

SKRIPSI

**OTOMASI PEMBANGUNAN KASUS TES MENGGUNAKAN
PENDEKATAN EVOLUSIONER**



ANDRE TIMOTI MAHADIKA

NPM: 2012730085

**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2017**

UNDERGRADUATE THESIS

**AUTOMATED TEST CASE GENERATION USING
EVOLUTIONARY APPROACH**



ANDRE TIMOTI MAHADIKA

NPM: 2012730085

**DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2017**



LEMBAR PENGESAHAN

OTOMASI PEMBANGUNAN KASUS TES MENGGUNAKAN PENDEKATAN EVOLUSIONER


ANDRE TIMOTI MAHADIKA

NPM: 2012730085

Bandung, 7 Desember 2017

Menyetujui,

Pembimbing Tunggal


Dett. Thomas Anung Basuki

Anggota Tim Penguji

Ketua Tim Penguji


Husnul Hakim, M.T.


**Kristopher David Harjono, S.Kom,
M.T.**

Mengetahui,

Ketua Program Studi


Mariskha Tri Adithia, P.D.Eng



PERNYATAAN

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

OTOMASI PEMBANGUNAN KASUS TES MENGGUNAKAN PENDEKATAN EVOLUSIONER

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 7 Desember 2017



Andre Timoti Mahadika
NPM: 2012730085

ABSTRAK

Dalam membangun perangkat lunak, pengujian merupakan salah satu langkah penting yang harus dilakukan. Pengujian bertujuan untuk mencari kesalahan yang terdapat pada perangkat lunak. Untuk mendapatkan hasil pengujian yang baik, diperlukan kasus tes yang baik juga. Kasus tes yang baik adalah kasus tes yang dapat menjangkau keseluruhan fungsi perangkat lunak. Pembangunan kasus tes yang baik akan memakan banyak waktu dan melelahkan bagi penguji.

Salah satu algoritma yang dapat digunakan untuk menyelesaikan masalah ini adalah algoritma genetik. Algoritma genetik merupakan cabang dari ilmu *Evolutionary Computation*. *Evolutionary Computation* merupakan ilmu yang didasarkan teori evolusi dari alam. Pada teori evolusi, hanya gen-gen yang baik yang akan bertahan hidup. Prinsip tersebut dipakai dalam algoritma genetik untuk membangun dan seleksi gen sehingga menghasilkan sebuah set gen yang baik.

Dalam skripsi ini, algoritma genetik digunakan untuk membangun kasus tes yang baik. Gen dalam algoritma genetik merupakan kasus tes. Teknik pengujian yang dipakai dalam skripsi ini adalah *white-box testing*. Perangkat lunak akan menghasilkan sebuah set kasus tes yang baik diakhir.

Berdasarkan hasil pengujian, perangkat lunak dapat menghasilkan set kasus tes terbaik yang dapat dicapai, disesuaikan dengan kombinasi variabel algoritma genetik. Dalam memaksimalkan kualitas set kasus tes yang baik, kombinasi variabel algoritma genetik harus disesuaikan dengan karakteristik perangkat lunak masukan.

Kata-kata kunci: Algoritma genetik, pengujian perangkat lunak, otomasi pembangunan kasus tes

ABSTRACT

Testing is an important step in generating software. Testing aims to find defects in a software. To achieve a good test result, a good test case is needed. A good test case have to cover all the functions of the software. Generating a good test case is a tiring and time-consuming activity for tester.

Genetic algorithm is one of many algorithms that can be used to solve this problem. Genetic algorithm is a branch of the "Evolutionary Computation" science. "Evolutionary Computation" is based on the theory of evolution applied in nature. In theory of evolution, only the best gene can survive. That principle is used in genetic algorithm to generate and select gene to produce a good set of genes.

In this undergraduate thesis, genetic algorithm is used to generate a set of good test case. A gene in genetic algorithm represent a test case. Testing technique used in this thesis is "white-box testing". The software will produce a good set of test case at the end.

Based on qualitative test results, the software can generate set of the best test case that can be achieved, given combination of genetic algorithm variable is adjusted. In order to maximize the set of test case quality, the genetic algorithm variable need to be adjusted according to input software.

Keywords: Genetic algorithm, testing a software, automated test case generation

*Untuk Tuhan Yesus Kristus yang telah menyertai saya sampai titik
ini.*

KATA PENGANTAR

Dengan ini saya ucapkan puji syukur kepada Tuhan Yesus Kristus yang telah membimbing dan menyertai saya dalam menyelesaikan skripsi ini.

