SKRIPSI

PENGUJIAN BERBASIS *BEHAVIOUR SPECIFICATION*

Muhammad Dipo Putra Wandara

NPM: 2016730091

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2019

UNDERGRADUATE THESIS

PENGUJIAN BERBASIS *BEHAVIOUR SPECIFICATION*

Muhammad Dipo Putra Wandara

NPM: 2016730091

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2019

LEMBAR PENGESAHAN

PENGUJIAN BERBASIS *BEHAVIOUR SPECIFICATION*

Muhammad Dipo Putra Wandara

NPM: 2016730091

Bandung, 20 Agustus 2019

Menyetujui,

Pembimbing Utama	Pembimbing Pendamping
Raymond Chandra	**«pembimbing pendamping/2»**

Ketua Tim Penguji	Anggota Tim Penguji
«penguji 1»	**«penguji 2»**

Mengetahui,

Ketua Program Studi

Mariskha Tri Adithia, P.D.Eng

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

PENGUJIAN BERBASIS *BEHAVIOUR SPECIFICATION*

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhkan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal 20 Agustus 2019

Meterai
Rp. 6000

Muhammad Dipo Putra Wandara
NPM: 2016730091

ABSTRAK

«Tuliskan abstrak anda di sini, dalam bahasa Indonesia»
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Kata-kata kunci: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Indonesia»

ABSTRACT

«Tuliskan abstrak anda di sini, dalam bahasa Inggris»
Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Keywords: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Inggris»

«kepada siapa anda mempersembahkan skripsi ini. . . ?»

KATA PENGANTAR

«Tuliskan kata pengantar dari anda di sini . . . »
Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Bandung, Agustus 2019

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Pengujian perangkat lunak merupakan salah satu tahapan dari *software development life cycle*. *Software Development Life Cycle*(SDLC) merupakan metodologi dari pengembangan perangkat lunak, metode ini dibagi menjadi beberapa fase seperti *analysis, design, coding, testing, installation* dan *maintenance*[7]. *Testing*/Pengujian perangkat lunak adalah proses untuk mencari kesalahan pada setiap *item* perangkat lunak, mencatat hasilnya, mengevaluasi setiap aspek pada setiap komponen (sistem) dan mengevaluasi fasilitas-fasilitas dari perangkat lunak yang akan dikembangkan[1].Pengujian pada perangkat lunak merupakan tahapan yang wajib dilakukan sebelum perangkat lunak tersebut digunakan, agar memastikan sudah tidak ada *error/bug* pada perangkat lunak yang sedang dikembangkan. Pengujian perangkat lunak memakan *resource* yang berat, baik itu waktu maupun waktu maupun tenaga kerja, karena untuk melakukan pengujian perangkat lunak dibutuhkan *developer* ahli *Software Quality Assurance*(SQA). SQA yang bertanggung jawab untuk memastikan bahwa perangkat lunak yang sedang dikembangkan bebas dari *bug* agar siap untuk di-*release* dan digunakan oleh pengguna.

Unit testing merupakan tahapan pertama dari pengujian perangkat lunak. *Unit Testing* adalah proses pengujian bagian kode secara individu, suatu komponen, untuk menentukan apakah kode tersebut berfungsi dengan semestinya[5]. Salah satu teknik untuk membuat *unit testing* yaitu dengan code generation. *Code generation* adalah teknik membuat suatu program yang membuat program lain. Menggunakan *code generation* untuk *unit testing* membuatnya mudah untuk mempertahankan dan memperpanjang *unit testing*[6].

Ada beberapa metode untuk melakukan pengujian perangkat lunak, salah satunya yaitu *Test-Driven Development*(TDD). TDD adalah praktik pemrograman yang menginstruksikan pengembang perangkat lunak untuk menulis kode baru hanya ketika tes yang dilakukan secara otomatis gagal, dan menghilangkan duplikasi. Tujuan TDD adalah kode bersih yang berfungsi[2]. Tetapi, TDD memiliki masalah, bahwa TDD berfokus pada keadaan sistem daripada *behaviour* yang diinginkan oleh sistem, dan kode pada pengujian *highly coupled* dengan implementasi sistem yang ada[3]. Maka dari itu metode yang akan digunakan pada skripsi ini adalah *Behaviour-Driven Development*.

