PEMBANGKIT *TEST CASE* (KASUS UJI) MENGGUNAKAN MODEL *UML* (*UNIFIED MODELING LANGUAGE*) *SEQUENCE DIAGRAM* (STUDI KASUS *SISTEM PENILAIAN PEMBELAJARAN*)

SKRIPSI

Oleh:

ARYA ARIYANI
NIM. 12650024

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

2017

PEMBANGKIT *TEST CASE* (KASUS UJI) MENGGUNAKAN MODEL *UML* (*UNIFIED MODELING LANGUAGE*) *SEQUENCE DIAGRAM* (STUDI KASUS *SISTEM PENILAIAN PEMBELAJARAN*)

SKRIPSI

Oleh:

ARYA ARIYANI
NIM. 12650024

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

2017

PEMBANGKIT *TEST CASE* (KASUS UJI) MENGGUNAKAN MODEL *UML* (*UNIFIED MODELING LANGUAGE*) *SEQUENCE DIAGRAM* (STUDI KASUS *SISTEM PENILAIAN PEMBELAJARAN*)

SKRIPSI

Diajukan kepada :
Dekan Fakultas Sains dan Teknologi
Universitas Islam Negeri (UIN) Maulana Malik Ibrahim Malang
untuk memenuhi salah satu persyaratan dalam Memperoleh Gelar Sarjana Komputer (S.Kom)

OLEH:

ARYA ARIYANI
NIM. 12650024

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERIMAULANA MALIK IBRAHIM MALANG
2017

HALAMAN PERSETUJUAN

PEMBANGKIT *TEST CASE* (KASUS UJI) MENGGUNAKAN MODEL *UML* (*UNIFIED MODELING LANGUAGE*) *SEQUENCE DIAGRAM* (STUDI KASUS *SISTEM PENILAIAN PEMBELAJARAN*)

SKRIPSI

Oleh :
ARYA ARIYANI
NIM. 12650024

Telah disetujui oleh :

Pembimbing I	Pembimbing II
Fatchurrochman, M. Kom	**Roro Inda Melani, M.T., M. Sc**
NIP. 19700731 200501 1 002	NIP. 19780925 200501 2 008

Tanggal, 20Oktober 2017

Mengetahui:
Ketua Jurusan Teknik Informatika

Dr. Cahyo Crysdian
NIP. 19740424 200901 1 008

HALAMAN PENGESAHAN

PEMBANGKIT *TEST CASE* (KASUS UJI) MENGGUNAKAN *MODEL UML (UNIFIED MODELING LANGUAGE) SEQUENCE DIAGRAM* (STUDI KASUS *SISTEM PENILAIAN PEMBELAJARAN*)

SKRIPSI

Oleh :

ARYA ARIYANI
NIM. 12650024

Telah Dipertahankan di Depan Dewan Penguji Skripsi dan Dinyatakan Diterima sebagai Salah Satu Persyaratan untuk Memperoleh Gelar Sarjana Komputer Strata Satu (S.Kom)

Tanggal, 30 Oktober 2017

Susunan Dewan Penguji :		**Tanda Tangan**
1. **Penguji Utama**	**: A'la Syauqi, M. Kom** **NIP. 19771201 200801 1 007**	( )
2. **Ketua Penguji**	**:Supriyono, M. Kom** **NIP. 19841010 20160801 1 078**	( )
3. **Sekretaris**	**:Fatchurrochman, M. Kom** **NIP. 19700731 200501 1 002**	( )
4. **Anggota Penguji**	**:Roro Inda Melani, M. T., M. Sc** **NIP. 19780925 200501 2 008**	( )

Mengetahui dan Mengesahkan
Ketua Jurusan Teknik Informatika

Dr. Cahyo Crysdian
NIP. 19740424 200901 1 008

SURAT PERNYATAAN ORISINALITAS PENELITIAN

Saya yang bertanda tangan di bawah ini:

Nama : Arya Ariyani

NIM : 12650024

Fakultas / Jurusan: Sains dan Teknologi / Teknik Informatika

Judul Penelitian : Pembangkit *Test case*(Kasus Uji) Menggunakan Model *UML* (*Unififed Modeling Language*) *Sequence diagram* (Studi Kasus *Sistem Penilaian Pembelajaran*)

Menyatakan dengan sebenarnya bahwa dalam hasil penelitian saya ini tidak terdapat unsur-unsur penjiplakan karya penelitian atau karya ilmiah yang pernah dilakukan atau dibuat oleh orang lain, kecuali yang secara tertulis dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila di kemudian hari ternyata hasil penelitian ini terbukti terdapat unsur-unsur penjiplakan dan ada klaim dari pihak lain, maka saya bersedia untuk diproses sesuai peraturan perundang-undangan yang berlaku.

Demikian surat pernyataan ini saya buat dengan sebenarnya dan tanpa paksaan dari siapapun.

Malang, 20 Oktober 2017

Hormat saya

Arya Ariyani

NIM. 12650024

HALAMAN MOTTO

MOTTO:

Apalah arti diri, apalah arti waktu, pikiran, tenaga, jiwa dan harta jika hanya memikirkan diri sendiri. Apakah orang yang hanya memikirkan diri sendiri pantas untuk meraih surgaNya?

خَيْرُ النَّاسِ أَنْفَعُهُمْ لِلنَّاسِ

"Sebaik-baik manusia adalah manusia yang bermanfaat untuk manusia (lainnya)"

HALAMAN PERSEMBAHAN:

Dengan mengucap segenap rasa syukur Alhamdulillah kepada Alloh, Skripsi ini ku persembahkan untuk kedua Orangtuaku (Bapak Abdur Rohim dan Ibu Musrifatum), Kakak-adikku(Arya Zkhrifah dan Arya Al-Mahdi), Keluargaku(Mbah Mustiyah dll), Sahabat-sahabatku, Teman-temanku, Dosen-dosenku, Almamaterku, dan semua orang yang ku sayang, yang telah memberiku support yang tak ternilai jumlah dan harganya…

Terimakasih banyak atas segalanya.

KATA PENGANTAR

Puji syukur penulis ucapkan pada Allah SWT. karena skripsi dengan judul **"PEMBANGKIT *TEST CASE* (KASUS UJI) MENGGUNAKAN MODEL *UML (UNIFIED MODELING LANGUAGE) SEQUENCE DIAGRAM* (STUDI KASUS *SISTEM PENILAIAN PEMBELAJARAN*)"**ini akhirnyadapat selesai. Banyak hambatan yang dihadapi penulis selama menyelesaikan skripsi ini, baik yang berasal dari diri penulis sendiri maupun dari luar.Namun berkat ridho Allah dan bimbingan serta dukungan dari banyak pihak, akhirnya skripsi ini dapat selesai dan bisa digunakan sebagai salah satu syarat kelulusan dalam menempuh studi di Jurusan Teknik Informatika Universitas Islam Negeri Maulana Malik Ibrahim Malang.

Terselesaikannya skripsi ini merupakan hal utama yang menjadi tanggungjawab yang harus diselesaikan oleh penulis, sehingga bantuan banyak pihak merupakan hal yang sangat berarti. Oleh karena itu ucapan terimakasih disampaikan kepada pihak-pihak berikut ini:

1. Orang tua tersayang, Bapak Abd. Rohim, S.Pd.I dan Ibu Musrifatun yang tidak pernah henti-hentinya memberikan dukungan baik berupa materi maupun nonmateri, perjuangan-perjuangan serta kesabaran mereka membuat penulis mampu menyelesaikan studi ini dengan baik tanpa suatu halangan yang berarti.
2. Bapak Prof. Dr. H. Mudjia Rahardjo, M.Si, selaku Rektor Universitas Islam Negeri Maulana Malik Ibrahim Malang.

3. Ibu Dr. Bayyinatul Muchtaromah, drh. M. Si, selaku dekan fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang.
4. Bapak Cahyo Crysdian, M. Cs, selaku ketua jurusan Teknik Informatika Universitas Islam Negeri Maulana Malik Ibrahim Malang.
5. Bapak H. Fatchurrohman, M. Kom dan Ibu Roro Inda Melani, M.T., M. Sc, selaku dosen pembimbing yang dengan tulus dan sabar memberikan motivasi dan arahan kepada penulis sehingga skripsi ini dapat terselesaikan.
6. Seluruh Dosen Fakultas Sains dan Teknologi beserta staff, atas ketulusannya dalam memberikan ilmunya.
7. Kakak tersayang Arya Zukhrifah, M. Pd.I dan adik tersayang Arya Al-Mahdi beserta Nenek, juga keluarga besar yang selalu memberikan dukungan dan nasehat-nasehatnya yang sangat bermanfaat bagi penulis.
8. Kakak-kakak Indri Nizma, M. Pd.I dan Sativa Isnani, M. Pd.I yang sering mengirimi makanan.
9. Teman kamarku Evi Zakiyah, teman bermainku Laili Dwi Oktavia, teman pertama masuk kuliah Ruri Nur Aini, teman yang paling *care* Ni'amah, teman nongkrong Finda Nur Arifah dan arek-arek gang Ampel yang telah mendampingi dan memberikan support baik secara langsung maupun tidak langsung.
10. Sahabat-sahabat seperjuangan angkatan 2012 khususnya kelas A Teknik Informatika atas kekompakan dan kerjasamanya dalam menimba ilmu di Universitas Islam Negeri Maulana Malik Ibrahim Malang.
11. Semua pihak yang tidak dapat penulis sebutkan satu persatu yang telah membantu dalam penyelesaian skripsi ini.

Penulis menyadari bahwa skripsi ini masih jauh dari sempurna. Oleh sebab itu, saran dan kritik yang membangun sangat diharapkan untuk penelitian lanjutan di masa mendatang.

Akhir kata, semoga skripsi ini bisa memberikan manfaat bagi pengembangan ilmu pengetahuan.

Malang, 20 Oktober 2017

Arya Ariyani

DAFTAR ISI

Halaman

DAFTAR TABEL

DAFTAR GAMBAR

ABSTRAK

Ariyani, Arya. 2017. *Pembangkit Test Case (Kasus Uji) Otomatis Menggunakan Model UML (Unified Modelling Language) Sequence Diagram (Studi Kasus Sistem Penilaian Pembelajaran).* Skripsi, Program Sarjana Teknik Informatika, Universitas Islam Negeri Maulana Malik Ibrahim Malang. Pembimbing: (I) Fatchurrohman, M. Kom., (II) Roro Inda Melani, M.T., M.Sc.

Kata Kunci: Pembangkitan *Test Case, Sequence Diagram* (UML), *Depth First Search* (DFS), *Path.*

Penelitian ini berupaya membangkitkan *test case* yang didasarkan pada model UML *sequence diagram*. Pembangkitan *test case* dilakukan dengan membuat *sequence diagram*, membangun *sequence dependency table* (SDT), membangun *sequence dependency graph* (SDG), dan mendapatkan *path* menggunakan metode *depth first search* (DFS). Uji coba dilakukan pada *medical consultational* system, *banking system, aircraft departure activity* dan desain aplikasi penilaian pembelajaran.Aplikasi penilaian pembelajaran ini diharapkan dapat digunakan untuk memberikan penilaian pada matakuliah di perguruan tinggi.Uji coba pada *medical consultational system* menunjukkan bahwa aplikasi yang dibangun sesuai dengan literatur rujukan.Hal ini juga terjadi pada persoalan *banking system* dan *aircraft departure activity*.Pada desain aplikasi penilaian pembelajaran, terdapat empat *sequence diagram* yang menghasilkan sembilan *path* yang dapat digunakan sebagai *test case*.

ABSTRACT

Ariyani, Arya. 2017.Test Case Generator Using UML Model (Unified Modeling Language) of Sequence Diagram (Case Study of Learning Assessment System). Thesis, Department of Informatics, Maulana Malik Ibrahim State Islamic University of Malang. Supervisor: (I) Fatchurrohman, M. Kom., (II) Roro Inda Melani, M.T., M.Sc.

Keywords: Test Case Generator, Sequence Diagram (UML), Depth First Search (DFS), Path.

The research attempted to generate test case based on the UML sequence diagram model. Test case generation was done by generating sequence diagram, creating sequence dependency table (SDT), creating sequence dependency graph (SDG), and getting path by using depth first search (DFS) method. Trials were conducted on the medical consultational system, banking system, aircraft departure activity and learning assessment application design. Theassessment applicationwas expected to be used to assess the courses at college. Trials on the medical consultational system showed that applications were according to the referral literature. This was occurred the banking system and aircraft departure activity problem also. In the design of learning appraisal applications, there were four sequence diagrams that produced nine paths that can be used as test case.

ملخص البحث

أرياني، آريا. 2017. مستحضر حالة الاختبار (*Test Case*) باستخدام نموذج لغة النمذجة الموحدة (*Unified Modelling Language*) فى مخطط تسلسل (*Sequence Diagram*) (دراسة حالة فى نظام تقييم التعلم). البحث الجامعى، شعبة الهندسة المعلوماتية ، جامعة الإسلامية الحكوميةمولانا مالك إبراهيم في مالانج. المشرف: فتح الرحمن، الماجستير، رورو إنداه ميلانى، الماجستيرة

الكلمات الرئيسية: باني حالة اختبار ، مخطط تسلسل (UML)، البحث الأولالعمق ***Depth First Search*** (**DFS**)، المسار.

استحضر هذا البحث حالة اختبار الذى يستند إلى نموذج مخطط تسلسل. اجريت حالة الاختبار من خلال جعلالمخطط التسلسل، بني جدول تبعية تسلسل *sequence dependency table* (SDT)، بني رسم البياني التبعية التسلسل *sequence dependency graph* (SDG)، وحصل على مسار باستخدام الأسلوبالبحث الأول العمق(DFS). وأجريت تجارب على نظام الاستشارات الطبية، والنظام المصرفي، وأنشطة المغادرة الطائرات وتصميم التطبيقات فى تقييم التعلم. واحتاج هذا التطبيق يمكن ان يعطائالقيم على المواضيع فى الجامعة. وظهر التجارب على النظام الاستشارات الطبية أن التطبيقات المبنية هي وفقا لأدبيات المراجع. اي هذا فى النظام المصرفي وأنشطة المغادرة الطائرات. في تصميم تطبيقات تقييم التعلم، هناك أربعة مخططات التسلسل التي تنتج تسعة مسارات التي تمكن ان تستخدمهاكحالة الاختبار.

BAB I

PENDAHULUAN

A. Latar Belakang

Dalam pengembangan perangkat lunak, 50% waktu dan biaya digunakan untuk proses pengujian (Tripathy, 2012). Jika proses pengujian dapat dilakukan secara otomatis, maka efisiensi pengujian akan meningkat dan biaya pengembangan perangkat lunak dapat dikurangi. Oleh karena itu pengujian otomatis harus dirancang dengan baik agar dapat menemukan klasifikasi kesalahan secara sistematis dan dapat diperbaiki dalam waktu dan biaya yang minimal (Panthy, 2012).

Teknologi perangkat lunak berorientasi objek telah meningkat dengan cepat dalam hal perancangan dan pemrograman, hal ini ditunjang dengan adanya model UML yang terdiri dari delapanjenis diagram(*use case diagram*, *activity diagram*, *sequence diagram*, *collaboration diagram*, *class diagram*, *statechart diagram*, *deployment diagram*, dan *component diagram*) yang dapat digunakan untuk mendesain atau mendeskripsikan sebuah aplikasi dengan mudah. Tentunya untuk menjamin bahwa sebuah aplikasi berbasis objek telah terbebas dari *defect* perlu dilakukan pengujian. Pengujian adalah sebuah proses, atau serangkaian proses yang dirancang untuk memastikan bahwa program telah berjalan sesuai dengan *requirement*atau kebutuhan. Pengujian harus dilakukan oleh seorang *developer* pada sebuah aplikasi, sebelum aplikasi tersebut diberikan kepada *user*. Salah satu tahap yang penting dalam pengujian adalah pembangkitan *test case* sebuah aplikasi (Putri, 2015).

Penelitian tentang pembangkitan *test case* telah dilakukan oleh Shanti et al (2012) dan Priya et al (2013).Pembangkitkan *test case* dapat dilakukan dengan beberapa cara yaitu pembangkit *test case* berbasis skenario, pembangkit *test case* berbasis model dan pembangkit *test case* berbasis genetik (Khurana, 2014). Pembangkitan *test case* pada aplikasi berbasis objek dapat dilakukan pada tahap desain dengan memanfaatkan model UML.

Dalam kaitannyadengan Al-Qur'an, Allah juga menjelaskan dalam firman-Nya:

الَّذِي خَلَقَ الْمَوْتَ وَالْحَيَاةَ لِيَبْلُوَكُمْ أَيُّكُمْ أَحْسَنُ عَمَلًا ۚ وَهُوَ الْعَزِيزُ الْغَفُورُ

> Artinya:*"Yang menjadikan mati dan hidup, supaya Dia menguji kamu, siapa di antara kamu yang lebih baik amalnya. Dan Dia Maha Perkasa lagi Maha Pengampun". (QS. Al-Mulk 67:02).*

Tafsir Ibnu Katsir menjelaskan potongan ayat, (الَّذِي خَلَقَ الْمَوْتَ وَالْحَيَاةَ لِيَبْلُوَكُمْ) "*Yang menjadikan mati dan hidup.*" Ayat ini dijadikan oleh orang-orang yang berpendapat bahwa kematian adalah sesuatu yang wujud karena ia diciptakan (makhluk). Sedangkan makna ayat itu sendiri bahwa Allah telah mengadakan makhluk ini dari ketiadaan untuk menguji mereka, yakni untuk menguji siapakah di antara mereka yang paling baik amalnya. Sebagaimana yang difirmankan Allah Ta'ala, (كَيْفَ تَكْفُرُونَ بِاللَّهِ وَكُنْتُمْ أَمْوَاتًا فَأَحْيَاكُمْ) "*Mengapa kamu kafir kepada Allah, padahal kamu tadinya mati, lalu Allah menghidupkan kamu.*"(*Qs. Al-Baqarah 2:28*). Dengan demikian, keadaan pertama, yaitu ketiadaan sebagai maut (kematian). Sedangkan

penciptaan disebut sebagai hayat (kehidupan). Oleh karena itu, Allah Ta'ala berfirman, (ثُمَّ يُمِيتُكُمْ ثُمَّ يُحْيِيكُمْ) "*Kemudian Dia mematikanmu dan setelah itu menghidupkanmu kembali.*" (*Qs. Al-Baqarah 2:28*).

Firman Allah Ta'ala, (لِيَبْلُوَكُمْ أَيُّكُمْ أَحْسَنُ عَمَلًا) "*Supaya Dia mengujimu, supaya di antara kamu yang lebih baik amalnya.*" Yakni, yang paling baik amalnya, sebagaimana yang dikatakan oleh Muhammad bin Ajlan. Dan Allah tidak mengatakan "*Yang paling banyak amalnya.*"

Selanjutnya Dia berfirman, (وَهُوَ الْعَزِيزُ الْغَفُورُ) "*Dan Dia MahaPerkasa lagi MahaPengampun.*" Yakni, Dia MahaPerkasa lagi MahaAgung, MahaMenolak, lagi MahaMenghindari. Meskipun demikian, Dia MahaPengampun bagi orang-orang yang bertaubat dan kembali kepada-Nya setelah sebelumnya bermaksiat dan mendurhakai perintah-Nya. Meskipun Dia MahaTinggi lagi MahaMulia, namun demikian Dia tetap mau memberikan ampunan, kasih sayang, serta memberikan maaf (Abdullah, 2005).

Allah menguji hamba-Nya untuk mengetahui yang lebih baik amal di antara hamba-hamba-Nya. Sedangkan dalam penelitian ini, pengujian dimaksudkan sebagai sarana untuk mendapatkan *software* dengan kualitas yang baik.

Dalam penelitian ini, proses pengujian akan dilakukan pada beberapa sistem aplikasi berbasis objek yakni *medical consultational system*, *ATM System*atau *Banking System* dan *aircraft departure activity* dari *aircraft control system*. Dengan pengujian yang dilakukan pada ketiga sistem ini dapat diharapkan bisa mewakili pengujian pada sistem lainnya.

Kemudian akan dilakukan pengujian pada *sistem penilain pembelajaran* sebagai studi kasus dalam skripsi ini. Dengan memanfaatkan desain sistem UML *sequence diagram*diharapkan dapat mempersingkat proses pengujian sistem.Peneliti mengambil judul ***Pembangkit Test Case (Kasus Uji) Menggunakan Model UML (Unified Modeling Language) Sequence Diagram (Studi Kasus Sistem Penilaian Pembelajaran)*** sebagai penelitian yang akan dilakukan dalam skripsi ini.

B. Identifikasi Masalah

Setelah melihat latar belakang di atas, maka penelitian ini dapat terfakus pada :

1. Dapatkah membangkitkan *test case* dengan menggunakan model UML (*Unified Modeling Language*) *Sequence diagram*pada aplikasi berorientasi objek ?

C. Tujuan Penelitian

Berdasarkan identifikasi masalah, maka penelitian ini bertujuan untuk membangun perangkat lunak untuk membangkitkan *test case* dari model UML (*Unified Modeling Language*) *Sequence diagram.*

D. Batasan Penelitian

Batasan dalam penelitian ini adalah sebagai berikut:

1. Secara umum program ini dapat digunakan oleh pembuat dan pembangun *software*.
2. Secara khusus pembuatan *sequence diagram* dilakukan pada *software Rational Rose*.

3. *Sequence Diagram* pada artikel Shanti et al (2012), Priya et al (2013) dan Rhmann (2016) dijadikan referensi objek pengujian. Ditambah sistem penilaian pembelajaran sebagai studi kasus dalam skripsi ini.
4. Implementasi metode DFS (*Depth First Search*) hanya sampai pada pembentukan *Test path.*

E. Manfaat Penelitian

Penelitian ini diharapkan dapat mempersingkat waktu penyusunan *test case* dalam pengujian sistem perangkat lunak.

F. Sistematika Penulisan

Sistematika penulisan ditunjukkan untuk memberikan gambaran dan uraian dari penulisan skripsi ini secara garis besar yang meliputi beberapa, sebagai berikut:

BAB I : PENDAHULUAN

Pada bab ini menguraikan mengenai latar belakang, identifikasi masalah, tujuan, manfaat dan batasan penelitian dan sistematika penelitian.

BAB II : TINJAUAN PUSTAKA

Pada bab ini menguraikan tentang dasar teori dan referensi sebagai parameter rujukan untuk dilaksanakannya penelitian ini. Adapun teori yang digunakan adalah hasil penelitian yang relevan, tentang pembangkitan *test case* dalam pengujian perangkat lunak.

BAB III : ANALISIS DAN PERANCANGAN SISTEM

Pada bab ini membahas analisis kebutuhan dan perancangan sistem pembangkitan *test case* dengan model UML *sequence diagram* sebagai pengujian yang diterapkan dalam sistem *medical consultational system*, *ATM system* atau *Banking system* dan desain sistem penilaian pembelajaran dengan menerapkan metode DFS (*Depth First Search*).

BAB IV : HASIL DAN PEMBAHASAN

Pada bab ini memuat hasil pengujian pada pembangkitan *test case* terhadap perangkat lunak yang telah diselesaikan.

BAB V : PENUTUP

Pada bab ini memuat kesimpulan yang diperoleh dari hasil pembuatan dan pengujian pembangkitan *test case* perangkat lunak yang dikembangkan dalam skripsi ini serta saran saran untuk pengembangan lebih lanjut.

