PEMBANGKIT *TEST CASE* BERDASARKAN MODEL *UNIFIED MODELING LANGUAGE* (UML) *SEQUENCE DIAGRAM* MENGGUNAKAN METODE *SIMULATED ANNEALING*

SKRIPSI

Oleh:
REZA ENDAH PANGESTUTI
NIM. 15650044

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

2020

PEMBANGKIT *TEST CASE* BERDASARKAN MODEL *UNIFIED MODELING LANGUAGE* (UML) *SEQUENCE DIAGRAM* MENGGUNAKAN METODE *SIMULATED ANNEALING*

SKRIPSI

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

Oleh :
REZA ENDAH PANGESTUTI
NIM. 15650044

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2020

LEMBAR PERSETUJUAN

PEMBANGKIT *TEST CASE* BERDASARKAN MODEL *UNIFIED MODELING LANGUAGE* (UML) *SEQUENCE DIAGRAM MENGGUNAKAN* METODE *SIMULATED ANNEALING*

SKRIPSI

Oleh :

REZA ENDAH PANGESTUTI
NIM.15650044

Telah diperiksa dan disetujui untuk Diuji
Tanggal: 18 Desember 2020

Pembimbing I	Pembimbing II
Fatchurrochman, M.Kom	Ainatul Mardhiyah, M.CS
NIP. 19700731 200501 1 002	NIDT. 19860330 20160801 2 075

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

Dr.Cahyo Crysdian
NIP.19740424 200901 1 008

LEMBAR PENGESAHAN

PEMBANGKIT *TEST CASE* BERDASARKAN MODEL *UNIFIED MODELING LANGUAGE* (UML) *SEQUENCE DIAGRAM MENGGUNAKAN* METODE *SIMULATED ANNEALING*

SKRIPSI

Oleh:
REZA ENDAH PANGESTUTI
NIM. 15650044

Telah Dipertahankan di Depan Dewan Penguji
dan Dinyatakan Diterima Sebagai Salah Satu Persyaratan
untuk Memperoleh Gelar Sarjana Komputer (S.Kom)
Pada Tanggal 18 Desember 2020

Susunan Dewan Penguji			**Tanda tangan**
1. Penguji Utama	:	Fajar Rohman Hariri, M.Kom NIP. 19890515 201801 1 001	( )
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	:	Ainatul Mardhiyah, M.CS NIDT. 19860330 20160801 2 075	( )

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 : Reza Endah Pangestuti

NIM : 15650044

Fakultas/Jurusan : Sains dan Teknologi/ Teknik Informatika

Judul Skripsi : Pembangkit *Test Case* Berdasarkan Model *Unified Modeling Language* (UML) *Sequence Diagram* Menggunakan Metode *Simulated Annealing*

Menyatakan dengan sebenarnya bahwa Skripsi yang saya tulis ini benar-benar merupakan hasil karya sendiri, bukan merupakan pengambil alihan 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, 18 Desember 2020
Yang membuat pernyataan,

Reza Endah Pangestuti
NIM. 15650044

HALAMAN MOTTO

"Jangan pernah menyesal dengan segala keputusan yang kau ambil"

HALAMAN PERSEMBAHAN

الْحَمْدُ للهِ رَبِّ الْعَالَمِينَ

Puji syukur atas kehadirat Allah Subhanahu wa ta'ala, dengan mengucapkan Alhamdulillah penulis mempersembahkan sebuah karya untuk orang-orang yang sangat berarti.

Terima kasih penulis ucapkan kepada kedua orang tua yang selalu memberikan cinta, kasih saying, motivasi, doa, harapan dan berusaha untuk memberikan yang terbaik untuk segala aspek dan tahap kehidupan penulis, Bapak Kuswari dan Ibu Roisah. Tidak lupa adik Renata yang selalu memberikan motivasi untuk segera lulus dan membuka tahap selanjutnya dalam hidup.

Terimakasih pula saya ucapkan untuk pembimbing saya Bapak Fatchurrochman dan Bu Ainatul Mardhiyah yang telah membimbing dalam melakukan penilitian ini dan memberikan motivasi serta dorongan hingga penelitian ini dapat terselesaikan dengan lancar.

Tidak lupa terimaksih saya ucapkan kepada teman-teman satu perjuangan jurusan Teknik Informatika 2015 UIN Maulana Malik Ibrahim Malang yang telah menemani dan mengisi hari-hari selama 5 tahun terakhir. Dan kepada teman – teman yang selalu memberikan semangat dan bantuan dalam mengerjakan penelitian ini Abdush Shomad, Berlian Gita, M. Fadhil, Prisna Anjar L., Alifiyah P., Ainun N., Syahrul K., Komarudin, Muqta, dan Silva.

Terima kasih untuk orang – orang yang tidak dapat disebutkan satu per satu yang telah memberikan motivasi, semangat dan doa sehingga penelitian skripsi ini dapat rampung dengan lancar.

KATA PENGANTAR

بِسْمِ اللهِ الرَّحْمٰنِ الرَّحِيْمِ

Segala puji bagi Allah SWT, karena atas rahmat, hidayah dan karunia-Nya, penulis dapat menyelesaikan skripsi yang berjudul "Pembangkit *Test Case* Berdasarkan Model *Unified Modeling* Language (UML) *Sequence Diagram* Menggunakan Metode *Simulated Annealing*" sebagai salah satu syarat untuk memperoleh gelar sarjana pada Program Studi Teknik Informatika pada jenjang Strata-1 Universitas Islam Negeri Maulana Malik Ibrahim Malang.

Shalawat serta salam senantiasa selalu terlimpahkan kepada junjungan Nabi Muhammada SAW, keluarga dan para sahabat yang telah membimbing umat dari zaman kebodohan yaitu zaman jahiliyah menuju jalan yang diridzoi Allah SWT.

Penulis menyadari banyak keterbatasan yang penulis miliki, sehingga banyak pihak yang telah memberikan bantuan baik moril maupun materil dalam proses menyelesaikan penelitian ini. Maka dari itu dengan segenap kerendahan hati penulis mengucapkan terimakasih kepada :

1. Prof Dr H Abd. Haris, M.Ag selaku rektor UIN Maulana Malik Ibrahim Malang.
2. Dr. Sri Harini, M.Si selaku Dekan Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.
3. Dr. Cahyo Crysdian selaku Ketua Jurusan Teknik Informatika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.

4. Bapak Fatchurrochman, M.Kom selaku pembimbing I dan Ibu Ainatul Mardhiyah, M.CS selaku pembimbing II yang senantiasa meluangkan waktu untuk membimbing, mengarahkan, dan memberi masukkan kepada penulis.
5. Seluruh Dosen Jurusan Teknik Informatika Fakultas Sain dan Teknologi UIN Maulana Malik Ibrahim Malang yang telah memberikan ilmu dan pengetahuan serta pengalaman yang sangat berharga dan bermanfaat.
6. Segenap civitas akademik Jurusan Teknik Informatika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.
7. Kedua orang tua serta seluruh keluarga besar penulis yang selalu dengan senantiasa mendukung dan medoakan penulis.
8. Sahabar-sahabat seperjuangan Jurusan Teknik Informatika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.

Penulis menyadari dalam karya ini masih banyak kekurangan. Oleh karena itu penulis selalu menerima segala kritik dan saran dari pembaca. Semoga karya ini dapat bermanfaat bagi seluruh pihak.

Malang, 18 Desember 2020

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

ABSTRAK

Pangestuti, Reza Endah. 2020. **Pembangkit *Test Case* Berdasarkan Model *Unified Modeling Language* (UML) *Sequence Diagram* Menggunakan Metode *Simulated Annealing*.** Skripsi. Jurusan Teknik Informatika Fakultas Sains Dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang. Pembimbing : (I) Fatchurrochman, M.Kom (II) Ainatul Mardhiyah, M.CS

Kata Kunci : Pembangkit *Test Case*, UML *Sequence Diagram, Simulated Annealing, Test Path.*

Pengujian perangkat lunak merupakan tahap yang sangat penting dalam mengembangkan sebuah perangkat lunak untuk mengetahui kualitas dari perangkat lunak yang dihasilkan. Proses pengujian perangkat lunak membutuhkan waktu yang lama sehingga dibutuhkannya pengujian perangkat lunak secara otomatis yang dapat meningkatkan efisiensi proses pengujian baik dalam hal waktu maupun keuangan. Penelitian ini berupaya membangkitkan *test case* secara otomatis berdasarkan model UML *sequence diagram* menggunakan metode *simulated annealing*. UML *sequence diagram* memudahkan dalam menganalis dan memvisualisasikan bebagai rancangan dari suatu sistem. Penggunaan metode *simulated annealing* bertujuan untuk menghasilkan *test path* terbaik dari masing-masing uji coba. Data uji coba dalam penelitian ini yaitu *sequence diagram* Sistem ATM, *sequence diagram* Sistem Konsultasi Medis, *sequence diagram* Aktifitas Pemberangkatan Pesawat, *sequence diagram* Sistem Keamanan *Smart Home* dan tiga *sequence diagram* dari Sistem Penilaian Pembelajaran. Setelah melakukan semua proses uji coba, penelitian ini berhasil membangkitkan *test case* secara otomatis.

ABSTRACT

Pangestuti, Reza Endah. 2020. **Test Case Generator Based on Unified Modeling Language (UML) Sequence Diagram Using Simulated Annealing Method.** Thesis. Department of Informatics, Faculty of Science and Technology Maulana Malik Ibrahim State Islamic University of Malang. Counselo: (I) Fatchurrochman, M.Kom (II) Ainatul Mardhiyah, M.CS.

Keywords: *Test Case Generator, UML Sequence Diagram, Simulated Annealing, Test Path.*

Software testing is a very important stage in developing a software to determine the quality of the software produced. The software testing process takes a long time so it takes automatic software testing that can improve the efficiency of the testing process both in terms of time and finance. This study seeks to generate test cases automatically based on the UML sequence diagram model using the simulated annealing method. UML sequence diagrams make it easy to analyze and visualize various designs of a system. The use of the simulated annealing method aims to produce the best test path from each trial. The test data in this study are ATM System sequence diagram, Medical Consultation System sequence diagram, Aircraft Departure Activity sequence diagram, Smart Home Security System sequence diagram and three sequence diagrams from Learning Assessment System. After carrying out all the testing processes, this research succeeded in generating test cases automatically.

الملخص

بانجستوتي ، رضا إنده. ٢٠٢٠. **مولد حالة الاختبار بناءً على نموذج تسلسل نموذج لغة النمذجة الموحدة باستخدام طريقة التلدين المحاكاة**. مقال. قسم المعلوماتية ، كلية العلوم والتكنولوجيا ، جامعة الدولة الإسلامية مولانا مالك إبراهيم مالانج. المستشارون: (١)فاتشورو همان, لماجستير(٢)عيناتول مردية ,لماجستير

الكلمات الدالة: مولد حالة الاختبار ، مخطط التسلسل ، التلدين المحاكي ، مسار الاختبار

يعد اختبار البرامج مرحلة مهمة للغاية في تطوير البرنامج لتحديد جودة البرنامج المنتج. تستغرق عملية اختبار البرنامج وقتًا طويلاً ، لذا فهي تحتاج إلى اختبار آلي للبرنامج يمكن أن يزيد من كفاءة عملية الاختبار من حيث الوقت والمال. تسعى هذه الدراسة إلى إنشاء حالات اختبار تلقائيًا بناءً على نموذج مخطط تسلسلي باستخدام طريقة التلدين المحاكاة. تسهل مخططات التسلسل تحليل وتصور التصميمات المختلفة للنظام. يهدف استخدام طريقة التلدين المحاكاة إلى إنتاج أفضل مسار اختبار من كل تجربة. بيانات التجربة في هذه الدراسة هي مخططات تسلسل أنظمة أجهزة الصراف الآلي ، ومخططات تسلسل نظام الاستشارات الطبية ، ومخططات تسلسل نشاط مغادرة الطائرات ، ومخططات تسلسل نظام أمن المنزل الذكي وثلاثة مخططات تسلسل لنظام تقييم التعلم. بعد إجراء جميع عمليات الاختبار ، نجحت هذه الدراسة في توليد حالات الاختبار تلقائيًا.

BAB I

PENDAHULUAN

1.1. Latar Belakang

Perkembangan pengguna komputer yang meluas menjadikan banyaknya kebutuhan perangkat lunak yang bertambah sehingga organisasi berlomba untuk mengembangankan perangkat lunak. Pengujian perangkat lunak merupakan tahap yang sangat penting dalam mengembangkan sebuah perangkat lunak. Menurut (Galin, 2004) pengujian perangkat lunak atau *software testing* diartikan sebagai proses formal dimana suatu perangkat lunak diuji dengan cara menjalankan perangkat lunak tersebut dalam komputer dan menjalankan prosedur serta kasus tertentu. (Galin, 2004) menyatakan bahwa terdapat hubungan langsung yang erat antara pengujian perangkat lunak dengan kualitas perangkat lunak yang dihasilkan, sehingga pengujian perangkat lunak menjadi tahapan yang sangat penting dalam siklus pengembangan perangkat lunak.

Dalam proyek pengembangan perangkat lunak, lebih dari 50% pengembangan perangkat lunak dihabiskan untuk pengujian dalam hal waktu maupun keuangan (Tripathy, 2012). Penelitian (Catelani, 2010) menunjukkan bahwa pengujian perangkat lunak secara otomatis dapat meningkatkan efisiensi proses pengujian untuk mengidentifikasi bagian dari perangkat lunak yang rawan mengalami kegagalan. *Test case* merupakan suatu rancangan atau rangkaian mengenai tindakan yang dilakukan oleh *user* untuk melakukan verifikasi terhadap fitur atau fungsi tertentu dari sebuah perangkat lunak. Seiring waktu, fitur yang ada pada system akan terus bertambah, *Quality Assurance* (QA) akan melakukan

pengujian mulai dari fitur yang sudah ada ditambah dengan fitur baru yang sedang dikembangkan. Rangkaian *test* ini sangat memakan waktu dan pekerjaan yang membosankan untuk *Quality Assurance* (QA) jalankan secara manual setiap kali, ini lah tanda-tanda pembangunan *test case* secara otomatis diperlukan untuk menjaga konsistensi hasil pengujian. Dengan adanya *test case* secara otomatis ini diharapkan beban *Quality Assurance* (QA) menjadi ringan dalam menjalankan *regression test* yang sama berulang kali. Hal ini dikarenakan metode-metode algoritma yang diterapkan pada pembuatan *test case* dapat membantu membuat *test case* yang sesuai dengan kebutuhan (Khan & Amjad, 2015).

Pada Al-qur'an juga dijelaskan dalam firman-Nya:

اَحَسِبَ النَّاسُ اَنْ يُّتْرَكُوْٓا اَنْ يَّقُوْلُوْٓا اٰمَنَّا وَهُمْ لَا يُفْتَنُوْنَ (٢) وَلَقَدْ فَتَنَّا الَّذِيْنَ مِنْ قَبْلِهِمْ فَلَيَعْلَمَنَّ اللّٰهُ الَّذِيْنَ صَدَقُوْا وَلَيَعْلَمَنَّ الْكٰذِبِيْنَ (٣)

Artinya: *"Apa manusia mengira bahwa mereka akan dibiarkan hanya dengan mengatakan: "Kami telah beriman", mereka tidak diuji lagi?. Sesungguhnya, Kami telah menguji orang-orang sebelum mereka, maka sesungguhnya Allah pasti mengetahui orang-orang yang benar dan pasti mengetahui orang-orang yang dusta." (QS. Al-Ankabut 29:02-03).*

Tafsir Ibnu Katsir Jilid VI menjelaskan Firman Allah SWT, اَحَسِبَ النَّاسُ اَنْ يُّتْرَكُوْٓا اَنْ يَّقُوْلُوْٓا اٰمَنَّا وَهُمْ لَا يُفْتَنُوْنَ *"Apa manusia mengira bahwa mereka akan dibiarkan hanya dengan mengatakan: "Kami telah beriman", mereka tidak diuji lagi."* Adalah bentuk istifham inkari (pertanyaan yang bersifat mengingkari). Maknanya bahwa Allah SWT harus menguji hamba-hambaNya yang beriman sesuai keimanan yang mereka miliki. وَلَقَدْ فَتَنَّا الَّذِيْنَ مِنْ قَبْلِهِمْ فَلَيَعْلَمَنَّ اللّٰهُ الَّذِيْنَ صَدَقُوْا وَلَيَعْلَمَنَّ الْكٰذِبِيْنَ *"Sesungguhnya, Kami telah menguji orang-orang sebelum mereka, maka sesungguhnya Allah pasti mengetahui orang-orang yang benar dan pasti mengetahui orang-orang yang dusta"* maknanya orang yang jujur dalam

keimanannya dari orang yang dusta dalam perkataan dan pengakuannya. Allah SWT Maha Mengetahui apa yang telah ada dan apa yang akan ada, apa yang belum ada dan bagaimana adanya.

Pada potongan ayat telah dijelaskan, bahwa Allah SWT mengetahui amalan yang diperbuat oleh hambaNya dengan cara mengujinya. Sedangkan dalam penelitian ini, pengujian dilakukan untuk mengetahui kualitas dari perangkat lunak yang sedang dikembangkan. Semakin baik pengujian yang dilakukan maka semakin baik pula kualitas dari perangkat lunak tersebut.

Berdasarkan penjelasan di atas, maka penulis akan membuat pembangkit *test case* untuk pegujian perangkat lunak menggunakan model UML *sequence diagram* dengan metode *simulated annealing*. UML merupakan standar yang digunakan untuk menganalisis dan memodelkan kebutuhan sehingga memudahkan menganalis dan memvisualisasikan bebagai rancangan dari suatu sistem. Sebelum membangkitkan aplikasi pembangkit *test case* otomatis, pada proses desain UML *sequence diagram* peneliti akan melakukan *generate test case* untuk mengetahui sistem ini sudah sesuai dengan yang direncanakan atau belum. Sehingga ketika membangun aplikasi apabila peneliti menemukan kesalahan akan lebih cepat terdeteksi. Pembuatan aplikasi pembangkit *test case* otomatis ini bertujuan mempermudah pengembang perangkat lunak dalam mengetahui *test case* dalam proses pengujian. Penggunaan metode *Simulated Annealing* dalam aplikasi ini dikarenakan *simulated annealing* mampu bertahan menghadapi lokal optima dan proses pencariannya dikendalikan oleh suhu (Sofianti, 2004). Lokal optima yaitu keadaan tetangga lebih buruk atau sama dengan keadaan dirinya.

Diharapkan menggunakan *simulated annealing* dapat menyelesaikan permasalahan yang terjadi yakni mendapatkan *path* terbaik.

1.2. Pernyataan Masalah

Berdasarkan identifikasi latar belakang tersebut, maka ditentukan rumusan masalah dari penelitian ini adalah bagaimana kinerja *Simulated Annealing* dalam membangkitkan *test case* secara otomatis dari UML *Sequence Diagram*?

1.3. Tujuan Penelitian

Membangkitkan *test case* secara otomatis menggunakan *simulated annealing* sehingga diharapkan dapat mempercepat waktu pengujian perangkat lunak.

1.4. Manfaat Penelitian

Adapun manfaat dari penelitian ini yaitu proses pengujian perangkat lunak menjadi lebih cepat.

1.5. Batasan Masalah

Adapun penelitian ini memiliki batasan-batasan masalah sebagai berikut :

1. Pembuatan *Sequence Diagram* menggunakan aplikasi Star UML
2. Pembangunan *test case* menggunakan aplikasi *Netbeans.*
3. *Sequence Diagram* yang digunakan untuk objek pengujian yaitu *sequence diagram* pada *paper* Shanti et al (2012), Priya et al (2013), Rhmann (2016), Mani & Prasanna (2017), dan Sistem Penilaian Pembelajaran.

1.6. Sistematika Penulisan

Laporan penelitian terdiri dari lima bab, isi dari setiap bab terdiri dari :

BAB I : PENDAHULUAN

Berisi latar belakang dari masalah yang akan diteliti, pernyataan masalah, tujuan dan manfaat penelitian dari penelitian, batasan masalah penelitian, metodologi penelitian, serta sistematika penulisan laporan.

BAB II: STUDI PUSTAKA

Bab ini berisi mengenai penelitian yang telah dilakukan ataupun teori dasar dan data-data terkait dengan pembangkitan *test case* dalam pengujian perangkat lunak.

BAB III : METODE PENELITIAN

Bab ini berisi metode penelitian yang menjelaskan bagaimana penelitian ini dijalankan. Meliputi hasil analisa dan rincian langkah yang digunakan dalam pembangkitan *test case* model UML *Sequence Diagram* dengan menerapkan metode *Simulated Annealing.*

BAB I V : UJI COBA DAN PEMBAHASAN

Bab ini berisi uji coba aplikasi yang telah dibuat dan dilakukan pembahasan terhadap proses pembangkit *test case*. Selanjutnya membahas tentang hasil yang di peroleh dari mengimplementasikan *Simulated Annealing.*

BAB V : KESIMPULAN DAN SARAN

Pada bab ini tentang kesimpulan dari penelitian yang telah dilakukan, saran dan kritik dari penelitian agar dapat dikembangkan pada penelitian selanjutnya.

BAB II

STUDI PUSTAKA

Studi pustaka membahas tentang kajian *paper* pendukung sebelumnya dan beberapa teori yang menjadi dasar dalam penyusunan skripsi.

2.1. Aplikasi Berorientasi Objek

Aplikasi berorientasi objek adalah aplikasi yang dibangun menggunakan pemrograman berorientasi objek. Pemrograman berorientasi objek atau *object-oriented programing* (OOP) merupakan suatu pendekatan pemrograman yang menggunakan *object* dan *class*. Berorientasi obyek merupakan paradigma baru dalam rekayasa perangkat lunak. Paradigma ini memandang sistem sebagai kumpulan obyek-obyek diskrit yang saling berinteraksi satu sama lain. Berorientasi objek juga bias bermakna kegiatan mengorganisasikan perangkat lunak sebagai kumpulan obyek-obyek diskrit yang bekerja sama antara informasi (atau struktur data) dan perilaku (*behavior)* yang mengaturnya. Salah satu keuntungan utama dari paradigma berorientasi objek adalah kemampuannya untuk membangun komponen sekali saja, kemudian dapat menggunakannya berulang-ulang (Sholiq, 2010).

2.2. Unified Modelling Language (UML)

Unified Modelling Language (UML) adalah sebuah "bahasa" yang telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. Bahasa pemodelan perangkat lunak UML, pertama kali diperkenalkan pada tahun 1997, saat ini telah berkembang menjadi sebuah bahasa pemodelan yang baku (*de facto)* di dalam sebuah

pengembangan perangkat lunak (Engels, et al., 2000). UML menawarkan sebuah standar untuk merancang model sebuah sistem.

Dengan menggunakan UML kita 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 lebih cocok untuk penulisan piranti lunak dalam bahasa-bahasa berorientasi obyek seperti C++, Java, C# atau VB.NET (Sulistyorini, 2009). UML merupakan kesatuan dari bahasa pemodelan yang dikembangkan oleh Grady Booch *Object Oriented Design* (OOD), Jim Rumbaugh *Object Modeling Technique* (OMT), dan Ivar Jacobson *Object Oriented Software Engineering* (OOSE) (Munawar, 2005).

