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

SKRIPSI

Oleh:

NI'AMAH
NIM. 12650013

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERIMAULANA MALIK IBRAHIM MALANG
2018

PEMBANGKIT *TEST CASE* (KASUS UJI) MENGGUNAKAN MODEL *UML* (*UNIFIED MODELING LANGUAGE*) *ACTIVITY 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:

NI'AMAH
NIM. 12650013

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERIMAULANA MALIK IBRAHIM MALANG
2018

HALAMAN PERSETUJUAN

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

SKRIPSI

Oleh :
NI'AMAH
NIM. 12650024

Telah disetujui oleh :

Pembimbing I	Pembimbing II
Fatchurrochman, M. Kom	**Suprivono, M. Kom**
NIP. 19700731 200501 1 002	NIDT. 19841010 20160801 1 078

Tanggal, 21 Desember 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 :
NI'AMAH
NIM. 12650013

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

Tanggal, 08 Januari 2018

Susunan Dewan Penguji :		**Tanda Tangan**
1. **Penguji Utama**	: **Dr. M. Amin Hariyadi, M.T** **NIP. 19670118 200501 1 001**	( )
2. **Ketua Penguji**	: **Ainatul Mardhiyah, M.CS** **NIDT. 19860330 20160801 2 075**	( )
3. **Sekretaris Penguji**	: **Fatchurrochman, M. Kom** **NIP. 19700731 200501 1 002**	( )
4. **Anggota Penguji**	: **Supriyono, M. Kom** **NIDT. 19841010 20160801 1 078**	( )

Mengetahui dan Mengesahkan
Ketua Jurusan Teknik Informatika

Dr. Cahyo Crysdian
NIP. 19740424 200901 1 008

iv

SURAT PERNYATAAN ORISINALITAS PENELITIAN

Saya yang bertanda tangan di bawah ini:

Nama : NI'AMAH

NIM : 12650013

Fakultas / Jurusan: Sains dan Teknologi / Teknik Informatika

Judul Penelitian : Pembangkit *Test case*(Kasus Uji) Menggunakan Model *UML* (*Unififed Modeling Language*) *Activity 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, 22 Desember 2017

Hormat saya

Ni'amah

NIM. 12650013

HALAMAN MOTTO

MOTTO

لاَ يَكْتُب إنْسَانٌ كِتَاباً فِي يَوْمِهِ إلاَّ قَالَ فِي غَدِهِ: "لَو غُيِّرَ هَذَا لَكَانَ أَحْسَن، وَلَو زِيْدَ هَذَا لَكَانَ يُسْتَحْسَن، وَلَو قُدِّمَ هَذَا لَكَانَ أفْضَل، وَلَو تَرَكَ هَذَا لَكَانَ أجْمَل" وَهَذَا مِنْ أعْظَمِ العِبَرِ وَهُوَ دَلِيْلٌ عَلَي إسْتِيْلاَءِ النَقْصِ عَلَي جُمْلَةِ البَشَرِ.

من رسالة القاضى عبد الرجم البيسانى (ت 596 هـ/ 1199 م)

Tak ada seorang penulis buku, tanpa esok hari ia berkata:

seandainya yang ini diganti, akan lebih baik;

seandainya yang ini ditambahkan, akan lebih pantas;

seandainya yang itu dimajukan, akan lebih disukai;

dan seandainya yang itu ditinggalkan, akan lebih indah.

Hal demikian merupakan suatu hikmah yang besar, dan sebagai bukti nyata akan kekurangan manusia.

(al-Qadi 'abd al-Rajam al-Bîsâny, w.596 H/1199 M)

HALAMAN PERSEMBAHAN:

Alhamdulillahirobbilalamin, Skripsi ini ku persembahkan untuk kedua Orangtuaku (Bapak H. Abdul Rozaq dan Ibu Siti Aminah), Kakak-kakakku serta ponakanku, juga untuk mas Herbylian Eka Maulana, Sahabat-sahabatku, Teman-temanku, Dosen-dosenku, Almamaterku, dan semua orang yang ku sayang, yang telah memberiku semangat yang tak ternilai jumlah dan ukurannya,……

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*) *ACTIVITY DIAGRAM* (STUDI KASUS *SISTEM PENILAIAN PEMBELAJARAN*)"** ini akhirnya dapat 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 H. Abdul Rozaq dan Ibu SitiAminah 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 Bapak Supriyono, M.Kom, 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 mbak zulfa, mbak rurin, mbak faizah dan mas saiful juga keluarga besar yang selalu memberikan dukungan dan nasehat-nasehatnya yang sangat bermanfaat bagi penulis.
8. Untuk Mas terkasih Herbylian Eka Maulana yang selalu mendukung dan mendampingi bagi penulis.
9. Sahabatku Mimin Fitrah Nur yang selalu menyemangati dan memberikan nasehat yang bermanfaat bagi penulis.
10. Teman teman kos popo laila, nana, aulia finda, windy dan teman sekosan terima kasih selalu memberikan semangat kepada penulis.
11. Teman serta sahabat terbaik Rury, cendana, arni, evi zakiah, neng pipit, dan cindy, serta teman teman dari club YVC-I yang telah memberi dukungan dan semangat bagi penulis
12. 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.

13. 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.

semoga skripsi ini bisa memberikan manfaat bagipengembangan ilmu pengetahuan.

Malang, 19 Desember 2017

Ni'amah

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

ABSTRAK

Ni'amah. 2017. *Pembangkit Test Case (Kasus Uji) Otomatis Menggunakan Model UML (Unified Modelling Language) Activity Diagram (Studi Kasus Sistem Penilaian Pembelajaran).* Skripsi, Program Sarjana Teknik Informatika, Universitas Islam Negeri Maulana Malik Ibrahim Malang. Pembimbing: (I) Fatchurrohman, M. Kom., (II) Supriyono, M.Kom.

Kata Kunci: Pembangkitan *Test Case*, *Activity Diagram* (UML), *Depth First Search* (DFS), *Path*, Sistem Penilaian Pembelajaran.

Pengujian adalah suatu proses eksekusi program yang bertujuan untuk menemukan kesalahan pada aplikasi. Penelitian ini membangkitkan *test case* dengan UML *activity diagram*. Uji coba contoh diambil pada *ATM*, *library management* dan uji coba aplikasi sistem penilaian pembelajaran.

Membangkitkan *test case* dibuat dari *activity diagram*, membuat *Activity Dependency Table*, membuat *Activity Dependency Graph* dari tabel tersebut mendapatkan path menggunakan metode *depth first search* (DFS). Aplikasi penilaian pembelajaran diharapkan dapat digunakan untuk memberikan penilaian pada matakuliah di perguruan tinggi. Hasil path uji coba dan hasil path pada aplikasi adalah valid.*Activity diagram* penilaian pembelajaran, terdapat lima *activity diagram* yang menghasilkan *path* yang dapat digunakan sebagai *test case*.

ABSTRACT

Ni'amah. 2017. Generating Automatic Test Case Using UML Model (Unified Modeling Language) of Activity 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) Supriyono, M.Kom.

Keywords: generating the Test Case, Activity Diagram (UML), Depth First Search (DFS), Path, Learning Assessment System.

Testing is a program execution process that aims to find errors in the application. This research generates test cases with UML activity diagrams. Trial samples were taken at ATMs, library management and trial appraisal of the learning appraisal system.

Generating test cases created from activity diagrams, creating Activity Dependency Table, making Activity Dependency Graph from the table get path using depth first search method (DFS). The appraisal of learning appraisal is expected to be used to provide an assessment of the course at the college. The result of the test path and the result of the path in the application is valid. Activity diagram of the learning appraisal, there are five activity diagrams that produce the path that can be used as test case

ملخص البحث

نعمة. 2017. توليد حالة الاختبار الآلي باستخدام نموذج *UML* (*Unified Modeling Language*) المخطط النشاط (دراسة حالة لنظام تقييم التعلم). البحث الجامعي، شعبة المعلوماتية، جامعة مولانا مالك إبراهيم الاسلامية الحكومية مالانج. المشرف الاول: فتح الرحمن، الماجستير، والمشرف الثانى: سوفريونو، الماجستير

الكلمات الرئيسية: توليد حالة الاختبار ، مخطط النشاط (UML)، البحث العمق الأول (DFS)، المسار(*Path*) نظام تقييم التعلم

البحث هذا يثير .التطبيق في خطأ على العثور إلى يهدف الذي البرنامج لتنفيذ عملية هو الاختبار الصراف أجهزة على المتخذة العينة اختبار .نشاط UML ل تخطيطي رسم مع اختبار حالة .للتعلم التطبيق لاختبار التقييم ونظام المكتبة وإدارة الآلي،

،"الجدول تبعية نشاط" وخلق للأنشطة، التخطيطية الرسومات من إنشاؤها تم الاختبار حالة بعث البحث عمق باستخدام الجدول على الحصول مسار أسلوب من "النشاط تبعية بياني رسم" إنشاء في الدورات هذه على حكم لإعطاء لاستخدامها المتوقع التقييم لدراسة تطبيقات .(DFS) الأولى للتعلم، للتقييم للنشاط تخطيطي رسم .صالحاً التطبيق في الناتج المسار ومسار الاختبار نتائج .كلية .اختبار كحالة استخدامه يمكن الذي مسار بإنشاء يقوم الذي النشاط مخططات خمسة وهناك

BAB I

PENDAHULUAN

1.1 Latar Belakang

Pengujian adalah suatu proses eksekusi program yang bertujuan untuk menemukan kesalahan (Berard,1994). Pengujian sebaiknya menemukan kesalahan yang tidak disengaja dan pengujian dinyatakan sukses jika berhasil memperbaiki kesalahan tersebut. Selain itu, pengujian juga bertujuan untuk menunjukkan kesesuaian fungsi-fungsi perangkat lunak dengan spesifikasinya. Sebuah perangkat lunak dinyatakan gagal, jika perangkat lunak tersebut tidak memenuhi spesifikasi (Fournier, Cs, 2009).

Pada awalnya pengujian merupakan aktifitas yang tidak hanya bertujuan untuk menemukan *error* tapi juga bertujuan untuk melakukan koreksi dan menghilangkannya sehingga pembahasan masalah pengujian saat itu lebih banyak kearah *debugging*, serta kesulitan dalam mengkoreksi dan menghilangkan error. Sejak konferensi pertama tentang pengujian *software*, yang diadakan pada bulan Juni 1972 di University of North Carolina, mulai banyak konferensi dan workshop yang bertemakan tentang kualitas, reliabilitas dan rekayasa s*oftware*, dimana secara bertahap telah memasukan disiplin tentang pengujian sebagai elemen yang terorganisasi dalam teknologi *software* (Bezeir, 2004).

Ada pun tujuan pengujian perangkat lunak itu sendiri yaitu untuk menemukan kesalahan yang menyebabkan perangkat lunak yang telah dibangun gagal. Selain tujuan itu, pengujian perangkat lunak juga bertujuan untuk

memperoleh produk yang berkualitas memberikan produktivitas tinggi (Oscar Pastor, Cs, 2007).

Testing adalah metode verifikasi perangkat lunak di mana programmer menguji suatu program untuk menguji kelayakan suatu unit program. Unit testing ini fokus pada verifikasi unit terkecil pada desain perangkat lunak (komponen atau modul perangkat lunak). Karena dalam sebuah perangkat lunak banyak memiliki unit-unit kecil maka untuk mengujinya biasanya dibuat program kecil atau main program) untuk menguji unit-unit perangkat lunak (feredi, 2016).

Test case adalah urutan langkah-langkah, *substep*, dan tindakan-tindakan lain yang dilakukan secara bertahap atau kombinasi urutan, yang membuat kondisi pengujian yang dirancang untuk di evaluasi *User requirement* harus diketahui sebelum merancang *test case*. *Test case* yang baik adalah mempunyai probabilitas tinggi untuk menemukan error yang tidak diketemukan. (Black, 2009).

Dalam Al-Qur'an juga dijelaskan pada Surah Al Ankabut ayat 2 -3 :

أَحَسِبَ النَّاسُ أَنْ يُتْرَكُوْا أَنْ يَقُولُوا آمَنَّا وَهُمْ لاَ يُفْتَنُونَ (٢) وَلَقَدْ فَتَنَّا الَّذِيْنَ مِنْ قَبْلِهِمْ فَلَيَعْلَمَنَّ اللَّهُ الَّذِيْنَ صَدَقُوْا وَلَيَعْلَمَنَّ الْكَاذِبِيْنَ (٣)

Artinya:*" Apa manusia itu mengira bahwa mereka dibiarkan (saja) mengatakan : "Kami telah beriman", sedang mereka tidak diuji lagi?. Dan sesungguhnya kami telah menguji orang-orang sebelum mereka, maka sesungguhnya Alloh mengetahui orang-orang yang benar dan sesungguhnya Dia mengetahui orang-orang yang dusta.". (QS. Al-Ankabut 29:02-03).*

Pada tafsir Ibn Katsir Jilid VI menerangkan Firman Alloh Ta'ala, أَحَسِبَ النَّاسُ أَنْ يُتْرَكُوْا أَنْ يَقُولُوا آمَنَّا وَهُمْ لاَ يُفْتَنُونَ *"Apakah manusia itu mengira bahwa mereka dibiarkan saja mengatakan: 'Kami telah beriman', sedang mereka tidak diuji lagi,"* adalah bentuk *istifham inkari* (pertanyaan yang bersifat mengingkari). Maknanya, bahwa Alloh SWT harus menguji hamba-hamba-Nya yang beriman sesuai keimana yang mereka miliki, sebagaimana dijelaskan dalam hadits shahih:

((أَشَدُّ النَّاسِ بَلاَءً الأَنْبِيَاءُ، ثُمَّ الصَّالِحُوْنَ ثُمَّ الْأَمْثَلُ فَالْأَمْثَلُ، يُبْتَلَى الرَّجُلُ عَلَى حَسْبِ دِيْنِهِ فَإِنْ كَانَ دِيْنِهِ صَلاَبَةٌ زِيْدَ لَهُ فِى الْبَلاَءِ.))

Artinya: *"Manusia yang paling berat ujianya adalah para Nabi, kemudian orang-orang shalih, kemudian yang semisal dan seterusnya. Seseorang diuji sesuai dengan agamanya. Jika agamanya semakin kuat, semakin bertambah pula ujiannya."*

Ayat ini sebagaimana firman Alloh Ta'ala : **(أَمْ حَسِبْتُمْ أَنْ تَدْخُلُوا الْجَنَّةَ وَمَا يَعْلَمِ اللَّهُ الَّذِيْنَ جَاهَدُوْا مِنْكُمْ وَلَيَعْلَمَ الصَّابِرِيْنَ)** *"Apakah kamu yakin akan masuk Surga, padahal belum nyata bagi Alloh orang-orang yang berjihad di antaramu, dan belum nyata orang-orang yang sabar."*(QS. Al-Imran:142) Untuk itu, di dalam ayat ini Dia berfirman : وَلَقَدْ فَتَنَّا الَّذِيْنَ مِنْ قَبْلِهِمْ فَلَيَعْلَمَنَّ اللَّهُ الَّذِيْنَ صَدَقُوْا وَلَيَعْلَمَنَّ الْكَاذِبِيْنَ *"Dan sesungguhnya kami telah menguji orang-orang sebelum mereka, maka sesungguhnya Alloh mengetahui orang-orang yang benar dan sesungguhnya Dia mengetahui orang-orang yang dusta."* Yaitu orang-orang yang jujur dalam pengakuan keimanannya dari orang-orang yang dusta dalam perkataan dan pengakuannya. Alloh SWT Maha Mengetahui apa yang telah ada dan apa yang

akan ada, apa yang belum ada seandainya ada dan bagaimana adanya. Ini merupakan sesuatu yang disepakati oleh para imam Ahlusunnah wal- Jama'ah. Ini pula yang dikatakan Ibnu 'Abbas dan selainnya ra. Pada contoh firman Alloh : إِلَّا لِنَعْلَمَ *"Kecuali agar Kami mengetahui,"* (QS. Al-Baqarah:143). Yaitu, kecuali agar kami melihat. hal itu diebabkan bahwa penglihatan berkaitan dengan sesuatu yang ada, sedangkan pengetahuan lebih umum dari pada penglihatan, karena ia berkaitan dengan sesuatu yang tidak ada dan sesuatu yang ada.

Jika dikaitkan dengan nash lainnya, ujian yang diberikan Allah SWT itu tidak selalu dalam bentuk yang berat dan dibenci. Ada juga ujian yang menyenang-kan sebagiamana dalam firman-Nya: Kami akan menguji kamu dengan keburukan dan kebaikan sebagai cobaan (yang sebenar-benarnya) (TQS al-Anbiya' [21]: 35).

UML activity diagram digambarkan dengan permodelan aspek dinamis dan manual. Penggunaan *activity diagram* ini untuk membuat waktu lebih relatif dengan pendekatan yang lebih praktis dan efisien waktu (A. Abdurazik and J. Offutt, 2000)

Membangkitkan *test case* dilakukan langkah pertama adalah berikut, yaitu membuat *activity diagram*. Langkah kedua adalah membuat *Activity Dependency Table*. Langkah ketika yaitu membangun *Activity Dependency Graph* dari tabel tersebut mendapatkan path (Shanthi dan G. Mohan Kumar, 2012)

Peneliti yang akan dilakukan mengambil judul ***"Pembangkit Test Case (Kasus Uji) Menggunakan Model UML (Unified Modelling Language) Activity Diagram"*** sebagai penelitian yang akan dilakukan dalam skripsi ini.

1.2 Identifikasi Masalah

Berdasarkan latar belakang, maka rumusan masalah dalam penelitian ini adalah bagaimana membangkitkan *test case* dari *Activity Diagram* yang diterapkan pada aplikasi penilaian pembelajaran?

1.3 Tujuan Penelitian

Berdasarkan rumusan masalah maka tujuan penelitian ini adalah mendapatkan path yang akan digunakan pada proses pengujian perangkat lunak.

1.4 Manfaat Penelitian

Penelitian ini diharapkan dapat mempercepat pengujian perangkat lunak, karena path telah membangkitkan secara otomatis oleh *system*.

1.5 Batasan Penelitian

Batasan penelitian dalam skripsi ini adalah :

1. Pembuatan *Activity Diagram* menggunakan aplikasi *Rational Rose versi 2002*.
2. *Activity Diagram* yang digunakan untuk objek pengujian yaitu jurnal A.V.K. Shanthi dan G. Mohan Kumar (2012), Gargi Bhattacharjee dan Priya Pati (2014) dan ditambah sistem penilaian pembelajaran sebagai studi kasus dalam skripsi ini.

Hasil *test path* dalam bentuk rute yang dinotasikan dalam angka.

1.6 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 *test case* dengan model UML *Activity Diagram* sebagai pengujian yang diterapkan dalam program berbasis berorientasi objek dengan 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

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 *object oriented* dengan menggunakan kombinasi model UML (*Unified Modeling Language*).