Behaviour-Driven Development(BDD) merupakan evolusi dari TDD untuk menyelesaikan masalah yang ada pada TDD. BDD adalah pendekatan pengembangan perangkat lunak yang *agile* untuk mendorong kolaborasi antara semua peserta dalam proses pengembangan perangkat lunak[4].

Prinsip inti dari BDD adalah "orang bisnis dan teknologi harus merujuk ke sistem yang sama dengan cara yang sama"[4]. Berbeda dengan TDD yang berpatokan pada test untuk *develope* lebih jauh program yang akan digunakan, BDD berpatokan pada keinginan *stakeholder/customer* untuk mengembangkan programnya. Untuk mencapai kesepakatan antara *developer* dan *stakeholder/customer*, maka dibutuhkan *language* untuk menspesifikasikan *behaviour* sebuah sistem agar kedua pihak paham, dan dapat mewujudkan hal berikut[4]:

- *Stakeholder/Customer* untuk menentukan persyaratan dari perspektif bisnis.
- Analis bisnis untuk melampirkan contoh konkret (skenario atau *acceptance tests*) yang menje-

laskan *behaviour* sistem.

- *Developer* untuk mengimplementasikan *behaviour* sistem yang diperlukan secara TDD.

Pengujian dengan basis *behaviour specification* akan berfokus kepada hal yang *stakeholder/customer* harapkan pada suatu sistem dengan *behaviour* yang sudah di spesifikasikan. Pengujian akan berawal dengan membuat *Behaviour Test* yang berisi [4]:

1. *Context/Starting state* : Posisi awal sistem sebelum terjadinya suatu *event*.
2. *Event* : *Task* yang dilakukan oleh pengguna pada sistem.
3. *Outcome* : Hasil yang diharapkan dari sistem.

Developer akan menjadikan *Behaviour Test* sebagai patokan dasar bekerjanya suatu sistem, dan akan melakukan pengujian berbasis *Behaviour Test*. Pengujian dilakukan untuk memastikan bahwa sistem sudah bekerja sesuai apa yang *stakeholder/customer* harapkan, dan jika *Behaviour Test* berhasil dilakukan, maka sistem sudah siap untuk digunakan.

1.2 Rumusan Masalah

- Bagaimana cara kerja pengerjaan Pengujian Berbasis Behaviour Specification?
- Bagaimana implementasi perangkat lunak yang dapat menguji program berbasis Behaviour Specification ?

1.3 Tujuan

- Mempelajari cara kerja Pengujian Berbasis Behaviour Specification
- Menghasilkan perangkat penguji yang dapat menguji sebuah perangkat lunak berbasiskan *behaviour specification*.

1.4 Batasan Masalah

Mengingat banyaknya perkembangan yang bisa ditemukan dalam permasalahan ini, maka perlu adanya batasan-batasan masalah yang jelas mengenai apa yang dibuat dan diselesaikan dalam program ini. Adapun batasan-batasan masalah pada penelitian ini sebagai berikut :

1.5 Metodologi

Bagian-bagian pekerjaan skripsi ini adalah sebagai berikut :

1. Melakukan Studi literatur mengenai pengujian perangkat lunak, *unit testing*, dan *behaviour specification*.
2. Melakukan eksplorasi perangkat lunak.
3. Menganalisis kebutuhan perangkat lunak.
4. Membuat rancangan desain perangkat lunak, basis data, antarmuka perangkat lunak.
5. Implementasi perangkat lunak yang menerima *input behaviour spesification*, mengolah, dan mengubah ke bentuk *unit testing*.
6. Melakukan pengujian terhadap perangkat lunak.
7. Melaporkan hasil penelitian dalam bentuk dokumen skripsi.