BAB II

TINJAUAN PUSTAKA

A. Tinjauan Pustaka

Bab ini berisi tinjauan pustaka. Yaitu kajian jurnal pendukung sebelumnya sehingga dapat diperoleh gambaran mengapa penelitian ini dilakukan. Juga berisi landasan teori yang membahas tentang pembangkit *test case* pada aplikasi berorientasi objek dengan menggunakan model UML (*Unified Modeling Language*) *sequence diagram*.

B. Penelitian Terkait

Hasil penelitian yang pernah dilakukan sehubungan dengan penelitian yang dilakukan dalam skripsi ini adalah :

1. Panthi et al (2012), Melakukan pengujian perangkat lunak untuk membangkitkan *test case* menggunakan model UML *Sequence diagram*. Dalam penelitiannya,pertama fitur-fitur diekstrak dari *sequence diagram* setelah itu melakukan penulisan *source code* java untuk fitur tersebut dengan menggunakan *library* model *JUnit*. Model *JUnit* merupakan *library* tambahan dari *JUnit library*. Dengan menggunakan *source code* tersebut, penelitian ini dapat secara otomatis membangkitkan *test case* dan *test coverage*. Penelitian ini menjelaskan teknik pembangkitan *test case* secara sistematis yang dilakukan pada model pengujian dasar atau *model based testing* (MBT) dengan menggunakan pendekatan *Sequence diagram*.

2. Shanti et al (2012), Melakukan penelitian tentang sebuah pendekatan baru yang digunakan untuk menguji perangkat lunak pada tahap awal atau *design*. Sehingga akan mudah bagi penguji perangkat lunak untuk menguji perangkat lunak tersebut pada tahap selanjutnya. Cara yang digunakan dalam penelitian ini yaitu pengujian otomatis dengan membuat *test case*. Sehingga penelitian ini berfokus pada cara membangkitkan *test case* darimodel UML *Sequence diagram* menggunakan Algoritma Genetika serta mencari *test case* terbaik atau prioritas *test case* dari beberapa *output test case* yang ada. Akhirnya, hasil penelitian menunjukkan bahwa metode ini memiliki kinerja yang baik.
3. Priya et al (2013), Melakukan penelitian tentang *test case* dalam proses pengujian perangkat lunak. Penelitian yang dilakukan menggunakan pendekatan model pengujian dasar atau *model based testing* (MBT) dari *test path* otomatis yang diperoleh sebelum atau selama proses pembuatan perangkat lunak berlangsung, ketika kode aplikasi tersedia, *test case* dapat dieksekusi sehingga dapat membantu dalam memperbaiki kesalahan di tahap awal pembuatan perangkat lunak. Model UML (*Unified Modeling Language*) *Sequence diagram* dari *medical consultational system* yang digunakan sebagai perancangan dan studi kasus dalam penelitian ini.

Artikel Shanti et al (2012) dan Priya et al (2013) di atas akan dijadikan rujukan dalam penelitian ini dan hasil pengujiannya akan

dicocokkan dengan penelitian karna sesuai dalam langkah-langkah penelitiannya.

C. Aplikasi Berorientasi Objek

Aplikasi berorientasi objek adalah sebuah aplikasi yang dibangun dengan pemrograman berorientasi objek. Pemrograman berorientasi objek (*object-oriented programming* disingkat OOP) merupakan paradigma pemrograman yang berorientasikan kepada objek. Semua data dan fungsi di dalam paradigma ini dibungkus dalam kelas-kelas atau objek-objek. Jika dibandingkan dengan logika pemrograman terstruktur, setiap objek dalam pemrograman berorientasi objek dapat menerima pesan, memproses data, dan mengirim pesan ke objek lainnya (Wikipedia, 2016).

1. UML *Diagram*

Unified Modeling Language (UML) merupakan sistem arsitektur yang bekerja dalam OOAD (*Object-Oriented Analysis*/*Design*) dengan satu bahasa yang konsisten untuk menentukan, memvisualisasi, mengkontruksi dan mendokumentasikan *artifact*(sepotong informasi yang digunakan atau dihasilkan dalam suatu proses rekayasa *software*, dapat berupa model, deskripsi, atau *software*) yang terdapat dalam sistem *software*. UML merupakan bahasa pemodelan yang paling sukses dari tiga metode *Object Oriented* yang telah ada sebelumnya, yaitu *Booch*, OMT(*Object Modeling Technique*), dan OOSE (*Object-Oriented Software Engineering*). UML merupakan kesatuan dari dari ketiga pemodelan tersebut dan ditambah kemampuan lebih karena mengandung metode

tambahan untuk mengatasi masalah pemodelan yang tidak dapat ditangani ketiga metode tersebut. UML dikeluarkan oleh OMG (*Object Management Group*, Inc)yaitu organisasi 11 internasional yang dibentuk pada 1989, terdiri dari perusahaan sistem informasi, *software developer*, dan para *user* sistem komputer. Dengan adanya UML, diharapkan dapat mengurangi kekacauan dalam bahasa pemodelan yang selama ini terjadi dalam lingkungan industri. UML diharapkan juga dapat menjawab masalah penotasian dan mekanisme tukar menukar model yang terjadi selama ini (Munawar, 2005).

Dalam pengertian lain, Unified Modeling Language (UML) adalah sebuah metode pemodelan secara visual sebagai sarana untuk merancang atau membuat sebuah perangkat lunak berorientasi objek. UML merupakan sebuah *tool* atau model untuk merancang pengembangan perangkat lunak yang berbasis objek. UML memberikan standar penulisan sebuah sistem *blue print*, yang meliputi konsep bisnis proses, penulisan kelas-kelas dalam bahasa pemrograman yang spesifik, skema database, dan komponen-komponen yang diperlukan dalam sistem sebuah perangkat lunak. UML menggunakan *class* dan *operation* sebagai konsep dasarnya, maka lebih cocok untuk penulisan perangkat lunak dalam bahasa berorientasi objek misalnya adalah C++, Java, C# atau VB.NET (Putri, 2015).

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.

2. *Sequence diagram*

Sequence diagram menggambarkan interaksi antara objek dalam urutan waktu. Umumnya sebuah *sequence diagram* menangkap *behavior* dari sebuah *scenario*. Diagram ini menunjukan sejumlah objek dan pesan yang dilewatkan antara objek-objek di dalam sebuah *scenario*. Terdapat beberapa simbol yang digunakan untuk membuat sebuah *sequence diagram* antara lain (Putri, 2015):

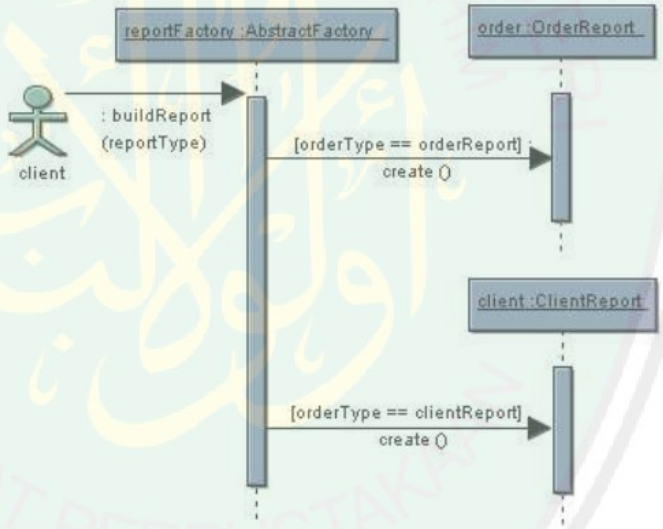
a. *Object*, merupakan *instance class* dan dituliskan tersusun secara horisontal.
b. *Found Message*, merupakan suatu pesan yang men-stimulus terjadinya skenario.
c. *Activation bar*, disebut juga dengan *focus of control* dalam setiap *lifeline* menunjukan kapan suatu *instance* aktif dalam interaksi, *activation bar* ini juga berhubungan dengan fungsi dari *instance* yang berada dalam *stack*.

d. *Lifeline*, merupakan jalur hidup suatu *instance* kelas tertentu dan berfungsi untuk mengetahui kapan suatu *instance* hidup dan dihapus, juga untuk melinierkan urutan pemanggilan pesan *instance* yang bersangkutan.

e. *Asynchrounous & synchronous message*, yaitu sebuah komunikasi antara peran.

f. *Interaction use*, merupakan referensi atau acuan untuk sebuah interaksi yang ada didalam definisi dari interaksi lain.

g. *Fragment*, merupakan alur dari sebuah *sequence diagram*. Berikut merupakan contoh dari *sequence diagram*.



Gambar 2.1. Contoh *Sequence diagram* (Timothy, 2002)

D. *SoftwareTesting*

Software testing merupakan aktivitas yang dilakukan untuk mengevaluasi dan meningkatkan kualitas suatu produk, dengan cara mengidetifikasi *bug* serta permasalahan yang terdapat pada produk tersebut (Utting, 2007). *Software testing* berperan dalam menentukan ukuran suatu *project*. Ukuran kualitas umumnya dibatasi oleh *correctness* (kebenaran),

completeness (kesempurnaan), *security* (keamanan). Namun, ukuran kualitas menurut ISO adalah *reliability, efficiency, portability, maintainability, compatibility, usability.*

E. Pengujian Jalur (*Testpath*)

Pengujian jalur atau *path testing* adalah strategi pengujian *structural* yang bertujuan untuk melatih setiap jalur eksekusi independen melalui komponen atau program. Jika setiap jalur independen dieksekusi, maka semua *statement* pada komponen harus dieksekusi paling tidak satu kali. Lebih jauh lagi, semua *statement* kondisional diuji untuk kasus *true* dan *false*. Pada proses pengembangan berorientasi objek, *test path* atau pengujian jalur dapat digunakan ketika menguji metode yang terkait dengan suatu program yang berorientasi objek ini.

Jumlah jalur yang dilalui program biasanya sebanding dengan ukuran program. Namun jika model diintegrasikan ke dalam sistem, pemakaian teknik pengujian *structural* menjadi tidak cocok. Teknik pengujian jalur dengan demikian paling cocok dipakai pada tahap pengujian unit dan pengujian modul pada proses pengujian terutama pengujian tahap awal sebelum program benar-benar jadi.

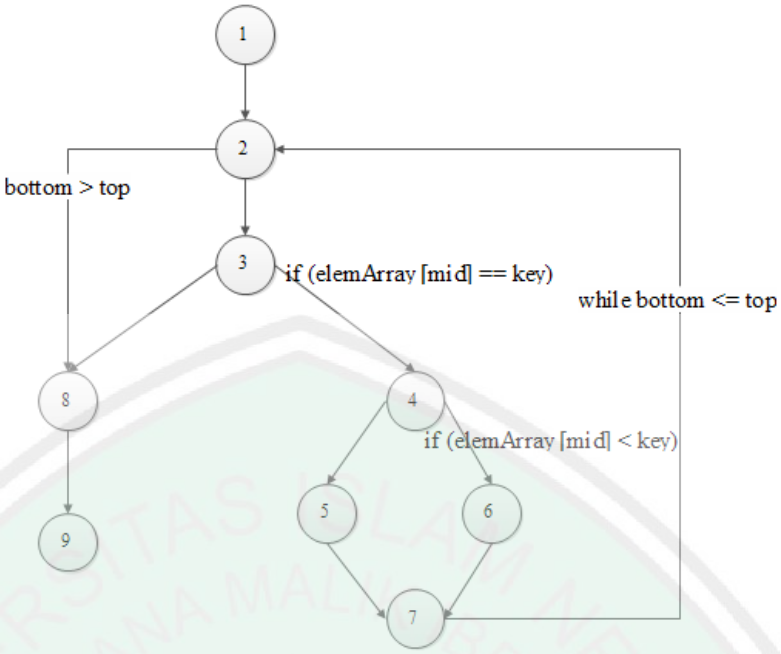
Dalam *test path* atau pengujian jalur, tidak menguji semua kombinasi jalur yang mungkin dilalui program. Untuk komponen yang tidak terdapat perulangan, maka tidak akan dieksekusi. Banyak kombinasi jalur yang akan dihasilkan pada program yang terdapat perulangan atau *loop*. Kekurangan atau kesalahan bisa terjadi ketika kombinasi jalur terbentuk bahkan ketika *statement* program telah dieksekusi paling tidak satu kali.

Titik pokok dalam *test path* atau pengujian jalur merupakan graf alir atau *flowgraph* suatu program. *Flow graph* ini merupakan kerangka model yang mewakili semua jalur (*path*) yang ada dalam program.*Flow graph* terdiri dari *node* yang mewakili keputusan dan *edge* yang menunjukkan aliran *control* dengan diagram yang ekuivalen. Jika tidak ada *statement goto* pada program, penurunan *flow graph* termasuk pada proses yang sederhana. *Statement* sekuensial (*assignment*, pemanggilan prosedur, dan *statement Input/Output*) dapat diabaikan pada alur *flow graph*. Setiap percabangan yang mewakili *statement* kondisional (*if-then-else* atau *case*) ditunjukkan sebagai jalur yang terpisah. Sedangkan *loop* atau percabangan ditunjukkan dengan tanda panah yang kembali ke *node* kondisi *loop*. *Loop* atau perulangan dan percabangan kondisional diilustrasikan pada *flow graph* dari *source code* untuk pengurutan *binary search* seperti di bawah (Sommerville, 2003).

```
class BinSearch {
   //ini merupakan enkapsulasi fungsi search biner yang mengambil array
   //sebagai objek terurut dan key dan mengembalikan objek dengan dua atribut
yaitu
   //index - nilai index array
   //found - yang menunjukkan apakah key ada dalam array atau tidak
   //objek dikembalikan karena tidak mungkin bagi java memberikan tipe dasar
   //dengan referensi ke suatu fungsi dan dengan demikian mengembalikan dua
nilai
   //key adalah -1 jika elemen tidak ditemukan

   public static void search(int key, int[] elemArray, Result r) {
      int bottom = 0;
      int top = elemArray.length - 1;
      int mid;
      r.found = false;
      r.index = -1;
      while (bottom <= top) {
         mid = (top + bottom) / 2;
         if (elemArray[mid] == key) {
            r.index = mid;
            r.found = true;
            return;
         } else {
            if (elemArray[mid] < key) {
               bottom = mid + 1;
            } else {
               top = mid - 1;
            }
         }
 }
 }
}
```

Tujuan pengujian *structural* adalah menjamin bahwa setiap jalur program yang independen dieksekusi paling tidak satu kali. Jalur program independen adalah jalur yang menelusuri paling tidak satu *edge* baru pada *flow graph* atau graf alir. Dalam istilah program, ini berarti melatih satu atau lebih kodisi baru. Percabangan *true* dan *false* harus dieksekusi untuk semua kondisi.



Gambar 2.2. *Flow Graph* untuk Pengurutan *Binary Search*

Flow graph untuk prosedur *binary search* sesuai *source code* *class BinSearch* ditunjukkan pada gambar 2.2. di atas dengan menelusuri aliran. Dengan demikian, kita lihat bahwa jalur independen *flow graph binary search* adalah sebagai berikut :

1, 2, 3, 8, 9
1, 2, 3, 4, 6, 7, 2
1, 2, 3, 4, 5, 7, 2
1, 2, 3, 4, 6, 7, 2, 8, 9

Jika semua jalur ini dieksekusi, kita dapat yakin bahwa :

1. Semua *statement* pada metode tersebut telah dieksekusi paling tidak satu kali.
2. Setiap percabangan telah dilatih untuk *true* dan *false*.

Jumlah jalur independen pada program dapat ditemukan dengan menghitung kompleksitas siklomatik (McCabe, 1976) dari graf alir program. Kompleksitas siklomatik (*cyclomatic complexity*). CC dari graf terhubung G dapat dihitung menurut rumus ini :

CC (G) = Jumlah (*edge*) – Jumlah (*node*) + 2

Untuk program tanpa *statement goto*, nilai kompleksitas siklomatik lebih satu dari jumlah kondisi pada program. Pada kondisi *compound* lebih dari satu kali uji, anda harus menghitung setiap uji. Dengan demikian, jika ada enam *statement if* dan satu *statement loop while*, dengan semua eksekusi kondisional sederhana, kompleksitas siklomatik adalah 8. Jika suatu eksekusi kondisional merupakan ekspresi *compound* dengan dua *operator* logika (‘*and*’ atau ‘*or*’) kompleksitas siklomatis adalah 10. Kompleksitas siklomatik pengurutan *binary search* pada gambar 2.2 di atas adalah 4.

Setelah menemukan jumlah jalur independen melalui *source code* dengan menghitung kompleksitas siklomatik, langkah berikut adalah merancang kasus uji untuk ekseskusi setiap jalur tersebut. jumlah minimum kasus uji yang dibutuhkan untuk menguji semua jalur program sama dengan kompleksitas siklomatik.

Desain kasus uji bersifat langsung pada kasus pengurutan *binary search*. Namun, ketika program memiliki struktur percabangan yang rumit, mungkin sulit untuk meramalkan bagaimana suatu kasus uji tersebut akan diproses. Sehingga dalam kasus ini, diperlukan penganalisis program secara dinamik untuk dapat melanjutkan pengujian pada program.

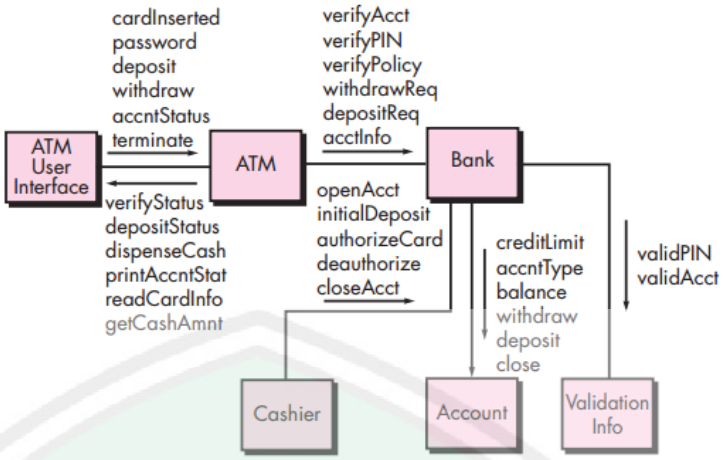
F. Kasus Uji (*Test case*)

Kasus uji *atau Test case* merupakan salah satu komponen dokumentasi pengujian. *Test case* digunakan sebagai panduan bagi *tester* untuk melukan pengujian suatu modul (Novelia, 2008).

Dalam pengertian lainnya, kasus uji atau *test case* adalah sebuah masukan, kondisi, dan ekspektasi hasil yang digunakan untuk menguji sebuah perangkat lunak atau aplikasi. Dengan menggunakan kasus uji atau *test case* seorang penguji dapat menemukan *defect* atau *bug* pada aplikasi sebelum aplikasi digunakan *user*. Kasus uji harus berisikan pengujian setiap menu sebuah aplikasi untuk mencegah adanya *defect*. Kasus uji atau *test case* yang baik adalah kasus uji atau *test case* yang terdiri dari beberapa unsur misalnya adalah menemukan kesalahan yang banyak, tidak menyalin kasus uji atau *test case* yang lain, dibuat untuk menemukan *error* atau *defect*, tidak terlalu sederhana atau terlalu kompleks, dan jelas untuk menguji ketika terjadi kesalahan pada aplikasi (Putri, 2015).

Suatu contoh, terdapat *diagram class class Bank* yang terelasi dengan *class ATM* :

Sebuah *test case* acak untuk *class Bank* akan seperti berikut :



Gambar 2.3. *Diagram Class Class Bank* yang Terelasi dengan *Class ATM* (Pressman, 2002)

Sehingga menghasilkan rangkaian operasi untuk *class Bank* yang terelasi dengan *class ATM*.:

verifyAcct*verifyPIN*[[verifyPolicy*withdrawReq] | depositReq | accInfoREQ].

Sebuah *test case* acak untuk *class Bank* akan seperti berikut:

Test case r1 = **verifyAcct*verifyPIN*depositReq**

Agar mempertimbangkan kolaborator yang terlibat dalam pengujian, message yang berhubungan dengan masing-masing operasi yang ditulis dalam test case r1 dipertimbangkan. Bank harus berkolaborasi dengan **ValidationInfo** untuk mengeksekusi verifyAcct() dan verifyPIN(). Bank harus berkolaborasi dengan **Account** untuk mengeksekusi depositReq(). Oleh karenanya, sebuah *test case* baru untuk memeriksa kolaborasi ini adalah :

Test case r2 = **verifyAcct [Bank: validationAcctValidationInfo]* verifyPIN [Bank: validPinValidationInfo]*depositReq [Bank: depositaccount].**

G. XML

eXtensible Markup Language (XML) dikembangkan pada tahun 1996 dan diakui oleh W3C. XML menggunakan elemen yang ditandai dengan *tag* pembuka (diawali dengan '<' dan diakhiri dengan '>'), *tag* penutup (diawali dengan '</' diakhiri dengan '>') dan atribut elemen (parameter yang dinyatakan dalam *tag* pembuka misalnya adalah <form name="isidata">). XML hanya digunakan untuk menyimpan informasi yang dikemas dengan *tag-tag* XML. Untuk mengirim, menerima, atau menampilkan informasi dari XML dibutuhkan sebuah perangkat lunak (Informatics, 2013/2014).

XMI (*XML Metadata Interchange*) adalah suatu fungsi yang diusulkan dari *Extensible Markup Language* (XML) yang dimaksudkan untuk menyediakan cara standar bagi *programmer* dan pengguna lain untuk bertukar informasi tentang metadata (singkatnya, informasi tentang terdiri dari apa saja satu set data tersebut dan bagaimana adanya data itu). Secara khusus, XMI dimaksudkan untuk membantu programer menggunakan *Unified Modeling Language* (UML) dengan bahasa yang berbeda dan sebagai alat pengembangan untuk bertukar model data antara satu dengan yang lainnya (Putri, 2014).

H. Hubungan *Testpath*dalam Membangun*Test case*

Metode uji coba berbasis jalur atau *test path* juga dapat diterapkan pada *source code* atau sumber program. Namun dalam penelitian ini pengujian jalur dilakukan dari rancangan UML *sequence diagram*dari rancangan program yang akan atau sudah dibangun. Pada bagian ini akan

menjelaskan hubungan dari *test path* sehingga akan berpengaruh pada hasil *test case* yang akan dihasilkan. Berikut langkah-langkah pembuatan*test path* hingga pada pembuatan *test case* (Pressman, 2002):

1. Menggambarkan *flow graph* atau diagram alir dari *source code*. Dalam hali ini *Source code* yang digunakan adalah contoh *source code* pada *class BinSearch* di atas. Sehingga menghasilkan *flow graph* atau diagram alir pada (Gambar 2.2. *Flow Graph* untuk Pengurutan *Binary Search*).
2. Menentukan *cyclomatic complexity* untuk *flow graph* yang telah dibuat.

   Dengan rumus :

   V (G) = E – N + 2

   Dengan :

   E= Jumlah *edge* pada grafik alir

   N= Jumlah *node* pada grafik alir.

   Sehingga dari *flow graph* yang ditunjukkan pada Gambar 2.2 di atas menghasilkan :

   V (G) = 11 – 9 + 2 = 4.
3. Menentukan *independent path* dari *flow graph* tersebut.

   Dari hasil perhitungan *cyclomatic complexity* terdapat 4 *independent path* yaitu :

   Path 1 : 1, 2, 3, 8, 9

   Path 2 : 1, 2, 3, 4, 6, 7, 2

   Path 3 : 1, 2, 3, 4, 5, 7, 2

   Path 4 : 1, 2, 3, 4, 6, 7, 2, 8, 9

4. Selanjutnya, membuat *test case* yang akan mengerjakan masing-masing *path* dalam satu baris set yang terdiri dari *path-path* yang sudah tersusun. Data yang dipilih harus tepat sehingga setiap kondisi dari *predicate node* dikerjakan semua.

