PEMBANGKIT *TEST CASE* (KASUS UJI) MENGGUNAKAN MODEL UML (*UNIFIED MODELING LANGUAGE) SEQUENCE DIAGRAM* DENGAN METODE ALGORITMA GENETIKA

SKRIPSI

Oleh :
AINUN NOFIKA SARI
NIM. 15650017

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2021

PEMBANGKIT *TEST CASE* (KASUS UJI) MENGGUNAKAN MODEL UML (*UNIFIED MODELING LANGUAGE) SEQUENCE DIAGRAM* DENGAN METODE ALGORITMA GENETIKA

SKRIPSI

Diajukan kepada:
Universitas Islam Negeri (UIN) Maulana Malik Ibrahim Malang
Untuk Memenuhi Salah Satu Persyaratan Dalam
Memperoleh Gelar Sarjana Komputer (S.Kom)

Oleh :
AINUN NOFIKA SARI
NIM. 15650017

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2021

LEMBAR PERSETUJUAN

PEMBANGKIT *TEST CASE* (KASUS UJI) MENGGUNAKAN MODEL UML (*UNIFIED MODELING LANGUAGE) SEQUENCE DIAGRAM* DENGAN METODE ALGORITMA GENETIKA

SKRIPSI

Oleh :
AINUN NOFIKA SARI
NIM. 15650017

Telah Diperiksa dan Disetujui untuk Diuji

Tanggal : 21 Juni 2021

Dosen Pembimbing I	Dosen Pembimbing II
Fatchurrochman, M.Kom	M. Ainul Yaqin, M.Kom
NIP. 19700731 200501 1 002	NIP. 19761013 200604 1 004

Mengetahui,
Ketua Jurusan Teknik Informatika
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang

Dr. Cahyo Crysdian
NIP. 19740424 200901 1 008

HALAMAN PENGESAHAN

PEMBANGKIT *TEST CASE* (KASUS UJI) MENGGUNAKAN MODEL UML (*UNIFIED MODELING LANGUAGE) SEQUENCE DIAGRAM* DENGAN METODE ALGORITMA GENETIKA

SKRIPSI

Oleh :
AINUN NOFIKA SARI
NIM. 15650017

Telah Dipertahankan di Depan Dewan Penguji
dan Dinyatakan Diterima Sebagai Salah Satu Persyaratan
untuk Memperoleh Gelar Sarjana Komputer (S.Kom)
Pada Tanggal ……

Susunan Dewan Penguji			**Tanda tangan**
1. Penguji Utama	:	Supriyono, M.Kom NIP. 19841010 201903 1 012	( )
2. Ketua Penguji	:	Ajib Hanani, M.T NIDT. 19840731 20160801 1 076	( )
3. Sekretaris Penguji	:	Fatchurrochman, M.Kom NIP. 19700731 200501 1 002	( )
4. Anggota Penguji	:	M. Ainul Yaqin, M.Kom NIP. 19761013 200604 1 004	( )

Mengetahui,
Ketua Jurusan Teknik Informatika
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang

Dr. Cahyo Crysdian
NIP. 19740424 200901 1 008

PERNYATAAN KEASLIAN TULISAN

Saya yang bertanda tangan dibawah ini:

Nama : Ainun Nofika Sari

NIM : 15650017

Fakultas/Jurusan : Sains dan Teknologi/Teknik Infomatika

Judul Skripsi : Pembangkit *Test Case* (Kasus Uji) Menggunakan Model Uml (*Unified Modeling Language) Sequence Diagram* dengan Metode Algoritma Genetika

Menyatakan dengan sebenarnya bahwa Skripsi yang saya tulis ini benar-benar merupakan hasil karya sendiri, bukan merupakan pengambilalihan data, tulisan atau pikiran orang lain yang saya akui sebagai hasil tulisan atau pikiran saya sendiri, kecuali dengan mencantumkan sumber cuplikan pada daftar pustaka.

Apabila dikemudian hari terbukti atau dapat dibuktikan Skripsi ini hasil jiplakan, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Malang, 05 Mei 2021
Yang membuat pernyataan,

Ainun Nofika Sari
NIM. 15650017

HALAMAN MOTTO

“no pain no gain”

HALAMAN PERSEMBAHAN

الْحَمْدُ لِلَّهِ رَبِّ الْعَالَمِينَ

Puji syukur kehadirat Allah, sholawat dan salam bagi Rasul-Nya
Saya persembahkan karya ini kepada:

Kepada ibu dan bapak yang senantiasa memanjatkan do'anya kepada Allah s.w.t sampai akhirnya penulis sukses menyelesaikan perkuliahan S1 ini.

Dosen pembimbing saya, Bapak Fatchurrochman, M.Kom dan Bapak M. Ainul Yaqin, M.Kom dan seluruh dosen Teknik Informatika UIN Maulana Malik Ibrahim Malang.

Teman-teman yang telah berkenan membantu dan yang selalu memberikan semangat kepada saya beserta semua pihak yang telah membantu selama penelitian maupun penyusunan skripsi ini.

KATA PENGANTAR

بِسْمِ اللهِ الرَّحْمٰنِ الرَّحِيْمِ

Puji dan syukur penulis panjatkan ke hadirat Allah subhanahu wa ta'ala yang telah melimpahkan rahmat dan hidayahNya kepada kita, sehingga penulis bisa menyelesaikan skripsi, yang berjudul "Pembangkit *Test Case* (Kasus Uji) Menggunakan Model UML (*Unified Modeling Language) Sequence Diagram* dengan Metode Algoritma Genetika". Tujuan dari penyusunan skripsi ini guna memenuhi salah satu syarat untuk bisa menempuh ujian sarjana (S.Kom). Penulis menyadari bahwa dalam pengerjaan skripsi ini masih terdapat kekurangan dan kesalahan.

Dalam menyusun skripsi ini tidak lepas dari hambatan dan rintangan serta kesulitan-kesulitan, namun berkat bimbingan, bantuan, nasihat dan dorongan, serta saran-saran dari berbagai pihak, khususnya pembimbing, maka segala hambatan dan rintangan, serta kesulitan dapat teratasi dengan baik. Untuk itu pada kesempatan ini saya sampaikan terima kasih dengan tulus hati kepada yang terhormat:

1. Prof. Dr. Abdul Haris, M.Ag selaku Rektor Universitas Islam Negeri (UIN) Maulana Malik Ibrahim Malang.
2. Dr. Sri Harini, M.Si, selaku Dekan Fakultas Sains dan Teknologi Universitas Islam Negeri (UIN) Maulana Malik Ibrahim Malang.
3. Dr. Cahyo Crysdian, Selaku Ketua Jurusan Teknik Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri (UIN) Maulana Malik Ibrahim Malang.

4. Fatchurrochman, M.Kom, Selaku Dosen Pembimbing I yang telah membimbing dalam penyusunan skripsi ini hingga selesai.
5. M. Ainul Yaqin, M.Kom Selaku Dosen Pembimbing II yang telah membimbing dalam penyusunan skripsi ini hingga selesai.
6. Segenap Dosen, Laboran dan Admin Jurusan Teknik Informatika Universitas Islam Negeri Maulana Malik Ibrahim Malang yang telah bersedia mengamalkan ilmunya, membimbing dan memberikan pengarahan serta membantu selama proses perkuliahan.
7. Ibu dan bapak yang telah memberikan dukungan, restu serta selalu mendoakan pada setiap langkah penulis.
8. Teman-teman Teknik Informatika Interface 2015 yang senantiasa memberi motivasi dan berjuang bersama selama menjadi mahasiswa.
9. Semua pihak yang telah banyak membantu dalam penyusunan skripsi ini yang tidak bisa penulis sebutkan semuanya.

Penulis menyadari bahwa dalam penyusunan skripsi ini masih terdapat kekurangan dan penulis berharap semoga skripsi ini bisa memberikan manfaat kepada para pembaca khususnya bagi penulis secara pribadi.

Malang, 23 Juni 2021

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

ABSTRAK

Sari, Ainun Nofika. 2021. Pembangkit *Test Case* (Kasus Uji) Menggunakan Model UML (*Unified Modeling Language*) *Sequence Diagram* dengan Metode Algoritma Genetika. Skripsi. Jurusan Teknik Informatika Fakultas Sains Dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang.
Pembimbing: (I) Fatchurrochman, M. Kom
(II) M. Ainul Yaqin, M. Kom

Kata Kunci: Pembangkitan *Test Case*, Model UML *Sequence Diagram*, Algoritma Genetika

Pengujian perangkat lunak memainkan peran utama dalam pengembangan perangkat lunak karena ini bertanggung jawab atas sebagian besar biaya pengembangan apalagi jika pengujiannya dilakukan secara manual. Pengujian Perangkat Lunak membutuhkan proses yang memakan waktu yang lama dan juga biaya yang tidak sedikit dalam siklus kehidupan pengembangan perangkat lunak. Maka dari itu, diperlukan pembangkitan kasus uji secara otomatis dengan tujuan untuk menurunkan biaya pembangunan sistem perangkat lunak karena kesalahan bisa dihilangkan lebih awal dan juga dapat menghemat waktu selama proses pembangkitan kasus ujinya. Penelitian ini berupaya membuat pembangkit kasus uji menggunakan UML *Sequence Diagram* dengan menerapkan metode Algoritma Genetika. Metode ini digunakan untuk membangkitkan *test path* berdasarkan data uji yang ada. Adapun data uji yang digunakan yaitu Sistem Konsultasi Kesehatan atau *Medical Consultation System*, Sistem ATM atau ATM *System*, Sistem Aktivitas Keberangan Pesawat atau *Aircraft Departure Activity* dan Sistem Evaluasi atau Penilaian Pembelajaran sebagai studi kasusnya.

ABSTRACT

Sari, Ainun Nofika. 2021. Generating Test Case (Test Case) Using UML Model (Unified Modeling Language) Sequence Diagram with Genetic Algorithm Method. Thesis. Department of Informatics Engineering, Faculty of Science and Technology, Maulana Malik Ibrahim State Islamic University of Malang.
Supervisor: (I) Fatchurrochman, M. Kom
(II) M. Ainul Yaqin, M. Kom

Keywords: Generate Test Case, Model UML Sequence Diagram, Genetic Algorithm

Software testing plays a major role in software development as it is responsible for most of the development costs especially if the testing is done manually. Software Testing requires a process that takes a long time and also costs a lot in the software development life cycle. Therefore, it is necessary to generate test cases automatically in order to reduce the cost of building a software system because errors can be eliminated early and can also save time during the process of generating test cases. This study seeks to create a test case generator using the UML Sequence Diagram by applying the Genetic Algorithm method. This method is used to generate a test path based on existing test data. The test data used are Medical Consultation System, ATM System, Aircraft Departure Activity and Learning Assessment or Evaluation System as a case study.

مستخلص

ساري، عين نوفيكا. ٢٠٢١. إنشاء حالة الاختبار (حالة الاختبار) باستخدام نموذج UML (*Unified Modeling Language*) مخطط التسلسل مع طريقة الخوارزمية الجينية. قسم التقنية المعلوماتية. كلية العلوم والتكنولوجيا. جامعة مولانا مالك إبراهيم الإسلامية الحكومية مالانج.

المشرف : (I) فتح الرحمن، الماجستير

(II) عين اليقين، الماجستير

الكلمات المفتاحية : مولد حالة الاختبار ، مخطط التسلسل (UML) ، الخوارزمية الجينية.

كان اختبار البرامج دورًا رئيسيًا يلعب في تطوير البرامج لأنه مسؤول عن معظم تكاليف التطوير خاصةً إذا تم الاختبار يدويًا. يتطلب اختبار البرامج عملية تستغرق وقتًا طويلاً وتكلف أيضًا الكثير في دورة حياة تطوير البرامج. ولذلك، من الضروري إنشاء حالات اختبار تلقائيًا لتقليل تكلفة إنشاء نظام برمجي لأنه يمكن التخلص من الأخطاء مبكرًا ويمكن أيضًا توفير الوقت أثناء عملية إنشاء حالات الاختبار. تسعى هذه الدراسة إلى إنشاء مولد حالة اختبار باستخدام مخططات تسلسل UML من خلال تطبيق طريقة الخوارزمية الجينية. تُستخدم هذه الطريقة لإنشاء مسار اختبار بناءً على بيانات الاختبار الموجودة. بيانات الاختبار المستخدمة هي نظام الاستشارات الطبية ونظام الصراف الآلي ونشاط مغادرة الطائرات ونظام في تقييم التعلمية.

BAB I

PENDAHULUAN

1.1 Latar Belakang

Pengujian perangkat lunak dalam *Software Development Life Cycle* (SDLC) atau siklus hidup pengembangan perangkat lunak adalah proses mengeksekusi dan mengevaluasi suatu program dengan tujuan menemukan kesalahan (Myers, 1979). Secara umum, pengujian perangkat lunak ini berfokus pada aktivitas yang bertujuan mengevaluasi atribut atau kemampuan suatu program atau sistem dan memastikan bahwa program atau sistem tersebut memenuhi persyaratan spesifikasinya (Hetzel, 1988). Berdasarkan pernyataan (Myers, 1979), terdapat beberapa poin yang menjelaskan mengenai pengujian perangkat lunak, yaitu:

a. Pengujian merupakan proses eksekusi program dengan tujuan utama untuk mencari kesalahan
b. Sebuah kasus pengujian dikatakan baik jika memiliki kemungkinan penemuan kesalahan yang tinggi
c. Pengujian yang berhasil adalah pengujian yang menemukan kesalahan.

Pengujian perangkat lunak memainkan peran utama dalam pengembangan perangkat lunak karena ini bertanggung jawab atas sebagian besar biaya pengembangan apalagi jika pengujiannya dilakukan secara manual (Kansomkeat & Rivepiboon, 2003). Pengujian Perangkat Lunak membutuhkan proses yang memakan waktu yang lama dan juga biaya yang tidak sedikit dalam siklus kehidupan pengembangan perangkat lunak. Hal ini ditunjukkan oleh (Nguyen, 2012) dalam penelitiannya, bahwa berdasarkan laporan industri pengembangan

produk perangkat lunak, pengujian perangkat lunak mengonsumsi sekitar 10-25% dalam usaha pengembangan produk dan dalam beberapa proyek nilai ini dapat meningkat hingga 50%. Menurut Universitas Cambridge, pengeluaran tahunan global untuk pengujian melebihi $ 300 miliar. Pengujian perangkat lunak memakan 25-40% dari anggaran tipikal TI, mengarah ke 40% pada 2018 (Ariola & Dunlop, 2016).

Pengujian perangkat lunak dilakukan oleh seorang *developer* sebelum program atau sistem diberikan kepada *user*. Salah satu tahap yang penting dalam pengujian perangkat lunak adalah pembangkitan kasus uji. Pembangkitan kasus uji pada aplikasi berbasis objek adalah dengan memanfaatkan UML model. Diagram UML model digunakan untuk mengoreksi sebuah perancangan dapat pula digunakan untuk membangkitkan serangkaian kasus uji sehingga kasus uji dapat dihasilkan pada saat tahap *design* (Putri, Widowati, & Hakim, 2015).

Salah satu keuntungan membangkitkan kasus uji pada saat tahap spesifikasi dan desain adalah kasus uji dapat dibuat lebih awal dalam *Software Development Life Cycle* (SDLC) atau siklus hidup pengembangan perangkat dan siap digunakan sebelum program dibuat. Selain itu, ketika kasus pengujian dibuat lebih awal, maka *developer* akan lebih cepat dalam menemukan kesalahan atau ketidaksesuaian dalam spesifikasi dan desain. Maka dari itu, diperlukan pembangkitan kasus uji secara otomatis dengan tujuan untuk menurunkan biaya pembangunan sistem perangkat lunak karena kesalahan bisa dihilangkan lebih awal dan juga dapat menghemat waktu selama proses pembangkitan kasus ujinya (Prasanna, dkk, 2005). Otomatisasi pada fase ini dapat mengatasi masalah di atas dan mengurangi tenaga yang dikeluarkan oleh *developer* dan juga membantu

dalam mendeteksi kesalahan yang mungkin terjadi dan kesalahan logis juga (Panthi & Mohapatra, 2012).

Firman Allah dalam *Al-qur'an* yang berbunyi:

وَاذْكُرْ عَبْدَنَا أَيُّوبَ إِذْ نَادَى رَبَّهُ أَنِّي مَسَّنِيَ الشَّيْطَانُ بِنُصْبٍ وَعَذَابٍ(٤١) ارْكُضْ بِرِجْلِكَ هَذَا مُغْتَسَلٌ بَارِدٌ وَشَرَابٌ (٤٢) وَوَهَبْنَا لَهُ أَهْلَهُ وَمِثْلَهُمْ مَعَهُمْ رَحْمَةً مِنَّا وَذِكْرَى لأُولِي الأَلْبَابِ (٤٣) وَخُذْ بِيَدِكَ ضِغْثًا فَاضْرِبْ بِهِ وَلا تَحْنَثْ إِنَّا وَجَدْنَاهُ صَابِرًا نِعْمَ الْعَبْدُ إِنَّهُ أَوَّابٌ(٤٤)

Artinya: *"Dan ingatlah akan hamba Kami Ayyub ketika ia menyeru Tuhannya, "Sesungguhnya aku diganggu setan dengan kepayahan dan siksaan."(Allah berfirman), "Hantamkanlah kakimu; inilah air yang sejuk untuk mandi dan untuk minum." Dan Kami anugerahi dia (dengan mengumpulkan kembali) keluarganya dan (Kami tambahkan) kepada mereka sebanyak mereka pula sebagai rahmat dari Kami dan pelajaran bagi orang-orang yang mempunyai pikiran. Dan ambillah dengan tanganmu seikat (rumput), maka pukullah dengan itu dan janganlah kamu melanggar sumpah. Sesungguhnya Kami dapati dia (Ayyub) seorang yang sabar. Dialah sebaik-baik hamba. Sesungguhnya dia amat taat (kepada Tuhannya)" (QS. Sad 38:41-44).*

Dalam kehidupan ini, Allah memberi ujian kepada manusia untuk mengetahui tingkat kesabaran, baik dan buruknya manusia tersebut. Ujian yang diberikan Allah kepada manusia ada banyak bentuknya, diantaranya yaitu ujian keimanan, ujian harta, dan ujian kesehatan. Seperti halnya kisah Nabi Ayyub as. diuji Allah dengan ujian yang teramat berat. Nabi Ayyub as. diuji hartanya, keluarganya, dan kesehatannya, namun Nabi Ayyub as. tetap bersabar. Hal ini membuktikan bahwa Nabi Ayyub adalah hamba Allah yang baik, taat dan patuh dalam apapun yang sudah ditetapkan oleh Allah swt. Sama halnya dengan tema

penelitian yang sedang penulis kerjakan, yaitu pengujian. Pengujian dibuat agar penguji bisa mengetahui seberapa kualitasnya suatu perangkat lunak. Perangkat lunak diuji untuk mengetahui seberapa baik kualitasnya. Pada pengujian, jika pengujian dilakukan dengan baik maka kualitas yang ada pada perangkat lunak juga baik.

Sesuai dengan penjelasan diatas, penulis membuat pembangkit kasus uji menggunakan UML *Seqeunce Diagram* dengan menerapkan metode algoritma genetika. *Unified Modeling Language* (UML) adalah himpunan struktur dan teknik untuk pemodelan rancangan perangkat lunak berorientasi objek. UML merupakan salah satu dari model yang banyak dipakai oleh para pengembang. Hal ini dikarenakan UML dapat dengan mudah dimengerti oleh pengembang. UML yang dibuat berdasar pada perspektif dari pengguna, sehingga mempermudah proses verifikasinya (Hasling, 2008). Penulis menerapkan metode algoritma genetika dalam penelitian ini. algoritma genetika dapat menyelesaikan masalah optimisasi ketika metode tradisional diasumsikan terlalu banyak memakan waktu pemrosesan untuk mendapatkan keefektifan outputnya. John Koza mengunakan algoritma genetika dalam pemrograman yang disebut dengan *Genetic Programming* (GP) untuk melakukan tugas-tugas tertentu secara efektif. *Genetic Programming* dapat digunakan di beberapa aplikasi dan bidang yang berbeda. Secara khusus, *Genetic Programmin*g ini digunakan untuk menyelesaikan beberapa jenis masalah optimisasi (Alsmadi, 2010).

1.2 Rumusan Masalah

Sesuai dengan latar belakang yang telah diuraikan, maka rumusan masalah yang diambil pada penelitian ini adalah bagaimana pembangkitan kasus uji

secara otomatis menggunakan model *Unified Modeling Language* (UML) *Sequence Diagram* dengan menerapkan algoritma genetika?

1.3 Tujuan Penelitian

Tujuan penelitian ini adalah untuk membangkitkan kasus uji (*test case*) secara otomatis pada UML *Unified Modeling Language* (UML) *Sequence Diagram* menggunakan metode algoritma genetika.

1.4 Batasan Penelitian

Batasan dalam penelitian ini adalah sebagai berikut:

a. Pembuatan *Sequence diagram* dilakukan dengan menggunakan aplikasi *Rational Rose*
b. Data uji pengujian yang digunakan penulis yaitu *sequence diagram* pada jurnal Priya et al (2013), jurnal milik Shanti et al (2012) dan jurnal dari Rhmann (2016). Kemudian menambahkan Sistem Evaluasi atau Penilaian Pembelajaran sebagai studi kasus penelitian.
c. Pembangkitan kasus uji atau *test case* menggunakan aplikasi *Netbeans*.