1.6 Sistematika Pembahasan

- BAB I. PENDAHULUAN
  Bab ini berisi tentang latar belakang masalah, rumusan masalah, tujuan, batasan masalah, dan metodologi.
- BAB II. LANDASAN TEORI
  Bab 2 memuat uraian tentang teori yang akan digunakan pada pembuatan perangkat lunak.

BAB 2

LANDASAN TEORI

2.1 Pengujian Perangkat Lunak

Pandangan umum pengujian perangkat lunak adalah bahwa kegiatan ini adalah untuk menemukan *bug*. Tujuan pengujian perangkat lunak adalah untuk memenuhi syarat kualitas program perangkat lunak dengan mengukur atribut dan kemampuannya terhadap ekspektasi dan standar yang berlaku. Pengujian perangkat lunak juga menyediakan informasi berharga untuk upaya pengembangan perangkat lunak.[9]

Kualitas perangkat lunak adalah sesuatu yang diinginkan semua orang. Manajer tahu bahwa mereka menginginkan kualitas tinggi, pengembang perangkat lunak tahu mereka ingin menghasilkan produk yang berkualitas, dan pengguna bersikeras bahwa perangkat lunak bekerja secara konsisten dan dapat diandalkan.[9]

Banyak kelompok kualitas perangkat lunak mengembangkan *software quality assurance plan*, dimana hal itu sama dengan *test plans*. Rencana jaminan kualitas perangkat lunak dapat mencakup berbagai kegiatan di luar yang termasuk dalam *test plan*. *Quality assurance plan* mencakup keseluruhan kualitas, rencana pengujian adalah salah satu alat kontrol kualitas dari rencana jaminan kualitas.[9]

Pada pembahasan pengujian perangkat lunak, ada *term* yang biasa digunakan, yaitu[8]:

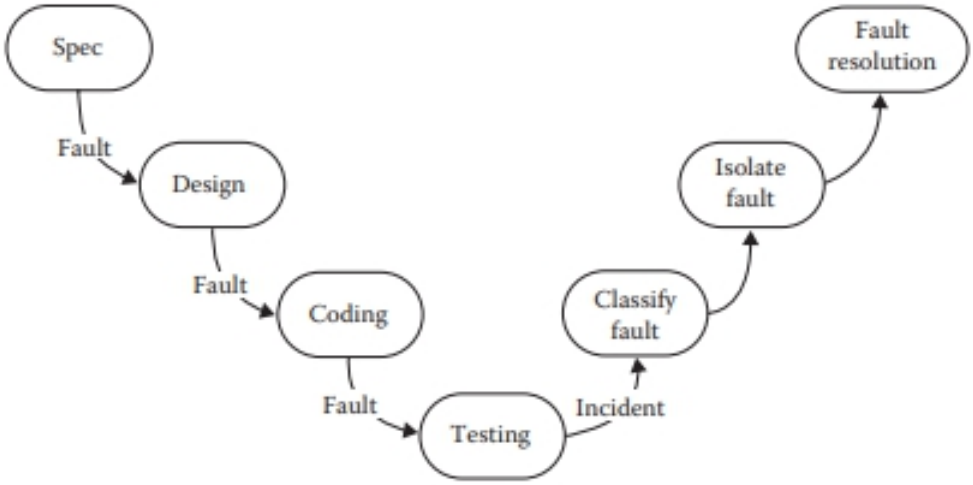
- **Error** — Orang membuat *error*. Sinonim yang baik adalah *mistake*. Ketika orang membuat *error* saat melakukan *coding*, kami menyebut *error* ini *bug*. *Error* cenderung menyebar; *requirements error* dapat diperbesar selama proses desain dan lebih diperkuat lagi selama pengkodean.