Tujuan UML diantaranya adalah (Munawar, 2005):

a. Memberikan model yang siap pakai, bahasa pemodelan *visual* yang ekspresif untuk mengembangkan dan saling menukar model dengan mudah dan dimengerti secara umum.
b. Memberikan bahasa pemodelan yang bebas dari berbagai bahasa pemrograman dan proses rekayasa.
c. Menyatukan praktek-praktek terbaik yang terdapat dalam pemodelan.

Beberapa diagram yang disediakan dalam UML antara lain:

a. *Use Case Diagram*,
b. Diagram Kelas,
c. Diagram Objek,
d. *Sequence Diagram*

e. *Collaboration Diagram*

f. *Statechart Diagram*

g. *Activity Diagram*

h. *Component Diagram*

i. *Deployment Diagram*

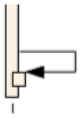
2.3. Sequence Diagram

Sequence diagram adalah grafik dua dimensi dimana obyek ditunjukkan dalam dimensi horizontal sedangkan *lifeline* ditunjukkan dalam dimensi vertikal. Urutan *message* ditunjukkan dari atas ke bawah. Biasanya *Sequence diagram* dibuat untuk setiap *use case*. *Message* bisa ditambahkan argumen. Argumen bisa berupa *input argument* (dari pengirim ke target) atau *output argument* (dari target kembali ke pengirim). *Input argument* ditandai dengan kata kunci *out* (Munawar, 2005).

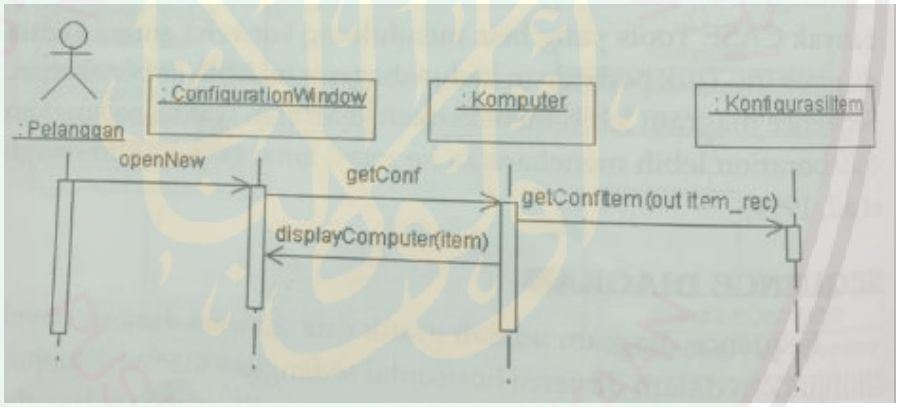
Simbol-simbol yang digunakan dalam *sequence diagram* yaitu (Hendini, 2016):

Tabel 2. 1 Simbol-simbol *Sequence Diagram*

No	Gambar	Keterangan
1		*Entity Class*, merupakan bagian dari sistem yang berisi kumpulan kelas berupa entitas-entitas yang membentuk gambaran awal sistem dan menjadi landasan untuk menyusun basis data.
2		*Boundary Class*, berisi kumpulan kelas yang menjadi *interface* atau interaksi antara satu atau lebih aktor dengan sistem, seperti tampilan form entry dan form cetak.
3		*Conrol class*, suatu objek yang berisi logika aplikasi yang tidak memiliki tanggung jawab kepada entitas, contohnya adalah kalkulasi dan aturan bisnis yang melibatkan berbagai objek.
4		*Message*, symbol mengirim pesan antar *class*.

5		*Recursive,* yakni menggambarkan pengiriman pesan yang dikirim untuk dirinya sendiri.
6		*Activation,* mewakili sebuah eksekusi operasi dari obyek, panjang kotak ini berbanding lurus dengan durasi aktivasi sebuah operasi.
7		*Lifeline,* garis titik-titik yang terhubung dengan obyek, sepanjang *lifeline* terdapat *activation.*

Berikut Gambar 2.1 adalah contoh *sequence diagram* untuk *"Display Curren Configuration"* (Munawar, 2005):



Gambar 2. 1 *Sequence diagram* untuk aktivitas *"Display Curren Configuration"*

Pada Gambar 2.1 dijelaskan bahwa saat aktor dari luar (pelanggan) memilih menu *display current configuration*, sebuah *message OpenNew* dikirim ke *boundary class: ConfigurationWindow*. Hasilnya adalah membuat *(instantiate)* obyek: *ConfigurationWidow* yang baru. Obyek: *ConfigurationWindow* perlu untuk menampilkan dirinya sendiri dengan data-data konfigurasi. Oleh karena itulah sebuah *message* dikirim ke obyek: Komputer. Faktanya: Komputer adalah

obyek dari *class StandardComputer* atau *ConfiguredComputer*. Jadi Komputer adalah abstrak *class.*

Obyek: Komputer menggunakan output argument *item_rec* untuk mengkomposisi dirinya sendiri dari obyek: *KonfigurasiItem*. Hasilnya adalah konfigurasi item yang dikirimkan ke: *ConfigurationWindow* sebagai argumen input item dari *message DisplayComputer*. Obyek: *ConfigurationWindow* sekarang bisa menampilkan dirinya sendiri.

2.4. *Graph*

Graph G didefinisikan sebagai pasangan himpunan *(V, E)*, ditulis dengan notasi *G = (V, E)*, yang dalam hal ini *V* adalah himpunan tidak kosong dari simpul-simpul *(vertices* atau *node)* dan *E* adalah himpunan sisi *(edges* atau *arcs)* yang menghubungkan sepasang simpul. Definisi diatas menyatakan bahwa *V* tidak boleh kosong, sedangkan *E* boleh kosong. Jadi, sebuah *graph* dimungkinkan tidak mempunyai sisi satu buah pun, tetapi simpulnya harus ada, minimal satu. *Graph* yang hanya mempunyai satu buah simpul tanpa sebuah sisi pun dinamakan graf trivial.

Dalam bidang rekayasa peraangkat lunak, sebuah program harus mengalami tahap pengujian untuk menemukan kesalahan *(bug)*. Salah satu pengujian program adalah pengujian eksekusi. Aliran kendali program harus diperiksa untuk memastikan apakah aliran tersebut sudah benar untuk berbagai kasus data uji. Aliran kendali program dimodelkan dengan graph berarah yang dinamakan graf alir *(flow graph)*. Pada *graph* berarah tersebut, simpul menyatakan pernyataan atau kondisi yang dievaluasi, sedangkan busur menyatakan aliran kendali program ke pernyataan atau kondisi berikutnya (Munir, 2014).

2.5. Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah proses menjalankan dan mengevaluasi sebuah perangkat lunak secara manual maupun otomatis untuk menguji apakah perangkat lunak sudah memenuhi persyaratan atau belum (Clune dan Rood, 2011). Menurut Jin dan Xue (2011) menyatakan bahwa pengujian bermaksud untuk mencari sebanyak mungkin kesalahan yang ada pada program serta mengevaluasi kualitasnya. Tujuan perangkat lunak menurut Jin dan Xie (2011) adalah menilai apakah perangkat lunak yang dikembangkan telah memenuhi kebutuhan pemakai, menilai apakah tahap pengembangan perangkat lunak telah sesuai dengan metodologi yang digunakan, dan membuat dokumentasi hasil pengujian yang menginformasikan kesesuaian perangkat lunak yang diuji dengan spesifikasi yang telah ditentukan. Data yang dikumpulkan pada saat pengujian dilakukan akan memberikan indikasi yang baik mengenai reliabilitas dan kualitas perangkat lunak secara keseluruhan.

Beberapa aturan yang dapat digunakan sebagai penjelasan tentang pengujian perangkat lunak adalah sebagai berikut:

a. Pengujian adalah proses eksekusi suatu program dengan maksud menemukan kesalahan.
b. *Test case* yang baik adalah *test case* yang memiliki probabilitas tinggi untuk menemukan kesalahan yang belum pernah ditemukan sebelumnya.
c. Pengujian yang sukses adalah pengujian yang mengungkapkan semua kesalahan yang belum pernah ditemukan sebelumnya.

d. Suatu pengujian harus mengacu pada suatu resiko-resiko pengembangan sistem.

Maka dapat disimpulkan bahwa pengujian yang baik tidak hanya ditujukan untuk menemukan kesalahan pada perangkat lunak tetapi juga untuk dapat menemukan data uji yang dapat menemukan kesalahan secara lebih teliti. Ada pendekatan pengujian perangkat lunak yakni pengujian *White-Box Testing*.

2.6. White Box Testing

Pengujian *White-box* adalah metode desain *test case* menggunakan struktur control desain prosedural untuk memperoleh *test case*. Dengan menggunakan metode pengujian *white-box*, perekayasa sistem dapat melakukan *test case* yang:

a. Memberikan jaminan bahwa semua jalur independen dari suatu modul telah digunakan paling tidak hanya satu kali.
b. Menggunakan keputusan logis pada sisi *true* dan *false*.
c. Mengeksekusi semua *loop* pada batasan dan pada batas operasional.
d. Menggunakan struktur data internal untuk menjamin keakuratannya (Pressman, 2002).

Teknik yang dapat digunakan yaitu teknik *Basis-path testing*. Pengujian *Basis-path* adalah teknik pengujian *white-box* yang diusulkan pertama kali oleh Tom McCabe. Metode *basis path* ini memungkinkan desainer *test case* mengukur kompleksitas logis dari desain prosedural dan menggunakannya sebagai pedoman untuk menetapkan basis set dari jalur eksekusi (Pressman, 2002). Langkah yang harus ditempuh pada teknik *Basis-path testing* adalah sebagai berikut:

a. Pembuatan *Flowgraph*

Flowgraph adalah alur dari logika program. Notasi pada *flowgraph* terdiri atas lingkaran dan panah. Lingkaran (*node)* menyatakan pernyataan prosedural. Panah (*edge*) menyatakan aliran kendali atau alur perjalanan logika.

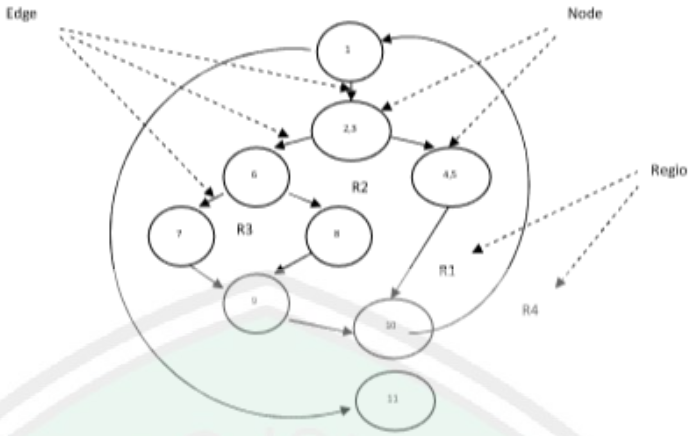
Gambar 2. 2 *Flowgraph* (Pressman, 2002)

b. Penghitungan *Cyclomatic Complexity*

Cyclomatic Complexity (CC) adalah metriks perangkat lunak yang memberikan pengukuran kuantitatif terhadap kompleksitas logis suatu program. CC dihitung dalam salah satu dari tiga cara berikut:

1. Jumlah *region* grafik alir sesuai dengan CC.
2. *Cyclomatic Complexity*, *V(G)*, untuk grafik alir *G* ditentukan sebagai *V(G) = E – N + 2* dimana *E* adalah jumlah *edge* grafik alir dan *N* adalah jumlah simpul grafik alir.
3. *Cyclomatic complexity*, *V(G)*, untuk grafik alir *G* juga ditentukan sebagai *V(G) = P + 1*, dimana *P* adalah jumlah simpul predikat yang diisikan dalam grafik alir *G*.

   Contohnya:



Gambar 2. 3 Grafik Alir (Pressman, 2002)

Kompleksitas siklomatis dapat dihitung dengan menggunakan masing-masing dari algoritma yang ditulis di atas:

1. Grafik alir mempunyai 4 *region.*
2. *V(G)* = 11 *edge* – 9 simpul + 2 = 4.
3. *V(G)* = 3 simpul yang diperkirakan + 1 = 4.

   Dengan demikian, kompleksitas siklomatis dari grafik alir pada gambar diatas adalah 4.

c. Penentuan *Independent Path*

Independent Path adalah jalur yang melalui program yang mengintroduksi sedikitnya satu rangkaian statemen proses baru atau suatu kondisi baru. Bila dinyatakan dengan terminologi grafik alir, jalur independen harus bergerak sepanjang paling tidak satu *edge* yang tidak dilewatkan sebelum jalur tersebut ditentukan. Contohnya, untuk serangkaian *Independent Path* untuk grafik alir yang ditunjukkan pada gambar adalah:

Jalur 1 : 1-11

Jalur 2 : 1-2-3-4-5-10-1-11

Jalur 3 : 1-2-3-6-8-9-10-1-11

Jalur 4 : 1-2-3-6-7-9-10-1-11

d. Menyiapkan *test case* yang akan menjadi tempat eksekusi setiap basis set yang terdiri dari *path-path* yang tersusun (Pressman, 2002).

2.7. Automated Testing

Automated Testing merupakan proses pengujian yang digunakan untuk mempermudah proses pengujian dan mengefektifkan proses pengeksekusian dan pengukuran kualitas pada pengujian (Novelia, 2008). *Automated testing* adalah pengujian perangkat lunak secara otomatis dengan menggunakan alat pengujian dari perangkat lunak yang dibuat. *Automated Testing* memiliki banyak manfaatnya baik dari segi biaya maupun sumber daya. Faktor yang mendukung pengujian perangkat lunak ini berkembang antara lain: menghemat biaya pengembangan, waktu pengujian menjadi singkat, meningkatnya kecermatan dalam pelaksanaan pengujian, meningkatnya akurasi pengujian, dan meningkatkan hasil pengujian seperti proses statistik (Rina, 2009).

2.8. Test Case dan Test Path

Test case umumnya terdiri dari kumpula nilai input,prasarat eksekusi, hasil yang diharapkan, dan kondisi setelah eksekusi, yang dibuat uantuk tujuan atau kondisi pengujian tertentu. Diperlukan lebih dari satu uji kasus untuk menguji fungsionalitas penuh aplikasi GUI. Kumpulan test case disebut test suite. Rangkaian uji berisi panduan terperinci atau tujuan untuk setiap kumpulan kasus uji (Isabella & Retna, 2012). Berikut contoh *test case:*

Tabel 2. 2 Contoh *Test Case*

No	*Test Case*	*Pre-requisites*	*Step*	*Expected Result*
1	*Login*	*User* mengklik 'Login dengan email' pada *button* di *main screen*, *user* sudah teregristasi.	1. *User* membuka app. 2. *User* memilih '*Login* dengan email'. 3. *User* memasukkan email yang benar. 4. *User* memasukkan *password* yang benar. 5. *User* mengklik '*Login'*.	Menampilkan halaman *home*.
2	*Leaving one or more fields to empty*	*User* mengklik '*Login* dengan email' pada *main screen*.	*1. User* membuka app. *2. User* memilih '*Login* dengan email'. *3. User* mengosongkan email atau *password*. *4. User* mengklik '*Login'*.	Menampilkan simbol *error* dan pesan *error* karena *error*.

Test Path merupakan salah satu metode pengujian struktural yang menggunakan kode program untuk menemukan semua jalur yang mungkin dapat dilalui program dan dapat digunakan untuk merancang data uji. Metode ini memastikan semua kemungkinan jalur dijalankan setidaknya satu kali (Basu, 2015). Contoh *test path* dapat dilihat pada sub bab 2.6.

2.9. Simulated Annealing

Sesuai dengan namanya, *Simulated Annealing (SA)* mensimulasikan proses *Annealing* dipembuatan materi yang terdiri dari butiran kristal *(glassy)* atau logam. Tujuan proses ini adalah menghasilkan struktur kristal yang terbaik dengan menggunakan energi seminimal mungkin. Pada tahun 1983, Kirkpatrick dan koleganya menggunakan ide dari algoritma Metropolis dan mengaplikasikannya pada permasalahan optimasi. Idenya adalah bagaimana menggunakan *Simulated Annealing* untuk mencari solusi-solusi yang layak dan konvergen pada solusi optimal (Suyanto, 2010).

Persyaratan *Simulated Annealing:*

1. 1 2 4 3 5 dapat diterima sebagai *state*

2. 1 4 2 5 tidak dapat diterima sebagai *state* karena titik ketiga tidak ada
3. 1 4 2 3 2 5 tidak diterima sebagai *state* karena titik 2 kembar
4. 2 5 3 1 4 diterima sebagai *state.*

Algoritma *Simulated Annealing* :

1. Evaluasi keadaan awal. Jika keadaan awal merupakan tujuan, maka pencarian berhasil dan KELUAR. Jika tidak demikian, lanjutkan dengan menetapkan keadaan awal sebagai kondisi sekarang menentukannya dengan cara random dan dihitung nilainya (Z_c).
2. Inisialisasi *T* sesuai dengan *annealing schedule*.
3. Jalankan iterasi dengan menukarkan tetangga berikutnya dari keadaan random tadi dengan cara membangkitkan bilangan random pada range [0,1]. Evaluasi keadaan baru dengan cara berikut :
   a. Jika keadaan baru merupakan tujuan, maka lanjutkan iterasi hingga maksimum jumlah iterasi yang dilakukan.
   b. Jika keadaan baru merupakan tujuan, namun memiliki nilai yang lebih baik dari keadaan sekarang $(Z_n \leq Z_c)$ maka tetapkan keadaan baru sebagai keadaan sekarang.
   c. Jika nilai keadaan baru tadi tidak lebih baik dari nilai rute sekarang $(Z_n > Z_c)$, maka tetapkan keadaan baru sebagai keadaan sekarang dengan probabilitas

$$p = e^{\frac{\Delta E}{T_i}} \quad (2.1)$$

$$\Delta E = nilai\ keadaan\ sekarang\ (Z_c) - nilai\ keadaan\ baru\ (Z_n) \quad (2.2)$$

Langkah ini biasanya dikerjakan dengan membangkitkan suatu bilangan random r pada range [0,1]. Jika $r \leq p$, maka perubahan keadaan baru menjadi keadaan sekarang diperbolehkan. Jika $r > p$, maka keadaan baru menjadi keadaan sekarang tidak diperbolehkan.

4. Perbaiki T sesuai dengan *annealing scheduling*.
5. Lakukan kembali langkah 3 dan langkah 4 untuk menentukan keadaan selanjutnya.
6. Hentikan iterasi sesuai dengan maksimum jumlah iterasi yang ditentukan (Samana et al, 2015).

2.10. Penelitian Terkait

Penelitian ini adalah sebuah pendekatan baru untuk menguji perangkat lunak pada tahap awal *(design)*. Sehingga mempermudah bagi penguji perangkat lunak pada proses selanjutnya. Penelitian ini menggunakan pengujian otomatis yaitu *test case*. Sehingga penelitian ini terfokus pada bagaimana cara membangkitkan *test case* dari model UML *Sequence diagram* menggunakan metode Algoritma Genetika untuk mencari *test case* terbaik dari beberapa *output test case*. Hasil penelitian menunjukkan metode ini memiliki kinerja yang baik. Sistem ATM digunakan sebagai studi kasus pada penelitian ini (Shanti et al 2012).

Penelitian ini yaitu *test case* pada proses pengujian perangkat lunak. Penelitian ini dilakukan menggunakan sebuah model pengujian dasar (*model based testing)* dari *test path* otomatis yang didapat sebelum atau dalam proses pembuatan perangkat lunak sedang berlangsung, setelah kode aplikasi telah tersedia, *test case* dapat dieksekusi sehingga dapat membantu untuk

memperbaiki kesalahan pada tahap awal pembuatan perangkat lunak. Model UML *Sequence diagram* dari Sistem Konsultasi Medis digunakan sebagai studi kasus dalam penelitian ini (Priya et al 2013).

Dalam penelitian ini, pendekatan eksternal digunakan untuk memodelkan masalah pengujian perangkat lunak. Pendekatan ini berguna untuk menghasilkan *test case* melalui UML *sequence diagram.* Penulis membuat *sequence diagram* dengan menggunakan property elemen materi n-dimensi yang diubah menjadi *flow graph*. Data yang diurai dihasilan melalui XML dari *sequence diagram* dan mengonversi dua kali lipat dari *flow graph.* Algoritma traversal dirancang dan diimplementasikan pada dua kali lipat *flow graph* untuk menghasilkan *test case* yang valid. *Test case* lebih lanjut diminimalkan yang memenuhi kriteria cukupan jalur. Seluruh pekerjaan diimplementasikan pada bahasa pemrograman Java yang berorientasi objek melalui studi kasus Aktifitas Pemberangkatan Pesawat (Rhmann, 2016).

Desain Unified Modeling Language (UML) memberikan informasi yang valid untuk proses pengembangan perangkat lunak. Pembuatan *test case* berbasis interaksi UML dapat digunakan untuk meningkatkan kualitas perangkat lunak. Makalah ini menyajikan metode untuk pembuatan *test case* dari diagraminteraksi UML di tingkat *cluster* yang membuat tiga proses. Pertama, diagram interaksi diubah menjadi tumpukan struktur data, kemudian tumpukan stimulus diminalkan menggunakan pengujian batas, dan akhirnya *test case* dibuat dari tumpukan yang diminimalkan. Studi kasus dalam makalah ini adalah Sistem Keamanan *Smart Home* (Mani & Prasanna, 2017).

Penelitian sebelumnya oleh (Ariyani, 2017) tentang pembangkit *test case* model UML *sequence diagram* menggunakan studi kasus Sistem Penilaian Pembelajaran. Pada penelitian tersebut, peneliti berupaya membangkitkan *test case* dengan pembuatan *sequence diagram*, *sequence dependency table* (SDT), *sequence dependency graph* (SDG), dan mendapatkan *test path* menggunakan metode *depth first search* (DFS). Metode DFS adalah pencarian dengan melakukan penulusuran terlebih dahulu pada suatu pohon atau *graph* sedalam-dalamnya. Setelah tidak bisa melanjutkan proses barulah algoritma DFS melakukan *backtrack* pada simpul-simpul tetangga (Kastogi, 2015). Hasil dari penelitian tersebut telah diperoleh Sistem Penialaian Pembelajaran menghasilkan sembilan *path* dari *input* sebanyak empat *sequence diagram.* Namun, kekurangan dari penelitian tersebut adalah terdapat angka 1 yang tercetak pada *generate path.*

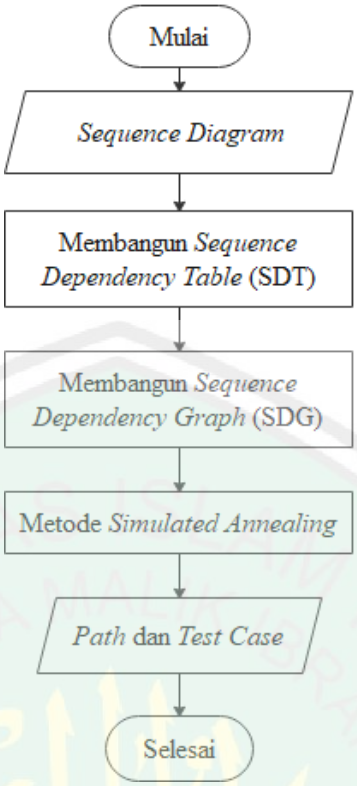
BAB III

PERANCANGAN DAN IMPLEMENTASI

Dalam bab ini dipaparkan tahap perancangan dan implementasi program komputer yang akan digunakan dalam penelitian. Program komputer ini akan dijadikan alat untuk menghasilkan *test case* dalam bentuk *path*. *Path* yang dihasilkan akan digunakan untuk mengetahui apakah algoritme simulated anneaing telah bekerja dengan baik atau tidak.