1.5 Manfaat Penelitian

Pada penelitian ini diharapkan bisa menghemat waktu selama proses pembangkitan kasus uji atau *test case* untuk pengujian perangkat lunak.

1.6 Sistematika Penulisan

Ditujukan untuk memberikan gambaran garis besar dan juga uraian pada penulisan skripsi ini yang terdiri sebagai berikut:

BAB I : PENDAHULUAN

Menjelaskan latar belakang dari permasalahan, rumusan masalah, tujuan penelitian, batasan penelitian, manfaat penelitian dan sistematika penulisan.

BAB II : TINJAUAN PUSTAKA

Tinjauan pustaka menjelaskan mengenai teori-teori dan referensi yang berhubungan dengan permasalahan penelitian sebagai parameter rujukan untuk penelitian ini.

BAB III : ANALISIS DAN PERANCANGAN SISTEM

Analisis dan perancangan sistem menguraikan mengenai analisis kebutuhan dan perancangan sistem pembangkit *test case* dengan model UML *Sequence diagram*.

BAB IV : HASIL DAN PEMBAHASAN

Hasil dan pembahasan menguraikan mengenai hasil uji coba pengujian dan membahas hasil uji coba yang telah dilakukan secara rinci.

BAB V : PENUTUP

Penutup menguraikan mengenai kesimpulan yang telah diperoleh berdasarkan hasil pembangkitan kasus uji atau *test case* pada perangkat lunak serta saran untuk penelitian selanjutnya.

BAB II

TINJAUAN PUSTAKA

2.1 Penelitian Terkait

Terdapat beberapa hasil penelitian yang sudah dilakukan sebelumnya yang terkait dengan penelitian pada skripsi ini. Berikut ini adalah paragraf yang menjelaskan mengenai penelitian-penelitian yang sudah pernah dikerjakan.

(Priya et al, 2013) Melakukan pengujian berbasis model dari test path yang mana diotomatisasi dan diperoleh sebelum atau selama proses pengembangan. Kasus uji dapat dibangkitakn lebih awal untuk membantu dalam memperbaiki kesalahan pada tahapan awal pembuatan perangkat lunak. Penelitian ini menggunakan diagram *Unified Modeling Language* (UML) *Sequence Diagram* dan *Medical Consultation System* sebagai studi kasusnya. Tahap awal yang dikerjakan yaitu membangun *Sequence Diagram* untuk *Medical Consultation System* yang dirancang menggunakan *Rational Rose*, produk IBM dan disimpan dengan ekstensi sebagai .mdl. Kemudian melakukan *Parsing file* .mdl menggunakan java swing untuk mendapatkan informasi yang diperlukan yang akan digunakan untuk menghasilkan SDT, yaitu *Sequence Dependency Table.* Dari SDT yang dihasilkan, lantas menggunakan objek sebagai node untuk membuat *Sequence Dependency Graph* (SDG). Tahapan terakhir yaitu mengaplikasikan metode *Depth First Search* (DFS) untuk mendapatkan jalur pengujian. Penelitian ini membantu dalam menghasilkan *test path* yang dapat diproduksi oleh penguji perangkat lunak selama pengujian untuk memeriksa apakah sistem berfungsi sebagaimana semestinya. Penelitian ini juga membantu pengembang untuk secara signifikan meningkatkan kualitas desain. Penguji dapat

menemukan kesalahan dalam sistem lebih awal dan mencegah kesalahan-kesalahan pada sistem disebarluaskan ke fase-fase berikutnya.

(Shanti et al, 2012) Melakukan pengujian menggunakan diagram *Unified Modeling Language* (UML) *Sequence Diagram* dan menerapkan Algoritma Genetika sebagai metodenya untuk mendapat kasus uji terbaik. Studi kasus pada penelitian ini adalah *ATM System*. Penelitian ini bertujuan untuk menguji perangkat lunak lebih awal sehingga memudahkan bagi penguji agar bisa menguji perangkat lunak pada tahap selanjutnya. *Test case* ini penting untuk mengidentifikasi lokasi kesalahan dalam implementasi sehingga mengurangi upaya pengujian. Selain itu, metode yang digunakan untuk pembuatan kasus uji (*test case*) ini menginspirasi pengembang untuk meningkatkan kualitas desain, menemukan kesalahan dalam implementasi lebih awal, dan mengurangi waktu pengembangan perangkat lunak. *Test case* ini akan mengurangi biaya pengembangan perangkat lunak dan meningkatkan kualitas perangkat lunak. Hasil percobaan pada penelitian ini yaitu bahwa metode yang digunakan ini dapat bekerja dengan baik.

(Putri, 2015) Melakukan pengujian berbasis objek dengan memanfaatkan salah satu model pada UML yaitu *Sequence Diagram*. Kasus uji yang dibuat menggunakan *Sequence Diagram* ini akan dihasilkan pada tahap *design* sehingga pada saat tahap *coding* selesai, sistem *test case* ini sudah siap untuk melakukan pengujian yang dilakukan dengan *automatic testing* menggunakan *Selenium WebDriver*. Tahapan pertama dalam pembangkitan kasus uji ini adalah membuat *Sequence Diagram* dalam bentuk *file* XMI. Pengubahan *sequence diagram* menjadi bentuk XMI dilakukan dengan IBM Rasional. Kemudian *Sequence*

diagram dalam *file* XMI yang telah dipilih kemudian diubah menjadi *GraphXML* tujuannya adalah untuk mempermudah melihat dan memvisualisasikan sebuah *graph* yang merepresentasikan *sequence diagram. Sequence diagram* dengan format *GraphXML* kemudian diubah menjadi *Sequence Dependency Graph.* Pengubahan dilakukan bertujuan untuk mempermudah menerapkan algoritma untuk menghasilkan sekumpulan kasus uji. Bentuk SDG digunakan untuk membangkitkan kasus uji. Kasus Uji yang dihasilkan merupakan sebuah skenario yang digunakan untuk menguji alur dari sebuah proses yang ada pada aplikasi. Skenario Kasus uji tersebut didapatkan dengan mengadopsi algoritma *depth first search*. Mengadopsi disini adalah menggunakan algoritma *depth first search* namun melakukan perubahan pada saat implementasi.

2.2 Landasan Teori

Landasan teori merupakan teori dalam sebuah penelitian yang menjabarkan tentang variabel yang akan diteliti. Beberapa cakupan yang dibahas dalam landasan teori diantaranya definisi dan metode yang digunakan dalam penelitian. Sehingga, landasan teori sangat penting dalam sebuah penelitian sebagai hipotesis atau jawaban sementara terhadap penelitian yang sedang dikerjakan.

2.2.1 UML (*Unified Modeling Language*)

Unified Modelling Language (UML) adalah sebuah "bahasa" yang telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. Dengan menggunakan UML dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut

dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan *operation* dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa berorientasi objek seperti C++, Java, C# atau VB.NET. Walaupun demikian, UML tetap dapat digunakan untuk *modeling* aplikasi prosedural dalam VB atau C. Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan *syntax*/semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap bentuk memiliki makna tertentu, dan UML *syntax* mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya: Grady Booch OOD (*Object-Oriented Design*), Jim Rumbaugh OMT (*Object Modeling Technique*), dan Ivar Jacobson OOSE (*Object-Oriented Software Engineering*) (Dharwiyanti & Wahono, 2003).

Firman Allah dalam *Al-qur'an* yang berbunyi:

لَقَدْ خَلَقْنَا ٱلْإِنسَٰنَ فِىٓ أَحْسَنِ تَقْوِيمٍ (٤)

Artinya: *"sesungguhnya Kami telah menciptakan manusia dalam bentuk yang sebaik-baiknya" (QS. At-Tin 95:4).*

Dalam Tafsir Ibnu Katsir Jilid 8 Firman Allah Ta'ala **فِىٓ أَحْسَنِ تَقْوِيمٍ لَقَدْ خَلَقْنَا ٱلْإِنسَٰنَ** *"sesungguhnya Kami telah menciptakan manusia dalam bentuk yang sebaik-baiknya"* Dan inilah yang menjadi obyek sumpah, yaitu bahwa Allah SWT telah menciptakan manusia dalam wujud dan bentuk yang sebaik-baiknya, dengan perawakan yang sempurna serta beranggotakan badan yang normal. Begitu pula dalam membangun sebuah perangkat lunak, diperlukan suatu perancangan yang matang agar menjadi perangkat lunak yang memiliki kualitas

yang baik dan tentunya dapat berfungsi sebagaimana yang sudah direncanakan. UML merupakan model perancangan yang digunakan dalam membangun perangkat lunak. Terdapat delapan diagram UML menurut (Dharwiyanti & Wahono, 2003), yaitu sebagai berikut :

a. *Use Case Diagram*
b. *Class Diagram*
c. *Statechart Diagram*
d. *Activity Diagram*
e. *Sequence Diagram*
f. *Collaboration Diagram*
g. *Component Diagram*
h. *Deployment Diagram*

Konsep dasar UML bisa dirangkum seperti gambar dibawah:

Tabel 2. 1 Konsep Dasar UML (Dharwiyanti & Wahono, 2003)

Major Area	***View***	***Diagrams***	***Main Concepts***
Structural	*Static view*	*Class Diagram*	*Class, association, generalization, dependency, realization, interface*
	Use case view	*Use Case Diagram*	*Use case, actor, association, extend, include, use case generalization*
	Implementation view	*Component Diagram*	*Component, interface, dependency, realization*
	Deployment view	*Deployment Diagram*	*Node, componenet, dependency, location*
dynamic	*State machine view*	*Statechart Diagram*	*State, event. Transition, action*
	Activity view	*Activity Diagram*	*State, activity, completion transition,*

			fork, join
	Interaction view	*Sequence Diagram*	*Interaction, object, message, activation*
		Collaboration Diagram	*Collaboration, interaction, collaboration role, message*
Model management	*Model management view*	*Class Diagram*	*Package, subsystem, model*
extensibility	*all*	*all*	*Constraint, stereotype, tagged values*

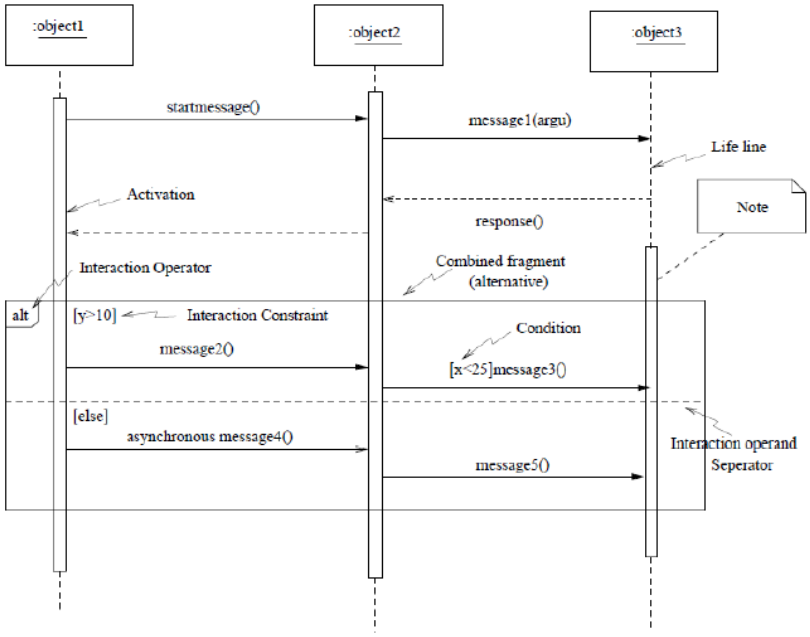
2.2.2 *Sequence Diagram*

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). *Sequence diagram* biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respon dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan. Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal. *Message* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, *message* akan dipetakan menjadi operasi/metode dari *class*. *Activation bar* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah *message*. Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan *icon* khusus untuk objek *boundary*, *controller dan persistent entity* (Dharwiyanti & Wahono, 2003).

Menurut (Putri, Widowati, & Hakim, 2015) Sequence diagram mendeskripsikan suatu interaksi antara objek selama beberapa masa. Diagram urutan atau disebut *sequence diagram* biasanya merekam perilaku skenario. Grafik ini menunjukkan jumlah objek dan pesan yang dikirimkan antar objek dalam skenario. Beberapa simbol yang digunakan untuk membuat diagram urutan atau *sequence diagram*, seperti:

a. Objek yaitu *instance* kelas ditulis secara horizontal.
b. Pesan yang ditemukan yaitu suatu pesan yang dapat menstimulus terjadinya sebuah skenario.
c. Bar Aktifasi, setiap *lifeline* itu menunjukkan waktu suatu objek yang aktif dalam suatu interaksi, *activation bar* atau bar aktifasi ini juga berhubungan dengan fungsi dari *instance* yang ada di dalam *stack*.
d. *Lifeline* adalah jalur kehidupan objek dari kelas tertentu. *Lifeline* ini digunakan agar bisa mengetahui kapan suatu objek itu aktif dan dihapus, dan untuk linierisasi urutan pemanggilan pesan untuk setiap objek.
e. Pesan asinkron dan sinkron yaitu komunikasi diantara peran.
f. Penggunaan Interaksi yaitu gambaran interaksi yanga ada dalam arti dari interaksi lain.
g. Pecahan, adalah suatu alur pada sebuah diagram urutan.

Gambar dibawah ini adalah contoh dari model UML *sequence diagram:*



Gambar 2. 1 Gambar *Sequence Diagram* (Panthi & Mohapatra, 2012)

2.2.3 *Software Testing*

Pengujian perangkat lunak (*Software Testing*) adalah salah satu teknik utama untuk mencapai perangkat lunak berkualitas tinggi. *Software testing* dilakukan untuk mendeteksi adanya kesalahan yang menyebabkan kegagalan pada perangkat lunak. Namun, pada pengujian membutuhkan banyak waktu dan biaya yang besar. Mengkonsumsi hampir 50% dari sumber daya pengembangan sistem perangkat lunak (Srivastava & Kim, 2009). *Software Testing* merupakan optimisasi masalah secara obyektif yang mana usaha yang dikeluarkan untuk melakukan pengujian harus diminimalkan dan jumlah kesalahan yang dideteksi pada saat pengujian harus dimaksimalkan. Pengujian perangkat lunak dianggap proses yang paling memakan banyak aktivitas dalam pengembangan perangkat lunak (Singh, Kaur, & Kumar, 2012). Pengujian perangkat lunak dapat dilakukan secara manual atau secara otomatis dengan menggunakan alat pengujian. Alat

pengujian otomatis dapat melakukan kinerja yang lebih baik dari pada melakukan pengujian secara manual (Sabharwal, Sibal, & Sharma, 2011).

Secara umum, tujuan pengujian perangkat lunak adalah untuk merancang jumlah minimal *test case* sehingga dapat menemukan kesalahan dalam perangkat lunak dalam jumlah yang banyak. Sebagaimana disebutkan sebelumnya, pengujian perangkat lunak adalah pekerjaan yang panjang dan memakan banyak waktu jika dilakukan manual. Sedangkan pengujian perangkat lunak secara otomatis dapat dengan signifikan mempersingkat waktu dan mengurangi biaya pengembangan perangkat lunak. Manfaat lainnya dari *software testing* secara otomatis adalah pengujian akan berjalan sangat cepat dan hasil yang didapatkan akan lebih baik. Namun, pengujian perangkat lunak secara otomatis bukanlah proses yang mudah. Selama bertahun-tahun, banyak peneliti telah mengusulkan berbagai metode untuk menghasilkan data uji secara otomatis, yaitu metode yang berbeda untuk mengembangkan data uji/generator kasus. Perkembangan teknik itu akan juga mendukung otomatisasi pengujian perangkat lunak yang akan menghasilkan penghematan biaya yang signifikan. Penerapan teknik kecerdasan buatan (AI) dalam Rekayasa Perangkat Lunak (RPL) adalah suatu bidang penelitian yang sedang berkembang yang menghasilkan persilangan ide pada dua domain (Srivastava & Kim, 2009).

2.2.4 *Test Case*

Kasus uji yang baik harus memiliki kualitas yang mencakup lebih banyak fitur dari kriteria pengujian. Dengan kata lain, efektivitas proses pengujian bergantung pada kualitas kasus uji bukan kuantitas kasus uji. Kasus uji yang baik juga akan mendapatkan jumlah kasus uji yang tepat (atau optimal) dengan kualitas

yang lebih baik dan menghilangkan kasus uji yang berlebihan (atau tidak perlu). Selain itu, permasalahan memakan waktu dalam fase pengujian dapat dikurangi. Tetapi mendapatkan semua kasus uji dalam waktu yang singkat (waktu untuk memberikan perangkat lunak kepada klien) adalah tugas yang rumit. Oleh karena itu, pembuatan kasus uji otomatis mengurangi upaya penguji dan pengembang begitu juga dengan biaya dan waktu (Biswal, 2010).

2.2.5 *Automatic Test Case*

Bagian yang penting dari kasus uji adalah untuk menentukan *output* atau hasil yang diharapkan. Jadi *test case* yang khas harus memiliki dua komponen yaitu:

a. *Input* data ke program.
b. Deskripsi *output* yang benar dari set *input* data.

Pembangkitan kasus uji adalah untuk memenuhi persyaratan pengujian. Banyak peneliti yang berfokus pada otomatisasi pembuatan kasus uji ini dan hasil yang dilaporkan terdiri dengan berbagai tingkat keberhasilan. Mereka telah menggunakan berbagai metodologi dan desain yang berbeda dari sistem yang sedang diuji untuk pembuatan kasus uji otomatis. Kasus uji otomatis akan mengambil desain sebagai inputnya, memprosesnya dan kemudian menghasilkan spesifikasi uji berdasarkan kriteria pengujian tertentu yang disebut dengan kriteria cakupan uji.

Selanjutnya, data uji yang tepat untuk setiap spesifikasi tes ditentukan untuk membentuk kasus uji. Pengujian perangkat lunak dilakukan secara berurutan dengan menghasilkan *test case* untuk *set input* data dan membangun kepercayaan pengembang.

Kasus uji dapat berasal dari persyaratan dan spesifikasi, desain, atau *source code*. Kasus uji biasanya dirancang berdasarkan *source code* program. Hal ini membuat generasi kasus uji sulit terutama untuk pengujian di tingkat klaster. Pembuatan kasus uji dari desain memiliki keuntungan tambahan yang memungkinkan *test case* tersedia pada awal siklus pengembangan perangkat lunak, sehingga membuat perencanaan pengujian lebih efektif. Keuntungan lain dari pengujian berbasis desain adalah untuk menguji kesesuaian implementasi dengan desain yang sudah dirancang. Pembuatan kasus uji secara manual memakan waktu dan melelahkan. Oleh karena itu sering diperlukan pembuatan *test case* secara otomatis atau semi-otomatis dari desain (Biswal, 2010)

2.2.6 XML

Menurut Hunter et al. (2007), *Extensible Markup Language* (XML) merupakan teknologi dengan aplikasi dunia nyata, khususnya untuk manajemen, tampilan, dan organisasi data. XML bekerja dengan tujuan *markup* dari setiap jenis data tetapi dengan kompleksitas yang dieliminasi, XML tidak benar–benar merupakan bahasa, tetapi lebih pada sintaks yang digunakan untuk menjelaskan markup lain. Secara sederhana XML adalah suatu bahasa yang digunakan untuk mendeskripsikan dan memanipulasi dokumen secara terstruktur, serta menyediakan format tertentu untuk dokumen-dokumen yang mempunyai data terstruktur.

Keunggulan XML dapat diringkas sebagai berikut :

a. Pintar (*Intelegence*), XML dapat menangani berbagai level kompleksitas.
b. Dapat beradaptasi untuk membuat bahasa sendiri, seperti *Mircrosoft* membuat bahasa MSXML.

c. Mudah dalam pemeliharaan.

d. XML lebih sederhana jika dibandingkan bahasa markup lainnya seperti *Standart Generalized Markup Language* (SGML). Namun teknologi XML dikembangkan dengan mengadopsi bagian paling penting SGML dan dengan berpedoman pada pengembangan HTML.

e. XML dapat dengan mudah dipindah – pindahkan atau portabilitas.

f. XML dapat memungkinkan pertukaran informasi atau data antar perangkat (*server, PCs, smart device*, aplikasi, dan situs web). Data ini akan menjadi independen (*unlocked*) sehingga memudahkannya untuk diorganisir, diprogram, dan diubah, dan ditukar antar situs web atau aplikasi apa saja. XML merubah cara kita berpikir untuk mengembangkan suatu *software* terutama aplikasi web.

