanan akan kesalahan. Pada penelitian ini, akan dibangun sebuah sistem untuk membangkitkan kemungkinan jalur-jalur dari sebuah program yang dapat dijadikan dasar untuk membangkitkan data uji agar data uji yang digunakan untuk pengujian dapat mewakili semua kemungkinan. Untuk memonitor jalur yang dilalui program ketika dijalankan dengan masukan data uji tertentu, maka sistem ini juga akan melakukan instrumentasi *source code* program secara otomatis. Program yang akan diuji dalam penelitian ini adalah program yang dibangun dengan menggunakan bahasa Matlab. Dalam pengembangannya, aplikasi ini akan dibangun dengan menggunakan bahasa pemrograman Java dan library *Graphviz* untuk memvisualisasikan *Control Flow Graph*.

Kata Kunci

Basis Path Testing; *Control Flow Graph*; Instrumentasi

***Alamat Email**: radenasrirf@gmail.com

PENDAHULUAN

Latar Belakang

Pengujian adalah serangkaian proses yang dirancang untuk memastikan sebuah perangkat lunak melakukan apa yang seharusnya dilakukan. Proses ini bertujuan untuk menemukan kesalahan pada perangkat lunak. Saat pengujian, bisa saja tidak ditemukan kesalahan pada Hasil pengujian. Hal ini dapat terjadi karena perangkat lunak yang sudah berkualitas tinggi atau karena proses pengujiannya berkualitas rendah. (Myers *et al.* 2012)

Teknik pengujian secara umum dibagi menjadi 2 kategori diantaranya *black box testing* dan *white box testing*. *Black box testing* bertujuan untuk memeriksa fungsional dari perangkat lunak apakah output sudah sesuai dengan yang ditentukan. Sedangkan *White box testing* atau biasa disebut dengan pengujian struktural merupakan pemeriksaan struktur dan alur logika suatu proses. *Basis Path testing* merupakan salah satu metode pengujian struktural yang menggunakan *source code* dari program untuk menemukan semua jalur yang mungkin dapat dilalui program dan dapat digunakan untuk merancang data uji. Metode ini memastikan semua kemungkinan jalur dijalankan setidaknya satu kali (Basu 2015). Untuk melakukan monitoring jalur mana yang diambil oleh sebuah masukan pada saat eksekusi program, maka diperlukan penanda yang dapat memberikan informasi cabang mana yang dilalui. Proses menyisipkan tanda tersebut disebut instrumentasi. Biasanya tanda tersebut disisipkan tepat sebelum sebuah percabangan (Tikir dan Hollingsworth 2011).

Idealnya, pengujian dilakukan untuk semua kemungkinan dari perangkat lunak. Tetapi untuk menguji perangkat lunak yang kompleks secara keseluruhan akan memakan waktu yang lama dan membutuhkan sumber daya manusia yang banyak. Kumar dan Mishra (2016) mengatakan bahwa pengujian perangkat lunak menggunakan hampir 60% dari total biaya pengembangan perangkat lunak. Jika proses pengujian perangkat lunak dapat dilakukan secara otomatis, maka hal ini dapat mengurangi biaya pengembangan secara signifikan.

Hermadi (2015) melakukan penelitian untuk membangkitkan data uji untuk *path testing* menggunakan algoritma genetika. Dalam penelitian tersebut, Hermadi membangkitkan *Control flow Graph* (CFG) dan instrumentasi masih secara manual sehingga membutuhkan banyak waktu dan rawan akan kesalahan ketika program sudah semakin besar. Sehingga mengotomasi hal tersebut dapat membuat *path testing* menjadi lebih cepat dan mengurangi kerawanan akan kesalahan.

Pada penelitian ini, akan dibangun sebuah perangkat lunak untuk membangkitkan kemungkinan jalur dari

sebuah program. Jalur-jalur ini dapat dijadikan dasar untuk membangkitkan data uji agar data uji yang digunakan untuk pengujian dapat mewakili semua kemungkinan. Untuk memonitor jalur mana yang dilalui ketika diberikan masukan data uji, maka sistem ini juga akan melakukan penyisipan tag-tag sebagai instrumentasi ke dalam *source code* secara otomatis.

Perumusan Masalah

Berdasarkan latar belakang di atas dapat dirumuskan masalahnya adalah bagaimana membangun sebuah aplikasi untuk melakukan instrumentasi secara otomatis untuk pengujian jalur dan *re-engineering* perangkat lunak.

Tujuan

Penelitian ini bertujuan untuk membangun sebuah aplikasi yang dapat digunakan untuk membangkitkan CFG dan melakukan instrumentasi secara otomatis.