Ucapan terimakasih untuk orang-orang yang sudah mendukung dan membantu saya dalam skripsi ini:

- Terimakasih kepada orang tua saya yang telah membiayai, mendukung dan mengajarkan nilai-nilai penting.
- Terimakasih kepada bapak dosen Dott. Thomas Anung Basuki yang telah dengan sabar menghadapi saya, membantu dan terus mendukung saya dalam penyelesaian perangkat lunak dan dokumen.
- Terimakasih kepada ibu dosen Mariskha Tri Adithia, P.D.Eng yang telah mendengar suara saya dan membolehkan saya untuk melanjutkan topik ini.
- Terimakasih kepada Angelina Nadya Giovani, pacar saya yang telah mendukung dan memberi semangat disaat-saat sulit.
- Terimakasih kepada Wych Dewangga, kawan seperjuangan skripsi dan teman baik saya, yang sering mengingatkan dan memberitahu segala informasi penting perkuliahan dan skripsi.
- Terimakasih kepada Astrid Soraya, yang membantu menyusun dokumen secara terstruktur.

Bandung, Desember 2017

Penulis

DAFTAR ISI

KATA PENGANTAR	xv
DAFTAR ISI	xvii
DAFTAR GAMBAR	xix
DAFTAR TABEL	xx
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Tujuan	3
1.4 Batasan Masalah	3
1.5 Metodologi Penelitian	3
1.6 Sistematika Pembahasan	3
2 DASAR TEORI	5
2.1 Menguji Perangkat Lunak	5
2.1.1 Failure, Defect and Error	5
2.1.2 Teknik Pengujian	6
2.1.3 <i>Cyclomatic Complexity</i>	7
2.2 <i>Genetic Algorithm</i>	8
3 ANALISIS	13
3.1 <i>Control Flow Diagram Generator</i>	13
3.2 <i>Test Requirement Generator</i>	14
3.3 <i>Genetic Algorithm Engine</i>	14
3.3.1 <i>Kromosom</i>	15
3.3.2 <i>Target</i>	15
3.3.3 <i>Fitness Function</i>	15
3.3.4 <i>Operator Seleksi</i>	16
3.3.5 <i>Operator Reproduksi</i>	16
3.4 <i>Kakas TRGeneration</i>	16
3.5 <i>File+temp</i>	19
4 PERANCANGAN	21
4.1 Perancangan Antarmuka	21
4.2 Perancangan Perangkat Lunak	22
4.2.1 GUI	24
4.2.2 GraphEngine	24
4.2.3 GeneticAlgorithm	27
5 IMPLEMENTASI DAN PENGUJIAN	43
5.1 Implementasi Antarmuka	43

5.2	Kode Program Masukan	44
5.2.1	<i>Fibonacci</i>	45
5.2.2	<i>Factorial</i>	46
5.2.3	<i>Greatest Common Divisor</i>	48
5.2.4	<i>Kelipatan Persekutuan Terkecil</i>	49
5.2.5	<i>Triangle Detector</i>	51
5.2.6	<i>Persamaan Kuadrat</i>	53
5.3	Pengujian Fungsional	54
5.4	Pengujian Eksperimental	59
5.4.1	Pengujian Eksperimental Mutation Rate	59
5.4.2	Pengujian Eksperimental Population Count	66
5.5	Kesimpulan Pengujian Eksperimental	74
6	KESIMPULAN DAN SARAN	77
6.1	Kesimpulan	77
6.2	Saran	77
	DAFTAR REFERENSI	79
	A KODE PROGRAM MASUKAN DAN <i>Filetemp</i>	81
	B SOURCE CODE	89