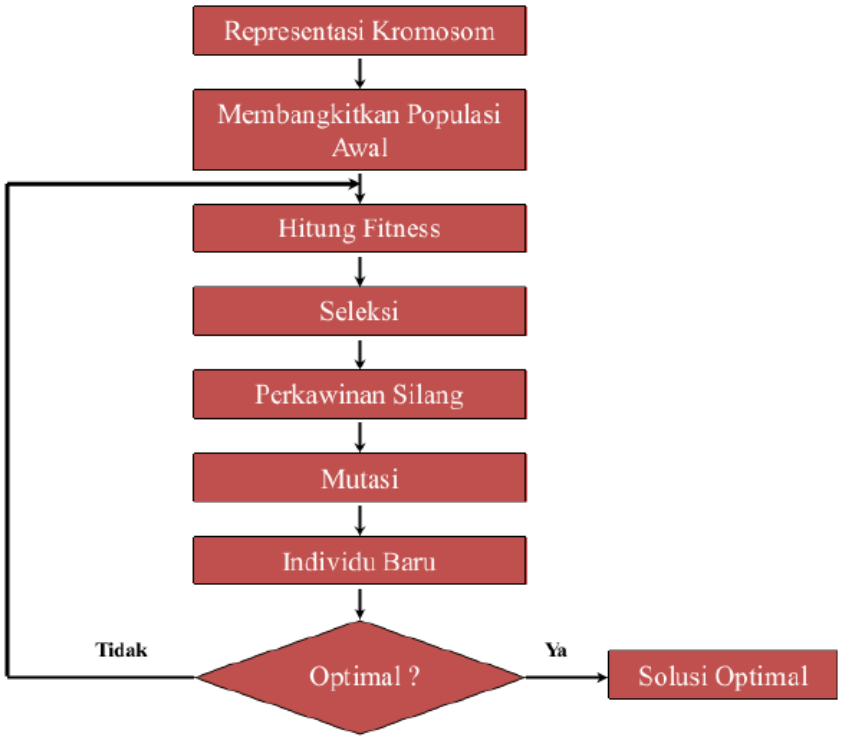
2.2.7 Algoritma Genetika

Algorima genetika diciptakan oleh John Holland dan murid-murid serta rekan-rekannya di College of Michigan. Algoritma Genetika mereplikasi proses evolusi untuk menangani masalah optimisasi uji kasus perangkat lunak. Algoritma genetika menjalankan semua proses dengan lebih produktif bagi yang tidak memiliki strategi positif dan batasan waktu yang terbatas. Algoritma genetika mungkin bukan teknik terbaik untuk tugas apa pun namun kuat dan cukup sesuai untuk tugas pengoptimalan yang lebih rumit, terutama untuk komputasi daya tinggi (Goyal, Mishra, Lamichhane, & Gandhi, 2018). Algoritma genetika biasanya adalah teknik pencarian yang digunakan dalam menghitung tepat atau perkiraan solusi untuk optimasi dan masalah pencarian dari berbagai domain, termasuk sains, perdagangan dan teknik. Alasan utama kesuksesan mereka adalah

penerapannya yang luas dan kemudahan penggunaan, yang juga menawarkan pencarian non-linear yang kuat yang melibatkan variabel besar. Namanya menunjukkan prinsip kerjanya, yang didasarkan pada konsep evolusi dalam sistem biologis. Dalam algoritma genetika, solusi kandidat dikodekan menggunakan kromosom. Algoritma kemudian mencari solusi yang lebih baik di antara sejumlah kromosom (kandidat solusi) atau juga disebut populasi solusi, berdasarkan pada prinsip *survival of the fittest* (juga disebut evolusi). Evolusi didasarkan pada dua operator utama: mutasi dan *crossover*. *Crossover* melibatkan pertukaran segmen antara dua kromosom sedangkan mutasi digunakan untuk sedikit mengubah kromosom. Mutasi pada dasarnya bermanfaat untuk mempertahankan keragaman dari satu generasi ke generasi populasi kromosom berikutnya (Biswal, 2010). Berikut ini adalah *pseudo code* untuk memahami konsep dengan lebih baik. Di sini t adalah jumlah generasi dan P populasi.

```
begin
      t=1;
      initialize P(t);
      While not finished
            evaluate P(t);
            select P(t+1) from P(t);
            Recombine P(t+1) using
                  crossover;
                  mutation;
            survive;
            t=t+1;
end;
```

Algoritma genetika memiliki struktur umum seperti pada gambar berikut.



Gambar 2. 2 Struktur Algoritma Genetika (Firmansyah, Ahmad, & Agustin, 2012)

Penjelasan struktur algoritma genetika:

a. Kromosom, individu yang terdapat dalam satu populasi dan merupakan suatu solusi yang masih berbentuk simbol.

b. Populasi, istilah pada teknik pencarian yang dilakukan sekaligus atas sejumlah kemungkinan solusi. Ukuran populasi tergantung pada permasalahan yang akan dipecahkan dan jenis operator genetika yang akan diimplementasikan. Setelah ukuran populasi telah ditentukan, kemudian harus dilakukan inisialisasi terhadap kromosom yang terdapat pada populasi tersebut. Inisialisasi kromosom dapat dilakukan secara acak, namun demikian harus tetap memperhatikan domain solusi dan kendala permasalahan yang ada.

c. Fungsi *Fitness*, alat ukur yang digunakan untuk proses evaluasi kromosom. Nilai *fitness* dari suatu kromosom akan menunjukkan kualitas kromosom dalam populasi tersebut. Setiap permasalahan memiliki nilai fitness yang berbeda. Pada penelitian ini rumus fitness yang digunakan berdasarkan referensi dari (Sabharwal, Sibal, & Sharma, 2011) yaitu:

$$\boldsymbol{FX} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{w_i}$$

Dimana $\boldsymbol{FX}$ adalah fitness value yang akan dicari dan $\boldsymbol{w_i}$ adalah bobot dari node pada graph. Jadi untuk mencari fitness nya yaitu dengan menjumlahkan bobot dari tiap-tiap node.

d. Seleksi, Memiliki tujuan untuk memberikan kesempatan reproduksi yang lebih besar bagi anggota populasi yang paling fit. Seleksi akan menentukan individu-individu mana saja yang akan dipilih untuk dilakukan rekombinasi dan bagaimana *offspring* terbentuk dari individu-individu terpilih tersebut. Langkah pertama yaitu pencarian nilai fitness. Langkah kedua adalah nilai fitness yang diperolah digunakan pada tahap-tahap seleksi selanjutnya. Ada beberapa metode seleksi dari induk, yaitu:

1. *Rank-based fitness assignment*

   Populasi diurutkan menurut nilai objektifnya. Nilai fitness dari tiap-tiap individu hanya tergantung pada posisi individu tersebut dalam urutan, dan tidak dipengaruhi oleh nilai objektifnya.

2. *Roulette wheel selection*

   Istilah lainnya adalah stochastic sampling with replacement. Individu-individu dipetakan dalam suatu segmen garis secara berurutan sedemikian hingga tiap-tiap segmen individu memiliki ukuran yang sama

dengan ukuran fitnessnya. Sebuah bilangan random dibangkitkan dan individu yang memiliki segmen dalam kawasan segmen dalam kawasan bilangan random tersebut akan terseleksi. Proses ini berulang hingga didapatkan sejumlah individu yang diharapkan.

3. *Stochastic universal sampling*

   Memiliki nilai bias nol dan penyebaran yang minimum. Individu-individu dipetakan dalam suatu segmen garis secara berurut sedemikian hingga tiap-tiap segmen individu memiliki ukuran yang sama dengan ukuran fitnessnya seperti halnya pada seleksi roda roulette. Kemudian diberikan sejumlah pointer sebanyak individu yang ingin diseleksi pada garis tersebut. Andaikan N adalah jumlah individu yang akan diseleksi, maka jarak antar pointer adalah 1/N, dan posisi pointer pertama diberikan secara acak pada range [1, 1/N].

4. *Local Selection*

   Setiap individu yang berada di dalam konstrain tertentu disebut dengan nama lingkungan lokal. Interaksi antar individu hanya dilakukan di dalam wilayah tersebut. Lingkungan tersebut ditetapkan sebagai struktur dimana populasi tersebut terdistribusi. Lingkungan tersebut juga dapat dipandang sebagai kelompok pasangan-pasangan yang potensial. Langkah pertama yang dilakukan adalah menyeleksi separuh pertama dari populasi yang berpasangan secara *random*. Kemudian lingkungan baru tersebut diberikan pada setiap individu yang terseleksi. Struktur lingkungan pada seleksi lokal dapat berbentuk : linear (*full ring* dan *half ring*), dimensi-2 (*full cross* dan *half cross*, *full star* dan *half star*), dan

dimensi-3 dan struktur yang lebih kompleks yang merupakan kombinasi dari kedua struktur diatas. Jarak antara individu dengan struktur tersebut akan sangat menentukan ukuran lingkungan. Individu yang terdapat dalam lingkungan dengan ukuran yang lebih kecil, akan lebih terisolasi dibandingkan dengan individu yang terletak pada lingkungan dengan ukuran yang lebih besar.

5. *Truncation selection*

   Merupakan seleksi buatan yang digunakan oleh populasi yang jumlahnya sangat besar. Individu-individu diurutkan berdasarkan nilai fitnessnya. Hanya individu yang terbaik saja yang akan diseleksi sebagai induk. Parameter yang digunakan adalah suatu nilai ambang *trunc* yang mengindikasikan ukuran populasi yang akan diseleksi sebagai induk yang berkisar antara 50% -10%. Individu-individu yang ada dibawah nilai ambang tidak akan menghasilkan keturunan.

6. *Tournament selection*

   Ditetapkan suatu nilai *tour* untuk individu-individu yang dipilih secara *random* dari suatu populasi. Individu-individu yang terbaik dalam kelompok ini akan diseleksi sebagai induk. Parameter yang digunakan adalah ukuran *tour* yang bernilai antara 2 sampai N (jumlah individu dalam populasi).

e. Crossover, Generasi berikutnya dikenal dengan anak *(offspring*) yang terbentuk dari gabungan dua kromosom generasi sekarang yang bertindak sebagai induk *(parent*) dengan menggunakan operator penyilang *(crossover*).

f. Mutasi, operator untuk memodifikasi kromosom.

Keuntungan penggunaan algoritma genetika terlihat dari kemudahan implementasi dan kemampuannya untuk menemukan solusi yang optimal dan bisa diterima secara cepat untuk masalah-masalah berdimensi tinggi. Efisien dan kuat untuk menyelesaikan permasalahan dalam spektrum yang luas, mendapatkan solusi optimal dari pertukaran informasi secara random dan memiliki kemampuan mengeksploitasi dengan cara yang efisien. Algoritma Genetika sangat berguna dan efisien untuk masalah dengan karakteristik sebagai berikut :

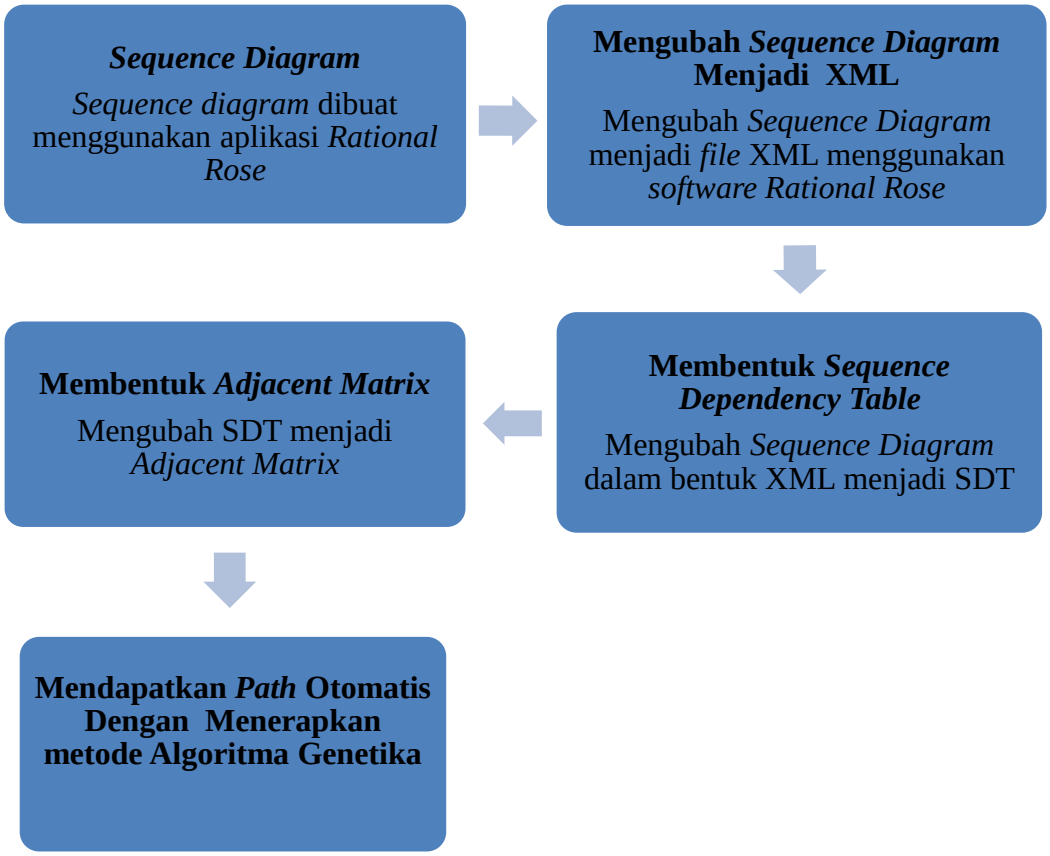
a. Ruang masalah sangat besar, kompleks, dan sulit dipahami,
b. Kurang atau bahkan tidak ada pengetahuan yang memadai untuk merepresentasikan masalah ke dalam ruang pencarian yang lebih sempit,
c. Tidak tersedianya analisis matematika yang memadai,
d. Ketika metode-metode konvensional sudah tidak mampu menyelesaikan masalah yang dihadapi,
e. Solusi yang diharapkan tidak harus paling optimal, tetapi cukup “bagus” atau bisa diterima,
f. Terdapat batasan waktu, misalnya dalam *real time system* atau sistem waktu nyata.

BAB III

DESAIN DAN PERANCANGAN SISTEM

3.1 Desain Sistem

Bab ini menjelaskan desain sistem yang akan dibangun. Sebelum membangkitkan sistem, tentunya ada tahapan yang harus dikerjakan terlebih dahulu, yaitu tahap desain dan perancangan sistem. Pada tahap ini akan digambarkan proses pembangkitan sistem secara general. Berikut ini adalah gambar desain sistem beserta penjelasannya.



Gambar 3. 1 Desain Sistem

Tahapan awal dalam pembangkitan *test case* ini adalah membuat *Sequence Diagram* pada *software Rational Rose* yang merupakan produk IBM lalu

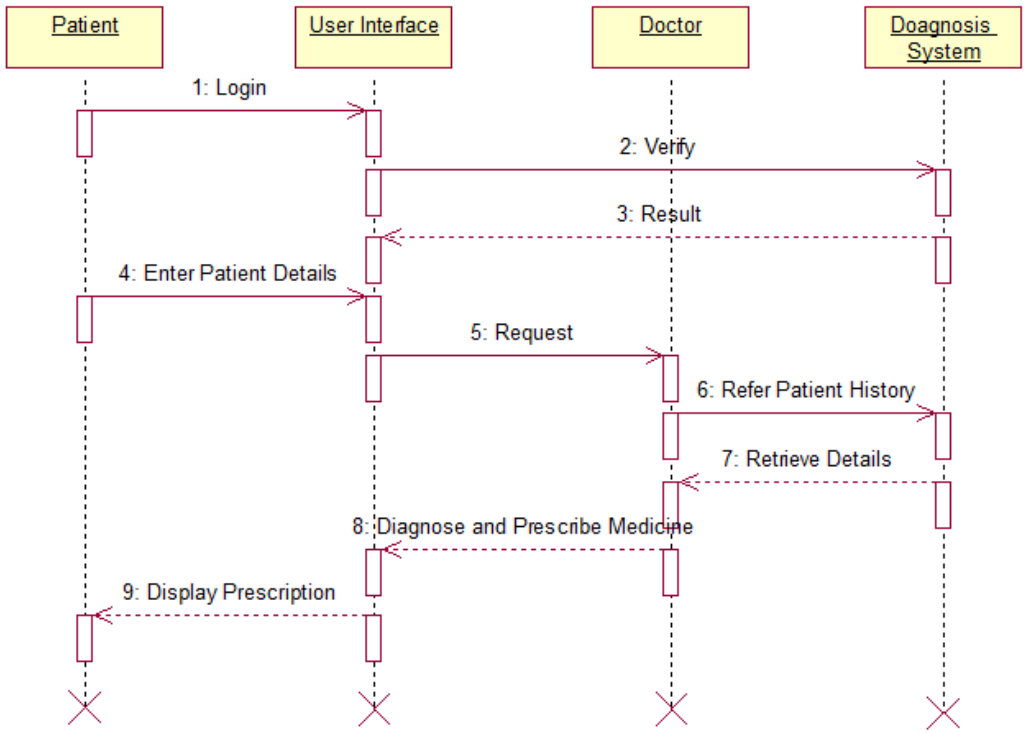
disimpan dengan ekstensi .xml. Kemudian dari *file* XML tersebut dicari informasi yang diperlukan untuk menghasilkan *Sequence Dependency Table*. Dari SDT yang dihasilkan, selanjutnya membentuk *Adjacent Matrix* menggunakan data dari SDT tersebut. Hasilnya adalah *path* yang dibangkitkan otomatis dengan menerapkan metode algoritma genetika.

3.2 Perancangan Sistem

3.2.1 Pembuatan UML *Sequence Diagram*

Pembuatan *Sequence Diagram* pada sistem ini dirancang menggunakan *software Rasional Rose* dan disimpan dalam ekstensi .xml. Dalam sebuah penelitian (Priya & Malarchervi, 2013) membuat sebuah *sequence diagram* dengan studi kasusnya yaitu sistem konsultasi medis. *Medical Consultation System* ini bisa disebut sebagai suatu layanan saran terkomputerisasi yang mampu menjadi sistem informasi rujukan. Layanan ini memberikan layanan agar para pasien bisa mendapatkan perawatan medis sesuai dengan yang dibutuhkan tanpa harus bertemu secara langsung dengan dokter. Sistem ini dapat menghubungkan antara pasien dan dokter agar bisa berkomunikasi secara daring. Sistem ini juga memudahkan untuk mengelola konsultasi dan permintaan bantuan harian melalui komputer. Pasien bisa mengakses layanan dari komputer manapun tanpa harus bertemu secara langsung. Untuk menggunakan sistem ini, pasien harus memiliki akun yang mana harus mencantumkan nama pengguna dan kata sandi. Jika pasien login dengan benar, maka pasien bisa melihat informasi rinci seperti nama, jenis kelamin, usia, hari dan tanggal kunjungan terakhir, nama dokter yang ingin ditemui dan gejala yang yang sesuai pasien rasakan. Jika dokter yang inin ditemui tersedia, maka dokter dapat merujuk pada riwayat medis pasien dari sistem

diagnostik dan gejala saat ini yang diberikan oleh pasien. Jika pasien menjelaskan gejala yang dirasakan dengan rinci, maka dokter merekomendasikan agar melakukan perawatan untuk membantu pasien pulih dari penyakit. Namun, apabila tidak, dokter dapat menyarankan pasien untuk menjalani pemeriksaan tambahan yang ditentukan dan melapor langsung kepadanya untuk saran tambahan.



Gambar 3. 2 UML *Sequence Diagram Sistem Konsultasi Kesehatan* (Priya, 2013)

a. *Login*: Pasien melakukan *login* menuju sistem dengan memasukkan nama pengguna dan kata sandi yang valid.

b. *Verify*: Jelajahi *database* untuk melihat apakah nama pengguna dan kata sandi yang dimasukkan valid.

c. *Result*: Mengembalikan hasil di antarmuka pengguna. Jika nama pengguna dan kata sandi yang dihasilkan cocok dengan sistem, pengguna

diautentikasi dan memberikan akses lebih lanjut ke sistem. Bila tidak, maka akses akan ditolak yang mana berarti pengguna yang tidak cocok.

d. *Enter Patient Details*: Setelah berhasil *login*, pasien akan diminta memasukkan informasi pribadi seperti nama, jenis kelamin, usia, hari atau tanggal kunjungan terakhir, dokter untuk berkonsultasi, gejala penyakit, dll.
e. *Request*: sistem untuk terhubung dengan dokter. Jika dokter bisa menanganinya, dia bisa menangani pasiennya. Jika tidak, pasien akan diberitahu tentang ketidakhadiran dokter.
f. *Refer Patient History*: Apabila pasien sudah mendapatkan dokter, maka dokter dapat mengirim permintaan ke sistem diagnostik untuk membuat diagnosis berdasarkan riwayat gejala pasien.
g. *Retrieve Details:* Memuat info-info mengenai pasien yang telah pulih.
h. *Diagnose and prescribe Medicine:* Setelah meninjau riwayat medis pasien dan gejala saat ini, dokter dapat mendiagnosis tingkat keparahan penyakit dan meresepkan obat, jika cukup. Jika gejalanya tidak cukup untuk membuat diagnosis, dokter dapat memesan tes lebih lanjut.
i. *Display Prescription:* Anda dapat mengirim resep dokter ke antarmuka pengguna untuk menunjukkannya kepada pasien Anda.