I. Automation *Testing*

Automated testing merupakan proses pengujian yang digunakan untuk mempermudah proses dan dokumentasi pengujian, serta mengefektifkan proses pengeksekusian dan pengukuran pada pengujian (Novelia, 2008).

Galin dalam Rina, Uji terotomatisasi atau *automation testing* merupakan uji perangkat lunak yang dibantu dengan alat pengujian yang juga berbentuk perangkat lunak. Faktor-faktor yang mendukung berkembangnya perangkat lunak pengujian ini antara lain : penghematan biaya pengembangan, durasi pengujian yang dipersingkat,peningkatan kecermatan dalam pelaksanaan pengujian, peningkatan akurasi pengujian, dan peningkatan hasil pengujian seperti pada proses statistik (Rina, 2009).

Keuntungan pengujian otomatis :

a. Meningkatkan produktivitas
b. Menghemat uang
c. Meningkatkan kualitas perangkat lunak
d. Mengurangi waktu pengujian
e. Mendukung berbagai aplikasi
f. Meningkatkan cakupan pengujian
g. Pengurangan pekerjaan berulang-ulang

h. konsistensi yang lebih besar.

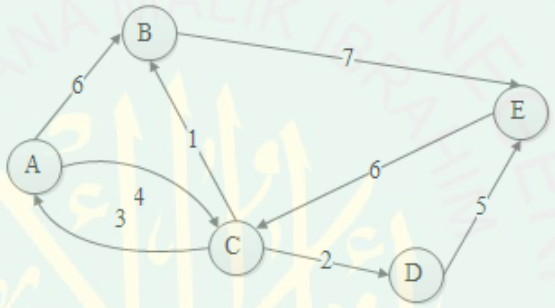
J. *Graph*

Dalam bidang matematika dan ilmu komputer, *graph* adalah struktur yang menggambarkan relasi antar obyek dari sebuah koleksi obyek. Jika struktur linear; misalnya *array*; memungkinkan pendefinisian keterhubungan sekuensial antara entitas data, struktur data *tree* memungkinkan pendefinisian keterhubungan hirarkis, maka struktur *graph* memungkinkan pendefinisian keterhubungan tak terbatas antara entitas data.

Banyak obyek dalam masalah-masalah nyata secara alamiah memiliki keterhubungan langsung secara tak terbatas. Contohnya informasi topologi dan jarak antar kota-kota di suatu pulau. Dalam masalah ini kota x bisa berhubungan langsung dengan hanya satu atau lima kota lainnya. Untuk memeriksa keterhubungan dan jarak tidak langsung antara dua kota dapat diperoleh berdasarkan data keterhubungan langsung dari kota-kota lainnya yang memperantarainya. Contoh lain penggunaan *graph* adalah untuk menggambarkan jaringan dan jalur kereta api, lintasan pesawat, sistem permipaan, saluran telepon, koneksi elektrik, ketergantungan diantara task pada sistem manufaktur dan lain-lain. Terdapat banyak hasil dan struktur penting yang didapatkan dari perhitungan dengan *graph*.

Representasi data dengan struktur data linear ataupun hirarkis pada masalah ini bisa digunakan namun membutuhkan operasi-operasi yang rumit sehingga kurang efisien. Struktur data *graph* secara eksplisit menyatakan keterhubungan ini sehingga pencariannya langsung dilakukan pada strukturnya sendiri. Definisi dari suatu *graph* adalah himpunan

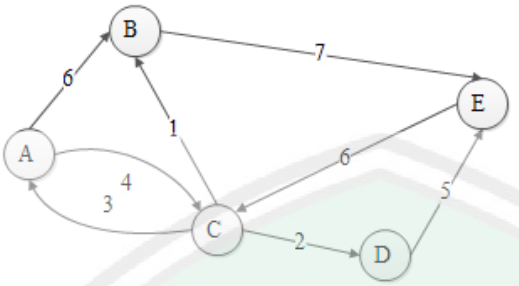
obyek-obyek yang disebut *node* (atau vertek) yang terhubung oleh *edge*. Biasanya *graph* digambarkan secara grafis sebagai kumpulan lingkaran yang melambangkan *node* yang dihubungkan oleh garis yang melambangkan *edge*. *Edge* dalam suatu *graph* bisa berupa *edge* berarah atau tidak berarah. Ilustrasi *graph* dapat dilihat pada Gambar 2.4. Pada gambar tersebut terlihat bahwa *graph* memiliki 5 buah *node*. Pada ilustrasi ini dimisalkan *node* mewakili sebuah kota. Maka dapat dilihat bahwa dari kota A menuju kota E bisa dilalui melalui *path* A-B-E atau *path* A-C-D-E.



Gambar 2.4. Ilustrasi *Graph* (Barakbah, 2013)

Untuk menyelesaikan permasalahan jalur terpendek (*sortest path*) dari *graph* pada Gambar 2.4. Maka harus dilakukan penerjemahan dari bentuk *graph* ke bentuk matrik keterhubungan langsung (*list adjancency*) yang dibuat dengan menggunakan *array* dua dimensi. Hasil pembuatan *list adjancency graph* dapat dilihat pada Gambar 2.5. Nilai-nilai yang tertera dalam list tersebut adalah jarak antara dua kota. Misalnya nilai 6 yang terletak pada posisi baris 1 kolom 2. Artinya adalah ada jalur dari kota A ke kota B dan jaraknya 6 satuan. Kebalikannya dari kota B ke kota A (baris 2 kolom 1) tidak ada jalur sehingga bernilai -. Untuk mendapatkan jalur terpendek dan nilai

jaraknya dapat diselesaikan dengan menggunakan beberapa algoritma sor*test path*, misalnya algoritma *Dijkstra* atau *Depth First Search*.



	A	B	C	D	E
A	0	6	4	-	-
B	-	0	-	-	7
C	3	1	0	2	-
D	-	-	-	0	5
E	-	-	6	-	0

Gambar 2.5. *List Adjacency* (Barakbah, 2013)

K. DFS (*Depth First Search*)

Algoritma *Depth First Search* atau DFS merupakan algoritma yang digunakan untuk pencarian dalam struktur data berupa pohon maupun *graph*. Pencarian dengan metode DFS merupakan pencarian yang dilakukan dengan menelusuri terlebih dahulu suatu pohon atau *graph* sedalam-dalamnya. Artinya algoritma ini akan melakukan pencarian sampai simpul paling dalam. Setelah tidak bisa dilanjutkan barulah algoritma DFS melakukan *backtrack* ke simpul-simpul tetangga (Kastogi, 2015).

Menurut Hafid Inggiantowi dalam Pribadi, DFS adalah pencarian yang berjalan dengan meluaskan anak akar pertama dari pohon pencarian yang dipilih dan berjalan dalam dan lebih dalam lagi sampai simpul tujuan ditemukan, atau sampai menemukan simpul yang tidak punya anak. Kemudian, pencarian *backtracking*, akan kembali ke simpul yang belum selesai ditelusuri.

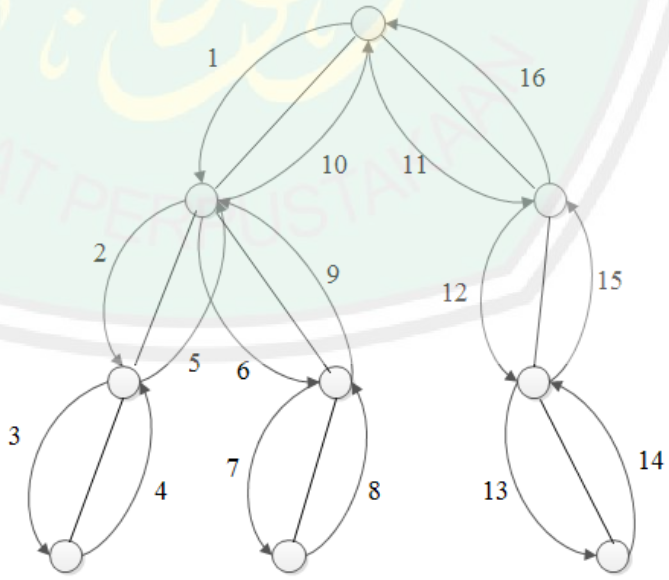
Berikut merupakan beberapa hal tentang DFS (*Depth First Search*) :

1. *Traversal* (penjalaran simpul) dari suatu graf G akan :
   a. Mengunjungi semua simpul dan sisi dari G
   b. Menentukan apakah G terhubung
   c. Memperhitungkan komponen terhubung dari G
   d. Memperhitungkan pohon merentang dari G
2. DFS pada graf dengan n simpul dan m sisi membutuhkan waktu :

O(n+m)

3. DFS dapat lebih jauh diperluas untuk menyelesaikan masalah-masalah pada graf seperti :
   a. Menemukan dan melaporkan lintasan antara dua simpul yang diberikan
   b. Menentukan sirkuit pada graf.
4. DFS untuk menggambar graf Euler ke pohon biner.

Berikut merupakan contoh *tree traversal* dari graf G menggunakan algoritma DFS :



Gambar 2.6. Contoh *Tree Traversal* G dengan DFS (Pribadi, 2015)

Pada gambar 2.6 terlihat bahwa DFS mengunjungi simpul atas (*ancestor*) dan akan terus menelusuri simpul di bawah-kirinya-nya (*descendant*) hingga simpul yang tidak lagi memiliki cabang (nomor 1, 2, dan 3) kemudian DFS akan melakukan *backtrack* ke simpul atas (nomor 4 dan 5) dan menelusuri simpul dibawah-kanan-nya hingga simpul paling bawah tidak memiliki cabang, kemudian DFS akan melakukan *backtrack* kembali ke simpul atas dan seterusnya hingga kembali lagi ke simpul paling atas. Salah satu cara dalam mengsimulasikan DFS adalah dengan menggunakan "*Stack*" yang memiliki sifat LIFO (*Last In First Out*).

BAB III

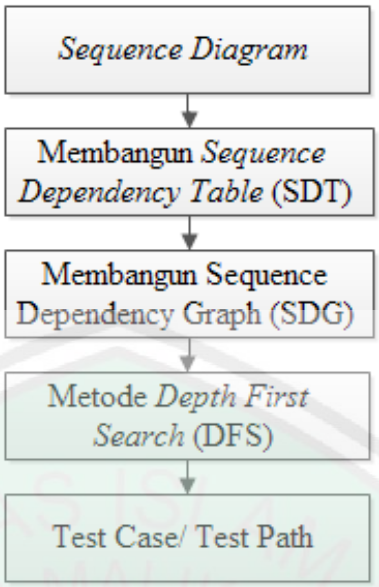
ANALISIS DAN PERANCANGAN SISTEM

A. Analisis Sistem

Pembangkitan *test case* dapat dilakukan dengan beberapa cara, sejauh ini pembangkitan *test case* pada sebuah aplikasi dilakukan pada saat tahap coding atau tahap pengujian. Pembangkitan *test case* pada aplikasi berbasis objek dapat dilakukan pada saat tahap *design* dengan memanfaatkan model UML. Dalam penelitian ini akan dibangun aplikasi pembangkit kasus uji (*Test case*) secara otomatis yang didasarkan dari model UML *sequence diagram*.

B. Perancangan Sistem

Bagian ini akan menjelaskan langkah-langkah dalam sistem yang akan dibangun. Adapun langkah-langkah dalam penyusunan sistem yang akan dibangun meliputi: (1) Pembuatan *Sequence diagram* dan eksport *Sequence diagram* dalam bentuk XML *File*, (2) Ekstrak Informasi dari *Sequence diagram* menjadi *Sequence Dependency Table*, (3) Membangun *Sequence Dependency Graph* dari *Sequence Dependency Table*, (4) Menerapkan *Metode Depth First Search* (DFS), dan (5) Membangun ***Test path*** (Jalur uji). Langkah Desain sistem ini dapat dilihat pada Gambar 3.1 di bawah ini:



Gambar 3.1.Desain Sistem

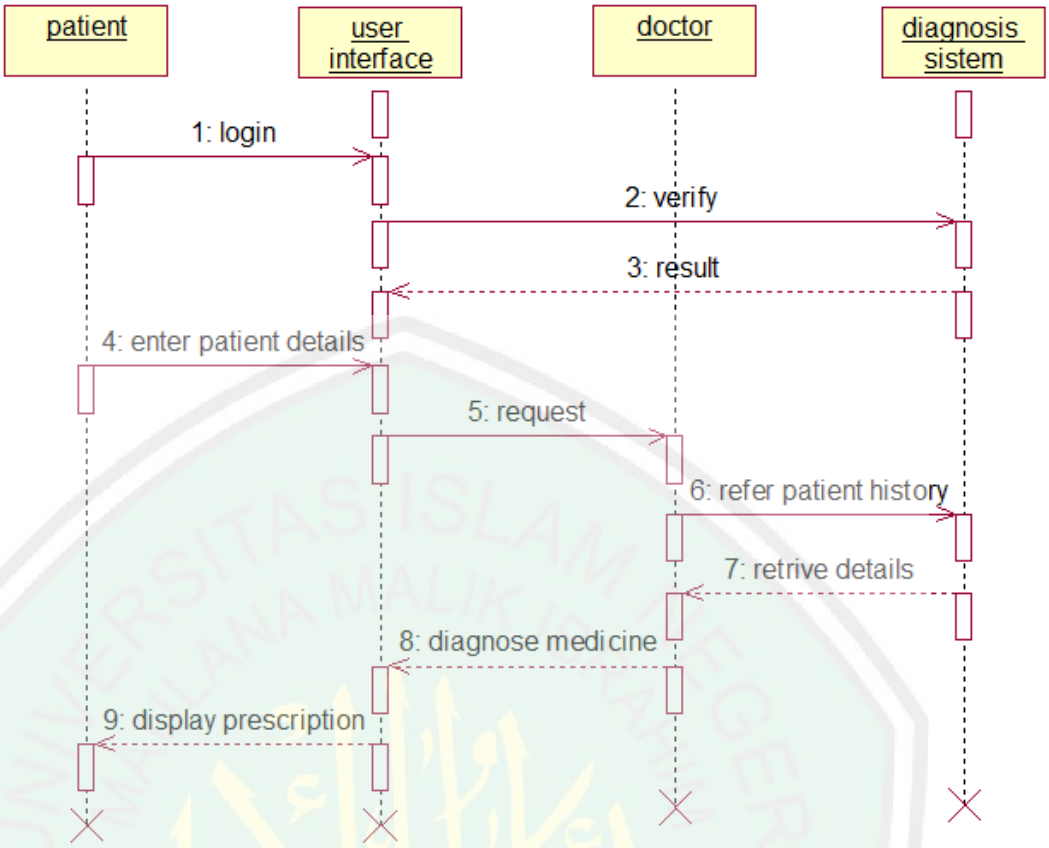
1. **Pembuatan *Sequence diagram***

Pemodelan berorientasi objek dalam penelitian ini menggunakan perangkat lunak *rational rose*. Dengan demikian *sequence diagram* disusun dengan *software* ini dan *output* dieksport dalam ekstensi .xml.

Sebuah kasus (Priya, 2013), *Medical consultational system* atau sistem konsultasi medis dapat dianggap sebagai layanan konsultasi komputerisasi yang dapat menjadi sistem informasi bimbingan antara pasien dengan dokter. Layanan ini menyediakan sarana bagi pasien untuk mendapatkan bantuan medis yang mereka butuhkan tanpa bertatap muka dengan dokter secara langsung. Sistem konsultasi medis yang diusulkan memungkinkan pasien dan dokter untuk berkomunikasi satu sama lain secara *online*. Ini adalah cara yang nyaman dan mudah bagi pasien untuk layanan konsultasi yang mendesak dan rutin yang bisa dilakukan melalui komputer. Dengan sistem ini, pasien dapat

mengakses layanan dari komputer manapun. Pasien harus memiliki akun pribadi dimana dia harus memberikan *username* dan *password* (jika dia adalah pengguna terdaftar). Jika berhasil *login*, pasien dapat melihat rincian informasi seperti nama pasien, usia, jenis kelamin, tanggal terakhir konsultasi, dokter kepada siapa mereka ingin berkonsultasi (bidang wajib), gejala mereka memiliki (bidang wajib). Jika dokter yang diminta sedang tersedia atau *online*, dokter dapat melihat riwayat pasien sebelumnya dari sistem diagnosis dan melihat gejala pada pasien saat ini sesuai keluhan yang dialami pasien. Jika rincian yang diberikan oleh pasien dirasa cukup untuk mendiagnosa masalah, dokter akan langsung merekomendasikan solusi kepada pasien. Namun jika tidak, dokter dapat menyarankan pasien untuk melakukan tes tambahan dan mengirimkan laporan hasil tes secara pribadi kepada dokter untuk konsultasi lebih lanjut.

Gambar 3.2 menunjukkan *sequence diagram* dari sistem kasus *Medical Consultational System* seperti yang sudah dijelaskan di atas. Di sini, angka mewakili urutan pesan yang dipertukarkan antara objek dalam komunikasi yang diminta. Vertikal garis putus-putus mewakili urutan peristiwa yang terjadi dalam sistem.



Gambar 3.2. *Sequence diagram* untuk *medical consultation system* (Priya, 2013)

a. *Login*: Memungkinkan pasien untuk login ke sistem dengan memberikan *username* dan *password* yang valid.

b. *Verify* : Verifikasi apakah *username* dan *password* yang dimasukkan valid dengan mencocokkan data pada database.

c. *Result* : Memberikan hasil untuk *user interface*. Jika *username* dan *password* yang dihasilkan cocok, sistem akan mengotentikasi pengguna dan memungkinkannya untuk melangkah lebih jauh. Jika tidak cocok, akses akan ditolak dengan menunjukkan sebagai pengguna tidak valid.

d. *Enter Patient Details* : Setelah berhasil *login*, pasien diharapkan untuk memasukkan data pribadi seperti nama pasien, usia, jenis kelamin, lalu tanggal konsultasi, kepada dokter siapa ingin berkonsultasi (bidang wajib), gejala yang mereka memiliki (bidang wajib).

e. *Request* : Sistem bekerja setelah mendapatkan rincian pasien kemudian menghubungkan pada dokter. Jika dokter tersedia, dokter dapat merespon pasien, atau pasien akan diberitahu jika dokter tidak tersedia.

f. *Refer Patient History* : Jika dokter tersedia, dokter dapat meminta sistem diagnosis untuk melihat riwayat pasien agar bisa mendiagnosis sesuai dengan gejala yang ada.

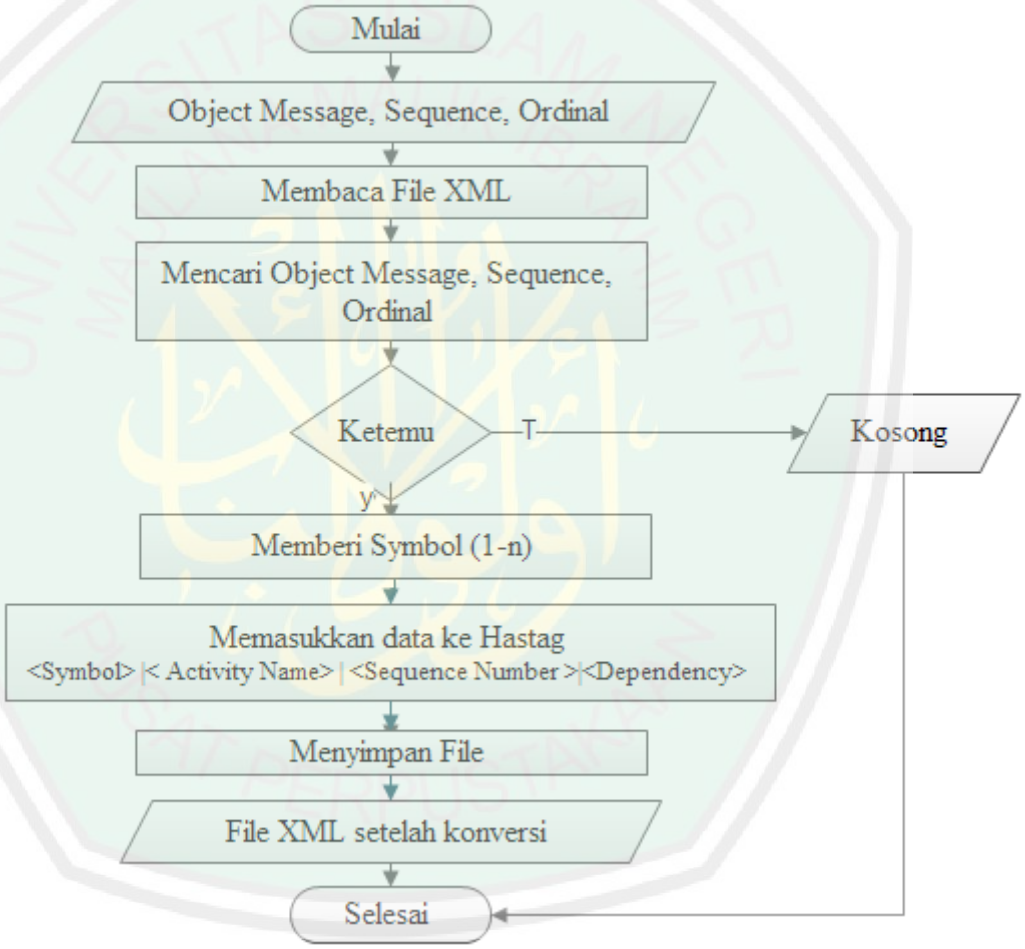
g. *Retrieve Details* : Informasi tentang pasien yang sudah diambil.

h. *Diagnose and Prescribe Medicine / Prescribe Additional Tests* : Dokter, setelah merujuk pada riwayat pasien dan dengan gejala yang sekarang, jika memungkinkan, dapat mendiagnosa tingkat keparahan penyakit dan bisa menyarankan obat-obatan. Lain jika gejala tidak cukup untuk mendiagnosa dokter bisa menyarankan tes tambahan yang harus dilakukan oleh pasien.

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

2. Membangun*Sequence Dependency Table* (SDT)

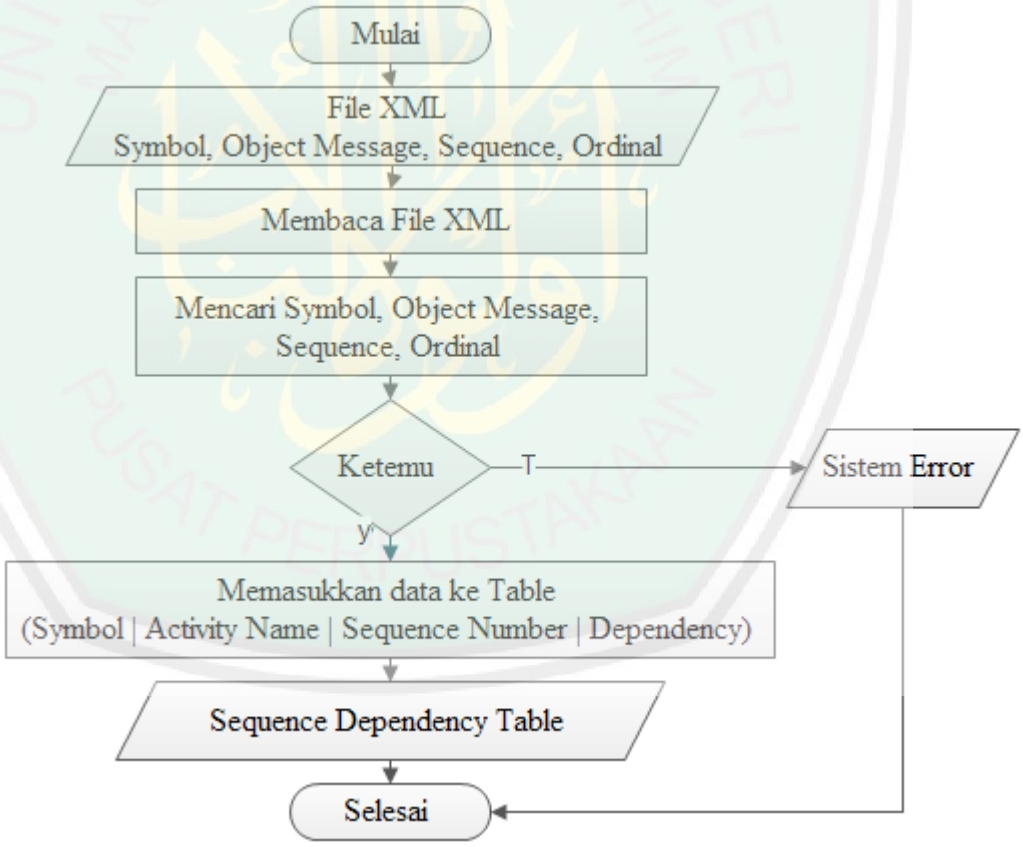
Tahapan pembuatan *Sequence Dependency Table* (SDT) ini, data XML diambil dan diekstrak dengan mengambil beberapa informasi terkait yang ada pada data XML. Informasi terkait terdiri dari *Symbol, Activity Name, Sequence Number, Dependency*. Berikut *flowchart* dalam pembangunan *Sequence Dependency Table* (SDT).