DAFTAR GAMBAR

2.1	AbstractServer flow graph(http://www.llosengcom)	6
2.2	Contoh <i>flow graph</i> untuk menghitung <i>cyclomatic complexity</i>	7
2.3	Ilustrasi penyilangan	12
3.1	Langkah kerja perangkat lunak	13
3.2	Diagram aktivitas <i>GAEngine</i>	14
4.1	Antarmuka bagian pertama	21
4.2	Antarmuka bagian kedua	22
4.3	Kelas diagram sederhana	22
4.4	Kelas diagram lengkap	23
4.5	Detil kelas diagram MainUI1	24
4.6	Detil kelas diagram MainUI2	24
4.7	Detil kelas diagram Graphs	25
4.8	Diagram aktivitas <i>generate path</i>	32
4.9	Detil kelas diagram Edge	33
4.10	Detil kelas diagram SimplePath	33
4.11	Detil kelas diagram Node	33
4.12	Detil kelas diagram defs	33
4.13	Detil kelas diagram GAEngine	33
4.14	Diagram aktivitas normalisasi	34
4.15	Kelas diagram <i>method fitnessFunctionPopulation</i>	35
4.16	Kelas diagram <i>method fitnessFunctionKromosom</i>	36
4.17	Diagram aktivitas bungkusMutate	37
4.18	Diagram aktivitas Mutate	38
4.19	Diagram aktivitas bungkusCrossOver	39
4.20	Diagram aktivitas CrossOver	40
4.21	Detil diagram aktivitas CompileNRun	41
4.22	Detil diagram aktivitas TimeoutProcessKiller	41
5.1	Antarmuka bagian pertama sesudah diimplementasi	43
5.2	Antarmuka bagian kedua sesudah diimplementasi	43
5.3	Antarmuka bagian pertama setelah dijalankan	43
5.4	Antarmuka bagian kedua setelah dijalankan	44
5.5	<i>Control Flow Diagram</i> GCD	55
5.6	Pembangunan <i>Test Requirement</i> manual GCD	55
5.7	Hasil dari perangkat lunak terhadap perangkat lunak masukan GCD	57

DAFTAR TABEL

5.1	Hasil pengujian eksperimental mutasi rendah terhadap <i>fibonnaci</i>	60
5.2	Hasil pengujian eksperimental mutasi menengah terhadap <i>fibonnaci</i>	60
5.3	Hasil pengujian eksperimental mutasi tinggi terhadap <i>fibonnaci</i>	60
5.4	Hasil pengujian eksperimental mutasi rendah terhadap <i>factorial</i>	61
5.5	Hasil pengujian eksperimental mutasi menengah terhadap <i>factorial</i>	61
5.6	Hasil pengujian eksperimental mutasi tinggi terhadap <i>factorial</i>	61
5.7	Hasil pengujian eksperimental mutasi rendah terhadap GCD	62
5.8	Hasil pengujian eksperimental mutasi menengah terhadap GCD	62
5.9	Hasil pengujian eksperimental mutasi tinggi terhadap GCD	62
5.10	Hasil pengujian eksperimental mutasi rendah terhadap KPK	63
5.11	Hasil pengujian eksperimental mutasi menengah terhadap KPK	63
5.12	Hasil pengujian eksperimental mutasi tinggi terhadap KPK	63
5.13	Hasil pengujian eksperimental mutasi rendah terhadap <i>triangle detector</i>	64
5.14	Hasil pengujian eksperimental mutasi menengah terhadap <i>triangle detector</i>	64
5.15	Hasil pengujian eksperimental mutasi tinggi terhadap <i>triangle detector</i>	64
5.16	Hasil pengujian eksperimental mutasi rendah terhadap persamaan kuadrat	65
5.17	Hasil pengujian eksperimental mutasi menengah terhadap persamaan kuadrat	65
5.18	Hasil pengujian eksperimental mutasi tinggi terhadap persamaan kuadrat	65
5.19	Rata-rata hasil pengujian eksperimental mutasi	66
5.20	Hasil pengujian eksperimental <i>population count</i> rendah terhadap <i>fibonnaci</i>	68
5.21	Hasil pengujian eksperimental <i>population count</i> menengah terhadap <i>fibonnaci</i>	68
5.22	Hasil pengujian eksperimental <i>population count</i> tinggi terhadap <i>fibonnaci</i>	68
5.23	Hasil pengujian eksperimental <i>population count</i> rendah terhadap <i>factorial</i>	69
5.24	Hasil pengujian eksperimental <i>population count</i> menengah terhadap <i>factorial</i>	69
5.25	Hasil pengujian eksperimental <i>population count</i> tinggi terhadap <i>factorial</i>	69
5.26	Hasil pengujian eksperimental <i>population count</i> rendah terhadap GCD	70
5.27	Hasil pengujian eksperimental <i>population count</i> menengah terhadap GCD	70
5.28	Hasil pengujian eksperimental <i>population count</i> tinggi terhadap GCD	70
5.29	Hasil pengujian eksperimental <i>population count</i> rendah terhadap KPK	71
5.30	Hasil pengujian eksperimental <i>population count</i> menengah terhadap KPK	71
5.31	Hasil pengujian eksperimental <i>population count</i> tinggi terhadap KPK	71
5.32	Hasil pengujian eksperimental <i>population count</i> rendah terhadap <i>triangle detector</i>	72
5.33	Hasil pengujian eksperimental <i>population count</i> menengah terhadap <i>triangle detector</i>	72
5.34	Hasil pengujian eksperimental <i>population count</i> tinggi terhadap <i>triangle detector</i>	72
5.35	Hasil pengujian eksperimental <i>population count</i> rendah terhadap persamaan kuadrat	73
5.36	Hasil pengujian eksperimental <i>population count</i> menengah terhadap persamaan kuadrat	73
5.37	Hasil pengujian eksperimental <i>population count</i> tinggi terhadap persamaan kuadrat	73
5.38	Rata-rata hasil pengujian eksperimental <i>population count</i>	74