- **Fault** — *Fault* adalah hasil dari *error*. Lebih tepat untuk mengatakan bahwa *fault* adalah representasi dari *error*, di mana representasi adalah mode ekspresi, seperti teks naratif, diagram Bahasa Pemodelan Bersatu, diagram hierarki, dan kode sumber. *Defect* adalah sinonim yang baik untuk *fault*, sama juga seperti *bug*. *Fault* bisa sulit dipahami. *Error* yang disebakan oleh kelalaian menghasilkan *fault* di mana ada sesuatu yang hilang yang seharusnya ada di dalam representasi.

- ***Failure*** — *Failure* terjadi ketika kode yang sesuai dengan *fault* dijalankan. Dua kehalusan muncul di sini: satu adalah bahwa *failure* hanya terjadi dalam representasi yang dapat dieksekusi, yang biasanya dianggap sebagai kode sumber, atau lebih tepatnya, kode objek yang dimuat; kehalusan kedua adalah bahwa definisi ini hanya mengaitkan *failure* dengan *fault* komisi.

- **Incident** — Ketika *failure* terjadi, itu mungkin atau mungkin tidak mudah terlihat oleh pengguna (atau pelanggan atau penguji). Suatu *incident* adalah gejala yang terkait dengan *failure* yang memberi tahu pengguna tentang terjadinya *failure*.

- **Test** — *Testing* jelas berkaitan dengan *errors, faults, failures, and incidents*. *Test* adalah tindakan melatih perangkat lunak dengan *test case*. *Test* memiliki dua tujuan berbeda: untuk menemukan *failures* atau untuk menunjukkan eksekusi yang benar.



Gambar 2.1: Siklus hidup pengujian.

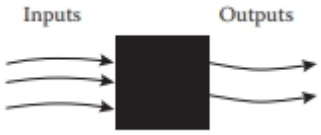
- **Test case** — *Test case* memiliki identitas dan dikaitkan dengan perilaku program. Ini juga memiliki serangkaian input dan output yang diharapkan.

Gambar 2.1 menggambarkan model siklus hidup untuk pengujian. Perhatikan bahwa, dalam fase pengembangan, tiga peluang muncul untuk membuat *error*, yang menghasilkan *fault* yang dapat menyebar melalui proses *development*. Langkah resolusi *fault* adalah kesempatan lain untuk *error* (dan *fault* baru). Ketika suatu perbaikan menyebabkan perangkat lunak yang sebelumnya benar untuk berperilaku salah, perbaikannya kurang[8].

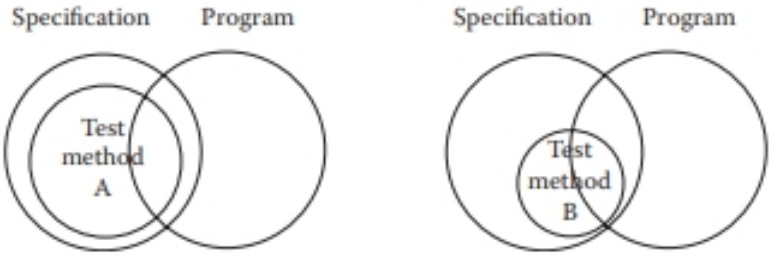
Dari urutan *term* ini, dapat dilihat bahwa *test case* menempati posisi sentral dalam pengujian. Proses pengujian dapat dibagi lagi menjadi langkah-langkah terpisah: *test planning, test case development*, menjalankan *test case*, dan mengevaluasi hasil pengujian.Untuk *test case* ada dua pendekatan mendasar digunakan untuk mengidentifikasi *test case*; secara tradisional, ini disebut pengujian fungsional dan struktural. *Specification-based* dan *code-based* adalah nama yang lebih deskriptif. Kedua pendekatan memiliki beberapa metode identifikasi *test case* yang berbeda; mereka umumnya hanya disebut metode pengujian.