Gambar 3.3. *Flowchart* Konversi *File* XML *Rational Rose*

Flowchart di atas merupakan proses pengambilan informasi penting dari *file* XML (*output* dari *Rational Rose*) yang kemudian akan dibangun ulang sebuah *file* XML yang di dalamnya hanya berisi poin-poin penting yang dibutuhkan dalam pembangunan *Sequence*

Dependency Table (SDT). *Flowchart* di atas dimulai dari pengambilan *file* hasil eksport *Rational Rose*, kemudian dicari nilai-nilai variabel yang ada pada variabel *Object Message*, *Sequence* dan *Ordinal*, jika variabel ditemukan maka sistem akan menambahkan variabel *Symbol*, jika tidak ketemu maka *output* akan kosong. Setelah variabel ditemukan, variabel-variabel tersebut akan disimpan dalam Hastag <Symbol>, <ObjectMessage>, <Sequence>, <Ordinal> kemudian dapat disimpan dengan ekstensi XML sebagai input untuk membangun *Sequence Dependency Table* (SDT). Proses pembuatan SDT, akan dijelaskan pada *flowchart* Gambar 3.4 di bawah ini.



Gambar 3.4. *Flowchart* Pembuatan *Sequence Dependency Table*

Dari data XML setelah konversi seperti yang telah dijelaskan pada Gambar 3.3 di atas, data yang diambil dalam membangun *Sequence Dependency Table* (SDT) adalah data yang menyimpan variabel *Symbol, Object Message, Sequence* dan *Ordinal*. Setelah data ditemukan kemudian *Sequence Dependency Table* (SDT) akan dibentuk dengan format empat kolom (*Symbol*, *Activity Name*, *Sequence Number*, *Dependency*). Dimana kolom *Symbol* mewakili *symbol* yang diberikan, kolom *Activity Name* mewakili *Object Message*, kolom *Sequence Number* mewakili *Sequence* dan kolom *Dependency* mewakili *Ordinal* dari data-data yang sudah didapatkan dari ekstraksi data XML.

Dari *sequence diagram Medical Consultational System* yang sudah dibangun, kemudian dibangun *Sequence Dependency Table* (SDT) Table 1 menunjukkan *Sequence Dependency Table* (SDT) yang terdiri dari enam kolom berisi informasi sebagai berikut :

a. *Symbol* : Huruf abjad yang diberikan untuk setiap kegiatan yang terlibat.
b. *Activity Name* : Bagian ini menjelaskan kegiatan yang dilakukan.
c. *Sequence Number* : Memberi nomor urut yang memberikan jejak aliran yang terjadi ketika pesan dipertukarkan antara entitas yang terlibat.
d. *Dependency* : Simbol (s) urutan yang menunjukkan bahwa kegiatan saat ini memiliki ketergantungan atau hubungan pada *symbol* yang lain. Misalnya, kegiatan "C" tergantung pada Activity "A" yang

berarti hasil yang dikembalikan tergantung pada nama pengguna dan *password* yang valid disediakan oleh Pasien.

e. *Input* : *Input* memberikan prasyarat yang harus terjadi.

f. *Expected* : memberikan *output* yang diharapkan dari peristiwa saat ini yang terjadi.

Tabel 3. 1. *Sequence Dependency Table* (Priya, 2013)

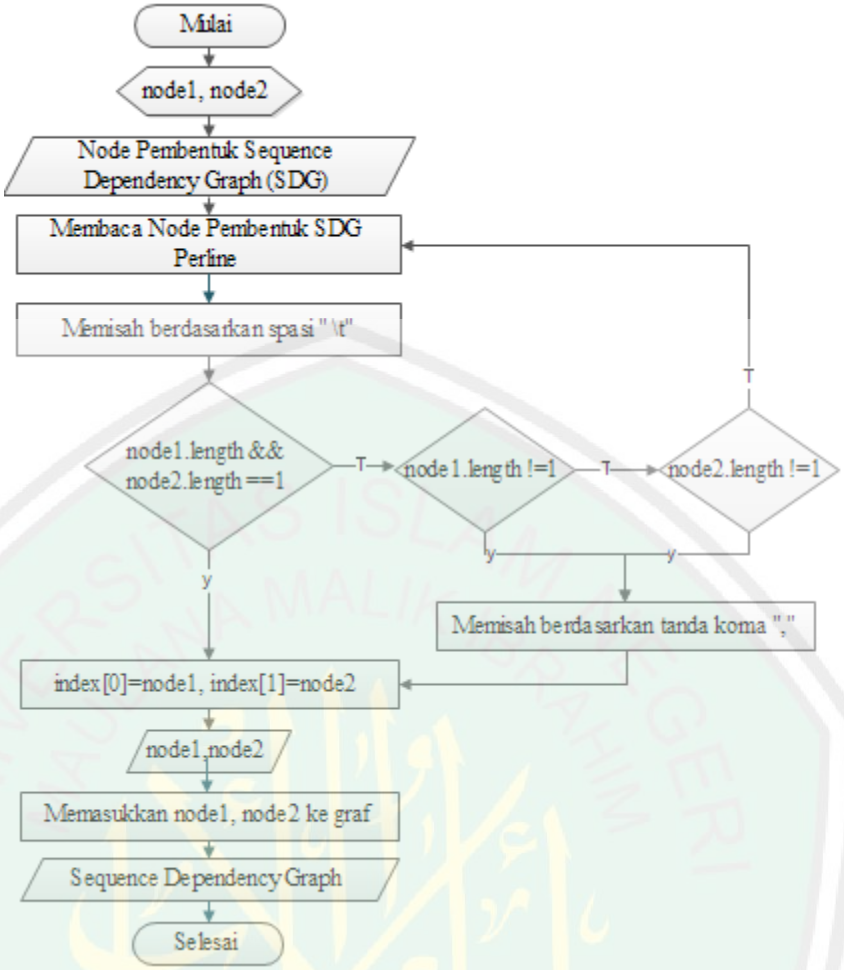
Simbol	Activity Name	Sequence Number	Dependency	*Input*	Expected *Output*
A	*Login*	1	-	Patient ID and *Password*	Valid Patient ID and *Password*
B	Verify	2	A	-	Validate Patient ID/ *Password*/ Invalid Patient ID/ *Password*
C	Result	3	B	-	Take to the next Screen on entering valid Patient ID and *Password* || End on entering invalid Patient ID/ *Password*
D	Patient details	4	C	Patients enter patient Name, Age, Gender, Last Consulted Date, Symptoms, Doctor Name	Checks for details (Valid) || Invalid details(End)
E	Request	5	D	-	Preeceds if doctor is available || End if Doctors is not available
F	Refer Patient History	6	E	-	Checks for details
G	Retrieve data	7	F	-	Retrieve and Display Patient

					history
H	Diagnosis/ Suggestion	8	G	-	Doctor make diagnosis and Prescribe medicine || Suggest to take some other *test* for further diagnosis
I	Display Result	9	H	-	Patient takes prescription/ Suggested Medical *Test*
J	End	-	C,D,E,I	-	-

Namun dalam penelitian ini, *Symbol* yang digunakan akan berbentuk angka. Dan tabel yang ditampilkan dalam program akan terdiri dari empat kolom sesuai *flowchart* pada Gambar 3.4 di atas.

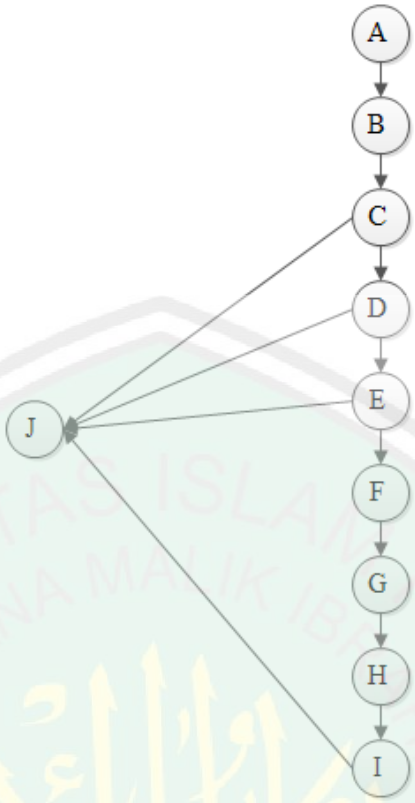
3. Membangun*Sequence Dependency Graph* (SDG)

Tahapan dalam membangun *Sequence Dependency Graph* (SDG) ini, proses dilakukan dengan mengambil kolom *Dependency* dan kolom *Sequence Number* pada hasil *Sequence Dependency Table* (SDT) yang kemudian dijadikan sebagai *node* dalam *graph* (*Sequence Dependency Graph*). Dimana kolom *Dependency* sebagai *node*1 (*node* awal) sedangkan kolom *Sequence Number* sebagai *node*2 (*node* tujuan). Berikut *flowchart* proses pembuatan *Sequence Dependency Graph* (SDG).



Gambar 3.5. *Flowchart* pembuatan *Sequence Dependency Graph*

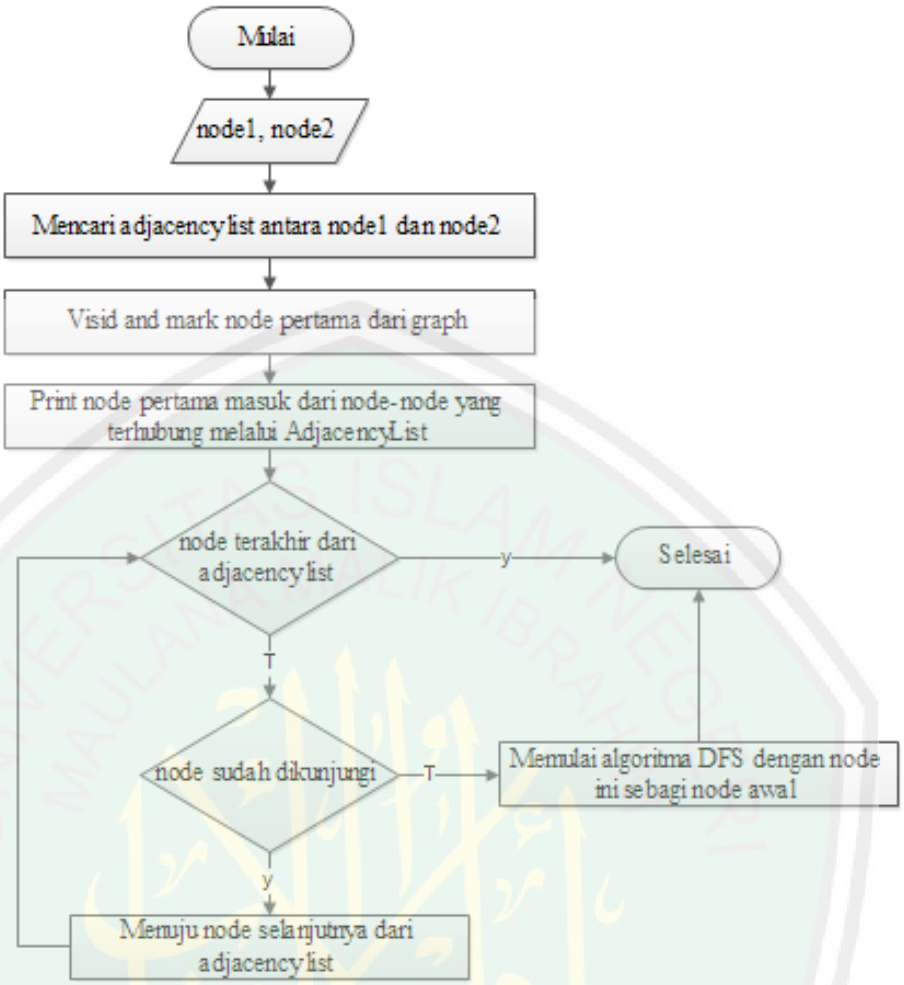
Gambar 3.6 menunjukkan *Sequence Dependency Graph* yang dihasilkan dari *Sequence Dependency Table* (SDT) dari *Medical Consultational System*.



Gambar 3.6. *Sequence Dependency Graph* (Priya, 2013)

4. Metode *Depth First Search* (DFS)

Metode *Depth First Search* (DFS) diterapkan dalam proses pencarian *node-node graph* yang saling terhubung. Untuk membentuk beberapa kemungkinan jalur yang akan dilalui dalam kasus pengujian (*Test path*). Berikut *flowchart* metode *Depth First Search* yang akan diterapkan dalam sistem yang akan dibangun.



Gambar 3.7. *Flowchart* metode *Depth First Search* dalam Sistem

5. *Test Case/ Path*

Test case yang akan dihasilkan dalam pengujian tahap desain dari *sequence diagram* ini akan berbentuk *path.*

Dari hasil graf pada Gambar 3.6 di atas, dengan menerapkan alur yang ada dalam *flowchart* metode *Depth First Search*. Sehingga dari *Sequence Dependency Graph Medical Consultational System* akan menghasilkan *test path*seperti berikut:

A→B→C→J
A→B→C→D→J
A→B→C→D→E→J
A→B→C→D→E→F→G→H→I→J

Gambar 3.8. *Test path* (Priya, 2013)

Test path di atas terdiri dari empat jalur *path* sebagai hasil *output* dari pengujian yang dilakukan pada *Medical Consultational System.*

BAB IV

HASIL DAN PEMBAHASAN

Dalam bab ini akan membahas tentang hasil uji dan pembahasan bagaimana proses pembuatan aplikasi, uji coba dilakukan untuk mengetahui apakah aplikasi dapat berjalan sesuai dengan yang diharapkan.

A. Hasil

1. Data Uji Coba

Ada empat data uji coba yang akan dilakukan dalam pengujian aplikasi yang sudah dibuat yakni, menggunakan data dari *Medical Consultational System*, data *ATM System*, data *Aircraft Departure Activity* dari *Aircraft Control System* dalam bentuk *sequence diagram* yang diambil dari artikel Shanti et al (2012), Priya et al (2013) dan Rhmann (2016) serta data sistem penilaian pembelajaranyang dijadikan studi kasus dalam penelitian ini. Berikut kasus-kasus dalam *medical consultational system*, *ATM System* (*Banking System*), *Aircraft Departure Activity* dan sistem penilaian pembelajaran.

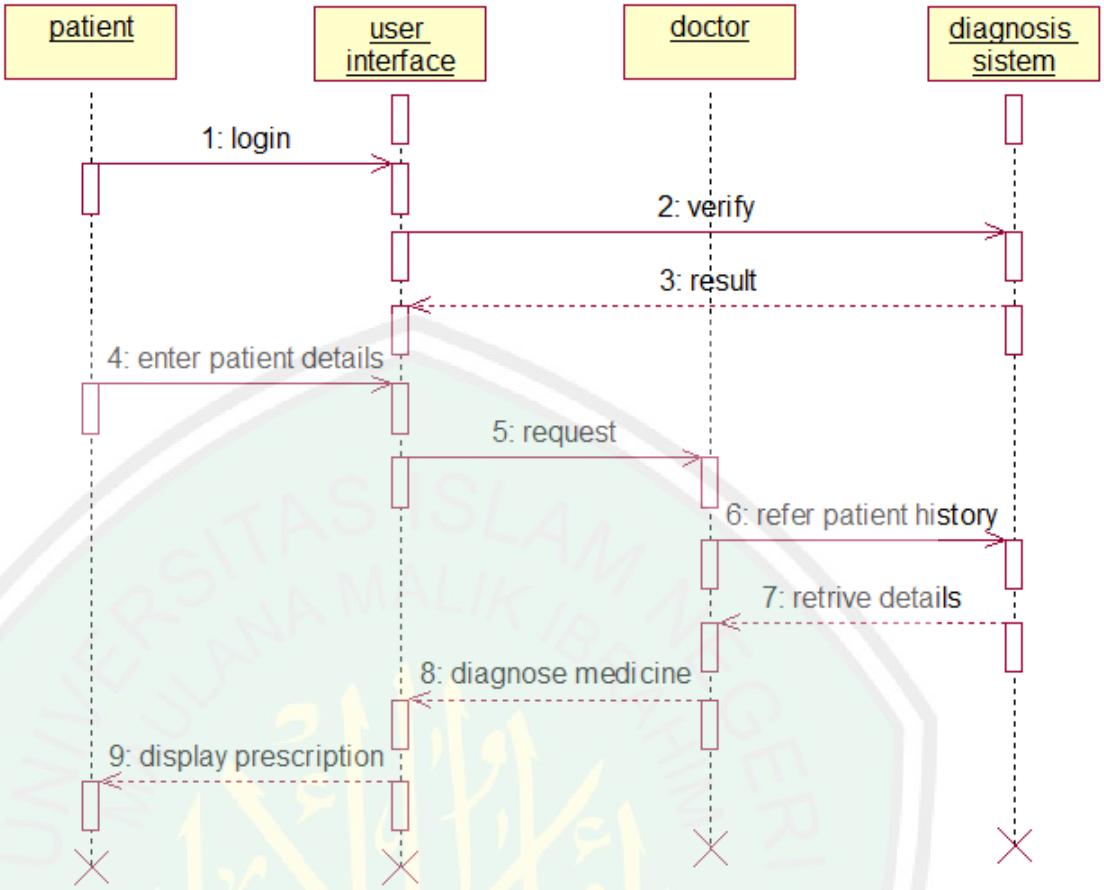
Sebuah kasus dalam ***medical consultational system*** atau sistem konsultasi medis dapat dianggap sebagai layanan konsultasi komputerisasi yang dapat menjadi sistem informasi bimbingan antara pasien dengan dokter. Layanan ini menyediakan sarana bagi pasien untuk mendapatkan bantuan medis yang mereka butuhkan tanpa bertatap muka dengan dokter secara langsung. Sistem konsultasi medis yang diusulkan memungkinkan pasien dan dokter untuk berkomunikasi satu sama lain secara *online*. Ini adalah cara yang nyaman dan mudah

bagi pasien untuk layanan konsultasi yang mendesak dan rutin yang bisa dilakukan melalui komputer. Dengan sistem ini, pasien dapat mengakses layanan dari komputer manapun. Pasien harus memiliki akun pribadi dimana dia harus memberikan *username* dan *password* (jika dia adalah pengguna terdaftar). Jika berhasil *login*, pasien dapat melihat rincian informasi seperti nama pasien, usia, jenis kelamin, tanggal terakhir konsultasi, dokter kepada siapa mereka ingin berkonsultasi (bidang wajib), gejala mereka memiliki (bidang wajib). Jika dokter yang diminta sedang tersedia atau *online*, dokter dapat melihat riwayat pasien sebelumnya dari sistem diagnosis dan melihat gejala pada pasien saat ini sesuai keluhan yang dialami pasien. Jika rincian yang diberikan oleh pasien dirasa cukup untuk mendiagnosa masalah, dokter akan langsung merekomendasikan solusi kepada pasien. Namun jika tidak, dokter dapat menyarankan pasien untuk melakukan tes tambahan dan mengirimkan laporan hasil tes secara pribadi kepada dokter untuk konsultasi lebih lanjut (Priya, 2013). Berikut Gambar 4.1 bentuk model *sequence diagram medical consultational system*.