2.1 UML (Unified Modeling Language)

Menurut Hend (2006:5) *Unified Modeling Language* (UML) adalah bahasa yang telah telah menjadi standard untuk visualisasi, menetapkan, membangun dan mendokumentasikan artifak suatu sistem perangkat lunak. UML berorientasi objek menerapkan banyak level abstraksi, tidak bergantung proses pengembangan, tidak bergantung bahasa dan teknologi. Pemaduan beberapa notasi di beragam metodologi usaha bersama dari banyak pihak, di dukung oleh kakas- kakas yang di integrasikan lewat XML . Standar UML di kelola oleh OMG *(Object Management Group).*

Menurut Nugroho (2010:10), Sesungguhnya tidak ada batasan yag tegas diantara berbagai konsep dan konstruksi dalam UML, tetapi untuk menyederhanakannya, kita membagi sejumlah besar konsep dan dalam UML menjadi beberapa *view.* Suatu *view* sendiri pada dasarnya merupakan sejumlah konstruksi pemodelan UML yang merepresentasikan suatu aspek tertentu dari sistem atau perangkat lunak yang sedang kita kembangkan. Pada peringkat paling atas, *view-view* sesungguhnya dapat dibagi menjadi tiga area utama, yaitu:

klasifikasi struktural (*structural classification*), perilaku dinamis (*dinamic behaviour*), serta pengolahan atau manajemen model (*model management*).

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. Jenis-jenis diagram UML (*Unified Modeling Language*) adalah :

Menurut Henderi (2008:5), Berikut ini adalah definisi mengenai 5 diagram UML :

1. *Use Case* Diagram secara grafis menggambarkan interaksi antara sistem, sistem eksternal dan pengguna. Dengan kata lain *use case* diagram secara grafis mendeskripsikan siapa yang akan menggunakan sistem dan dalam cara apa pengguna (user) mengharapkan interaksi dengan sistem itu. *Use case* secara naratif digunakan untuk secara tekstual menggambarkan sekuensi langkah-langkah dari setiap interaksi.

2. *Class Diagram* menggambarkan struktur *object* sistem. Diagram ini menunjukkan class *object* yang menyusun sistem dan juga hubungan antara class object tersebut.

3. *Sequence Diagram* secara grafis menggambarkan bagaimana objek berinteraksi dengan satu sama lain melalui pesan pada sekuensi sebuah *use case* atau operasi.

4. *State Chart Diagram* digunakan untuk memodelkan behaviour objek khusus yang dinamis. Diagram ini mengilustrasikan siklus hidup objek berbagai keadaan yang dapat diasumsikan oleh objek dan *event-event* (kejadian) yang menyebabkan objek beralih dari satu *state* ke *state* yang lain.

Activity Diagram secara grafis digunakan untuk menggambarkan rangkaian aliran aktivitas baik proses bisnis maupun *use case*. *Activity diagram* dapat juga digunakan untuk memodelkan action yang akan dilakukan saat sebuah operasi dieksekusi, dan memodelkan hasil dari action tersebut.

2.2 *Activity Diagram*

Activity diagram menurut Martin Fowler (2005) adalah teknik untuk menggambarkan logika prosedural, proses bisnis, dan jalur kerja. Dalam beberapa hal, *activity diagram* memainkan peran mirip diagram alir, tetapi perbedaan prinsip antara notasi diagram alir adalah activity diagram mendukung *behavior* paralel. Node pada sebuah *activity diagram* disebut sebagai *action*, sehingga diagram tersebut menampilkan sebuah *activity* yang tersusun dari *action*.

1. Komponen-komponen *Activity Diagram*

   Komponen-komponen yang ada pada *activity diagram* adalah sebagai berikut:

   a. *Nodes* (initial dan final)

   Adalah simbol untuk memulai (*initial*) dan mengakhiri (*final*) suatu *activity diagram*.

   b. *Activity* (aktifitas)

Adalah proses komputasi yang bisa berupa kata kerja atau ekspresi dan bersifat atomiak atau todak dapat didekomposisi

c. *Flow*

Adalah awal dari proses yang paralel dan mampu menggambarkan aktivitas yang mungkin terjadi secara concurrent.

d. *Join*

Adalah akhir dari suatu proses paralel.

e. *Decision*

Adalah pilihan untuk mengambil keputusan.

f. *Partition*

Digunakan untuk menjelaskan siapa yang melakukan aktivitas dalam activity diagram. Untuk melakukan partisi dapat dilakukan dengan menggunakan *Swim Lane*.

g. *Signal*

Adalah tanda untuk memulai sebuah aktivitas.

2. Elemen-elemen pada *activity diagram*

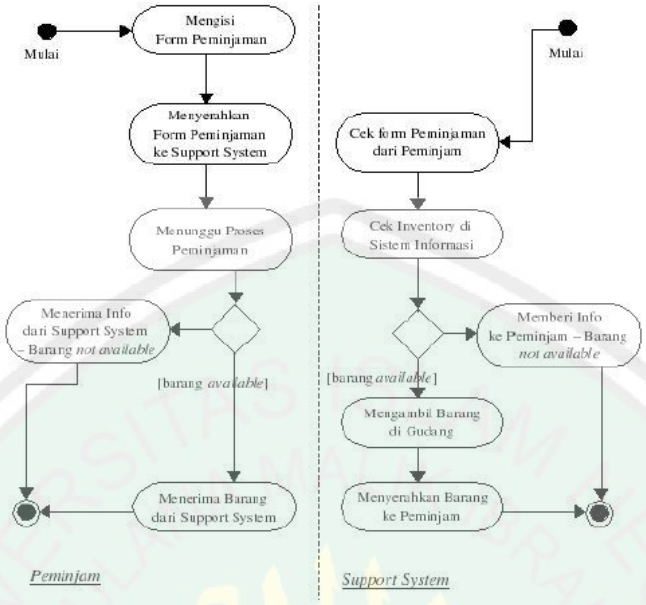
Pada Tabel 2.1 adalah tabel mengenai simbol-simbol *activity diagram* beserta penjelasannya.

Tabel 2.1 simbol-simbol *activity diagram* (Ariel, 2012)

No	Gambar	Nama	Keterangan
1.		Status awal	Status awal aktivitas sistem, sebuah diagram aktivitas memiliki sebuah status awal

2.		Aktivitas	Aktivitas yang dilakukan sistem, aktivitas biasanya diawali dengan kata kerja
3.		Percabangan / decision	Asosiasi percabangan dimana jika ada pilihan aktivitas lebih dari satu
4.		Penggabungan / join	Asosiasi penggabungan dimana lebih dari satu aktivitas digabungkan menjadi satu
5.		Status akhir	Status akhir yang dilakukan sistem, sebuah diagram aktivitas memiliki sebuah status akhir
6.		Swimlane	Memisahkan organisasi bisnis yang bertanggung jawab terhadap aktivitas yang terjadi
7.		Fork	Digunakan untuk menunjukkan kegiatan yang dilakukan secara parallel
8.		Join	Digunakan untuk menunjukkan kegiatan yang digabungkan

Berikut adalah contoh *activity diagram* dari contoh kasus pembuatan peminjaman peralatan.

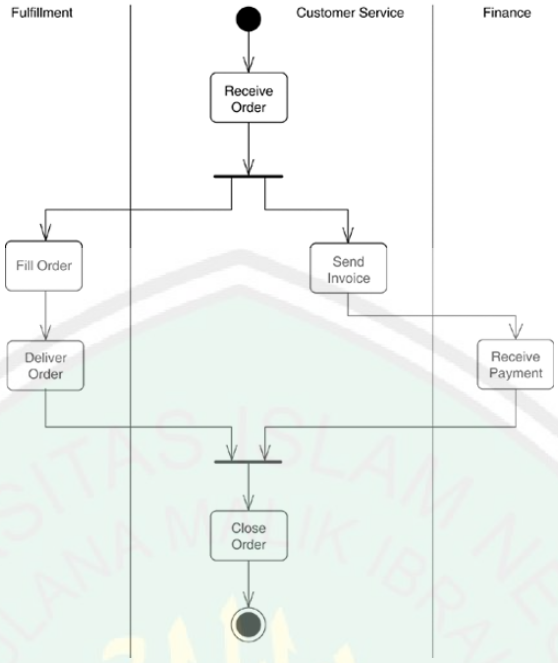


Gambar 2.1 *activity diagram* sistem peminjaman peralatan (Ariel, 2012)

Pada Gambar 2.1 dijelaskan bahwa contoh *activity diagram* tentang sub sistem peminjaman peralatan. Pertama yaitu pada mengisi form peminjaman lalu menyerah *support system* setalah itu menunggu proses peminjaman lalu di proses. Jika barang tersedia, maka menerima barang dan jika barang tidak tersedia akan menerima info.

Pada form *support system* mengecek peminjaman dari peminjam, mengecek dan diproses. Jika barang tersedia mengambil barang di gudang lalu menyerahkan pada peminjam, jika barang tida tersedia maka akan memberi info (Ariel.2012).

Berikut ini adalah contoh *activity diagram* menggunakan *Swim Lanes* pada Gambar 2.2.



Gambar 2.2 Contoh *Activity Diagram* Menggunakan *Swim Lanes* (Pratama, 2016)

Pada Gambar 2.2 jika menggunakan *swim lines*, *activity diagram* akan dibagi menjadi baris dan kolom sesuai dengan tanggung jawab objek - objek yang melakukan aktifitas. (Pratama, 2016).

2.3 Pengujian Perangkat Lunak

menurut Myers (1979) Pengujian adalah proses menjalankan program dengan maksud menemukan kesalahan. Definisi tersebut menyangkut kegiatan mulai dari cek kode yang dilakukan oleh seorang pemimpin tim untuk menjalankan percobaan dari perangkat lunak yang dilakukan oleh seorang rekan, serta tes yang dilakukan oleh suatu unit pengujian, semua bisa dianggap kegiatan pengujian. Terdapat cukup banyak pendekatan yang dilakukan untuk melakukan testing. Salah satu definisi testing adalah sebuah proses yang melakukan pertanyaan terhadap sebuah produk untuk dinilai di mana pertanyaan merupakan

segala sesuatu yang diberikan kepada produk sebagai pengujian. Sasaran pengujian *software testing* yaitu :

1. Menjalankan program untuk menemukan error.
2. *Test case* yang bagus adalah yang memiliki kemungkinan terbesar untuk menemukan error yang tersembunyi.
3. Pengujian yang sukses adalah yang berhasil menemukan error yang tersembunyi
4. Pengujian didesain secara sistematis untuk mencari kelas kesalahan yang berbeda
5. Pengujian dilakukan dalam waktu dan usaha minimum
6. Pengujian yang sukses adalah yang berhasil menemukan kesalahan dalam software, dan dapat menunjukkan reliabilitas software

Pengujian tidak dapat memperlihatkan tidak adanya kesalahan.

2.4 *Automated Testing*

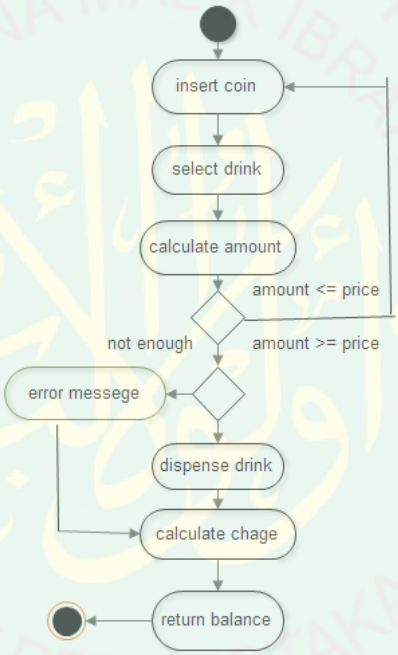
Automated testing merupakan proses pengujian yang digunakan untuk mempermudah proses dan dokumentasi pengujian, serta mengefektifkan proses pengeksekusian dan pengukuran pada pengujian (Novelia, 2008).

Otomatisasi testing akan sangat terasa manfaatnya (peningkatan efisiensi biaya dan efektifitas sumber daya) dalam regression testing. Terbatasnya waktu merupakan hambatan terbesar dalam melakukan regression testing, sehingga pada testing secara manual jumlah test cases untuk regression testing dibatasi hanya 10% dari jumlah test cases yang dilakukan pada awal testing. Berdasarkan pada studi yang dilakukan oleh Software Engineering Institute – Bellcore Study,

terdapat kecenderungan terjadinya defect/bug baru setelah dilakukan perubahan atau perbaikan pada sistem (lebih dari 60%) atau error baru akan muncul setiap 6 baris kode dirubah.

2.5 Pengertian *Test Case*

Test case merupakan salah satu komponen dokumentasi pengujian. *Test case* digunakan sebagai panduan bagi tester untuk melakukan pengujian suatu modul (Novelia, 2008). Berikut ini adalah contoh *activity diagram* dalam .



Gambar 2.3 *Activity diagram* SVM (Prafulla kumar behera,dkk, 2013)

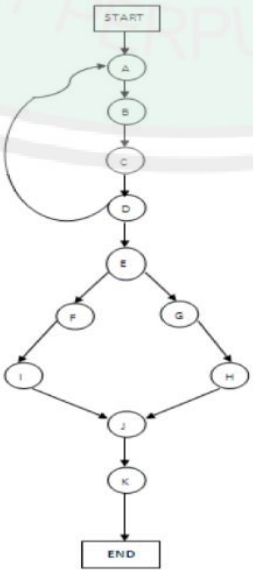
Pada Gambar 2.3 di jelaskan bahwa mengenai cara pengambilan minuman bersoda pada mesin pengambilan minum. Memasukkan koin atau uang logam pada mesin, lalu memilih minuman dan menjumlahkan harga. Jika uangnya tidak cukup maka memasukkan uang lagi, jika cukup maka pengambilan minuman. Minuman bisa diambil dan dihitung oleh mesin, lalu mesin akan kembali ke menu

awal. Apabila pesanan eror, maka mesin menghitung dan mengembalikan uang (Prafulla kumar behera,dkk, 2013)

Tabel 2.2 *Activity Dependency Table* (ADT) (Prafulla kumar behera,dkk, 2013)

Name of the node	Name of the activities	Predicate conditions
A	Insert coin	NULL
B	Select drink	NULL
C	Calculate amount	NULL
D	Amount < Price	C1 : Amount < Price
E	Amount >= *Price*	C2 : Amount >= Price
F	Item Available	D1 : Item Available
G	Item not Available	D2 : Item not Available
H	Error Message	NULL
I	Dispense drink	NULL
J	Calculate change	NULL
K	Return balance	NULL

Pada Tabel 2.2 di jelaskan *Activity Dependency Table* (ADT) diperoleh dari kegiatan *activity diagram* pengambilan minuman pada mesin SVM. *Activity Dependency Table* (ADT) yang dihasilkan adalah tabel terdiri yang terdiri nama *activity* dan *predicate conditions* di mana nama *activity* digunakan untuk menunjukkan node dalam diagram. ADT ini diperoleh dari *activity diagram* pengambilan minuman (Ranjita,dkk, 2013)



Gambar 2.4 *Activity Dependency Graph* (AGD) (Prafulla kumar behera,dkk, 2013)

Gambar 2.4 menunjukkan Jalur *test case* pada *Activity Dependency Graph* (AGD) dalam graph. Gambar ini menjelaskan tentang *graph* yang telah dihasilkan dari *Activity Dependency Table* (ADT) (Prafulla kumar behera,dkk, 2013).

AP_1 = A →B →C →D →A
AP_2 = A → B → C → D → E → F → I → J → K
AP_3 = A → B → C → D → E → G → H → J → k
AP_4 = A → B → C → D → A → B → C → D → E → F → I → J → K
AP_5 = A → B → C → D → A → B → C → D → E → G → H → J → k

Gambar 2.5 *Test Path Obtained* (Prafulla kumar behera,dkk, 2013)

Pada Gambar 2.5 alur *test path* akan dimulai dengan node awal sampai akhir. Dari awal node bergerak sepanjang ADT sampai mencapai keadaan akhir .jalur.

2.6 *Graph*

Teori *graf* merupakan sebuah pokok bahasan yang sudah tua usianya namun memiliki banyak terapan sampai saat ini. *Graf* digunakan untuk merepresentasikan objek*objek diskrit dan hubungan antara objek*objek tersebut. *Graf* pertama kali digunakan untuk memecahkan masalah jembatan *Königsberg* pada tahun 1736. Pada tahun tersebut, seorang matematikawan Swiss bernama L. *Euler* berhasil memecahkan masalah jembatan *Königsberg* tersebut. Lalu memodelkan masalah ini ke dalam bentuk *graf* dengan daratan (titik*titik yang

dihubungkan oleh jembatan) dimodelkan sebagai noktah atau vertex dan jembatan dinyatakan sebagai garis atau edge (Hidayatullah, 2015).

Menurut Siang (2002:187), suatu graf *G t*erdiri dari 2 himpunan yang berhingga, yaitu himpunan simpul-simpul tidak kosong *(V(G))* dan himpunan garis-garis (E(G)). Jadi, suatu graf Gadalah pasangan himpunan Vdan E, dituliskan *G= (V,E),* dengan *V* adalah suatu himpunan berhingga dan Eadalah suatu himpunan rusuk yang bersisian dengan *V*. Berikut adalah beberapa istilah yang sering digunakan dalam graf.

1. Gelang (*Loop)*

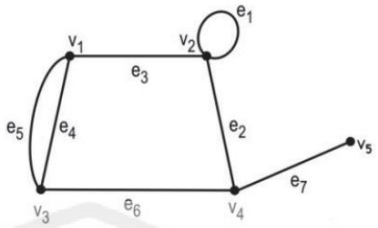
Menurut Munir (2005), suatu rusuk dikatakan gelang apabila ujung rusuknya berawal dan berakhir pada simpul yang sama.

2. Rusuk Ganda *(Multiple Edges)*

Pada sebuah *graf*, terdapat kemungkinan bahwa terdapat lebih dari satu rusuk yang bersisian dengan sepasang simpul. Rusuk tersebut dinamakan rusuk ganda.

3. Bertetangga *(Adjacent)*

Dua buah simpul pada graf tak berarah *G* dikatakan bertetangga bila keduanya terhubung langsung dengan sebuah rusuk. (Harju, 2012). Dengan kata lain, *u* bertetangga dengan vjika *(u, v)* adalah sebuah rusuk pada graf.



Gambar 2.6 *Graf A* (Siang, 2002:187)

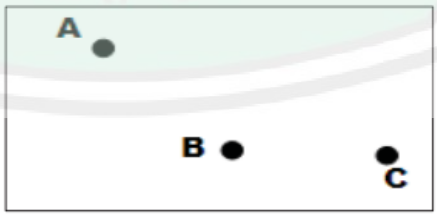
Pada Gambar 2.6 dijelaskan simpul v1bertetangga dengan simpul v2, e1 merupakan gelang, dan antara v1dan v3terdapat rusuk ganda e5dan e4.

a. Bersisian *(Incident)*

Untuk sembarang rusuk *e =(u, v),* rusuk edikatakan bersisian dengan simpul udan simpul *v.* Pada Gambar 2.6 rusuk *e7* bersisian dengan v4dan *v5*. Sedangkan *e2* tidak bersisiang dengan *v1* maupun *v2*.

b. *Graf* Kosong (*Null Graph* atau *Empty Graph*)

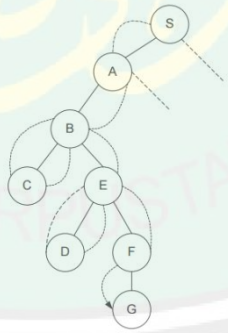
Graf kosong adalah *graf* yang himpunan rusuknya merupakan himpunan kosong. *Graf* kosong dapat dinotasikan dalam Nn, dimana nadalah banyaknya simpul yang di tunjukkan pada Gambar 2.8 di bawah ini.



