Instrumentasi Source Code Otomatis Untuk Logic Path Testing

ABSTRACT

PENDAHULUAN

## Latar Belakang

Pengujian perangkat lunak merupakan suatu proses mengeksekusi sebuah perangkat lunak dan membandingkan perilaku apakah sesuai dengan perilaku yang diinginkan. Proses ini bertujuan untuk menemukan kesalahan pada perangkat lunak. Pengujian perangkat lunak juga bertujuan untuk membangun kepercayaan dalam pengoprasian perangkat lunak ketika tidak ditemukan kesalahan pada hasil pengujian. 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. (Mc Cabe)

Pengujian perangkat lunak merupakan suatu proses mengeksekusi sebuah perangkat lunak dan membandingkan apakah perilaku suatu perangkat lunak sesuai dengan perilaku yang diinginkan. Proses ini bertujuan untuk menemukan kesalahan pada perangkat lunak. Pengujian perangkat lunak juga bertujuan untuk membangun kepercayaan dalam pengoprasian perangkat lunak ketika tidak ditemukan kesalahan pada hasil pengujian. 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. (\cite{MCCABE})

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 \textit{Cyclomatic Complexity}. \textit{Cyclomatic Complexity} merupakan suatu sistem pengukuran yang ditemukan oleh \citeauthor{MCCABE} untuk menentukan banyaknya \textit{independent path}. \textit{Independent path} adalah jalur yang melintas dalam program yang sekurang-kurangnya terdapat kondisi baru. Perhitungan \textit{Cyclomatic Complexity} dapat dilihat pada persamaan berikut:

https://books.google.co.id/books?id=neWaoJKSkvgC&pg=PT13&dq=why++software+must+testing&hl=en&sa=X&redir\_esc=y#v=onepage&q=why software must testing&f=false

Pengujian merupakan proses mengeksekusi sebuah perangkat lunak yang bertujuan untuk menemukan kesalahan.

https://books.google.co.id/books?id=GjyEFPkMCwcC

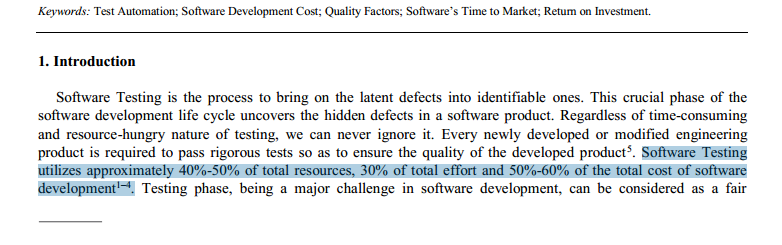
Pengujian adalah serangkaian proses yang dirancang untuk memastikan sebuah perangkat lunak melakukan apa yang seharusnya dilakukan dan untuk menemukan kesalahan pada sebuah program sebelum digunakan. Idealnya, pengujian dilakukan untuk semua kemungkinan dari perangkat lunak. Tetapi untuk menguji perangkat lunak yang kompleks secara keseluruhan akan memakan waktu yang lama dan membutuhkan sumber daya manusia yang banyak. (\cite{GLENFORD2012})

Pengujian dimaksudkan untuk menunjukkan bahwa sebuah program melakukan apa yang dimaksudkan untuk dilakukan dan untuk menemukan cacat program sebelum digunakan. Saat Anda mengetes perangkat lunak, Anda menjalankan program dengan menggunakan data buatan. Anda memeriksa hasil uji coba untuk kesalahan, anomali, atau informasi tentang atribut non-fungsional program.

\citeauthor{KUMAR20168} (\cite\*{KUMAR20168}) mengatakan bahwa pengujian perangkat lunak menggunakan kira-kira 30\%-50\% dari total sumber daya, 30\% dari total upaya yang dilakukan, dan 50\%-60\% dari total biaya pengembangan perangkat lunak.