3.1. Perancangan Sistem

Dalam penelitian ini sistem yang akan dibangun adalah sistem yang mampu menghasilkan *output* berupa *test case* dalam bentuk *path*. Input sistem berupa *sequence diagram* dalam format XML. Diantara *input* dan *output* tersebut terdapat proses yang di dalamnya terdapat algoritma *simulated annealing* yang akan menngubah *sequence diagram* menjadi *path* secara otomatis.



Gambar 3. 1 Alur Sistem Pembangkit *Test Case* Otomatis

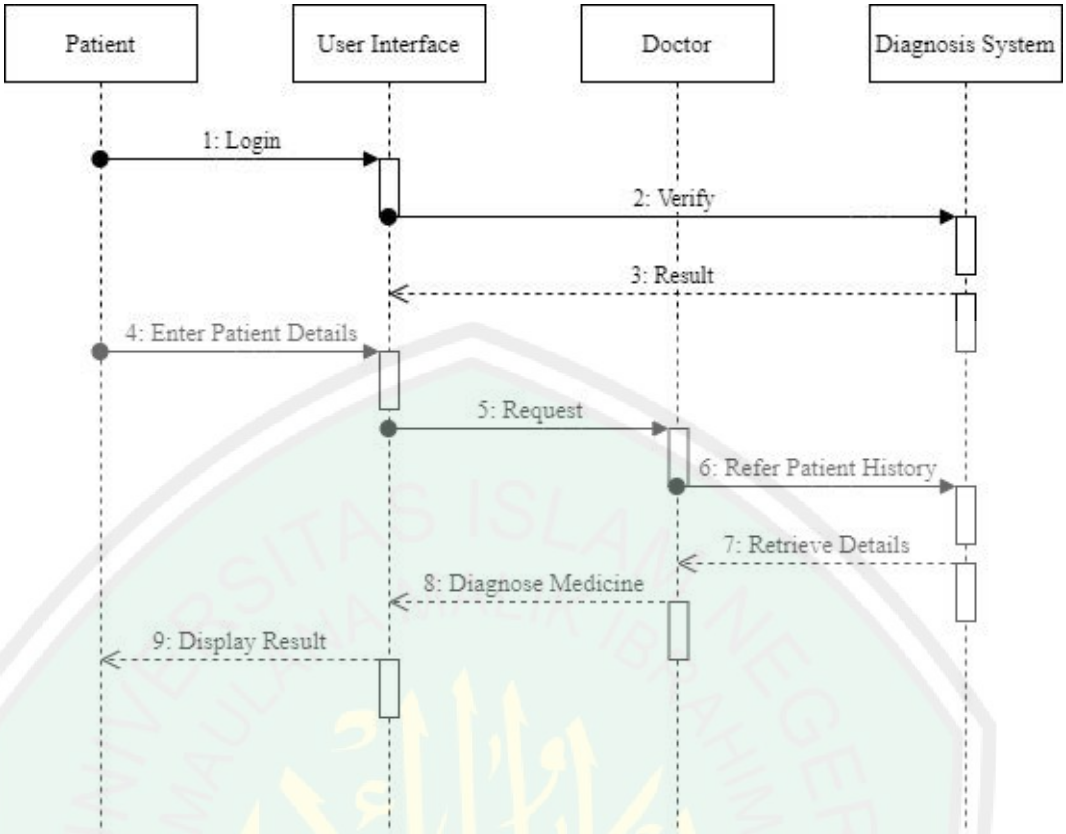
Gambar 3.1 adalah alur sistem pembangkit *test case* secara otomatis. Dalam sistem ini langkah pertama adalah membuat *sequence diagram* menggunalan perangkat lunak StarUML. Langkah kedua adalah mengubah *sequence diagram* tersebut menjadi *Sequence Dependency Table* (SDT). Langkah ketiga adalah mengubah *Sequence Dependency Table* (SDT) menjadi *Sequence Dependency Graph* (SDG). Langkah terakhir adalah melakukan *generate test case* dengan menggunakan algoritma *Simulated Annealing* (SA), yang akan menghasilkan *output* nya berupa *test path.* Penjelasan lebih detil dari berbagai proses tersebut akan dijelaskan pada subbab selanjutnya.

Sebagai ilustrasi bagaimana sistem bekerja, dalam bab ini akan dijelaskan secara detil menggunakan studi kasus Sistem Konsultasi Medis atau sistem konsultasi medis dari *paper* yang ditulis oleh Priya (2013). Layanan Sistem Konsultasi Medis ini memudahkan pasien untuk berkonsultasi dengan dokter

secara *online* tanpa harus bertatap muka. Dengan sistem ini, pasien dengan nyaman dan mudah dalam melakukan konsultasi dimanapun dengan menggunakan komputer. Pasien harus memiliki akun pribadi yang berupa *username* dan *password* (pasien adalah pengguna sudah terdaftar). Pasien yang sudah berhasil *login* dapat menampilkan seperti nama pasien, usia, jenis kelamin, tanggal terakhir konsultasi, kepada dokter mana yang pasien inginkan untuk berkonsultasi (wajib), gejala yang pasien miliki (wajib). Jika dokter yang diinginkan tersedia, maka dokter dapat melihat riwayat pasien sebelumnya dari sistem diagnosis dan dapat melihat gejala pada pasien yaitu sesuai keluhan yang dialami pasien. Jika rincian yang diberikan pasien cukup untuk mendiagnosis masalah, maka dokter dapat merekomendasikan pengobatan yang sesuai kepada pasien. Namun jika tidak, dokter dapat memberikan saran pasien untuk melakukan tes tambahan dan mengirimkan laporan hasil tes secara pribadi kepada dokter untuk konsultasi lebih lanjut.

3.2. Pembuatan *Sequence Diagram*

Tools untuk analisis dan desain sistem berorientasi objek memiliki banyak fitur yang digunakan untuk menggambarkan sistem dari berbagai sudut pandang, salah satunya adalah *sequence diagram*. Dalam penelitian ini pembuatan *sequence diagram* ini dibuat menggunakan perangkat lunak StarUML. *Sequence diagram* ini berbentuk grafis, dan dengan menggunakan fitur *export* pada StarUML dapat diperoleh *sequence diagram* dalam format xml.



Gambar 3. 2 *Sequence Diagram* untuk Sistem Konsultasi Medis (Priya, 2013)

Pada Gambar 3.2 adalah *sequence diagram* dari sistem Sistem Konsultasi Medis yang mana setiap angka adalah mewakili setiap pesan untuk dipetukarkan antara objek pada komunikasi yang diminta. Di dalam sistem urutan peristiwa yang terjadi digambarkan dengan garis vertical putus-putus. Berikut penjelasan dari *sequence diagram* dari Gambar 3.2:

a. *Login*: Untuk pasien *login* ke sistem dengan memberikan *username* dan *password* yang sesuai.
b. *Verify:* Memverifikasi *usename* dan *password* dimasukkan sesuai dengan memeriksa di dalam database.
c. *Result:* Memperlihatkan hasil untuk *user interface*. Jika *usename* dan *password* yang dihasilkan sudah sesuai, sistem memeriksa pengguna

dan memungkinkan pasien untuk melanjutkan lebih jauh. Jika tidak sesuai, akses dapat ditolak dan menunjukkan bahwa pengguna tidak cocok.

d. *Enter Patient Details:* Pasien setalah berhasil *login* untuk dapat memasukkan rincian data pribadi seperti nama, umur, jenis kelamin, tanggal berkonsultasi, berkonsultasi dengan dokter yang diinginkan (wajib), gejala pasien yang dimiliki (wajib).

e. *Request:* Sistem mendapatkan rincian data pasien menghubungkan ke dokter. Jika dokter tersedia, dokter dapat menanggapi pasien, atau pasien akan diberitahu dengan tidak tersedianya dokter.

f. *Refer Patient History:* Jika dokter yang diinginkan tersedia, maka dokter dapat melihat riwayat pasien sebelumnya dari sistem diagnosis dan dapat melihat gejala pada pasien.

g. *Retrieve Details:* Informasi pasien yang dapat diambil.

h. *Diagnose and Prescribe Medicine/Prescribe Additional Tests:* Setelah dokter merujuk keriwayat pasien sebelumnya dan gejala yang telah masuk sekarang, jika memadai, dapat mendiagnosis tingkat keparahan penyakit dan bisa meresepkan obat-obatan. Berbeda jika gejala tidak memadai untuk mendiagnosa, dokter dapat menyarankan untuk tes tambahan yang diambil.

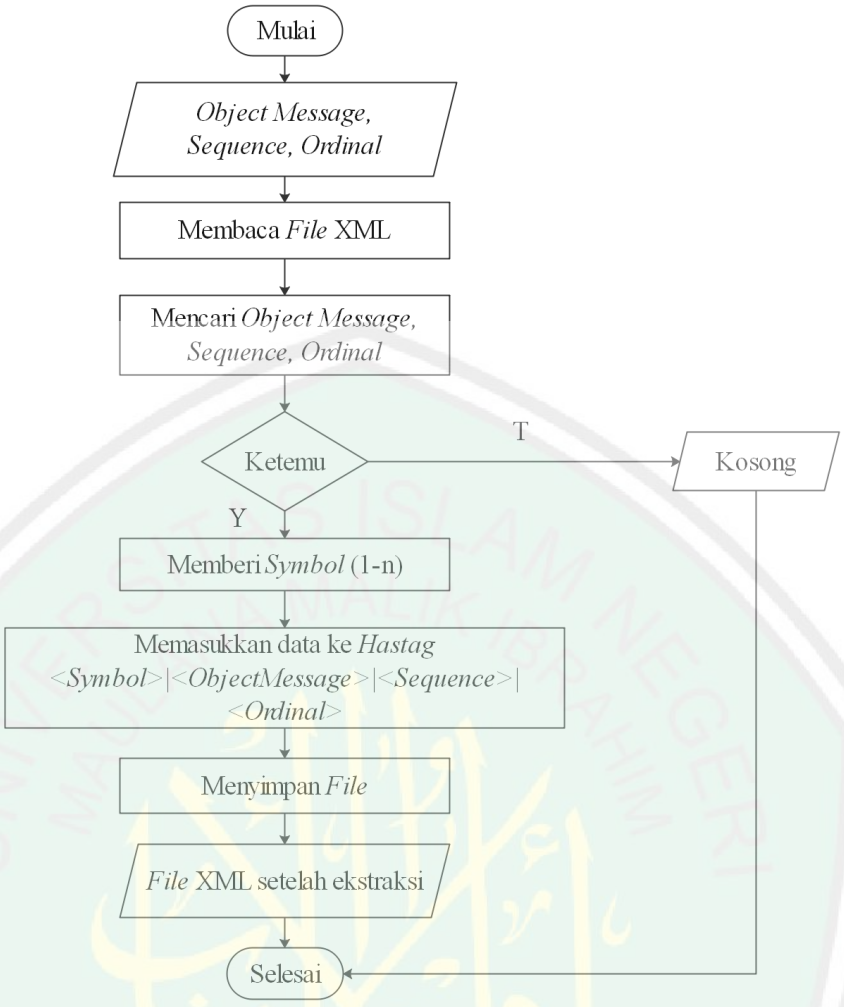
i. *Display Prescription/Suggest Additional Tests:* Resep dokter dapat dikirim melalui *user interface* untuk dapat menampilkan resep tersebut kepada pasien.

3.3. Membangun *Sequence Dependency Table* (SDT)

Langkah selanjutnya adalah membangun *Sequence Dependency Table* (SDT). Langkah ini dibagi menjadi dua tahap. Tahap pertama, data *sequence diagram* dari StarUML dalam format XML diekstrak untuk mengambil beberapa data yang diperlukan untuk membuat *Sequence Dependency Table* (SDT). Tahap kedua adalah membangun *Sequence Dependency Table* (SDT) itu sendiri.

3.3.1. Ekstraksi *File* XML StarUML

Data *sequence diagram* dari StarUML yang berekstensi XML diekstrak dengan mengambil beberapa data yang penting untuk membentuk SDT. Informasi terkait yang dibutuhkan seperti *Symbol*, *Activity Name*, *Sequence Number*, dan *Dependency*.

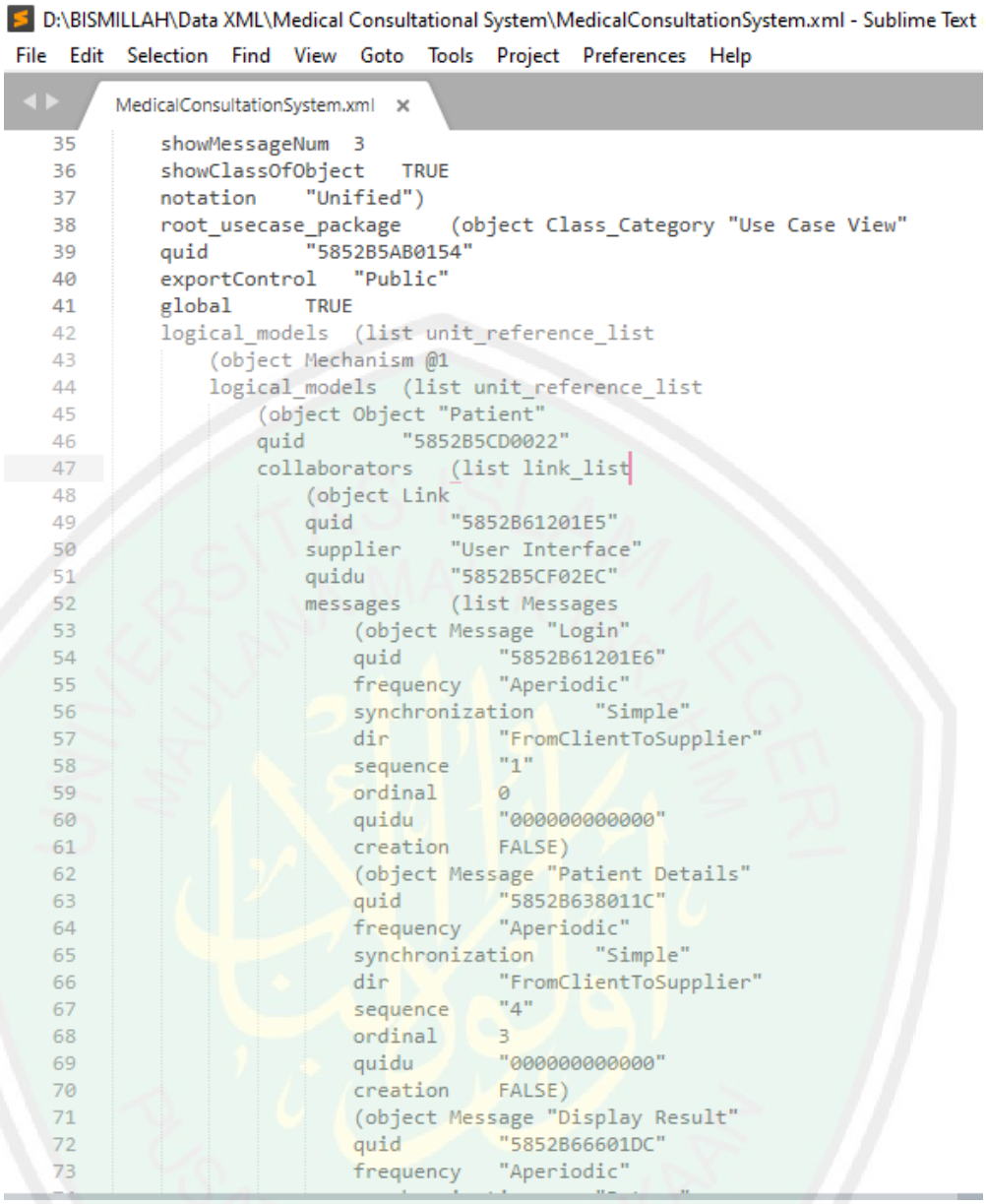


Gambar 3. 3 *Flowchart* ekstraksi *File* XML StarUML

Gambar 3.3 merupakan *Flowchart* dari proses pengambilan informasi penting dari *file* XML (*output* dari StarUML). Proses dimulai dengan mengambil file XML dari ekspor StarUML. Kemudian dicari nilai-nilai *variable* dari *Object Message*, *Sequence*, dan *Ordinal*. Jika *variable* didapatkan maka diberi *Symbol*, apabila tidak ditemukan maka *output* akan kosong. Apabila *variable* telah ditemukan, *variable-variable* tersebut disimpan dalam *Hastag* <*Symbol*>, <*ObjectMessage*>, <*Sequence*>, <*Ordinal*> yang kemudian disimpan dalam bentuk *file* dengan ekstensi *file*

XML. *File* ini merupakan dalam pembuat *Sequence Dependency Table* (SDT).

Dalam persoalan Sistem Konsultasi Medis model UML *sequence diagram* Sistem Konsultasi Medis disimpan dengan bentuk ekstensi *file* .xml (Gambar 3.4). Kemudian dilakukan pengolahan *String* menggunakan *Reguler Expressions* (REGEX) untuk mengambil data yang dibutuhkan. *Source code* pengolahan *String* dengan REGEX seperti pada Gambar 3.5. Hasil dari proses REGEX akan disimpan dalam ekstensi .xml.



Gambar 3. 4 Data uji *sequence diagram* dalam ekstensi .xml sebelum ekstraksi

Source code yang menerapkan REGEX guna mengambil data-data yang diperlukan dari *file* ekstensi .XML pada Gambar 3.4.

```
while ((strLine = br.readLine()) != null) {
    Pattern pattern = Pattern.compile("(?<=object Message).*");
    Pattern pattern1 = Pattern.compile("(?<=sequence).*");
    Pattern pattern2 = Pattern.compile("(?<=ordinal).*");
    Pattern pattern3 = Pattern.compile("(?<=synchronization).*");
    Matcher matcher = pattern.matcher(strLine);
    Matcher matcher1 = pattern1.matcher(strLine);
    Matcher matcher2 = pattern2.matcher(strLine);
    Matcher matcher3 = pattern3.matcher(strLine);
    boolean found = false; while (matcher.find()) {
        count++; System.out.println("banyak=" + count);
        System.out.println("<symbol id=" + count + ">\n<objectmessage>" + matcher.group().toString() + "</objectmessage>\n");
        jTextAreal.append("<symbol id='" + count + "'>\n<objectmessage>" + matcher.group().toString() + "</objectmessage>\n");
        //found = true;
    }
    while (matcher1.find()) {
        System.out.println("<sequence>" + matcher1.group().toString() + "</sequence>\n");
        jTextAreal.append("<sequence>" + matcher1.group().toString() + "</sequence>\n");
        //found = true;
    }
    while (matcher3.find()) {
        System.out.println("<synchronization>" + matcher3.group().toString() + "</synchronization>\n");
        jTextAreal.append("<synchronization>" + matcher3.group().toString() + "</synchronization>\n");
        //found = true;
    }
    while (matcher2.find()) {
        System.out.println("<ordinal>" + matcher2.group().toString() + "</ordinal>\n</symbol>\n");
        jTextAreal.append("<ordinal>" + matcher2.group().toString() + "</ordinal>\n</symbol>\n"); } }
jTextAreal.append("<symbol id='" + String.valueOf(count + 1) + "'>\n<objectmessage> End</objectmessage>");
jTextAreal.append("\n<synchronization>" + " \t" + "Simple</synchronization>");
jTextAreal.append("\n<sequence>    \t\"" + String.valueOf(count + 1) + "\"</sequence>");
jTextAreal.append("\n<ordinal>    \t"  + String.valueOf(count) + "" + "</ordinal>\n</symbol>");
jTextAreal.append("\n</root>");
```

Gambar 3. 5 *Source code* REGEX

```
int jumlah = 0;
int indeks = -1;
String kata = "symbol id";
String teks = jTextAreal.getText();

indeks = teks.indexOf(kata);
while (indeks >= 0) {
    ++jumlah;
    indeks += kata.length();
    indeks = teks.indexOf(kata, indeks);
}
System.out.println("Teks berisi kata: " + "\n" + "symbol id = " + jumlah);

m = teks.split("\n");
try {
    int starts = jTextAreal.getLineStartOffset((jumlah * 6) - 5);
    int ends = jTextAreal.getLineEndOffset(m.length - 8);
    jTextAreal.replaceRange("", starts, ends);

} catch (Exception e) {
    System.out.println("error hapusan :" + e.getMessage());
}
```

Gambar 3. 6 Menghilangkan kata yang tidak digunakan

Setelah proses pemotongan kata menggunakan REGEX berhasil, ternyata masih ada data yang tidak dibutuhkan seperti banyaknya kata yang mengandung "*symbol* id" ikut terambil. *Source code* pada Gambar 3.6 bekerja

untuk menghilangkan jumlah kata yang mengandung "*symbol* id" yang tidak diperlukan. Setelah semua proses dilakukan, data *file* yang dibutuhkan akan tersimpan menjadi ekstensi .xml yang kemudian *file* akan disimpan pada lokasi keinginan *user*. Berikut Gambar 3.7 Data uji *sequence diagram* setelah ekstraksi *file* XML.

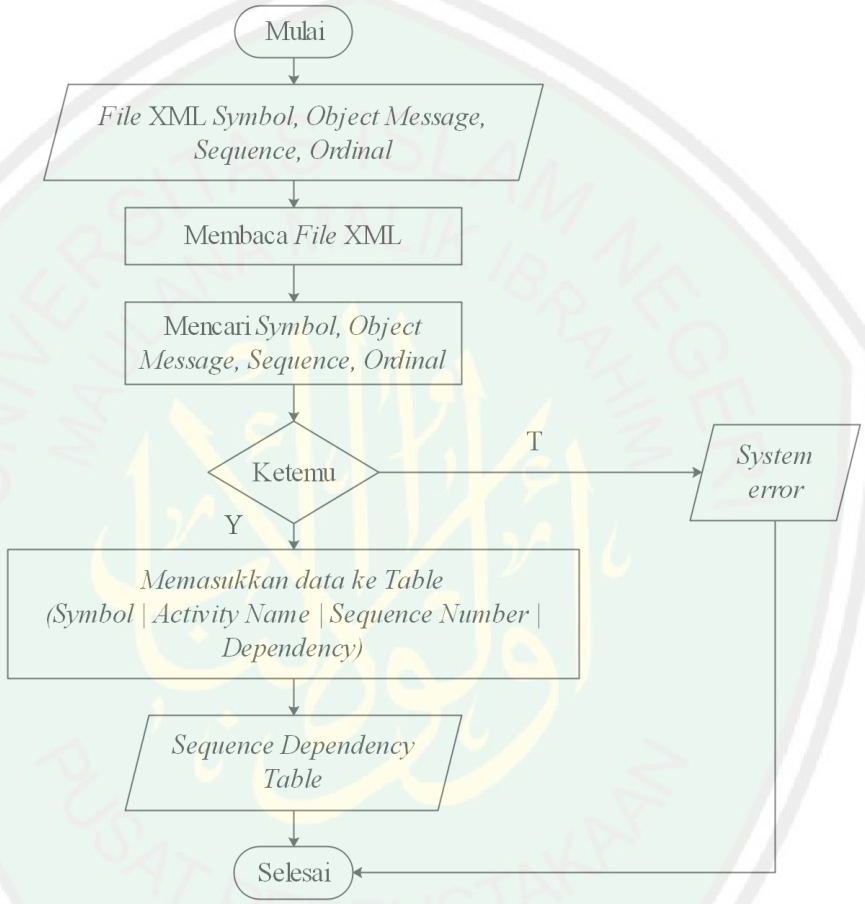
MedicalConsultationSystemOutput.xml

```
<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>
</symbol>
<symbol id='7'>
<objectmessage> "Diagnosis/Suggestion"</objectmessage>
```

Gambar 3. 7 Data uji setelah ekstraksi *file* XML

3.3.2. *Sequence Dependency Table* (SDT).