Ruang Lingkup

Bahasa pemrograman yang diakomodasi adalah Matlab dan model diagram yang dibangkitkan adalah CFG.

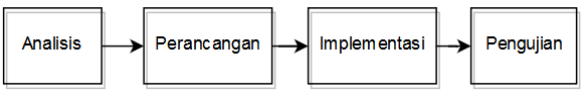
Manfaat

Hasil penelitian diharapkan dapat membantu pengembang dan penguji aplikasi untuk:

1. Menyisipkan tag-tag sebagai instrumentasi program ke dalam *source code* secara otomatis sehingga proses tersebut dapat dilakukan dengan lebih cepat.
2. Membangkitkan jalur-jalur dasar yang dapat digunakan sebagai dasar untuk pembangkitan data uji.
3. Membangkitkan diagram CFG yang dapat memudahkan pengembang dalam memahami struktur dan alur dari suatu program yang dapat dimanfaatkan ketika akan melakukan *re-engineering* perangkat lunak.

METODE PENELITIAN

Penelitian yang dilakukan terbagi menjadi beberapa tahapan proses. Gambar 1 menunjukan tahapan proses tersebut.



Gambar 1. Metode Penelitian

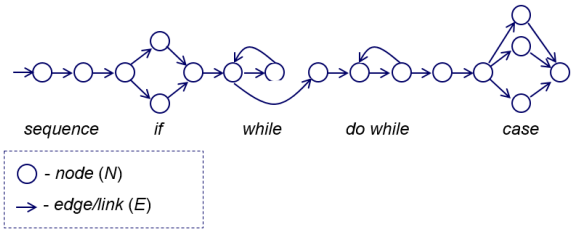
Analisis

Pada tahap ini dimulai dari membaca literatur terkait dan mengumpulkan beberapa contoh program yang akan digunakan dalam penelitian. Contoh program yang digunakan dalam penelitian ini didapatkan dari penelitian yang dilakukan oleh Hermadi (2015).

Basis path testing merupakan salah satu metode pengujian struktural yang menggunakan *source code* dari program untuk menemukan semua jalur yang mungkin dapat dilalui program dan dapat digunakan untuk merancang data uji. Metode ini memastikan semua kemungkinan jalur dijalankan setidaknya satu kali (Basu 2015). Metode ini terbagi menjadi 4 tahapan, yaitu:

1. Menggambarkan jalur dalam bentuk *Control Flow Graph* (CFG)
2. Menghitung *cyclomatic complexity*
3. Memilih satu set jalur dasar
4. Membangkitkan data uji untuk setiap jalur dasar

Control Flow Graph (CFG) adalah graph berarah yang merepresentasikan aliran dari sebuah program. Setiap CFG terdiri dari *nodes* dan *edges*. *Nodes* merepresentasikan *statement* atau *expressions*. Sedangkan *edges* merepresentasikan transfer kontrol antar *nodes* (Watson dan McCabe 1996). Notasi dari CFG dapat dilihat pada Gambar 2.



Gambar 2. Notasi Control Flow Graph (CFG)

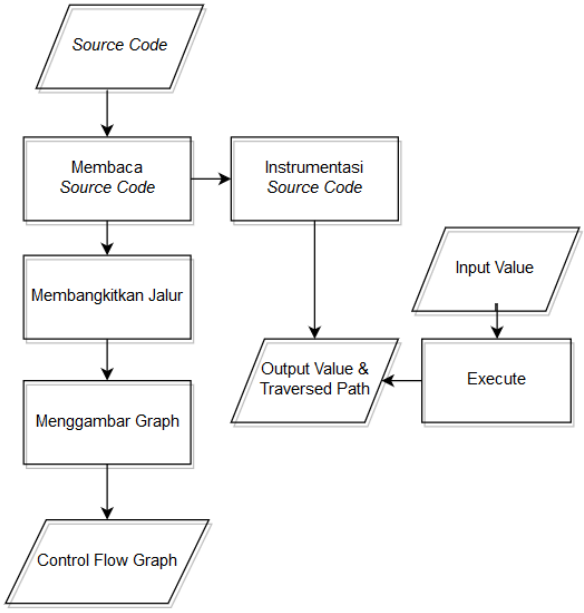
Cyclomatic complexity merupakan suatu sistem pengukuran yang ditemukan oleh Watson dan McCabe untuk menentukan banyaknya *independent path* dan menunjukan tingkat kompleksitas dari suatu program. *Independent path* adalah jalur yang melintas dalam program yang sekurang-kurangnya terdapat kondisi baru. Perhitungan *Cyclomatic Complexity* dapat dilihat pada persamaan berikut:

$$V(G) = E - N + 2$$

Dimana, E menunjukkan jumlah *edges* dan N menunjukkan jumlah *nodes*.