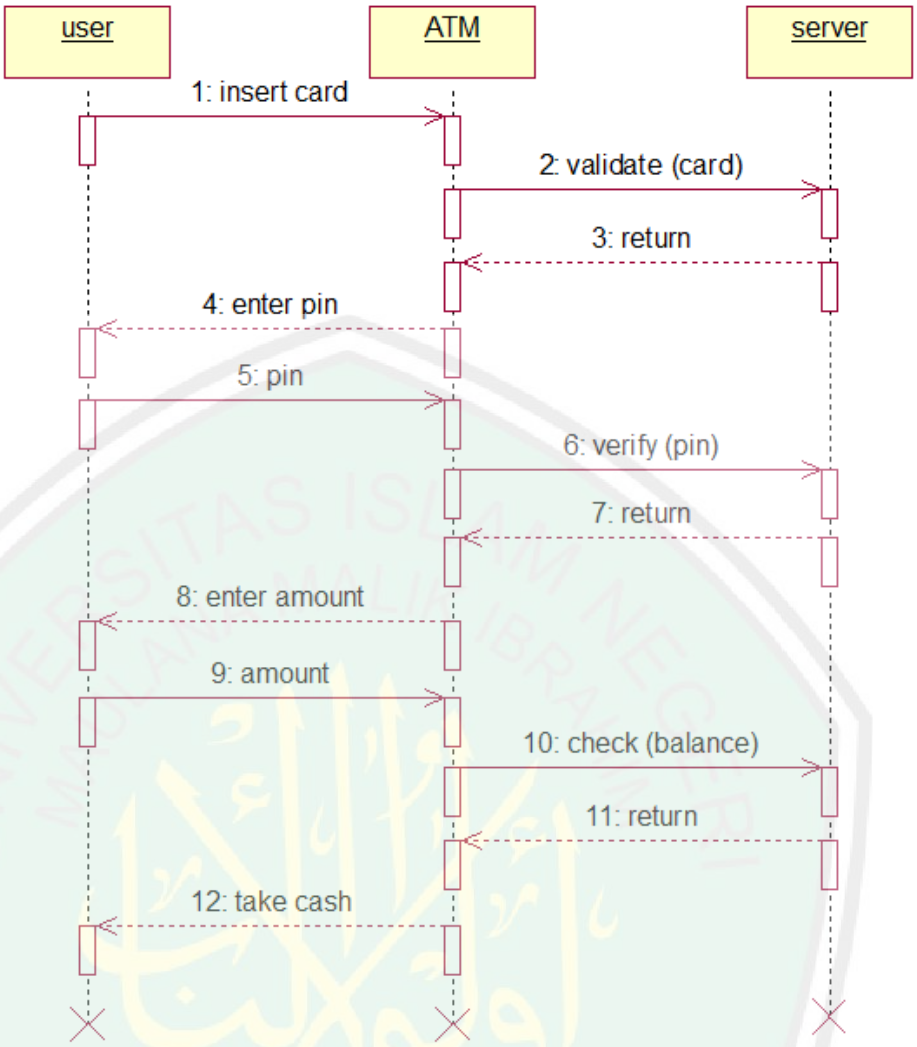
Dalam kasus lain pada ***ATM System* (*Banking System*)** (Shanthi, 2012), sebagai sarana untuk mempermudah *customer* *Bank* dalam pengambilan uang tabungan tanpa harus melalui layanan kantor *Bank*. Sehingga pihak *Bank* menyediakan sebuah sistem untuk mempermudah *customer* yaitu dengan membuat sistem ATM, hanya dengan memanfaatkan kartu ATM yang didapat saat pendaftaran

pembukaan rekening *Bank* dan hanya dengan memanfaatkan kode *pin*, *customer* bisa mengambil uang tabungannya dengan mudah. *Customer* akan memasukkan kartu ke dalam mesin ATM dan mesin akan memverikasi kartu tersebut pada *server*. Jika kartu sudah berhasil diverivikasi, *server* akan mengirimkan pada *user* untuk memasukkan nomor pin, yang kemudian akan diverifikasi lagi oleh *server*. Setelah terverifikasi, *user* bisa memasukkan jumlah uang yang akan diambil, kemudian mesin ATM akan memproses dan menghitung jumlah uang sesuai *input user*. *User* bisa mengambil uang pada mesin ATM. Berikut Gambar 4.2 bentuk model *sequence diagram* pada kasus *ATM System* atau *Banking System.*

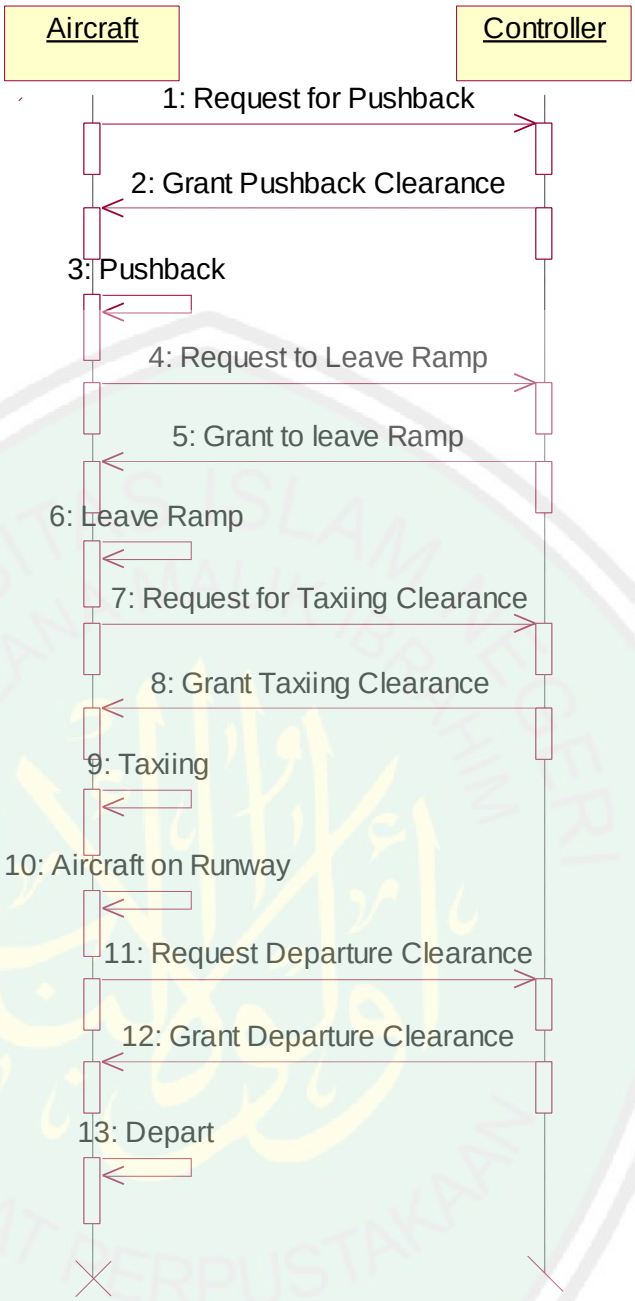
Kasus lain pada ***Aircraft Departure Activity*** (Rhmann, 2016), aktivitas pemberangkatan pesawat terbang dari sistem kontrol pesawat terbang. Sistem control pesawat yang mengkoordinasikan pesawat terbang untuk keberangkatan. Dengan banyaknya armada pesawat, perlu adanya koordinasi aktivitas pemberangkatan pesawat untuk menghindari tabrakan antar pesawat terbang. Berikut Gambar 4.3 bentuk model *sequence diagram* pada kasus *Aircraft Departure Activity*.



Gambar 4.1. Data Uji model UML*Sequence Diagram Medical Consultational System* (priya, 2013)



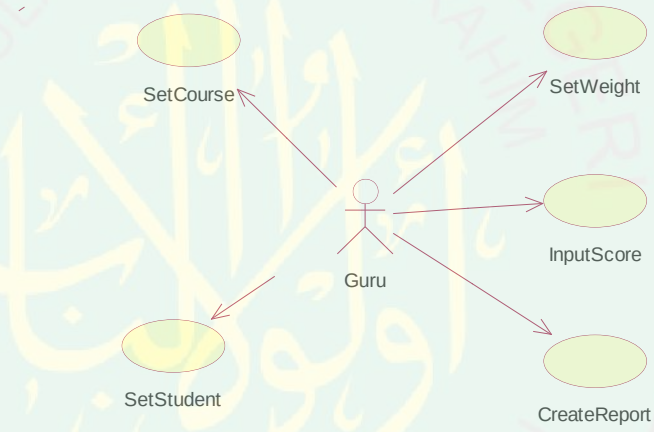
Gambar 4.2. Data Uji Model Sistem ATM (*Banking System*) (Shanthi, 2012).



Gambar 4.3. Data Uji Model *Aircraft Departure Activity* (Rhmann, 2016)

Data uji selanjutnya yaitu pada kasus sistem penilaian pembelajaran yang digunakan sebagai studi kasus dalam penelitian ini. Sistem Penilaian Pembelajaran merupakan *software* yang digunakan untuk membantu guru atau dosen dalam melakukan evaluasi pembelajaran yang dilakukan saat proses belajar mengajar.

Proses evaluasi dilakukan oleh guru dengan memasukan nilai tugas, ujian tengah semester, dan ujian akhir semester. Dengan demikian sistem evaluasi pembelajaran ini merupakan bagian dari sistem informasi akademik karena pada saat guru memasukkan nilai sistem evaluasi harus dapat menampilkan nama mahasiswa yang akan diberi nilai.Nilai-nilai ini perlu diolah berdasarkan bobot masing-masing komponen evaluasi pembelajaran untuk mendapatkan nilai akhir dan akan dikonversi dengan nilai abjad A, B+, B, C+, C, D dan E.Gambar 4.4 di bawah ini akan menunjukkan *use case* dari sistempenilaian pembelajaran.



Gambar 4.4. *Use Case Diagram* Sistem Penilaian Pembelajaran

Use case ***SetCourse*** digunakan untuk memilih matakuliah yang diajar dosen, *use case* ***SetStudent*** digunakan untuk memilih mahasiswa yang akan diberikan penilaian, *use case* ***SetWeigth*** digunakan untuk memberikan bobot pada berbagai komponen penilaian. *Use case* ***inputScore*** digunakan untuk memasukkan berbagai nilai pada komponen penilaian. *Use case* ***CreateReport*** digunakan untuk menghasilkan laporan yang diperlukan.

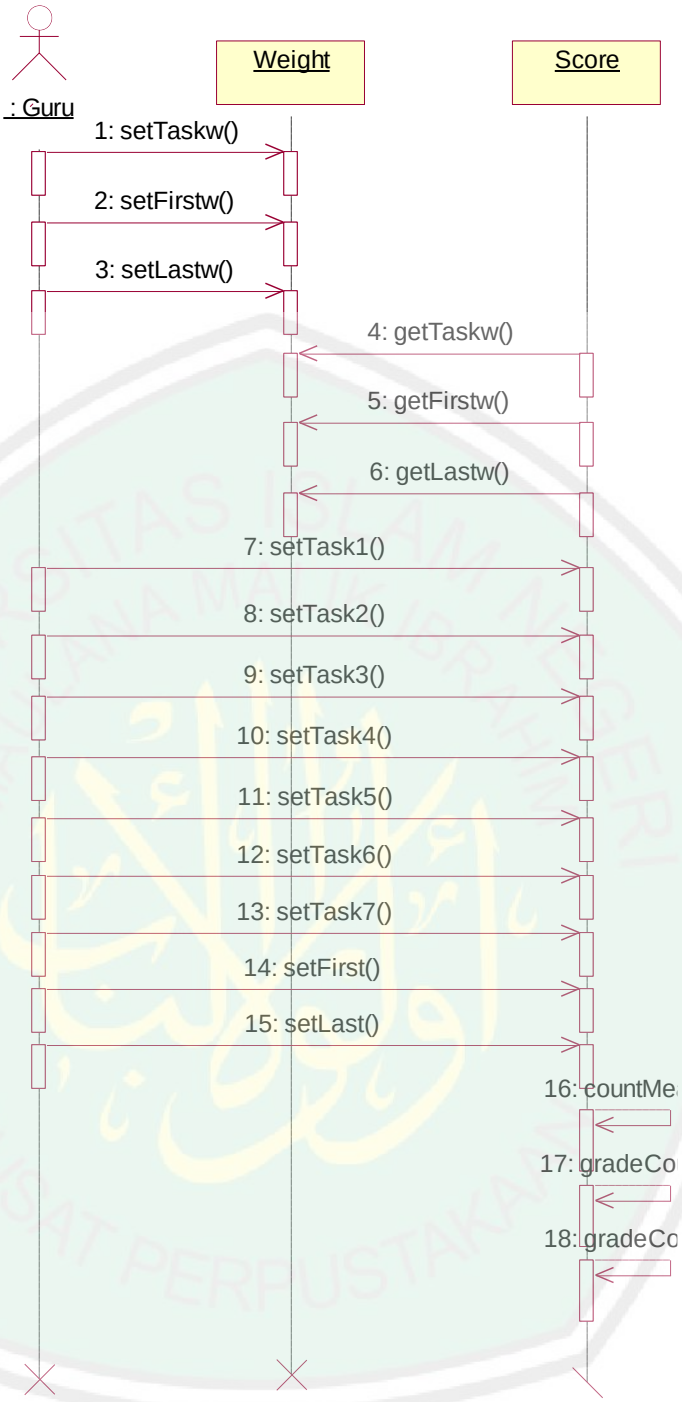
Dari kelima isi *use case* di atas, dibangun empat macam *sequence diagram*. (1). *Sequence diagram* perhitungan nilai (***SetWeight*** dan ***InputScore***), (2). *Sequence diagram* mendapatkan nama matakuliah (***SetCourse***), (3). *Sequence diagram* mendapatkan nama mahasiswa (***SetStudent***) dan (4). *Sequence diagram* menampilkan *report* (**CreateReport**). Selanjutnya akan dilakukan pengujian pada ketiga *sequence diagram* sistem penilaian pembelajaran ini.

(1). Gambar 4.5 *sequence diagram* perhitungan nilai pada sistem penilaian pembelajaran.

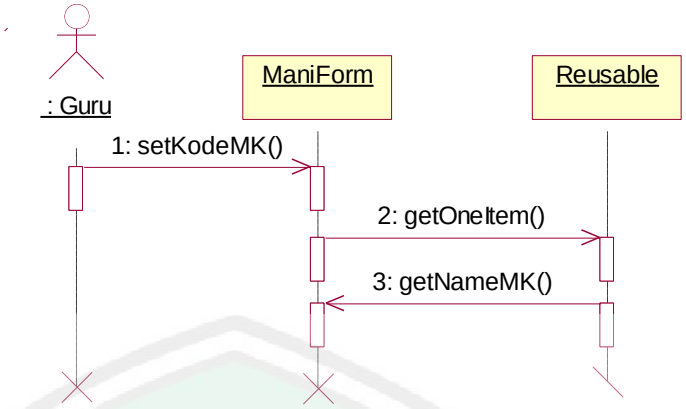
(2). Gambar 4.6 *sequence diagram* mendapatkan nama matakuliah pada sistem penilaian pembelajaran.

(3). Gambar 4.7 *sequence diagram* mendapatkan nama mahasiswa pada sistem penilaian pembelajaran.

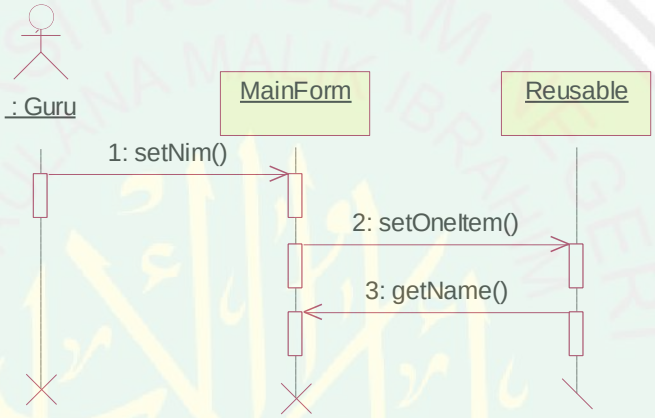
(4). Gambar 4.8 *sequence diagram* menampilkan *report* pada sistem penilaian pembelajaran yang dibuat pada *software rational rose* dan dieksport dengan ekstensi .xml.



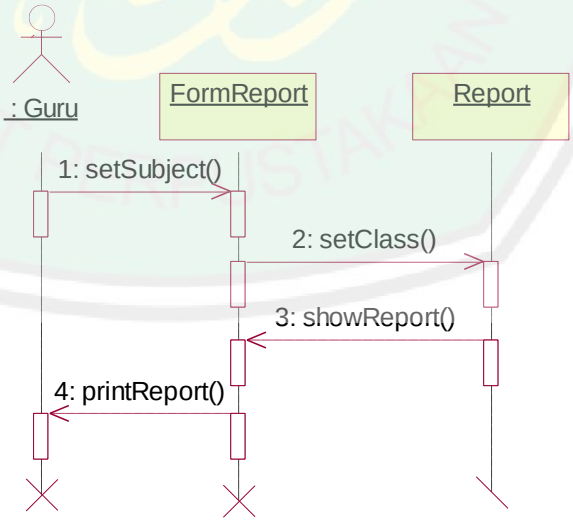
Gambar 4.5. *Sequence Diagram* Perhitungan Nilai pada Sistem Penilaian Pembelajaran



Gambar 4.6. *Sequence Diagram* Mendapatkan Nama Matakuliah pada Sistem Penilaian Pembelajaran



Gambar 4.7. *Sequence Diagram* Mendapatkan Nama Mahasiswa Sistem Penilaian Pembelajaran



Gambar 4.8. *Sequence Diagram* Menampilkan *Report* pada Sistem Penilaian Pembelajaran

2. Proses Pembuatan Aplikasi

Ada beberapa tahapan dalam pembuatan aplikasi *generate test path menggunakan model UML Sequence diagram* yang akan dibuat, berikut merupakan tahapan-tahapan proses yang ada dalam pembuatan aplikasi :

a. Konversi ke *FileXML*

Pada tahapan konversi ke *file* XML, proses yang terjadi adalah data dari model UML s*equence diagram Banking System* disave as dengan ekstensi *file* .XML seperti pada Gambar 4.9 di bawah. Yang kemudian dilakukan proses pemotongan kata untuk dicari data-data yang dibutuhkan dalam proses pembutan *test path* otomatis ini dengan menggunakan REGEX (*Regular Expression*) dan hasil dari proses regex seperti pada Gambar 4.10 di bawah yang otomatis akan tersimpan dengan ekstensi .XML pada lokasi *file* yang ditentukan oleh *user*. Berikut gambar data UML *Sequence diagram* sistem *Banking System* sebelum dan setelah proses konversi *file*.

Gambar 4.9. Data Uji Model UML *Sequence diagram* ke *File* XML sebelum Konversi *File*

Source code penerapan Regex untuk mengambil data-data yang dipelukan dari *file* dengan ekstensi .XML di atas.

```
while ((strLine = br.readLine()) != null) {
Pattern pattern = Pattern.compile("(?<=object Message).*");
Pattern pattern1 = Pattern.compile("(?<=sequence).*");
Pattern pattern2 = Pattern.compile("(?<=ordinal).*");
     Matcher matcher = pattern.matcher(strLine);
     Matcher matcher1 = pattern1.matcher(strLine);
     Matcher matcher2 = pattern2.matcher(strLine);
     boolean found = false;

      while (matcher.find()) {
         count++;
         jTextArea1.append("<symbol id='" + count +
"'>\n<objectmessage>" + matcher.group().toString() +
"</objectmessage>\n");
         found = true;
      }
 while (matcher1.find()) {
         jTextArea1.append("<sequence>" +
matcher1.group().toString() + "</sequence>\n");
         found = true;
      }
      while (matcher2.find()) {
         jTextArea1.append("<ordinal>" +
matcher2.group().toString() + "</ordinal>\n</symbol>\n");
         found = true;
      }
}
      jTextArea1.append("<symbolid='"+String.valueOf(count+1)+
"'>\n<objectmessage> End</objectmessage>");
      jTextArea1.append("\n<synchronization>"+"\t"+"Simple</sy
nchronization>");
      jTextArea1.append("\n<sequence>\t'"+String.valueOf(count
+1)+"'</sequence>");
      jTextArea1.append("\n<ordinal>\t"+0+"</ordinal>\n</symbo
l>");
      jTextArea1.append("\n</root>");
```

Gambar 4.10*File* XML setelah Konversi *File*

Setelah proses konversi *file* dilakukan dan hasilnya ditunjukkan pada Gambar 4.10 di atas, ternyata ada beberapa data yang tidak diperlukan ikut terambil. Datanya tampak seperti di bawah ini :

```
<symbol id='12'>
<objectmessage> "return"</objectmessage>
<synchronization>        "Return"</synchronization>
<sequence>       "11"</sequence>
<ordinal>        10</ordinal>
</symbol>
<ordinal>        0)</ordinal>
</symbol>
<ordinal>        1)</ordinal>
</symbol>
<ordinal>        2)</ordinal>
</symbol>
<ordinal>        3)</ordinal>
</symbol>
<ordinal>        4)</ordinal>
</symbol>
<ordinal>        5)</ordinal>
</symbol>
<ordinal>        6)</ordinal>
</symbol>
<ordinal>        7)</ordinal>
</symbol>
<ordinal>        8)</ordinal>
</symbol>
<ordinal>        9)</ordinal>
</symbol>
<ordinal>        10)</ordinal>
</symbol>
<ordinal>        11)</ordinal>
</symbol>
<symbol id='13'>
<objectmessage> End</objectmessage>
<synchronization>        Simple</synchronization>
<sequence>       “13”</sequence>
<ordinal>        0,12</ordinal>
</symbol>
</root>
```

Sehingga harus dilakukan *filtering* untuk menghilangkan data seperti di atas yang ditandai dengan warna merah. *Source code* untukmenghilangkan data yang tidak diperlukan seperti di atas adalah sebagai berikut :

```
int jumlah = 0;
int indeks = -1;
  int indeks1=2;
  String kata = "symbol id";
  String teks = jTextArea1.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 =
jTextArea1.getLineStartOffset((jumlah * 6) - 5);
    int ends = jTextArea1.getLineEndOffset(m.length
- 8);
    jTextArea1.replaceRange("", starts, ends);

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

Source code di atas, bekerja dengan menghitung jumlah kata yang mengandung "symbol id" dan disimpan dalam variabel *jumlah*, setelah itu membaca data-data dalam jTextArea perbaris. Kemudian mencari baris batas awal (*starts*) dan baris batas akhir (*ends*) data yang tidak diperlukan dengan rumus :

```
starts = (jumlah * 6)-5
ends = panjang baris - 8
```

Berdasarkan dari rumus tersebut, data yang tidak diperlukan dapat dideteksi dan dihapus. Sehingga hasil dari proses penghapusan data di atas, akan tampak seperti di bawah ini dan garis merah sebagai tanda titik letak posisi data sebelum terhapus.

```
<symbol id='12'>
<objectmessage> "return"</objectmessage>
<synchronization>       "Return"</synchronization>
<sequence>      "11"</sequence>
<ordinal>       10</ordinal>
</symbol>
<symbol id='13'>
<objectmessage> End</objectmessage>
<synchronization>       Simple</synchronization>
<sequence>      "13"</sequence>
<ordinal>       0,12</ordinal>
</symbol>
</root>
```

b. Konversi *XML* ke *Sequence Dependency Table* (SDT)

Pada tahapan konversi XML ke *Sequence Dependency Table* (SDT), data XML diambil berdasarkan *tag* yang ada dalam XML dan dimasukkan ke dalam empat kolom tabel (*Symbol, Activity Name, Sequence Number dan Dependency*). Berikut *Source Code* konversi XML ke SDT.

Source Code Konversi XML ke *Sequence Dependency Table* (SDT).

```
 public Relation(String Symbol, String ActivityName,
String SequenceNumber, String Dependency) {

        this.Symbol=Symbol;
        this.ActivityName = ActivityName;
        this.SequendeNumber = SequenceNumber;
        this.Dependency = Dependency;
    }
    public String toString() {
       return "<" + Symbol + ", " + ActivityName +
", " +SequendeNumber + ", " + Dependency + ">";
   //      return Symbol +","+Dependency;
    }

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

Tabel 4.1. Hasil *Sequence Dependency Table*

Symbol	Activity Name	Sequence Number	Dependency
1	insert card	1	0
2	enter pin	4	3
6	take cash	12	11
7	validate (card)	2	1
8	Return	3	2
	. . .		
13	End	13	2,3,6,7,10,11,12

c. Konversi *Sequence Dependency Table* (SDT) ke *Sequence Dependency Graph*(SDG)

Pada tahapan konversi *Sequence Dependency Table* (SDT) ke *Sequence Dependency Graph* (SDG), proses dilakukan dengan mengambil kolom *Dependency* dan kolom *Sequence* pada hasil *Sequence Dependency Table* (SDT) yang kemudian dijadikan sebagai *node* dalam *graph* (*Sequence Dependency Graph*). Dimana kolom *Dependency* sebagai *node*1 (*node* awal) sedangkan kolom *Sequence* sebagai *node*2 (*node* tujuan). Pada tahap ini akan terjadi dua proses yaitu (1) Proses mengambil *node* untuk di masukkan ke dalam *graph* dan (2) Proses identifikasi relasi antar dua *node* yang ada dalam satu *line*. Berikut *Source Code* untuk mengambil *node* ke dalam *graph* dan *source code* untuk pembangunan hubungan antar *node graph* dalam *Sequence Dependency Graph* (SDG).

Source code (1) Proses mengambil *node* ke dalam *graph*

```
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 data1 = arrayTeksSplit[1];
    String data2 = arrayTeksSplit[2];
    String[] data1a = data1.split(" ");
    String[] data2b = data2.split("\"");

for (int a = 0; a < data1a.length; a++) {
for (int b = 0; b < data2b.length; b++) {
String data1aa = data1a[0];
        String data2bb = data2b[1];

if (data1aa.length() != 1) {
      String[] data1aaa = data1a[0].split(",");
          for (int c = 0; c < data1aaa.length; c++){
             String data1mod = data1aaa[c];
             graph.addEdge(data1mod, data2bb);
  }
          } else {
             graph.addEdge(data1aa, data2bb);
}
        break;
}
    }
break;
   }
}
```

Source code di atas, akan bekerja dengan memisahkan *node* berdasarkan tanda *spasi* dan tanda "\t" untuk dimasukkan ke dalam *node*1 dan *node*2. Namun jika *node*1 memiliki panjang ***index*** lebih dari satu maka akan dilakukan pemisahan *node* lagi berdasarkan tanda koma (,). Begitu pula jika terjadi pada *node*2.

