PEMBANGUNAN KAKAS BANTU PEMBANGKITAN KASUS UJI *BLACK-BOX* BERDASARKAN SKENARIO PENGGUNAAN SISTEM

SKRIPSI

Untuk memenuhi sebagian persyaratan memperoleh gelar sarjana komputer

Disusun oleh:
LAODE MUHAMAD FAUZAN
NIM: 155150200111158

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2019

PENGESAHAN

PEMBANGUNAN KAKAS BANTU PEMBANGKITAN KASUS UJI *BLACK-BOX* BERDASARKAN SKENARIO PENGGUNAAN SISTEM

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Laode Muhamad Fauzan
NIM: 155150200111158

Skripsi ini telah diuji dan dinyatakan lulus pada
27 Juni 2019
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing 2
Bayu Priyambadha, S.Kom, M.Kom	Arief Andy Soebroto, S.T, M.Kom
NIP: 19820909 200812 1 004	NIP: 19720425 199903 1 002

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar referensi.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 9 Juli 2019

Laode Muhamad Fauzan
NIM: 155150200111158

PRAKATA

Puji syukur kehadirat Allah SWT yang telah memberikan rahmat dan karuniaNya, ssehingga penulis dapat menyelesaikan skripsi yang berjudul "Pembangunan Kakas Bantu Pembangkitan Kasus Uji *Black-box* berdasarkan skenario penggunaan sistem" untuk memenuhi salah satu syarat lulus dan mendapatkan gelar sarjana Komputer di Fakultas Ilmu Komputer Universitas Brawijaya.

Penulisan skripsi ini tidak lepas dari bimbingan serta dukungan berbagai pihak, baik secara moril maupun materiil. Oleh karena itu, penulis ingin mengucakan banyak terima kasih kepada :

1. Laode Suharman selaku ayah penulis, Ade Amelia selaku ibu penulis, dan Irvan Rabbani selaku adik saya yang selalu mendukung saya dalam hal apapun.
2. Adi Adrian, Waode Mardiana, dan segenap keluarga besar saya yang selalu mendukung saya dalam hal materil maupun moril.
3. Bapak Bayu Priyambadha, S.Kom, M.Kom selaku dosen pembimbing I yang telah memberikan waktu, nasehat, dan ilmu serta dengan sabar membimbing selama penulisan skripsi.
4. Bapak Arief Andy Soebroto, S.T, M.Kom selaku dosen pembimbing II yang telah memberikan waktu, nasehat, ilmu, dan kesempatan untuk menjadi asisten dosen serta dengan sabar membimbing selama penulisan skripsi.
5. Odhia Yustika Putri sebagai yang tidak perlu dideskripsikan.
6. Dosen dan karyawan Fakultas Ilmu Komputer yang telah memberikan ilmu dan bantuan dalam masa perkuliahan penulis.
7. Jeva, gilang, aji, hanafi, irvan, yuki dan seluruh teman penulis yang telah menjadi bagian penting dalam kehidupan sosial penulis.
8. Rekan – rekan di komunitas robotiika FILKOM yang telah menjadi bagian penting untuk penulis.

Penulis menyadari bahwa dalam penulisan skripsi ini masih terdapat banyak kekurangan. Oleh sebab itu, penulis mengharapkan kritik dan saran yang membangun, sehingga skripsi ini bisa menjadi lebih baik lagi.