Gambar 2.7 contoh Graf kosong (Siang, 2002:187)

2.7 Algoritma *Depth First Search* (DFS)

Pada algoritma DFS, pencarian dilakukan pada satu node dalam setiap level dari yang paling kiri. Jika pada level yang paling dalam solusi belum ditemukan, maka pencarian dilanjutkan pada node sebelah kanan. Node yang di kiri dapat dihapus dari memori. Jika pada level yang paling dalam belum ditemukan solusi, maka pencarian dilanjutkan ke level sebelumnya. Demikian seterusnya sampai ditemukannya solusi. Jika solusi ditemukan, maka tidak diperlukan proses backtracking (penelusuran untuk mendapatkan jalur yang diinginkan). Beberapa kelebihan dari algoritma DFS adalah pemakaian memori hanya sedikit karena hanya menyimpan lintasan yang aktif saja. Selain itu kelebihannya adalah jika solusi berada pada level yang paling dalam dan paling kiri, maka DFS akan menemukannya secara cepat. Misal suatu ruang keadaan masalah ditunjukkan dengan suatu seperti gambar 2.8 berikut ini :



Gambar 2.8 Tree untuk Algoritma *Depth First Search*

Dari Gambar 2.8 dalam pencarian menggunakan algoritma DFS, simpul-simpul yang paling dalam pada tree yang akan dicari paling awal. Sebagai contoh pada Gambar 2.3. Urutan pencarian keadaan awal (S) sampai keadaan tujuan (G) adalah dimulai dari node S, kemudian ke node A, kemudian ke node B, kemudian

ke node C, setelah itu akan melewati node B kembali dan menuju ke node E, selanjutnya akan menuju node D, setelah itu akan menuju node F setelah melewati node E, dan yang terakhir akan menuju node G. Adapun algoritma DFS berisi antara lain:

1. Bentuk satu elemen *Queue* yang terdiri dari *root node*.

2. *Until Queue empty* atau *goal* sudah dicapai, maka tentukan apakah elemen pertama dalam *Queue* adalah *goal node*.

   a. Jika elemen pertama adalah *goal node*, maka keluar. Universitas Sumatera Utara

   b. Jika elemen pertama tidak *goal*, maka remove elemen pertama dari *Queue* dan *add* anak elemen pertama.

3. Jika goal node sudah ditemukan maka sukses maka yang lain gagal.

2.8 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)

2.9 Pengujian Jalur (*Test Path*)

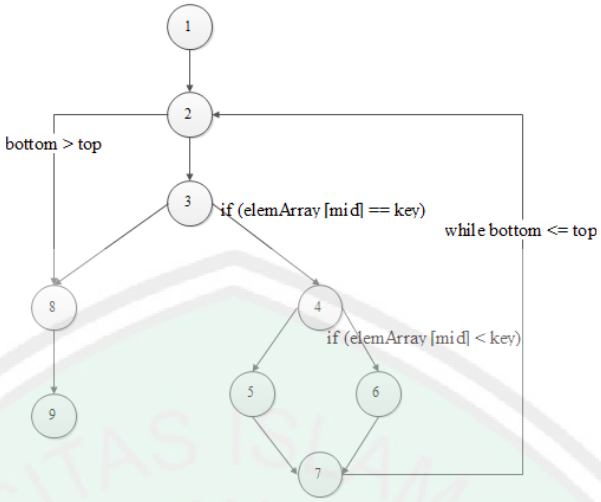
Pengujian jalur atau *path testing* adalah strategi pengujian *structural* yang bertujuan untuk melatih setiap jalur eksekusi independen melalui komponen atau program. Jika setiap jalur independen dieksekusi, maka semua *statement* pada komponen harus dieksekusi paling tidak satu kali. Lebih jauh lagi, 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 *flow graph* 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)

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 2.9. *Flow Graph* untuk Pengurutan *Binary Search* (Sommerville, 2003)

Flow graph untuk prosedur *binary search* sesuai *source code class BinSearch* ditunjukkan pada gambar 2.9. 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 2.9 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.

2.10 Pengujian

Pengujian berorientasi objek dilakukan dengan menggunakan metode pengujian *level class*. Metode ini dapat meimplementasikan dengan *membuat test case* dari berbagai *method* pada *class* berorientasi objek secara acak, atau pengujian partisi. Juga dapat dilakukan dengan membuat *test case* antar *class.*

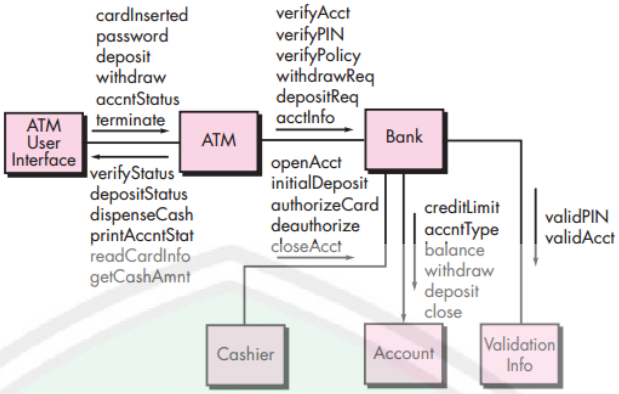
Maka pengujian yang akan dilakukan akan didasarkan pada diagram class yang telah di buat sebelmnya.

Dalam penelitian ini perlu dipastikan terlebih dahulu bahwa *diagram class* yang dibuat telah dapatdiwujudkan bentuk perogram komputer untuk meyakinkan pada tingkat tertentu model yang dibangun adalah model yang benar sesuai dengan kebutuhan yang telah dissusun pada awal pengembangan perangkat lunak.

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 2.10. *Diagram 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]*.

2.11 Penelitian Terkait

Software pengujian adalah fase terakhir dari siklus pengembangan. Peran penting dalam pengembangan perangkat lunak adalah Pengujian. Dalam industri perangkat lunak saat ini, desain tes *software* sebagian besar didasarkan pada keahlian tester, sedangkan alat otomatisasi tes dibatasi untuk eksekusi tes direncanakan saja. Upaya pengujian dapat diklasifikasikan ke dalam tiga bagian, tes kasus generasi, pelaksanaan tes dan evaluasi uji. Makalah ini menyajikan pendekatan baru untuk menghasilkan jalur tes otomatis. Karena keterlambatan dalam pengembangan perangkat lunak, pengujian harus dilakukan dalam waktu singkat waktu. Hal ini menyebabkan otomatisasi pengujian karena yang efisiensi dan juga membutuhkan tenaga kurang. Di dalam diusulkan pendekatan, dengan menggunakan salah satu yang paling standar *Unified Modeling Language* (UML) *Activity Diagram*, membangun tabel Kegiatan Dependency (ADT), maka menghasilkan jalur tes. Kemudian jalur tes diprioritaskan dengan menggunakan algoritma pencarian Tabu. *Path* dapat digunakan dalam pengujian sistem, regresi pengujian dan pengujian integrasi. Kemudian juga membentuk *Cyclomatic diagram* untuk memeriksa efisiensi skenario uji (Shanti,dkk, 2012).

UML diagram aktivitas adalah notasi yang cocok untuk pemodelan sistem bersamaan di mana beberapa objek beorientasi satu sama lain. Makalah ini mengusulkan sebuah metode untuk menghasilkan kasus uji dari UML diagram yang meminimalkan jumlah kasus uji yang dihasilkan saat menurunkan semua kasus. Metode pertama kami membangun sebuah I/O Kegiatan eksplisit diagram dari diagram aktivitas UML biasa dan kemudian mengubahnya ke grafik diarahkan, dari mana tes untuk diagram aktivitas awal berasal. Konversi dilakukan

berdasarkan stimulus prinsip, yang membantu menghindari ledakan negara masalah dalam generasi untuk sistem konkuren (Inyoung Ko, dkk, 2007).

Unified Modeling Language (UML) diagram aktivitas merupakan sistem kunci. UML diagram cocok untuk pengujian tingkat sistem satu ke sistem lainnya. Dalam tulisan ini, pertama grafik aliran aktivitas berasal dari diagram aktivitas. Kemudian, semua informasi yang diperlukan diekstrak dari grafik aliran aktivitas (AFG). Grafik aliran aktivitas (AFG) untuk diagram aktivitas dibuat dengan melintasi diagram aktivitas dari awal sampai akhir, menunjukkan pilihan, kondisi, eksekusi bersamaan, laporan lingkaran. Grafik yang berbeda urutan aliran kontrol yang melintasi AFG. Selanjutnya, algoritma yang diusulkan untuk menghasilkan semua jalur. Akhirnya, uji kasus dihasilkan menggunakan kriteria cakupan aktivitas jalan. Di sini, studi kasus pada *Soft drink Machine Vending* (SVM) memiliki disajikan untuk menggambarkan pendekatannya (Prafulla kumar behera,dkk, 2013).

Pengujian perangkat lunak adalah sangat penting untuk pengembangan perangkat lunak apapun. Pengujian menjamin kualitas perangkat lunak dan pada gilirannya meningkatkannya keandalan dan ketahanan. Fokus utama terletak pada meminimalkan biaya yang dikeluarkan dalam pengujian. Untuk menguji perangkat lunak, masalah utama terletak pada generasi uji kasus. spesifikasi perangkat lunak adalah sumber utama untuk menghasilkan mereka. Spesifikasi perangkat lunak mungkin diagram UML, resmi spesifikasi bahasa, atau deskripsi bahasa alami. Pengujian profesional sekarang telah mengalihkan perhatian mereka ke UML. Spesifikasi perangkat lunak digunakan dalam makalah ini adalah UML *Activity Diagram*. Sementara pengujian perangkat lunak, kita perlu

mengidentifikasi semua jalan untuk memastikan pengujian lengkap. Uji jalur dalam perangkat lunak dapat di identifikasi melalui algoritma yang ada. Tapi masalahnya muncul ketika kita harus mencari tahu jalan yang paling penting di antaramereka karena mereka adalah orang-orang yang paling mungkin untuk dieksekusi. Dalam tulisan ini, telah mempekerjakan heuristik pencarian pendekatan yang disebut tabu Meta untuk mencari tahu jalan dengan prioritas tertinggi sehingga dapat diuji pertama. Mengidentifikasi jalur kenaikan paling penting, efisiensi pengujian (Priya Pati, 2014)

BAB III

METODOLOGI PENELITIAN

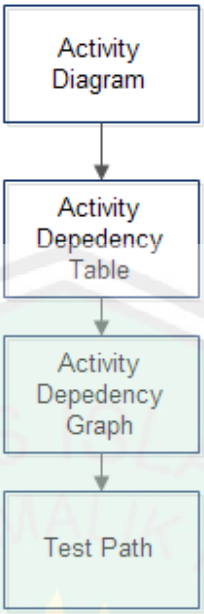
3.1 Analisis Sistem

Pada bagian ini akan menjelaskan langkah-langkah yang akan ditempuh dalam menyelesaikan skripsi. Tahap design dan implementasi aplikasi tidak menjamin suatu aplikasi terbebas dari kesalahan. Ditemukan banyaknya pengembangan yang dilakukan pada suatu aplikasi yang berbasis objek, sehingga aplikasi berbasis objek perlu dilakukan pengujian. Menurut Glen Myers (1979) bahwa *test case* yang baik adalah *test case* yg memiliki probabilitas tinggi untuk menemukan kesalahan yang belum pernah ditemukan sebalumnya.

Pembangkitan *test case* pada aplikasi berbasis objek dapat dilakukan pada saat tahap design dengan memanfaatkan UML Model. Pada penelitian ini dengan menggunakan model *UML Activity Diagram* akan dibangun sebuah aplikasi pembangkit kasus uji (*Test Case*) secara otomatis. Karena pembangkitan kasus uji (*Test Case*) secara otomatis akan lebih menghemat waktu dan pekerjaan jika dibandingkan dengan membangkitkan kasus uji (*Test Case*) secara analisis manual

3.2 Perancangan Sistem

Pada bagian ini menerangkan desain sistem untuk *Activity Diagram*. Ada beberapa tahap yang akan dilakukan untuk pengujian pada Gambar 3.1



Gambar 3.1 Diagram Alur Proses *Test Case*

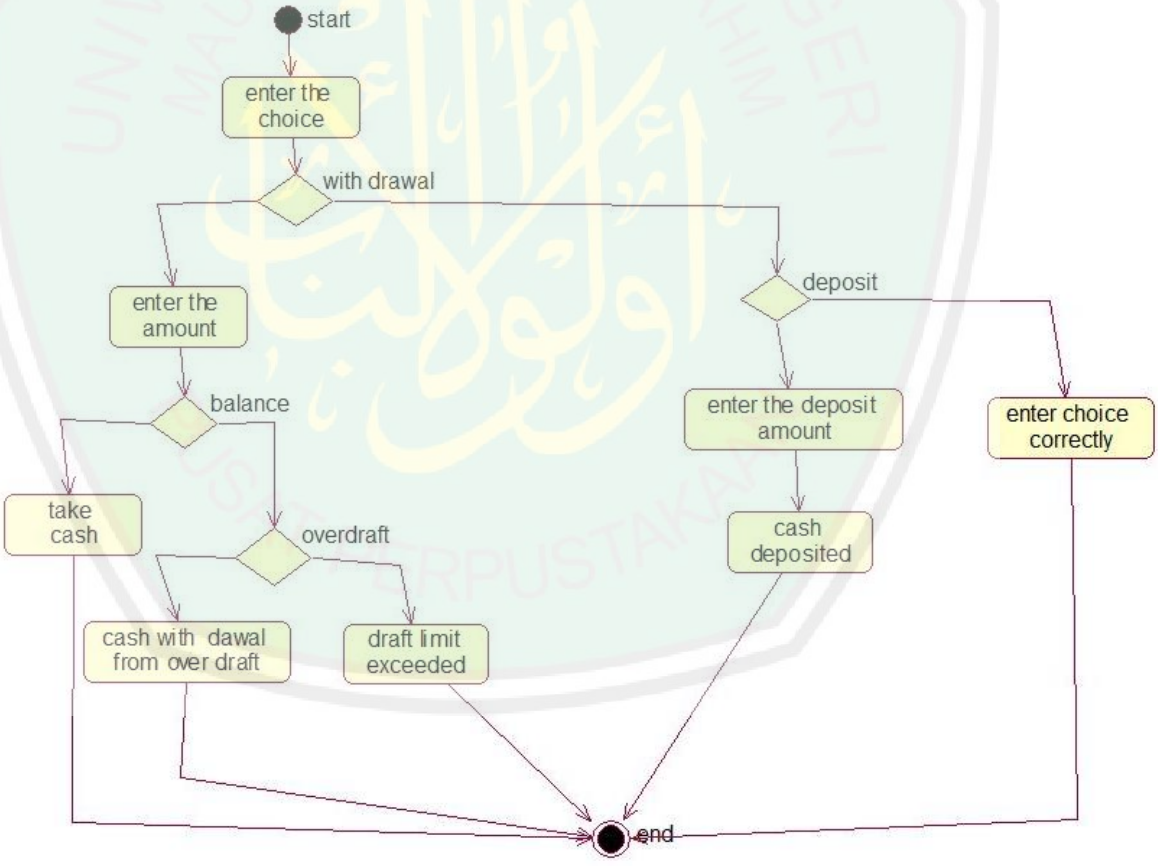
Pada Gambar 3.1 di jelaskan bahwa alur dalam pembuatan *test case Input* adalah *activity diagram* yang dibuat dari jurnal A.V.K. Shanthi dan G. Mohan Kumar (2012), Gargi Bhattacharjee dan Priya Pati (2014). setelah itu di ubah menjadi *activity dependency table* (ADT). Selanjutnya di proses menjadi *activity dependency graph* (AGD) yang berupa *graph* lalu *generate test case* dengan menggunakan algoritma *Depth First Search* (DFS). Ouput yaitu *test path* yaitu jalur *path* yang diperoleh dari graph. perioritas *path* yang dimana *path* ini berfungsi sebagai langkah cepat dalam penyelesaian aplikasi ini.

3.3 Pembuatan *Activity Diagram*

Pada *activity diagram* ini sebagai contoh data uji. *Activity diagram* di ambil dari studi kasus pengambilan uang di ATM dari jurnal A.V.K. Shanthi dan G. Mohan Kumar (2012). penelitian ini menggunakan perangkat lunak *Rational Rose*. Dengan demikian *Activity Diagram* disusun dengan *software* ini dan *output*

dieksport dalam ekstensi .xml. Dalam *Activity Diagram* ini terdapat *decision, fork* dan *join.*

Sebuah kasus *(*A.V.K. Shanthi dan G. Mohan Kumar, 2012) mengenai perbankan, lebih tepatnya pada mesin ATM. Dalam jurnal ini, *Activity Diagram* sebagai desain spesifikasi untuk menerapkan pendekatan uji otomatis dalam perangkat lunak, mengidentifikasi jalur rawan kesalahan pada aplikasi. Dijelaskan pada Gambar 3.4 yaitu tentang pengambilan uang di ATM atau mengecek *deposit.* Terlihat pada Gambar 3.4 berikut ini.

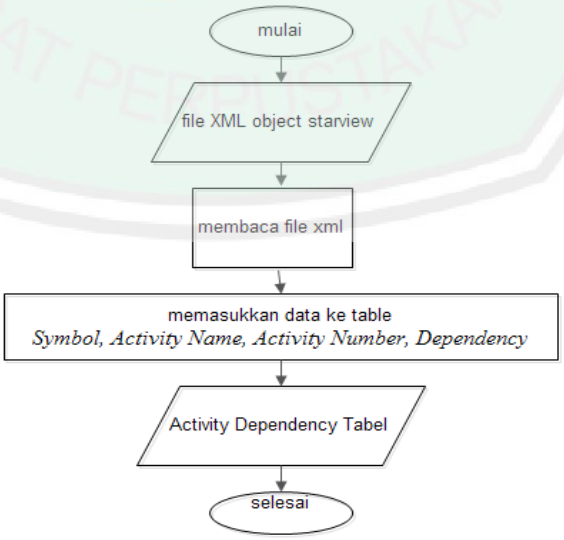


Gambar 3.2 *Activity Diagram* untuk ATM *(*A.V.K. Shanthi dan G. Mohan Kumar, 2012)

Pada Gambar 3.2 dijelaskan bahwa *Activity Diagram* mengenai cara pengambilan uang melalui mesin atm yaitu berupa langsung menulis nominal dan .pengecekan saldo terlebih dahulu. Pertama memasukkan pin lalu memilih pengambilan langsung atau mengecek saldo terlebih dahulu pada tabungan. Pengambilan uang bisa langsung atau dengan memasukkan nominal uang terlebih dahulu. Nominal tertera di layar maka bisa langsung di ambil, jika tidak memasukkan nominal nilai uang yang di ambil. Mengecek saldo terlebih dahulu lalu memasukkan nominal uang, jika saldo di tabungan tidak mencukupi tidak bisa di ambil.