2.1.1 *Specification-Based/Black-box Testing* [8]

Alasan bahwa pengujian berbasis spesifikasi pada awalnya disebut *functional testing* adalah bahwa setiap program dapat dianggap sebagai fungsi yang memetakan nilai dari domain input ke nilai dalam rentang outputnya. Gagasan ini umumnya digunakan dalam *engineering*, ketika suatu sistem dianggap sebagai *black box*. Ini mengarah pada istilah sinonim lainnya — pengujian *black-box*, di mana konten (implementasi) kotak hitam tidak diketahui, dan fungsi kotak hitam dipahami sepenuhnya dalam hal input dan outputnya (lihat Gambar 2.2). Sering kali, pengujian beroperasi sangat efektif dengan pengetahuan *black box*; pada kenyataannya, ini adalah pusat orientasi objek. Sebagai contoh, kebanyakan orang berhasil mengoperasikan mobil dengan hanya pengetahuan "*black box*".



Gambar 2.2: Engineer's black box.



Gambar 2.3: Comparing specification-based test case identification methods

Dengan pendekatan berbasis spesifikasi untuk menguji identifikasi kasus, satu-satunya informasi yang digunakan adalah spesifikasi perangkat lunak. Oleh karena itu, *test case* memiliki dua keunggulan berbeda:

1. Mereka tidak tergantung pada bagaimana perangkat lunak diimplementasikan, jadi jika implementasi berubah, *test case* masih berguna.

2. Pengembangan *test case* dapat terjadi secara paralel dengan implementasi, sehingga mengurangi keseluruhan interval pengembangan proyek.

Di sisi negatif, kasus uji berbasis spesifikasi sering mengalami dua masalah, redudansi yang signifikan mungkin ada di antara kasus uji, dan diperparah oleh kemungkinan kesenjangan perangkat lunak yang tidak diuji.

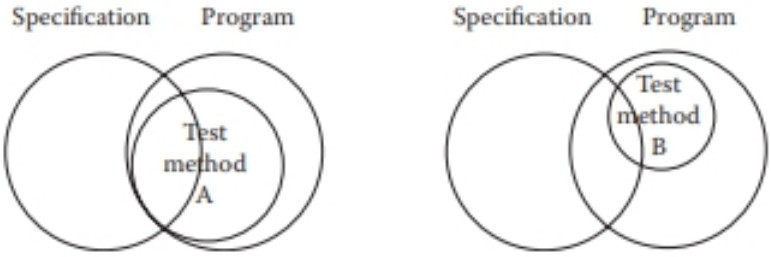
Gambar 2.3 menunjukkan hasil *test case* yang diidentifikasi oleh dua metode berbasis spesifikasi. Metode A mengidentifikasi serangkaian kasus uji yang lebih besar daripada metode B. Perhatikan bahwa, untuk kedua metode, rangkaian kasus uji sepenuhnya terkandung dalam rangkaian perilaku tertentu. Karena metode berbasis spesifikasi didasarkan pada perilaku yang ditentukan, sulit untuk membayangkan metode ini mengidentifikasi perilaku yang tidak ditentukan.

2.1.2 *Code-Based Testing/White-box Testing*

Pengujian berbasis kode adalah pendekatan mendasar lainnya untuk menguji *test case*. Untuk membandingkannya dengan pengujian *black box*, kadang-kadang disebut pengujian *white box* (atau bahkan *clear box*). Metafora *clear box* mungkin lebih tepat karena perbedaan mendasar adalah bahwa implementasi (*black box*) diketahui dan digunakan untuk mengidentifikasi *test case*. Kemampuan untuk "melihat ke dalam" *black box* memungkinkan tester untuk mengidentifikasi *test case* berdasarkan bagaimana fungsi tersebut benar-benar dijalankan.