Setelah mendapatkan *file* hasil ekstraksi *sequence diagram* dalam format XML, langkah selanjutnya yakni membangun *Sequence Dependency Table* (SDT). Berikut Gambar 3.8 *flowchart* pembangunan *Sequence Dependency Table* (SDT).



Gambar 3. 8 *Flowchart* pembangunan *Sequence Dependency Table* (SDT)

Pada Gambar 3.8 merupakan *Flowchart* pembangunan *Sequence Dependency Table* (SDT). *Input* yang digunakan adalah *file* ekstraksi *sequence diagram* dalam format XML. Dalam membangung *Sequence Dependency Table* (SDT) data yang diambil adalah data yang menyimpan *variable Symbol, Object Message, Sequence* dan *Ordinal*. Setelah data

didapatkan kemudian membuat empat kolom yang nantinya membentuk sebuah tabel. Kolom tersebut terdiri dari kolom *Symbol* mewakili simbol yang diberikan, kolom *Activity Name* mewakikan *Object Message*, kolom *Sequence Number* mewakilkan *Sequence* dan kolom *Dependency* mewakilkan *Ordinal*. Gambar 3.9 adalah *source code* pembuatan *Sequence Dependency Table* (SDT).

```
 * @author reza
 */
class Relation {

    private String Symbol;
    private String ActivityName;
    private String SequenceNumber;
    private String Dependency;

    public Relation(String Symbol, String ActivityName, String SequenceNumber, String Dependency) {
        this.Symbol=Symbol;
        this.ActivityName = ActivityName;
        this.SequenceNumber = SequenceNumber;
        this.Dependency = Dependency;

    }
    @Override
    public String toString() {

       return "<" + Symbol + ", " + ActivityName + ", " +SequenceNumber + ", " + Dependency + ">";
  //     return Symbol +","+Dependency;
    }

    public String toString1(){
        return Dependency+" "+Symbol;
    }
    public String toString2(){
        return Dependency+" "+SequenceNumber;
    }
}
```

Gambar 3. 9 *Source code* ekstraksi *file* XML menjadi SDT

Sequence Dependency Table (SDT) yang diperoleh ketika *source code* pada gambar 3.9 *running* ditampilkan pada tabel 3.1.

Tabel 3. 1 *Sequence Dependency Table* (SDT) Sistem Konsultasi Medis

No	*Symbol*	*Activity Name*	*Sequence Number*	*Dependency*
1	1	*Login*	1	0
2	2	*Patient Details*	4	3
3	3	*Display Result*	9	8
4	4	*Verify*	2	1
5	5	*Result*	3	2

6	6	*Request*	5	4
7	7	*Diagnosis/Suggestion*	8	7
8	8	*Refer Patient History*	6	5
9	9	*Retrive Data*	7	6
10	10	*End*	10	3,4,5,9

Pada Tabel 3.1 menunjukkan *Sequence Dependency Table* (SDT) Sistem Konsultasi Medis yang terdiri dari empat kolom berisi informasi sebagai berikut:

a. *Symbol:* Angka yang diberikan untuk setiap kegiatan yang terlibat.
b. *Activity Name:* Menggambarkan kegiatan yang dilakukan.
c. *Sequence Number:* Memberi nomor urut untuk memberikan jejak aliran yang terjadi saat pesan ditukar antara entitas yang terlibat.
d. *Dependency: Symbol* (s) dari setiap urutan yang menunjukkan bahwa aktivitas saat ini memiliki ketergantungan atau hubungan pada *symbol* yang lain. Sebagai contoh, kegiatan "C" tergantung pada *activity* "A" yang artinya hasil yang dikembalikan tergantung pada *username* dan *password* yang valid oleh pasien.

3.4. Membangun *Sequence Dependency Graph* (SDG)

Sequence Dependency Graph (SDG) adalah grafik bearah yang mewakili ketergantungan beberapa objek satu sama lain yang bertujuan untuk mendapatkan urutan evaluasi atau tidak adanya urutan evaluasi yang menghormati dependensi yang diberikan dari grafik dependensi. Proses *Sequence Dependency Table* (SDT) menjadi *Sequence Dependency Graph* (SDG) yang harus dilakukan adalah mengambil nilai kolom *Dependency* dan *Sequence Number* yang ada di *Sequence Dependency Table* (SDT). Kolom tersebut memiliki fungsi kolom *Dependency* sebagai *node*1 (*node* awal) sedangkan kolom *Sequence Number* sebagai *node*2

(*node* tujuan) untuk dimasukkan dalam *Sequence Dependency Graph* (SDG). Pada pembuatan *Sequence Dependency Graph* (SDG) ini memiliki dua tahap yaitu tahap pertama mengambil *node* untuk dimasukkan ke dalam *graph* dan tahap kedua mengidentifikasi relasi antar dua *node* yang ada dalam satu *line*. Pada tahap pertama *source code* bekerja memisahkan *node* untuk dimasukkan ke dalam *node*1 dan *node*2. Gambar 3.10 merupakan *source code* untuk mengambil *node* untuk dimasukkan ke dalam *graph.*

```
Graph graph = new Graph();
String fileSDT = jTextAreal.getText();
String[] s = jTextAreal.getText().split("\n");
String st = "";
for (int i = 1; i < s.length; i++) {
    st = s[i];
    String[] arrayTeksSplit = st.split("\t");
    for (int j = 0; j < arrayTeksSplit.length; j++) {
        String datal = arrayTeksSplit[1];
        String data2 = arrayTeksSplit[2];
        String[] datala = datal.split(" ");
        String[] data2b = data2.split("\"");
        for (int a = 0; a < datala.length; a++) {
            for (int b = 0; b < data2b.length; b++) {
                String datalaa = datala[0];
                String data2bb = data2b[1];

                if (datalaa.length() != 1) {
                    String[] datalaaa = datala[0].split(",");
                    for (int c = 0; c < datalaaa.length; c++) {
                        String datalmod = datalaaa[c];
                        graph.addEdge(datalmod, data2bb);
                    }
                } else {
                    graph.addEdge(datalaa, data2bb);
                }
                break;
            }}
        break;
    }
}
```

Gambar 3. 10 *Source code* untuk memasukkan node ke dalam *graph*

Setelah tahap memasukkan *node* ke dalam *graph* berhasil, tahap selanjutnya yakni membangun relasi antar *node*. Relasi antar *node* disebut juga dengan *adjacency node*. Pada tahap ini berfungsi untuk menunjukkan kedekatan antar *node* dalam *graph* yang kemudian disimpan kedalam *Hashmap*. Berikut Gambar 3.11 merupakan *source code* membangun relasi antar *node* dan hasil antar relasi *node* bentuk *Graph* dalam *Hashmap* pada Tabel 3.3:

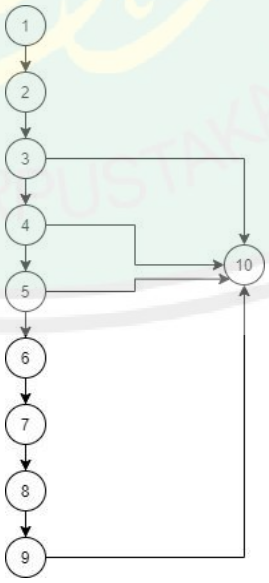
```
private Map<String, LinkedHashSet<String>> map = new HashMap();

public void addEdge(String node1, String node2) {
    LinkedHashSet<String> adjacent = map.get(node1);
    if(adjacent==null) {
        adjacent = new LinkedHashSet();
        map.put(node1, adjacent);
    }
    adjacent.add(node2);
}

public void addTwoWayVertex(String node1, String node2) {
    addEdge(node1, node2);
    addEdge(node2, node1);
}

public boolean isConnected(String node1, String node2) {
    Set adjacent = map.get(node1);
    if(adjacent==null) {
        return false;
    }
    return adjacent.contains(node2);
}
public LinkedList<String> adjacentNodes(String last) {
    LinkedHashSet<String> adjacent = map.get(last);
    if(adjacent==null) {
        return new LinkedList();
    }
    return new LinkedList<String>(adjacent);
}
```

Gambar 3. 11 *Source code* relasi antar *node*

Tabel 3. 2 Hasil relasi *node* bentuk *Graph* dalam *Hashmap*

Key	***Value***
3	[4, 10]
2	[3]
1	[2]
7	[8]
6	[7]
5	[6,10]
4	[5,10]
9	[10]
8	[9]

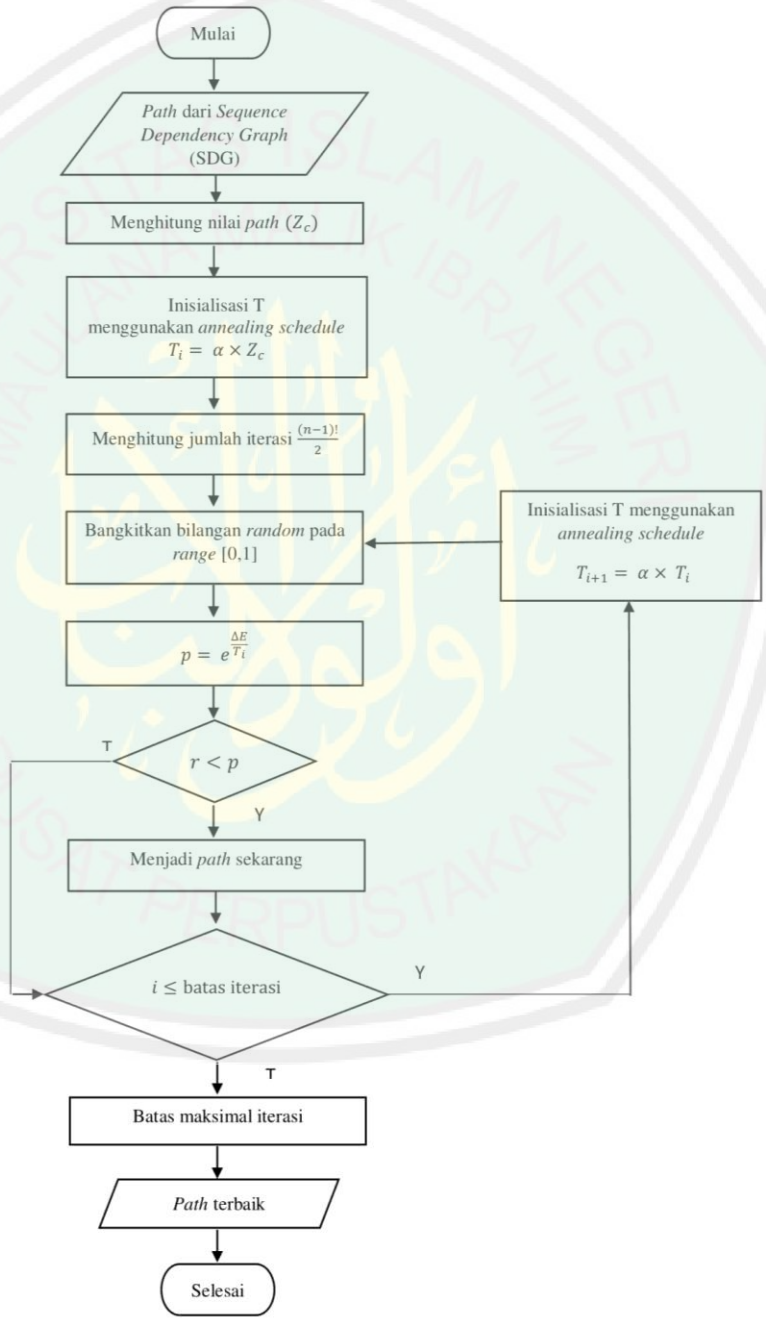
Penjelasan untuk Tabel 3.3 yaitu *Graph* yang disimpan dalam *Hashmap*. *Key* (*node*1) memiliki relasi dengan *Value* (*node*2). *Key* "2" terhubung dengan *Value* "3", *Key* "3" memiliki hubungan ganda pada *Value* "4" dan"10", begitu seterusnya. Untuk bentuk representasi *graph* nya akan ditunjukkan pada Gambar 3.12 *Sequence Dependency Graph* (SDG) dari Sistem Konsultasi Medis.



Gambar 3. 12 *Sequence Dependency Graph* (SDG) Sistem Konsultasi Medis.

3.5. Metode *Simulated Annealing* (SA)

Metode *Simulated Annealing* (SA) diterapkan pada proses pencarian *node-node graph* yang saling terhubung. Untuk membuat beberapa kemungkinan jalur yang dilalui dalam kasus pengujian (*Test Path*). Berikut Gambar 3.6 *Flowchart* metode *simulated annealing* yang akan diterapkan:



Gambar 3. 13 *Flowchart* Metode *Simulated Annealing*

Langkah-langkah yang harus diperhatikan dalam pelaksanaan proses SA yaitu:

1. Menentukan *path* awal secara *random* dan dihitung nilai *path. Path* ini didapatkan dari *Sequence Dependency Graph* (SDG).

   Path = 1 – 3 – 2 – 4 – 5 – 6 – 7 – 8 – 9 – 10

   Hitung Z_c dengan cara jumlah perjalanan alur *path*. Bila *path* melewati alur yang salah maka diberi nilai 0 dan apabila benar diberi nilai 1.

   1 ⟶ 3 = 0

   3 ⟶ 2 = 0

   2 ⟶ 4 = 0

   4 ⟶ 5 = 1

   5 ⟶ 6 = 1

   6 ⟶ 7 = 1

   7 ⟶ 8 = 1

   8 ⟶ 9 = 1

   9 ⟶ 10 = 1

   $$Z_c = 0 + 0 + 0 + 1 + 1 + 1 + 1 + 1 + 1 = 6$$

2. Inisialisasi *T* sesuai dengan *annealing schedule.* Tentukan *T* awal yaitu $T_i = \alpha \times Z_c$ (3.1) dengan *i* = 1, α = 0,95 dan Z_c adalah jumlah nilai *path.*

   T_1 = 0,95 x 6 = 5,7

3. Kemudian dilakukan iterasi dengan maksimum jumlah iterasi yang dilakukan adalah $\frac{(n-1)!}{2}$

   $\frac{(10-1)!}{2} = \frac{9!}{2} = 181440$

   Iterasi 1:

Langkah yang dilakukan yaitu membangkitkan suatu bilangan *random* untuk menentukan 3, 2, 4, 5, 6, 7, 8, dan 9 pada *range* [0,1]:

0,00 – 0,12 3

0,13 – 0,24 2

0,25 – 0,36 4

0,37 – 0,48 5

0,49 – 0,60 6

0,61 – 0,72 7

0,73 – 0,84 8

0,85 – 0,96 9

Bilangan *random* yang dibangkitkan adalah *r* = 0,00125 maka terletak di 3. Selanjutnya membangkitkan lagi suatu bilangan *random* pada untuk menentukan 2, 4, 5, 6, 7, 8, dan 9 *range* [0,1]:

0,00 – 0,14 2

0,15 – 0,28 4

0,29 – 0,42 5

0,43 – 0,54 6

0,55 – 0,68 7

0,69 – 0,82 8

0,83 – 0,96 9

Bilangan random yang dibangkitkan adalah *r* = 0,00125 maka terletak pada 2. langkah selanjutnya dengan menukar antara 3 dan 2 sehingga diperoleh *path*:

Path = 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10

1 ⟶ 2 = 1

2 ⟶ 3 = 1

3 ⟶ 4 = 1

4 ⟶ 5 = 1

5 ⟶ 6 = 1

6 ⟶ 7 = 1

7 ⟶ 8 = 1

8 ⟶ 9 = 1

9 ⟶ 10 = 1

$$Z_n = 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 9$$

Jika Z_n > Z_c maka tetapkan *path* baru sebagai *path* sekarang dengan probabilitas sesuai pada persamaan 2.1 dan 2.2:

$$p = e^{\frac{\Delta E}{T_i}}$$

$\Delta E = 6 - 9 = -3$

$p = e^{\frac{-3}{0,95}} = 0,04251$

karena $r < p$, maka *path* baru dapat ditetapkan sebagai *path* sekarang.

4. Inisialisasi $T_{i+1} = \alpha \times T_i$ sesuai dengan *annealing schedule* dengan

   $T_2 = 0,95 \times T_1$

   $T_2 = 0,95 \times 5,7 = 5,415$

   Untuk perhitungan iterasi 2 sampai iterasi ke 181440 juga menggunakan perhitungan yang sama seperti iterasi 1 sesuai dengan *path* yang dilalui.

Setelah menghitung secara manual, kemudian perhitungan tersebut akan diterapkan kedalam bentuk *source code* untuk membangun aplikasi. Berikut Gambar 3.13 dan Gambar 3.14 *Souece code Simulated Annealing*:

```
public void SimulatedAnnealing() {

    //Set initial temp
    double temp = 100000;

    //Cooling rate
    double coolingRate = 0.95;

    //create random intial solution
    Path currentSolution = new Path();
    currentSolution.generateIndividual();

      System.out.println("Total distance of initial solution: " + currentSolution.getTotalDistance());
      System.out.println("Random Path: " + currentSolution);
    // We would like to keep track if the best solution
    // Assume best solution is the cu55urrent solution
    Path best = new Path(currentSolution.getTour());

    ArrayList<String> arrayListUrut = new ArrayList<>();

    String tempe2 = "";
    // Loop until system has cooled
    while (temp > 1) {
        // Create new neighbour tour
        Path newSolution = new Path(currentSolution.getTour());

        // Get random positions in the tour
        int tourPos1 = Utility.randomInt(0, newSolution.tourSize());
        int tourPos2 = Utility.randomInt(0, newSolution.tourSize());
```

Gambar 3. 14 *Source Code Simulated Annealing* (1)

```
    //to make sure that tourPos1 and tourPos2 are different
        while(tourPos1 == tourPos2) {tourPos2 = Utility.randomInt(0 , newSolution.tourSize());}

    // Get the cities at selected positions in the tour
    Bismillah.Node positionSwap1 = newSolution.getPosition(tourPos1);
    Bismillah.Node positionSwap2 = newSolution.getPosition(tourPos2);

    // Swap them
    newSolution.setPosition(tourPos2, positionSwap1);
    newSolution.setPosition(tourPos1, positionSwap2);

    // Get energy of solutions
    int currentDistance   = currentSolution.getTotalDistance();
    int neighbourDistance = newSolution.getTotalDistance();

    // Decide if we should accept the neighbour
    double rand = Utility.randomDouble();
    if (Utility.acceptanceProbability(currentDistance, neighbourDistance, temp) > rand) {
        currentSolution = new Path(newSolution.getTour());
    }

    // Keep track of the best solution found
    if (currentSolution.getTotalDistance() < best.getTotalDistance()) {
        best = new Path(currentSolution.getTour());
      System.out.println("Temperature = " + temp + ", Path: " + best);
    TextAreaGeneratePath.append(best + "\n");
        System.out.println(best);
    }
    // Cool system
    temp *= 1 - coolingRate;
```

Gambar 3. 15 *Source Code Simulated Annealing* (2)

3.6. *Path*

Dari hasil perhitungan menggunakan metode *Simulated Annealing*, sehingga dari *Sequence Dependency Graph* Sistem Konsultasi Medis dapat menghasilkan *path* seperti berikut:

Path : 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10

3.7. *Test Case*

Test Case yang dihasilkan dari *Sistem Konsultasi Medis* setelah melalui beberapa proses menjadi:

Tabel 3. 3 *Test Case*

No	Hasil	
	Testpath	***Test case***
1	1-2-3-6-4-5-7-8-9-10	*Login, Verify, Result, Enter Patient Details, Request, Refer Patient History, Retrieve Details, Diagnosis/suggestion, Display Result, End.*

BAB IV

UJI COBA DAN PEMBAHASAN

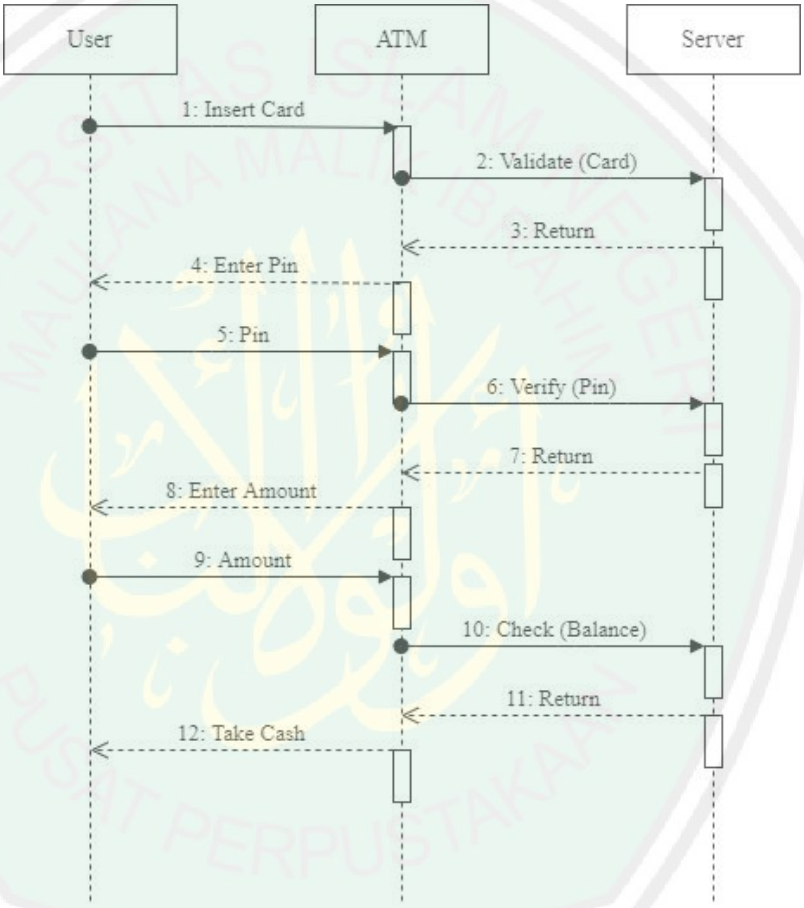
Bila pada bab III telah dilakukan desain dan implementasi aplikasi pembangkit test case, maka pada bab ini akan dilaporkan uji coba aplikasi yang telah dibangun tersebut. *Output* dari aplikasi akan menjadi bahan pembahasan untuk mendapatkan gambaran apakah algoritma *simulated annealing* yang digunakan telah bekerja dengan baik dalam membangkitkan test case.