Proses pengujian menggunakan 25% sampai dengan 50% dari total anggaran dari banyak proyek pengembangan perangkat lunak.

https://books.google.co.id/books?id=noEyxwGQ6SkC&printsec=frontcover&dq=automated+testing&hl=en&sa=X&redir\_esc=y#v=onepage&q=automated%20testing&f=false



<http://www.sciencedirect.com/science/article/pii/S1877050916001277>

Secara umum pengujian dibagi menjadi 2 kategori diantaranya blackbox testing dan whitebox Testing. White box testing atau biasa disebut dengan pengujian struktural merupakan pemeriksaan struktur dan alur logika suatu proses. Struktur program dalam pengujian ini direpresentasikan oleh Control Flow Graph (CFG).

https://books.google.co.id/books?id=aNTiCQAAQBAJ

Struktural testing memerlukan ekseskusi satu set jalur uji dalam program yang akan diuji. Hal kritis yang dapat mempengaruhi efiensi dan biaya pengujian adalah dalam menghasilkan rangkaian jalur uji.

https://pdfs.semanticscholar.org/ce99/da35d54110c900058302b882a3ffafc232e0.pdf

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.206.4616&rep=rep1&type=pdf

Dalam kategori pengujian kotak putih, salah satu kelompok kriterianya adalah coverage testing.

Untuk menciptakan

\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.

Instrumentasi manual dan fungsi fitness generasi untuk tes relatif tidak sulit. Namun, kedua tugas ini menjadi lebih dan lebih rumit dan rawan kesalahan karena program semakin besar dalam hal jumlah pilihan, jumlah loop, dan kompleksitas pernyataan seleksi. Mengotomasi tugas akan membuat pengujian jalur lebih cepat, kurang sulit, dan sedikit rawan kesalahan.

Konstruksi CFG manual membutuhkan banyak waktu untuk dilakukan.  
2. Identifikasi jalur target manual memerlukan kreativitas tester.  
3. Proses instrumentasi program manual tergantung pada bahasa pemrograman, yang harus dilakukan sedemikian rupa sehingga tidak mengubah semantik program. Oleh karena itu, instrumentasi manual memerlukan sejumlah besar waktu dan kerja.

Untuk itu, perlu dilakukan pengujian dengan mencoba segala kemungkinan jalur

## Perumusan Masalah

Bagaimana membuat control flow graph dan melakukan instrumentasi source code secara otomatis untuk melacak jalur mana yang dilalui jika suatu program di eksekusi.

Memberikan jaminan bahwa semua jalur independen pada suatu modul telah digunakan paling tidak satu kali.

Menggunakan semua keputusan logis pada sisi true and false

Kesalahan logika dan asumsi yang tidak benar kebanyakan dilakukan ketika coding untuk “kasus tertentu”. Dibutuhkan kepastian bahwa eksekusi jalur ini telah dites. Asumsi bahwa adanya kemungkinan terhadap eksekusi jalur yang tidak benar. Dengan white box testing  dapat ditemukan kesalahan ini Kesalahan penulisan yang acak. Seperti berada pada jalur logika yang membingungkan pada jalur normal.

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.

* Apakah penerapan metode klasifikasi Rocchio sesuai untuk analisa hoax?
* Apakah metode klasifikasi klasifikasi Rocchio memiliki nilai akurasi lebih baik dari Naive Bayes?

## Tujuan Penelitian

Dapat membangkitkan jalur program dan instrumentasi source code secara otomatis

Tuuan utamanya membangkitkan CFG Biar gampang ngetestingnya

Penting untuk dibuat berguna untuk testing dan untuk reengineering

* Dapat menggunakan metode klasifikasi Rocchio untuk analisa hoax
* Dapat membandingkan hasil akurasi antara metode klasifikasi Rocchio dan Naive Bayes