Source code (2) Proses membangun relasi antar *node*

```
 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);
}
```

Source code di atas berfungsi untuk membangun hubungan antar *node* atau yang disebut dengan *adjacency node* berfungsi menunjukkan kedekatan antar *node* dalam *graph* dan disimpan dalam *Hashmap*. *Output* yang dihasilkan akan seperti berikut:

Tabel 4.2. Hasil Relasi antar node dalam *Sequence Dependency Graph*

Dependency	Sequence Number
0	1
1	2
2	3
3	4
4	5
5	6
. .	
2,3,6,7,10,11,12	13

Tabel 4.3. *Graph* dalam *Hashmap*

Key	Value
1	2
2	3,13
3	4,13
4	5
5	6
6	7,13
7	8,13
Dst….	

Maksud tabel *hashmap* di atas merupakan bentuk *graph* yang disimpan dalam *hashmap*, key (node1) memiliki relasi pada value (node2). Node1 "1" terhubung pada node2 "2", node1 "2" memiliki hubungan ganda pada node2 "3" dan "13", begitu seterusnya.

d. Penerapan Metode *Depth First Search* (DFS)

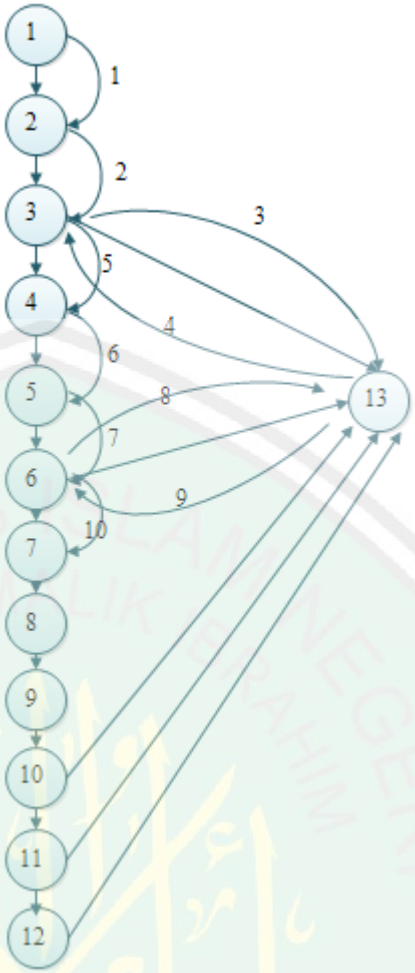
Pada tahapan penerapan metode *Depth First Search* (DFS), proses ini dilakukan ketika *graph* sudah berhasil mengambil semua *node* yang terhubung dari dua kolom *Sequence Dependency*

Table (SDT) dan disimpan dalam *hashmap*. Kemudian metode ini bekerja sebagai pencari jalur pada *graph* sehingga menghasilkan jalur pengujian (*Test path*). Berikut *source code* mengambil *node* ke dalam *graph* dan *source code* penerapan metode *Depth First Search* (DFS).

Source Code Penerapan *Depth First Search* (DFS)

```
        for (String node : nodes) {
            if (visited.contains(node)) {
                continue;
            }
            if (node.equals(END)) {
                visited.add(node);
                printPath(visited);
                visited.removeLast();
                break;
            }
        }
        for (String node : nodes) {
            if (visited.contains(node) ||
node.equals(END)) {
                continue;
            }
            visited.addLast(node);
            depthFirst(graph, visited);
            visited.removeLast();
        }
```

Source code di atas menyimpan semua *node* yang sudah saling terhubung ke dalam *hashmap*. Selanjutnya proses pencarian dilakukan dengan mengunjungi *node* awal terlebih dahulu hingga tiba di simpul terakhir dan dicetak. Jika tujuan yang diinginkan belum tercapai maka pencarian dilanjutkan ke cabang sebelumnya, turun ke bawah jika memang masih ada cabangnya yang belum dikunjungi. Representasi pengambilan *path* pada *graph* dari *source code* di atas dapat dilihat pada Gambar 4.11 di bawah ini.



Gambar 4.11. Representasi DFS pada *Graph*

Representasi DFS pada *graph* di atas bekerja sesuai dengan alur penomoran hingga menuju alur ke node terakhir "13". Pada *path* pertama :

a. Alur nomor 1 yang mengarahkan dari *node* 1 ke ***node*** 2, sehingga mencetak *path* 1 2
b. Dilanjutkan alur nomor 2 mengarahkan dari node 2 ke node 3, sehingga *path* 1 2 3
c. Dilanjutkan alur nomor 3 mengarahkan dari node 3 ke node 12, sehingga *path* bertambah 1 2 3 13.

Mengambil *Test path* dilakukan setelah *Sequence Dependency Graph* (SDG) diproses dengan metode *Depth First Search* (DFS). Berikut *source code* pengambilan *path* dari *Sequence Dependency Graph* (SDG).

Source code menampilkan *Path.*

```
for (String node : visited) {
        System.out.print(node);
        System.out.print(" ");
        TextAreaGeneratePath.append(node + " ");
    }
 TextAreaGeneratePath.append("\n");
```

Source code di atas akan mencetak semua *node* yang sudah dikunjungi.

Jalur *path* yang dihasilkan
1 2 13
1 2 3 13
1 2 3 4 5 6 13
1 2 3 4 5 6 7 13
1 2 3 4 5 6 7 8 9 10 13
1 2 3 4 5 6 7 8 9 10 11 13
1 2 3 4 5 6 7 8 9 10 11 12 13

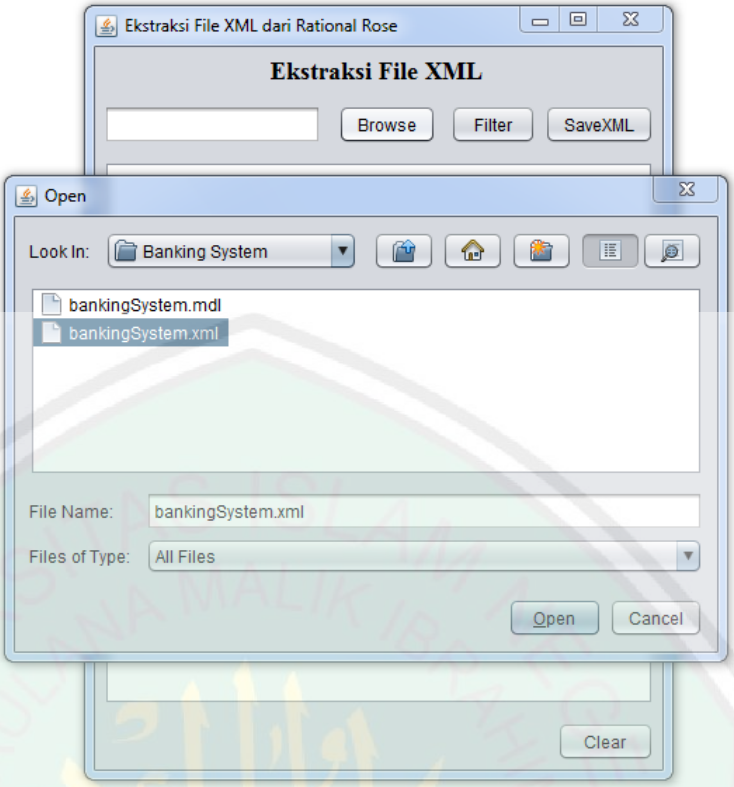
3. Pengujian Aplikasi

Aplikasi pembangkit *test case* otomatis ini dibangun dengan menggunakan bahasa pemrograman Java. Pembuatan aplikasi ini memanfaatkan Netbeans IDE 7.2 sebagai editor untuk penulisan kode program dan *Rational Rose Enterprise Edition* untuk mendesain data awal atau *sequence diagram* dari suatu sistem yang akan diuji. Untuk mempermudah pengguna dalam mengoperasikan aplikasi ini, sehingga dibuat sebuah tampilan (*interface)* berbasis GUI (*Graphic User Interface*) yang kemudian akan dijelaskan lebih rinci dibawah ini.

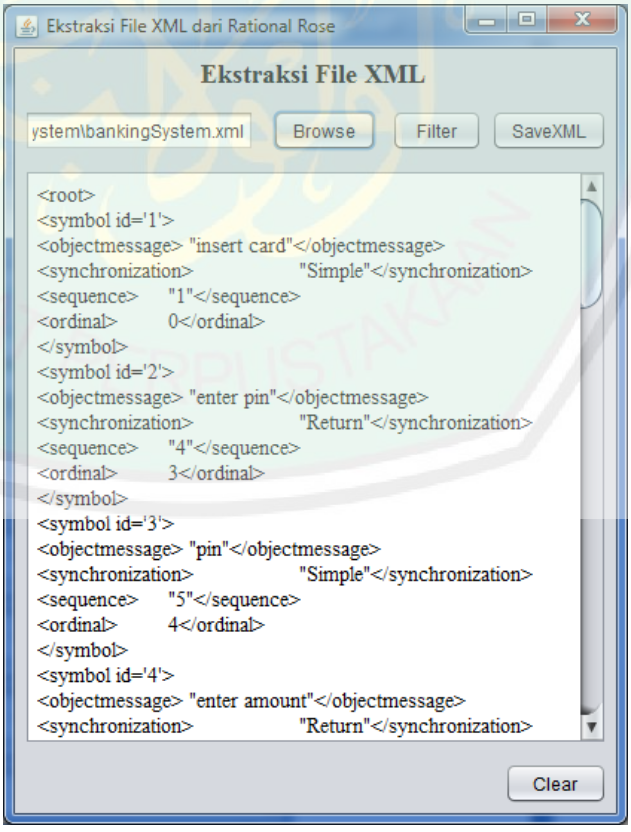
Aplikasi pembangkit *test case* otomatis ini, terdiri dari dua *interface*: (1) *interface* ekstraksi XML dan (2) *interface* pembangkit *test path*. (1) *interface* ekstraksi XML, digunakan untuk mengambil data-data penting yang ada dari *file* XML yang merupakan *output* dari *file sequence diagram* di *Rational Rose* dan akan disimpan kembali dengan ekstensi XML.

Gambar 4.12. *Interface* Ekstraksi *File* XML

Ekstraksi *file* XML ini dilakukan dengan mencari *file* XML hasil *software rational rose* setelah membuat *sequence diagram*.



Gambar 4.13. *Browsing File* XML



Gambar 4.14. Hasil Ekstraksi *File* XML

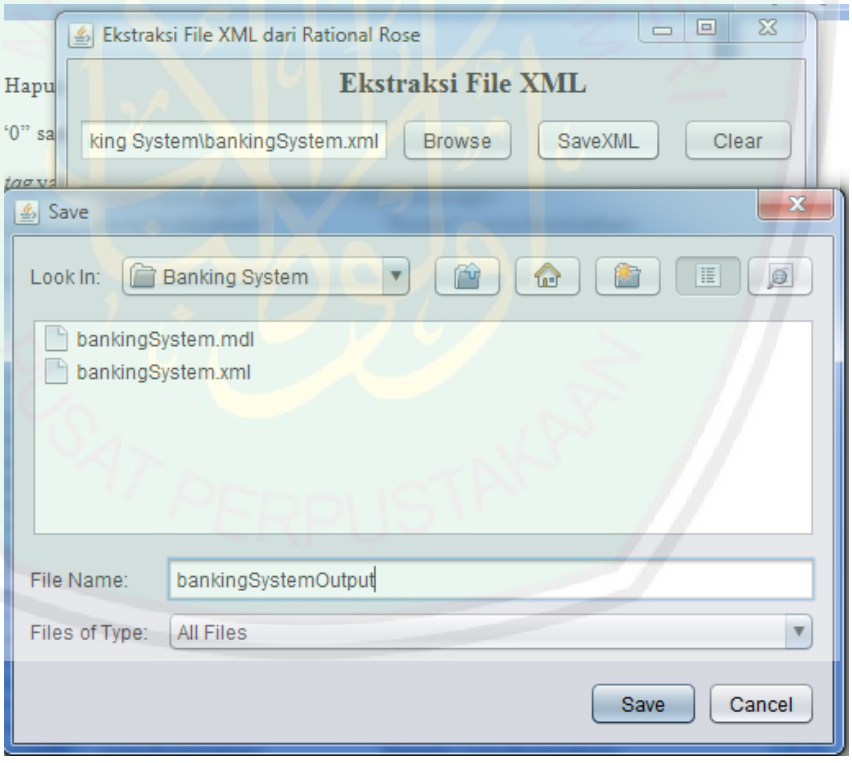
Tekan tombol *filter* untuk menghapus data yang tidak diperlukan dan satu hal yang dilakukan secara manual dalam program konversi xml ini, yaitu mencari nilai *tag ordinal* yang *tag synchronization*-nya bernilai “Return” dan tulis nilai *ordina-ordinal* itu pada *tag ordinal* terakhir untuk menggantikan nilai “0” di <symbol id= ‘13’>. Dari lingkaran merah di bawah, dituliskan pada lingkaran biru.

```
<root>
<symbol id='1'>
<objectmessage> "insert card"</objectmessage>
<synchronization>       "Simple"</synchronization>
<sequence>      "1"</sequence>
<ordinal>       0</ordinal>
</symbol>
<symbol id='2'>
<objectmessage> "enter pin"</objectmessage>
<synchronization>       "Return"</synchronization>
<sequence>      "4"</sequence>
<ordinal>       3</ordinal>
</symbol>
<symbol id='3'>
<objectmessage> "pin"</objectmessage>
<synchronization>       "Simple"</synchronization>
<sequence>      "5"</sequence>
<ordinal>       4</ordinal>
</symbol>
<symbol id='4'>
<objectmessage> "enter amount"</objectmessage>
<synchronization>       "Return"</synchronization>
<sequence>      "8"</sequence>
<ordinal>       7</ordinal>
</symbol>
.
.
.
.
.
<symbol id='13'>
<objectmessage> End</objectmessage>
<synchronization>       Simple</synchronization>
<sequence>      '13'</sequence>
<ordinal>       0,12</ordinal>
</symbol>
</root>
```

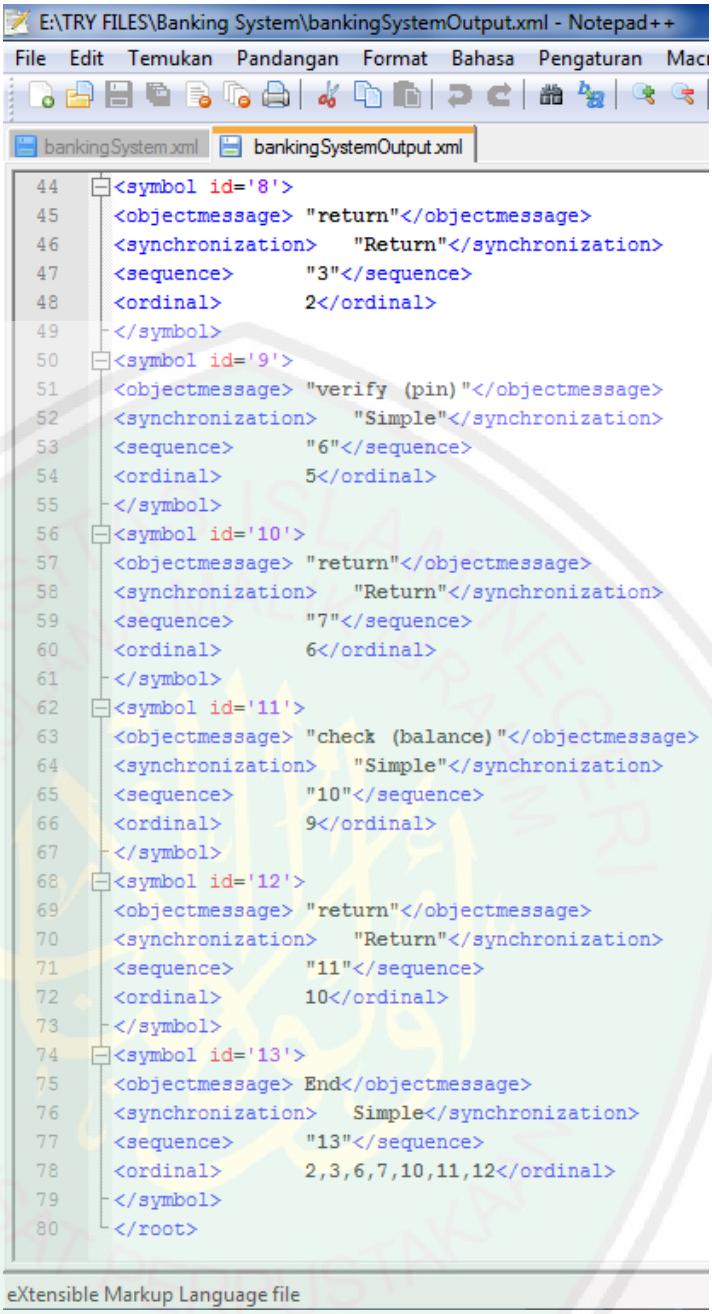
Sehingga hasilnya akan seperti ini :

```
<symbol id='12'>
<objectmessage> "return"</objectmessage>
<synchronization>       "Return"</synchronization>
<sequence>      "11"</sequence>
<ordinal>       10</ordinal>
</symbol>
<symbol id='13'>
<objectmessage> End</objectmessage>
<synchronization>       Simple</synchronization>
<sequence>      "13"</sequence>
<ordinal>       2,3,6,7,10,11,12</ordinal>
</symbol>
</root>
```

Setalah mengerjakan secara manual *file* di atas, simpan *file* pada direktori.

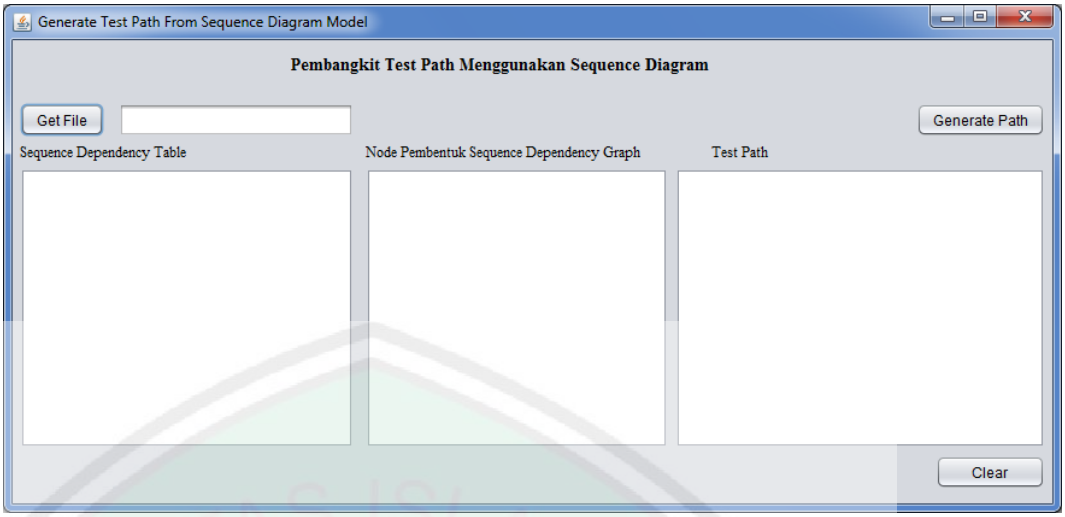


Gambar 4.15.Menyimpan Hasil *File* Ekstraksi XML



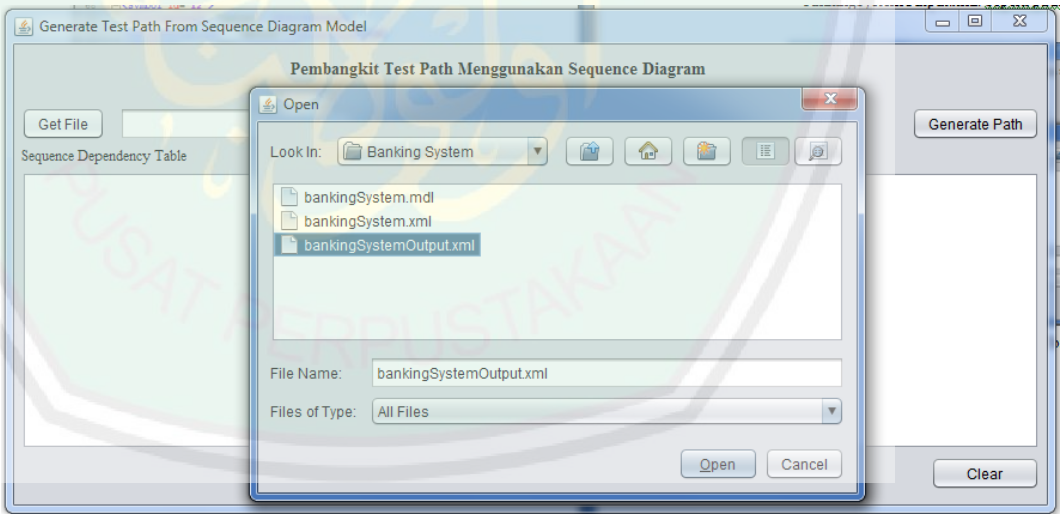
Gambar 4.16. Hasil *Output* Konversi *File* XML

Hasil *output* konversi *file* XML ini digunakan sebagai *input* pada aplikasi Pembangkit *Test Path*. (2) *Interface* pembangkit *Test Path*. Tampilan pada aplikasi ini ditunjukkan pada gambar 4.17 di bawah ini.



Gambar 4.17. Tampilan Awal Aplikasi Pembangkit *Testpath*

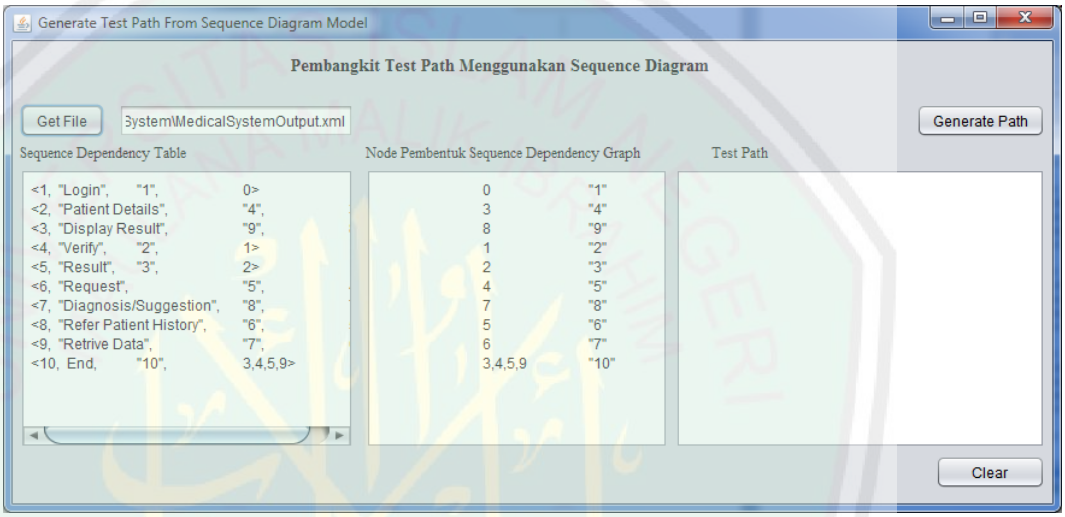
Untuk menggunakan program pada tampilan pembangkit *test path* ini, pertama cari *file* XML yang sebelumnya sudah dilakukan konversi dari *file* awal XML yang disimpan dengan nama *file* bankingSystemOutput.xml. Seperti pada gambar 4.18 di bawah ini.