4.1. Uji Coba

4.1.1. Data Uji Coba

Input dari aplikasi pembangkit test case adalah UML *sequence diagram* yang telah diubah dalam format XML. Terdapat empat data uji coba yang akan digunakan dalam pengujian aplikasi yakni *sequence diagram* Sistem ATM dari Shanti et al (2012), *sequence diagram* Sistem Konsultasi Medis dari Priya et al (2013), *sequence diagram* Aktifitas Pemberangkatan Pesawat dari Rhmann (2016), *sequence diagram* Sistem Keamanan *Smart Home* dari Mani & Prasanna (2017) dan *sequence diagram* dari Sistem Penilaian Pembelajaran. Uji coba pertama yakni *sequence diagram* Sistem ATM dari Shanti et al (2012) yang merupakan sarana mudah bagi *customer* Bank dalam pengambilan uang tanpa harus melalui layanan kantor Bank. *Customer* dapat mengambil uang dalam mesin yang disebut ATM. Langkah pertama yang dilakukan yakni memasukkan kartu kedalam mesin kemudian divalidasi. Kartu ATM ini didapat dari pembuatan rekening saat di Bank. Langkah kedua masukkan pin kemudian pin akan diverifikasi, jika pin benar maka akan masuk ke proses

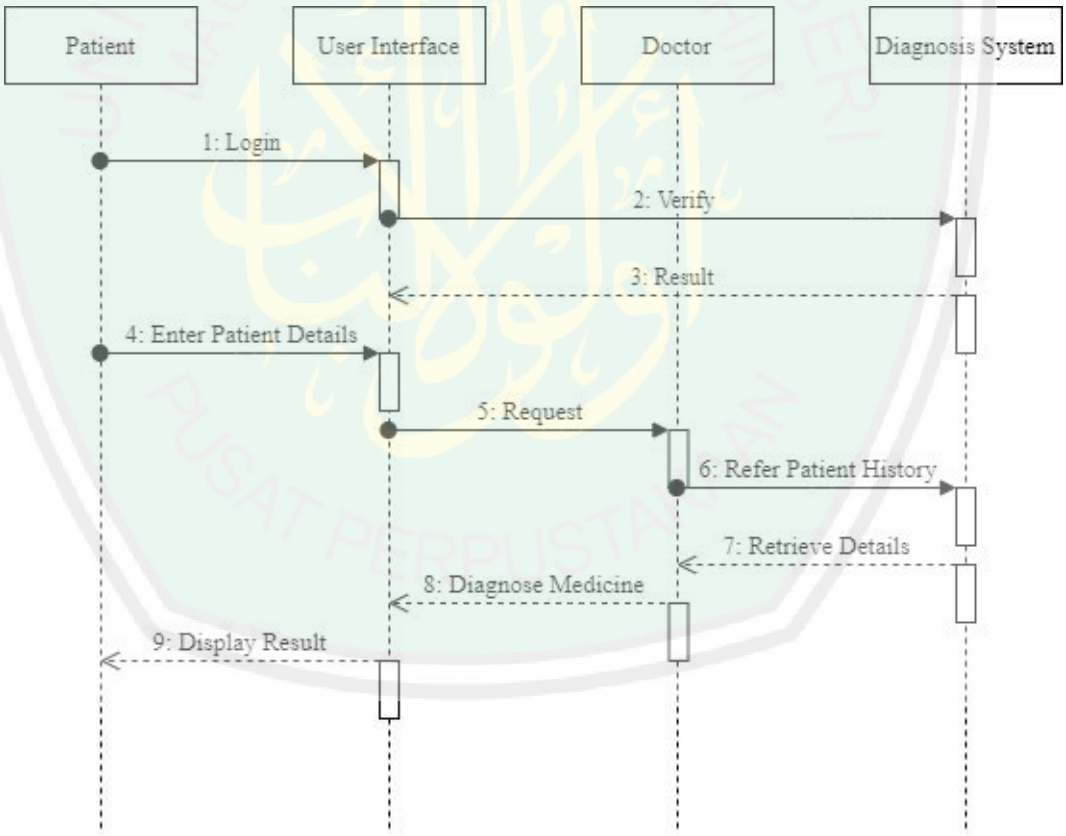
selanjutnya. Langkah ketiga masukkan nominal uang yang akan diambil, jika nominal yang dimasukkan sesuai dengan keinginan dan benar maka akan berlanjut ke proses selanjutnya yakni mengecek saldo. Setelah semua proses sesuai dan benar maka uang dapat diambil dari mesin ATM tersebut. Berikut gambar 4.1 yang menunjukkan *sequence diagram* Sistem ATM dari Shanti et al (2012).



Gambar 4. 1 *Sequence Diagram* Sistem ATM dari Shanti et al (2012)

Uji coba yang kedua yakni *sequence diagram* Sistem Konsultasi Medis dari Priya et al (2013) yang merupakan layanan konsultasi komputerisasi yang dapat menjadi system informasi bimbingan antara pasien dengan dokter. Layanan ini memudahkan pasien untuk berkonsultasi dengan dokter secara *online* tanpa harus bertatap muka. Pasien harus memiliki akun pribadi yang

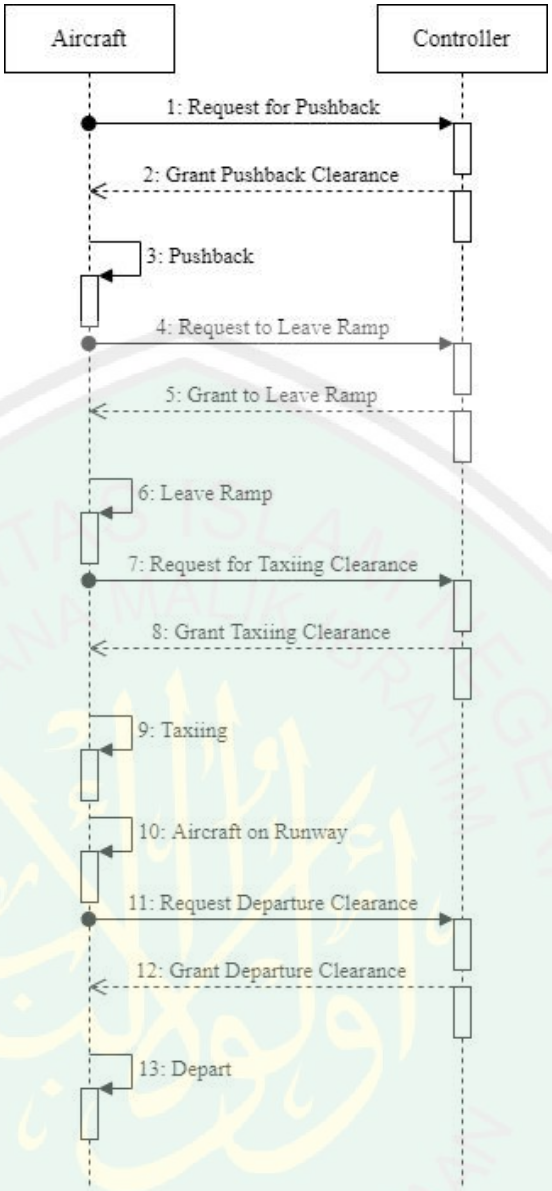
harus memberikan *username* dan *password* (pasien adalah pengguna yang telah terdaftar) yang digunakan untuk *login.* Setelah pasien berhasil *login,* pasien akan masuk ke halaman *Patient Details.* Pasien dapat melakukan permintaan terhadap dokter di halaman *Request* dan dokter dapat melihat riwayat dari pasien di halaman *Refer Patient History*. Pada halaman *Retrivie Details* dokter dapat melihat informasi pasien yang diambil agar dapat mendiagnosis dan merekomendasikan pengobatan terhadap pasien. Pada halaman *Display Prescription* ini, dokter dapat mengirimkan resep ke *User Interface* agar dilihat pasien. Berikut gambar 4.2 yang merupakan *sequence diagram* Sistem Konsultasi Medis dari Priya et al (2013).



Gambar 4. 2 *Sequence Diagram* Sistem Konsultasi Medis dari Shanti et al (2012)

Uji coba ketiga yakni *sequence diagram* Aktifitas Pemberangkatan Pesawat dari Rhmann (2016) yang merupakan kegiatan keberangkatan

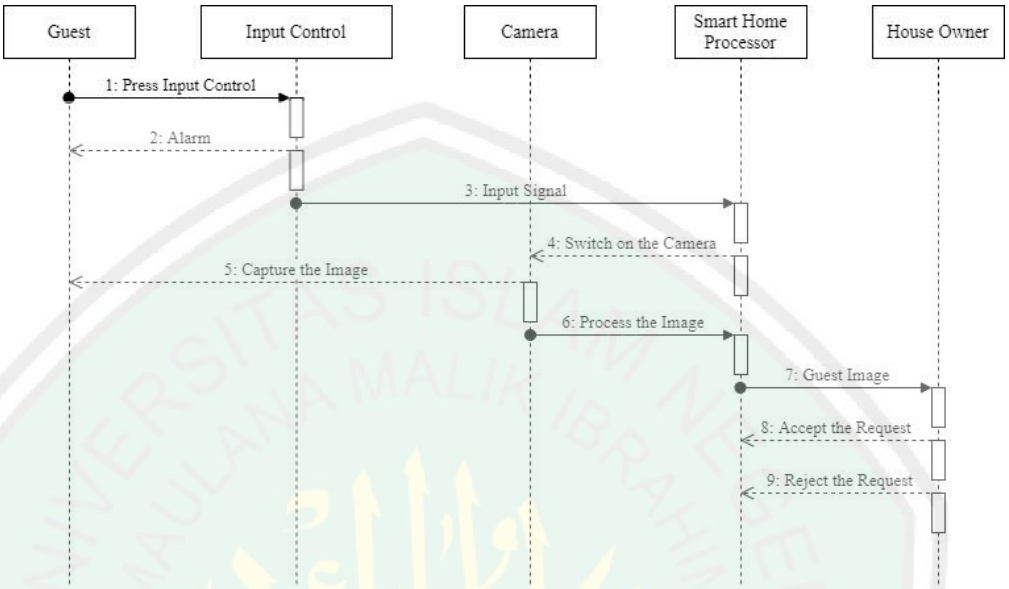
pesawat. Dalam diagram ini ada dua objek yaitu *Aircraft* dan *Controller*. Pesan dipertukarkan antara dua objek ini. Pesan pertama *Aircraft* mengirim pesan permintaan untuk izin *pushback*. *Pushback* adalah prosedur mendorong mundur pesawat dari gerbang bandara. Dalam prosedur *pushback*, pesawat bergerak mundur dengan kekuatan eksternal. Pesan kedua dari *Aircraft* ke *Controller* adalah permintaan untuk *Leave Ramp*. Jika *Controller* memberikan *Leave* maka pesawat meninggalkan *Ramp*. *Ramp* adalah area dimana aktivitas perawatan pesawat dilakukan. Pesan ketiga adalah permintaan pesawat untuk *Taxiing Clearence*. *Taxiing* adalah proses pesawat dapat bergerak dengan kekuatannya sendiri. Jika izin *taxiing* diberikan maka pesawat bergerak dilandasan. Pesan terakhir yaitu untuk izin keberangkatan (*Departure Clearance*) dari *Controller*. Jika *Controller* memberikan izin keberangkatan, maka pesawat berangkat.



Gambar 4. 3 *Sequence Diagram* Aktifitas Pemberangkatan Pesawat dari Rhmann (2016)

Uji coba yang keempat *sequence diagram* Sistem Keamanan *Smart Home* dari Mani & Prasanna (2017) yang merupakan prinsip kerja sistem keamanan rumah pintar. Sistem ini adalah kombinasi dari *input control*, *processor* S3C210 dengan platform tertanam, prosesor video, jaringan komunikasi dan *output smart mobile*. Ketika *input control* ditekan, sistem segera mengirimkan sinyal peringatan ke prosesor. Kamera yang dikendalikan prosesor, dipasang diatas *input control*, menangkap gambar yang kemudian

ditransfer ke platform tertanam untuk memproses gambar dan menyimpannya sebagai format *file* gambar. Kemudian *file* gambar masukan ditransfer melalui jaringan komunikasi ke *smart mobile* yang ditargetkan.

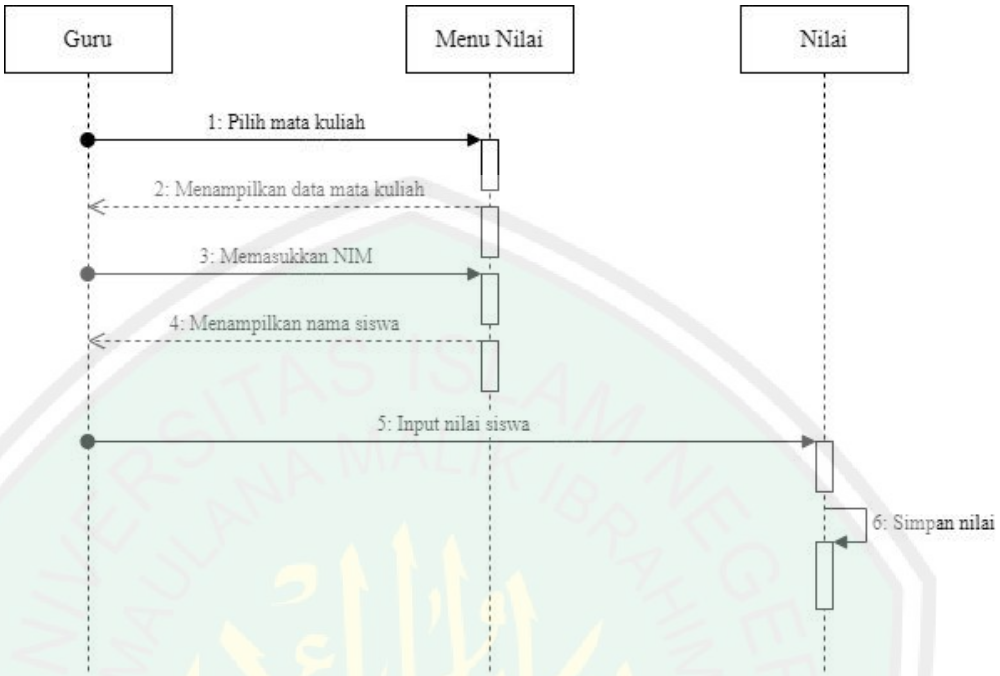


Gambar 4. 4 *Sequence Diagram* Sistem Keamanan *Smart Home* dari Mani & Prasanna (2017)

Uji coba kelima yaitu Sistem Penilaian Pembelajaran yang juga digunakan sebagai studi kasus dalam penelitian ini. Penilaian pembelajaran merupakan aplikasi yang membantu dosen atau guru untuk memberikan penilaian kepada mahasiswanya dalam evaluasi proses belajar mengajar. Pada aplikasi ini dosen dapat memasukkan nilai tugas, nilai ujian tengah semester, dan nilai ujian akhir semester. Nilai-nilai yang dimasukkan perlu diolah berdasarkan dari bobot masing-masing komponen untuk mendapatkan nilai akhir yang dikonversi menjadi nilai abjad A, B+, B, C+, C, D, dan E.

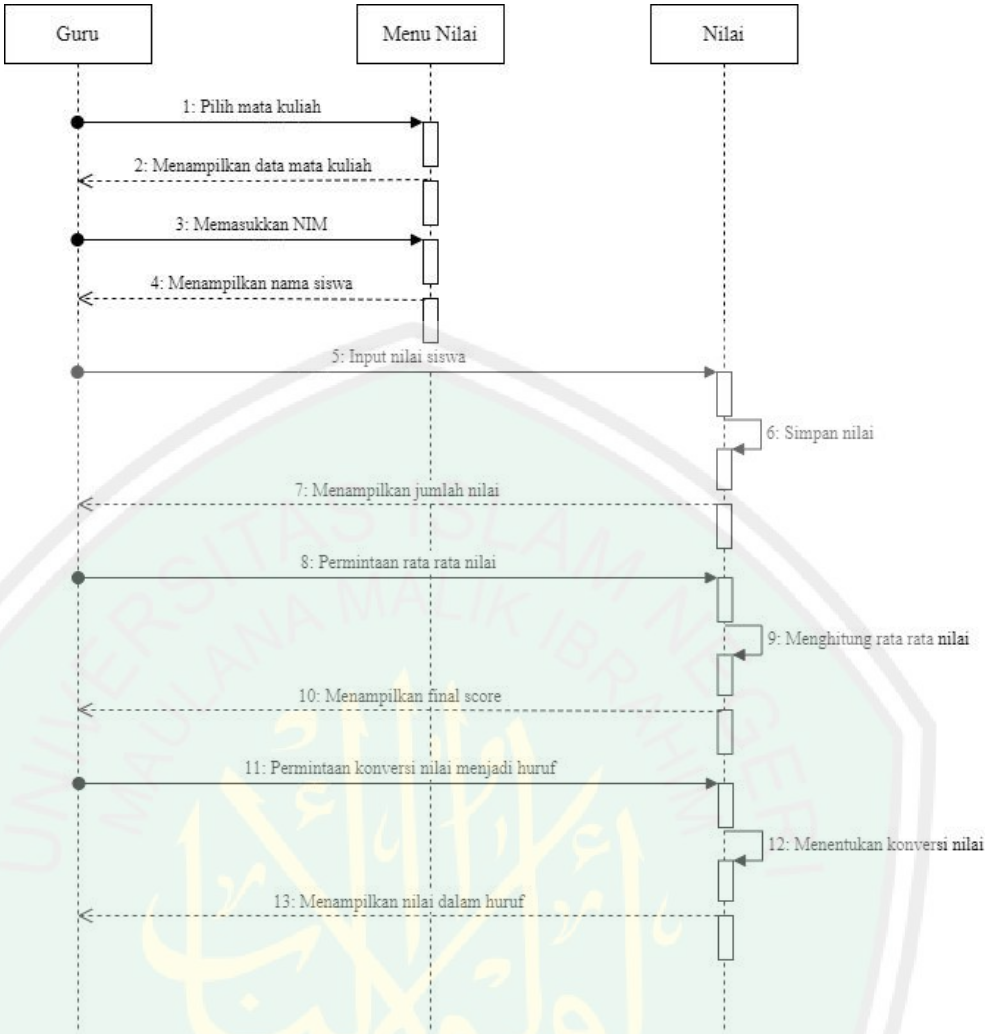
Sistem Penilaian Pembelajaran ini memiliki tiga *sequence diagram. Sequence diagram* pertama dari Sistem Penilaian Pembelajaran yaitu *sequence diagram* Memasukkan Nilai. Pada *sequence diagram* ini berfungsi

untuk memasukkan nilai siswa ke sistem. Berikut Gambar 4.4 *sequence diagram* Memasukkan Nilai:



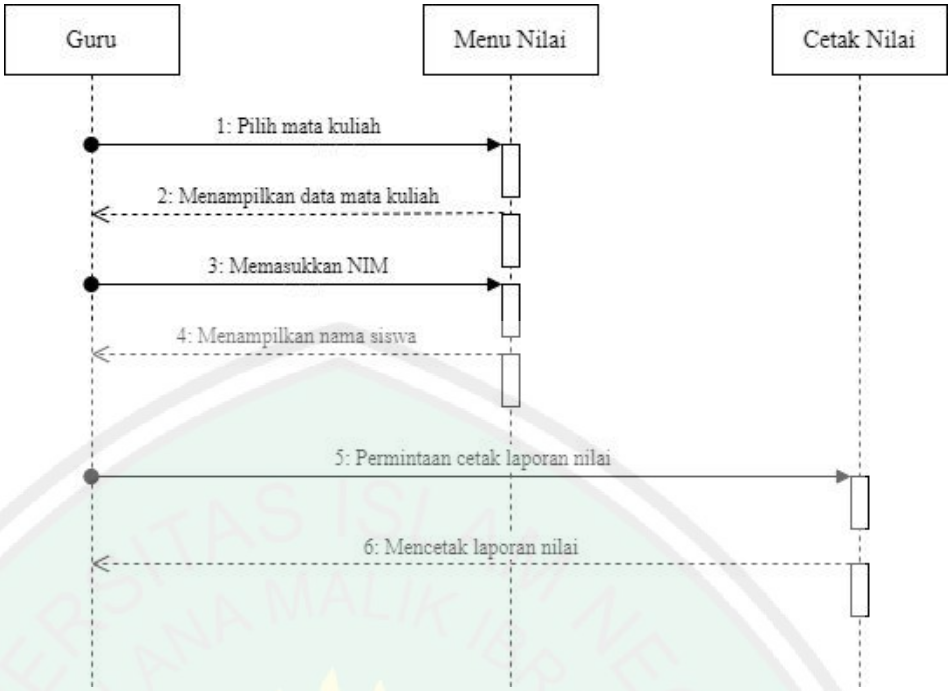
Gambar 4. 5 *Sequence diagram* Memasukkan Nilai

Sequence diagram kedua dari Sistem Penilaian Pembelajaran yaitu *sequence diagram* Perhitungan Nilai. Pada *sequence diagram* ini berfungsi untuk menghitung jumlah nilai sehingga memperoleh *final score* yang kemudian akan dikonversi dalam bentuk huruf. Berikut Gambar 4.5 *sequence diagram* Perhitungan Nilai:



Gambar 4. 6 *Sequence Diagram* Perhitungan Nilai

Sequence diagram yang ketiga dari Sistem Penilaian Pembelajaran yaitu *sequence diagram* Laporan. Pada *sequence diagram* laporan ini, berfungsi untuk menghasilkan laporan yang dibutuhkan. Berikut Gambar 4.6 *sequence diagram* Laporan:



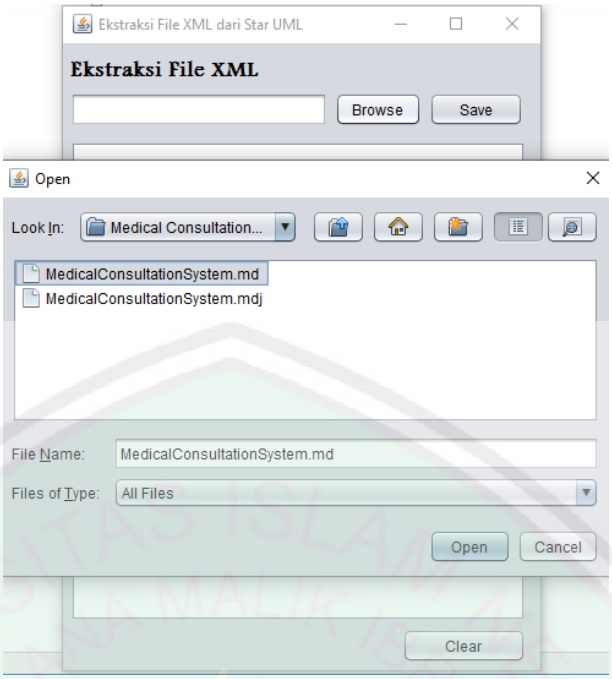
Gambar 4. 7 *Sequence diagram* Laporan

4.1.2. Pengujian Aplikasi

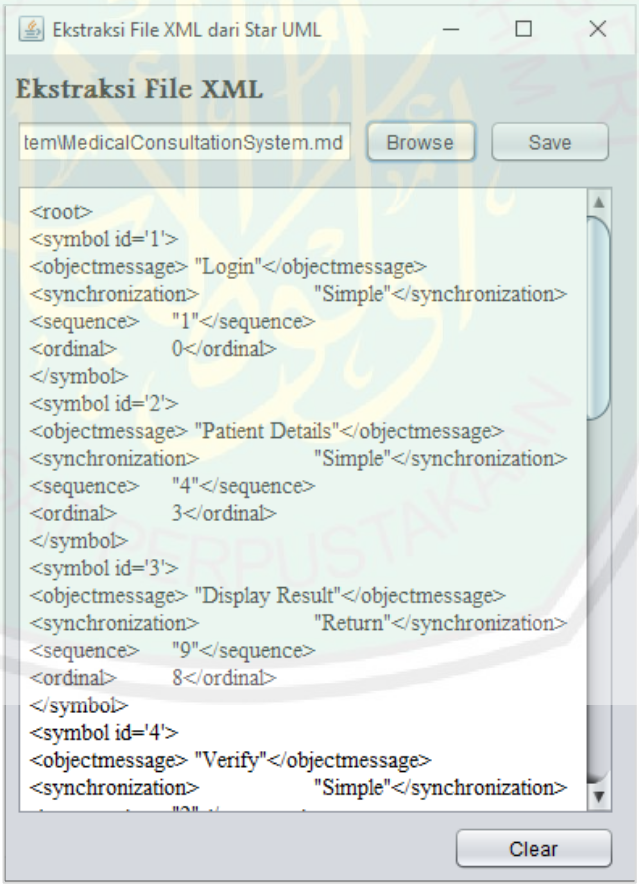
Telah dijelaskan di bagian sebelumnya bahwa input dari sistem yang dibangun adalah format XML dari *sequence diagram*. Dalam penelitian ini sequence diagram dalam format XML diperoleh dengan menggunakan software StarUML. Aplikasi pembangkit *test case* otomatis ini terdiri dari dua komponen yaitu komponen yang digunakan untuk melakukan ekstraksi XML dan komponen yang digunakan untuk pembangkit *test path*. Komponen ekstraksi XML digunakan untuk mengambil data-data penting yang dibutuhkan dari *sequence diagram* format XML dan akan disimpan dengan ekstensi XML. Berikut Gambar 4.7 Komponen ekstraksi XML:

Gambar 4. 8 Komponen ekstraksi XML

Pada komponen ekstraksi XML terdapat dua *button* yaitu *browser* dan *save*. *Button browser* berfungsi untuk mencari *file sequence diagram* yang akan dikonversi. *Button save* berfungsi untuk menyimpan hasil *file* XML yang telah dikonversi. Berikut Gambar 4.8 *Browsing file XML* dan Gambar 4.9 Hasil ekstraksi *file* XML:

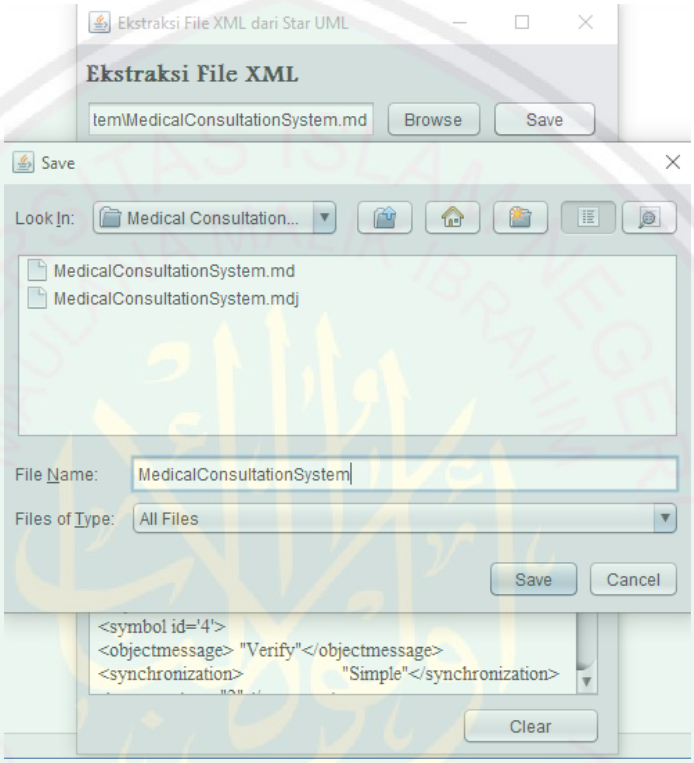


Gambar 4. 9 *Browsing file* XML



Gambar 4. 10 Hasil ekstraksi *file* XML

Hasil dari ekstraksi XML dapat disimpan dengan cara mengklik *button save*. Setelah *button save* diklik, kemudian akan muncul tampilan untuk menyimpan *file* hasil dari ekstraksi XML kedalam direktori. Berikut Gambar 4.10 Menyimpan hasil ekstraksi *file* XML dan Gambar 4.11 Hasil *output* ekstraksi *file* XML:



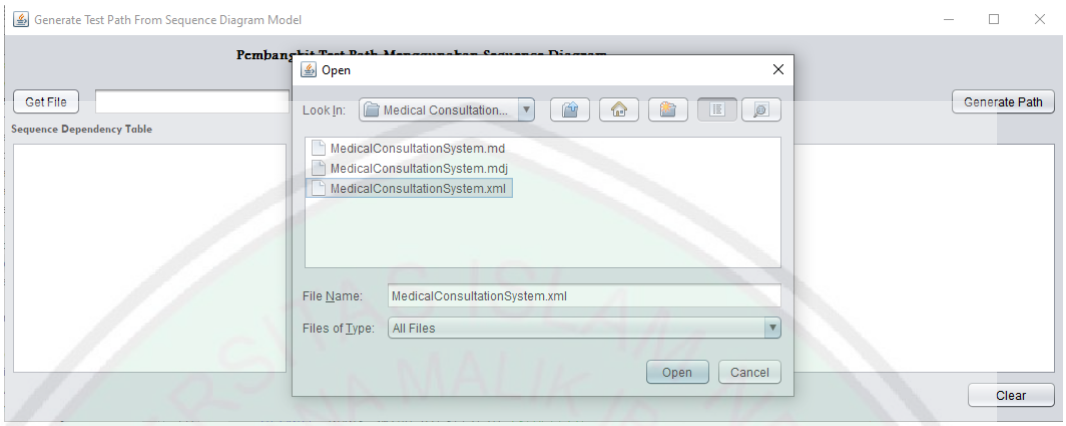
Gambar 4. 11 Menyimpan hasil ekstraksi *file* XML

Gambar 4. 12 Hasil *output* ekstraksi *file* XML

Hasil *output* ekstraksi *file* XML akan dijadikan *input* untuk aplikasi pembangkit *test case* pada komponen pembangkit *test path*. Berikut Gambar 4.12 Komponen pembangkit *test path*:

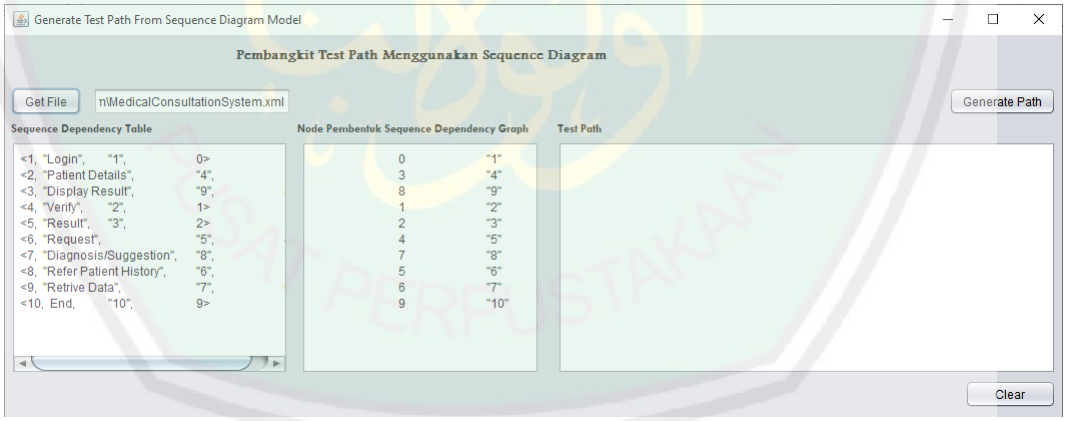
Gambar 4. 13 Komponen pembangkit test path

Pada komponen pembangkit *test path* ini, hal yang pertama dilakukan adalah mengambil *file* XML yang telah dikonversi dengan mengklik *button get file*. Berikut Gambar 4.13 *Button get file*:



Gambar 4. 14 *Button get file*

Pada penelitian ini, data uji coba akan diujikan secara bergantian. Berikut Gambar 4.14 tampilan hasil tombol "*Get File" file* XML Sistem Konsultasi Medis.



Gambar 4. 15 Tampilan Hasil Tombol *"Get File" file* XML Sistem Konsultasi Medis

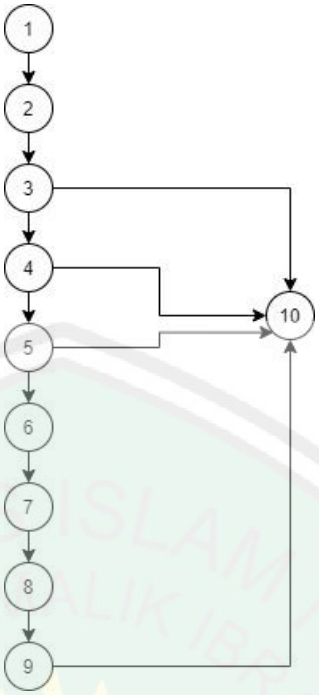
Penjelasan untuk *Sequence Dependency Table* (SDT) pada aplikasi pembangkit test case Gambar 4.14 hasil dari *get file* Sistem Konsultasi Medis yaitu *Symbol* "1", *Activity Name "Login", Sequence Number* "1", dan *Dependency* "0". *Sequence Dependency Graph* (SDG) pada komponen

pembangkit *test path* diambil dari *Sequence Dependency Table* (SDT) pada kolom *dependency* sebagai *node*1 dan kolom *sequence number* sebagai *node*2. *Node* pembentuk *Sequence Dependency Graph* (SDG) tersimpan di dalam *hashmap*. Berikut bentuk *graph* yang tersimpan di *hashmap* dalam sistem pembangkit ini.

Tabel 4. 1 *Graph* dari *Sequence Diagram* Sistem Konsultasi Medis

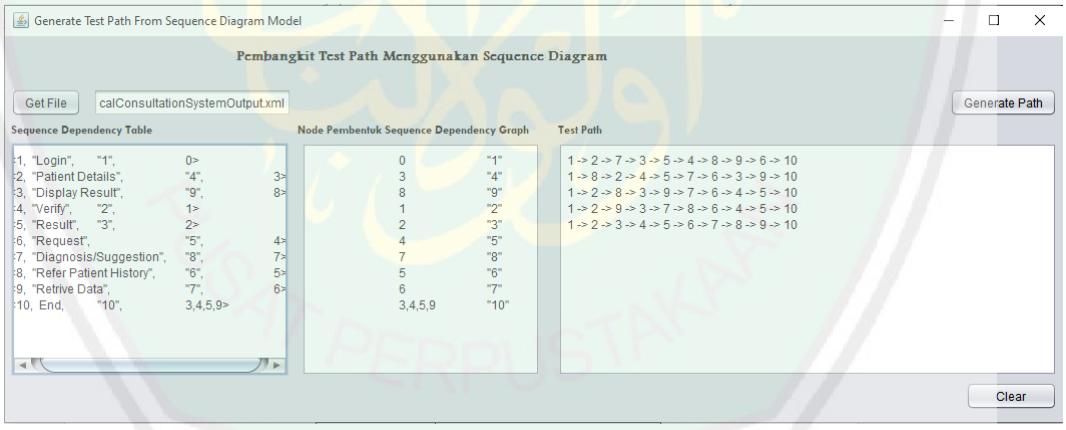
Key	Value
1	[2]
2	[3]
3	[4, 10]
4	[5,10]
5	[6,10]
6	[7]
7	[8]
8	[9]
9	[10]

Graph yang tersimpan dalam *hashmap*, kemudian ditambahkan metode *Simulated Annealing* akan menghasilkan *test path*. *Test path* bisa didapatkan dengan mengklik *button generate path*. Untuk represntasi *Graph* yang dihasilkan dari *sequence diagram* Sistem Konsultasi Medis akan ditunjukkan pada Gambar 4.16.



Gambar 4. 16 Represntasi *Graph* dari *sequence diagram* Sistem Konsultasi Medis

Berikut tampilan hasil dari *generate path* pada seluruh uji coba pada penelitian ini.



Gambar 4. 17 Tampilan hasil *Button Generate Path File* Sistem Konsultasi Medis

Penjelasan untuk *generate path*, pada Gambar 4.17 tampilan hasil *button generate path file* Sistem Konsultasi Medis yaitu *Symbol* yang menunjukkan hubungan antar *node* menjadi satu baris disebut dengan *test path*. *Test path* tersebut diperoleh dari menerapkan metode *simulated annealing*. Dalam metode *simulated annealing* proses pertama adalah membangkitkan bilangan *random*.

Bilangan *random* yang dibangkitkan pada *Sistem Konsultasi Medis* adalah 7 – 2 – 10 – 1 – 4 – 8 – 5 – 3 – 9 – 6. Untuk hasil proses *simulated annealing* sehingga menghasilkan lima *test path* seperti pada Gambar 4.17 akan dijelaskan pada Tabel 4.2 Hasil metode *simulated annealing* Sistem Konsultasi Medis. Berikut Tabel Tabel 4.2 Hasil metode *simulated annealing* Sistem Konsultasi Medis dan Tabel 4.3 *Output* Sistem Konsultasi Medis dari aplikasi pembangkit *test case*:

Tabel 4. 2 Hasil metode *simulated annealing* Sistem Konsultasi Medis

No	Iterasi	*Test Path*	Temperatur	Z_c
1	22	1-2-7-3-5-4-8-9-6-10	100000.0	3
2	24	1-8-2-4-5-7-6-3-9-10	91106.61429076765	2
3	26	1-2-8-3-9-7-6-4-5-10	75169.21927868745	3
4	28	1-5-4-2-7-3-9-8-6-10	44165.49077612001	1
5	30	1-5-4-2-6-9-7-3-8-10	26184.278225749116	1
6	32	1-2-9-3-7-8-6-4-5-10	5673.828280214992	4
7	38	1-2-3-4-5-6-7-8-9-10	0.9996469615182734	9

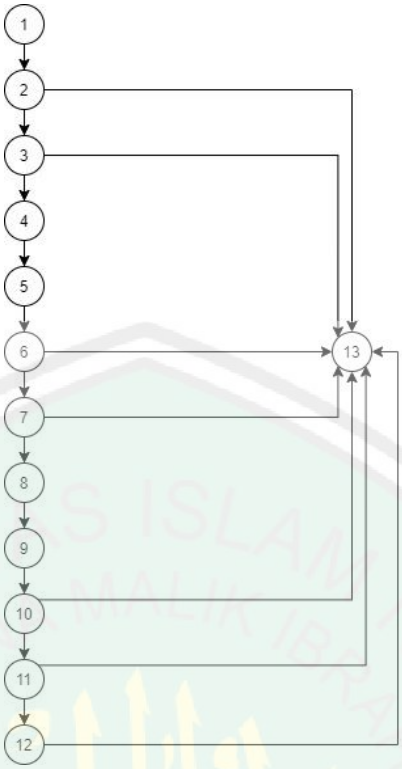
Tabel 4.2 menampilkan beberapa *test path* yang berhasil di*generate* menggunakan metode *simulated annealing*. Iterasi menjelaskan letak dimana *test path* itu berhasil dibangkitkan pada iterasi keberapa. Fungsi dari temperatur adalah suatu nilai kendali yang membuat suatu *node* acak dapat bergerak naik atau tidak. Z_c adalah nilai *path* dari jumlah perjalanan alur *path*. Dalam pengambilan solusi untuk *path* terbaik menggunakan Z_c dengan jumlah melebihi satu. Mengambil *test path* Z_c melebihi satu dikarenakan jika Z_c bernilai satu maka di dalam *test path* tersebut tidak ada *path* yang saling terhubung satu sama lain. Sehingga dari penjelasan tersebut, *test path* yang dijadikan sebagai solusi terbaik akan ditampilkan pada Tabel 4.3 *Output* Sistem Konsultasi Medis dari aplikasi pembangkit *test case* dan juga terdapat pada Gambar 4.17.

Tabel 4. 3 *Output* Sistem Konsultasi Medis dari aplikasi pembangkit *test case*

No	Hasil	
	Test path	***Test case***
1	1-2-7-3-5-4-8-9-6-10	*Login, Verify, Retrieve Details, Result, Request, Enter Patient Details, Diagnosis Medicine, Display Result, Refer Patient History, End.*
2	1-8-2-4-5-7-6-3-9-10	*Login, Diagnosis Medicine, Verify, Enter Patient Details, Request, Retrieve Details, Refer Patient History, Result, Display Result, End.*
3	1-2-8-3-9-7-6-4-5-10	*Login,Verify, Diagnosis Medicine, Result, Display Result, Retrieve Details, Refer Patient History, Enter Patient Details, Request, End.*
4	1-2-9-3-7-8-6-4-5-10	*Login, Verify, Display Result, Result, Retrieve Details, Diagnosis Medicine, Refer Patient History, Enter Patient Details, Request, End.*
5	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, hasil *test path* dari Sistem Konsultasi Medis, Sistem ATM, Aktifitas Pemberangkatan Pesawat, Sistem Keamanan *Smart Home* akan dibandingkan dengan hasil pada *paper* Shanti et al (2012), Priya et al (2013), Rhmann (2016) dan Mani & Prasanna (2017). Perbandingan pertama yakni dari hasil *generate path* Sistem ATM pada aplikasi pembangkit *test case* otomatis dengan hasil dari *paper* Shanti et al (2012). Gambar 4.18 Representasi *Graph* yang dihasilkan dari *sequence diagram* Sistem ATM.



Gambar 4. 18 Representasi *Graph* dari *sequence diagram* Sistem ATM

Bilangan *random* yang dibangkitkan pada Sistem ATM adalah 3 – 8 – 11 – 10 – 5 – 9 – 13 – 1 – 7 – 2 – 12 – 4 – 6. Untuk hasil proses *simulated annealing* sehingga menghasilkan *test path* akan dijelaskan pada Tabel 4.4 Hasil metode *simulated annealing* Sistem ATM dan Tabel 4.5 Perbandingan *output* Sistem ATM dari aplikasi pembangkit *test case* dan *paper* Shanti et al (2012):

Tabel 4. 4 Hasil metode *simulated annealing* Sistem ATM

No	Iterasi	*Test Path*	Temperatur	Z_c
1	24	1-2-8-7-10-3-9-11-6-5-4-12-13	99102.6973	2
2	28	1-4-11-12-6-10-5-2-8-7-3-9-13	72725.51092802135	2
3	34	1-4-5-6-12-10-11-9-7-8-3-2-13	72507.33439523728	4
4	36	1-5-4-2-8-7-3-9-10-12-11-6-13	60546.833899004756	2
5	40	1-5-10-8-2-7-3-9-4-11-12-6-13	60365.19339730774	2
6	48	1-5-3-7-8-2-9-10-4-11-12-6-13	2.8957469523433024	4
7	50	1-2-3-4-5-6-7-8-9-10-11-12-13	0.9996469615182734	12

Tabel 4. 5 Perbandingan *output* Sistem ATM dari aplikasi pembangkit *test case* dan *paper* Shanti et al (2012)

No	Hasil Uji Coba Aplikasi		No	Hasil Uji Coba *Paper*	
	Testpath	***Test case***		***Testpath***	***Test case***
1	1-2-8-7-10-3-9-11-6-5-4-12-13	*Insert Card, Validate (Card), Enter Amount, Return, Check (Balance), Return, Amount, Return, Verify (Pin), Pin, Enter Pin, Take Cash, End*	1	A B M	*Insert Card, Validate (Card), End*
2	1-4-11-12-6-10-5-2-8-7-3-9-13	*Insert card, Enter Pin, Return, Take Cash, Verify (pin), Check (Balance), Pin, Validate (Card), Enter Amount, Return, Return, Amount, End.*	2	A B D E F M	*Insert Card, Validate (Card), Enter Pin, Pin, Verify (Pin), End*
3	1-4-5-6-12-10-11-9-7-8-3-2-13	*Insert card, Enter Pin, Pin, Verify (pin), Take Cash, Check (Balance), Return, Amount, Enter Amount, Return, Validate (Card), End.*	3	A B D E F H I J M	*Insert Card, Validate (Card), Enter Pin, Pin, Verify (Pin), Enter Amount, Amount, Check (Balance), End*
4	1-5-4-2-8-7-3-9-10-12-11-6-13	*Insert card, Pin, Enter Pin, Validate (Card), Enter Amount, Return, Return, Amount, Check (Balance), Take Cash, Return, Verify (pin), End.*	4	A B D E F H I J L M	*Insert Card, Validate (Card), Enter Pin, Pin, Verify (Pin), Enter Amount, Amount, Check (Balance), Take Cash, End*
5	1-5-10-8-2-7-3-9-4-11-12-6-13	*Insert card, Pin, Check (Balance),Enter Amount, Validate (Card), Return, Return, Amount, Enter Pin, Return, Take Cash, Verify (pin), End.*			
6	1-5-3-7-8-2-9-10-4-11-12-6-13	*Insert card, Pin, Return, Return, Enter Amount, Validate (Card), Amount, Check (Balance), Enter Pin, Return, Take Cash, Verify (pin), 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 4.5 merupakan perbandingan hasil dari pembangkit *test case* secara otomatis dan *paper* dari Shanti et al (2012). Aplikasi pembangkit *test case* otomatis ini berhasil menghasilkan *output* tujuh *test path* sedangkan *paper* dari Shanti et al (2012) hanya menghasilkan empat *test path* saja. Disini ada terjadinya perbedaan dalam mengeksekusi *path*, dalam aplikasi pembangkit *test case* yang mengandung pesan "*Return"* akan secara otomatis mengeksekusi semua jalur yang ada pada *graph*, sedangkan dari Shanti et al (2012) tidak melakukan eksekusi pesan *"Return"*. Jumlah *path* dalam *test path* pada aplikasi pembangkit *test case* otomatis mempunya jumlah yang sama seperti yang dijelaskan pada sub bab 2.9 tentang persyaratan metode *Simulated Annealing* sedangkan pada *paper* Shanti et al (2012) memiliki jumlah *path* yang berbeda pada setiap *test path* nya.