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Perangkat lunak sudah menjadi aspek yang penting dalam kehidupan manusia. Oleh karena itu proses pembangunan perangkat lunak menjadi sangat penting untuk menghasilkan perangkat lunak yang memiliki kualitas tinggi. Terdapat enam langkah yang menjadi pedoman dalam pembangunan perangkat lunak[1], yaitu:

- Definisi kebutuhan.
- Desain sistem dan perangkat lunak.
- Implementasi dan pengujian unit.
- Integrasi dan pengujian sistem.
- Perawatan.

Salah satu langkah dalam pembangunan perangkat lunak adalah *software testing* yang berfungsi untuk melakukan tes terhadap perangkat lunak. Tujuannya adalah ditemukan cacat pada program yang tidak terlihat secara langsung pada proses *compiling*. Metode *testing* secara umum dibagi menjadi dua, yaitu:

- *white-box testing (glass-box testing)*.
- *black-box testing*.

Black-box testing merupakan metode testing yang memiliki rutin: memasukkan *input* lalu melihat dan mendiagnosa *output* untuk menemukan cacat yang berada pada kode program, di mana *output* yang diberikan tidak sesuai dengan apa yang diharapkan dengan *input* yang sama tanpa melihat potongan kode programnya sama sekali. *Glass-box testing* memiliki metode testing yang berbeda dari *black-box testing*, di mana pada *black-box testing*, penguji tidak melihat potongan kode program sama sekali, pada *glass-box testing*, penguji melihat dan menganalisa potongan kode program yang dites. Penguji akan melihat, mempelajari dan menganalisa potongan kode program yang akan dites, sehingga penguji dapat memfokuskan *test design* pada aspek yang lebih mendalam, yang tentunya akan meningkatkan kualitas tes pada aspek yang

lebih dalam juga (menemukan cacat pada aspek-aspek yang lebih rinci), seperti algoritma atau struktur data pada potongan kode program[2].

Testing pada perangkat lunak melibatkan banyak kasus berbeda dengan banyak variasi. Kasus-kasus tes tersebut akan menjadi *input* bagi perangkat lunak untuk melihat bagaimana perilaku perangkat lunak jika dihadapkan dengan masukan yang bervariasi luas. Tujuannya adalah untuk mengetahui pada *input* mana perangkat lunak akan bereaksi di luar kehendak perancang. Proses pembangunan berbagai macam kasus tersebut disebut sebagai *test case generation*. Pada dasarnya *test case generation* merupakan permasalahan kombinatorial.

Permasalahan pada bidang komputer sangatlah luas, salah satunya adalah masalah komputasi. Banyak orang berlomba-lomba untuk menyelesaikan masalah komputasi, untuk menemukan cara terbaik dalam menyelesaikan masalah komputasi, baik dalam hal *resource* maupun kecepatan komputasi dengan *resource* terbatas. Pada sekitar tahun 1970, seorang peneliti bernama *I.Rechenberg* mengembangkan sebuah pendekatan untuk menyelesaikan masalah komputasi dengan pendekatan evolusi (*evolutionary approach*) [3]. *Evolutionary approach* adalah sebuah teknik pendekatan yang terinspirasi dari proses evolusi dan seleksi alam. *Genetic algorithm* adalah teknik yang dikembangkan dari *evolutionary approach* dan cocok untuk menyelesaikan masalah kombinatorial. Cara kerja *genetic algorithm* secara umum adalah: dimulai dari pembangkitan beberapa benih (yang akan menjadi set solusi) yang dipasangkan dan menghasilkan set solusi baru, lalu set baru ini akan diseleksi dengan menghitung (*fitness value*) menggunakan (*fitness function*). Hal tersebut dilakukan berulang-ulang hingga mencapai generasi tertentu atau set solusi sudah memenuhi syarat minimum sebuah set solusi yang baik.