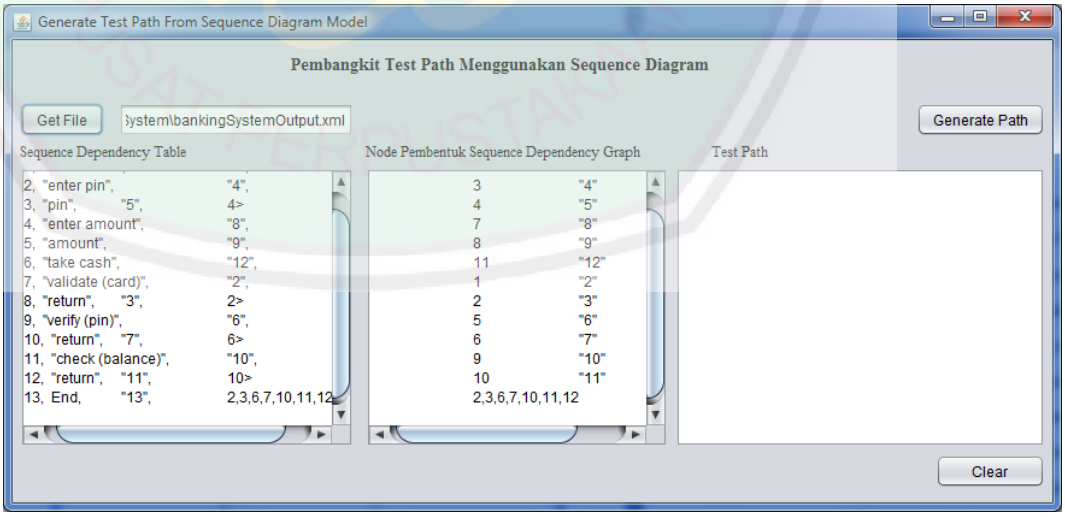
Gambar 4.18. Tampilan Tombol *Get File*

Dalam percobaan ini, data yang dicari adalah data XML *Medical Consultational System*, data *ATM System atau Banking System*, data *Aircraft Departure Activity* dan data pada sistem penilaian

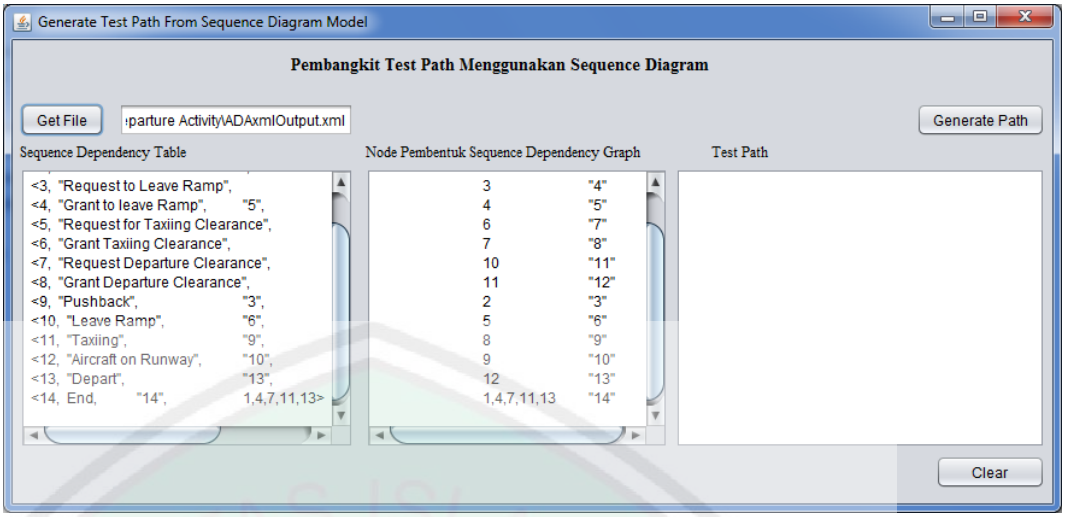
pembelajaran yang akan diujikan secara bergantian. Pada pengujian sistem penilaian pembelajaran, akan diwakilkan pada data "*sequence diagram* perhitungan nilai". Berikut tampilan setelah proses pencarian data *XML Medical Consultational System*, data *XML banking System*, *XML Aircraft Departure Activity* dan data XML perhitungan nilai pada sistem penilaian pembelajaran.



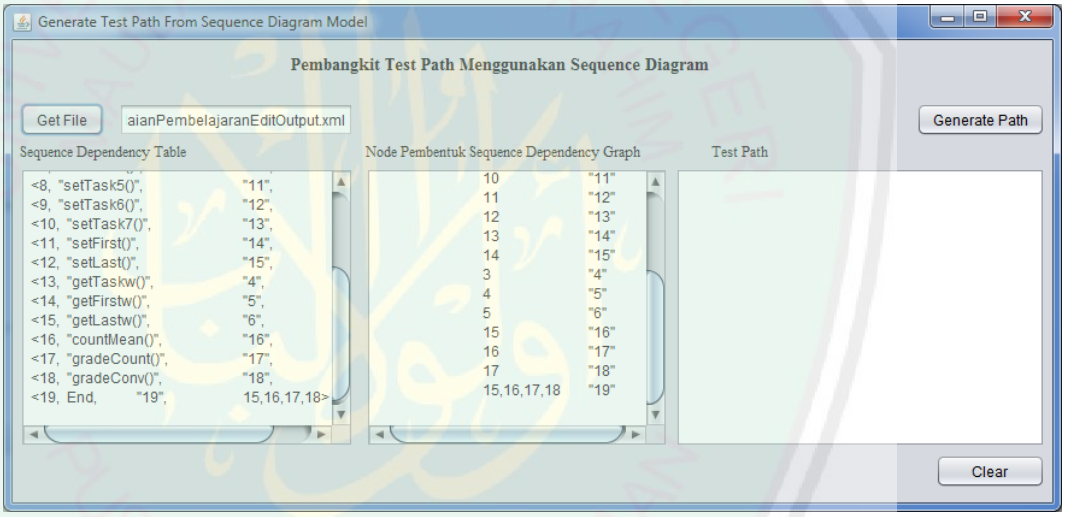
Gambar 4.19. Tampilan Hasil Tombol "Get *File*" Data *XML Medical Consultational System*



Gambar 4.20. Tampilan Hasil Tombol "Get *File*" Data *XML Banking*



Gambar 4.21. Tampilan Hasil Tombol "Get *File*" Data *XML Aircraft Departure Activity*

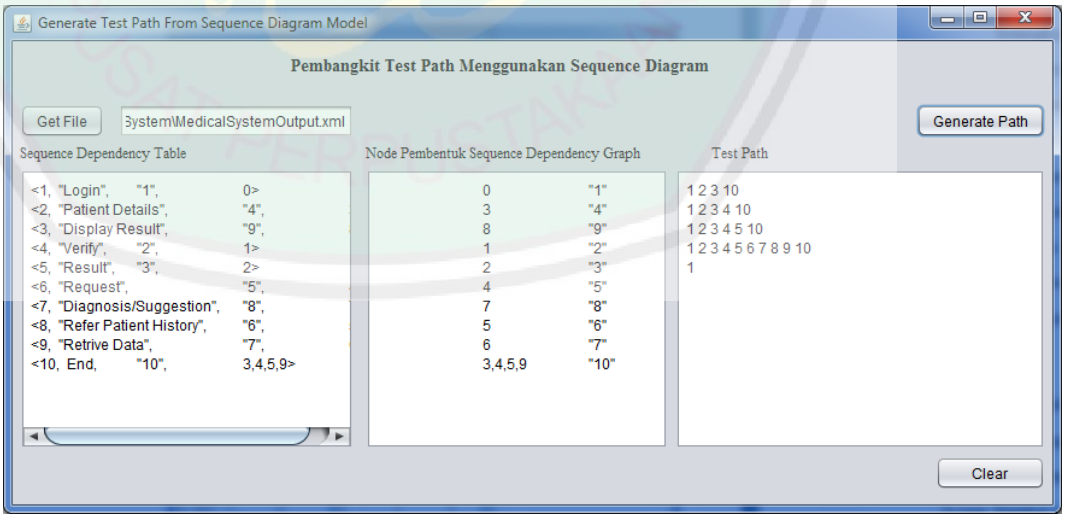


Gambar 4.22. Tampilan Hasil Tombol "Get *File*" Data Perhitungan Nilai pada Sistem Penilaian Pembelajaran

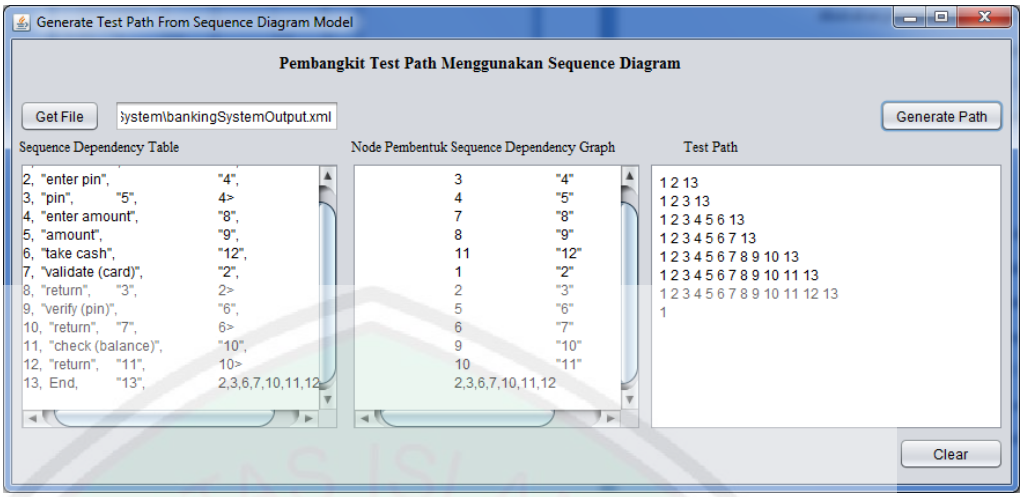
Sequence Dependency Table (SDT) diatas terdiri dari empat kolom yaitu *Symbol*, *Activity Name*, *Sequence Number* dan *Dependency*. Sebagai penjelasan pada *line* satu Gambar 4.20 yaitu hasil *Sequence Dependency Table* (SDT) data *Banking System* terdapat *Symbol* "1", *Activity Name* "*Insert Card*", *Sequence Number* "1", dan *Dependency* "-". Pada *line* satu memiliki nilai *Dependency* "-" atau

dikatakan kosong karena *node* tidak memiliki keterkaitan dengan *node* sebelumnya atau bisa dikatakan sebagai *node* awal pada *line* satu adalah "1". Selanjutnya dalam pembentukan *Sequence Dependency Graph* (SDG), dari data *Sequence Dependency Table* (SDT) diambil data dari kolom *Dependency* sebagai *node*1 dan kolom *Ordinal* sebagai *node*2. Sehingga didapatkan *node-node* yang saling terhubung sebagai representasi graf.

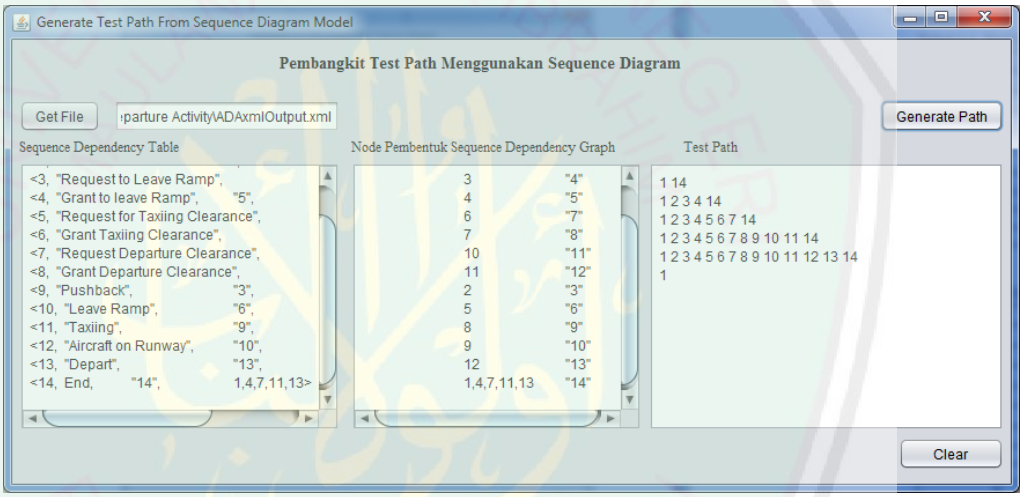
Setelah didapatkan data *Sequence Dependency Table* (SDT) dan node pembentuk *Sequence Dependency Graph* (SDG) pada Gambar 4.19, Gambar 4.20, Gambar 4.21 dan Gambar 4.22 di atas, *test case* bisa didapatkan dengan meng-klik tombol "Generate *Path*" sehingga hasil *path* akan tampil pada kolom *test path* yang sudah tersedia. Berikut tampilan hasil *test case* dari data *Medical Consultational System*, data *ATM System atau Banking System*, data *Aircraft Departure Activity* dan data perhitungan nilai pada sistem penilaian pembelajaran.



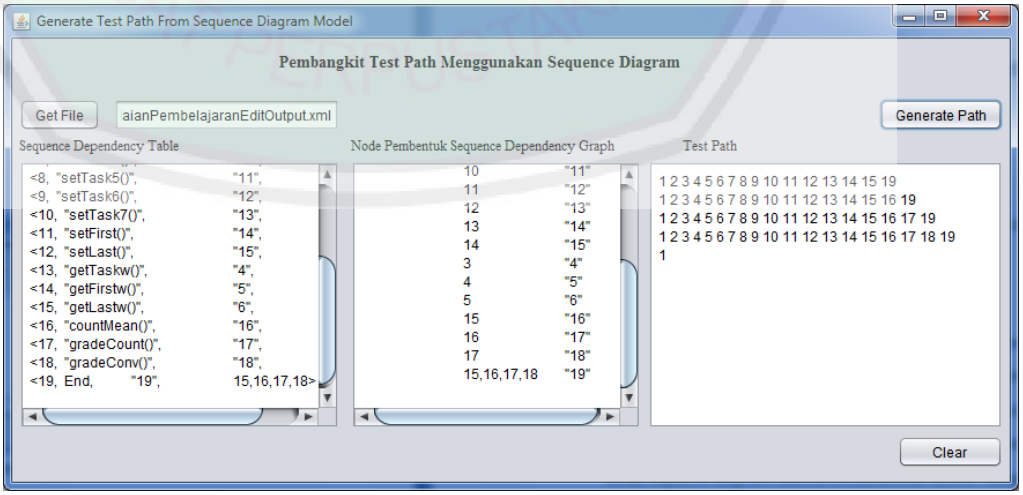
Gambar 4.23. Tampilan Tombol Generate *Path* Data *Medical Consultational System*



Gambar 4.24. Tampilan Tombol Generate *Path* Data *Banking System*



Gambar 4.25. Tampilan Tombol Generate *Path* Data *Aircraft Departure Activity*



Gambar 4.26. Tampilan Tombol Generate *Path* Data Perhitungan Nilai pada Sistem Penilaian Pembelajaran

Path dihasilkan dari data *node* pembentuk *Sequence Dependency Graph* (SDG) yang ketika *path* dibangkitkan, *node* pembentuk SDG akan disimpan dalam *hashmap* pada sistem aplikasi pembangkit *test path* ini. Berikut penjelasan bentuk graf dalam sistem pembangkit ini, node-node tersimpan dalam *hashmap*. Tabel 4.4 graf sistem dari data *medical consultational system*, Tabel 4.5 graf sistem dari data *banking system*, Tabel 4.6 graf sistem dari data *aircraft departure activity* dan Tabel 4.7 graf sistem dari data perhitungan nilai pada sistem penilaian pembelajaran.

Tabel 4.4. Graf Sistem dari Data *Medical Consultational System*

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

Tabel 4.5. Graf Sistem dari Data *Banking System*

Key	Value
3	[4, 13]
2	[3, 13]
10	[11, 13]
1	[2]
7	[8, 13]
6	[7, 13]
5	[6]
4	[5]
9	[10]
8	[9]
11	[12, 13]
12	[13]

Tabel 4.6. Graf Sistem dari Data *Aircraft Departure Activity*

Key	Value
3	[4]
2	[3]
10	[11]
1	[2,14]
7	[8, 14]
6	[7]
5	[6]
4	[5,14]
9	[10]
8	[9]
13	[14]
11	[12, 14]
12	[13]

Tabel 4.7. Graf Sistem dari Data Perhitungan Nilai pada Sistem Penilaian Pembelajaran

Key	Value
17	[18, 19]
18	[19]
15	[16,19]
16	[17,19]
13	[14]
14	[15]
11	[12]
12	[13]
3	[4]
2	[3]
1	[2]
10	[11]
7	[8]
6	[7]
5	[6]
4	[5]
9	[10]
8	[9]

Dari graf yang disimpan dalam memori sistem tersebut, kemudian dengan menggunakan metode *depth first search* akan dihasilkan *path*.

Sebagai penjelasan pada Gambar 4.24 *generate path* data *banking system* di atas, terdiri dari tiga belas baris yang menunjukkan hubungan antar *node*. Baris satu adalah "- spasi 1" menunjukkan 1 sebagai *node* awal yang tidak memiliki hubungan dengan *node* sebelumnya.

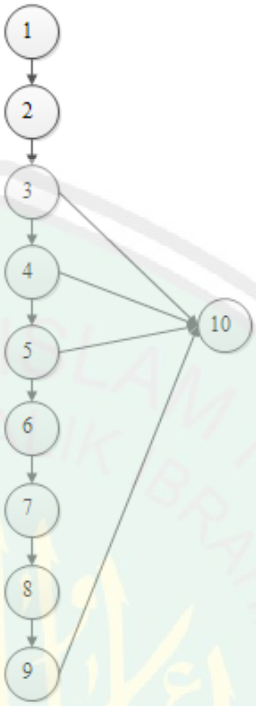
Sedangkan pada baris terakhir "2,3,6,7,10,11,12 spasi 13" menunjukkan *node* "2,3,6,7,10,11,12" sebagai *node*1 yang sama-sama terhubung pada "13" atau *node*2. Agar graf sistem dapat membaca *node-node* yang menjadi *node*1 dalam baris tersebut, sehingga dilakukan pemisahan data berdasarkan tanda koma "," yang kemudian sistem akan mengartikan sebagai "2 spasi 13", "3 spasi 13", "6 spasi 13", "7 spasi 13", "10 spasi 13", "11 spasi 13" dan "12 spasi 13" kemudian sistem akan menghasilkan jalur *path* seperti pada Gambar 4.23, Gambar 4.24, Gambar 4.25 dan Gambar 4.26 di atas.

B. Pembahasan

Setelah dilakukan uji coba, diperoleh hasil *path medical consultational system, banking system, aircraft departure activity* dan sistem penilaian pembelajaran. Hasil *path medical consultational system, path banking system* dan *path aircraft departure activity* akan dibandingkan dengan hasil pada artikel Shanti et al (2012), Priya et al (2013) dan Rhmann (2016). Perbandingan pertama dilakukan pada data *Medical Consultational System* yang dibandingkan dengan hasil artikel Priya et al (2013).

Gambar 4.27 merupakan representasi *graph* yang terbentuk dari model *sequence diagram* ***Medical Consultational System***. Selanjutnya,

perbandingan *output* penelitian yang dilakukan dan *output* dari artikel Priya et al (2013) akan ditampikan pada Tabel 4.8 di bawah ini.



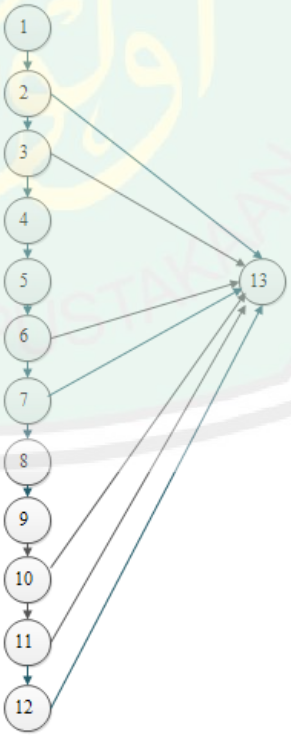
Gambar 4.27. Representasi *GraphMedical Consultational System*

Tabel 4.8. Perbandingan uji coba aplikasi dan artikel data *Medical consultational system*

Hasil Uji	**Coba Aplikasi**	**Hasil Uji**	**Coba Artikel**
Testpath	***Test case***	***Testpath***	***Test case***
1 2 3 10	*Login*, Verify, Result, End	A B C J	*Login*, Verify, Result, End
1 2 3 4 10	*Login*, Verify, Result, Patient History, End	A B C D J	*Login*, Verify, Result, Patient History, End
1 2 3 4 5 10	*Login*, Verify, Result, Patient History, Request, End	A B C D E J	*Login*, Verify, Result, Patient History, Request, End
1 2 3 4 5 6 7 8 9 10	*Login*, Verify, Result, Patient History, Request, Refer Patient History, Retrieve Details, Diagnosis/suggestion, Display Result, End	A B C D E F G H I J	*Login*, Verify, Result, Patient History, Request, Refer Patient History, Retrieve Details, Diagnosis/suggestion, Display Result, End

Tabel di atas merupakan hasil perbandingan hasil pengujian otomatis dari data *medical consultational system*. Aplikasi pembangkit pengujian otomatis ini berhasil mendeteksi empat alternatif jalur pengujian (*Test path*) dari data tersebut dan memberikan hasil yang sama pula dari *output* jalur *path* yang ada pada artikel rujukan. *Best path* dapat diambil dari jalur terlengkap dari beberapa *path* yang ada, dalam hal ini *Best Path* dalam pengujian data di atas adalah 1 2 3 4 5 6 7 8 9 10.

Selanjutnya hasil perbandingan percobaan kedua pada ***Banking System* atau *ATM System*** dengan hasil artikel Shanti et al (2012). Gambar 4.28 merupakan representasi *graph* yang dihasilkan dari *sequence diagram* data *Banking System* atau *ATM System* dan Tabel 4.9 menunjukkan perbandingan *output* program data *Banking System* atau *ATM System* dalam bentuk tabel.



Gambar 4.28. Representasi *Graph Banking System atau ATM System*

Tabel 4.9. Perbandingan uji coba aplikasi dan artikel data *Banking system* atau *ATM system*

Hasil Uji	**Coba Aplikasi**	**Hasil Uji**	**Coba Artikel**
Testpath	***Test case***	***Testpath***	***Test case***
1 2 13	Insert Card, Validate (Card), End	A B M	Insert Card, Validate (Card), End
1 2 3 13	Insert Card, Validate (Card), Return, End	A B D E F M	Insert Card, Validate (Card), Enter Pin, Pin, Verify (Pin), End
1 2 3 4 5 6 13	Insert Card, Validate (Card), Return, Enter Pin, Pin, Verify (Pin), End	A B D E F H I J M	Insert Card, Validate (Card), Enter Pin, Pin, Verify (Pin), Enter Amount, Amount, Check (Balance), End
1 2 3 4 5 6 7 13	Insert Card, Validate (Card), Return, Enter Pin, Pin, Verify (Pin), Return, End	A B D E F H I J L M	Insert Card, Validate (Card), Enter Pin, Pin, Verify (Pin), Enter Amount, Amount, Check (Balance), Take Cash, End
1 2 3 4 5 6 7 8 9 10 13	Insert Card, Validate (Card), Return, Enter Pin, Pin, Verify (Pin), Return, Enter Amount, Amount, Check (Balance), End		
1 2 3 4 5 6 7 8 9 10 11 13	Insert Card, Validate (Card), Return, Enter Pin, Pin, Verify (Pin), Return, Enter Amount, Amount, Check (Balance), Return, End		
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 di atas merupakan tabel perbandingan dari data *Banking System* atau *ATM System* antara *output* dari aplikasi yang dibuat dengan *ouput* artikel rujukan. Hasil *output* aplikasi pembangkit pengujian otomatis ini menghasilkan tujuh jalur pengujian (*Test path*). Sedangkan *output* aplikasi pada artikel Shanti et al (2012) menunjukkan empat jalur pengujian (*Test path*). Hal ini disebabkan karena aplikasi pada artikel tidak mengeksekusi *path* (Symbol) yang menangani pesan "Return" dalam *sequence diagram* yang ada. Sedangkan aplikasi pembangkit jalur pengujian otomatis ini akan mengeksekusi semua *path* yang tercantum pada *graph*. Penjelasan lebih rinci akan dijelaskan pada tabel di bawah ini.