3.2.2 Mengeksport *Sequence Diagram* menjadi XML

Sequence diagram yang telah dibuat dengan menggunakan *Rational Rose* disimpan dalam *file* XML agar memudahkan dalam proses pengujian. Berikut ini adalah *file* XML hasil *export* dari *sequence diagram*:

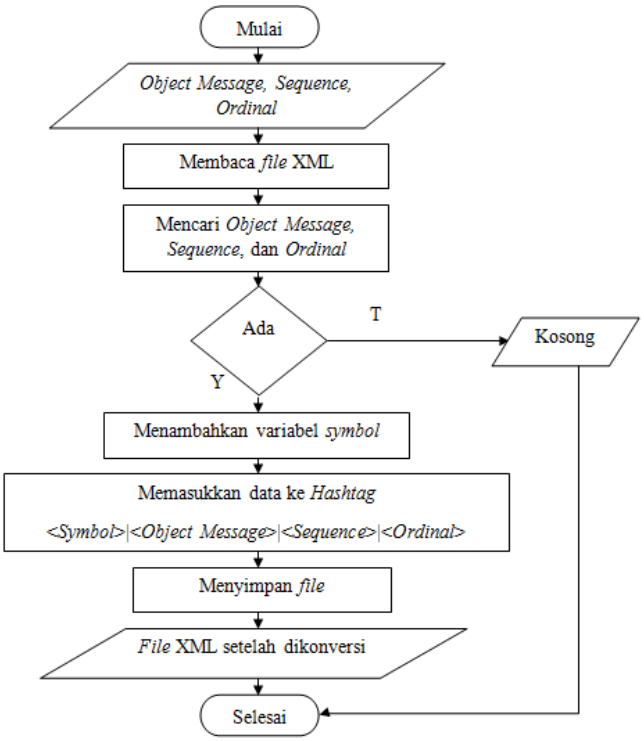
MedicalSystem.xml

```
(object Petal
    version     50
    _written    "Rose 8.2.0310.2800"
    charSet     0)

(object Design "Logical View"
    attributes  (list Attribute_Set
    (object Attribute
        tool        "Java"
        name        "IDE"
        value       "Internal Editor"))
    quid        "5852B5AB0152"
    enforceClosureAutoLoad  FALSE
    defaults    (object defaults
    rightMargin     0.250000
    leftMargin  0.250000
    topMargin   0.250000
    bottomMargin    0.500000
    pageOverlap     0.250000
    clipIconLabels  TRUE
    autoResize  TRUE
    snapToGrid  TRUE
    gridX       0
    gridY       0
    defaultFont     (object Font
        size        12
        face        "Arial"
        bold        FALSE
        italics     FALSE
        underline   FALSE
        strike      FALSE
        color       0
        default_color   TRUE)
    showMessageNum  3
```

Gambar 3. 3 *File* XML hasil *export* dari *Sequence Diagram*

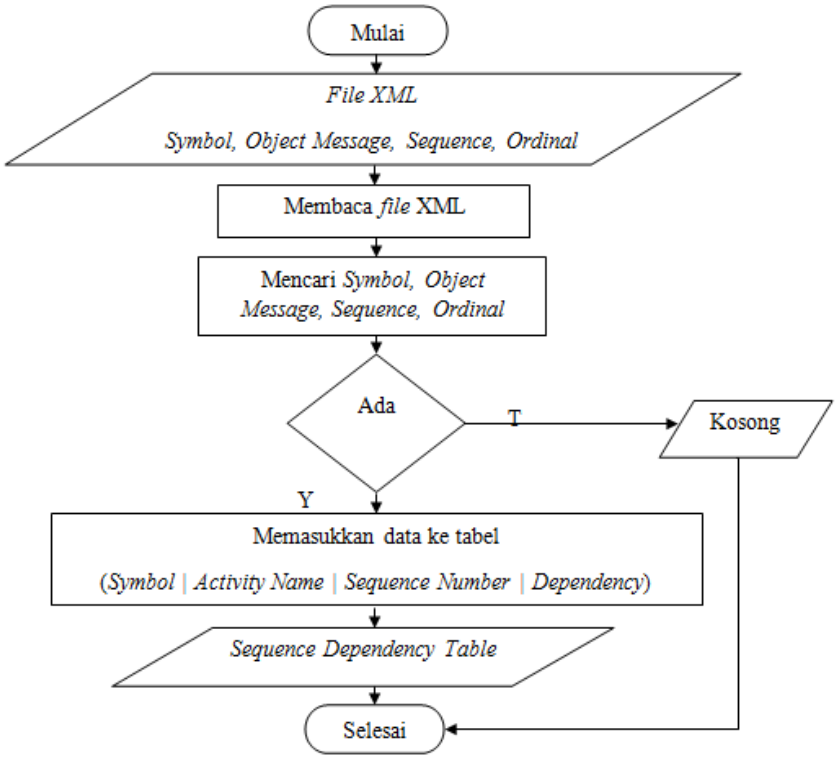
3.2.3 Pembuatan *Sequence Dependency Table* (SDT)

Dari *sequence diagram* yang telah dibuat seperti pada gambar 3.2, kemudian akan dibangun *Sequence Dependency Table* (SDT). Namun sebelumnya, *Sequence Diagram* yang sudah dibuat dikonversikan terlebih dahulu ke dalam *file* XML karena ada beberapa informasi penting yang akan diambil dari *file* XML dari *Sequence Diagram* tersebut. Informasi tersebut adalah simbol atau *symbol*, nama aktifitas atau *activity name*, nomor sequence atau *sequence number* dan *dependency*. Berikut ini adalah *flowchart* untuk mengambil informasi-informasi penting dari *file* XML.



Gambar 3. 4 *Flowchart* mengambil informasi penting dari *file* XML (Ariyani, 2017)

Adanya pengambilan informasi penting ini karena dalam *file* XML memuat banyak informasi yang tidak dibutuhkan dalam pembuatan *Sequence Dependency Table* (SDT). *Flowchart* pada gambar 3.3 adalah proses bagaimana informasi-informasi penting diambil dari *file* XML untuk dibangun *file* XML baru yang hanya memuat informasi yang dibutuhkan untuk membangun *Sequence Dependency Table* (SDT). Tahapan awalnya yaitu mengambil *file* XML *Rational Rose* yang diekspor, kemudian membaca variabel *object message*, *object sequence* dan object ordinal. Apabila variabel yang dicari itu ada atau ditemukan, sistem akan menambahkan variabel simbol, sebaliknya output akan kosong jika nilai tidak ada atau tidak ditemukan. Proses selanjutnya adalah menyimpan variabel-variabel ini dalam tagar, dengan ekstensi XML baru. Prosedur untuk membuat *Sequence Dependency Table* dijelaskan dalam diagram berikut.



Gambar 3. 5 *Flowchart* pembuatan *Sequence Dependency Table* (SDT) (Ariyani, 2017)

Setelah pengambilan informasi penting untuk pembuatan *Sequence Dependency Table* (SDT) seperti yang sudah dijelaskan sebelumnya, kemudian *file* XML yang baru akan dibaca oleh sistem untuk dicari variabel *Symbol, Object Message, Sequence* dan Ordinal. Variabel-variabel tersebut yang dibutuhkan dalam pembuatan *Sequence Dependency Table* (SDT). Setelah variabel tersebut ditemukan, maka dibentuk tabel dengan empat kolom yang berisi *Symbol, Activity Name, Sequence Number* dan *Dependency*. Namun sebaliknya, jika variabel tidak ditemukan maka *output* akan kosong. *Symbol* mewakili variabel *Symbol*, *Activity Name* yang berarti variabel *Object Message*, *Sequence Number* yang berarti variabel *Sequence* dan *Dependency* yang berarti variabel *Dependency*.

Berdasarkan *Sequence Diagram* pada penelitian (Priya & Malarchervi, 2013) gambar 3.2 yaitu *Medical Consultation System*, berikut ini adalah hasil *Sequence Dependency Tabel* yang sudah dibangun yang memuat enam kolom.

Tabel 3. 1 *Sequence Dependency Table* (SDT) *Medical Consultation System* (Priya & Malarchervi, 2013)

Symbol	***Activity Name***	***Sequence Number***	***Dependency***	***Input***	***Expected Output***
1	*Login*	1	-	Nama pengguna dan kata sandi	Memvalidasi nama pengguna dan kata sandi
2	*Verify*	2	1	-	Validasi ID pasien dan kata sandi / kata sandi atau ID pasien tidak valid
3	*Result*	3	2	-	Masukkan ID pasien dan kata sandi yang valid untuk pindah ke layar berikutnya / Masukkan ID pasien dan kata sandi yang tidak valid untuk keluar
4	*Patient Details*	4	3	Pasien memasukkan nama, usia, jenis kelamin, tanggal konsultasi terakhir, gejala, nama dokter	Konfirmasi detail (aktif) / detail nonaktif (selesai)
5	*Request*	5	4	-	Lanjutkan jika ada dokter / Akhiri jika tidak ada dokter
6	*Refer Patient History*	6	5	-	Cek detail

7	*Retrieve Details*	7	6	-	Mendapatkan dan melihat riwayat pasien
8	*Diagnose and Prescribe Medicine*	8	7	-	Dokter mendiagnosis dan meresepkan obat /Merekomendasikan pengujian lebih lanjut untuk diagnosis lebih lanjut
9	*Display Prescription/Result*	9	8	-	Pasien Menerima resep / tes yang direkomendasikan dokter
10	*End*	-	3,4,5, 9	-	-

Menurut penelitian Priya et al (2013), *Sequence Dependency Table* pada sistem konsultasi medis memuat dari enam informasi yaitu:

a. Simbol atau *symbol*: angka atau huruf yang mewakili setiap aktifitas
b. Nama aktifitas atau *activity name:* nama aktifitas yang dilakukan
c. Nomor *sequence* atau *sequence number*: Nomor urut dari setiap aktifitas
d. *Dependency*: Nomor dari aktifitas yang memiliki keterkaitan dengan aktifitas yang lain
e. *Input*: Ini adalah prasyarat yang diperlukan dalam melakukan aktifitas
f. Output yang diharapkan atau *expected output*: Hasil yang diharapkan dari aktivitas yang terjadi.

Pada penelitian yang dikerjakan penulis, tabel hanya akan berisi empat infomasi yaitu simbol, nama aktifitas, nomor *sequence* dan *Dependency*.

3.2.4 Membentuk *Adjacent Matrix* Berdasarkan SDT (*Sequence Dependency Table*)

Proses pembentukan *adjacent matrix* ini mengambil data dari *dependency sequence number*, yang mana merupakan hasil dari *Sequence Dependency Table*. Baris *dependency* sebagai *node* 1 (*node* awal) dan kolom *sequence number* sebagai *node* 2 (*node* akhir). Dalam *adjacent matrix* ini, *node* yang terhubung dengan *node* lainnya bernilai 1 dan *node* yang tidak terhubung bernilai 0. Berikut ini *adjacent matrix Medical Consultation System* berdasarkan *Sequence Dependency Tabel* (SDT) pada tabel 3.1.

Tabel 3. 2 *Adjacent Matrix Medical Consultation System*

Node		**SEQUENCE**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
D	**1**	0	1	0	0	0	0	0	0	0	0
E	**2**	0	0	1	0	0	0	0	0	0	0
P	**3**	0	0	0	1	0	0	0	0	0	1
E	**4**	0	0	0	0	1	0	0	0	0	1
N	**5**	0	0	0	0	0	1	0	0	0	1
D	**6**	0	0	0	0	0	0	1	0	0	0
E	**7**	0	0	0	0	0	0	0	1	0	0
N	**8**	0	0	0	0	0	0	0	0	1	0
C	**9**	0	0	0	0	0	0	0	0	0	1
Y	**10**	0	0	0	0	0	0	0	0	0	0

Source code yang digunakan untuk membentuk *adjacent matrix* adalah sebagai berikut:

```
            List<Relation> Relations2 = Relations;

            System.out.println(Relations.get(0));
            System.out.println(Relations.get(0).getSequendeNumber());
            System.out.println(Relations.get(0).getDependency());

//              memisahkan string berdasarkan tab
             String[] arrayTeksSplitseq = Relations.get(0).getSequendeNumber().split("\t");
             String[] arrayTeksSplitseq2 = arrayTeksSplitseq[1].split("\"");
             String[] arrayTeksSplitdep = Relations.get(0).getDependency().split("\t");
             System.out.println(arrayTeksSplitseq[1]);
             System.out.println(arrayTeksSplitdep[1]);

             //variabel buat sequence dan dependency
            int cari = Integer.parseInt(arrayTeksSplitseq2[1]);
            int seq = Integer.parseInt(arrayTeksSplitdep[1]);
            System.out.println(cari);
            System.out.println(seq);

            int[][] adjmat = new int [Relations.size()][Relations.size()];
```

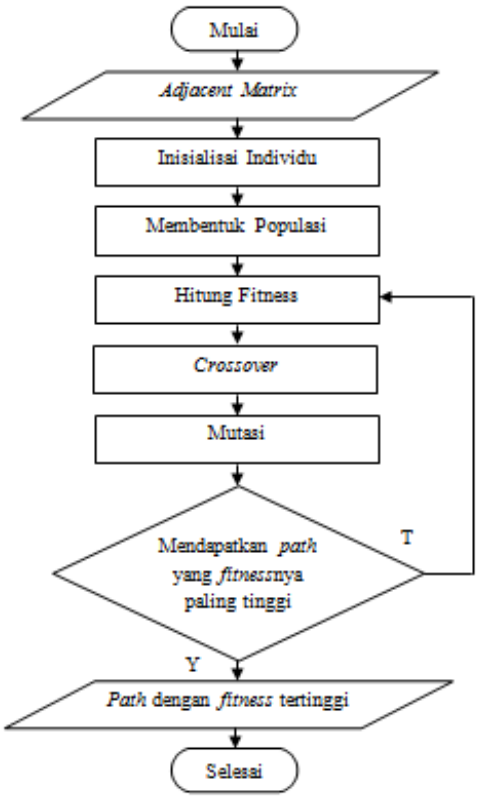
```
// pengisian untuk adjacent matrix
for (int i = 0; i < Relations.size(); i++) {
    String[] arrayTeksSplitseqloop = Relations.get(i).getSequendeNumber().split("\t");
    String[] arrayTeksSplitseq2loop = arrayTeksSplitseqloop[1].split("\"");
    String[] arrayTeksSplitdeploop = Relations.get(i).getDependency().split("\t");
    System.out.println(arrayTeksSplitseqloop[1]);
    System.out.println(arrayTeksSplitdeploop[1]);

    int kolomseq = Integer.parseInt(arrayTeksSplitseq2loop[1]);
    if (arrayTeksSplitdeploop[1].length() == 1) {
        int barisdep = Integer.parseInt(arrayTeksSplitdeploop[1]);
        if (kolomseq != adjmat[1].length) {
            adjmat[barisdep][kolomseq] = 1;
        }

    } else {
        String[] splitdep = arrayTeksSplitdeploop[1].split(",");
        for (int j = 0; j < splitdep.length; j++) {
            int barisdeploop = Integer.parseInt(splitdep[j]);
                adjmat[barisdeploop - 1][kolomseq - 1] =1;
        }
```

```
            }
        }

        for (int i = 0; i < adjmat.length; i++) {
            for (int j = 0; j < adjmat[i].length; j++) {
                System.out.print(adjmat[i][j] + " ");
            }
            System.out.println();
        }

        for (Relation empl : Relations) {
            TextAreaGetFile1.append(empl.toString() + "\n");
            TextAreaGetFile1.setEditable(false);
            System.out.println(empl.toString());

        }

        for (int i = 0; i < adjmat.length; i++) {
            for (int j = 0; j < adjmat[i].length; j++) {
                  jTextArea1.append( String.valueOf(adjmat[i][j])+" ");
            }
            jTextArea1.append( "\n");
        }
```

Gambar 3. 6 *Source Code* untuk membentuk *adjacent matrix*

3.2.5 Penerapan Algoritma Genetika

Pada penelitian ini, Algoritma Genetika diterapkan untuk mencari *test path* berdasarkan *Adjacent Matrix* yang telah terbentuk. Berikut *flowchart* penerapan metode Algoritma Genetika.



Gambar 3. 7 *Flowchart* penerapan algoritma genetika

Algoritma Genetika memiliki beberapa proses didalamnya untuk diterapkan dalam *test case* ini. Proses awalnya yaitu inisialisasi individu awal secara random kemudian membentuk populasi. Populasi disini adalah kumpulan dari beberapa individu yang telah dibangkitkan sebelumnya. Proses selanjutnya yaitu menghitung nilai *fitness*. Nilai *fitness* digunakan untuk menghitung tingkat keberhasilan nilai individu terhadap solusi dari sistem ini dengan mengambil nilai yang paling tinggi. Kemudian memilih individu yang dipertahankan untuk individu selanjutnya melalui *crossover*, kemudian melakukan mutasi untuk menghasilkan individu baru. Berikut ini adalah tahapan-tahapan perhitungannya.

a. Membentuk Individu

Individu dibentuk dari *node*. Dalam 1 individu minimal ada 3 *node* dan maksimalnya adalah sesuai dengan banyaknya *node*. *Node* pertama adalah *node start* dan *node* akhir adalah *node end* dimana *node start* dan *node end* sudah ditentukan sebelumnya. Individu ini dibentuk secara acak. Misal akan dibangkitkan individu yang terdiri dari 5 *node*. Berikut adalah individu yang terdiri dari 5 *node*:

Tabel 3. 3 Pembentukan individu dengan 5 *node*

1	2	4	8	10

b. Membangkitkan Populasi

Populasi adalah kumpulan dari beberapa individu yang sudah dibentuk secara acak. Berikut ini adalah populasi dengan individu yang terdiri dari 5 *node*:

Tabel 3. 4 pembangkitan populasi

1	3	4	5	10
1	2	3	4	10
1	7	4	3	10
1	8	9	4	10

c. Menghitung Nilai *Fitness*

Menghitung nilai *fitness* dari setiap individu dengan berdasarkan *Adjacent Matrix* dimana bila antara *node* satu dan *node* lain terhubung semua maka nilainya sama dengan 1 dan apabila satu dan keseluruhan *node* tidak terhubung maka nilainya sama dengan 0.

Tabel 3. 5 Menghitung nilai *fitness* masing-masing individu

	Node 1	**Node 2**	**Node 3**	**Node 4**	**Node 5**	**Nilai Fitness**
Individu 1	1	3	4	5	10	0
Individu 2	1	2	3	4	10	1
Individu 3	1	7	4	3	10	0
Individu 4	1	8	9	4	10	0

Mengacu berdasarkan *Adjacent Matrix* seperti pada tabel diatas, pada individu 1 antara *node* 1 dan *node* 3 tidak memiliki hubungan, antara *node* 3 dan 4 memiliki hubungan, antara *node* 4 dan *node* 5 memiliki hubungan, antara *node* 5 dan *node* 10 memiliki hubungan, sehingga didapatkan nilai *fitness* nya 0 karena tidak semua *node* memiliki hubungan. Pada individu 2 antara *node* 1 dan *node* 2 memiliki hubungan, antara *node* 2 dan *node* 3 memiliki hubungan, antara *node* 3 dan *node* 4 memiliki hubungan, antara *node* 4 dan *node* 10 memiliki hubungan, sehingga didapatkan nilai *fitness* nya 1 karena semua *node* memiliki hubungan. maka nilainya 0. Begitu seterusnya dengan individu 3 dan individu 4.

d. Crossover

Proses selanjutnya adalah proses *crossover* yaitu melakukan penukaran gen-gen dari satu kromosom dengan kromosom lain untuk menghasilkan kromosom baru melalui titik potong. Kromosom induk akan dipasangkan sebanyak setengah populasi dan menentukan nilai probabilitas *crossover* (Pc). Nilai Pc yang digunakan adalah 0.5. Misal susunan pasangan induk adalah sebagai berikut:

Tabel 3. 6 Susunan pasangan induk

Pasangan 1	Induk 1
	Induk 2
Pasangan 2	Induk 4
	Induk 3

Kemudian membangkitkan bilangan acak. Bilangan acak ini disimbolkan dengan Ri. Lalu membangkitkan Ri antara 0 sampai 1 sebanyak jumlah pasangan induk. Jika bilangan acak Ri kurang dari atau sama dengan Pc, *crossover* akan terjadi pada pasangan pertama. Jika tidak, pasangan akan melanjutkan ke proses selanjutnya. Misalnya, bilangan acak yang dihasilkan adalah:

R1 = 0.743

R2 = 0.345

Dari bilangan acak yang telah dibangkitkan, terdapat satu nilai yang kurang dari nilai Pc yaitu pada R2. Ini berarti pasangan yang ke-2 akan mengalami *crossover* yaitu individu 3 dan individu 4. Metode yang digunakan adalah *single point crossover* untuk menukar informasi pada posisi random dua individu yang terpilih untuk menghasilkan individu baru. Proses dari *crossover* sebagai berikut:

Individu 3 yaitu 1-7-4-3-10 dengan individu 4 yaitu 1-8-9-4-10 dilakukan *crossover* dengan titik potong 2:

Individu 3 = 1-7|-4-3-10

Individu 4 = 1-8|-9-4-10

Sehingga akan dihasilkan *offspring* hasil *crossover* seperti berikut:

Offspring 3 = 1-7-9-4-10

Offspring 4 = 1-8-4-3-10

e. Mutasi

Hal pertama yang dilakukan dalam mutasi yaitu menentukan berapa probabilitas mutasi (pm). Misalkan pada penelitian ini probabilitas mutasinya adalah 0.1. Lalu membangkitkan bilangan acak antara 0 sampai 1 sebanyak jumlah populasi. Misalkan bilangan acak yang telah dibangkitkan adalah sebagai berikut:

R1 = 0.964

R2 = 0.835

R3 = 0.056

R4 = 0.235

Berdasarkan bilangan acak yang telah dibangkitkan, R3 memiliki nilai lebih kecil dari Pm, sehingga hanya individu ke-3 yang terpilih untuk dimutasi. Proses mutasi dilakukan dengan menggunakan metode *inversion mutation*, langkah-langkahnya adalah sebagai berikut:

a. Membangkitkan bilangan bulat acak antara 2 sampai n-1 sebagai titik potong T1 dan T2, yang mana n merupakan jumlah *node* yaitu 5. Misalkan T1 = 2 dan T2 = 4, sehingga:

   Individu 3 = [1-|7-9-4|-10]

b. Selanjutnya balikkan posisi *node* yang telah ditentukan dengan titik potong, sehingga:

   Individu 3 = [1-4-9-7-10]

Setelah proses mutasi selesai maka akan didapatkan populasi baru. Selanjutnya yaitu menghitung nilai *fitness* dari populasi baru tersebut. Berikut ini hasil dari perhitungan nilai *fitness* populasi baru:

Tabel 3. 7 Menghitung nilai *fitness* populasi baru

	Node 1	**Node 2**	**Node 3**	**Node 4**	**Node 5**	**Nilai Fitness**
Individu 1	1	3	4	5	10	0
Individu 2	1	2	3	4	10	1
Individu 3	1	4	9	7	10	0
Individu 4	1	8	4	3	10	0

Setelah berakhir proses mutasi didapatkan 1 individu yang memiliki nilai *fitness* yang paling tinggi yaitu pada individu 2. Individu 2 tersebut adalah *path* yang dihasilkan setelah satu generasi dengan membangkitkan individu 5 *node*. Untuk membangkitkan *path* lain, maka perlu dibangkitkan individu dengan minimal 3 *node* dan maksimalnya sesuai dengan banyaknya *node*. Dari hasil penerapan algoritma genetika pada *Medical Consultation System* dengan individu 5 *node* maka didapatkan *output* sebagai berikut:

Tabel 3. 8 *Test Case* dari *Medical Consultation System*

Path	***Test Case***
1-2-3-4-10	*Login - Verify - Result - Enter Patient Details - End*

Source code yang digunakan untuk menerapkan metode algoritma genetika yang menghasilkan *test case* secara otomatis adalah sebagai berikut:

```
Genetika gen = new Genetika();

Populasi pop = new Populasi();
List<Integer> kromosomUnggul = new ArrayList<>();
pop = generateKromosom(jumlahIndividu, panjangKromosom);
kromosomUnggul = steadyState(jumlahGenerasi, pop, jumlahIndividu, panjangTournament);
double x1 = dekodeKromosom(kromosomUnggul.subList(0, kromosomUnggul.size() / 2), rMinX1, rMaxX1);
double x2 = dekodeKromosom(kromosomUnggul.subList(kromosomUnggul.size() / 2, kromosomUnggul.size()), rMinX2, rMaxX2);
String fileSDT = jTextArea1.getText();
String[] s = jTextArea1.getText().split("\n");
String st = "";
System.out.println("s" + s.length);

//matrix
for (int i = 1; i < s.length; i++) {
    System.out.println("generation :" + s[i]);
    TextAreaGeneratePath.setText("Hasil cross over : "+s[i]);

    st = s[i];
    String[] arrayTeksSplit = st.split("\t");
    System.out.println("Hasil cross over : "+s[i]);
    TextAreaGeneratePath.setText("\n"+"Hasil cross over : "+s[i]);
    for (int j = 0; j < arrayTeksSplit.length; j++) {
        String data1 = arrayTeksSplit[1];
        String data2 = arrayTeksSplit[2];
        String[] data1a = data1.split(" ");
        String[] data2b = data2.split("\"");
        for (int a = 0; a < data1a.length; a++) {

    for (int b = 0; b < data2b.length; b++) {
        String data1aa = data1a[0];
        String data2bb = data2b[1];
        System.out.println("cvcv");
        if (data1aa.length() != 1) {
            String[] data1aaa = data1a[0].split(",");
            for (int c = 0; c < data1aaa.length; c++) {
                String data1mod = data1aaa[c];
                System.out.println("kromosom terbaik="+data1mod);
                gen.addEdge(data1mod, data2bb);
                TextAreaGeneratePath.setText("Hasil cross over :"+s[i]+"\n"+"Kromosom Terbaik :"+data1aa+"\n"+"Track Node"
                + " Populasi :"+"\n");
            }
        } else {
            gen.addEdge(data1aa, data2bb);
        }
        break;
    }
}
break;
```

Gambar 3. 8 *Source code* penerapan algoritma genetika

BAB IV

HASIL DAN PEMBAHASAN

Pada bab ini akan dipaparkan hasil penelitian yang telah dilakukan dan hasil tersebut akan dibahas untuk mengetahui apakah algoritma genetika mampu menghasilkan *test case* secara otomatis dari *sequence diagram*.

4.1 Hasil

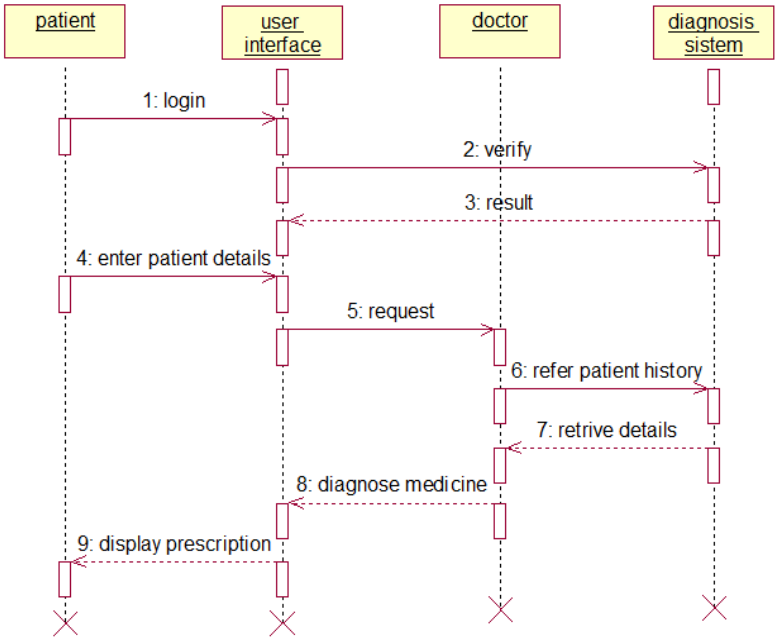
Hasil penelitian ini adalah berupa *path* yang dibangkitkan secara otomatis menggunakan algoritma genetika dari sebuah *sequence diagram*. Untuk mendapatkan *path* maka perlu dibuat aplikasi yang didalamnya terdapat algoritma genetika. Untuk menguji apakah aplikasi yang dibangun dapat bekerja dengan baik diperlukan data uji coba. Data uji dan cara kerja aplikasi akan dijelaskan dalam subbab ini.

4.1.1 Data Uji Coba

Data uji coba diperlukan sebagai bahan untuk menguji apakah aplikasi yang telah dibuat telah bekerja sesuai dengan harapan yaitu menghasilkan *path* secara otomatis. Terdapat empat data uji coba yang digunakan, yaitu Seqeunce Diagram *Medical Consultation System* (Priya et al, 2013), *Sequence Diagram* Sistem ATM atau *ATM System* pada paper (Shanti et al, 2012), *Sequence Diagram Aircraft Departure Activity (*Rhmann, 2016) dan *Sequence Diagram* Sistem Evaluasi atau Penilaian Pembelajaran sebagai studi kasusnya.

A. *Sequence Diagram Sistem Konsultasi Medis atau Medical Consultation System*

Pada jurnal Priya (2013), *Sequence diagram* Sistem Konsultasi Medis atau *Medical Consultation System* adalah bisa disebut sebagai suatu layanan saran terkomputerisasi yang mampu menjadi sistem informasi rujukan. Pada sistem ini memberikan layanan agar para pasien bisa mendapatkan perawatan medis sesuai dengan yang dibutuhkan tanpa harus bertemu secara langsung dengan dokter. Sistem ini bisa dijadikan alat untuk berkomunikasi antara pasien dan dokter secara daring. Sistem ini juga memudahkan untuk mengelola konsultasi dan permintaan bantuan harian melalui komputer. Pasien bisa mengakses layanan dari komputer manapun tanpa harus bertemu secara langsung. Untuk menggunakan sistem ini, pasien harus memiliki akun yang mana harus mencantumkan nama pengguna dan kata sandi. Jika pasien login dengan benar, maka pasien bisa melihat informasi rinci seperti nama, usia, jenis kelamin, hari dan tanggal kunjungan terakhir, nama dokter yang ingin ditemui dan gejala yang yang sesuai pasien rasakan. Jika dokter yang inin ditemui tersedia, maka dokter dapat merujuk pada riwayat medis pasien dari sistem diagnostik dan gejala saat ini yang diberikan oleh pasien. Jika pasien menjelaskan gejala yang dirasakan dengan rinci, maka dokter merekomendasikan agar melakukan perawatan untuk membantu pasien pulih dari penyakit. Apabila tidak, dokter dapat memeberi saran kepada pasien untuk menjalani pemeriksaan tambahan yang ditentukan dan melapor langsung kepadanya untuk saran lebih lanjut. Di bawah ini adalah gambar dari *sequence diagram Medical Consultation System*.

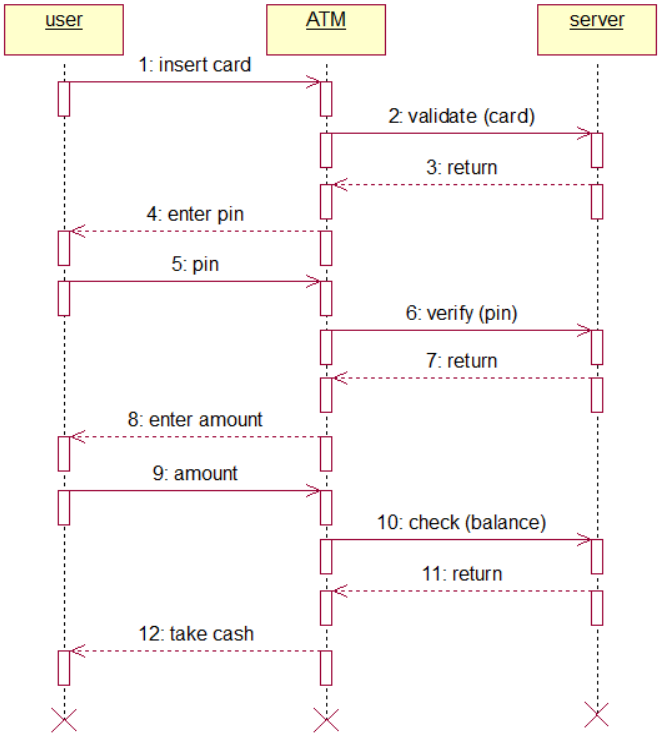


Gambar 4. 1 *Sequence Diagram Medical Consultation System* (Priya et al, 2013)

B. *Sequence Diagram ATM System*

Sequence Diagram pada *ATM System* sesuai pada jurnal (Shanti et al, 2012) adalah layanan bagi *customer bank* untuk memudahkan mereka dalam pengambilan dan transfer uang tanpa harus melalui layanan kantor bank. *Customer bank* bisa mengakses layanan tersebut hanya jika memiliki kartu ATM yang diperoleh saat pembuatan rekening di kantor bank. Saat menggunakan mesin ATM, hal pertama yang dilakukan yaitu memasukkan kartu ATM kemudian mesin meneruskannya ke server untuk divalidasi. Selanjutnya *customer* akan diminta untuk memasukkan pin sejumlah sekian digit yang sudah ditentukan *customer* saat pembuatan rekening. Jika pin yang dimasukkan benar, customer bisa mengakses layanan mesin ATM tersebut. Tahap selanjutnya *customer* diminta untuk memasukkan nominal uang yang akan diambil dengan tidak melebihi batas pengambilan uang pada mesin ATM. Jika nominalnya sudah sesuai maka mesin akan memproses permintaan *customer* dan uang tunai bisa diambil

dari mesin ATM tersebut. Berikut ini adalah gambar *Sequence Diagram* dari *ATM System*:

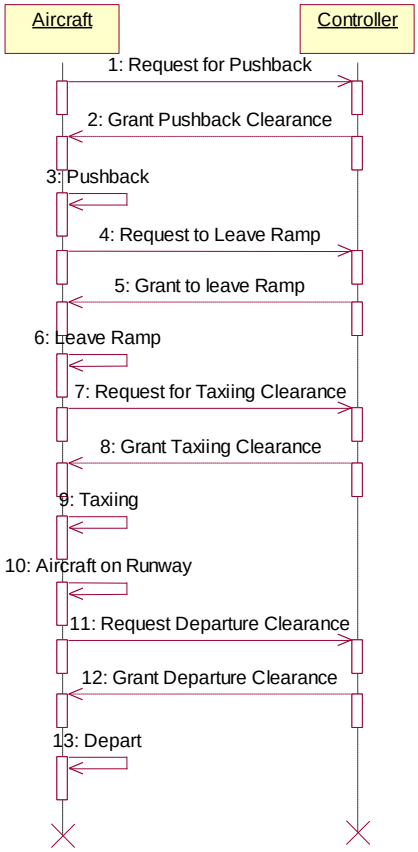


Gambar 4. 2 *Sequence Diagram ATM System* (Shanti et al, 2012)

C. *Seqeunce Diagram Aircraft Departure Activity*

Seqeunce Diagram *Aircraft Departure Activity* (Rhmann, 2016) merupakan rangkaian operasi pemberangkatan pesawat udara. Ada dua objek dalam gambar ini: pesawat (*aircraft)* dan pengontrol (*controller)*. Kedua objek ini bertukar pesan. Pertama, pesawat mengirim pesan permintaan persetujuan ke pengontrol sebagai tanggapan. Prosedur untuk mendorong pesawat kembali dari gerbang bandara disebut *pushback.* Selama prosedur *pushback*, pesawat akan mundur di bawah pengaruh kekuatan eksternal. Jika pengontrol mengkonfirmasi permintaan *pushback*, pesawat dapat melakukan *pushback*. Pesawat kemudian mengirimkan pesan yang meminta izin kepada pengontrol untuk meninggalkan *ramp*. Jika operator mengizinkan pesawat untuk meninggalkan *ramp*. *Ramp* itu

sendiri adalah tempat berlangsungnya perawatan atau perawatan pesawat. Pesawat kemudian mengirim pesan yang meminta operator untuk mengotorisasi taksi (*taxiing)*, sebuah proses di mana pesawat bergerak sendiri. Jika *controller* membersihkan *taxiway*, pesawat dapat bergerak ke landasan. Landasan pacu adalah area berbentuk persegi panjang yang digunakan untuk pendaratan (*landing*) dan lepas landas (*take-off*). Pesan berikutnya adalah bahwa pesawat meminta izin kepada operator untuk lepas landas. Jika operator menerima permintaan tersebut, pesawat dapat lepas landas. Di bawah ini adalah gambar diagram urutan operasi keberangkatan pesawat (Rhmann, 2016).

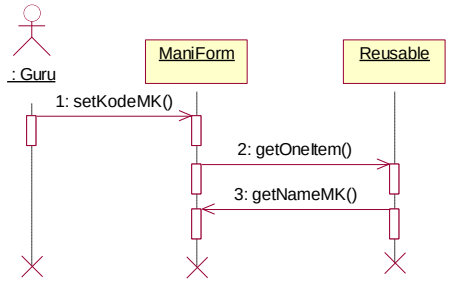


Gambar 4. 3 *Sequence Diagram Aircraft Departure Activity* (Rhmnn, 2016)

D. Sequence Diagram Sistem Evaluasi atau Penilaian pada Pembelajaran

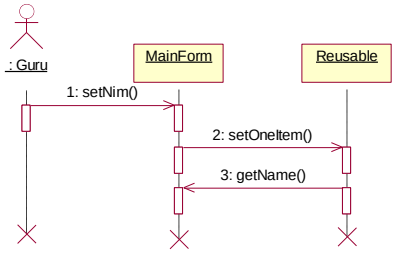
Sistem evaluasi atau penilaian pembelajaran merupakan studi kasus dari penelitian ini. Sistem evaluasi pembelajaran ini membantu guru atau guru memberikan umpan balik kepada siswa atau siswa ketika mereka mengevaluasi proses belajar mengajar. Sistem ini memungkinkan guru untuk menginputkan hasil tugas, hasil ujian tengah semester (UTS), dan nilai ujian akhir semester (UAS). Nilai-nilai tersebut diolah berdasarkan bobot masing-masing komponen evaluasi pembelajaran dan mendapatkan nilai akhir dikonversikan menjadi nilai karakter A, B, B+, C, C+, D, dan E. Ada empat *sequence diagram*. Pada sistem evaluasi pembelajaran ini yaitu diagram sequence nama mata kuliah, diagram sequence nama mahasiswa, diagram sequence penilaian atau perhitungan nilai, dan diagram sequence untuk menampilkan raport. Berikut akan dijelaskan masing-masing *sequence diagram* beserta gambarnya.

Sequence diagram yang pertama *yaitu sequence diagram* nama mata kuliah. Pada *sequence diagram* ini, *user* atau dosen selaku penggunanya memilih kode dari mata kuliah yang diampu dosen tersebut. Berikut ini adalah gambar *sequence diagram* nama mata kuliah.



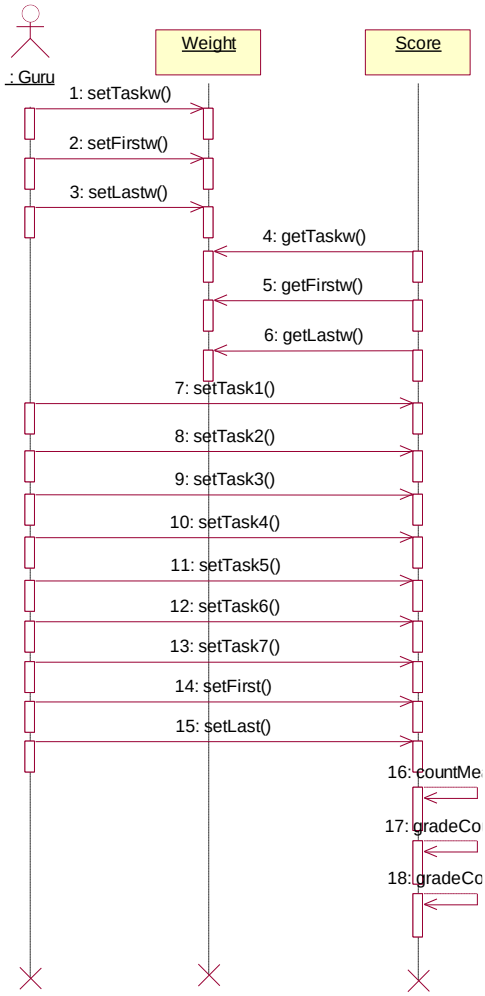
Gambar 4. 4 *Sequence Diagram* untuk nama mata kuliah

Sequence diagram yang kedua yaitu diagram sequence nama mahasiswa. Pada diagram sequence ini, dosen memilih nim dari mahasiswa yang akan diberi nilai. Berikut ini adalah *sequence diagram* nama mahasiswa.



Gambar 4. 5 *Sequence Diagram* untuk nama mahasiswa

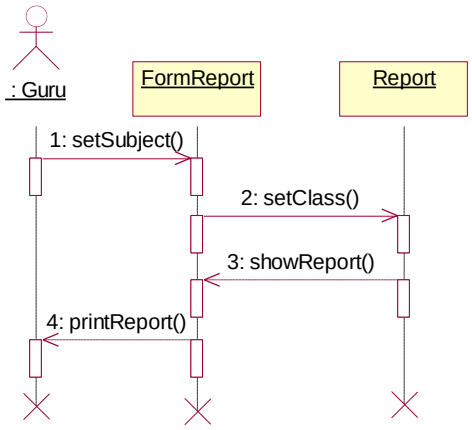
Sequence diagram yang ketiga yaitu diagram sequence perhitungan nilai. Dosen atau guru menginputkan nilai dan mengkonversinya kedalam bentuk huruf. Berikut ini adalah *sequence diagram* perhitungan nilai.