Pengujian berbasis kode telah menjadi subjek dari beberapa teori yang cukup kuat. Dengan konsep-konsep ini, tester dapat dengan ketat menggambarkan dengan tepat apa yang akan diuji. Karena dasar teorinya yang kuat, pengujian berbasis kode cocok untuk menjadi definisi dan penggunaan *test coverage metrics*. *Test coverage metrics* menyediakan cara untuk secara eksplisit menyatakan sejauh mana *item* perangkat lunak telah diuji, dan ini pada gilirannya membuat manajemen pengujian lebih jelas.

Gambar 2.4 menunjukkan hasil *test case* yang diidentifikasi oleh dua metode berbasis kode. Seperti sebelumnya, metode A mengidentifikasi satu set kasus uji yang lebih besar daripada metode B. Apakah satu set kasus uji yang lebih besar tentu lebih baik? Ini adalah pertanyaan yang bagus, dan pengujian berbasis kode menyediakan cara-cara penting untuk mengembangkan jawaban. Perhatikan bahwa, untuk kedua metode, himpunan kasus uji sepenuhnya terkandung dalam himpunan perilaku yang diprogram. Karena metode berbasis kode didasarkan pada program, sulit membayangkan metode ini mengidentifikasi perilaku yang tidak diprogram. Sangat mudah untuk membayangkan, bagaimanapun, bahwa serangkaian kasus uji berbasis kode relatif kecil sehubungan dengan perilaku lengkap yang diprogram.



Gambar 2.4: Comparing code-based test case identification methods.

2.2 Bahasa Pemrograman PHP

2.3 Bahasa Pemrograman HTML

2.4 Codeigniter

2.5 Code Coverage

2.6 Xampp

2.7 Travis CI

LAMPIRAN A

KODE PROGRAM

Listing A.1: MyCode.c

```c

// This does not make algorithmic sense,
// but it shows off significant programming characters.

#include<stdio.h>

void myFunction( int input, float* output ) {
    switch ( array[i] ) {
        case 1: // This is silly code
            if ( a >= 0 || b <= 3 && c != x )
                *output += 0.005 + 20050;
            char = 'g';
            b = 2^n + ~right_size - leftSize * MAX_SIZE;
            c = (--aaa + &daa) / (bbb++ - ccc % 2 );
            strcpy(a,"hello $@?");
    }
    count = ~mask | 0x00FF00AA;
}

// Fonts for Displaying Program Code in LATEX
// Adrian P. Robson, nepsweb.co.uk
// 8 October 2012
// http://nepsweb.co.uk/docs/progfonts.pdf
```

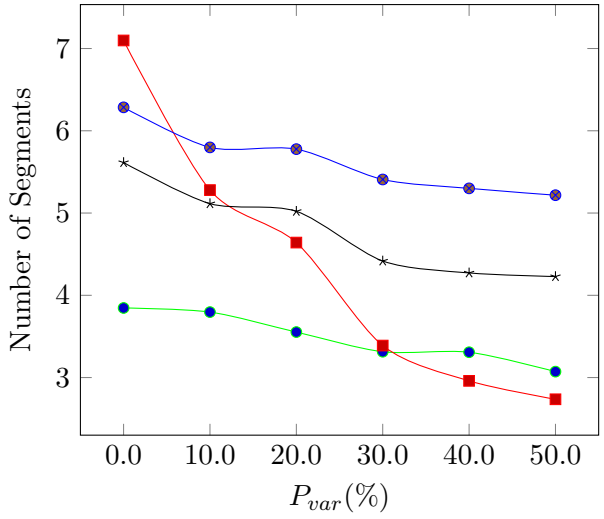
Listing A.2: MyCode.java

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;

//class for set of vertices close to furthest edge
public class MyFurSet {
    protected int id;                                   //id of the set
    protected MyEdge FurthestEdge;                      //the furthest edge
    protected HashSet<MyVertex> set;                    //set of vertices close to furthest edge
    protected ArrayList<ArrayList<Integer>> ordered;    //list of all vertices in the set for each trajectory
    protected ArrayList<Integer> closeID;               //store the ID of all vertices
    protected ArrayList<Double> closeDist;              //store the distance of all vertices
    protected int totaltrj;                             //total trajectories in the set