Tabel 4.10. Penjelasan Symbol *Banking System*

Symbol		Activity Name
Artikel	Penelitian	
A	1	Insert Card
B	2	Validate (Card)
C	3	Return
D	4	Enter Pin
E	5	Pin
F	6	Verify (Pin)
H	7	Return
I	8	Enter Amount
J	9	Amount
K	10	Check (Balance)
L	11	Return
M	12	Take Cash
N	13	End

Tabel *output path* pada artikel Shanti (2012) :

Tabel 4.11. *Output Path* artikel Shanti (2012)

Jurnal									
A	B	M							
Insert Card	Validate (Card)	End							
A	B	D	E	F	M				
Insert Card	Validate (Card)	Enter Pin	Pin	Verify (Pin)	End				
A	B	D	E	F	H	I	J	M	
Insert Card	Validate (Card)	Enter Pin	Pin	Verify (Pin)	Enter Amount	Amount	Check (Balance)	End	
A	B	D	E	F	H	I	J	L	M
Insert Card	Validate (Card)	Enter Pin	Pin	Verify (Pin)	Enter Amount	Amount	Check (Balance)	Take Cash	End

Tabel *Output path*pada penelitian ini :

Tabel 4.12. *Output Path* Penelitian ini

Penelitian												
1	2	13										
Insert Card	Validate (Card)	End										
1	2	3	13									
Insert Card	Validate (Card)	Return	End									
1	2	3	4	5	6	13						
Insert Card	Validate (Card)	Return	Enter Pin	Pin	Verify (Pin)	End						
1	2	3	4	5	6	7	13					
Insert Card	Validate (Card)	Return	Enter Pin	Pin	Verify (Pin)	Return	End					
1	2	3	4	5	6	7	8	9	10	13		
Insert Card	Validate (Card)	Return	Enter Pin	Pin	Verify (Pin)	Return	Enter Amount	Amount	Check (Balance)	End		
1	2	3	4	5	6	7	8	9	10	11	13	
Insert Card	Validate (Card)	Return	Enter Pin	Pin	Verify (Pin)	Return	Enter Amount	Amount	Check (Balance)	Return	End	
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

Setelah angka-angka yang mewakili *Activity Name* "Return" pada *path* yang telah tersusun di atas dihilangkan, akan menjadi seperti tabel di bawah ini :

Tabel 4.13. *Output Path* setelah Nilai "Return" Dihilangkan

Penelitian									
1	2	13							
Insert Card	Validate (Card)	End							
1	2	13							
Insert Card	Validate (Card)	End							
1	2	4	5	6	13				
Insert Card	Validate (Card)	Enter Pin	Pin	Verify (Pin)	End				
1	2	4	5	6	13				
Insert Card	Validate (Card)	Enter Pin	Pin	Verify (Pin)	End				
1	2	4	5	6	8	9	10	13	
Insert Card	Validate (Card)	Enter Pin	Pin	Verify (Pin)	Enter Amount	Amount	Check (Balance)	End	
1	2	4	5	6	8	9	10	13	
Insert Card	Validate (Card)	Enter Pin	Pin	Verify (Pin)	Enter Amount	Amount	Check (Balance)	End	
1	2	4	5	6	8	9	10	12	13
Insert Card	Validate (Card)	Enter Pin	Pin	Verify (Pin)	Enter Amount	Amount	Check (Balance)	Take Cash	End

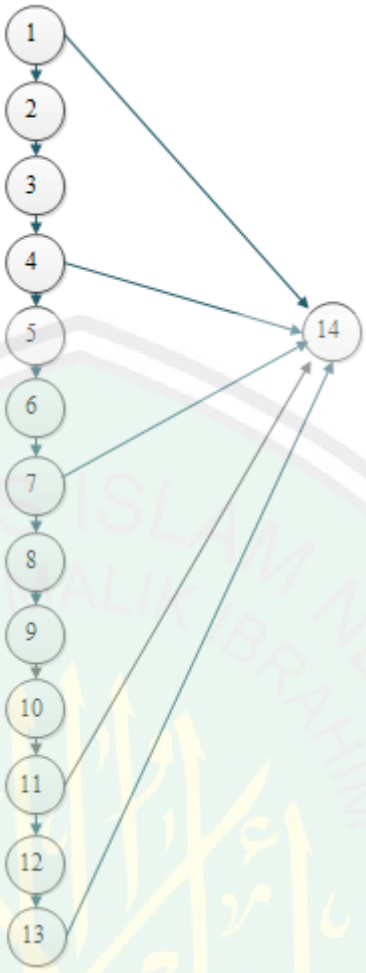
Setelah disederhanakan:

Tabel 4.14. *Output Path* setelah Disederhanakan

Penelitian									
1	2	13							
Insert Card	Validate (Card)	End							
1	2	4	5	6	13				
Insert Card	Validate (Card)	Enter Pin	Pin	Verify (Pin)	End				
1	2	4	5	6	8	9	10	13	
Insert Card	Validate (Card)	Enter Pin	Pin	Verify (Pin)	Enter Amount	Amount	Check (Balance)	End	
1	2	4	5	6	8	9	10	12	13
Insert Card	Validate (Card)	Enter Pin	Pin	Verify (Pin)	Enter Amount	Amount	Check (Balance)	Take Cash	End

Hasil akhir menunjukkan bahwa output yang dihasilkan oleh program penelitian ini dengan *output* artikel Shanti (2012) adalah sama. Dengan demikian *best path* yang dihasilkan pada artikel adalah *path* "A B D E F H I J L M" sedangkan pada program yang dibuat adalah *path* "1 2 3 4 5 6 7 8 9 10 11 12 13" dengan tetap menyertakan angka *path* yang mewakili *Activity Name* "Return".

Selanjutnya hasil perbandingan percobaan ketiga pada ***Aircraft Departure Activity*** dengan hasil artikel Rhmann (2016). Gambar 4.29 merupakan representasi *graph* yang dihasilkan dari *sequence diagram* data *Aircraft Departure Activity* dan Tabel 4.15 menunjukkan perbandingan *output* program data *Aircraft Departure Activity* dalam bentuk tabel.



Gambar 4.29. Representasi *Graph Aircraft Departure Activity*

Tabel 4.15. Perbandingan uji coba aplikasi dan artikel data *Aircraft Departure Activity* dari *Aircraft Control System*

Hasil Uji	**Coba Aplikasi**	**Hasil Uji**	**Coba Artikel**
Testpath	***Test case***	***Testpath***	***Test case***
1 14	Request for Pushback, End	1 14	Request for Pushback, Permission not Granted, End
1 2 3 4 14	Request for Pushback,Grant Pushback Clearance, Pushback, Request to Leave Ramp, End	1 2 3 4 14	Request for Pushback,Grant Pushback Clearance, Pushback, Request to Leave Ramp, Permission not Granted, End
1 2 3 4 5 6 7 14	Request for Pushback, Grant Pushback Clearance, Pushback, Request to Leave Ramp,	1 2 3 4 5 6 7 14	Request for Pushback, Grant Pushback Clearance, Pushback, Request to Leave Ramp,

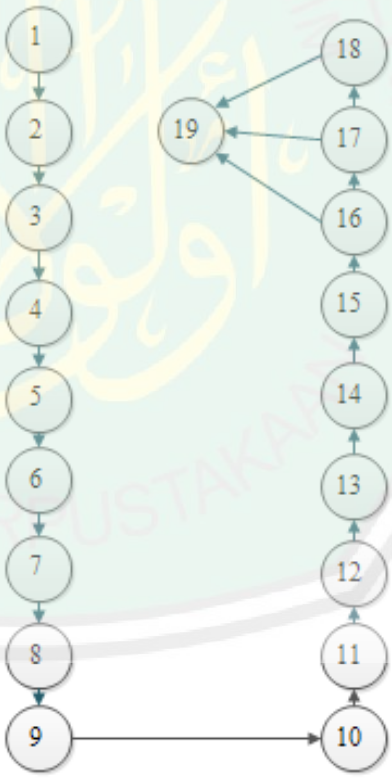
	Grant to Leave Ramp, Leave Ramp, Request Taxiing Clearance, End		Grant to Leave Ramp, Leave Ramp, Request Taxiing Clearance, Permission not Granted, End
1 2 3 4 5 6 7 8 9 10 11 14	Request for Pushback, Grant Pushback Clearance, Pushback, Request to Leave Ramp, Grant to Leave Ramp, Leave Ramp, Request Taxiing Clearance, Grant Taxiing Clearance, Taxiing, Aircraft on Runway, Request Departure Clearance, End	1 2 3 4 5 6 7 8 9 10 11 14	Request for Pushback, Grant Pushback Clearance, Pushback, Request to Leave Ramp, Grant to Leave Ramp, Leave Ramp, Request Taxiing Clearance, Grant Taxiing Clearance, Taxiing, Aircraft on Runway, Request Departure Clearance, Permission not Granted, End
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	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

Output aplikasi dan *output* artikel Rhmann (2016) dari data *Aircraft Departure Activity* menunjukkan hasil yang sama. Namun, *output* dari artikel sebelum *node End* ditambahi pesan "*Permission not Granted*" yang artinya bahwa *request* yang diminta dari sistem ini bisa saja ditolak. Dari

path di atas, *best path* ada pada *path* terakhir yaitu 1 2 3 4 5 6 7 8 9 10 11 12 13 14.

Pembahasan selanjutnya yaitu hasil *path* pada **sistem penilaian pembelajaran**. (1). *Output sequence diagram* perhitungan nilai, (2). *Output sequence* diagram mendapatkan nama matakuliah, (3). *Output sequence diagram* mendapatkan nama mahasiswa, dan (4). *Output sequence diagram* menampilkan *report*.

(1). *Output sequence diagram perhitungan nilai* pada sistem penilaian pembelajaran. Berikut Gambar 4.30 sebagai representasi *graph* pada perhitungan nilai.



Gambar 4.30. Representasi *Graph* Perhitungan Nilai pada Sistem Penilaian Pembelajaran

Untuk lebih jelas hasil pembangkit *test path* yang diujikan pada *sequence diagram* Perhitungan nilai Gambar 4.26 di atas, *Sequence*

Dependency Table, *node* pembentuk *Sequence Dependency Graph* dan *Output Test Path* akan ditampilkan pada Tabel 4.16 di bawah ini.

Tabel 4.16. *Output Sequence Dependency Table*, *Node* Pembentuk *Sequence Dependency Graph* dan *Test Path* dari *Sequence Diagram* Perhitungan Nilai pada Sistem penilaian Pembelajaran.

Sequence Dependency Table				***Nod.Pem.Sequence Dependency Graph***		***Test Path***
Symbol	**Activity Name**	**Sequence**	**Ordinal**	**Node1**	**Node2**	
1	setTaskw()	"1"	0	0	"1"	- 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 19
2	setFirstw()	"2"	1	1	"2"	
3	setLastw()	"3"	2	2	"3"	
4	setTask1()	"7"	6	6	"7"	- 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 19
5	setTask2()	"8"	7	7	"8"	
6	setTask3()	"9"	8	8	"9"	
7	setTask4()	"10"	9	9	"10"	
8	setTask5()	"11"	10	10	"11"	
9	setTask6()	"12"	11	11	"12"	- 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 19
10	setTask7()	"13"	12	12	"13"	
11	setFirst()	"14"	13	13	"14"	
12	setLast()	"15"	14	14	"15"	
13	getTaskw()	"4"	3	3	"4"	- 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
14	getFirstw()	"5"	4	4	"5"	
15	getLastw()	"6"	5	5	"6"	
16	countMean()	"16"	15	15	"16"	
17	gradeCount()	"17"	16	16	"17"	
18	gradeConv()	"18"	17	17	"18"	
19	End	"19"	15,16,17,18	15,16,17,18	"19"	

Berikut bentangan *path* dari hasil pengujian di atas :

Path 1 = **setTaskw() * setFirstw() * setLastw() * getTaskw() * getFirstw() * getLastw() * setTask1() * setTask2() * setTask3() * setTask4() * setTask5() * setTask6() * setTask7() * setFirst() * setLast() * End**

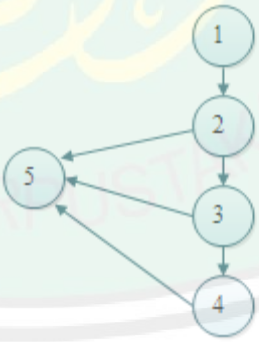
Path 2 = **setTaskw() * setFirstw() * setLastw() * getTaskw() * getFirstw() * getLastw() * setTask1() * setTask2() ***

setTask3() * setTask4() * setTask5() * setTask6() * setTask7() * setFirst() * setLast() * countMean() * End

Path 3 = **setTaskw() * setFirstw() * setLastw() * getTaskw() * getFirstw() * getLastw() * setTask1() * setTask2() * setTask3() * setTask4() * setTask5() * setTask6() * setTask7() * setFirst() * setLast() * countMean() * gradeCount() * End**

Path 4 = **setTaskw() * setFirstw() * setLastw() * getTaskw() * getFirstw() * getLastw() * setTask1() * setTask2() * setTask3() * setTask4() * setTask5() * setTask6() * setTask7() * setFirst() * setLast() * countMean() * gradeCount() * gradeConv() * End**

Gambar 4.31. Representasi Graf Mendapatkan Nama Matakuliah dan Mendapatkan Nama Mahasiswa pada Sistem Penilaian Pembelajaran



Gambar 4.32. Representasi Graf Menampilkan *Report* pada Sistem Penilaian Pembelajaran

Selanjutnya (2). O*utput* pengujian *sequence diagram* mendapatkan nama matakuliah akan dijelaskan pada Tabel 4.17, (3). *Output* pengujian *sequence diagram* mendapatkan nama mahasiswa akan dijelaskan pada

Tabel 4.18 dan (4). *Output* pengujian *sequence diagram* menampilkan *report* akan dijelaskan pada Tabel 4.19 di bawah ini.

Tabel 4.17. *Output SDT, Node* Pembentuk *SDG* dan *Test Path* dari *Sequence Diagram* Mendapatkan Nama Matakuliah pada Sistem penilaian Pembelajaran.

Sequence Dependency Table				***Nod.Pem.Sequence Dependency Graph***		***Test Path***
Symbol	**Activity Name**	**Sequence**	**Ordinal**	**Node1**	**Node2**	
1	setKodeMK()	"0"	1	1	"0"	- 1 2 3 4
2	getOneItem()	"1"	2	2	"1"	
3	getNameMK()	"2"	3	3	"2"	
4	End()	"3"	4	4	"3"	

Tabel 4.18. *Output SDT, Node* Pembentuk *SDG* dan *Test Path* dari *Sequence Diagram* Mendapatkan Nama Mahasiswa pada Sistem penilaian Pembelajaran.

Sequence Dependency Table				***Nod.Pem.Sequence Dependency Graph***		***Test Path***
Symbol	**Activity Name**	**Sequence**	**Ordinal**	**Node1**	**Node2**	
1	setNim()	"0"	1	1	"0"	- 1 2 3 4
2	getOneItem()	"1"	2	2	"1"	
3	getName ()	"2"	3	3	"2"	
4	End()	"3"	4	4	"3"	

Tabel 4.19. *OutputSDT, Node* Pembentuk *SDG* dan *Test Path* dari *Sequence Diagram* Menampilkan Report pada Sistem penilaian Pembelajaran.

Sequence Dependency Table				***Nod.Pem.Sequence Dependency Graph***		***Test Path***
Symbol	**Activity Name**	**Sequence**	**Ordinal**	**Node1**	**Node2**	
1	setSubject()	"1"	0	0	"1"	- 1 2 5
2	printReport()	"4"	3	3	"4"	- 1 2 3 5
3	setClass ()	"2"	1	1	"2"	- 1 2 3 4 5
4	showReport()	"3"	2	2	"3"	
5	End	"5"	2,3,4	2,3,4	"5"	

Bentangan *path* dari Tabel 4.17 :

Path 1 = **setKodeMK() * getOneItem() * getNameMK() * End**

Bentangan *path* dari Tabel 4.18 :

Path 1 = **getNim() * getOneItem() * getName() * End**

Bentangan *path* dari Tabel 4.19 :

Path 1 = **setSubject() * setClass() * End**

Path 2 = **setSubject() * setClass() * showReport * End**

Path 3 = **setSubject() * setClass() * showReport * printReport() * End**

Output pengujian yang telah dilakukan dari *sequence diagram* perhitungan nilai ditunjukkan pada Tabel 4.16 di atas. Dari tabel di atas, dapat dilihat jika *test path* yang dapat dilalui terdapat empat *path* atau jalur dengan *best path* "1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23" pada *path* ke empat sebagai *path* yang paling sempurna dilalui pengujian pada perhitungan nilai program sistem penilaian pembelajaran.

Output pengujian pada *sequence diagram* mendapatkan nama matakuliah ditunjukkan pada Tabel 4.17 dan *sequence diagram* mendapatkan nama mahasiswa yang ditunjukkan pada Tabel 4.18 di atas, dari kedua tabel di atas dapat diketahui bahwa hasil *generate* dari *sequence diagram* hanya menunjukkan satu jalur "1 2 3 4" karena kedua *sequence diagram* ini memiliki kinerja satu jalur dan termasuk dalam *sequence diagram* dalam bentuk sederhana. Sedangkan *Output* pengujian pada *sequence diagram* menampilkan *report* yang ditunjukkan pada Tabel 4.19 di atas, menghasilkan tiga *path* dengan *path* sempurna "1 2 3 4 5".

BAB V

PENUTUP

A. Kesimpulan

Dari hasil penelitian yang telah lakukan, dapat disimpulkan bahwa pada skripsi ini telah dibuat aplikasi yang dapat membangkitkan *test case* dari *sequence diagram* dengan *output* yang sama ketika dilakukan percobaan data dari beberapa artikel dan hasilnya dibandingkan dengan *output* pada artikel terkait yakni Shanti et al (2012), Priya et al (2013) dan Rhmann (2016). Sedangkan *output* dari sistem penilaian pembelajaran menghasilkan sembilan *path* dari empat *input sequence diagram*.

B. Saran

Berikut beberapa saran yang dapat dijadikan masukkan untuk penelitian yang akan dilakukan selanjutnya :

1. Satu hal yang dilakukan secara manual dari program konversi file xml yaitu mengambil nilai-nilai ordinal berdasarkan nilai tag *synchronization* "Return" kemudian dituliskan pada nilai tag *ordinal* pada *Symbol id* erakhir diharapkan dapat bekerja secara otomatis.
2. Kekurangan pada *output* program, adanya angka 1 yang ikut tercetak pada saat *path* di*generate*.
3. Pengujian dapat menggunakan dan mengkombinasikan diagram UML lain misalnya adalah *class diagram, activitydiagram* atau *state chart* agar *test case* yang dihasilkan lebih detail dan lengkap.

DAFTAR PUSTAKA

Abdullah, Dr. 2005. *Tafsir Ibnu Katsir Jilid 8*. Tt. Bogor: Pustaka Imam Asy-Syafi'i.

Adi Nugroho. 2010. *Rekayasa Perangkat Lunak Berbasis Objek dengan Metode USDP*. Andi. Yogyakarta.

Agarwal, B. B.; Tayal, S.P.; Gupta, M. 2010, *Software Engineering &Testing*, Johanes and Bartlett Publishers, Sudbury, Massachusetts.

Al Bahra. 2006. *Rekayassa Perangkat Lunak*. Yogyakarta: Graha Ilmu.

Barakbah, Ali Ridho. 2013. *Logika dan Algoritma*. Departemen Teknik Informatika dan Komputer Politeknik Elektronika Negeri Surabaya.

Bartlett, Jake. 2015. *"What's the Difference between a Test Case and Test Scenario?"*. https://blog.testlodge.com/whats-the-difference-between-test-case-and-test-scenario/. Diakses tanggal 17 Februari 2017 12:05

Bennet, Roger dan Anna Barkensjo. 2005."*Relationship Quality, relationship Marketing, and Client Perceptions of The Level of Service Quality of Charitable Organisation*". International Journal of Service Industry Management. Vol.16. No.1:81-106.

Dalai, Swagatika et al. 2011. International Journal of Advanced Computer Science and Applications : *Test case Generation For Concurrent Object-Oriented Systems Using Combinational Uml Models*. Vol. 3, No.5.

Fowler, Martin. (2004*). UML Distilled (Panduan Singkat bahasa Pemodelan Objek Standar)*(Vol. 3). Yogyakarta: ANDI.

Informatics Lab, *Modul Praktikum RPL Teknik Berorientasi Objek*, Bandung: Laboratorium Informatika, Telkom Engineering School, 2013/2014.

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

Khurana, Namita., Chillar, R.S. 2014. *Literature Review of test Case Generation Techniques for Object Oriented System*. International Journal of Computer Application (0975-8887). Vol.105-No.15

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

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 Systems : *Automatic Test case Generation using Sequence diagram*. ISSN : 2249-0868.

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

Pribadi, Octa. 2015. TIMES *: Maze Generator Dengan Menggunakan Algoritma Depth-First-Search*. Vol. IV No 1 : 1-5.

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

Putri, Tafifa Redita et al. 2015. *Pembangkit Kasus Uji untuk Pengujian Aplikasi Berbasis Sequence diagram*. Universitas Telkom : Bandung.

Rhmann, Wasiur., Saxena, 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.

Sarma, Monalisa et al. 2007. IEEE : *Automatic Test case Generation from UML Sequence diagrams*.

Shanthi, A.V.K et al. 2012. International Conference on Software and Computer Applications : *Automated Test cases Generation from UML Sequence diagram*. vol. 41.

Sommerville. 2003. *Software Engineering (Rekayasa Perangkat Lunak)/Edisi 6/ Jilid 2*. Jakarta: Erlangga.

Timothy, Lethbridge dan Robert, Laganiere. 2002. *Object Oriented Software Engineering: Pratical Software Development Using UML and Java*. Mc Graw Hill-Education, UK.

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

Utting, M., Legeard, B. 2007. *Pactical Model-based testing: A Tools Approach*. Morgan Kaufmann Publisher Inc : San Francisco.USA.

http://www.jelajahinternet.com/2014/10/pengertian-rational-rose-komponen-gui.html. Diakses tanggal 17 Februari 2017 12:30.

http://www.softwaretestingclass.com/what-is-automation-testing/. Diakses tanggal 17 Februari 2017 12:50

Wikipedia. 2016. "Pemrograman Berorientasi Objek". https://id.wikipedia.org/wiki/Pemrograman_berorientasi_objek. Diakses tanggal 08 Agustus 2017 10.30