Perbandingan kedua yaitu Sistem Konsultasi Medis dari hasil aplikasi pembangkit *test case* dan *paper* dari Priya et al (2013). Bentuk *Graph* dari Sistem Konsultasi Medis sudah dijelaskan pada Gambar 4.16 dan hasil *simulated annealing* sudah dijelaskan pada Tabel 4.2. Berikut Tabel 4.6 Perbandingan *output* Sistem Konsultasi Medis dari aplikasi pembangkit *test case* dan *paper* Priya et al (2013):

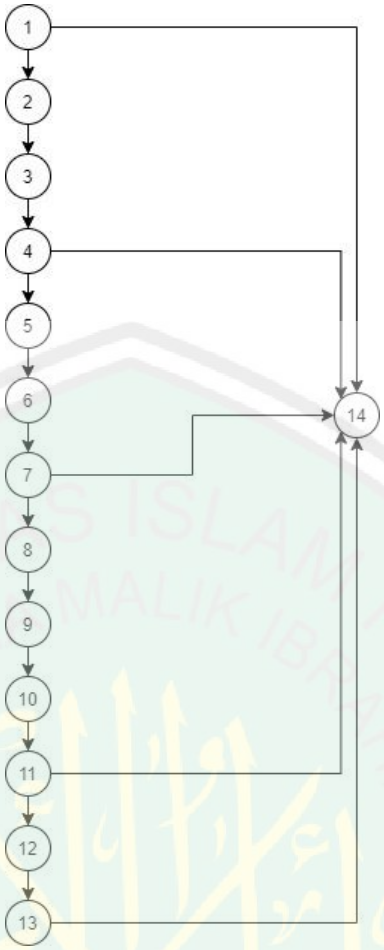
Tabel 4. 6 Perbandingan *output* Sistem Konsultasi Medis dari aplikasi pembangkit *test case* dan *paper* Priya et al (2013)

No	Hasil Uji Coba Aplikasi		No	Hasil Uji Coba *Paper*	
	Testpath	***Test case***			***Testpath***
1	1-2-7-3-5-4-8-9-6-10	*Login, Verify, Retrieve Details, Result, Request, Enter Patient Details, Diagnosis/suggestion, Display Result, Refer Patient History, End.*	1	A B C J	*Login*, Verify, *Result*, End
2	1-8-2-4-5-7-6-3-9-10	*Login, Diagnosis/suggestion, Verify, Enter Patient Details, Request, Retrieve Details, Refer Patient History, Result, Display Result, End.*	2	A B C D J	*Login, Verify, Result, Enter Patient Details, End*

3	1-2-8-3-9-7-6-4-5-10	*Login,Verify Diagnosis/suggestion, Result, Display Result, Retrieve Details, Refer Patient History, Enter Patient Details, Request, End.*	3	A B C D E J	*Login, Verify, Result, Enter Patient Details, Request, End*
4	1-2-9-3-7-8-6-4-5-10	*Login, Verify, Display Result, Result, Retrieve Details, Diagnosis/suggestion, Refer Patient History, Enter Patient Details, Request, End.*	4	A B C D E F G H I J	*Login, Verify, Result, Enter Patient Details, Request, Refer Patient History, Retrieve Details, Diagnosis/suggestion, Display Result, End.*
5	1-2-3-4-5-6-7-8-9-10	*Login, Verify, Result, Enter Patient Details, Request, Refer Patient History, Retrieve Details, Diagnosis/suggestion, Display Result, End.*			

Tabel 4.6 merupakan perbandingan hasil *Sistem Konsultasi Medis* dari pembangkit *test case* secara otomatis dan *paper* dari Priya et al (2013). Aplikasi pembangkit *test case* otomatis ini berhasil menghasilkan *output* lima *test path* sedangkan *paper* dari Priya et al (2013) menghasilkan empat *test path*. Hasil *test path* pada Sistem Konsultasi Medis dari pembangkit *test case* otomatis dan *Test path* Sistem Konsultasi Medis dari Priya et al (2013) terdapat yang sama yang membedakan adalah aplikasi pembangkit *test case* otomatis menggunakan simbol angka sedangkan *paper* Priya et al (2013) menggunakan simbol huruf. *Test path* pada Sistem Konsultasi Medis dari pembangkit *test case* otomatis menghasilkan 1-2-3-4-5-6-7-8-9-10 dan *test path* optimal Sistem Konsultasi Medis dari Priya et al (2013) menghasilkan A-B-C-D-E-F-G-H-I-J.

Perbandingan ketiga yaitu Aktifitas Pemberangkatan Pesawat dari hasil aplikasi pembangkit *test case* dan *paper* dari Rhmann (2016). Gambar 4.19 Representasi *Graph* yang dihasilkan dari *sequence diagram* Aktifitas Pemberangkatan Pesawat.



Gambar 4. 19 Representasi *Graph* dari *sequence diagram* Aktifitas Pemberangkatan Pesawat

Bilangan *random* yang dibangkitkan pada Aktifitas Pemberangkatan Pesawat adalah 10 – 7 – 8 – 13 – 3 – 4 – 14 – 2 – 12 – 1 – 5 – 9 – 6 – 11. Untuk hasil proses *simulated annealing* sehingga menghasilkan *test path* akan dijelaskan pada Tabel 4.7 Hasil metode *simulated annealing* Aktifitas Pemberangkatan Pesawat. Berikut Tabel 4.7 Hasil metode *simulated annealing* Aktifitas Pemberangkatan Pesawat dan Tabel 4.8 Perbandingan *output* Aktifitas Pemberangkatan Pesawat dari aplikasi pembangkit *test case* dan *paper* Shanti et al (2012):

Tabel 4. 7 Hasil metode *simulated annealing* Aktifitas Pemberangkatan Pesawat

No	Iterasi	*Test Path*	Temperatur	Z_c
1	24	1-13-8-4-5-2-6-10-3-9-11-7-12-14	100000.0	2
2	26	1-6-11-13-8-12-7-2-5-3-4-9-10-14	97918.80578299021	2
3	28	1-9-2-4-6-11-12-8-13-7-5-3-10-14	894.1256407951954	2
4	32	1-5-9-2-3-6-12-13-7-8-11-10-4-14	175.9864358650526	4
5	34	1-5-9-2-3-4-12-13-7-8-11-6-10-14	174.93210112778507	4
6	38	1-4-9-2-3-10-5-12-7-13-8-11-6-14	9.602679666688696	2
7	40	1-6-5-10-3-9-2-4-7-8-13-12-11-14	3.2459706759314635	2
8	44	1-2-3-4-5-6-7-8-9-10-11-12-13-14	0.9996469615182734	13

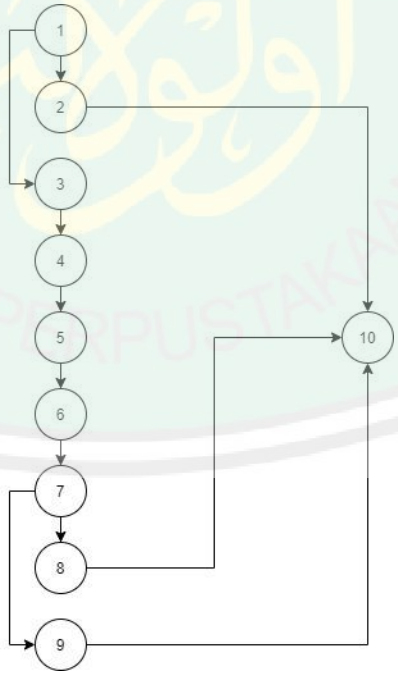
Tabel 4. 8 Perbandingan *output* Aktifitas Pemberangkatan Pesawat dari aplikasi pembangkit *test case* dan *paper* Rhmann (2016)

No	Hasil Uji Coba Aplikasi		No	Hasil Uji Coba *Paper*	
	Testpath	*Test case*		*Testpath*	*Test case*
1	1-13-8-4-5-2-6-10-3-9-11-7-12-14	*Request for Pushback, Depart, Grant Taxiing Clearance, Request to Leave Ramp, Grant to Leave Ramp, Grant Pushback Clearance, Leave Ramp, Aircraf on Runway, Pushback, Taxiing, Request Departure Clearance, Request Taxiing Clearance, Grant Departure Clearance, End.*	1	1 14	*Request for Pushback, Permission not Granted, End*
2	1-6-11-13-8-12-7-2-5-3-4-9-10-14	*Request for Pushback, Leave Ramp, Request Departure Clearance, Depart, Grant Taxiing Clearance, Grant Departure Clearance, Request Taxiing Clearance, Grant Pushbac Clearance, Grant to Leave Ramp, Pushback, Request to Leave Ramp, Taxiing, Aircraft on Runway, End.*	2	1 2 3 4 14	*Request for Pushback, Grant Pushback Clearance, Pushback, Request to Leave Ramp, Permission not Granted, End*
3	1-9-2-4-6-11-12-8-13-7-5-3-10-14	*Reques for Pushback, Taxiing, Grandh Pushback Clearance, Request to Leave Ramp, Leave Ramp, Request Departure Clearance, Grant Departure Clearance, Grant Taxiing, Clearance, Depart, Request Taxiing Clearance, Grant to Leave Ramp, Pushback, Aircraft on Runway, 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 Taxiing Clearance, Permission not Granted, End*
4	1-5-9-2-3-6-12-13-7-8-11-10-4-14	*Request for Pushback, Grant to Leave Ramp, Taxiing, Grant Pushback Clearance, Pushback, Leave Ramp, Grant Departure Clearance, Depart, Request Taxiing Clearance, Grant Taxiing Clearance, Request Departure Clearance,*	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,*

		Aircraft on Runway, Request to Leave Ramp, End.			*Request Taxiing Clearance, Grant Taxiing Clearance, Taxiing, Aircraft on Runway, Request Departure Clearance, Permission not Granted, End*
5	1-5-9-2-3-4-12-13-7-8-11-6-10-14	*Request for Pushback, Grant to Leave Ramp, Taxiing, Grant Pushback Clearance, Pushback, Request to Leave Ramp, Grant Departure Clearance, Depart, Request Taxiing Clearance, Grant Taxiing Clearance, Request Departure Clearance, Leave Ramp, Aircraft on Runway, End.*	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, Grant to Leave Ramp, Leave Ramp, Request Taxiing Clearance, Grant Taxiing Clearance, Taxiing, AircraftKon Runway, Request Departure Clearance, Grant Departure Clearance, Depart, End*
6	1-4-9-2-3-10-5-12-7-13-8-11-6-14	*Request for Pushback, Request to Leave Ramp, Taxiing, Grant Pushback Clearance, Pushback, Aircraft on Runway, Grant to Leave Ramp, Request Taxiing Clearance, Depart, Grant Taxiing Clearance, Request Departure Clearance, Leave Ramp, End.*			
7	1-6-5-10-3-9-2-4-7-8-13-12-11-14	*Request for Pushback, Leave Ramp, Grant to Leave Ramp, Aircraft on Runway, Pushback, Taxiing, Grant Pushback Clearance, Request to Leave Ramp, Reque Taxiing Clearance, Grant Taxiing Clearance, Depart, Grant Departure Clearance, Request Departure Clearance, End.*			
8	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, Grant to Leave Ramp, Leave Ramp, Request Taxiing Clearance, Grant Taxiing Clearance, Taxiing, Aircraft on Runway, Request Departure Clearance, Grant Departure Clearance, Depart, End.*			

Tabel 4.8 merupakan perbandingan hasil Aktifitas Pemberangkatan Pesawat dari pembangkit *test case* secara otomatis dan *paper* Rhmann (2016). Aplikasi pembangkit *test case* otomatis ini berhasil menghasilkan *output* delapan *test path* sedangkan *paper* dari Rhmann (2016) menghasilkan lima *test path*. Namun, *output* dari *paper* Rhmann (2016) sebelum *node End* ada penambahan pesan "*Permission not Granted"* yang artinya *request* dari sistem ini bisa ditolak. Salah satu *test path* Aktifitas Pemberangkatan Pesawat dari pembangkit *test case* otomatis dan *test path* Aktifitas Pemberangkatan Pesawat dari Priya et al (2013) mempunyai hasil yang sama yaitu 1-2-3-4-5-6-7-8-9-10-11-12-13-14.

Perbandingan keempat yaitu Sistem Keamanan *Smart Home* dari hasil aplikasi pembangkit *test case* dan *paper* dari Mani & Prasanna (2017). Gambar 4.20 Representasi *Graph* yang dihasilkan dari *sequence diagram* Sistem Keamanan *Smart Home.*



Gambar 4. 20 Representasi *Graph* yang dihasilkan dari *sequence diagram* Sistem Keamanan *Smart Home.*

Bilangan *random* yang dibangkitkan pada Sistem Keamanan *Smart Home* adalah 10 – 7 – 8 – 6 – 2 – 4 – 5 – 1 – 3 – 9. Untuk hasil proses *simulated annealing* sehingga menghasilkan *test path* akan dijelaskan pada Tabel 4.9 Hasil metode *simulated annealing* Sistem Keamanan *Smart Home*. Berikut Tabel 4.9 Hasil metode *simulated annealing* Sistem Keamanan *Smart Home* dan Tabel 4.10 Perbandingan *output* Sistem Keamanan *Smart Home* dari aplikasi pembangkit *test case* dan *paper* Mani & Prasanna (2017):

Tabel 4. 9 Hasil metode *simulated annealing* Sistem Keamanan *Smart Home*

No	Iterasi	*Test Path*	Temperatur	Z_c
1	14	1-5-7-8-6-2-4-3-9-10	99400.9	2
2	16	1-6-7-8-3-4-2-5-9-10	76997.8809405273	4
3	18	1-7-6-3-5-4-2-8-9-10	11188.462463866495	2
4	20	1-2-3-4-5-6-7-8-9-10	0.9996469615182734	9

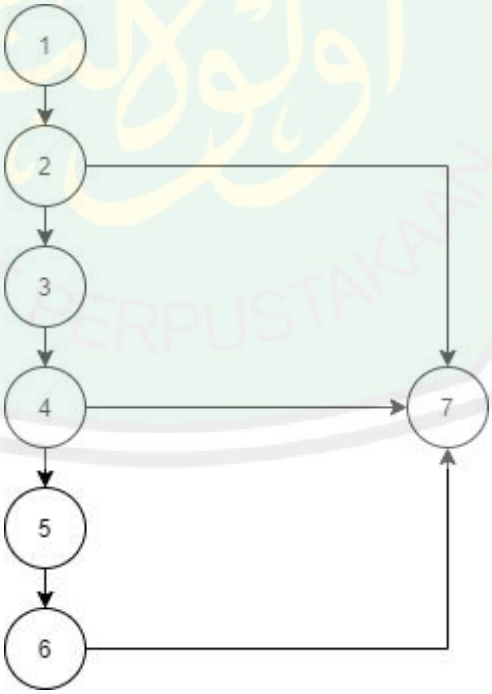
Tabel 4. 10 Perbandingan *output* Sistem Keamanan *Smart Home* dari aplikasi pembangkit *test case* dan *paper* Mani & Prasanna (2017)

No	Hasil Uji Coba Aplikasi		No	Hasil Uji Coba *Paper*	
	Testpath	*Test case*		*Testpath*	*Test case*
1	1-5-7-8-6-2-4-3-9-10	*Press input control, Capture the image, Guest image, Accept the request, Process the image,* Alarm, *Switch on the camera, Input signal, Reject the request, End.*	1	1-2-10	*Press input control,* Alarm, *End.*
2	1-6-7-8-3-4-2-5-9-10	*Press input control, Process the image, Guest image, Accept the request, Input signal, Switch on the camera,* Alarm, *Capture the image, Reject the request, End.*	2	1-2-3-4-5-6-7-9-10	*Press input control,* Alarm, *Input signal, Switch on the camera, Capture the image, Process the image, Guest image, Reject the request, End.*
3	1-7-6-3-5-4-2-8-9-10	*Press input control, Guest image, Process the image, Input signal, Capture the image, Switch on the camera,* Alarm, *Accept the request, Reject the request, End.*	3	1-2-3-4-5-6-7-8-10	*Press input control,* Alarm, *Input signal, Switch on the camera, Capture the image, Process the image, Guest image, Accept the request, End.*
4	1-2-3-4-5-6-7-8-9-10	*Press input control,* Alarm, *Input signal, Switch on the camera,*			

		Capture the image, Process the image, Guest image, Accept the request, Reject the request, End.			

Tabel 4.10 merupakan perbandingan hasil Sistem Keamanan *Smart Home* dari pembangkit *test case* secara otomatis dan *paper* Mani & Prasanna (2017). Aplikasi pembangkit *test case* otomatis ini berhasil menghasilkan *output* empat *test path* sedangkan *paper* dari Mani & Prasanna (2017) menghasilkan tiga *test path*.

Pembahasan selanjutnya yaitu dari studi kasus penelitian ini Sistem Penilaian Pembelajaran. Dalam Sistem Penilaian Pembelajaran ini sendiri memiliki empat uji coba. Uji coba pertama dari Sistem Penilaian Pembelajaran yaitu *sequence diagram* Memasukkan Nilai. Gambar 4.21 Representasi *Graph* yang dihasilkan dari *sequence diagram* Memasukkan Nilai.



Gambar 4. 21 Representasi *Graph* dari *sequence diagram* Memasukkan Nilai.

Bilangan *random* yang dibangkitkan pada Memasukkan Nilai adalah 4 – 1 – 3 – 2 – 5 – 7 – 6. Untuk hasil proses *simulated annealing* sehingga menghasilkan *test path* akan dijelaskan pada Tabel 4.11 Hasil metode *simulated annealing* Memasukkan Nilai. Berikut Tabel 4.11 Hasil metode *simulated annealing* Memasukkan Nilai dan Tabel 4.12 *Output* Memasukkan Nilai:

Tabel 4. 11 Hasil metode *simulated annealing* Memasukkan Nilai

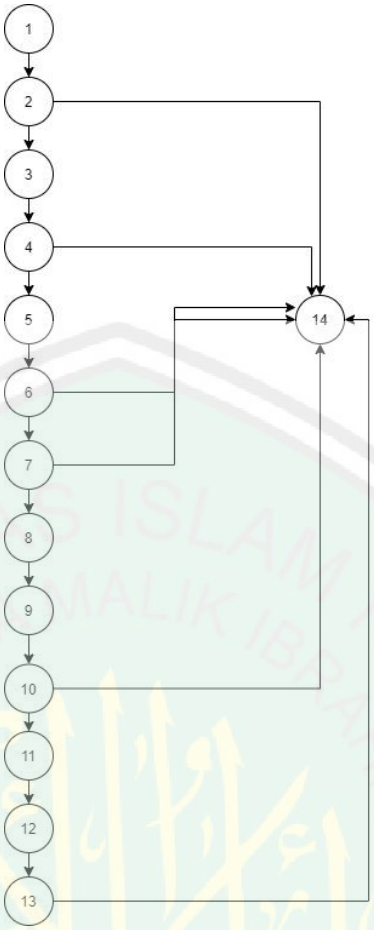
No	Iterasi	*Test Path*	Temperatur	Z_c
1	10	1-5-4-2-3-6-7	100000.0	2
2	14	1-2-3-5-6-4-7	99102.6973	4
3	18	1-2-3-4-5-6-7	0.9996469615182734	6

Tabel 4. 12 *Output* Memasukkan Nilai

No	Hasil Uji Coba Aplikasi	
	Testpath	*Test case*
1	1-5-4-2-3-6-7	Pilih mata kuliah, *input* nilai siswa, menampilkan nama siswa, menampilkan data mata kuliah, memasukkan NIM, simpan nilai, *End*.
2	1-2-3-5-6-4-7	Pilih mata kuliah, menampilkan data mata kuliah, memasukkan NIM, *input* nilai siswa, simpan nilai, menampilkan nama siswa, *End*.
3	1-2-3-4-5-6-7	Pilih mata kuliah, menampilkan data mata kuliah, memasukkan NIM, menampilkan nama siswa, *input* nilai siswa, simpan nilai, *End*.

Tabel 4.12 merupakan *output* dari *sequence diagram* Memasukkan Nilai yang menghasilkan tiga *test path*.

Uji coba kedua dari Sistem Penilaian Pembelajaran yaitu *sequence diagram* Perhitungan Nilai. Gambar 4.22 Representasi *Graph* yang dihasilkan dari *sequence diagram* Perhitungan Nilai.