Gambar 4. 6 *Sequence Diagram* untuk perhitungan nilai

Sequence diagram yang terakhir yaitu diagram sequence menampilkan raport. Diagram sequence ini akan menampilkan laporan sistem evaluasi atau

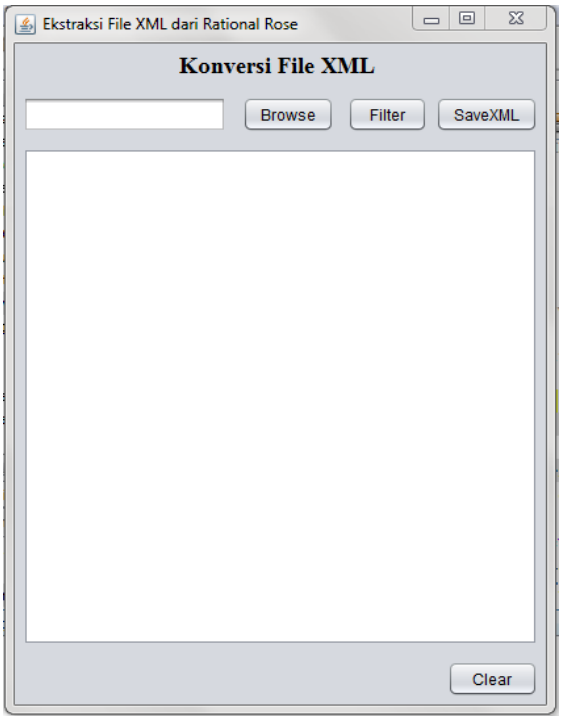
penilaian pembelajaran. Berikut ini adalah *sequence diagram* menampilakan raport.



Gambar 4. 7 *Sequence Diagram* untuk menampilkan raport

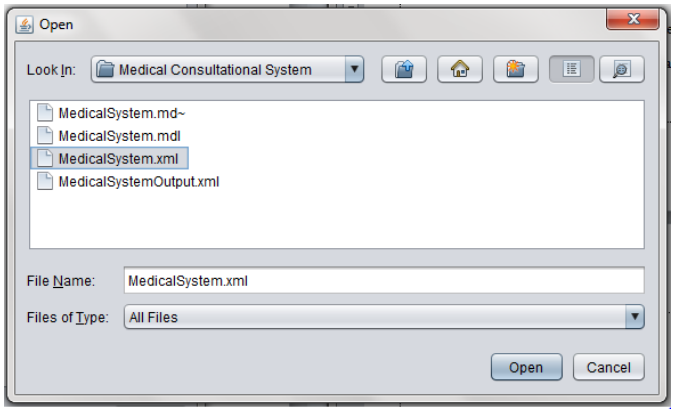
4.1.2 Hasil Uji Coba Aplikasi

Pembangkit *test case* otomatis ini memiliki dua tampilan utama yaitu tampilan untuk konversi XML dan tampilan untuk pembangkit *test case*. Tampilan konversi XML berfungsi mengambil info atau data penting yang diperlukan dari *file* XML hasil konversi *sequence diagram* dari *Rational Rose* menjadi *file* XML. *Output file* yang sudah dikonversi dari tampilan konversi XML inilah yang nantinya digunakan sebagai *input* untuk membangkitkan *test case*. Berikut ini gambar tampilan konversi XML.



Gambar 4. 8 tampilan konversi XML

Terdapat empat *button* pada tampilan konversi XML ini yaitu *button browse*, *button filter*, *button save xml*, dan *button clear*. *Text field* disamping *button browse* pada tampilan ini digunakan untuk menampung nama *file* yang sedang dicari atau ditelusuri. *Text area* yang ada pada tengah-tengah tampilan berfungsi untuk menampilkan isi *file* yang sedang dibuka. Berikut ini gambar dari masing-masing fitur yang ada pada tampilan konversi XML.

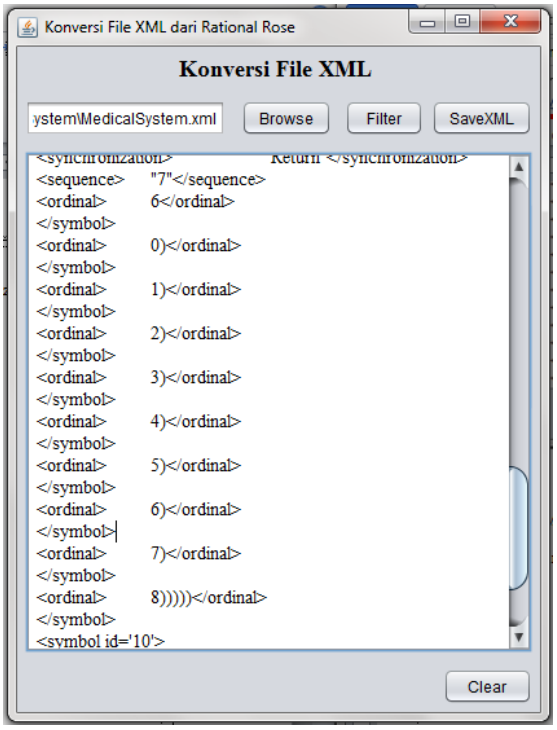


Gambar 4. 9 *Button browse* untuk *browse file* XML

Button browse memiliki fungsi untuk mengambil *file* XML data uji yang nantinya *file* XML ini akan dimunculkan pada tampilan konversi XML untuk dikonversi. Berikut ini gambar yang menunjukkan saat *file* XML muncul pada tampilan konversi XML:

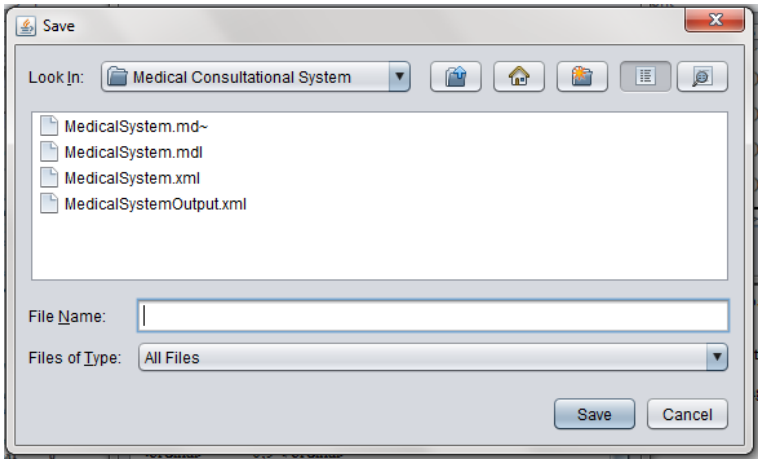
Gambar 4. 10 tampilan *file* XML

Pada *file* XML *Medical Consultation System* yang telah ditampilkan tersebut terdapat informasi atau data yang tidak diperlukan ikut terambil sehingga akan dilakukan proses filter dengan menggunakan *button filter*. *Button* ini berfungsi untuk menghapus data yang tidak diperlukan tersebut. Data yang dimaksud adalah seperti pada gambar berikut:



Gambar 4. 11 data yang akan difilter

Data tersebut adalah data yang tidak diperlukan untuk membangun *Seqeunce Dependency Table* (SDT), maka dari itu data tersebut akan dihilangkan. Setelah dilakukan proses filter, maka *file* yang sudah terkonversi tersebut akan disimpan menjadi *file* XML yang baru menggunakan *button savexml*. Berikut gambar saat *file* akan disimpan:



Gambar 4. 12 menyimpan *file* yang sudah dikonversi

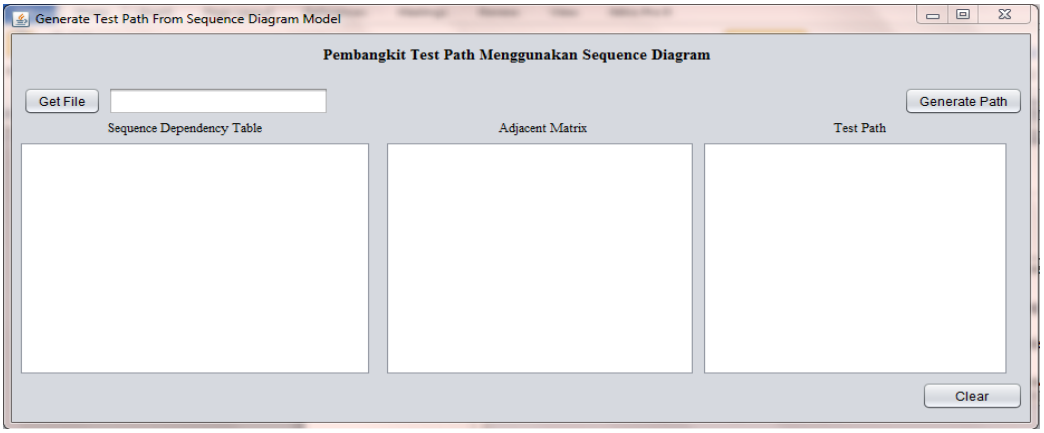
File XML yang sudah dikonversi akan disimpan menjadi *file* XML yang baru. Terdapat empat variabel penting yang diambil dari *file* XML selama proses

konversi yaitu variabel *symbol*, *object message*, *sequence* dan *ordinal*. Berikut ini gambar *file* XML dari *Medical Consultation System* yang sudah dikonversi:

```
<root>
<symbol id='1'>
<objectmessage> "Login"</objectmessage>
<synchronization>   "Simple"</synchronization>
<sequence>      "1"</sequence>
<ordinal>       0</ordinal>
</symbol>
<symbol id='2'>
<objectmessage> "Patient Details"</objectmessage>
<synchronization>   "Simple"</synchronization>
<sequence>      "4"</sequence>
<ordinal>       3</ordinal>
</symbol>
<symbol id='3'>
<objectmessage> "Display Result"</objectmessage>
<synchronization>   "Return"</synchronization>
<sequence>      "9"</sequence>
<ordinal>       8</ordinal>
</symbol>
<symbol id='4'>
<objectmessage> "Verify"</objectmessage>
<synchronization>   "Simple"</synchronization>
<sequence>      "2"</sequence>
<ordinal>       1</ordinal>
</symbol>
<symbol id='5'>
<objectmessage> "Result"</objectmessage>
<synchronization>   "Return"</synchronization>
<sequence>      "3"</sequence>
<ordinal>       2</ordinal>
</symbol>
<symbol id='6'>
<objectmessage> "Request"</objectmessage>
<synchronization>   "Simple"</synchronization>
<sequence>      "5"</sequence>
<ordinal>       4</ordinal>
```

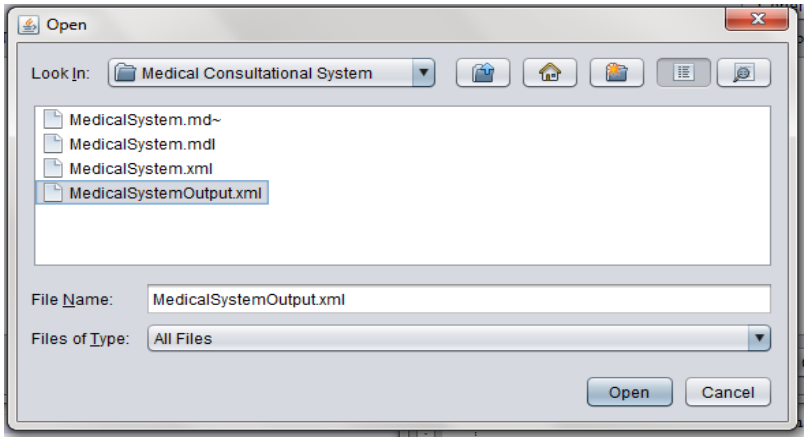
Gambar 4. 13 *file* XML yang sudah dikonversi

Hal yang sama dilakukan terhadap data uji yang lain yaitu mengkonversi *file* XML kemudian menyimpannya menjadi *file* XML yang baru. *File* XML yang sudah dikonversi ini akan menjadi *input* untuk membangkitkan *test case* otomatis pada sistem pembangkit *test case*. Berikut ini tampilan sistem pembangkit kasus uji atau *test case*:



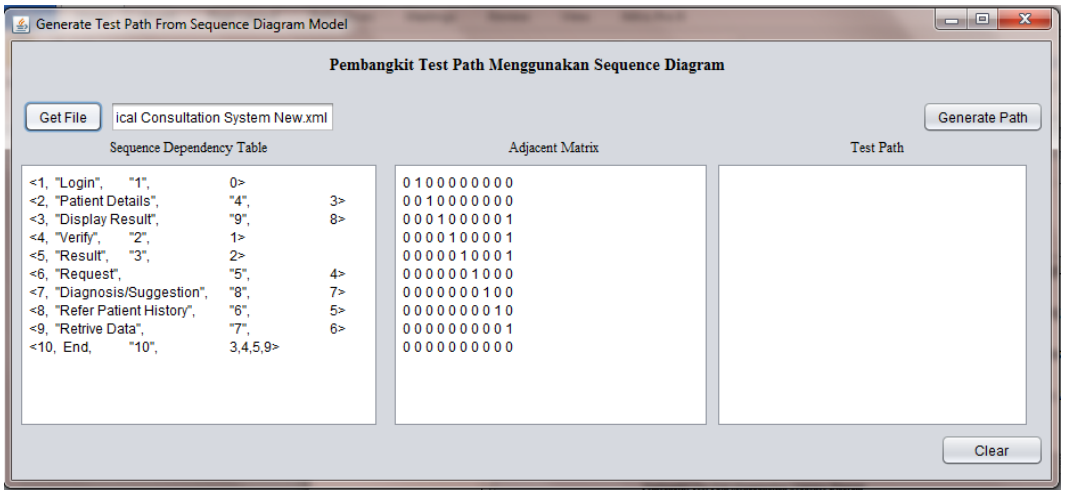
Gambar 4. 14 tampilan pembangkit kasus uji atau *test case*

Terdapat tiga *button* pada tampilan pembangkit *test case* yaitu *button get file, button generate path* dan *button clear*. *Button get file* digunakan untuk memanggil *file* XML yang sudah dikonversi. *Text field* disamping *button get file* digunakan untuk menampung nama *file* yang sedang dicari atau ditelusuri. Berikut ini gambar saat mencari XML yang sudah dikonversi:



Gambar 4. 15 *button get file* untuk mencari *XML* yang sudah dikonversi

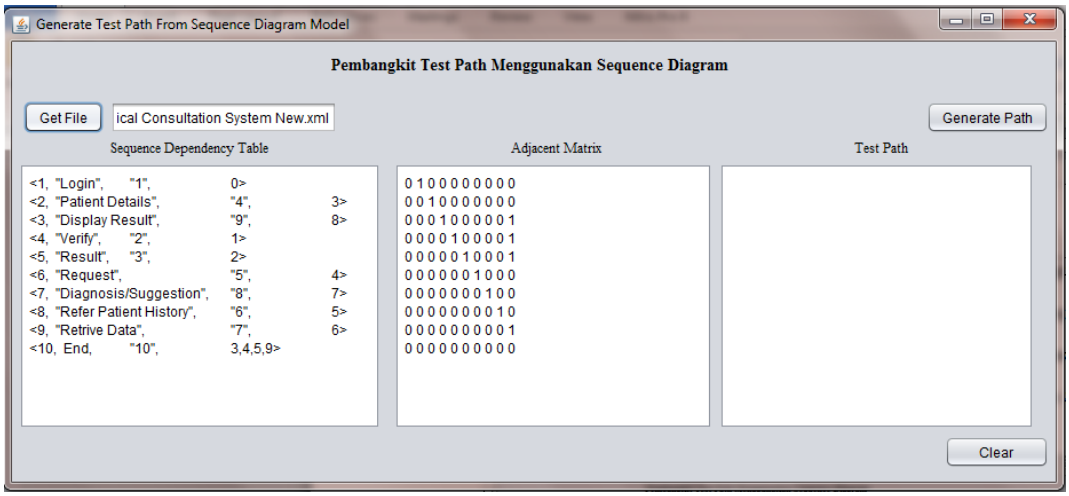
Gambar 4.15 merupakan tampilan saat pencarian *file* XML menggunakan *button get file*. *Button get file* memiliki fungsi untuk mengambil *file* XML yang sudah dikonversi kemudian *file* nya akan ditampilkan pada tampilan pembangkit *test case*. Berikut ini gambar yang menunjukkan saat *file* XML *Medical Consultation System* muncul pada tampilan pembangkit *test case*:



Gambar 4. 16 *Get file XML Medical Consultation System*

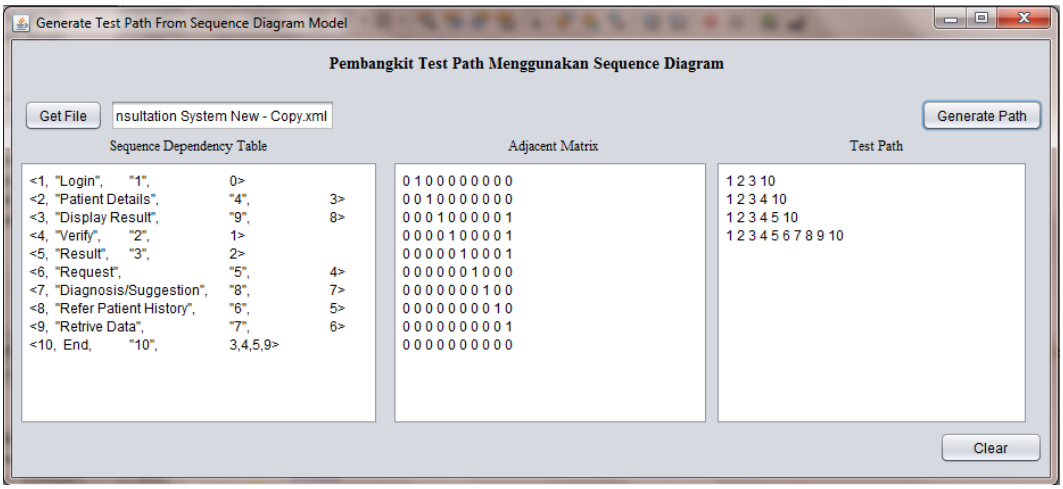
Pada kolom *Sequence Dependency Table* terdapat empat data yang ditampilkan dari kiri ke kanan yaitu *symbol*, *activity name*, *sequence* dan *dependency*. Angka "1" merupakan *symbol* yang menunjukkan setiap aktivitas yang terlibat. "*Login*" merupakan *activity name* atau nama aktivitas yang dilakukan. Angka "1" selanjutnya menunjukkan *sequence* yaitu nomor yang menyediakan urutan setiap aktifitas bertukar pesan. Angka "0" yaitu menunujukkan *dependency* yang merupakan simbol dari setiap urutan yang memiliki hubungan atau ketergantungan pada aktivitas yang lain. Begitu juga dengan *symbol* 2 dan seterusnya.

Pada kolom kedua merupakan bentuk dari *Adjacent Matrix*. Dalam *Adjacent Matrix* ini memuat *node* yang dihasilkan dari *Sequence Dependency Table* (SDT), yaitu pada kolom *sequence* dan *dependency*. Pada *Adjacent Matrix* ini, *dependency* ada di posisi baris dan *sequence* ada di posisi kolom. *Adjacent Matrix* ini hanya memuat 2 angka yaitu, 1 dan 0. 1 merupakan nilai dari *node* yang memiliki hubungan dengan *node* lain, sedangkan 0 adalah nilai dari *node* yang tidak memiliki hubungan antar *node*.



Gambar 4. 17 *Adjacent Matrix Medical Consultation System*

Proses selanjutnya yaitu membangkitkan *test case* secara otomatis dengan menggunakan tombol *generate path*. *Output* dari *generate path* ini akan ditampilkan pada *textarea Test Path* seperti pada gambar berikut:



Gambar 4. 18 *Generate Path Medical Consultation System*

Gambar 4.18 merupakan *test path* yang dihasilkan secara otomatis dengan menerapkan metode algoritma genetika sehingga didapatkan beberapa *path*. *Path* tersebut merupakan *path* dengan *node-node* yang tersambung dengan *node* lainnya sehingga nilai *fitness* yang didapatkan juga lebih tinggi. Pada path 1-2-3-10 merupakan *path* yang setiap *node* nya memiliki hubungan atau tersambung, sehingga *path* tersebut bisa dijadikan solusi yang terbaik. Begitu juga dengan *path* kedua, ketiga dan keempat. *Output Medical Consultation system* akan digambarkan dalam bentuk tabel sebagai berikut:

Tabel 4. 1 *Output Medical Consultation System*

No	***Path***	***Test Case***
1	1-2-3-10	*Login - Verify - Result - End*
2	1-2-3-4-10	*Verify - Result - Enter Patient Details - End*
3	1-2-3-4-5-10	*Login – Verify – Result - Enter Patient Details - Request - End*

No	*Path*	*Test Case*
4	1-2-3-4-5-6-7-8-9-10	*Login – Verify – Result - Enter Patient Details – Request - Refer Patient History - Retrieve Details - Diagnosis Medicine - Display Result – End*

4.2 Pembahasan

Setelah melakukan uji coba pada Sistem Konsultasi Medis atau *Medical Consultation System*, ATM *System* atau Sistem ATM, *Aircraft Departure Activity* atau Aktifitas Keberangkatan Pesawat dan Sistem Evaluasi atau Penilaian Pembelajaran, selanjutnya akan dilakukan perbandingan dengan hasil data uji pada jurnal Priya et al (2013), jurnal pada Shanti et al (2012), dan jurnal dari Rhmann (2016).

A. Data Uji *Medical Consultation System*

Sub bab ini memuat hasil data uji yang telah dilakukan pada *sequence diagram Medical Consultation System* dan membandingkan hasil uji tersebut dengan jurnal pada Priya et al (2013).