Malang, 9 Juli 2019

Penulis

lmfauzan@student.ub.ac.id

ABSTRAK

Laode Muhamad Fauzan, Pembangunan Kakas Bantu Pembangkitan Kasus Uji *Black-box* Berdasarkan Skenario Penggunaan Sistem

Pembimbing: Bayu Priyambadha, S.Kom, M.Kom dan Arief Andy Soebroto, S.T, M.Kom

Pengujian adalah salah satu fase dalam pengembangan perangkat lunak. Umumnya, pengujian perangkat lunak dibagi menjadi 3 fase, yaitu pembuatan kasus uji, eksekusi kasus uji dan evaluasi pengujian. Fase pengujian sistem adalah fase yang memakan waktu dan memakan sumber daya. Salah satu cara untuk mengurangi waktu dan usaha adalah dengan melakukan otomatisasi pada pembuatan kasus uji. Kasus uji dapat diturunkan dari skenario penggunaan sistem. Kasus uji ini disebut kasus uji *black-box*. Untuk menyelesaikan permasalahan yang ada, Penelitian ini membangun sebuah kakas bantu yang dapat membangkitkan kasus uji secara otomatis berdasarkan skenario penggunaan sistem. Skenario penggunaan sistem memiliki format yang berbeda-beda, oleh karena itu pada penelitian ini skenario penggunaan sistem yang akan dipakai adalah skenario dengan format *Cockburn*. Untuk mentransformasi skenario penggunaan sistem, penelitian ini akan menggunakan pemrosesan bahasa alami dan model *EFSM* (*Extended Finite State Machine*). Pemrosesan bahasa alami akan digunakan untuk membangun tabel aktivitas. Tabel aktivitas akan dimodelkan menjadi *EFSM* dan kemudian ditelusuri untuk menghasilkan kasus uji. Pembangunan kakas bantu dilakukan dengan pendekatan berorientasi objek yang mencangkup pemodelan, perancangan, dan implementasi. Kakas bantu dikembangkan dengan teknologi *java* yang memungkinkan sistem dapat dijalankan pada platform yang mendukung *JRE* (*Java Runtime Environment*). Kakas bantu diuji menggunakan pengujian unit, pengujian integrasi, dan pengujian validasi. Pengujian unit dan pengujian integrasi dilakukan dengan menggunakan metode *whitebox*, sedangkan pengujian validasi dilakukan dengan menggunakan *black-box.* Sistem dapat mengenerasi kasus uji dari 20 *use case scenario* dalam waktu 40 detik.

Kata kunci: perangkat lunak, otomatisasi, pengujian, kasus uji *black-box*, *Java, Extended Finite State Machine*, pemrosesan bahasa alami

ABSTRACT

Laode Muhamad Fauzan, Development of Automated Black-box Test Case Generation Tool Based On Use Case Scenario

Adviser: Bayu Priyambadha, S.Kom, M.Kom dan Arief Andy Soebroto, S.T, M.Kom

Testing is part of the software development phase. Generally, testing consist of 3 phase,specifically generate test case, execute test case, and evaluate the testing result. Testing phase is time and resources consuming. One way to reduce the time and resources is by automate the test case generation. Test case can be derived from use case scenario. This kind of test case is called black-box test case. In order to solve the problem, this research develop a tool that can generate test case automatically from use case scenario. Use case scenario has many different formats, this paper will use the Cockburn format. In order to transform the use case scenario, this research will use natural language processing and EFSM (Extended Finite State Machine) model. Natural language processing will be used for generating activity table. The activity table will modeled with EFSM and tranversed to produce a test case. Development of this tool is done by using the object oriented methodology that include modeling, design, and implementation. This research will develop the tool using java that allows the tools to run on platform that supports JRE (Java Runtime Environment). The tools is tested using unit testing, integration testing, and validation testing. Unit and integration will be conducted with whitebox method, while validation testing is done using black-box method. The tool can generate test case from 20 use case scenario in 40 seconds.

Keyword: *software, automation, testing, black-box test case, Java, Extended Finite State Machine, natural language processing*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

DAFTAR PSEUDOCODE

DAFTAR KODE PROGRAM

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Pengujian perangkat lunak merupakan salah satu tahap dalam pengembangan perangkat lunak. Pengujian perangkat lunak dapat didefinsikan sebagai cara untuk memastikan kebenaran dari implementasi sebuah sistem dengan melakukan percobaan pada sistem tersebut (Jiang & Ding, 2011). Umumnya, pengujian perangkat lunak dibagi menjadi 3 fase, yaitu pembuatan kasus uji, eksekusi kasus uji dan evaluasi pengujian (Shanti & Kumar, 2012). Menurut Meiliana et al. (2017) Fase pengujian sistem menjadi aktivitas yang memakan waktu dan membutuhkan banyak sumber daya.