3.4 *Activity Diagram Dependency Tabel* (ADT)

Tahapan pembuatan *Activity Diagram Dependency Table (ADT)* ini, data XML diambil dan diekstrak dengan mengambil beberapa informasi terkait yang ada pada data XML. Informasi terkait terdiri dari *Symbol, Activity Name, Activity Number, Dependency*. Pada *flowchart*\ berikut adalah pembuatan dalam membangunan *Activity Diagram Dependency Table (ADT).*



Gambar 3.3 Flowchart Pembuatan *Activity Diagram Dependency Tabel*

Gambar 3.3 menjelaskan *Flowchart* diatas merupakan proses pengambilan informasi penting dari file *XML* yang kemudian akan dibangunkan sebuah *Activity Diagram Dependency Tabel*. Dari data XML, data yang diambil adalah data yang menyimpan variable *object starview*. Setelah data ditemukan kemudian setiap proses message akan diberikan Symbol berdasarkan urutan pada nilai *activity* yang didapat dari data *XML*. Kemudian *Activity Diagram Dependency Tabel* akan dibentuk dengan format empat kolom (*Symbol, Activity Name, Activity Number, Dependency* Tahapan pembuatan ini *Activity Diagram Dependency Tabel* (ADT), data XML diambil dan diekstrak dengan mengambil beberapa informasi terkait yang ada pada data XML. Informasi terkait terdiri dari *Symbbol, Activity Name, Activity Number, Dependency*.

Tabel 3.1 *Activity Diagram Dependency Tabel (*A.V.K. Shanthi dan G. Mohan Kumar, 2012)

Symbol	*Activity name*	*Dependency*	*Input*	*Output*
A	*Start*			
B	*Enter the Choice*	A	*Choice*	*Type of choice*
C	*Check Withdrawal*	B	*Choice*	*Test score*
D	*Check deposit*	B	*Choice*	*True(deposit) false(enter choice correctly)*
E	*Enter the amount*	C	*Valid amount*	*Check(balance)*
F	*Check balance*	E	*Amount*	*True(take cash) False(check over Draft)*
G	*Check over draft*	F	*Amount*	*True(take cash) False(limit exceeded*

H	*Enter the deposit Amount*	D	*Valid amount*	*Cash deposited*
I	*Cash deposited*	H	*Cash*	*End*
J	*Take cash*	F G	*Cash*	*End*

Tabel 3.1 diperoleh dari *Activity Diagram* Gambar 3.2 menerangkan Symbol Huruf abjad yang diberikan untuk setiap kegiatan yang terlibat.

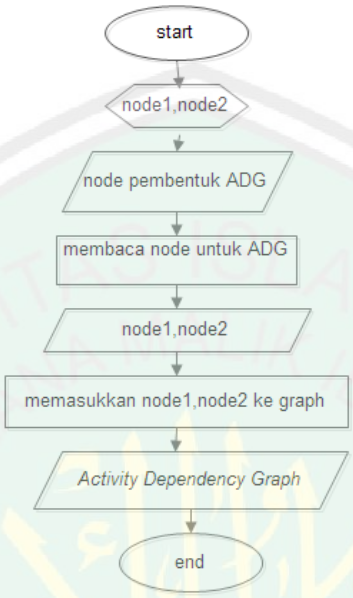
a. *Symbol* : Huruf abjad yang diberikan untuk setiap kegiatan yang terlibat.
b. *Activity Name* : Bagian ini menjelaskan kegiatan yang dilakukan..
c. *Dependency* : Simbol (s) urutan yang menunjukkan bahwa kegiatan saat ini memiliki ketergantungan atau hubungan pada symbol yang lain. Misalnya, kegiatan "C" tergantung pada Activity "A" yang berarti hasil yang dikembalikan tergantung pada nama pengguna dan password yang valid disediakan oleh Pasien.
d. *Input* : Input memberikan prasyarat yang harus terjadi.

Output : memberikan output yang diharapkan dari peristiwa saat ini yang terjadi.

3.5 Membangun *Activity Dependency Graph* (AGD)

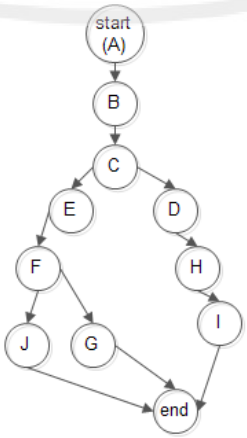
Tahapan dalam membangun *Activity Dependency Graph* (AGD) ini, proses dilakukan dengan mengambil kolom *Dependency* dan kolom *Symbol* pada hasil *Activity Dependency Table* yang kemudian dijadikan sebagai *node* dalam *graph* (*Activity Dependency Graph*). Dimana kolom *Dependency* sebagai (node awal) sedangkan kolom *Symbol* sebagai *node-node (node* tujuan). Berikut

flowchart proses pembuatan *Activity Dependency Graph* (ADG). Berikut ini adalah gambar *flowchart* dalam pembuatan *Activity Dependency Graph* (ADG).



Gambar 3.4 Flowchart pembuatan *Activity Dependency Graph* (ADG)

Pada Gambar 3.4 dijelaskan bahwa *node-node* pembentuk *Activity Dependency Graph* (ADG) dibuat berdasarkan mengambil kolom *Dependency* dan kolom *Symbol* pada hasil *Activity Dependency Table* lalu memasukkan *node* ke graph. *Node1* ini adalah *node* awal, sedangkan *node-node* sebagai *node* tujuan. Berikut ini adalah hasil *Graph* yang telah dihasilkan. Tedapat pada Gambar 3.8



Gambar 3.5 *Activity Dependency Graph* (ADG)

3.6 Hasil *Test Path* dan Penerapan Metode

Metode *Depth First Search* (DFS) diterapkan dalam proses pencarian node-node graph yang saling terhubung. Metode ini untuk mencari jalur *path* dari *start* sampai *end* melalui *object startview* yang ada di format.xml tersebut. Untuk membentuk beberapa kemungkinan jalur yang akan dilalui dalam kasus pengujian (*Test Path*). Dengan menerapkan metode *Depth First Search* (DFS). Sehingga dari *Activity Dependency Graph* akan menghasilkan *test path* seperti berikut :

Start → enter the choice → withdrawal → enter the amount → balance → take cash → end
Start → enter the choice → withdrawal → enter the amount → balance → over draft → cash withdrawal from over draft → end
Start → enter the choice → withdrawal → enter the amount → balance →over draft → draft limit exceed → end
Start → enter the choice → withdrawal → deposit → enter the amount for deposit → cash deposited → end
Start → enter the choice → withdrawal → deposit →enter the choice correctly → end

Gambar 3.6 hasil *path* jurnal A.V.K. Shanthi dan G. Mohan Kumar (2012)

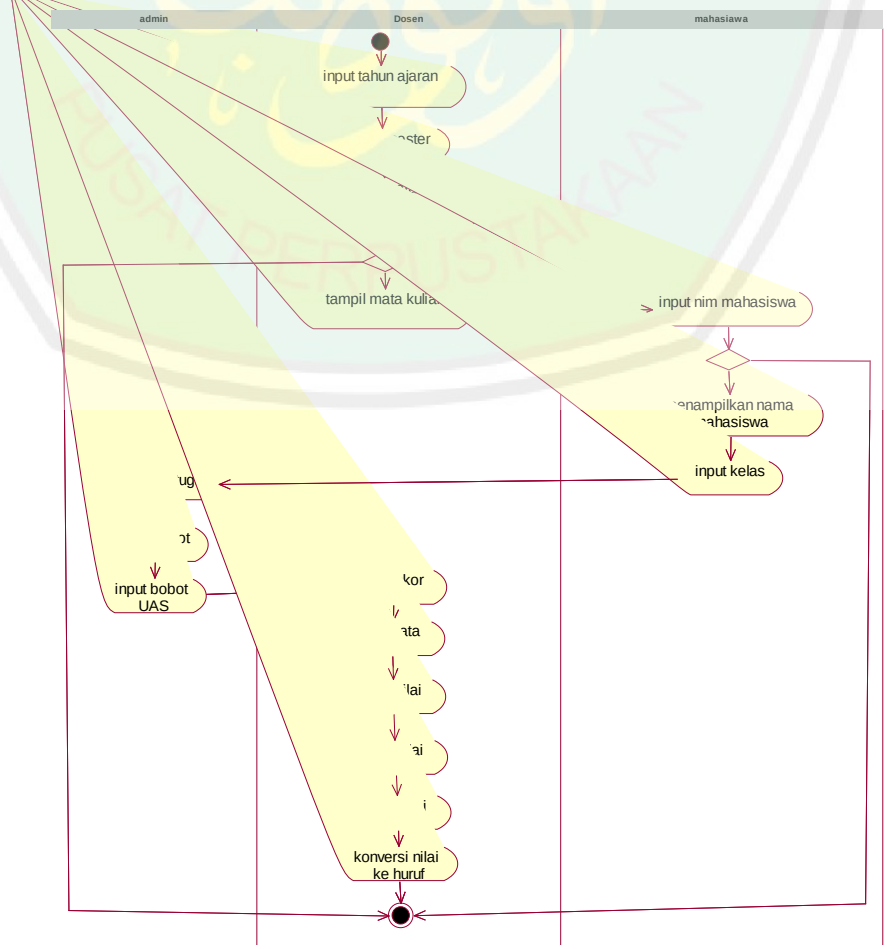
Gambar 3.6 adalah hasil path dari jurnal A.V.K. Shanthi dan G. Mohan Kumar (2012) sebagai beriku:

1. *Start- enter the choice- withdrawal-enter the amount-balance-take cash-end*
2. *Start- enter the choice- withdrawal-enter the amount-balance-over draft-cash withdrawal from over draft-end*
3. *Start- enter the choice- withdrawal-enter the amount-balance-over draft-draft limit exceed-end*
4. *Start- enter the choice- withdrawal- deposit- enter the amount for deposit-cash deposited- end*
5. *Start- enter the choice- withdrawal- deposit- enter the choice correctly – end*

3.7 Uji Kasus Sistem Penilaian Pembelajaran

3.7.1 *Activity Diagram*

Uji kasus selanjutnya yaitu Sistem Penilaian Pembelajaran sebagai bahan uji *activity diagram*. Pada uji kasus terdapat tiga aktor yaitu admin, dosen dan mahasiswa. Admin bertugas menginputkan bobot tugas, bobot UTS dan bobot UAS. Dosen menginputkan tahun ajaran, semester kode mata kuliah, jika kode yang di masukkan ada maka akan tampil kode kuliah, jika tidak maka *end*. Selain itu, dosen juga menginputkan skor tugas, rata-rata tugas, nilai UTS dan UAS, nilai akhir dan mengkonversi nilai ke huruf. Mahasiswa menginputkan nim mahasiswa, jika nim ada maka akan tampil nama mahasiwa, jika tidak maka *end*, setelah itu menginputkan kelas. *Activity diagram* ini diterapkan di Sistem Penilaian Pembelajaran. Pada Gambar 3.7 menunjukkan *activity diagram* Sistem Penilaian pembelajaran.



Gambar 3.7 *Activity Diagram* data uji

Pada Gambar 3.7 adalah *Activity Diagram* yang akan diterapkan pada Sistem Penilaian pembelajaran

3.7.2 *Activity Diagram Dependency Tabel* (ADT)

Pembuatan *Activity Diagram Dependency Tabel* (ADT) dari *Activity Diagram* Sistem Penilaian Pembelajaran akan di masukkan pada tabel *Symbol, Activity Name, Activity Number, Dependency*.

Tabel 3.2 *Activity Diagram Dependency Tabel*

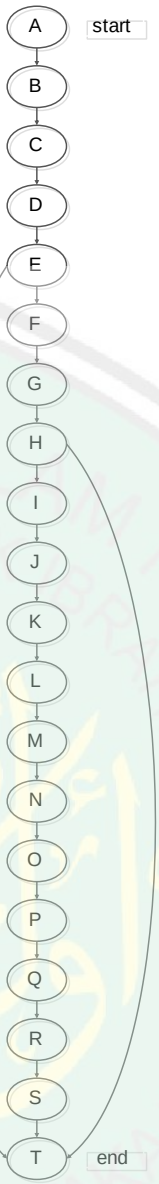
Symbol	*Activity name*	*Dependency*	*Input*	*Output*
A	*Start*			
B	Input tahun ajaran	C	*Choice*	Tahun ajaran
C	Input semester	D	*Choice*	Semester
D	Input kode mata kuliah	E	*Choice*	Kode mata kuliah
E	Cek kode mata kuliah	F	*Chloice*	Tampil mata kuliah
	Cek kode mata kuliah	T	*Valid*	*End*
F	Tampil mata kuliah	G	*Choice*	Mata kuliah
G	*Input* nim mahasiswa	H	*Choice*	Nim mahasiswa
H	Cek nim mahasiswa	I	*Choice*	Nim mahasiswa
	Cek nim mahasiswa	T	*Valid*	*End*
I	Menampilkan nama mahasiswa	J	*Choice*	Nama mahasiswa
J	*Input* kelas	K	*Choice*	Nama kelas
K	*Input* bobot tugas	L	*Choice*	Bobot tugas
L	*Input* bobot UTS	M	*Choice*	Bobot UTS
M	*Input* bobot	N	*Choice*	Bobot UAS

	UAS			
N	*Input* skor tugas	O	*Choice*	Skor tugas
O	*Input* rata rata tugas	P	*Choice*	Rata rata tugas
P	*Input* nilai UTS	Q	*Choice*	Nilai UTS
Q	*Input* nilai UAS	R	*Choice*	Nilai UAS
R	*Input* nilai akhir	S	*Choice*	Nilai akhir
S	*Konversi* nilai ke huruf	T	*Valid*	*End*

Pada Tabel 3.2 *Activity Diagram Dependency Tabel* Sistem Penilaian Pembelajaran terdapat *node-node* yang untuk mendapatkan jalur *path* yang dibutuhkan dalam pembetukan *path.*

3.7.3 Membangun *Activity Dependency Graph* (AGD)

Pembentukan *Activity Dependency Graph* (AGD) diperoleh dari Tabel 3.2 lalu disusun kolom *Dependency* sebagai (node awal) sedangkan kolom *Symbol* sebagai *node-node (node* tujuan) ditunjukkan pada Gambar 3.7



Gambar 3.8 *Activity Dependency Graph* (AGD)

3.7.4 Hasil *Test Path*

Test path mengggunakan Metode *Depth First Search* (DFS). Metode ini untuk mencari jalur *path* dari *start* sampai *end* melalui *object startview* yang ada di format.xml tersebut. Hasil *pathnya* adalah sebagai berikut :

1. A- B- C- D- E- T
2. A- B- C- D- E- F- G-H- T
3. A- B- C- D- E-F-G-H-I-J-K-L-M-N-O-P-Q-R-S-T

BAB IV

UJI COBA DAN PEMBAHASAN

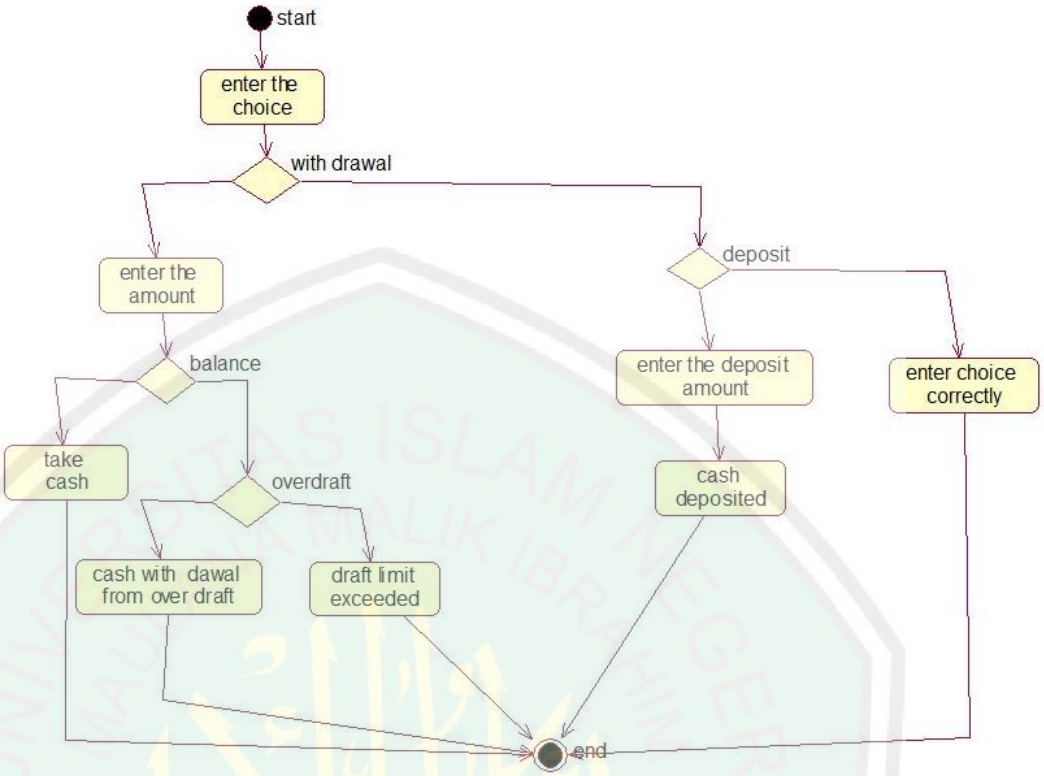
Dalam bab ini akan membahas tentang hasil uji dan pembahasan bagaimana proses pembuatan aplikasi, uji coba dilakukan untuk mengetahui apakah aplikasi dapat berjalan sesuai dengan yang diharapkan.

4.1 Hasil

Pengujian merupakan bagian yang terpenting dalam proses pembuatan perangkat lunak. Pengujian ini dilakukan untuk menjamin kulitas dari perangkat lunak yang dibangun dan mengetahui kelemahan dari perangkat lunak yang dibangun. Kasus uji yang baik adalah yang memiliki tingkat kemungkinan tinggi untuk menemukan kerusakan yang belum ditemukan..