1. Hasil Uji *Medical Consultation System*

Berikut ini adalah *Adjacent Matrix Medical Consultation System* yang dihasilkan dari *sequence diagram Medical Consultation System* dan *path* yang dihasilkan dengan menerapkan algoritma genetika:

Tabel 4. 2 *Adjacent Matrix Medical Consultation System*

Node		***SEQUENCE***									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
D	**1**	0	1	0	0	0	0	0	0	0	0
E	**2**	0	0	1	0	0	0	0	0	0	0
P	**3**	0	0	0	1	0	0	0	0	0	1
E	**4**	0	0	0	0	1	0	0	0	0	1

NODENCY	*Node*	*SEQUENCE*									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	5	0	0	0	0	0	1	0	0	0	1
	6	0	0	0	0	0	0	1	0	0	0
	7	0	0	0	0	0	0	0	1	0	0
	8	0	0	0	0	0	0	0	0	1	0
	9	0	0	0	0	0	0	0	0	0	1
	10	0	0	0	0	0	0	0	0	0	0

Tabel 4. 3 *Path* yang dihasilkan dengan menerapkan algoritma genetika pada *Medical Consultation System*

No	Generasi	*Path*	Nilai *Fitness*
1	1	1-2-3-10	1
2	1	1-2-3-4-10	1
3	1	1-2-3-4-5-10	1
4	1	1-2-3-4-5-6-7-8-9-10	1

2. Perbandingan Hasil Uji *Medical Consultation System* dengan Jurnal dari Priya et al (2013)

Perbandingan hasil uji yang dilakukan pada *Medical Consultation System* dengan jurnal dari Priya et al (2013) sebagai berikut:

Tabel 4. 4 Perbandingan hasil uji pada Medical Consultation System dengan paper Priya et al (2013)

No	Hasil Uji Sistem		Hasil Uji pada Jurnal Priya	
	Path	Kasus Uji	*Path*	Kasus Uji
1	1-2-3-10	*Login - Verify - Result - End*	A-B-C-J	*Login - Verify - Result - End*
2	1-2-3-4-10	*Login - Verify - Result - Enter Patient Details - End*	A-B-C-D-J	*Login - Verify - Result - Enter Patient Details - End*
3	1-2-3-4-5-10	*Login - Verify - Result - Enter Patient Details - Request - End*	A-B-C-D-E-J	*Login - Verify - Result - Enter Patient Details - Request - End*

No	Hasil Uji Sistem		Hasil Uji pada Jurnal Priya	
	Path	**Kasus Uji**	***Path***	**Kasus Uji**
4	1-2-3-4-5-6-7-8-9-10	*Login - Verify - Result - Enter Patient Details - Request - Refer patient History - Retrieve Details - Diagnosis Medicine - Display Result - End*	A-B-C-D-E-F-G-H-I-J	*Login - Verify - Result - Enter Patient Details - Request - Refer Patient History - Retrieve Details - Diagnosis Medicine - Display Result - End*

Pada tabel 4.4 merupakan perbandingan hasil uji yang dihasilkan dari sistem dan jurnal milik Priya. Sesuai dengan data yang ada pada tabel 4.4, bahwa hasil uji dengan menggunakan aplikasi menghasilkan empat *test path* dan output dari jurnal Priya juga mendapatkan empat *path*. Keduanya memiliki kesamaan jumlah hasil, hanya saja yang membedakannya yaitu pada simbol yang digunakan. Hasil uji pada aplikasi menggunakan simbol angka untuk mendefinisikan aktifitas yang terjadi pada sistem, sedangkan hasil uji pada paper menggunakan alphabet.

B. Data Uji ATM *System*

Sub bab ini memuat hasil data uji yang telah dilakukan pada *sequence diagram ATM System* dan membandingkan hasil uji coba tersebut dengan jurnal dari Shanti et al (2012).

1. Hasil Uji ATM *System*

Berikut ini adalah *Adjacent Matrix ATM System* yang dihasilkan dari *sequence diagram* dan *path* yang dihasilkan dengan menerapkan algoritma genetika:

Tabel 4. 5 *Adjacent Matrix ATM System*

Node		***SEQUENCE***												
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
DEPENDENCY	**1**	0	1	0	0	0	0	0	0	0	0	0	0	0
	2	0	0	1	0	0	0	0	0	0	0	0	0	1
	3	0	0	0	1	0	0	0	0	0	0	0	0	1
	4	0	0	0	0	1	0	0	0	0	0	0	0	0
	5	0	0	0	0	0	1	0	0	0	0	0	0	0
	6	0	0	0	0	0	0	1	0	0	0	0	0	1
	7	0	0	0	0	0	0	0	1	0	0	0	0	1
	8	0	0	0	0	0	0	0	0	1	0	0	0	0
	9	0	0	0	0	0	0	0	0	0	1	0	0	0
	10	0	0	0	0	0	0	0	0	0	0	1	0	1
	11	0	0	0	0	0	0	0	0	0	0	0	1	1
	12	0	0	0	0	0	0	0	0	0	0	0	0	1
	13	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabel 4. 6 *Path* yang dihasilkan dengan menerapkan algoritma genetika pada ATM *System*

No	**Generasi**	***Test Path***	**Nilai *Fitness***
1	1	1-2-13	1
2	1	1-2-3-13	1
3	1	1-2-3-4-5-6-13	1
4	1	1-2-3-4-5-6-7-13	1
5	1	1-2-3-4-5-6-7-8-9-10-13	1
6	1	1-2-3-4-5-6-7-8-9-10-11-13	1
7	1	1-2-3-4-5-6-7-8-9-10-11-12-13	1

2. Perbandingan Hasil Uji Coba *ATM System* dengan Jurnal dari Shanti et al (2012)

Perbandingan hasil uji *ATM System* dengan jurnal Shantiet al (2012) sebagai berikut:

Tabel 4. 7 Perbandingan hasil uji *ATM System* dengan Jurnal dari Shanti et al (2012)

No	**Hasil Uji Sistem**		**Hasil Uji pada Jurnal Shanti**	
	Path	***Test Case***	***Path***	***Test Case***
1	1-2-13	*Insert Card - Validate (Card) - End*	A-B-M	*Insert card - Validate (Card) - End*
2	1-2-3-14	*Insert Card - Validate (Card) - Return - End*	A-B-D-E-F-M	*Insert Card - Validate (Card) - Enter Pin - Pin - Verify (Pin) - End*
3	1-2-3-4-5-6-13	*Insert Card - Validate (Card) - Return - Enter Pin - Pin - Verify (Pin) - End*	A-B-D-E-F-H-I-J-M	*Insert Card - Validate (Card) - Enter Pin - Pin - Verify (Pin) - Enter Amount - Amount - Check (Balanve) - End*
4	1-2-3-4-5-6-7-13	*Insert Card - Validate (Card) - Return - Enter Pin - Pin - Verify (Pin) - Return - End*	A-B-D-E-F-H-I-J-L-M	*Insert Card - Validate (Card) - Enter Pin - Pin - Verify (Pin) - Enter Amount - Amount - Check (Balance) - TakeCash - End*
5	1-2-3-4-5-6-7-8-9-10-13	*Insert Card - Validate (Card) - Return - Enter Pin - Pin - Verify (Pin) - Return - Enter Amount - Amount - Check (Balance) - End*		
6	1-2-3-4-5-6-7-8-9-10-11-13	*Insert Card - Validate (Card) - Return - Enter Pin - Pin - Verify (Pin) -Return - Enter Amount - Amount -Check (Balance) -Return -*		

		End		
7	1-2-3-4-5-6-7-8-9-10-11-12-13	*Insert Card - Validate(Card) - Return - enter Pin - Pin - Verify (Pin) - Return - Enter Amount - Amount - Check (Balance) - Return - Take Cash - End*		

Tabel 4.7 merupakan perbandingan hasil uji yang dihasilkan dari sistem dan papernya Shanti et al (2012). Sesuai dengan data yang ada pada tabel 4.7, bahwa hasil uji dengan menggunakan aplikasi mendapatkan tujuh *path* dan hasil uji jurnal dari Shanti et al (2012) mendapatkan empat *path*. Keduanya memiliki perbedaan dalam jumlah hasil dikarenakan hasil uji dari paper tidak mengeksekusi aktifitas sistem yang bernama *"return"*, sedangkan pada hasil uji aplikasi semua *node* pada *adjacent matrix* dilakukan eksekusi.

C. Data Uji *Aircraft Departure Activity*

Sub bab ini memuat hasil data uji yang telah dilakukan pada *sequence diagram Aircraft Departure Activity* dan membandingkan hasil uji coba tersebut dengan paper Rhmann (2016).

1. Hasil Uji Coba *Aircraft Departure Activity*

Berikut ini adalah *Adjacent Matrix* dari *Aircraft Departure Activity* yang dihasilkan dari *sequence diagram* dan *path* yang dihasilkan dengan menerapkan algoritma genetika*:*

Tabel 4. 8 *Adjacent Matrix Aircraft Departure Activity*

Node		***SEQUENCE***													
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
DEPENDENCY	**1**	0	1	0	0	0	0	0	0	0	0	0	0	0	1
	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	3	0	0	0	1	0	0	0	0	0	0	0	0	0	0
	4	0	0	0	0	1	0	0	0	0	0	0	0	0	1
	5	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	6	0	0	0	0	0	0	1	0	0	0	0	0	0	0
	7	0	0	0	0	0	0	0	1	0	0	0	0	0	1
	8	0	0	0	0	0	0	0	0	1	0	0	0	0	0
	9	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	10	0	0	0	0	0	0	0	0	0	0	1	0	0	0
	11	0	0	0	0	0	0	0	0	0	0	0	1	0	1
	12	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	13	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabel 4. 9 *Path* yang dihasilkan dengan menerapkan algoritma genetika pada *Aircraft Departure Activity*

No	**Generasi**	***Test Path***	**Nilai *Fitness***
1	1	1-14	1
2	1	1-2-3-4-14	1
3	1	1-2-3-4-5-6-7-14	1
4	1	1-2-3-4-5-6-7-8-9-10-11-14	1
5	1	1-2-3-4-5-6-7-8-9-10-11-12-13-14	1

2. Perbandingan Hasil Uji Coba *Aircraft Departure Activity* dengan Jurnal dari Rhmann (2016)

Perbandingan hasil uji *Aircraft Departure Activity* dengan paper Rhmann (2016) sebagai berikut:

Tabel 4. 10 Perbandingan Hasil Uji *Aircraft Departure Activity* pada Sistem dan Jurnal dari Rhmann (2016)

No	**Hasil Uji Sistem**		**Hasil Uji pada Jurnal Rhmann**	
	Path	***Test Case***	***Path***	***Test Case***
1	1-14	*Request for Pushback - End*	A-N	*Request for Pushback - Permission not Granted - End*
2	1-2-3-4-14	*Request for Pushback - Grant Pushback Clearance - Pushback - Request to Leave Ramp - End*	A-B-C-D-N	*Request for Pushback - Grant Pushback Clearance - Pushback - Request to Leave Ramp Permission not Grante - End*
3	1-2-3-4-5-6-7-14	*Request for Pushback - Grant Pushback Clearance - Pushback - request to Leave Ramp - Grant to Leave Ramp - Leave Ramp - Request for Taxiing Clearance - End*	A-B-C-D-E-F-G-N	*Request for Pushback - Grant Pushback Clearance - Pushback - Request to Leave Ramp -Grant to Leave Ramp - Leave Ramp - Request Taxiing Clearance - Permission not Granted - End*
4	1-2-3-4-5-6-7-8-9-10-11-14	*Request for Pushback - Grant Pushback Clearance - Pushback - Request to Leave Ramp - Grant to Leave Ramp - Leave Ramp - Request for Taxiing Clearance - Grant Taxiing Clearance - Taxiing - Aircraft on Runway - Request Departure Clearance - End*	A-B-C-D-E-F-G-H-I-J-K-N	*Request for Pushback - Grant Pushback Clearance - Pushback - Request to Leave Ramp - Grant to Leave Ramp - Leave Ramp - Request Taxiing Clearance - Grant Taxiing Clearance - Taxiing - Aircraft on Runway - Request Departure Clearance - Permission not Granted - End*

No	Hasil Uji Sistem		Hasil Uji pada Jurnal Rhmann	
	Path	***Test Case***	***Path***	**Test Case**
5	1-2-3-4-5-6-7-8-9-10-11-12-13-14	*Request for Pushback - Grant Pushback Clearance - Pushback - Request to Leave Ramp - Leave Ramp - Request for Taxiing Clearance - Grant Taxiing Clearance - Taxiing - Aircraft on Runway - Requesti Departure Clearance - Grant Departure Clearance - Depart - End*	A-B-C-D-E-F-G-H-I-J-K-L-M-N	*Request for Pushback - Grant Pushback Clearanc - Pushback - Request to Leave Ramp - Grant to Leave Ramp - Leave Ramp - Request Taxiing Clearance - Grant Taxiing Clearance - Taxiing - Aircraf on Runway - Request Departure Clearance - Grant Departure Clearance - Depart - End*

Tabel 4.10 merupakan perbandingan *output* yang dihasilkan dari aplikasi dan paper milik Rhmann (2016). Sesuai dengan data yang ada pada tabel 4.10, bahwa hasil uji dengan menggunakan sistem mendapatkan lima *path* dan hasil uji pada jurnal dari Rhmann (2016) juga menghasilkan lima *path*. Keduanya memiliki persamaan dalam jumlah hasil, hanya saja yang membedakan adalah pada hasil uji yang dihasilkan jurnal yaitu sebelum pada *node End* terdapat pesan yang berisi *"Permission not Granted"* yang memiliki maksud yaitu tidak memberikan izin terhadap aktifitas yang diminta.

D. Data Uji Sistem Penilaian Pembelajaran

Pembahasan yang selanjutnya yaitu pada data uji Sistem Evaluasi atau Penilaian Pembelajaran. Sistem ini memiliki empat bentuk *sequence diagram* dan juga memiliki empat bentuk *adjacent matrix*, yaitu sistem untuk memasukkan nama mata kuliah, sistem memasukkan nama mahasiswa, sistem perhitungan nilai dan sistem menampilkan rapor.

1. Hasil Uji Coba pada Sistem Memasukkan Nama Mata Kuliah

Bentuk *adjacent matrix* yang pertama yaitu *adjacent matrix* memaksukkan nama mata kuliah yang dihasilkan dari *sequence diagram* nama mata kuliah dan *path* yang dihasilkan dengan menerapkan algoritma genetika seperti pada tabel berikut:

Tabel 4. 11 *Adjacent Matrix* memasukkan nama mata kuliah

Node		***SEQUENCE***			
		1	**2**	**3**	**4**
DEPENDENCY	***1***	0	1	0	0
	2	0	0	1	0
	3	0	0	0	1
	4	0	0	0	0

Tabel 4. 12 *Path* yang dihasilkan dengan menerapkan algoritma genetika pada memasukkan nama mata kuliah

No	**Generasi**	***Test Path***	**Nilai *Fitness***
1	1	1-2-3-4	1

Hasil uji coba berdasarkan *adjacent matrix* memasukkan nama mata kuliah digambarkan seperti pada tabel berikut:

Tabel 4. 13 Hasil uji coba memasukkan nama mata kuliah

No	**Hasil Uji Sistem**	
	Path	***Test Case***
1.	1-2-3-4	*setKodeMK - getOneItem - getNameMK - End*

Dapat diketahui pada tabel 4.13 adalah hasil uji dari *sequence diagram* nama mata kuliah. *Generate path* yang dibangkitkan hanya menghasilkan satu *path* yaitu 1-2-3-4.

2. Hasil Uji Coba pada Sistem Memasukkan Nama Mahasiswa

Bentuk *adjacent matrix* yang kedua yaitu *adjacent matrix* dari memaksukkan nama mahasiswa yang dihasilkan dari *sequence diagram* nama mahasiswa dan *path* yang dihasilkan dengan menerapkan algoritma genetika seperti pada tabel berikut:

Tabel 4. 14 *Adjacent Matrix* memasukkan nama mahasiswa

Node		***SEQUENCE***			
		1	**2**	**3**	**4**
DEPENDENCY	**1**	0	1	0	0
	2	0	0	1	0
	3	0	0	0	1
	4	0	0	0	0

Tabel 4. 15 *Path* yang dihasilkan dengan menerapkan algoritma genetika pada memasukkan nama mahasiswa

No	**Generasi**	***Test Path***	**Nilai *Fitness***
1	1	1-2-3-4	1

Hasil uji coba berdasarkan *adjacent matrix* memasukkan nama mahasiswa digambarkan seperti pada tabel berikut:

Tabel 4. 16 Hasil uji coba memasukkan nama mahasiswa

No	**Hasil Uji Sistem**	
	Path	***Test Case***
1.	1-2-3-4	*setNim - getOneItem - getName - End*

Dapat diketahui pada tabel 4.16 adalah hasil uji pada *sequence diagram* nama mahasiswa. *Generate path* yang dibangkitkan hanya menghasilkan satu *path* yaitu 1-2-3-4.

3. Hasil Uji Coba pada Sistem Perhitungan Nilai

Bentuk *adjacent matrix* yang ketiga yaitu *adjacent matrix* dari perhitungan nilai yang didapatkan dari *sequence diagram* perhitungan nilai dan *path* yang dihasilkan dengan menerapkan algoritma genetika seperti pada tabel berikut:

Tabel 4. 17 *Adjacent Matrix* perhitungan nilai

Node		***SEQUENCE***																		
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
DEPENDENCY	**1**	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	6	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	8	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
	9	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	10	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	11	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
	12	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
	13	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabel 4. 18 *Path* yang dihasilkan dengan menerapkan algoritma genetika pada perhitungan nilai

No	**Generasi**	***Path***	**Nilai *Fitness***
1	1	1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-19	1
2	1	1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-19	1
3	1	1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19	1

Hasil uji coba berdasarkan *adjacent matrix* perhitungan nilai digambarkan seperti pada tabel berikut:

Tabel 4. 19 Hasil uji perhitungan nilai

No	**Hasil Uji Sistem**	
	Path	***Test Case***
1	1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-19	*setTaskw - setFirstw - setLastw - getTaskw - getFirstw - getLastw - setTask1 - setTask2 - setTask3 - setTask4 - setTask5 - setTask6 - setTask7 - setFirst - setLast - countMean - End*
2	1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-19	*setTaskw - setFirstw - setlastw - setTaskw - getFirstw - getLastw - setTask1 - setTask2 - setTask3 - setTask4 - setTask5 - setTask6 - setTask7 - setFirst - setLast - countMean - gradeCount - End*
3	1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19	*setTaskw - setFirstw - setLastw - setTaskw - getFirstw - getLastw - setTask1 - setTask2 - setTAsk3 - setTask4 - setTask5 - setTask6 - setTask7 - setFirst - setLast - countMean - gradeCount - gradeConv - End*

Dapat diketahui pada tabel 4.19 adalah hasil uji dari *sequence diagram* perhitungan nilai. *Generate path* yang dibangkitkan mendapatkan tiga *path*.

4. Hasil Uji pada Sistem Menampilkan Rapor

Bentuk *adjacent matrix* yang keempat yaitu *adjacent matrix* dari menampilkan raport yang dihasilkan dari *sequence diagram* menampilkan raport dan *path* yang dihasilkan dengan menerapkan algoritma genetika seperti pada tabel berikut:

Tabel 4. 20 *Adjacent Matrix* menampilkan raport

Node		***SEQUENCE***				
		1	**2**	**3**	**4**	**5**
DEPENDENCY	**1**	0	1	0	0	0
	2	0	0	1	0	1
	3	0	0	0	1	1
	4	0	0	0	0	1
	5	0	0	0	0	0

Tabel 4. 21 *Path* yang dihasilkan dengan menerapkan algoritma genetika pada menampilkan raport

No	**Generasi**	***Test Path***	**Nilai *Fitness***
1	1	1-2-5	1
2	1	1-2-3-5	1
3	1	1-2-3-4-5	1

Hasil uji coba berdasarkan *adjacent matrix* menampilkan raport digambarkan seperti pada tabel berikut:

Tabel 4. 22 Hasil uji menampilkan raport

No	**Hasil Uji Sistem**	
	Path	***Test Case***
1	1-2-5	*setSubject, setClass, End*
2	1-2-3-5	*setSubject, setClass, showReport, End*
3	1-2-3-4-5	*setSubject, setClass, showReport, printReport, End*

Dapat diketahui pada tabel 4.22 adalah hasil uji dari *sequence diagram* menampilkan rapor. *Generate path* yang dibangkitkan pada *sequence diagram* menampilkan raport ini menghasilkan tiga *path*.

4.3 Analisa Hasil

Pada penelitian ini dilakukan pembangkitan *test case* secara otomatis pada empat data uji dengan menerapkan metode algoritma genetika. Data uji tersebut

yaitu *Medical Consultation System* pada jurnal dari Priya et al (2013), ATM *System* jurnal Shantiet al (2012), *Aircraft Departure Activity* pada paper Rhmann (2016), dan Sistem Penilaian Pembelajaran. Proses pertama yang dilakukan yaitu membuat *sequence diagram* dari masing-masing data uji dengan menggunakan *software Rasional Rose*. *Sequence diagram* yang telah dibuat disimpan dengan ekstensi .xml. *Sequence diagram* yang telah dibuat dengan menggunakan *Rational Rose* disimpan dalam *file* XML tujuannya adalah agar mudah diproses saat pembangkitan *test case* pada *software Netbeans*.