Pembuatan kasus uji adalah fase awal dari pengujian perangkat lunak. Menurut (Ammann & Offutt, 2016) kasus uji adalah sekumpulan dari nilai kasus uji, hasil yang diinginkan, nilai sebelumnya, dan nilai sesudahnya yang dibutuhkan untuk mengeksekusi dan mengevaluasi perangkat lunak yang sedang diuji. Pembuatan kasus uji adalah proses yang memakan waktu dan menjadi tantangan di bawah kendala waktu (Wang et al., 2015). Salah satu cara untuk mengurangi usaha dalam pengujian dan memastikan efektivitas dari pengujian adalah dengan mengenerasi kasus uji dari kebutuhan fungsional selama fase rekayasa kebutuhan, sebuah fase awal dari pengembangan perangkat lunak (Hue et al., 2018). Namun, kebutuhan fungsional perangkat lunak seringkali berubah, sehingga kasus uji sistem harus dibangun dan dieksekusi ulang (Hue et al., 2018). Oleh karena itu, otomatisasi dari pengujian perangkat lunak, terutama dalam pembuatan kasus uji dapat mengurangi waktu yang dibutuhkan untuk melakukan pengujian sistem.

Untuk melakukan otomatisasi pembangkitan kasus uji *black-box* dibutuhkan kebutuhan fungsional sistem yang dapat direpresentasikan dalam sebuah *use case* (kasus penggunaan). Kasus penggunaan sistem dipilih karena bahasa dalam kasus penggunaan sistem lebih mudah dipahami terutama dalam sebuah sistem yang besar dan kompleks. Selain itu kasus penggunaan sistem juga menjamin keterlacakan antara kebutuhan dengan kasus uji sistem (Wang *et al.*, 2015). Namun salah satu kelemahan dari kasus penggunaan sistem adalah bahasanya yang tidak terikat aturan sehingga memiliki sintaks dan semantic yang tidak beraturan, sehingga menjadi dasar yang buruk untuk membangun model formal (Jiang & Ding, 2011). Hal ini dapat menyebabkan kasus uji yang dihasilkan tidak maksimal. Untuk mengatasi hal tersebut, kasus penggunaan sistem yang digunakan pada penelitian ini adalah format kasus penggunaan *Cockburn* (Cockburn, 2001). Format kasus penggunaan *Cockburn* yang akan dipakai dalam penelitian ini selanjutnya akan disebut sebagai skenario penggunaan sistem.

Berdasarkan permasalahan yang ada, penelitian ini akan membangun kakas bantu untuk membangkitkan kasus uji. Penelitian ini akan menggunakan pemrosesan bahasa alami yang merupakan metode untuk menerjemahkan bahasa alami manusia sehingga dapat dipahami oleh komputer. Selain itu penelitian ini akan menggunakan model *EFSM* karena model tersebut sering

digunakan sebagai input untuk verifikasi sistem yang didasari oleh model, serta model ini banyak digunakan untuk pengujian yang didasari oleh model(Petrenko, Boroday and Groz, 2004). Bahasa pemrograman *java* akan digunakan untuk mengembangkan kakas bantu pembangkitan kasus uji. *Java* dipilih karena bahasa tersebut adalah bahasa pemrograman yang paling populer dengan rating 15,035% dibanding bahasa pemrograman lainnya (TIOBE, 2019).