    /*
     * Constructor
     * @param id : id of the set
     * @param totaltrj : total number of trajectories in the set
     * @param FurthestEdge : the furthest edge
     */
    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
        this.id = id;
        this.totaltrj = totaltrj;
        this.FurthestEdge = FurthestEdge;
        set = new HashSet<MyVertex>();
        ordered = new ArrayList<ArrayList<Integer>>();
        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
        closeID = new ArrayList<Integer>(totaltrj);
        closeDist = new ArrayList<Double>(totaltrj);
        for (int i = 0;i <totaltrj;i++) {
            closeID.add(-1);
            closeDist.add(Double.MAX_VALUE);
        }
    }

}
```

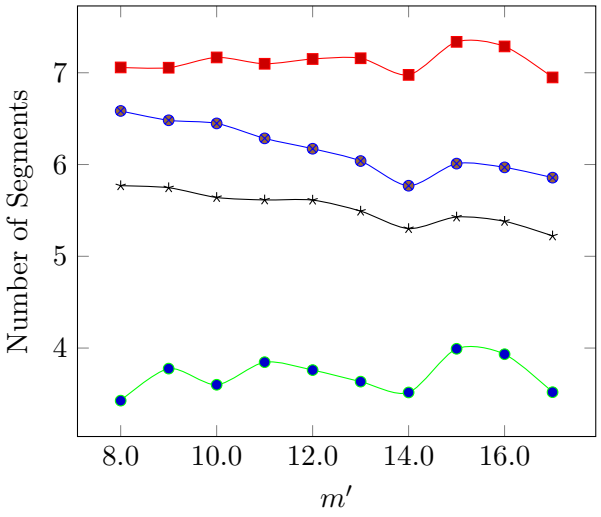
LAMPIRAN B

HASIL EKSPERIMEN

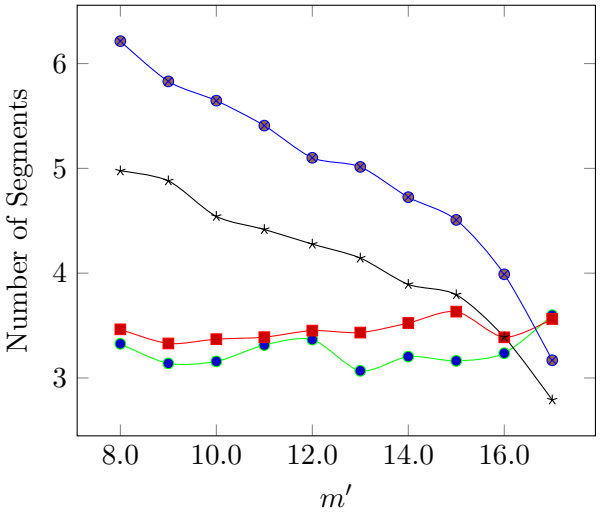
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



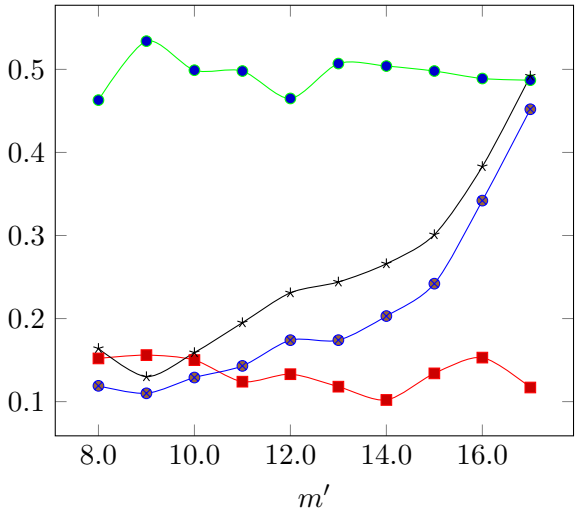
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4