## Ruang Lingkup Penelitian

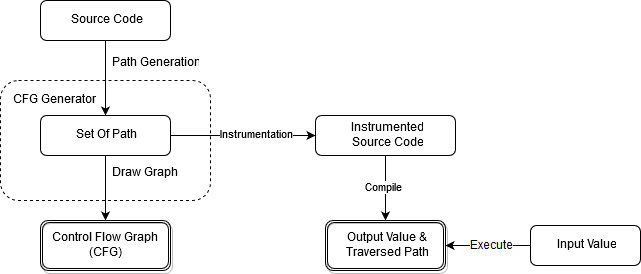
Penelitian ini melakukan pembangkitan control flow graph dan instrumentasi source code bahasa C

TINJAUAN PUSTAKA

Path Testing adalah

Control Flow Graph adalah

METODE PENELITIAN



Pengumpulan Data

Data dikumpulkan dari contoh penelitian yang

code coverage dilaksanakan dengan menggunakan instrumentation.

Instrumentation adalah istilah yang digunakan untuk menunjukkan manipulasi aplikasi dengan cara melakukan injeksi kode-kode untuk reporting ke dalam posisi-posisi tertentu yang bisa menunjukkan bagian yang telah dicakup oleh test atau belum. Instrumentasi dilakukan pada level

source code atau level compiled

. Pada level source code, tools yang digunakan akan menginjeksikan kode instrumentasi ke source code (mengubah source code)

baru kemudian di kompilasi. Pada level compiled

, kode instrumentasi diinjeksikan setelah dikompilas

Pada penelitian ini, data yang digunakan menggunakan test yang digunakan pada penelitian

Teridiri atas beberapa tingkatan yang diurutkan dari yang terkecil ke yang terbesar lingkupnya.

Yang paling luas pengujian jalur (path)

Path testing dimana semua jalur logika dalam sebuah program harus dieksekusi minimal 1 kali.

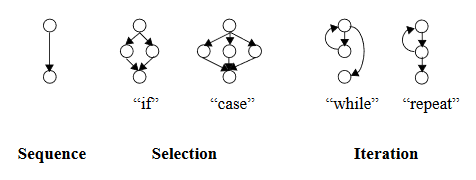
Pada kasus loop harus dibatasi agar banyaknya jalur tidak tak terhingga jumlahnya.

Dalam hal ini, pada sebuah penulangan, jalur dengan satu kali pengulangan, dan jalur dengan beberapa kali pengulangan yang diwakili dengan dua kali pengulangan saja.

Dalam pengujian jalur ini, beberapa hal yang harus dilakukan adalah konstruksi CFG, pembangkitan jalur, dan instrumentasi.

Setelah CFG terbentuk, langkah selanjutnya adalah melakukan pembangkitan jalur. Jalur direpresentasikan dalam bentuk sequens pasangan

Untuk melakukan monitoring jalur mana yan diambil oleh sebuah measukan pada saat eksekusi program, maka diperlukan tanda (atau tag) yang dapat memberikan informasi cabang mana yang diambil (1 atau 0) pada sebuah percabangan. Instrumentasi dilakukan dengan cara menyisipkan tanda-tanda tersebut ke dalam program, biasanya tepat sebelum sebuah percabangan. Selain untuk mengetahuii cabang mana yang dilalui pada saat eksekusi, tanda-tanda tersebut juga digunakan untuk menghitung subnilai fitness dari sebuah data masukan



DAFTAR PUSTAKA

Basis path testing, a structured testing or white box testing technique used for designing test cases intended to examine all possible paths of execution at least once. Creating and executing tests for all possible paths results in 100% statement coverage and 100% branch coverage.