1.2 Rumusan masalah

Berdasarkan latar belakang yang telah dijabarkan, maka rumusan masalah yang didapat adalah :

1. Bagaimana analisis kebutuhan kakas bantu pembangkitan kasus uji *black-box* ?
2. Bagaimana perancangan kakas bantu pembangkitan kasus uji *black-box* ?
3. Bagaimana implementasi kakas bantu pembangkitan kasus uji *black-box* ?
4. Bagaimana hasil pengujian kakas bantu pembangkitan kasus uji *black-box* ?

1.3 Tujuan

Didasari oleh rumusan masalah yang ada, maka tujuan dari pembangunan kkas bantu pembangkitan kasus uji *black-box* adalah :

1. Melakukan analisis kebutuhan kakas bantu pembangkitan kasus uji *black-box*.
2. Merancang kakas bantu pembangkitan kasus uji *black-box* yang menerima masukan skenario penggunaan sistem.
3. mengimplementasikan kakas bantu pembangkitan kasus uji *black-box*.
4. Menguji kakas bantu pembangkitan kasus uji *black-box*.

1.4 Manfaat

Manfaat dari pembangunan kakas bantu pembangkitan kasus uji *black-box* adalah :

1. Mempersingkat waktu penguji sistem untuk membuat kasus uji yang sesuai dengan kebutuhan
2. Membantu pemahaman mahasiswa jurusan ilmu komputer dalam otomatisasi pembangkitan kasus uji *black-box*.

1.5 Batasan Masalah

Batasan masalah dari penelitian ini antara lain:

1. Masukan sistem adalah skenario penggunaan berbasis teks bahasa Inggris dengan format yang sesuai dengan format *Cockburn*.
2. Keluaran sistem adalah sekuens dari aktivitas yang dilakukan pada setiap kasus pengujian.
3. Sistem yang dikembangkan berbasis *desktop*.

1.6 Sistematika Pembahasan

BAB 1 PENDAHULUAN

Latar belakang penelitian, rumusan masalah, tujuan penelitian, manfaat penelitian, batasan masalah, dan sistematika pembahasan terkait pembangunan kakas bantu pembangkitan kasus uji *black-box* dibahas pada bab ini.

BAB 2 LANDASAN PUSTAKA

Kajian pustaka terkait yang digunakan sebagai rerefensi dalam melakukan penelitian serta dasar-dasar teori yang melandasi penulisan dan penelitian pada pembangunan kakas bantu pembangkitan kasus uji *black-box* diuraikan dalam bab ini.

BAB 3 METODOLOGI

Alur kerja peneliti dalam proses penyelesaian permasalahan penelitian pembangunan kakas bantu pembangkitan kasus uji *black-box* diuraikan dalam bab tiga.

BAB 4 REKAYASA KEBUTUHAN

Analisis kebutuhan membahas tentang proses penggalian kebutuhan dalam proses pengembangan sistem, pemodelan sistem berdasarkan hasil analisis kebutuhan terkait pembangunan kakas bantu pembangkitan kasus uji *black-box* akan dibahas pada bab empat.

BAB 5 PERANCANGAN DAN IMPLEMENTASI

Perancangan sistem melalui beberapa diagram serta gambaran sistem dan deskripsi sistem hasil analisis kebutuhan dan perancangan yang diimplementasikan ke dalam perangkat lunak pada pembangunan kakas bantu pembangkitan kasus uji *black-box* dibahas pada bab lima.

BAB 6 PENGUJIAN

Analisis dan hasil pengujian dari penelitian yang telah dilakukan terkait pembangunan kakas bantu pembangkitan kasus uji *black-box* diuraikan dalam bab enam.