Pembangkitan *test case* ini merupakan aplikasi berbasis desktop dengan menggunakan *software Netbeans*. Terdapat dua tampilan yang dibangun, yaitu tampilan untuk mengkonversi *file* XML dan tampilan untuk membangkitkan *test case*. Dari *file* XML yang sudah disimpan tadi akan dicari informasi yang dibutuhkan untuk menghasilkan *Sequence Dependency Table* (SDT). Informasi tersebut adalah *symbol, object message*, *sequence* dan *ordinal.* Adanya pengambilan informasi penting ini karena dalam *file* XML memuat banyak informasi yang tidak dibutuhkan dalam pembuatan *Sequence Dependency Table* (SDT). Kemudian *file* XML tersebut disimpan menjadi *file* XML yang baru yang nantinya akan menjadi input untuk membangkitkan *test case*.

Dari *file* XML yang baru tersebut selanjutnya dibentuk *Sequence Dependency Table* (SDT). *Sequence Deependency Table* ini memiliki empat kolom yang berisi *Symbol, Activity Name, Sequence Number* dan *Dependency. Sequence Dependency Table* (SDT) terdiri dari informasi-informasi sebagai berikut:

a. Simbol atau *symbol*: angka atau huruf yang mewakili setiap aktifitas

b. Nama aktifitas atau *activity name:* nama aktifitas yang dilakukan
c. Nomor sequence atau *sequence number*: Nomor urut dari setiap aktifitas
d. *Dependency*: Nomor dari aktifitas yang memiliki keterkaitan dengan aktifitas yang lain.

Proses selanjutnya yaitu membentuk *Adjacent Matrix* menggunakan data dari *Sequence Dependency Table* (SDT) tersebut. Proses pembentukan *adjacent matrix* ini mengambil data pada *dependency* dan *sequence number*, yang mana merupakan hasil dari *Sequence Dependency Table* (SDT). Baris *dependency* sebagai *node* 1 (*node* awal) dan kolom *sequence number* sebagai *node* 2 (*node* akhir). Dalam *adjacent matrix* ini, *node* yang terhubung dengan *node* lainnya bernilai 1 dan *node* yang tidak terhubung bernilai 0.

Setelah membentuk *adjacent matrix*, kemudian membangkitkan *test case* dengan menggunakan metode algoritma genetika. Algoritma genetika memiliki beberapa tahapan didalamnya untuk diterapkan dalam *test case* ini. Tahapan awalnya yaitu inisialisasi individu awal secara random kemudian membentuk populasi. Populasi disini adalah kumpulan dari beberapa individu yang telah dibangkitkan sebelumnya. Tahap selanjutnya yaitu menghitung nilai *fitness*. Nilai *fitness* digunakan untuk menghitung tingkat keberhasilan nilai individu terhadap solusi dari sistem ini dengan mengambil nilai yang paling tinggi. Kemudian memilih individu yang dipertahankan untuk individu selanjutnya melalui *crossover*, kemudian melakukan mutasi untuk menghasilkan individu baru. Hasilnya adalah beberapa *path* yang dibangkitkan dengan otomatis.

Pada data uji *Medical Consultation System*, *test path* yang dibangkitkan berjumlah empat dengan masing-masing *path* memiliki nilai *fitness* tertinggi yaitu

1. Pada data uji ATM *System, test path* yang dibangkitkan berjumlah tujuh dengan masing-masing *path* memiliki nilai *fitness* tertinggi yaitu 1. Pada data uji *Aircraft Departure Activity, test path* yang dibangkitkan berjumlah lima dengan masing-masing *path* memiliki nilai *fitness* tertinggi yaitu 1. Pada data uji Sistem Penilaian Pembelajaran memiliki empat sistem, yaitu sistem untuk menampilkan nama mata kuliah yang mana *test path* yang dibangkitkan berjumlah satu dengan nilai *fitness* tertinggi yaitu 1. Sistem yang kedua yaitu sistem untuk menampilkan nama mahasiswa yang mana *test path* yang dibangkitkan berjumlah satu dengan nilai *fitness* tertinggi yaitu 1. Sistem yang ketiga yaitu sistem untuk perhitungan nilai yang mana *test path* yang dibangkitkan berjumlah tiga dengan masing-masing path memiliki nilai *fitness* tertinggi yaitu 1. Sistem yang keempat yaitu sistem untuk menampilkan raport yang mana *test path* yang dibangkitkan berjumlah tiga dengan masing-masing *path* memiliki nilai *fitness* tertinggi yaitu 1.

4.4 Integrasi Islam

Agama dan Sains merupakan satu keilmuan yang saling berkaitan dan juga utuh. Ilmu pengetahuan tidak akan lepas dari ilmu *Al-Quran* dan *Hadis* yang tidak ada keraguan di dalam nya. Sebagaimana berhubungan dengan pengujian, telah dijelaskan pada ayat *Al-Quran* dalam *Surah Muhammad* ayat 31:

وَلَنَبْلُوَنَّكُمْ حَتَّىٰ نَعْلَمَ الْمُجَاهِدِينَ مِنْكُمْ وَالصَّابِرِينَ وَنَبْلُوَ أَخْبَارَكُمْ

Artinya: *"Dan sesungguhnya Kami benar-benar akan menguji kamu agar Kami mengetahui orang-orang yang berjihad dan bersabar di antara kamu, dan agar Kami menyatakan (baik buruknya) hal ihwalmu."* (*Q.S Muhammad*: 31).

Dalam kitab tafsir Ibnu Katsir Jilid 7 dijelaskan sebagaimana Allah SWT berfirman **وَلَنَبْلُوَنَّكُمْ** *"Dan sesungguhnya Kami benar-benar akan mengujimu,"*

yakni, pasti akan menguji kalian melalui perintah dan larangan, **حَتَّىٰ نَعْلَمَ الْمُجَاهِدِينَ مِنْكُمْ وَالصَّابِرِينَ وَنَبْلُوَ أَخْبَارَكُمْ** *"Sehingga kami menyatakan orang-orang yang berjihad dan bersabar diantara kamu, dan agar Kami menyatakan (baik-buruknya) hal ihwalmu".* Pengetahuan Allah Ta'ala lebih awal atas apa yang akan terjadi itu tidak akan menjadi keraguan. Karena yang dimaksudkan dengan hal itu adalah, sehingga Kami mengetahui kejadiannya. Oleh karena itu, berkenaan dengan hal ini, Ibnu Abbas mengatakan: "Kecuali agar Kami (Allah) mengetahui, maksudnya agar Kami dapat melihat". Dalam Tafsir Jalalain disebutkan bahwa (Dan sesungguhnya Kami benar-benar akan menguji kalian) mencoba kalian dengan berjihad dan lainnya (agar Kami mengetahui) dengan pengetahuan yang tampak (orang-orang yang berjihad dan bersabar di antara kalian) dalam berjihad dan lainnya (dan agar Kami menyatakan) menampakkan (hal ikhwal kalian) tentang ketaatan kalian dan kedurhakaan kalian di dalam masalah jihad dan masalah-masalah lainnya. Ketiga *Fi'il* yang ada dalam ayat ini, ketiga-tiganya dapat dibaca dengan memakai huruf *Mudhara'ah Ya* atau *Nun* (Tafsir Jalalain, 2009).

Ayat *Al-Quran* selanjutnya yang berhubungan dengan pengujian yaitu pada ayat *Al-Quran* dalam *Surah Al-Baqarah* ayat 214:

أَمْ حَسِبْتُمْ أَنْ تَدْخُلُوا الْجَنَّةَ وَلَمَّا يَأْتِكُمْ مَثَلُ الَّذِينَ خَلَوْا مِنْ قَبْلِكُمْ ۖ مَسَّتْهُمُ الْبَأْسَاءُ وَالضَّرَّاءُ وَزُلْزِلُوا حَتَّىٰ يَقُولَ الرَّسُولُ وَالَّذِينَ آمَنُوا مَعَهُ مَتَىٰ نَصْرُ اللَّهِ ۗ أَلَا إِنَّ نَصْرَ اللَّهِ قَرِيب

Artinya: *"Apakah kamu mengira bahwa kamu akan masuk surga, padahal belum datang kepadamu (cobaan) sebagaimana halnya orang-orang terdahulu sebelum kamu? Mereka ditimpa oleh malapetaka dan kesengsaraan, serta digoncangkan*

(dengan bermacam-macam cobaan) sehingga berkatalah Rasul dan orang-orang yang beriman bersamanya: "Bilakah datangnya pertolongan Allah?" Ingatlah, sesungguhnya pertolongan Allah itu amat dekat." (*Q.S Al-Baqarah*: 214).

Dalam kitab tafsir Ibnu Katsir Jilid 1 Allah SWT berfirman, **أَمْ حَسِبْتُمْ أَنْ تَدْخُلُوا الْجَنَّةَ** *"Apakah kamu mengira bahwa kamu akan masuk surga"* Sebelum kamu diuji dan dicoba, sebagaimana yang Allah Ta'ala tumpakan kepada orang-orang yang sebelum kamu. Oleh karena itu, Dia pun berfirman: **وَلَمَّا يَأْتِكُمْ مَثَلُ الَّذِينَ خَلَوْا مِنْ قَبْلِكُمْ ۖ مَسَّتْهُمُ الْبَأْسَاءُ وَالضَّرَّاءُ** *"Padahal belum datang kepadamu (cobaan) sebagaimana halnya orang-orang terdahulu sebelum kamu? Mereka ditimpa oleh malapetaka dan kesengsaraan"* Yaitu berupa berbagai macam penyakit, musibah, dan cobaan.

Dalam *Surah Al-Anbiya* ayat 35, Allah SWT juga berfirman:

كُلُّ نَفْسٍ ذَائِقَةُ الْمَوْتِ ۗ وَنَبْلُوكُمْ بِالشَّرِّ وَالْخَيْرِ فِتْنَةً ۖ وَإِلَيْنَا تُرْجَعُونَ

Artinya: *"Tiap-tiap yang berjiwa akan merasakan mati. Kami akan menguji kamu dengan keburukan dan kebaikan sebagai cobaan (yang sebenar-benarnya). Dan hanya kepada Kamilah kamu dikembalikan."* (*Q.S Al-Anbiya*: 35).

Dalam kitab tafsir Ibnu Katsir Jilid 5 Allah SWT berfirman, **كُلُّ نَفْسٍ ذَائِقَةُ الْمَوْتِ** *"Tiap-tiap yang berjiwa akan merasakan mati".* Firman-Nya, **وَنَبْلُوكُمْ بِالشَّرِّ وَالْخَيْرِ فِتْنَةً** *"Kami akan menguji kamu dengan keburukan dan kebaikan sebagai cobaan"* yaitu Kami terkadang menguji kalian dengan berbagai musibah dan terkadang dengan berbagai nikmat, lalu Kami akan melihat siapa yang bersyukur dan siapa yang kufur serta siapa yang bersabar dan siapa yang putus asa.

Sebagaimana ‘Ali bin Abi Thalhah berkata bahwa Ibnu Abbas berkata: “**وَنَبْلُوكُمْ,** Kami menguji kalian dengan keburukan dan kebaikan sebagai cobaan, yaitu dengan kesulitan dan kelapangan, kesehatan dan penyakit, kaya dan faqir, halal dan haram, taat dan maksiat, petunjuk dan kesesatan. Firman-Nya, **وَإِلَيْنَا تُرْجَعُونَ** **“***Dan hanya kepada Kamilah kamu dikembalikan”* lalu, Kami akan membalas amal-amal kalian.

Allah memberi ujian kepada manusia untuk mengetahui tingkat kesabaran dan baik buruknya seorang hamba. Ujian yang diberikan Allah kepada manusia ada banyak bentuknya, diantaranya yaitu ujian keimanan, ujian harta, dan ujian kesehatan. Seperti pada kisah Nabi Ayyub as. yang diuji oleh Allah berupa ujian yang sangat berat. Nabi Ayyub as. diuji hartanya, keluarganya, dan kesehatannya, namun Nabi Ayyub as. tetap bersabar. Hal ini membuktikan bahwa Nabi Ayyub adalah hamba Allah yang baik, taat dan patuh dalam apapun yang sudah ditetapkan oleh Allah swt.

Sama halnya dengan tema penelitian ini, yaitu pengujian. Pengujian dibuat agar penguji bisa mengetahui seberapa kualitasnya suatu perangkat lunak. Perangkat lunak diuji untuk mengetahui seberapa baik kualitasnya. Pada pengujian, jika pengujian dilakukan dengan baik maka kualitas yang ada pada perangkat lunak juga baik.

BAB V

PENUTUP

5.1 Kesimpulan

Berdasarkan hasil uji coba dan pembahasan yang telah dilakukan, maka kesimpulan yang diperoleh dari rumusan masalah yang melatarbelakangi penelitian ini adalah hasil pengujian dengan menggunakan algoritma genetika bisa membangkitkan *test case* secara otomatis dari masing-masing data uji coba. *Sequence Diagram Medical Consultation System* mendapatkan empat *test case*. *Sequence Diagram ATM System* mendapatkan tujuh *test case*. Sistem *Aircraft Departure Activity* mendapatkan lima *test case*. Sistem evaluasi atau penilaian pembelajaran memiliki empat *sequence diagram* yaitu, *sequence diagram* menampilkan nama mata kuliah menghasilkan satu *test case*, *sequence diagram* nama mahasiswa menghasilkan satu *test case*, *sequence diagram* untuk perhitungan nilai mendapatkan tiga *test case* dan *sequence diagram* menampilkan raport mendapatkan tiga *test case*.

5.2 Saran

Saran yang bisa diberikan adalah peneliti menyadari dalam melakukan penelitian ini terdapat kekurangan yang perlu dikembangkan agar mendapatkan hasil yang lebih baik. Berharap sistem ini dapat digunakan sebagaimana mestinya dan mendapatkan hasil yang maksimal. Berikut beberapa saran yang dapat dijadikan masukan untuk penelitian yang selanjutnya:

1. Pengujian dapat digunakan dan dikombinasikan dengan diagram UML yang lain misalnya pada *activity diagram*, *class diagram* atau *state chart* diagram agar *test case* yang dihasilkan menjadi lebih lengkap.
2. Pada aplikasi setelah membangkitkan *adjacent matrix* terkadang aplikasinya harus di*close* terlebih dahulu sebelum akan membangkitkan data uji yang lain.

DAFTAR PUSTAKA

Abdullah bin Muhammad bin Abdurahman bin Ishaq Al-Syeikh. (2004). *Tafsir Ibnu Katsir Jilid 7*. Bogor. Pustaka Imam asy-Syafi'i.

Alsmadi, I. (2010). *Using Genetic Algorithm for Test Case Generation and Selection Optimozation*. Yarmouk University .

Ariola, W., & Dunlop, C. (2016). *Continous Testing for IT Leaders*. Parasoft .

Ariyani, A. (2017). *Pembangkit Test Case (Kasus Uji) Menggunakan Model UML (Unified Modeling Language) Sequence Diagram (Studi Kasus Sistem Penilaian Pembelajaran)*. UIN Malang .

Biswal, B. N. (2010). *Test Case Generation and Optimization of Object-Oriented Software using UML Behaviour Models*. Department of Computer Science and Engineering National Institute of Technology .

Chuaychoo, N., & Kansomkeat, S. (2017). *Path Coverage Test Case Generation Using genetic Algorithms*. Thailand Prince of Songla University e-ISSN: 2289-8131 Vol. 9, No.2-2 .

Dharwiyanti, S., & Wahono, R. S. (2003). *Pengantar Unified Modeling Language (UML)*. IlmuKomputer.com .

Firmansyah, E. R., Ahmad, S. S., & Agustin, N. H. (2012). *Algoritma Genetika*. Fakultas Sains dan Teknologi Universitas Islam Negeri Syarif Hidayatullah .

Goyal, S., Mishra, P., Lamichhane, A., & Gandhi, P. (2018). *Software Test Case Optimization Using Genetic Algorithm*. International Journal of Scientific Engineering and Science .

Hasling, B. (2008). *Model Based Testing of System Requirement using UML Use Case Models*. ICST .

Hetzel, W. C. (1988). *The Complete Guide of Software Testing Second Edition.* Wellesley, MA, USA: QED Information Sciences, Inc.

Imam Jalaluddin AL-Mahalli dan Imam Jalaluddin As-Suyuti. (2009). *Tafsir Jalalain Jilid 2*. Bandung: Sinar Baru Algensindo.

Javatpoint. (2018). *Software Engineering Information Flow Metrics*. Retrieved November 13, 2019, from javatpoint.com: https://www.javatpoint.com/software-engineering-information-flow-metrics

Kansomkeat, S., & Rivepiboon, W. (2003). *Automated-Generating Test Case Using UML Statechart Diagram*. Chulalongkorn University .

Khurana, N., & Chillar, R. S. (2015). *Test Case Generation and Optimization using UML Models and Genetic Algorithm*. Procedia Computer Science 57 (2015) 996 – 1004 .

Munawar. (2005). *Pemodelan Visual dengan UML.* Yogyakarta: Graha Ilmu.

Myers, G. J. (1979). *The Art of Software Testing John Wiley & Sons.* New York, USA.

Nguyen, V. (2012). *Test Case Point Analysis*. QASymphony, LLC .

Panthi, V., & Mohapatra, D. P. (2012). *Automatic Test Case Generation Using Sequence Diagram*. International Journal of Applied Information System (IJAIS) Vol. 2, No. 4, ISSN: 2249-0868 .

Priya, S. S., & Malarchervi, S. K. (2013). *Test Path Generation Using UML Sequence Diagram*. International Journal of Advanced Research in Computer Science and Software Engineering Vol. 3, Issue 4, ISSN: 2277 128X .

Putri, T. R., Widowati, S., & Hakim, I. L. (2015). *Pembangkitan Kasus Uji untuk Pengujian Aplikasi Berbasis Sequence Diagram*. Universitas Telkom .

Sabharwal, S., Sibal, R., & Sharma, C. (2011). *Applying Genetic Algorithm for Prioritization of Test Case Scenarios Derived from UML Diagrams*. International Journal of Computer Science Issues, Vol. 8, Issue 3, No. 2 .

Samuel, P., Mall, R., & Bothra, A. K. (2008). *Automatic Test Case Generation Using Unified Modeling Language (UML) State Diagrams*. IET Software, Vol. 2, No. 2. ISSN 1751-8806 .

Shanti, A. V., & Kumar, G. M. (2012). *Automated Test Cases Generation from UML Sequence Diagram*. International Conference on Software and Computer Application (ICSCA) IPCSIT Vol. 41 .

Singh, K., Kaur, I., & Kumar, R. (2012). *Automatic Test Case Generation Using Genetic Algorithm with Antirandom Population*. IJACE Vol. 5 .

Srivastava, P. R., & Kim, T.-H. (2009). *Application of Genetic Algorithm in Software Testing*. International Journal of Software Engineering and Its Application Vol. 3, No. 4 .

LAMPIRAN

Proses Algoritma Genetika dalam Software Netbeans

```
Genetika gen = new Genetika();
Populasi pop = new Populasi();
    List<Integer> kromosomUnggul = new ArrayList<>();
    pop = generateKromosom(jumlahIndividu, panjangKromosom);
    kromosomUnggul = steadyState(jumlahGenerasi, pop, jumlahIndividu,
        panjangTournament);
    double x1 = dekodeKromosom(kromosomUnggul.subList(0,
        kromosomUnggul.size() / 2), rMinX1, rMaxX1);
    double x2 = dekodeKromosom(kromosomUnggul.subList(kromosomUnggul.size() /
        2, kromosomUnggul.size()), rMinX2, rMaxX2);
    String fileSDT = jTextArea1.getText();
    String[] s = jTextArea1.getText().split("\n");
    String st = "";
    System.out.println("s" + s.length);

      for (int i = 1; i < s.length; i++) {
      System.out.println("generation :" + s[i]);
      TextAreaGeneratePath.setText("Hasil cross over : "+s[i]);

            st = s[i];
            String[] arrayTeksSplit = st.split("\t");
            System.out.println("Hasil cross over : "+s[i]);
            TextAreaGeneratePath.setText("\n"+"Hasil cross over : "+s[i]);
            for (int j = 0; j < arrayTeksSplit.length; j++) {
               String data1 = arrayTeksSplit[1];
               String data2 = arrayTeksSplit[2];
               String[] data1a = data1.split(" ");
               String[] data2b = data2.split("\"");
```

```
            for (int a = 0; a < data1a.length; a++) {
for (int b = 0; b < data2b.length; b++) {
                String data1aa = data1a[0];
                String data2bb = data2b[1];
                System.out.println("cvcv");
                if (data1aa.length() != 1) {
                   String[] data1aaa = data1a[0].split(",");
                   for (int c = 0; c < data1aaa.length; c++) {
                      String data1mod = data1aaa[c];
                     System.out.println("kromosom terbaik="+data1mod);
                      gen.addEdge(data1mod, data2bb);
                      TextAreaGeneratePath.setText("Hasil cross over
 :"+s[i]+"\n"+"Kromosom Terbaik :"+data1aa+"\n"+"Track Node"
                      + " Populasi :"+"\n");
                   }
                } else {
                   gen.addEdge(data1aa, data2bb);
                }
                break;
             }
          }
```