(1)

Automatic test case generation tool would be an automated test case generation system that

allows the tester/user to input a ‘C’ program. It will parse through the program. Based on the

internal structure of the program, CFG (Control Flow Graph) for the input program is generated.

the program. (2) The automatic test case

generation tool will use this CFG generated to

generate the CFG matrix. Using this CFG matrix,

various possible paths through the program will

be identified. (3) First, the automatic test case

generation tool will be implemented using

random testing based technique, then it will

allow up to some extent to get the information

about the feasibility or infeasibility of a path /

difficulty in solving the path. Finally, this tool

will generate the test cases for input program. (4)

The automatic test case generation tool will

develop test cases based on best available testing

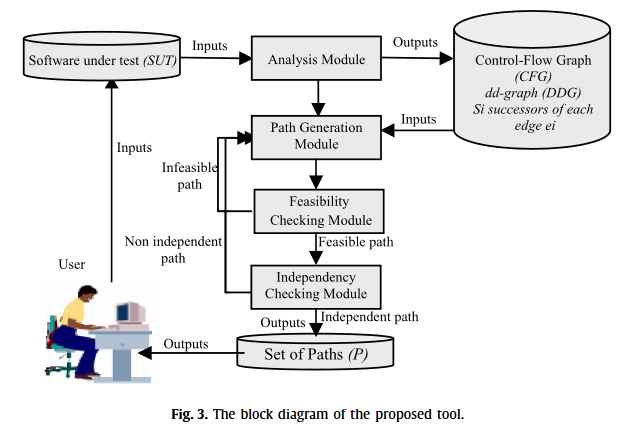
criteria that will provide the assurance of

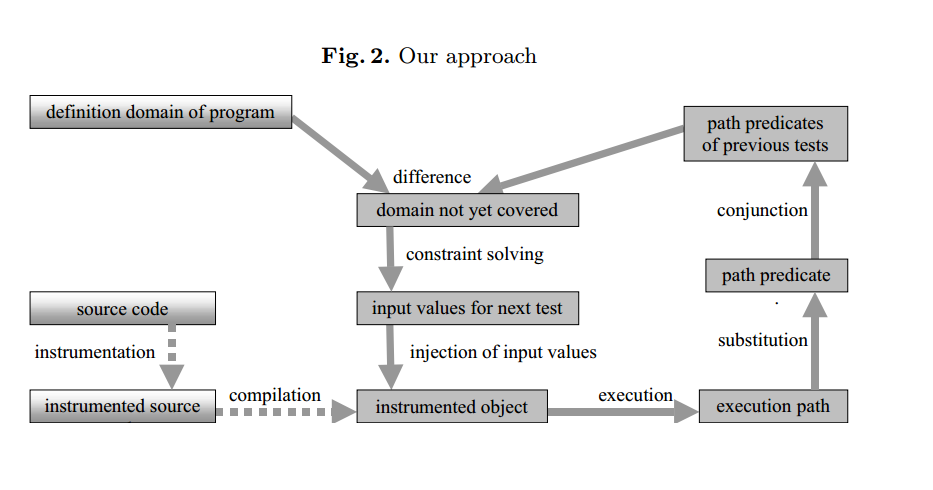
complete possible testing and also provide the

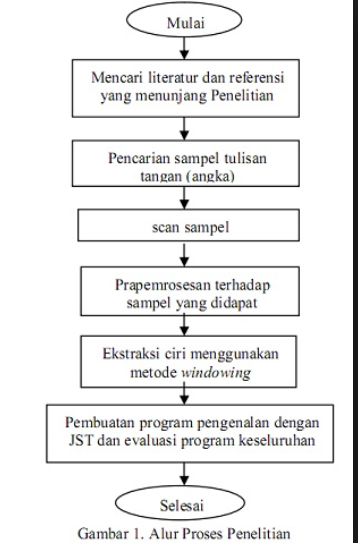
information about the termination of the testing

process. Unit testing is used for this research by

analysis of all testing strategies [14]