Gambar 4. 22 Representasi *Graph* yang dihasilkan dari *sequence diagram* Perhitungan Nilai

Bilangan *random* yang dibangkitkan pada Perhitungan Nilai adalah 14 – 7 – 13 – 1 – 6 – 11 – 2 – 10 – 9 – 8 – 4 – 3 – 5 – 12. Untuk hasil proses *simulated annealing* sehingga menghasilkan *test path* akan dijelaskan pada Tabel 4.13 Hasil metode *simulated annealing* Perhitungan Nilai. Berikut Tabel 4.13 Hasil metode *simulated annealing* Perhitungan Nilai dan Tabel 4.14 *Output* Perhitungan Nilai:

Tabel 4. 13 Hasil metode *simulated annealing* Perhitungan Nilai

No	Iterasi	*Test Path*	Temperatur	Z_c
1	24	1-2-5-12-8-7-4-6-10-11-9-13-3-14	100000.0	2
2	30	1-2-5-12-11-7-4-6-10-8-9-13-3-14	99102.6973	3
3	32	1-2-5-3-11-4-6-7-10-8-9-12-13-14	98805.3892081	5
4	36	1-7-3-2-4-12-13-8-5-11-6-10-9-14	67463.01382227564	2
5	42	1-11-6-7-12-9-13-8-10-2-4-5-3-14	119.44132654027358	3
6	52	1-12-7-8-3-2-11-4-5-6-13-9-10-14	98.25166264304144	5
7	58	1-11-7-6-12-8-10-9-13-2-3-4-5-14	1.9244300877817426	4
8	60	1-2-3-4-5-6-7-8-9-10-11-12-13-14	0.9996469615182734	13

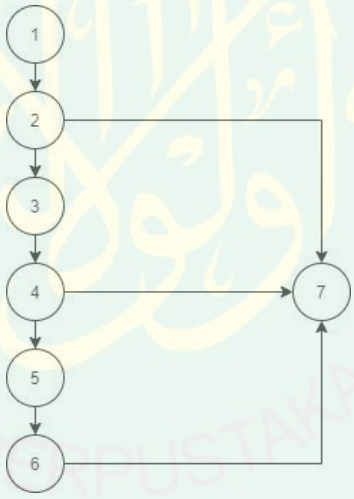
Tabel 4. 14 *Output* Perhitungan Nilai

No	Hasil Uji Coba Aplikasi	
	Testpath	***Test case***
1	1-2-5-12-8-7-4-6-10-11-9-13-3-14	Pilih mata kuliah, menampilkan data mata kuliah, permintaan hitung jumlah nilai, menentukan konversi nilai, permintaan rata-rata nilai, menghitung rata-rata nilai, menampilkan jumlah nilai, menampilkan nama siswa, menghitung jumlah nilai, menampilkan *final score*, permintaan konversi nilai menjadi huruf, menghitung rata-rata nilai, menampilkan nilai dalam huruf, memasukkan NIM, *End.*
2	1-2-5-12-11-7-4-6-10-8-9-13-3-14	Pilih mata kuliah, menampilkan data mata kuliah, permintaan hitung jumlah nilai, menentukan konversi nilai, permintaan konversi nilai menjadi huruf, menampilkan jumlah nilai, menampilkan nama siswa, menghitung jumlah nilai, menampilkan *final score*, permintaan rata-rata nilai, menghitung rata-rata nilai, menampilkan nilai dalam huruf, memasukkan NIM, *End.*
3	1-2-5-3-11-4-6-7-10-8-9-12-13-14	Pilih mata kuliah, menampilkan data mata kuliah, permintaan hitung jumlah nilai, memasukkan NIM, permintaan konversi nilai menjadi huruf, menampilkan nama siswa, menghitung jumlah nilai, menampilkan jumlah nilai, menampilkan *final score*, permintaan rata-rata nilai, menghitung rata-rata nilai, menentukan konversi nilai, menampilkan nilai dalam huruf, *End.*
4	1-7-3-2-4-12-13-8-5-11-6-10-9-14	Pilih mata kuliah, menampilkan jumlah nilai, , memasukkan NIM, menampilkan data mata kuliah, menampilkan nama siswa, menentukan konversi nilai, menampilkan nilai dalam huruf, permintaan rata-rata nilai, permintaan hitung jumlah nilai, permintaan konversi nilai menjadi huruf, menghitung jumlah nilai, menampilkan *final score*, menghitung rata-rata nilai, *End.*
5	1-11-6-7-12-9-13-8-10-2-4-5-3-14	Pilih mata kuliah, permintaan konversi nilai menjadi huruf, menghitung jumlah nilai, menampilkan jumlah nilai, menentukan konversi nilai, menghitung rata-rata nilai, menampilkan nilai dalam huruf, permintaan rata-rata nilai, menampilkan *final score*, menampilkan data mata kuliah, menampilkan nama siswa, permintaan hitung jumlah nilai, memasukkan NIM, *End.*
6	1-12-7-8-3-2-11-4-5-6-13-9-10-14	Pilih mata kuliah, menentukan konversi nilai, menampilkan jumlah nilai, permintaan rata-rata nilai, memasukkan NIM, menampilkan data mata kuliah, permintaan konversi nilai menjadi huruf, menampilkan nama siswa, permintaan hitung jumlah nilai, menghitung jumlah nilai, menampilkan nilai dalam huruf, menghitung rata-rata nilai, menampilkan *final score*, *End.*
7	1-11-7-6-12-8-10-9-13-2-3-4-5-14	Pilih mata kuliah, permintaan konversi nilai menjadi huruf, menampilkan jumlah nilai, menghitung jumlah nilai, menentukan konversi nilai, permintaan rata-rata nilai, menampilkan *final*

		score, menghitung rata-rata nilai, menampilkan nilai dalam huruf, menampilkan data mata kuliah, memasukkan NIM, menampilkan nama siswa, permintaan hitung jumlah nilai, *End.*
8	1-2-3-4-5-6-7-8-9-10-11-12-13-14	Pilih mata kuliah, menampilkan data mata kuliah, memasukkan NIM, menampilkan nama siswa, permintaan hitung jumlah nilai, menghitung jumlah nilai, menampilkan jumlah nilai, permintaan rata-rata nilai, menghitung rata-rata nilai, menampilkan *final score*, permintaan konversi nilai menjadi huruf, menentukan konversi nilai, menampilkan nilai dalam huruf, *End.*

Tabel 4.14 merupakan *output* dari *sequence diagram* Perhitungan Nilai yang menghasilkan delapan *test path*.

Uji coba ketiga dari Sistem Penilaian Pembelajaran yaitu *sequence diagram* Laporan. Berikut Gambar 4.x Representasi *Graph* yang dihasilkan dari *sequence diagram* Laporan:



Gambar 4. 23 Representasi *Graph* dari *sequence diagram* Laporan

Bilangan *random* yang dibangkitkan pada Laporan adalah 7 – 5 – 4 – 6 – 2 – 1 – 3. Untuk hasil proses *simulated annealing* sehingga menghasilkan *test path* akan dijelaskan pada Tabel 4.15 Hasil metode *simulated annealing* Laporan. Berikut Tabel 4.15 Hasil metode *simulated annealing* Laporan dan Tabel 4.16 *Output* Laporan:

Tabel 4. 15 Hasil metode *simulated annealing* Laporan

No	Iterasi	*Test Path*	Temperatur	Z_c
1	10	1-2-3-6-4-5-7	100000.0	4
2	14	1-2-4-5-6-3-7	96169.43356035063	4
3	16	1-2-3-4-5-6-7	0.9996469615182734	6

Tabel 4. 16 *Output* Laporan

No	Hasil Uji Coba Aplikasi	
	Testpath	*Test case*
1	1-2-3-6-4-5-7	Pilih mata kuliah, menampilkan data mata kuliah, memasukkan NIM, mencetak laporan nilai, menampilkan nama siswa, permintaan cetak laporan nilai, *End*.
2	1-2-4-5-6-3-7	Pilih mata kuliah, menampilkan data mata kuliah, menampilkan nama siswa, permintaan cetak laporan nilai, mencetak laporan nilai, memasukkan NIM, *End*.
3	1-2-3-4-5-6-7	Pilih mata kuliah, menampilkan data mata kuliah, memasukkan NIM, menampilkan nama siswa, permintaan cetak laporan nilai, mencetak laporan nilai, *End*.

Tabel 4.16 merupakan *output* dari *sequence diagram* Laporan yang menghasilkan tiga *test path*.

Setelah dilakukannya pengujian pada seluruh uji coba, dapat dilihat bahwa dari setiap uji coba menghasilkan *test path* yang berbeda-beda. Contohnya pada perbandingan hasil Sistem Konsultasi Medis dari pembangkit *test case* secara otomatis dan *paper* dari Priya et al (2013), aplikasi pembangkit *test case* otomatis ini menghasilkan *output* lima *test path* sedangkan *paper* dari Priya et al (2013) menghasilkan empat *test path*. Pada pembangkit *test case* secara otomatis menghasilkan *test path* lebih banyak dari pada *paper* Priya et al (2013). Perbedaan hasil *test path* ini tidak mempengaruhi kualitas dari pengujian aplikasi *test case* otomatis ini karena penggunaan metode *simulated annealing* sudah dapat menghasilkan *test path* terbaik.

4.3. Integrasi Islam

Posisi Al-Qur'an dan Hadis dalam ilmu pengetahuan dan teknologi sangat penting sebagai sumber ilmu. Seperti halnya dalam pengembangan teknologi pengujian perangkat lunak. Pengujian *test case* secara otomatis ini dapat memudahkan dalam pengembangan teknologi rekayasa perangkat lunak yang bermanfaat bagi orang lain. Hal ini terdapat dalam kandungan surat Ali-Imran/3:186**:**

لَتُبْلَوُنَّ فِي أَمْوَالِكُمْ وَأَنْفُسِكُمْ وَلَتَسْمَعُنَّ مِنَ الَّذِينَ أُوتُوا الْكِتَابَ مِنْ قَبْلِكُمْ وَمِنَ الَّذِينَ أَشْرَكُوا أَذًى كَثِيرًا وَإِنْ تَصْبِرُوا وَتَتَّقُوا فَإِنَّ ذَلِكَ مِنْ عَزْمِ الْأُمُورِ

Artinya: "Kamu sungguh-sungguh akan diuji terhadap hartamu dan dirimu. Dan (juga) kamu benar-benar akan mendengar dari orang-orang yang diberi al-Kitab sebelum kamu dan dari orang-orang yang mempersekutukan Allah, gangguan yang banyak yang menyakitkan hati. Jika kamu bersabar dan bertakwa, maka sesungguhnya yang demikian itu termasuk urusan yang patut diutamakan." (QS. Ali-Imran/3 : 186).

Tafsir Ibnu Katsir menerangkan Firman Allah SWT, لَتُبْلَوُنَّ فِي أَمْوَالِكُمْ وَأَنْفُسِكُمْ *"Kamu sungguh-sungguh akan diuji terhadap hartamu dan dirimu"* ayat ini memiliki makna pada ayat yang lain, yaitu firman-Nya *"Dan sungguh akan Kami berikan cobaan kepada kalian dengan sedikit ketakutan, kelaparan, kekurangan harta, jiwa, dan buah-buahan"* (QS. Al-Baqarah 2: 155), hingga akhir ayat berikutnya. Dengan kata lain, seorang mukmin itu harus diuji terhadap sesuatu dari hartanya atau dirinya atau anaknya atau istrinya sesuai dengan tingkatan kadar agamanya, apabila agamanya kuat, maka ujiannya lebih dari yang lain. وَأَنْفُسِكُمْ وَلَتَسْمَعُنَّ مِنَ الَّذِينَ أُوتُوا الْكِتَابَ مِنْ قَبْلِكُمْ وَمِنَ الَّذِينَ أَشْرَكُوا أَذًى كَثِيرً "*Dan (juga) kamu benar-benar akan mendengar dari orang-orang yang diberi al-Kitab sebelum kamu dan dari orang-orang yang mempersekutukan Allah, gangguan yang banyak yang menyakitkan hati"* maknanya Allah SWT berfirman kepada orang-orang mukmin ketika mereka tiba di Madinah sebelum Perang Badar untuk

meringankan beban mereka dari tekanan gangguan yang menyakitkan hati yang dilakukan oleh kaum Ahli Kitab dan kaum musyrik. Sekaligus memerintahkan mereka agar bersikap pemaaf dan bersabar serta memberikan ampunan hingga Allah memberikan jalan keluar dari hal tersebut. وَإِنْ تَصْبِرُوا وَتَتَّقُوا فَإِنَّ ذَلِكَ مِنْ عَزْمِ الْأُمُورِ "*Jika kamu bersabar dan bertakwa, maka sesungguhnya yang demikian itu termasuk urusan yang patut diutamakan"* maknanya bahwa Nabi dan para sahabatnya di masa lalu selalu bersikap pemaaf terhadap orang-orang musyrik dan Ahli Kitab, sesuai dengan perintah Allah kepada mereka, dan mereka bersabar dalam menghadapi gangguan yang menyakitkan. Tersebutlah bahwa Rasulullah bersikap pemaaf sesuai dengan pengertiannya dari apa yang diperintahkan oleh Allah kepadanya, sehingga Allah mengizinkan kepada beliau terhadap mereka (yakni bertindak terhadap mereka).

Ujian hidup setiap makhluk-Nya bertujuan untuk mengangkat derajat mereka. Begitu pula dengan pengujian perangkat lunak, dengan membuat *test case* secara otomatis dapat menguji perangkat lunak yang dikembangkan sehingga apabila ada kekurangan dapat diperbaiki. Dengan itu perangkat lunak akan semakin berkualitas untuk kedepannya.

BAB V

PENUTUP

5.1. Kesimpulan

Berdasarkan pada hasil penelitian dalam proses uji coba yang telah dilakukan, dalam skripsi ini berhasil membangkitkan aplikasi pembangkit *test case* otomatis dari *sequence diagram*. Kinerja metode *simulated annealing* pada aplikasi pembangkit *test case* secara otomatis berhasil menghasilkan *test case* terbaik dari masing-masing uji coba dalam waktu singkat. *Sequence diagram* Sistem ATM menghasilkan *output* tujuh *test path, sequence diagram* Sistem Konsultasi Medis menghasilkan *output* lima *test path, sequence diagram* Aktifitas Pemberangkatan Pesawat menghasilkan *output* delapan *test path, sequence diagram* Sistem Keamanan *Smart Home* menghasilkan *output* empat *test path. Output* Sistem Penilaian Pembelajaran yang terdiri dari tiga *sequence diagram*. *Sequence diagram* Memasukkan Nilai menghasilkan tiga *test path. Sequence diagram* Perhitungan Nilai yang menghasilkan delapan *test path. Sequence diagram* Laporan yang menghasilkan tiga *test path.*

5.2. Saran

Peneliti menyadari bahwa penelitian ini masih banyak kekurangan yang diperlukan pengembangannya agar mencapai kinerja yang lebih baik lagi. Berikut beberapa saran yang dapat digunakan sebagai masukkan untuk penelitian selanjutnya:

1. Pengujian dapat menggunakan atau mengkombinasikan diagram UML lain seperti *class diagram, activity diagram, collaboration diagram.*

Statechart diagram, agar menghasilkan *test case* yang lebih baik dan lengkap.

2. Pada aplikasi ini setelah *generate test case* harus di*close* terlebih dahulu sebelum men*generate* pengujian yang lain. Sehingga peneliti selanjutnya dapat memperbaiki sistem.

DAFTAR PUSTAKA

Ariyani. 2017. *Pembangkit Test Case (Kasus Uji) menggunakan Model UML (Unified Modeling Language) Sequence Diagram (Studi Kasus Sistem Penilaian Pembelajaran)*. Skripsi. Universitas Islam Negeri Maulana Malik Ibrahim Malang.

Basu A. 2015. *Software Quality Assurance, Testing and Metrics.* PHI *Learning Privat Limited.*

Catelani, M., Ciani, L., Scarano, V. L. dan Bacioccola, A., 2010, *Software Automated Testing: A Solution to Maximize The Test Plan Coverage and to Increase Software Reliability and Quality in Use, Computer Standards & Interfaces.* 33, 152-158.

Clune, T.L., and R.B. Rood. 2011.*Software testing and verification in climate model development.* IEEE Journal. Focus: climate change software. September-October, pp. 49-55.

Engels, G., Heckel, R. & Sauer, S., 2000. *UML-A Universal Modeling Language?*. In: M. Nielsen & D. Simpson (Editors.). *Application and Theory of Petri Nets* 2000: 21st *International Conference,* ICATPN 2000 Aarhus, Denmark, June 26--30, 2000 Proceedings. Springer Berlin Heidelberg, pp. 24-38

Galin, D., 2004, *Software Quality Assurance.* Pearson Education, Inc. New Jersey.

Hendini. 2016. *Pemodelan UML Sistem Informasi Monitoring Penjualan dan Stok Barang (Studi Kasus: Distro Zhezha Pontianak)*. Jurnal Khatulistiwa Informatika. Vol. IV- No. 2.

Ibn Katsir, Abu al-Fida' Ismail. *Tafsir Ibnu Katsir.* Terj. M. Abdul Ghoffar E.M dan Abu Ihsan Al-Atsari. Jilid VI Cet. I. Bogor: Pustaka Imam asy-Syafi'i, 2004.

Isabella., Retna, Emi. 2012. *Study Paper On Test Case Generation for GUI Based Testing. International Journal of Software Engineering & Applications* (IJSEA). Vol.3, No.1.

Jin, J., and F. Xue. 2011. *Rethinking software testing based on software architecture*. In IEEE Proceeding of 7th International Conference on

Semantics, Knowledge and Grids, pp. 148-151. DOI 10.1109/SKG.2011.32.

Kastogi, Gerry. 2015. *Boggle Solver dengan Algoritma Depth First Search dalam Java*. Institut Teknologi Bandung.

Khan, R. & Amjad, M., 2015. *Automatic Test Case Generation fot Unit Software Testing Using Genetic Algorithm and Mutation Analysis*. IEEE UP Section Conference on Electrical Computer and Electronics (UPCON), pp. 1-5.

Kusuma Dewi S, Purnomo H. 2005. *Artificial Intelligence*. Graha Ilmu: Yogyakarta.

Mani, P., Prasanna, M. 2017. *Test Case Generation For Embedded System Software Using UML Interaction Diagram.* Journal of Engineering Science and Technology. Vol. 12, No. 4 (2017) 860-874

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

Munir, Rinaldi. 2014. *Matematika Diskrit Logika, Himpunan, Matriks, Relasi, Fungsi, Algoritma, Kombinatorian, Peluang Diskrit Edisi Kelima*. Bandung : Informatika.

Novelia, Dwi Putri. 2008. IPB : *Implementasi Model Based Testing untuk Pembangkitan Test Case (* Studi Kasus: Sistem Ritel Enterprise Resource Planning).

Panthi, Vikas et al. 2012. *International Journal of Applied Information Syatems: Automatic Test Case Generation using Sequence Diagram*. ISSN: 2249-0868.

Pressman, Roger S. 2002. *Rekayasa Perangkat Lunak: Pendekatan Praktisi Edisi II*. Terjemahkan LN Hananingrum. ANDI: Yogyakarta.

Priya, S. Shanmuga et al. 2013. *International Journal oof Advanced Research in Computer Science and Software Angineering: Test Path Generation Using UML Sequence* Diagram.ISSN: 2277 128X.

Rhmann, Wasiur., Sanexa, Vipin. 2016. *British Journal of Mathematics & Computer Science: Generation of Test Case from UML Sequence Diagram Based on Extenics Theory.* ISSN: 2231-0851.

Rina, Violyta. 2009. *Pengujian Perangkat Lunak.* Fakultas Ilmu Komputer Universitas Indonesia.

Samana E, et al. 2015. *Aplikasi Simulated Annealing untuk Menyelesaikan Travelling Salesman Problem.* Buletin Ilmiah Mat. Stat. dan Terapannya (Bimaster). Vol. 03- No.1. Hal 25-32.

Shanti, A.V.K et al. 2012. *International Conference on Software and Computer Applications: Automated Test Cases Generation from UML Sequence Diagram.* Vol. 41.

Sholiq. 2010. *Analisis dan Perancangan Berorientasi Obyek*. Bandung: CV. Muara Indah

Sofianti, D. T. 2004. *Penjadwalan Multipurpose Batch Chemical Plant dengan Metode Optimasi Gabungan: Algoritma Genetika- Simulasi Annealing.* Proceedings, Komputer dan Sistem Intelijen (KOMMIT2004). 297-308.

Sulistyorini. 2009. *Pemodelan Visual dengan Menggunakan UML dan* Star UML. Jurnal Teknologi Informasi DINAMIK. Vol: XIV. No: 1.

Suyanto. 2010. Algoritma Optimasi: Deterministik atau Probabilistik?. Yogyakarta: Graha Ilmu

Taisirul Karimir Rahman fi Tafsir Kalamil Mannan.

Tripathy, Abinas., Mitra, Anirba. 2012. *Test Case Generation Using Activity Diagram and Sequence Diagram.* Proceedings of ICAdc. AISC. pp.121-129.

LAMPIRAN

1. *Sequence diagram* dalam format XML contoh dari Sistem Konsultasi Medis

```
(object Design "Logical View"
   attributes        (list Attribute_Set)
   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
    showClassOfObject     TRUE
    notation         "Unified")
  root_usecase_package  (object Class_Category "Use Case View"
    quid             "5852B5AB0154"
    exportControl    "Public"
    global           TRUE
    logical_models        (list unit_reference_list
       (object Mechanism @1
             logical_models        (list unit_reference_list
                (object Object "Patient"
                      quid             "5852B5CD0022"
                      collaborators    (list link_list
                         (object Link
                               quid             "5852B61201E5"
                               supplier         "User Interface"
                               quidu            "5852B5CF02EC"
                               messages         (list Messages
                                  (object Message "Login"
                                        quid             "5852B61201E6"
```

```
                frequency       "Aperiodic"
                synchronization         "Simple"
                dir             "FromClientToSupplier"
                sequence        "1"
                ordinal         0
                quidu           "000000000000"
                creation        FALSE)
            (object Message "Patient Details"
                quid            "5852B638011C"
                frequency       "Aperiodic"
                synchronization         "Simple"
                dir             "FromClientToSupplier"
                sequence        "4"
                ordinal         3
                quidu           "000000000000"
                creation        FALSE)
            (object Message "Display Result"
                quid            "5852B66601DC"
                frequency       "Aperiodic"
                synchronization         "Return"
                dir             "ToClientFromSupplier"
                sequence        "9"
                ordinal         8
                quidu           "000000000000"
                creation        FALSE))))
        persistence     "Transient"
        creationObj     FALSE
        multi           FALSE)
    (object Object "User Interface"
        quid            "5852B5CF02EC"
        collaborators   (list link_list
          (object Link
              quid              "5852B62000B6"
              supplier          "Diagnosis System"
              quidu             "5852B5DB0385"
              messages          (list Messages
                (object Message "Verify"
                    quid            "5852B62000B7"
                    frequency       "Aperiodic"
                    synchronization         "Simple"
                    dir             "FromClientToSupplier"
                    sequence        "2"
                    ordinal         1
                    quidu           "000000000000"
                    creation        FALSE)
                (object Message "Result"
                    quid            "5852B6290327"
                    frequency       "Aperiodic"
```

```
                    synchronization     "Return"
                    dir                 "ToClientFromSupplier"
                    sequence            "3"
                    ordinal             2
                    quidu               "000000000000"
                    creation            FALSE)))
            (object Link
                quid            "5852B63F0278"
                supplier        "Doctor"
                quidu           "5852B5D40120"
                messages        (list Messages
                  (object Message "Request"
                    quid            "5852B63F0279"
                    frequency       "Aperiodic"
                    synchronization     "Simple"
                    dir             "FromClientToSupplier"
                    sequence        "5"
                    ordinal         4
                    quidu           "000000000000"
                    creation        FALSE)
                  (object Message "Diagnosis/Suggestion"
                    quid            "5852B65D03BE"
                    frequency       "Aperiodic"
                    synchronization     "Return"
                    dir             "ToClientFromSupplier"
                    sequence        "8"
                    ordinal         7
                    quidu           "000000000000"
                    creation        FALSE))))
            persistence     "Transient"
            creationObj     FALSE
            multi           FALSE)
        (object Object "Doctor"
            quid            "5852B5D40120"
            collaborators   (list link_list
              (object Link
                quid            "5852B64E01D3"
                supplier        "Diagnosis System"
                quidu           "5852B5DB0385"
                messages        (list Messages
                  (object Message "Refer Patient History"
                    quid            "5852B64E01D4"
                    frequency       "Aperiodic"
                    synchronization     "Simple"
                    dir             "FromClientToSupplier"
                    sequence        "6"
                    ordinal         5
                    quidu           "000000000000"
```

```
                creation        FALSE)
            (object Message "Retrive Data"
                quid            "5852B65402F2"
                frequency       "Aperiodic"
                synchronization     "Return"
                dir             "ToClientFromSupplier"
                sequence        "7"
                ordinal         6
                quidu           "000000000000"
                creation        FALSE))))
        persistence     "Transient"
        creationObj     FALSE
        multi           FALSE)
    (object Object "Diagnosis System"
        quid            "5852B5DB0385"
        persistence     "Transient"
        creationObj     FALSE
        multi           FALSE))))
```

2. Hasil ekstraksi *file* XML dari Sistem Konsultasi Medis

```
<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>
</symbol>
<symbol id='7'>
<objectmessage> "Diagnosis/Suggestion"</objectmessage>
<synchronization>          "Return"</synchronization>
<sequence>          "8"</sequence>
<ordinal>          7</ordinal>
</symbol>
<symbol id='8'>
<objectmessage> "Refer Patient History"</objectmessage>
<synchronization>          "Simple"</synchronization>
<sequence>          "6"</sequence>
<ordinal>          5</ordinal>
</symbol>
<symbol id='9'>
<objectmessage> "Retrive Data"</objectmessage>
<synchronization>          "Return"</synchronization>
<sequence>          "7"</sequence>
<ordinal>          6</ordinal>
</symbol>
<symbol id='10'>
<objectmessage> End</objectmessage>
<synchronization>          Simple</synchronization>
<sequence>          "10"</sequence>
<ordinal>          3,4,5,9</ordinal>
</symbol>
    </root>
```