4.1.1 *Activity Diagram* untuk ATM

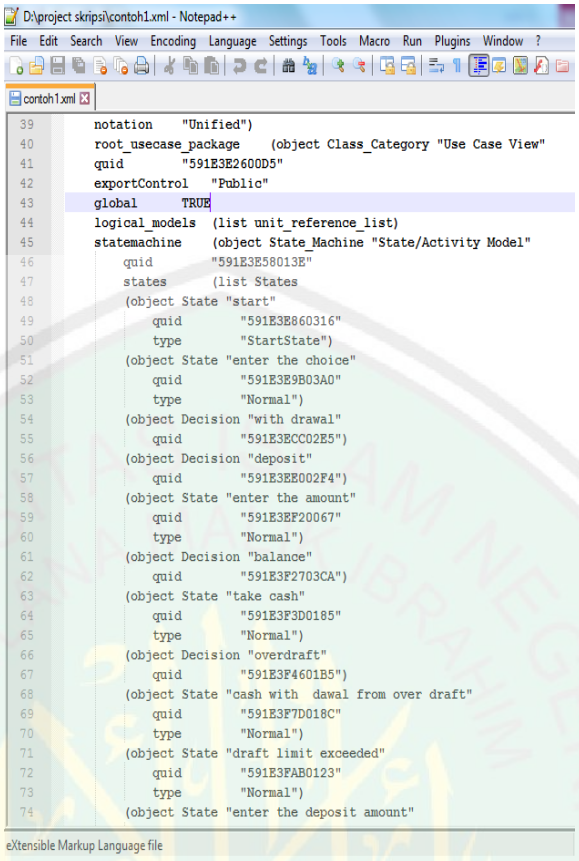
Data yang pertama yaitu mengenai pengambilan uang pada mesin ATM. Langkah pertama mengenai cara pengambilan uang melalui mesin atm yaitu berupa langsung menulis nominal dan .pengecekan saldo terlebih dahulu. Pertama memasukkan pin lalu memilih pengambilan langsung atau mengecek saldo dahulu pada tabungan. Pada pengambilan uang bisa langsung atau dengan memasukkan nominal uang terlebih dahulu, jika nominal keluar tertera di layar maka bisa langsung di ambil, jika tidak memasukkan nominal nilai uang yang di ambil. Langkah kedua yaitu mengecek saldo terlebih dahulu lalu memasukkan nominal uang, jika saldo di tabungan tidak mencukupi maka uang tidak bisa di ambil. Beriku adalah gambar *activity diagram* pada Gambar 4.1



Gambar 4.1 *Activity Diagram* untuk ATM *(*A.V.K. Shanthi dan G. Mohan Kumar.2012)

4.1.2 Konversi ke *XML* pada persoalan ATM

Pada tahap adalah konversi *file* ke *XML* di ambil dari data pertama yaitu *activity diagram* ATM. Pertama *save as* di tempat *file* yang dinginkan dengan format .*XML*, seperti Gambar 4.2 berikut :



Gambar 4.2 Data Uji Model *UML Activity diagram* ke *File* XML sebelum Konversi *File*

4.1.3 Pembuatan *Activity Depedency Table* (ADT)

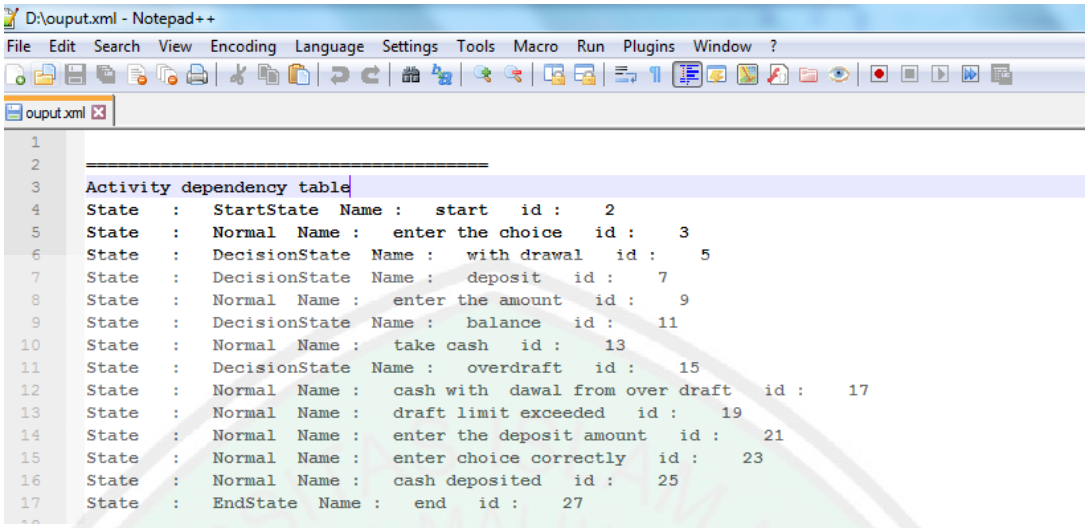
Tahap selanjutnya yaitu mengambil data data yang diperlukan. *Source code* yang terapkan menggukan *arraylist* yaitu dengan mencari data dari s*tart* sampai *end.*

Source Code Konversi XML ke *Sequence Activity Dependency Table* (ADT) (i)

```
While (line != null) {
if (line.contains("client         @")) {
String y[] = line.split("@");
client.add(y[y.length - 1]);
} else if (line.contains("supplier       @")) {
String y[] = line.split("@");
supplier.add(y[y.length - 1]);
}
sb.append(line).append("\n");
if (line.contains("(object StateView") || line.contains("(object
```

```
DecisionView") || line.contains("(object ActivityStateView") ||
line.contains("Parent_View       ") || line.contains("label
       (object ItemLabel")) {
sb2.append(line).append("\n");
if (line.contains("(object StateView") || line.contains("(object
DecisionView") || line.contains("(object ActivityState")) {
String y[] = line.split("\"");
if (line.contains("(object ActivityStateView") || line.contains("(object
DecisionView")) {
String r;
if (line.contains("State")) {
r = "AcivityState";
} else {
r = "DecisionState";
}
sb5.append("\nState   :   " + r + "  Name :   " + y[y.length - 2] + "
id :    " + y[y.length - 1].substring(2));
System.out.println("State   :   " + r + "  Name :   " + y[y.length - 2] +
"   id :    " + y[y.length - 1].substring(2));

id.add(y[y.length - 1].substring(2));
name.add(y[y.length - 2]);
state.add(r);
} else {

sb5.append("\nState   :   " + y[y.length - 4] + "  Name :   " +
y[y.length - 2] + "   id :    " + y[y.length - 1].substring(2));
System.out.println("State   :   " + y[y.length - 4] + "  Name :   " +
y[y.length - 2] + "   id :    " + y[y.length - 1].substring(2));

id.add(y[y.length - 1].substring(2));
name.add(y[y.length - 2]);
state.add(y[y.length - 4]);
if (y[y.length - 4].contains("EndState")) {
end.add(y[y.length - 1].substring(2));
} else if (y[y.length - 4].contains("StartState")) {
start.add(y[y.length - 1].substring(2));
}
}
sb4.append("\n\n\t\tpublic void " + y[y.length - 2] + "(){\n\n\n\t\t}");

}
}
```

Berikut adalah hasil pengambilan data yang penting sehingga menjadi *Activity Depedency Table* (ADT) ditunjukan pada Gambar 4.3.



Gambar 4.3 *Activity Depedency Table*

Gambar 4.3 adalah data yang sudah di ambil dari beberapa sekian data dalam satu *XML*. Data yang diambil yaitu melalui potongan *Source Code* berikut, yaitu data yang diambil object *StateView, object DecisionView, object Activity StateView, Parent_View, label (object ItemLabel). Id* yang dihasilkan adalah otomatis tertera pada masing-masing data di atas.

4.1.4 Pembuatan *Activity Depedency Graph* (ADG)

Tahap pembentukan *node graph* yaitu dengan menggabungkan pada *id* sebelumnya dan *id* sesudahnya. *Source code* terlihat dibawah ini :

Source Code Activity Depedency Table (ADT) ke *Activity Dependency Graph* (ADG) (ii)

```
sb3.append("\n\n\t\tpublic " + jTextField2.getText() + "(){\n");
        for (int i = 0; i < name.size(); i++) {
            sb3.append("\t\t\t" + name.get(i) + "();" + "\n");
        }
        sb3.append("\t\t}");
        sb3.append(sb4.toString());
        sb3.append("\n\n}");
       ArrayList a2 = new ArrayList();
        ArrayList a3 = new ArrayList();
        int check = 0;
```

```
        int y = 0;
        AllPathsFromASource z = new AllPathsFromASource();
        System.out.println("");
        System.out.println("Node Pembentuk Graph");
        sb5.append("\n=====================================");
        sb5.append("\nNode Pembentuk Graph");
        System.out.println("");
        for (int i = 0; i < client.size(); i++) {
            sb5.append("\n" + client.get(i) + "->" + supplier.get(i));
            System.out.println(client.get(i) + "->" + supplier.get(i));
            z.addEdge(Integer.parseInt(client.get(i).toString()),
Integer.parseInt(supplier.get(i).toString()));
        }
```

Hasil dari *source code* di atas mendapatkan *node graph* berdasarkan *ID* yang tertera pada data dari *Activity Depedency Table*.

```
=====================================
Node Pembentuk Graph
2->3
3->5
5->7
5->9
9->11
11->13
11->15
15->17
15->19
7->21
7->23
21->25
13->27
23->27
17->27
19->27
25->27
-------------------------------------
```

Gambar 4.4 *node graph* dari *Activity Depedency Table*

Pada Gambar 4.4 adalah hasil *node graph* dari *ID* yang sudah ada. *ID* di gabungkan dari *ID* sebelumnya ke *ID* sesudahnya.

4.1.5 Menerapkan Metode *Depth First Search (DFS)*

Dalam mendapatkan hasil *path* dari *activity diagram* data satu, mulai dari *start* sampai *end* menerapkan metode *Depth First Search* (DFS). Pertama di inisialisasi dari *line-line* pada *.XML* yang sudah menjadi *Activity Depedency*

Table. Lalu mencari *object value start* dan *end*, maksudnya adalah mencari *path* mana saja yang terdapat dalam *start* sampai *end*, di masukkan ke dalam *array*. *Array* di *looping*, di masukkan ke dalam *class AllPaths*. Kemudian metode ini bekerja sebagai pencari jalur pada *graph* sehingga menghasilkan jalur pengujian (*Test path*) berikut adalah potongan source codenya.

Source Code penerapan metode DFS pada aplikasi (iii)

```
public class AllPathsFromASource {
		int V=100000;
		Map<Integer, List<Integer>> adj; // Adjacency
		AllPathsFromASource(){
			adj = new HashMap<Integer, List<Integer>>();
		}
		void addEdge(int u, int v){
			if(!adj.containsKey(u)){
				adj.put(u, new ArrayList<Integer>());
			}
			adj.get(u).add(v);
		}

		List<List<Integer>> getAllPaths(int u, int v){
			List<List<Integer>>	result	=	new
ArrayList<List<Integer>>();
if(u==v){
				List<Integer>	temp	=	new
ArrayList<Integer>();
				temp.add(u);
				result.add(temp);
				return result;
			}
			boolean[] visited = new boolean[V];
			Deque<Integer> path = new ArrayDeque<Integer>();
			getAllPathsDFS(u, v, visited, path, result);
			return result;
		}

		void	getAllPathsDFS(int	u,	int	v,	boolean[]	visited,
Deque<Integer> path, List<List<Integer>> result){
			visited[u] = true; // Mark visited
			path.add(u); // Add to the end
			if(u==v){
				result.add(new ArrayList<Integer>(path));
			}
			else{
				if(adj.containsKey(u)){
					for(Integer i : adj.get(u)){
						if(!visited[i]){
							getAllPathsDFS(i,	v,
visited, path, result);
						}}}}
			path.removeLast();
			visited[u] = false;
		}
	public static void main(String[] args){
```

```
AllPathsFromASource g = new AllPathsFromASource();
   g.addEdge(2,100);
   g.addEdge(100,3);
   g.addEdge(100, 1);
g.addEdge(0,1);
g.addEdge(0,2);
g.addEdge(0,3);
g.addEdge(2,0);
g.addEdge(2,1);
g.addEdge(1,3);
List<List<Integer>> results = g.getAllPaths(0,3);
for(List<Integer> l : results){
      System.out.println(l);
```

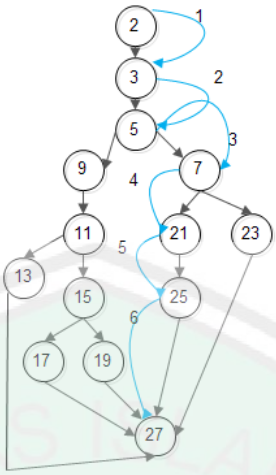
Selanjutnya, mengeluarkan semua *path* dengan otomatis setelah *path* di *looping* dan berikut adalah potongan source codenya.

Source Code menampilkan seluruh *path* pada aplikasi (iv)

```
  sb5.append("\n=====================================");
        sb5.append("\nAll path from StartState to EndState");
        System.out.println("All path from StartState to EndState");
        System.out.println("");
        for (int i = 0; i < start.size(); i++) {
            for (int j = 0; j < end.size(); j++) {
                List<List<Integer>>                results                =
z.getAllPaths(Integer.parseInt(start.get(i).toString()),
Integer.parseInt(end.get(j).toString()));
                sb5.append("\n    Path    from    id                "    +
start.get(i).toString() + " to id " + end.get(j).toString());
                System.out.println("Path    from    id                "    +
start.get(i).toString() + " to id " + end.get(j).toString());
                for (List<Integer> l : results) {
                    sb5.append("\n"+l);
                    System.out.println(l);
                }
            }
        }

        for (int i = 0; i < client.size(); i++) {
            if (allClient.contains(client.get(i)) == false) {
                allClient.add(client.get(i));
            }
            if (allSupplier.contains(supplier.get(i)) == false) {
                allSupplier.add(supplier.get(i));
            }
        }
  String g= sb5.toString();
        jTextArea1.setText(g);
    }

    private  void  jButton3ActionPerformed(java.awt.event.ActionEvent  evt)
{
        // TODO add your handling code here:
        Writer writer = null;
        Writer writer1 = null;
        try {
```

```
            writer = new BufferedWriter(new OutputStreamWriter(
                    new FileOutputStream("D:\\ouput.xml"), "utf-8"));
            writer.write(jTextArea1.getText());
        } catch (IOException ex) {
            // report
        } finally {
            try {
                writer.close();
            } catch (Exception ex) {/*ignore*/
            }
        }

        try {
            writer1 = new BufferedWriter(new OutputStreamWriter(
                    new FileOutputStream("D:\\" + jTextField2.getText() +
".java"), "utf-8"));
            writer1.write(sb3.toString());
        } catch (IOException ex) {
            // report
        } finally {
            try {
                writer1.close();
            } catch (Exception ex) {/*ignore*/
            }
        }
    }
```

Source code di atas menyimpan semua *node* yang sudah saling terhubung. 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 4.5 di bawah ini.



Gambar 4.5 Representasi DFS pada *Graph*

Representasi DFS pada *graph* di atas bekerja sesuai dengan alur penomoran hingga menuju alur ke node terakhir "27". Pada *path* pertama :

a. Alur nomor 1 yang mengarahkan dari *node* 2 ke *node* 3 sehingga mencetak *path* 1 2

b. Dilanjutkan alur nomor 2 mengarahkan dari node 3 ke node 5 sehingga *path* 1 2 3

Path yang dihasilkan bisa dilihat di Gambar 4.6 yaitu ada lima *path* yang dikeluarkan dari jalur *path* mulai *start* sampai *end*.

```
37  ========================================
38  All path from StartState to EndState
39   Path from id    2 to id 27
40  [2, 3, 5, 7, 21, 25, 27]
41  [2, 3, 5, 7, 23, 27]
42  [2, 3, 5, 9, 11, 13, 27]
43  [2, 3, 5, 9, 11, 15, 17, 27]
44  [2, 3, 5, 9, 11, 15, 19, 27]
```

Gambar 4.6 Jalur hasil *path* dari data uji satu

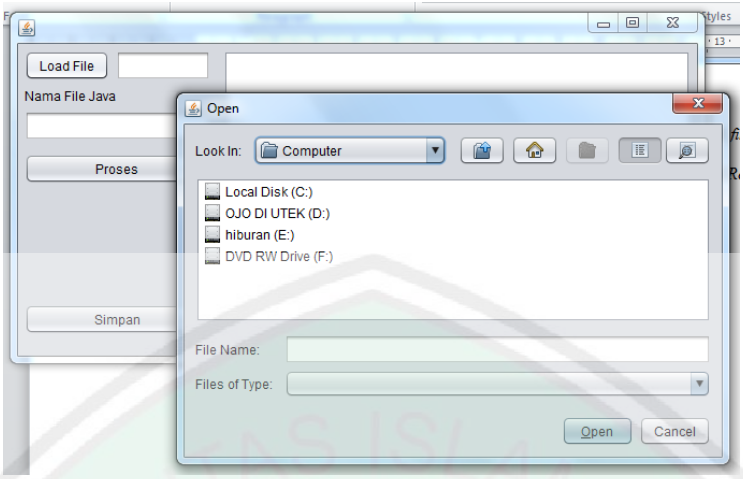
Pada Gambar 4.6 adalah jalur hasil uji *path* yang di hasilkan pada data uji satu. *Path* di hasilkan melalui *looping* pada jalur *path* mulai *start* sampai *end.*

4.1.6 Hasil

Pada pengujian ini menggunakan Netbeans IDE 6.7.1 sebagai editor untuk penulisan kode program dan *Rational Rose Enterprise Edition* 2002 untuk mendesain data awal atau *activity diagram* dari suatu sistem yang akan diuji Aplikasi pembangkit *test case* otomatis ini dibangun dengan menggunakan bahasa pemrograman Java. 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.

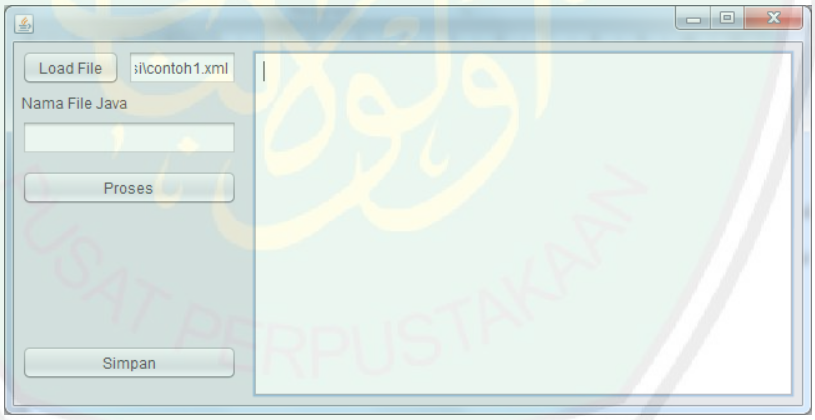
Gambar 4.7 *Interface* Program

Pada gambar 4.7 adalah tampilan awal program. Lalu *clik "load file"* untuk mengambil *file* data uji yang sudah di gambar pada aplikasi *Rasional Rose* dengan format *.XML*. Terlihat pada gambar berikut ini



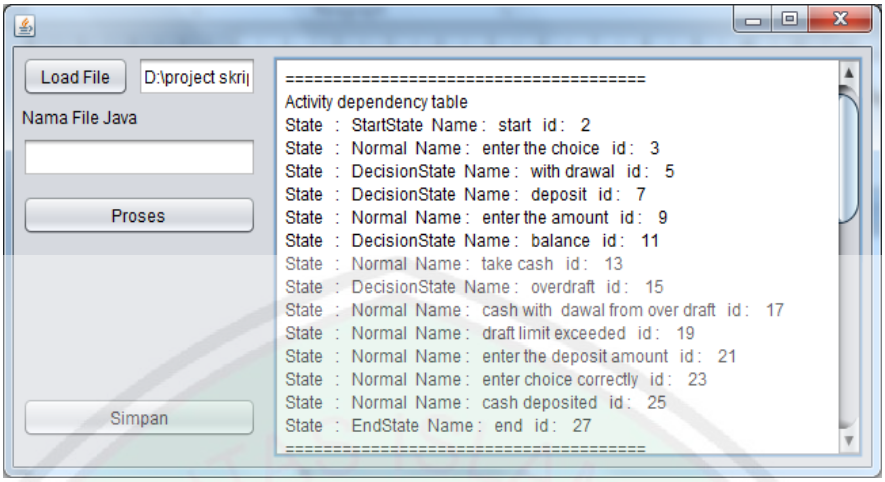
Gambar 4.8 pengambilan data uji

Pada gambar 4.8 adalah pengambilan data uji yang akan di olah pada aplikasi di atas. Setelah itu pilih *file*nya lalu tekan tombol *"load file"*, maka langsung otomatis akan muncul semua hasilnya mulai dari *Activity Depedency Table* (ADT), *node* pembentuk *graph* dan hasil keseluruhan *path* nya.