Pembangunan *test case* dapat dilakukan dengan menggunakan *genetic algorithm*. Pengujian perangkat lunak membutuhkan banyak kasus tes yang memiliki variasi luas, tetapi tetap dalam batasan dan ketentuan yang dibutuhkan untuk menjadi masukan program yang valid. *Genetic algorithm* cocok untuk masalah ini karena *genetic algorithm* dapat mencari set solusi terbaik dari banyak set solusi dengan variasi yang luas dengan batasan-batasan dan ketentuan tertentu.

1.2 Rumusan Masalah

Berdasarkan latar belakang, rumusan masalah penelitian adalah sebagai berikut:

- Bagaimana cara kerja dan implementasi algoritma genetik dalam melakukan pembangunan kasus tes?

- 1 – Bagaimana cara untuk mengetahui apakah set kasus tes sudah cukup baik untuk melaku-
2 kukan tes?

3 **1.3 Tujuan**

4 Berdasarkan identifikasi masalah, tujuan penelitian adalah sebagai berikut:

- 5 – Mengimplementasikan algoritma genetik dalam melakukan pembangkitan kasus tes.
6 – Melakukan pengujian set kasus tes untuk mengetahui apakah set kasus tes sudah cukup
7 baik untuk melakukan tes.

8 **1.4 Batasan Masalah**

9 Batasan-batasan masalah untuk karya ilmiah adalah sebagai berikut:

- 10 1. Pembangunan *control flow diagram* menggunakan kakas *TRGeneration*[4].
11 2. Kode program yang dipilih dibatasi dengan input integer karena keterbatasan kakas
12 *TRGeneration*[4] yang dipakai.
13 3. Metode *testing* menggunakan *white-box testing*.

14 **1.5 Metodologi Penelitian**

15 Metodologi yang digunakan dalam penyusunan skripsi adalah:

- 16 1. Studi literatur pada *genetic algorithm*.
17 2. Studi literatur pada topik pembangkitan kasus tes.
18 3. Mencari kode program untuk data kasus tes.
19 4. Merancang perangkat lunak yang akan dibuat.
20 5. Implementasi perangkat lunak dalam bahasa Java.
21 6. Melakukan pengujian perangkat lunak yang dibangun dengan kode program (*test case*
22 *data*).
23 7. Mengambil kesimpulan berdasarkan hasil pengujian.

24 **1.6 Sistematika Pembahasan**

25 Sistematika pembahasan dalam skripsi ini adalah:

- 26 – Bab 1: Penjelasan deskripsi umum skripsi melalui beberapa poin:
27 * Latar belakang permasalahan pembangunan kasus tes yang baik.
28 * Rumusan masalah akan persoalan.

- 1 * Tujuan yang ingin dicapai dalam skripsi ini.
- 2 * Metodologi penelitian yang menjelaskan alur skripsi.
- 3 – Bab 2: Dasar Teori yang menjelaskan dua teori yang diperlukan dalam skripsi ini. Per-
- 4 tama, membahas proses pembangunan sebuah perangkat lunak dari sisi "Rekayasa Per-
- 5 angkat Lunak". Kedua, membahas *Genetic Algorithm* yang merupakan cabang dari
- 6 *Evolutionary Computation*.
- 7 – Bab 3: Menjelaskan rincian kebutuhan perangkat lunak.
 - 8 * *control flow diagram generator* untuk membangun *control flow diagram*.
 - 9 * *Test requirement generator* untuk membangun *test requirement*.
 - 10 * *Genetic algorithm engine* yang merupakan bagian utama perangkat lunak untuk
 - 11 membangun set kasus tes yang baik. Menjelaskan kakas
 - 12 * *TRGeneration*[4] yang dipakai untuk membangun *control flow diagram*.
 - 13 * Menjelaskan pembangunan *file+temp* yang dibutuhkan dalam melakukan pengujian
 - 14 kasus tes.
- 15 – Bab 4: Menjelaskan perancangan antarmuka, rancangan kelas dengan diagram kelas,
- 16 diagram aktivitas dan penjelasan setiap *method* yang dibangun dan dibutuhkan. Untuk
- 17 perancangan antarmuka, dijelaskan struktur antarmuka dan fungsi dari bagian-bagian
- 18 antarmuka.
- 19 – Bab 5: Menjelaskan hasil implementasi antarmuka, hasil pengujian terhadap perangkat
- 20 lunak dan memaparkan analisis terhadap hasil pengujian.
- 21 – Bab 6: merupakan kesimpulan dan saran mengenai perangkat lunak.