Instrumentasi Source Code Secara Otomatis untuk*Basis* Path Testing**Abstrak**Pengujian adalah serangkaian proses yang dirancang untuk memastikan sebuah perangkat lunak melakukan apa yang seharusnya dilakukan dan bertujuan untuk menemukan kesalahan pada perangkat lunak. Path testing merupakan salah satu metode pengujian struktural yang menggunakan source code dari program untuk menemukan semua jalur yang mungkin dapat dilalui program dan memastikan  
semua jalur dijalankan setidaknya satu kali. Untuk menguji perangkat lunak yang kompleks secara keseluruhan akan memakan waktu yang  
lama dan membutuhkan sumber daya manusia yang banyak. 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 memastikan semua jalur dijalankan setidaknya 1 kali ketika diberikan masukan data  
uji, 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 bahasan C. Dalam pengembangannya, aplikasi ini akan dibangun dengan  
menggunakan bahasa pemrogramana Java dan library Graphvis untuk memvisualisasikan Control Flow Graph.

Kata Kunci

Path Testing; Control Flow Graph, Instrumentasi

PENDAHULUANLatar 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. Maka dibutuhkan metode pengujian yang tepat untuk memastikan perangkat lunak siap untuk digunakan (Watson dan McCabe 1996).

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. Sedangkan White box testing atau biasa disebut dengan pengujian struktural merupakan pemeriksaan struktur dan alur logika suatu proses. Path testing merupakan salah satu metode pengujian struktural yang menggunakan source code dari program untuk menemukan semua jalur yang mungkin dapat dilalui program dan memastikan semua 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 kirakira 30%-50% dari total sumber daya, 30% dari total upaya yang dilakukan, dan 50%-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.

\citeauthor{GAIRH2008} (\cite\*{GAIRH2008}) melakukan penelitian untuk membangkitkan beberapa data uji yang dapat mencakup beberapa target jalur dalam sekali proses menggunakan algoritma genetika. Dalam penelitian tersebut, \citeauthor{GAIRH2008} membangkitkan CFG masih secara manual sehingga membutuhkan banyak waktu untuk dilakukan.

Sehingga dibutuhkan otomatisasi untuk melakukan pengujian perangkat lunak. Pada penelitian ini, akan  
dibangun sebuah sistem 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 memastikan semua jalur dijalankan setidaknya 1 kali ketika diberikan masukan data uji, maka sistem ini juga akan melakukan instrumentasi source code program secara otomatis.  
Perumusan Masalah

Berdasarkan latar belakang di atas dapat dirumuskan masalah adalah bagaimana membangun sebuah aplikasi untuk melakukan instrumentasi secara otomatis untuk logika pengujian jalur dan reengineering perangkat lunak.

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

Ruang Lingkup

Penelitian ini hanya melakukan pembangkitan CFG dan instrumentasi dari bahasa pemrograman C.

ManfaatHasil penelitian diharapkan dapat membantu pengembang dan penguji aplikasi untuk:

1. Melacak jalur yang dilalui ketika suatu program dieksekusi.

2. Membangkitkan jalur-jalur yang dapat digunakan sebagai dasar untuk pembangkitan data uji.

3. Memudahkan pengembang dalam memahami struktur dari suatu program yang dapat dimanfaatkan ketika akan melakukan reengineering perangkat lunak.

METODE PENELITIAN

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

Analisis dan Perancangan

Pada tahap ini ditentukan bagaimana perangkat lunak akan dibangun. Mulai dari membaca literatur terkait,  
mengumpulkan sampel source code bahasa C untuk percobaan, mendesign struktur dari perangkat lunak yang akan dibangun. Sampel source code yang digunakan dalam penelitian ini diambil dari penelitian Watson (1996). Ilustrasi gambaran umum sistem dapat dilihat pada Gambar 2

Gambar 2. Gambaran Umum Sistem

Inputan yang akan dibaca oleh sistem adalah source code C. Proses dimulai dari membangkitkan Control  
Flow Graph (CFG). CFG adalah graph berarah yang merepresentasikan aliran dari sebuah program. Setiap  
CFG terdiri dari nodes dan edges. Nodes merepresentasikan statement atau expressions. Sedangkan edgesmerepresentasikan transfer kontrol antar nodes (Watson dan McCabe 1996). Notasi dari CFG dapat dilihat pada Gambar 3. Setelah CFG terbentuk, langkah selanjutnya adalah melakukan pembangkitan semua jalur logika yang mungkin dari program.

Ketika terdapat suatu fungsi yang memiliki \textit{loops} (\textit{for, while, do-while}) maka jumlah jalur akan menjadi tak terhingga. Untuk mengurangi jumlah jalur yang akan diuji, dapat digunakan ukuran yang disebut *Cyclomatic Complexity.*  *Cyclomatic Complexity* merupakan suatu sistem pengukuran yang ditemukan oleh \citeauthor{MCCABE} untuk menentukan banyaknya *independent path*. *Independent path* adalah jalur yang melintas dalam program yang sekurang-kurangnya terdapat kondisi baru. Perhitungan *Cyclomatic Complexity* dapat dilihat pada persamaan berikut:

Instrumentasi adalah 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 (Ahmed dan Hermadi 2008)

Gambar 3. Notasi Control Flow Graph (CFG)

ImplementasiTahapan 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. Visualisasi CFG menggunakan library Graphviz. Graphviz adalah perangkat lunak open source untuk visualisasi grafik. Visualisasi adalah cara untuk mewakili informasi structural sebagai diagram atau jaringan.  
Evaluasi Hasil

Tahapan ini adalah melakukan evaluasi dari tahapan implementasi. Sistem yang telah dibangun akan dilakukan pengujian dengan menggunakan black box testing. Evaluasi dilakukan dengan membandingkan hasil yang dikeluarkan oleh sistem dengan pembangkitan secara manual.

DAFTAR PUSTAKA

Ahmed, M A dan Hermadi, I. 2008. “GA-based multiple paths test data generator” dalam: Computers Operations Research 35, pp. 3107–3124. [Internet]. [Diunduh tanggal 4/7/2017 ]. Dapat diunduh dari: http:  
/ / www . sciencedirect . com / science /article/pii/S0305054807000251.

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

Khan, M E. 2011. “Different Approaches to White Box Testing Technique for Finding Errors” dalam: International Journal of Software Engineering andIts 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 ofSoftware 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 andIts Applications. [Internet]. [Diunduh tanggal 21/8/2017 ]. Dapat diunduh dari: https : / / www .researchgate.net/publication/2835608\_  
Efficient\_Instrumentation\_for\_Code\_Coverage\_Testing.  
Watson, A H. 1996. “Structured Testing: Analysis and Extenstions”. PhD thesis. Pricenton University.  
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.