Functional veri cation comprises a large p ortion of the e ort in designing a pro cessor [1]. The investment in exp ert time and computer resources is huge, and so is the cost of deliver- ing faulty pro ducts [4]. In current industrial practice, most of the veri cation is done by generating a massive amount of tests by random test generators [1, 2, 9]. The use of ad- vanced random test generators can increase the qualit yof generated tests, but it cannot detect cases in which some ar- eas of the design are not tested, while other parts are tested rep eatedly. The main technique for checking and showing that the testing has b een thorough, is called test coverage analy- sis [10]. Simply stated, the idea is to create, in a systematic fashion, a large and comprehensive list of tasks and check that each task was covered in the testing phase. Coverage can help in monitoring the quality of testing, and assist in directing the test generators to create tests that cover areas that have not b een tested b efore. Man ycov erage to ols exist in the market. Most co verage to ols are program-based and chec kcodecov erage of execu- tion of programs. Examples include co de coverage in C [8] or VHDL [12], and covering states and transitions of state machines [13]. The main disadvantage of co de coverage is that it do es not \understand" the applicatio n domain. It is,

%----------------------------------------------------------------------------------------

% PENDAHULUAN

%----------------------------------------------------------------------------------------

\section\*{PENDAHULUAN} % Sub Judul PENDAHULUAN

% Tuliskan isi Pendahuluan di bagian bawah ini.

% Jika ingin menambahkan Sub-Sub Judul lainnya, silakan melihat contoh yang ada.

% Sub-sub Judul

\subsection\*{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. Pengujian perangkat lunak juga bertujuan untuk membangun kepercayaan dalam pengoprasian perangkat lunak ketika tidak ditemukan kesalahan pada hasil pengujian. 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. (\cite{MCCABE}).

Teknik pengujian secara umum dibagi menjadi 2 kategori diantaranya \textit{black box testing} dan \textit{white box testing}. \textit{White box testing} atau biasa disebut dengan pengujian struktural merupakan pemeriksaan struktur dan alur logika suatu proses. Salah satu metode \textit{white box testing} adalah pengujian jalur (\textit{path testing}). Path testing merupakan metode pengujian struktural yang menggunakan \textit{source code} dari program untuk menemukan semua jalur yang mungkin dapat dilalui program dan memastikan semua jalur dijalankan setidaknya satu kali (\cite{BASU2015}). 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. (\cite{TIKIR2011})

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. \citeauthor{KUMAR20168} (\cite\*{KUMAR20168}) mengatakan bahwa pengujian perangkat lunak menggunakan kira-kira 30\%-50\% dari total sumber daya, 30\% dari total upaya yang dilakukan, dan 50\%-60\% dari total biaya pengembangan perangkat lunak.

Sehingga dibutuhkan otomasi untuk melakukan instrumentasi \textit{source code} program yang akan diuji. Pengujian perangkat lunak otomatis dapat mengurangi biaya pengembangan secara signifikan. Jalur yang dibangkitkan juga dapat direpresentasikan oleh \textit{Control Flow Graph} (CFG). CFG dapat digunakan ketika akan melakukan \textit{reengineering} program untuk memahami alur program. Hal ini dapat mempermudah ketika minimnya dokumentasi pengembangan dari suatu perangkat lunak.

% Sub-sub Judul

\subsection\*{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.

\subsection\*{Tujuan}

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

\subsection\*{Ruang Lingkup}

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

\subsection\*{Manfaat}

Hasil penelitian diharapkan dapat membantu pengembang dan penguji aplikasi untuk:

\begin{enumerate}[noitemsep]

\item Melacak jalur yang dilalui ketika suatu program dieksekusi.

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

\item Memudahkan pengembang dalam memahami struktur dari suatu program yang dapat dimanfaatkan ketika akan melakukan \textit{reengineering} perangkat lunak.

\end{enumerate}

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. White box testing atau biasa disebut dengan pengujian struktural merupakan pemeriksaan struktur dan alur  
logika suatu proses. Salah satu metode white box testingadalah pengujian jalur (path testing). Path testing merupakan 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 dapat mengurangi biaya pengembangan secara  
signifikan.  
Sehingga dibutuhkan otomatisi 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 secara  
otomatis. Untuk memastikan semua jalur dijalankan  
setidaknya 1 kali, maka sistem ini juga akan melakukan  
instrumentasi source code program secara otomatis