BAB 7 PENUTUP

Penutup berisi tentang kesimpulan dari hasil penelitian serta saran untuk pengembangan selanjutnya terkait pembangunan kakas bantu pembangkitan kasus uji *black-box*.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Tabel 2. 1 Tabel kajian pustaka

Judul	Objek	Metode	Hasil
	Input	**Proses**	
Automation of test case generation from textual use cases (Jiang & Ding, 2011)	Data berupa *use case* dengan format *Cockburn use case (*Cockburn, 2001).	• *Post Tagging use case* • Hasil *Post tagging* diolah sehingga menghasilkan tabel aktivitas. • Proses selanjutnya adalah pembentukan model *EFSM*. • Dari model EFSM akan ditelusuri dan dihasilkan kasus uji.	Penelitian menghasilkan kakas bantu yang mengenerasi kasus uji berdasarkan kasus penggunaan sistem dan diujikan pada sebuah sistem informasi pemasaran. Manfaat dari generasi kasus uji tersebut adalah kasus uji dapat dengan muda diupdate jika ada perubahan kebutuhan sistem dan kebutuhan sistem yang bermasalah dapat diketahui lebih cepat.

2.2 Rekayasa Perangkat Lunak

Sebuah aplikasi yang logis, dapat diukur dan terencana mengenai pendekatan dalam perawatan, pengoperasian dan pengembangan suatu perangkat lunak merupakan definisi dari rekayasa perangkat lunak (Pressman,2010). Terdapat beberapa karakterisitik dari perangkat lunak yang dihasilkan,antara lain *reusable, usable, reliable,maintainable, testable, portable, efficient, modifiable, correct,* dan *interoperable* (Leach, Ronald J., 2016).

2.3 Pengembangan Perangkat Lunak

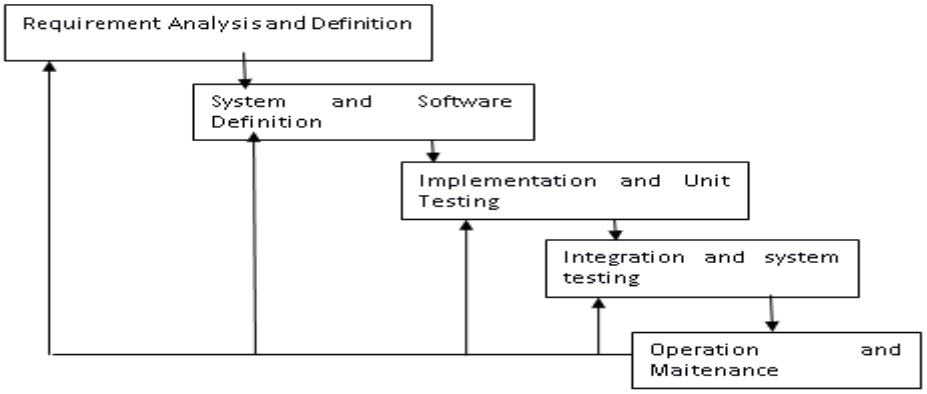
Software evolution, Software specification, Software implementation and dedsign, dan *Software validation* adalah 4 proses yang menjadi pokok untuk pengembangan perangkat lunak (Sommerville, 2011). Dari 4 proses tersebut dikembangkan berbagai siklus hidup perangkat lunak yang disebut SDLC (*Software Development Life Cycle*)., *Spiral development, extreme programming , waterfall, rapid application development, Prototyping,* dan *incremental and iterative development* adalah contoh dari *SDLC*.

2.4 *Software Development Life Cycle*

Teknik dimana sebuah perangkat lunak dikembangkan dalam sebuah pola yang terstruktur sehingga memungkinkan penyelesaian proyek dengan kualitas yang sesuai dengan standar dan waktu yang diinginkan merupakan definisi dari *Software development life cycle*. *Software development life cycle* sering kali digolongkan sebagai penggalan dari sebuah *system development life cycle* (Mishra dan Dubey, 2017)

2.4.1 *Waterfall Model*

Terdapat lima tahapan pada *waterfall model,* yaitu *implementation and unit testing , system and software design, requirement analysis and definition,* dan *, maintenance and operation,* dan *integration and system testing (*Sommervile, 2011).