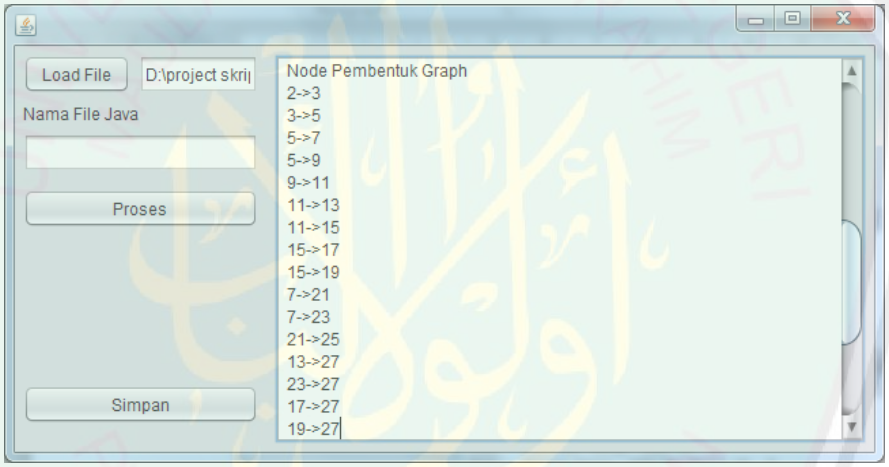


Gambar 4.9 pengambilan file

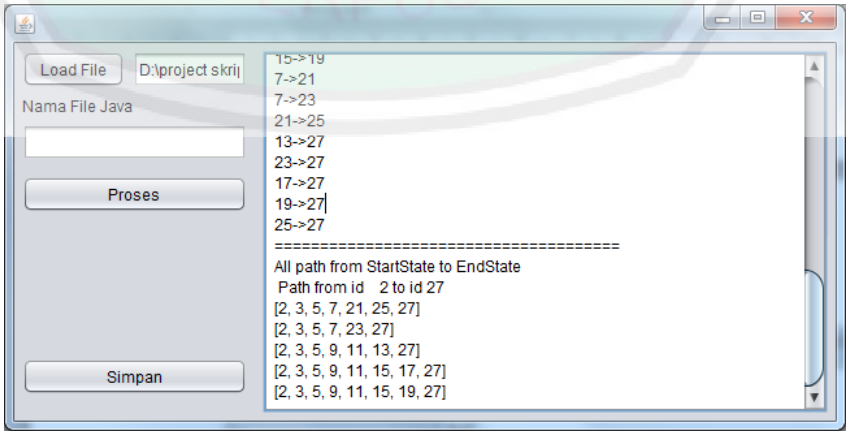
Gambar 4.9 adalah pengambilan *file* yang akan di olah, keluar otomatis semua hasilnya. Sesuai gmbar dibawah ini



Gambar 4.10 hasil *Activity Depedency Table* otomatis

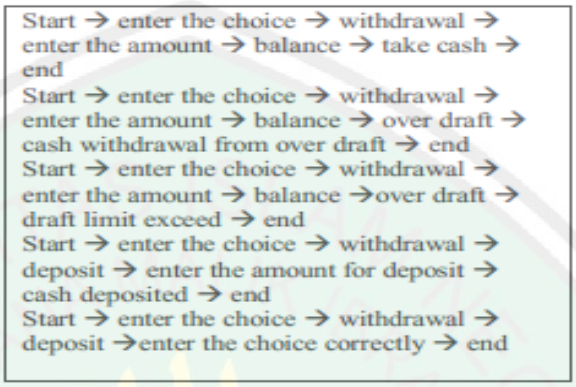


Gambar 4.11 hasil *node* pembentuk *graph*



Gambar 4.12 hasil *Path* otomatis

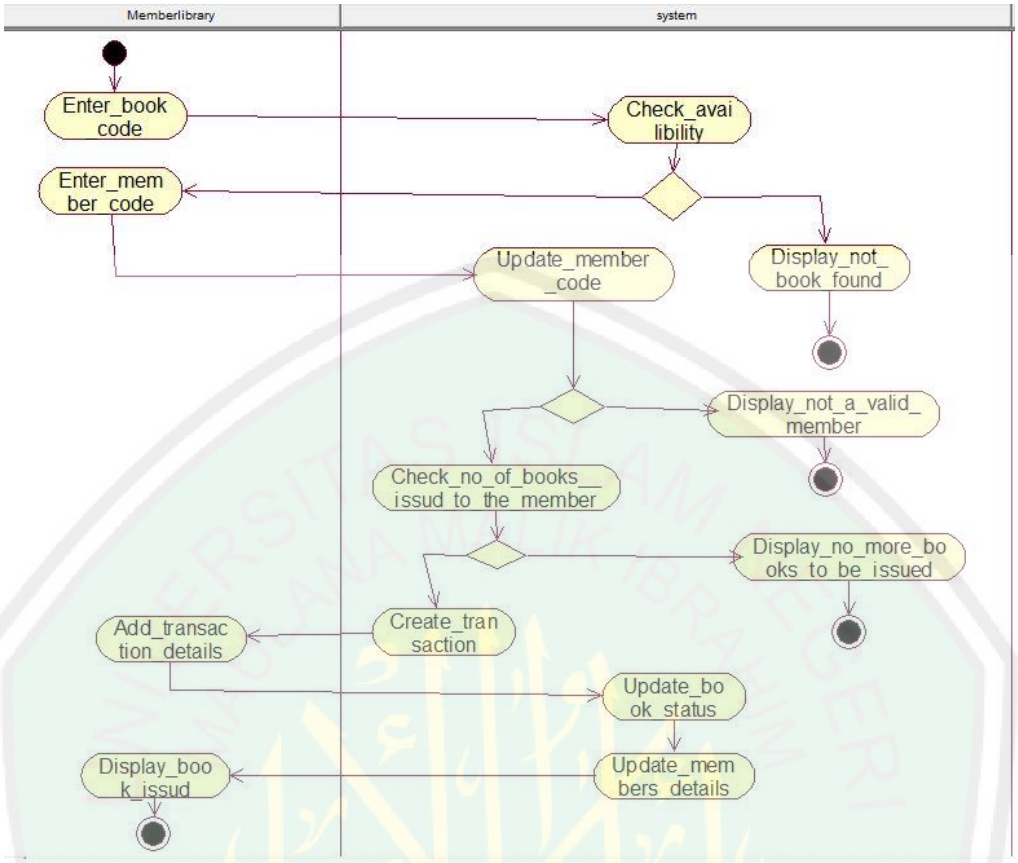
Hasil *path* pada program sama dengan hasil *path* pada jurnal. Pada jurnal jurnal A.V.K. Shanthi dan G. Mohan Kumar (2012), hasil pathnya yaitu seperti Gambar 4.13



Gambar 4.13 hasil jurnal A.V.K. Shanthi dan G. Mohan Kumar(2012)

4.1.7 *Library Management*

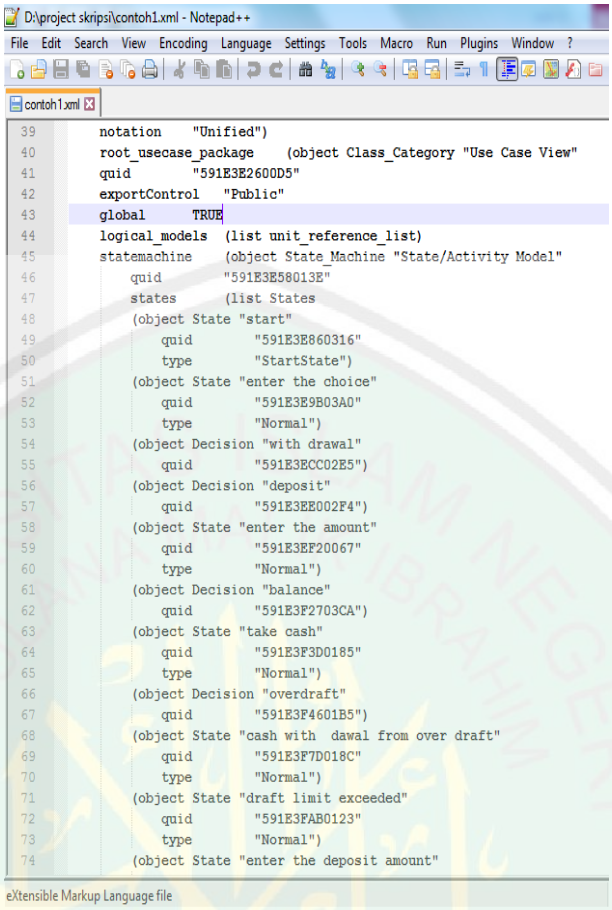
Data yang kedua yaitu tentang *Library Management* atau Manajemen Perpustakaan, *activity diagram* ini menggunkan *swimline*. Pertama memasukkan *code* buku yg di cari terlebih dahulu, lalu *system* melakukan pengecakan,jika ketemu bukunya maka *member* memasukkan *code member* dan jika tidak maka buku tidak ditemukan. Setelah memasukkan *code member*, *system* mengupdate apakah benar *code member*, jika tidak ada maka *member* tidak *valid*. *System* mengecek buku yang telah dipinjam oleh *member* jika tidak ada buku yang dipinjam lalu membuat membuat transaksi baru. *Member* lalu melihat detail transaksi, lalu *system mengupdate* status buku dan *mengupdate member*. Buku bisa di tampilkan terlihat pada Gambar 4.14.



Gambar 4.14 *Activity diagram Library Management*

4.1.8 Konversi ke *XML* pada persoalan

Pada tahap adalah konversi *file* ke *XML* di ambil dari data pertama yaitu *activity diagram* ATM. Pertama *save as* di tempat *file* yang dinginkan dengan format .*XML*, seperti Gambar 4.15 berikut :



Gambar 4.15 Data Uji Model *UML Activity diagram* ke *File* XML sebelum Konversi *File*

4.1.9 Proses Pembuatan *Activity Depedency Table* (ADT)

Tahap selanjutnya yaitu mengambil data data yang diperlukan. *Source code* yang terapkan menggukan *arraylist* yaitu dengan mencari data dari *start* sampai *end*. Berikut adalah hasil pengambilan data yang penting sehingga menjadi *Activity Depedency Table* (ADT) ditunjukan pada Gambar 4.16.

```
==========================================
Activity dependency table
State   :   StartState  Name :   $UNNAMED$0   id :    2
State   :   AcivityState  Name :   Enter_book_code   id :    3
State   :   AcivityState  Name :   Enter_member_code   id :    5
State   :   AcivityState  Name :   Add_transaction_details   id :    6
State   :   AcivityState  Name :   Display_book_issud   id :    7
State   :   EndState  Name :   $UNNAMED$7   id :    8
State   :   AcivityState  Name :   Check_availibility   id :    10
State   :   DecisionState  Name :   $UNNAMED$1   id :    12
State   :   AcivityState  Name :   Update_member_code   id :    15
State   :   AcivityState  Name :   Display_not_book_found   id :    17
State   :   EndState  Name :   $UNNAMED$2   id :    19
State   :   DecisionState  Name :   $UNNAMED$3   id :    21
State   :   AcivityState  Name :   Check_no_of_books__issud_to_the_member   id :    23
State   :   AcivityState  Name :   Display_not_a_valid_member   id :    25
State   :   EndState  Name :   $UNNAMED$4   id :    27
State   :   DecisionState  Name :   $UNNAMED$5   id :    29
State   :   AcivityState  Name :   Display_no_more_books_to_be_issued   id :    30
State   :   AcivityState  Name :   Create_transaction   id :    31
State   :   EndState  Name :   $UNNAMED$6   id :    32
State   :   AcivityState  Name :   Update_book_status   id :    33
State   :   AcivityState  Name :   Update_members_details   id :    34
==========================================
```

Gambar 4.16 *Activity Depedency Table*

Gambar 4.16 adalah data yang sudah di ambil dari beberapa sekian data dalam satu *XML*. Data yang diambil yaitu melalui potongan *Source Code* berikut, yaitu data yang diambil object *StateView, object DecisionView, object Activity StateView, Parent_View, label (object ItemLabel). Id* yang dihasilkan adalah otomatis tertera pada masing-masing data di atas.

4.1.10 Pembuatan *Activity Depedency Graph* (ADG)

Tahap pembentukan *node graph* yaitu dengan menggabungkan pada *id* sebelumnya dan *id* sesudahnya. Hasil dari *source code* di atas mendapatkan *node graph* berdasarkan *ID* yang tertera pada data dari *Activity Depedency Table*.

```
25  ====================================
26  Node Pembentuk Graph
27  2->3
28  3->10
29  10->12
30  12->5
31  5->15
32  12->17
33  17->19
34  15->21
35  21->23
36  21->25
37  25->27
38  23->29
39  29->30
40  29->31
41  30->32
42  31->6
43  33->34
44  34->7
45  7->8
46  6->33
47  ====================================
```

Gambar 4.17 *node graph* dari *Activity Depedency Table*

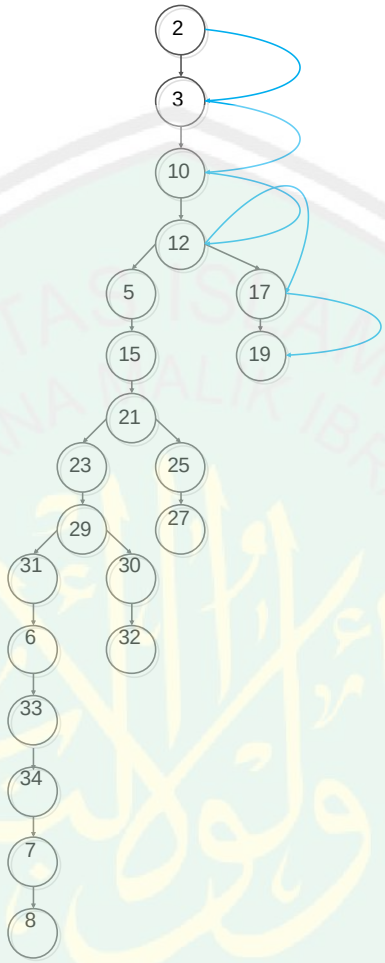
Pada Gambar 4.17 adalah hasil *node graph* dari *ID* yang sudah ada. *ID* di gabungkan dari *ID* sebelumnya ke *ID* sesudahnya

4.1.11 Menerapkan Metode *Depth First Search* (DFS)

Dalam mendapatkan hasil *path* dari *activity diagram* data satu, mulai dari *start* sampai *end* menerapkan metode *Depth First Search* (DFS). Pertama di inisialisasi dari *line-line* pada *.XML* yang sudah menjadi *Activity Depedency Table*. Lalu mencari *object value start* dan *end*, maksudnya adalah mencari *path* mana saja yang terdapat dalam *start* sampai *end* lalu di masukkan ke dalam *array*. *Array* di *looping*, di masukkan ke dalam *class AllPaths*. *Path* yang yaitu ada lima *path* yang dikeluarkan dari jalur *path* mulai *start* sampai *end*.

Semua *node* yang sudah saling terhubung. 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 4.18 di bawah ini.



Gambar 4.18 Representasi DFS pada *Graph*

Representasi DFS pada *graph* di atas bekerja sesuai dengan alur penomoran hingga menuju alur ke node terakhir yaitu "27". Pada *path* pertama :

a. Alur nomor 1 yang mengarahkan dari *node* 2 ke *node* 3 sehingga mencetak *path* 1 2
b. Dilanjutkan alur nomor 2 mengarahkan dari node 3 ke node 10 sehingga *path* 1 2 3

Path yang dihasilkan bisa dilihat di Gambar 4.6 yaitu ada lima *path* yang dikeluarkan dari jalur *path* mulai *start* sampai *end*.