Design

Pada tahap ini ditentukan bagaimana perangkat lunak akan dibangun. Ilustrasi arsitektur sistem dapat dilihat pada Gambar 3.



Gambar 3. Arsitektur Sistem

Source code akan dibaca sebagai inputan, lalu akan dibaca baris perbaris dengan mengabaikan *whitespace*. Pada tahap awal akan dilakukan akan dilakukan pembangkitan semua jalur yang mungkin dari program dan instrumentasi *source code*. Setelah jalur terbentuk, jalur akan divisualisikan dalam bentuk CFG.

Instrumentasi merupakan sebuah proses menyisipkan sebuah penanda (tag) di awal atau di akhir setiap blok kode seperti awal setiap fungsi, sebelum atau sesudah kondisi terpenuhi atau tidak. Dalam pengujian *path testing*, penanda ini dapat digunakan untuk memonitor jalur yang dilalui program ketika dijalankan dengan masukan data uji tertentu (Arkeman *et al.* 2014)

Implementasi

Tahapan ini adalah melakukan implementasi dari tahap sebelumnya ke dalam bentuk aplikasi web. Aplikasi ini akan dibangun dengan menggunakan bahasa pemrograman Java dan menggunakan IDE Eclipse.

Setelah jalur terbentuk, CFG akan divisualisasikan dengan menggunakan library Graphviz. Graphviz merupakan perangkat lunak *open source* untuk visualisasi grafik.

Testing

Tahapan ini adalah melakukan evaluasi dari tahapan implementasi. Evaluasi dilakukan dengan membandingkan hasil yang dikeluarkan oleh sistem dengan pembangkitan secara manual dari segi waktu eksekusi.

Jadwal Kegiatan

Penelitian ini akan dilakukan selama 5 bulan dengan rincian kegiatan seperti tercantum pada Tabel 1.

DAFTAR PUSTAKA

Arkeman, Y, Herdiyeni, Y, Hermadi, I, dan Laxmi, G F. 2014. *Algoritma Genetika Tujuan Jamak (Multi-Objective Genetic Algorithm*. IPB Press.

Basu, A. 2015. *Software Quality Assurance, Testing and Metrics*. PHI Learning Privat Limited. [Internet]. [Diunduh tanggal 14/8/2017]. Dapat diunduh dari: `https://books.google.co.id/books`.

Hermadi, I. 2015. "Path Testing using Genetic Algorithm". Disertasi. University of New South Wales.

Khan, M E. 2011. "Different Approaches to White Box Testing Technique for Finding Errors" dalam: *International Journal of Software Engineering and Its Applications* 5, p. 3. [Internet]. [Diunduh tanggal 21/8/2017]. Dapat diunduh dari: `http://www.sersc.org/journals/IJSEIA/vol5_no3_2011/1.pdf`.

Kumar, D dan Mishra, K K. 2016. "The Impacts of Test Automation on Software's Cost, Quality and Time to Market" dalam: *Procedia Computer Science* 79, pp. 8–15. [Internet]. [Diunduh tanggal 20/8/2017]. Dapat diunduh dari: `http://www.sciencedirect.com/science/article/pii/S1877050916001277`.

Myers, G J, Sandler, C, dan Badgett, T. 2012. *The Art of Software Testing*. John Willey dan Sons, Inc, Hoboken, New York. [Internet]. [Diunduh tanggal 14/8/2017]. Dapat diunduh dari: `https://books.google.co.id/books`.

Tikir, M M dan Hollingsworth, J K. 2011. "Efficient Instrumentation for Code Coverage Testing" dalam: *International Journal of Software Engineering and Its Applications*. [Internet]. [Diunduh tanggal 21/8/2017]. Dapat diunduh dari: `https://www.researchgate.net/publication/2835608_Efficient_Instrumentation_for_Code_Coverage_Testing`.

Tabel 1. Rencana Jadwal Penelitian

Kegiatan	Jul		Agu				Sep				Okt				Nov				Des	
	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2
Analisis	■	■	■																	
Perancangan		■	■	■																
Penyusunan Proposal Skripsi				■	■	■														
Kolokium							■													
Perbaikan proposal							■	■												
Implementasi								■	■	■	■	■								
Pengujian Sistem												■								
Penulisan draft skripsi												■	■	■	■	■				
Seminar																	■			
Perbaikan makalah dan sistem																	■	■		
Sidang skripsi																			■	
Perbaikan laporan penelitian																				■

Watson, A H dan McCabe, T J. 1996. "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric)" dalam: *NIST Special Publication*. [Internet]. [Diunduh tanggal 14/8/2017]. Dapat diunduh dari: `http://www.mccabe.com/pdf/mccabe-nist235r.pdf`.