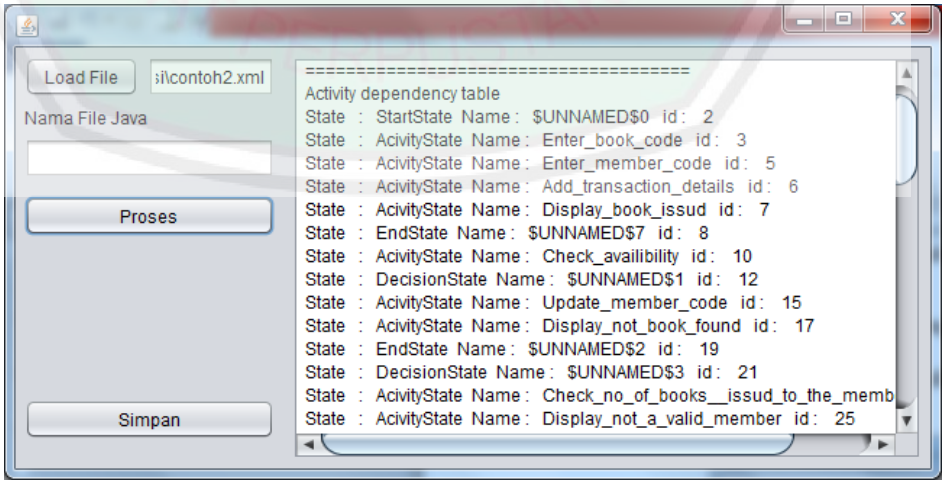
```
47  ========================================
48  All path from StartState to EndState
49   Path from id    2 to id 8
50  [2, 3, 10, 12, 5, 15, 21, 23, 29, 31, 6, 33, 34, 7, 8]
51   Path from id    2 to id 19
52  [2, 3, 10, 12, 17, 19]
53   Path from id    2 to id 27
54  [2, 3, 10, 12, 5, 15, 21, 25, 27]
55   Path from id    2 to id 32
56  [2, 3, 10, 12, 5, 15, 21, 23, 29, 30, 32]
```

Gambar 4.19 Hasil *Path*

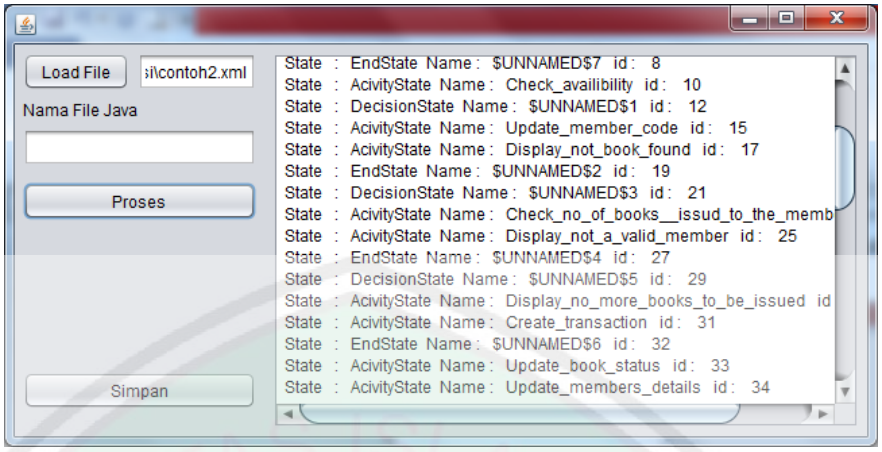
Pada Gambar 4.19 yaitu hasil *path* yang dihasilkan dari 4 jalur. Jalur ini didapat dari *looping* yang dihasilkan oleh *array* sehingga menjadi otomatis

4.1.12 Hasil

Pada pengujian yang kedua yaitu tampilan sama seperti pengujian petama, yaitu dari pengambilan *file .XML*. Selanjutnya yaitu terlihat pada Gambar 4.20 adalah *Activity Depedency Table*.

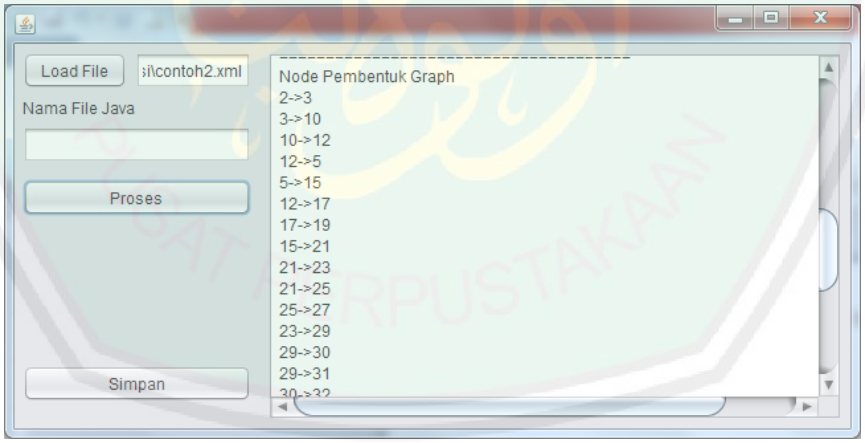


Gambar 4.20 hasil *Activity Depedency Table* otomatis

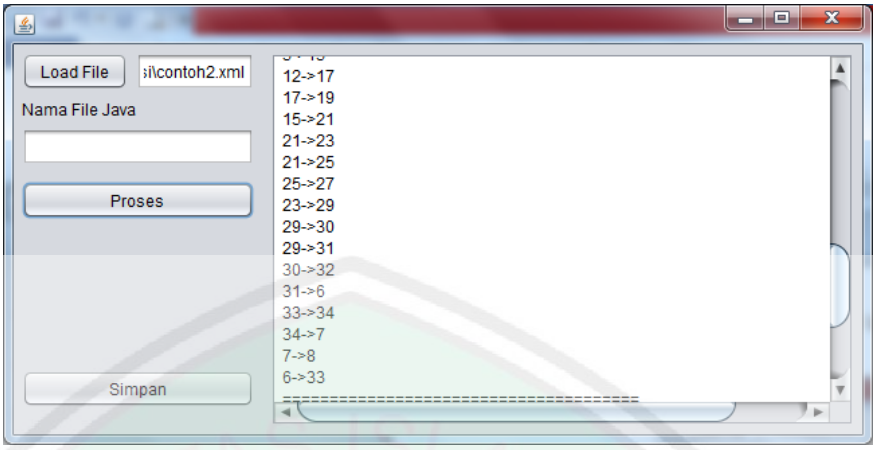


Gambar 4.21 hasil *Activity Depedency Table* otomatis

Gambar 4.20 dan Gambar 4.21 adalah hasil *Activity Depedency Table* otomatis. Pada hasil *Activity Depedency Table* terdapat yang bertuliskan "*unname"* di karenakan pada gambar jurnal tidak ada keterangan sehingga keterangan pada *state* bertuliskan *unname.*

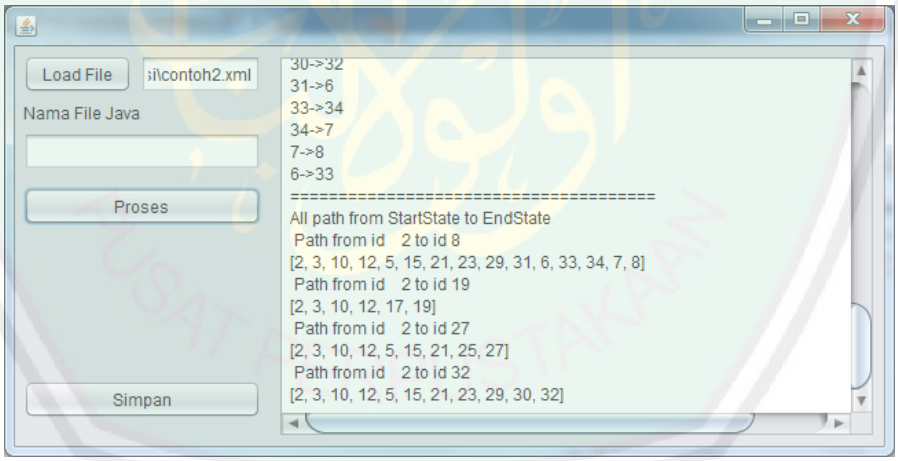


Gambar 4.22 hasil *node* pembentuk *graph*



Gambar 4.23 hasil *node* pembentuk *graph*

Gambar 4.22 dan Gambar 4.23 hasil *Activity Depedency Graph* otomatis di dapat dari menghubungkan node-node yang terdapat pada hasil *Activity Depedency Tale*.



Gambar 4.24 hasil *path* otomatis

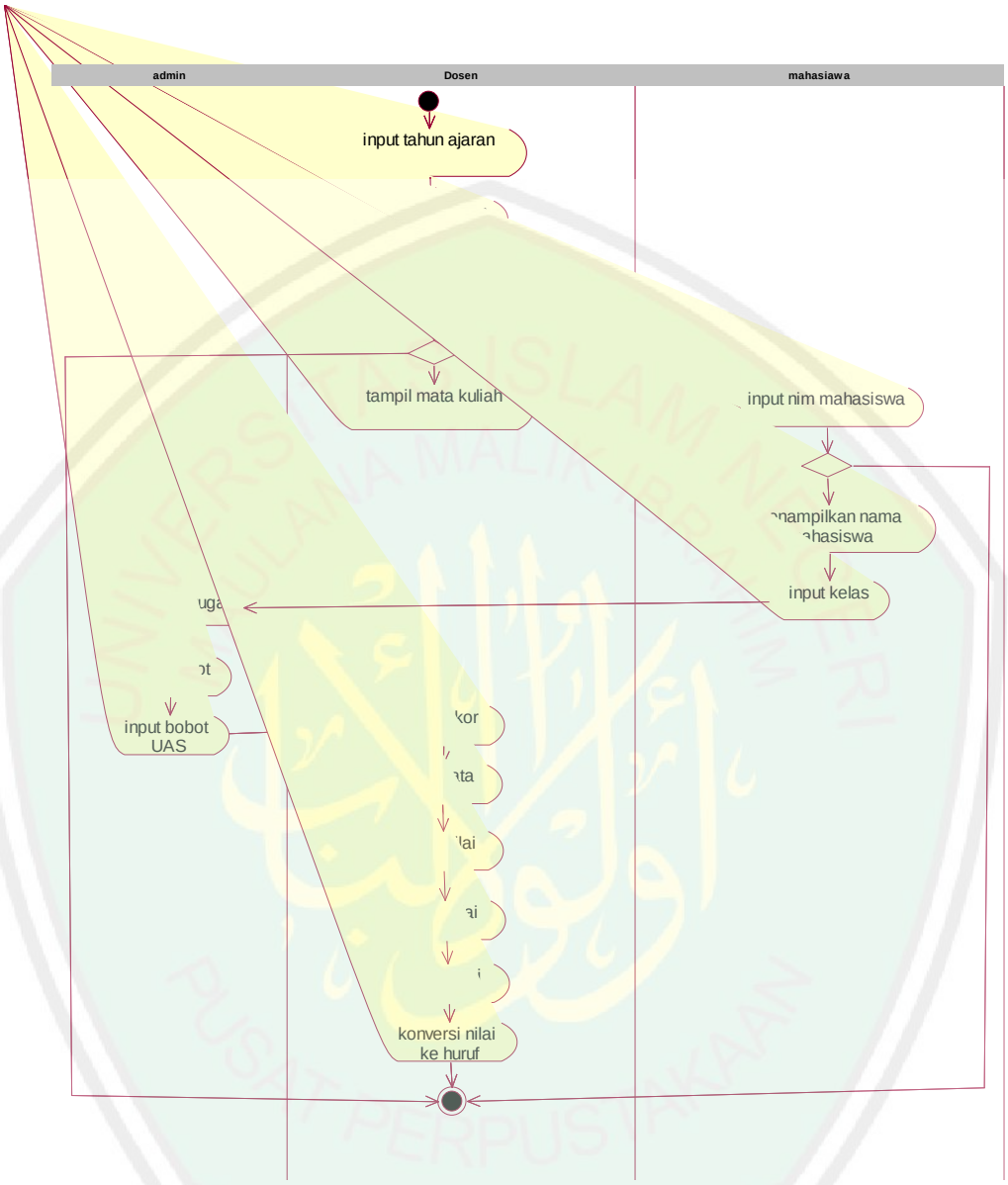
Hasil *path* pada program sama dengan hasil *path* pada jurnal yaitu terdapat 4 jalur. Pada jurnal Gargi Bhattacharjee dan Priya Pati (2014) hasil pathnya yaitu seperti Gambar 4.25

Gambar 4.25 Hasil path jurnal Gargi Bhattacharjee dan Priya Pati (2014)

4.2 Pengujian Sistem Penelitian Pembelajaran

Pada pengujian ketiga yaitu dari *activity diagram* Sistem Penilaian Pembelajaran sebagai data uji untuk aplikasi *tase case* ini. Sistem Penilaian Pembelajaran sebagai bahan uji *activity diagram*. Admin bertugas menginputkan bobot tugas, bobot UTS dan bobot UAS. Dosen menginputkan tahun ajaran, semester kode mata kuliah, jika kode yang di masukkan ada maka akan tampil kode kuliah, jika tidak maka *end*. Selain itu, dosen juga menginputkan skor tugas, rata-rata tugas, nilai UTS dan UAS, nilai akhir dan mengkonversi nilai ke huruf. Mahasiswa menginputkan nim mahasiswa, jika nim ada maka akan tampil nama mahasiwa, jika tidak maka *end*, setelah itu menginputkan kelas. *Activity diagram* ini diterapkan di Sistem Penilaian Pembelajaran. Pada Gambar 2.26 menunjukkan *activity diagram* Sistem Penilaian pembelajaran.

4.2.1 *Activity Diagram*



Gambar 4.26 *Activity diagram* Sistem penilaian Pembelajaran

Pada Gambar 4.26 *Activity Diagram* penerapan pada Sistem Penilaian Pembelajaran.

4.2.2 Konversi ke *XML*

Pada tahap adalah konversi *file* ke *XML* di ambil dari data yaitu *activity diagram bobot.*. Pertama *save as* di tempat *file* yang dinginkan dengan format .*XML*, seperti Gambar 4.27 berikut :

```
80    logical_models  (list unit_reference_list)
81    statemachine    (object State_Machine "State/Activity Model"
82        quid        "5A4DBA2A0118"
83        states      (list States
84        (object ActivityState "input tahun ajaran"
85            quid        "5A4DBA5E03C1")
86        (object State "NewState"
87            quid        "5A4DBA6003DD"
88            type        "Normal")
89        (object State "$UNNAMED$0"
90            quid        "5A4DBA7A0143"
91            type        "StartState")
92        (object ActivityState "input semester"
93            quid        "5A4DBAB703AF")
94        (object ActivityState "input nim mahasiswa"
95            quid        "5A4DBACF032F")
96        (object ActivityState "input kode mata kuliah"
97            quid        "5A4DBAF20123")
98        (object ActivityState "tampil mata kuliah"
99            quid        "5A4DBB560220")
100       (object ActivityState "menampilkan nama mahasiswa"
101           quid        "5A4DBB6B0172")
102       (object ActivityState "input kelas"
103           quid        "5A4DBB8D0225")
104       (object ActivityState "input bobot tugas"
105           quid        "5A4DBBA000EE")
```

Gambar 4.27 Data Uji Model *UML Activity diagram* ke *File* XML sebelum Konversi *File*

4.2.3 Proses Pembuatan *Activity Depedency Table* (ADT)

Tahap selanjutnya yaitu mengambil data data yang diperlukan. *Source code* yang terapkan menggukan *arraylist* yaitu dengan mencari data dari *start* sampai *end.* Berikut adalah hasil pengambilan data yang penting sehingga menjadi *Activity Depedency Table* (ADT) ditunjukan pada Gambar 4.28.

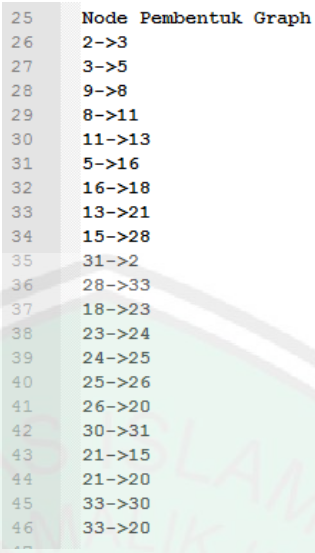
```
2   ====================================
3   Activity dependency table
4   State   :   AcivityState  Name :   input bobot tugas   id :    2
5   State   :   AcivityState  Name :   input bobot UTS   id :    3
6   State   :   AcivityState  Name :   input bobot UAS   id :    5
7   State   :   AcivityState  Name :   input tahun ajaran   id :    8
8   State   :   StartState  Name :   $UNNAMED$0   id :    9
9   State   :   AcivityState  Name :   input semester   id :    11
10  State   :   AcivityState  Name :   input kode mata kuliah   id :    13
11  State   :   AcivityState  Name :   tampil mata kuliah   id :    15
12  State   :   AcivityState  Name :   input skor tugas   id :    16
13  State   :   AcivityState  Name :   input rata tugas   id :    18
14  State   :   EndState  Name :   $UNNAMED$1   id :    20
15  State   :   DecisionState  Name :   $UNNAMED$3   id :    21
16  State   :   AcivityState  Name :   input nilai UTS   id :    23
17  State   :   AcivityState  Name :   input nilai UAS   id :    24
18  State   :   AcivityState  Name :   input nilai akhir   id :    25
19  State   :   AcivityState  Name :   konversi nilai ke huruf   id :    26
20  State   :   AcivityState  Name :   input nim mahasiswa   id :    28
21  State   :   AcivityState  Name :   menampilkan nama mahasiswa   id :    30
22  State   :   AcivityState  Name :   input kelas   id :    31
23  State   :   DecisionState  Name :   $UNNAMED$2   id :    33
24  ====================================
```

Gambar 4.28 *Activity Depedency Table*

Gambar 4.28 adalah data yang sudah di ambil dari beberapa sekian data dalam satu *XML*. Data yang diambil yaitu melalui potongan *Source Code* berikut, yaitu data yang diambil object *StateView, object DecisionView, object Activity StateView, Parent_View, label (object ItemLabel). Id* yang dihasilkan adalah otomatis tertera pada masing-masing data di atas.

4.2.4 Pembuatan *Activity Depedency Graph* (ADG)

Hasil dari tabel di atas mendapatkan *node graph* berdasarkan *ID* yang tertera pada data dari *Activity Depedency Table*.

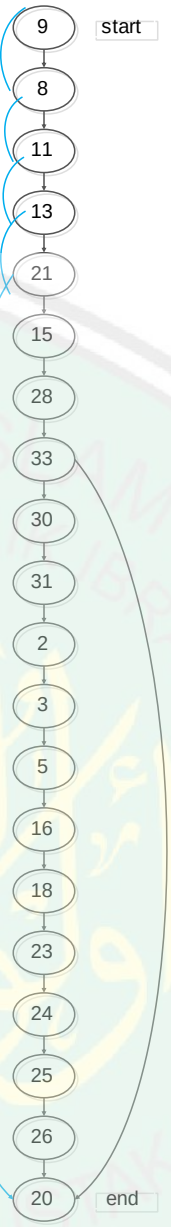


Gambar 4.29 *node graph* dari *Activity Depedency Table*

Pada Gambar 4.29 adalah hasil *node graph* dari *ID* yang sudah ada. *ID* di gabungkan dari *ID* sebelumnya ke *ID* sesudahnya.

4.2.5 Menerapkan Metode *Depth First Search* (DFS)

Dalam mendapatkan hasil *path* dari *activity diagram* data satu, mulai dari *start* sampai *end* menerapkan metode *Depth First Search* (DFS). Pertama di inisialisasi dari *line-line* pada *.XML* yang sudah menjadi *Activity Depedency Table*. Lalu mencari *object value start* dan *end*, maksudnya adalah mencari *path* mana saja yang terdapat dalam *start* sampai *end* lalu di masukkan ke dalam *array*. *Array* di *looping*, di masukkan ke dalam *class AllPaths*.



Gambar 4.30 Representasi *DFS*

Representasi DFS pada *graph* di atas bekerja sesuai dengan alur penomoran hingga menuju alur ke node terakhir ..

Path yang dihasilkan bisa dilihat di Gambar 4.31 yaitu ada lima *path* yang dikeluarkan dari jalur *path* mulai *start* sampai *end*.

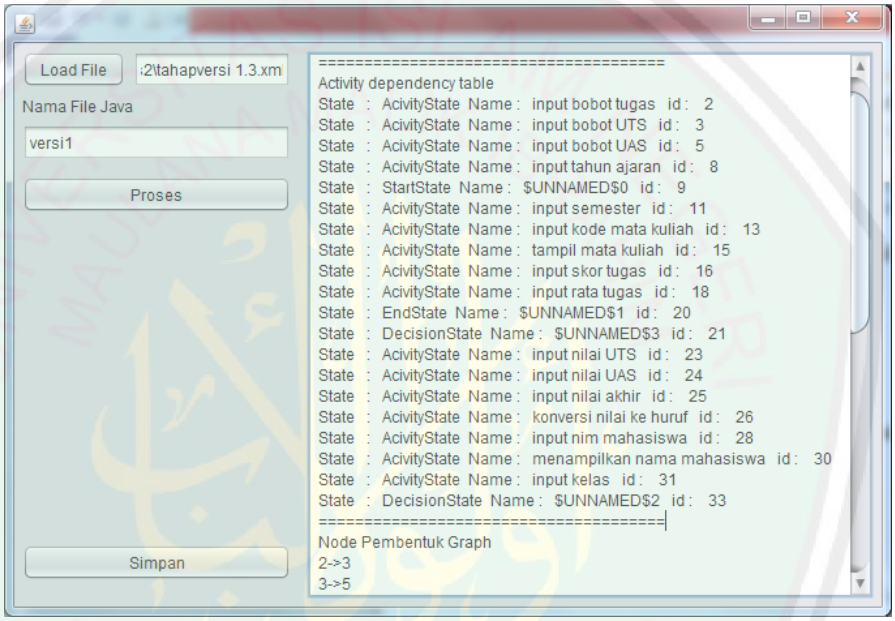
```
All path from StartState to EndState
 Path from id    9 to id 20
[9, 8, 11, 13, 21, 15, 28, 33, 30, 31, 2, 3, 5, 16, 18, 23, 24, 25, 26, 20]
[9, 8, 11, 13, 21, 15, 28, 33, 20]
[9, 8, 11, 13, 21, 20]
```

Gambar 4.31 *path* otomatis

Pada Gambar 4.31 yaitu hasil *path* yang di hasilkan dari perulangan mulai *start* sampai *end* pada jalur *path.*

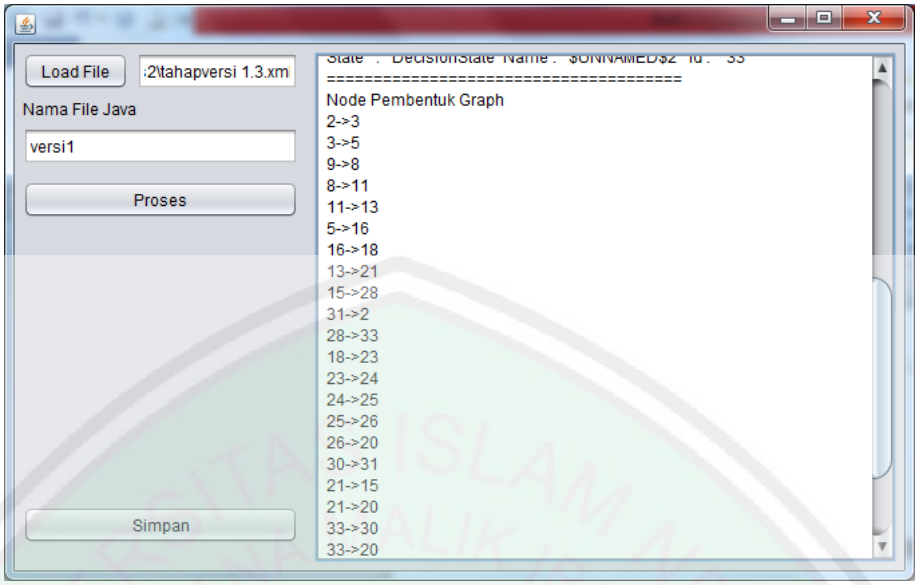
4.2.6 Hasil

Pada *activity diagram* Sistem penilaian pembelajaran,



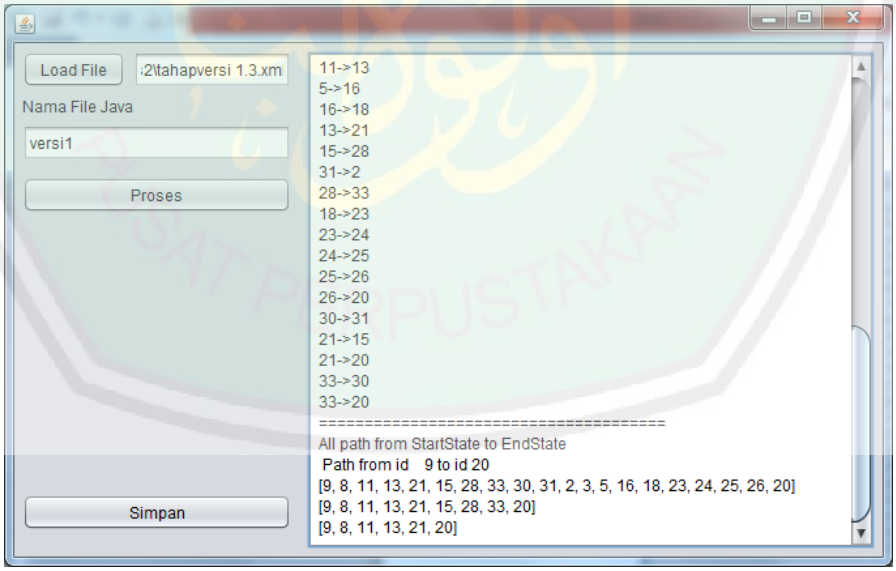
Gambar 4.32 hasil *Activity Depedency Table* otomatis

Gambar 4.32 adalah hasil *Activity Depedency Table* otomatis yang dihasilkan oleh *activity diagram* bobot



Gambar 4.33 hasil *node* pembentuk *graph*

Gambar 4.33 hasil *Activity Depedency Graph* otomatis di dapat dari menghubungkan node-node yang terdapat pada hasil *Activity Depedency Tale*.



Gambar 4.34 hasil *Path* otomatis

Hasil *path* pada program sama dengan hasil *path* secara manual pada data uji sistem penilaian

4.3 Pembahasan

Setelah melakukan uji coba dengan Data Uji di ambil dari jurnal *Activity Diagram* yang digunakan untuk uji coba pengujian yaitu jurnal A.V.K. Shanthi dan G. Mohan Kumar (2012), Gargi Bhattacharjee dan Priya Pati (2014) juga *activity diagram* Sistem Penilaian Pembelajaran.

Pada hasil *path* yang dihasilkan dari jurnal dan *path* dari aplikasi adalah *valid.* Dikatakan *valid* jika hasil *path* dari jurnal dan aplikasi *path* adalah sama.

Tabel 4.1 hasil persamaan *path* yang dihasilkan

Hasil *Path* Jurnal	Hasil *Path* Aplikasi
Start- enter the choice- withdrawal- deposit- enter the amount for deposit-cash deposited- end	[2, 3, 5, 7, 21, 25, 27] *Start- enter the choice- withdrawal- deposit- enter the amount for deposit-cash deposited- end*
Start- enter the choice- withdrawal- deposit- enter the choice correctly -end	[2, 3, 5, 7, 23, 27] *Start- enter the choice- withdrawal- deposit- enter the choice correctly –end*
Start- enter the choice- withdrawal-enter the amount- balance-take cash-end	[2, 3, 5, 9, 11, 13, 27] *Start- enter the choice- withdrawal- enter the amount-balance-take cash- end*
Start- enter the choice- withdrawal-enter the amount- balance-over draft- cash withdrawal from over draft-end	[2, 3, 5, 9, 11, 15, 17, 27] *Start- enter the choice- withdrawal- enter the amount-balance-over draft- cash withdrawal from over draft-end*
Start- enter the choice- withdrawal-enter the amount- balance-over draft- draft limit exceed-end	[2, 3, 5, 9, 11, 15, 19, 27] *Start- enter the choice- withdrawal- enter the amount-balance-over draft- draft limit exceed-end*

Pada Tabel 4.1 yaitu hasil sama antara hasil *path* dari jurnal dan dari aplikasi yang telah dibuat.

Tabel 4.2 hasil persamaan *path* yang dihasilkan

Hasil *Path* Jurnal	Hasil *Path* Aplikasi
Enter the book code- Check availibility- enter member code- validate member code- check no of books issued to the member- create transaction- add transaction detail- update book status- update member details-display book issued- end	[2, 3, 10, 12, 5, 15, 21, 23, 29, 31, 6, 33, 34, 7, 8] *Start- Enter book code- Check availibility- unnamed- enter member code- update member code- unnamed- check no of books issued to the member- unname-add transaction- update book status- update member detail- display book issued- unname*
enter the book code- *Check availibility- Display not book found- end*	[2, *3, 10, 12, 17, 19]* *Start- enter book code- Check availibility- unnamed- Display not book found-unname*
enter the book code- *Check availibility- enter member code- validate member code- display not a valid member- end*	[2, 3, 10, 12, 5, 15, 21, 25, 27] *Start- Enter book code- Check availibility- unnamed- enter member code- update member code-unnamed- Display not a valid member – end*
enter member code- validate member code- check no of books to the member- display no more books to be issued- end	[2, 3, 10, 12, 5, 15, 21, 23, 29, 30, 32] *Start- Enter book code- Check availibility- unnamed- enter member code- update member code- Check no of books issud to the member- unnamed- Display no more books to be issued- end*

Pada Tabel 4.2 yaitu hasil *valid* antara hasil *path* dari jurnal dan dari aplikasi yang telah dibuat. Data uji diambil dari jurnal dimaksudkan untuk validasi jurnal dan program sama, juga mengetahui menghasilkan output yang sesuai paper. Software ini digunakan untuk proses pengembangan tahap analisis, maka mengambil data uji dari jurnal yang terdapat *path* sebagai hasil akhirnya.

Sistem Penilaian Pembelajaran adalah untuk proses penilaian yang dilakukan oleh dosen dengan memasukkan nilai tugas, UTS dan UAS. Dengan demikian Sistem Penilaian Pembelajaran ini merupakan bagian dari sistem

informasi akademik karena pada saat dosen memasukkan penilaian harus dapat menampilkan nama mahasiswa yang akan diberi nilai.

Nilai- nilai ini perlu diolah berdasarkan bobot masing-masing komponen penilaian pembelajaran untuk mendapatkan nilai akhir. Setelah dosen memasukkan nilai dan sistem melakukan proses perhitungan penilaian maka dosen menyimpan hasilnya ke dalam sistem.

4.4 Integrasi Islam

Posisi Al-Qur'an terhadap ilmu pengetahuan dan teknologi dapat dijelaskan dengan jalan mencari sumber ilmu dan sumber cara mengembangkan ilmu menjadi teknologi. Pengembangan teknologi ini meliputi pengujian terhadap *tase case* yang dibuat *output* secara otomatis. Sistem ini dibuat untuk mempermudah dalam pengembangan teknologi di bidang rekasa perangkat lunak yang bermanfaat bagi orang lain.

Hal ini dijelaskan dalam kandungan surat Al-Mulk: 19 , yaitu :

أَوَلَمْ يَرَوْا إِلَى الطَّيْرِ فَوْقَهُمْ صَافَّاتٍ وَيَقْبِضْنَ ۚ مَا يُمْسِكُهُنَّ إِلَّا الرَّحْمَٰنُ ۚ
إِنَّهُ بِكُلِّ شَيْءٍ بَصِيرٌ

> Artinya :" Dan apakah mereka tidak memperhatikan burung-burung yang mengembangkan dan mengatup sayapnya diatas mereka? Tidak ada yang menahan di (udara) selain Yang Maha Pemurah Dia Maha Melihat Segala Sesuatu*"(QS. Al-Mulk: 19).*

Menurut Tafsir Ibnu Katsir oleh Syaikh Shafiyyurrahman al-Mubarakfuri Ayat diatas menegaskan bahwa terbangnya burung dengan kekuasaan Allah, menunjukkan bahwa dia Maha Melihat setiap perkara yang kecil dan yang besar. Kemudian Allah berfirman " dan apakah mereka tidak memperhatikan burung-

burung yang mengembangkan dan mengatupkan sayapnya di atas mereka?" yakni, terkadang burung mengepakkan sayapnya di udara dan terkadang melipatnya dan mengembangkan-nya. " tidak ada yang menahannya , " yakni di udara, " selain Yang Maha Pemurah". karena rahmat dan kelembutannya, dia menundukkan udara untuk burung-burung agar dapat terbang menembusnya.

Ayat diatas menceritakan tentang bagaimana burung bisa terbang mengembangkan sayapnya. Itu karena burung dilengkapi dengan organ-organ tertentu, misalnya sayap, bulu-bulu yang dapat menahan angin dan badan yang lebih ringan dari pada tenaganya, tentu hal serupa juga tidak mustahil bagi manusia untuk bisa terbang, Bila dilengkapi dengan organ-organ yang mampu menerbangkannya. Hai ini pernah dicoba oleh manusia terdahulu ketika mereka mencoba terbang seperti burung. Mereka membuat sayap kemudian diikatkan pada kedua tangannya, lalu terbang dari atas, namun sayang mereka tidak bisa terbang ke atas karena tidak seimbang antara berat badannya dan kekuatan sayapnya.

Pengembangan ini dimaksudkan untuk kemajuan teknologi melalui pengujian *test case* menggunakan *activity diagram*.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil penelitian yang telah dilakukan, dapat disimpulkan bahwa pembangkit *test case* dari *activity diagram* dengan kasus uji Sistem Penilaian Pembelajarn. Sistem Penilaian Pembelajarn menggunakan *activity diagram* sebagai data uji, langkah pertama mengubah *activity diagram* menjadi *activity depedency table* (ADT) , selanjutnya mengubah menjadi *activity depedency graph* (ADG) lalu didapatkan *path* menggunakan metode DFS (*Depth First Search)* untuk mendapatkan hasil akhir. Hasil pengujian, *path* yang telah dihasilkan terdapat tiga *path* yaitu sebagai berikut ini :

1. A- B- C- D- E- T
2. A- B- C- D- E- F- G-H- T
3. A- B- C- D- E-F-G-H-I-J-K-L-M-N-O-P-Q-R-S-T

Penjelasan untuk salah satu pada *path* pertama yaitu terdapat jalur path A- B- C- D- E- T. Simbol A yaitu start, B adalah input tahun ajaran, C adalah input semester, D input kode mata kuliah, T adalah end.

5.2 Saran

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

1. Pada aplikasi belum bisa disimpan secara otomatis dalam setiap pengerjaannya.

2. *Output* secara gambar *graph* belum ada pada aplikasi.
3. Pengujian dapat menggunakan dan mengkombinasikan diagram UML lain misalnya adalah *class diagram*, *sequence diagram* atau *state chart* agar *test case* yang dihasilkan lebih detail dan lengkap.

DAFTAR PUSTAKA

A.Black, J. Champion. *Metode dan Masalah Penelitian Sosial*. Bandung: Refika Aditama, 2009.

A. Abdurazik and J. Offutt. *Using UML collaboration diagrams for static checking and test generation*. In In 3rd International Conference on the UML, pages 383 – 395, 2000.

B, Bezeir. *Software Testing Techiques.* Cet.II. tt: Dreamtech Press, 2004.

Berard, C. *Issues in the Testing of Object-Oriented Software*. 1994

Bhattacharjee, Gargi dan Priya Pati. *A Novel Approach For Test Path Generation and Prioritization of UML Activity Diagrams Using Tabu Search Algorithm.* India: BIT Mesra, 2014.

Ehmer, Mohd Khan. "Different Approaches to White Box Testing Technique for Finding Errors". *International Journal of Software Engineering and Its Applications*. Vol. 5 No. 3. July, 2011.

Farooq, Sheikh Umar dan Quadri. "Evaluating Effectiveness of Software Testing Techniques with Emphasis on Enhancing Software Reliability". *Journal of Emerging Trends in Computing and Information Sciences*. 2011.

feridi. Pentingnya *Belajar Perangkat Lunak*. https://www.codepolitan.com/pentingnya-pengujian-perangkat-lunak. Diakses pada 10 oktober 2017

Fournier, Cs.*Essential Software Testing A Use-Case Approach*.2009.

Fowler, Martin. *UML Distilled Edisi 3*. Yogyakarta: Andi, 2005.

Glover,F. *Candidate List strategies and Tabu search*. University Of Colorado, Boulder: CAAI Research report, 1989.

Henderi. *Unified Modelling Language*. Tangerang: Raharja Enrichment Centre (REC), 2006.

Henderi. *UML: Konsep dan Penerapannya Menggunakan Visual Paradigm*. http://www.blogster.com/henderi/uml-konsep-dan-penerapannya-menggunakan-visual-paradigm-171108195848. Diakses pada 10 Oktober 2016.

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

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

Jaygarl, Hojun dkk. *GenRed: A Tool for Generating and Reducing Object-Oriented Test Cases*. USA: Annual IEE, 2010.

Kim, Hyungchoul, dkk. *Test Case Generation From UML Activity Diagrams.* Korea: Korea Advance Institute of Science and Technology, 2010.

Mingsong, Chen, dkk. *Automatic Test Case Generation for UML Activity Diagrams.* Shanghai: P.R China 210093, 2006.

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

Myers, Glen. *The Art of Software Testing.* Wiley: tt, 1979.

Myers, S.C. "Determininants of Corporate Borrowing". *Journal of Financial Economics*, No. 5, 1977.

Nidhra, Srinivas, dan J. Dondeti. "Black Box and White Box Testing Technique - A Literature Review". *International Journal of Embedded Systems and Applications (IJESA).* Vol.2 No.2. DOI: 10.5121/ijesa.2012.2204, 2012.

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

Nugroho, Adi. *Rekayasa Perangkat Lunak Berbasis Objek dengan Metode USDP*. Jogjakarta: Andi, 2010.

Nuris, Abidin, dan Fatchurrohman. *White Box Testing On The Learning Assement Software Development.* Malang: Green Technology Faculty of Science and Tecnology, 2015.

Oscar Pastor, Cs. *Model-Driven Architecture in Practice, A Software Production Environment Based on Conceptual Modeling*.2007

Pathi, Vikas dan Durga Prasad Mohapatra. *Automatic Test Case Generation using Sequence Diagram.* New York: Fondation of Computer Science FCS, 2012.

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

Pratama, Aditya Rahmatullah. *Belajar UML - Activity Diagram*. https://www.codepolitan.com/mengenal-uml-contoh-uml-diagram-model-activity-diagram. Diakses pada 12 Oktober 2016

Pressman, S. Roger. *Rekayasa Perangkat Lunak.* Cet.VII. Yogyakarta: Andi, 2012.

Shanthi dan Mohan Kumar. *A Novel Approach For Automated Test Path Generation Using TABU Search Algorithm.* Chennai: Sathyabama University, 2012.

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

Swain, Ranjita Kumari, dkk. *Generation of Test Cases Using Activity Diagram.* India: Bhubaneswar, 2013.

Tripathy, Abinash dan Anirban Mitra. *Test Case GenerationUsing ctivity Diagram and Sequance Diagram.* India: Rayagada, 2012.

Trisnawati, Ade. *Implementasi Metode Tabu Search Untuk Penjadwalan Kelas.* Jakarta Barat, 2011.