Gambar 2. 1 Langkah- langkah Waterfall model

Sumber : Sommerville (2011)

Berikut ini penjelasan tahapan-tahapan dari *waterfall model*:

1. ***Requirement Analysis and Definition***
   Tahap analisis kebutuhan sistem, kendala pembuatan, penentuan fitur, dan tujuan sistem melalui konsultasi dengan *stakeholder*.
2. ***System and Software Design***
   Tahapan perancangan antarmuka sistem, perancanggan arsitektur sistem, dan perancangan algoritme berdasarkan hasil dari rekayasa kebutuhan.
3. ***Implementation and Unit Testing***
   Tahap untuk merealisasikan rancangan perangkat lunak menjadi satu unit program dengan menggunakan bahasa pemrograman.
4. ***Integration and System Testing***
   Sebuah tahap untuk menggabungkan unit-unit program dan diuji sebagai satu kesatuan sistem yang utuh untuk memastikan sistem dapat memenuhi ketentuan yang sudah ditetapkan.
5. ***Operation and Maintenance***
   Tahapan penerapan dan instalasi sistem. Pengujian saat *runtime* dijalankan untuk mengetahui error yang sebelumnya belum ditemukan.

2.5 Pendekatan Berorientasi Objek

Gabungan 4 konsep yaitu klasifikasi, objek, komunikasi, dan pewarisan merupakan definisi dari pendekatan berorientasi objek (Pressman, 2010). Terdapat 3 bagian pada pendekatan berorientasi objek, yaitu analisis berorientasi objek, perancangan berorientasi objek, dan pemrograman berorientasi objek.

2.5.1 Pemodelan Berorientasi Objek

Pada pemodelan berorientasi objek, kita berfokus pada analisis dan pemodelan. Analisis berorientasi objek artinya analisis yang berfokus pada pendefinisian kelas dan bagaimana kelas tersebut berkolaborasi (Pressman, 2010).

2.5.1.1 *Use Case*

Use Case (Kasus penggunaan) adalah kontrak yang menggambarkan perilaku sistem dalam berbagai kondisi di mana sistem tersebut memberikan respon dari permintaan salah satu *stakeholders (*Cockburn*,* 2001*)*. Pada intinya, sebuah kasus penggunaan meceritakan bagaimana pengguna sistem berinteraksi dengan sistem dalam berbagai kondisi yang mungkin. Kasus penggunaan dapat berupa teks naratif, deskripsi yang sudah dicontohkan, garis besar tugas atau interaksi, maupun representasi diagram (Pressman, 2010).

2.5.1.2 Format Use Case Cockburn

Use case adalah deskripsi dari kemungkinan urutan interaksi antara sistem yang sedang dibahas dan aktor eksternal, yang berkaitan dengan tujuan tertentu (Cockburn, 2001). Dalam penelitian ini, format *use case Cockburn* disebut sebagai skenario penggunaan sistem karena didalam format tersebut terdapat skenario dalam menggunakan sistem.

Sampel dari penggunaan format *use case Cockburn*

Use case#1:Seller submits an offer. Scope:MP.
Precondition:Seller has logged in.
SuD:MIS.
Primary actor:Seller.
Main scenario.
1 Seller submits item description.
2 System validates the item.
3 System asks Seller to input billing information.
4 Seller enters price and enters contact and billing information.
5 System validates history information.
6 System validates the whole offer with the Trade Commission.
7 Trade Commission gives response.
8 System sends authorization number to Seller.
Extensions.
2a Item is not valid.
2a1 Use case is aborted.
5a History information is not valid.
6a1 Use case is aborted.
7a Trade Commission rejects the offer.
7a1 Use case is aborted.
Variations.
2b Price assessment is available.
2b1 System provides assessment for price.

Gambar 2. 2 *Use Case* dengan Format *Cockburn*

Sumber : Jiang dan Ding (2011)

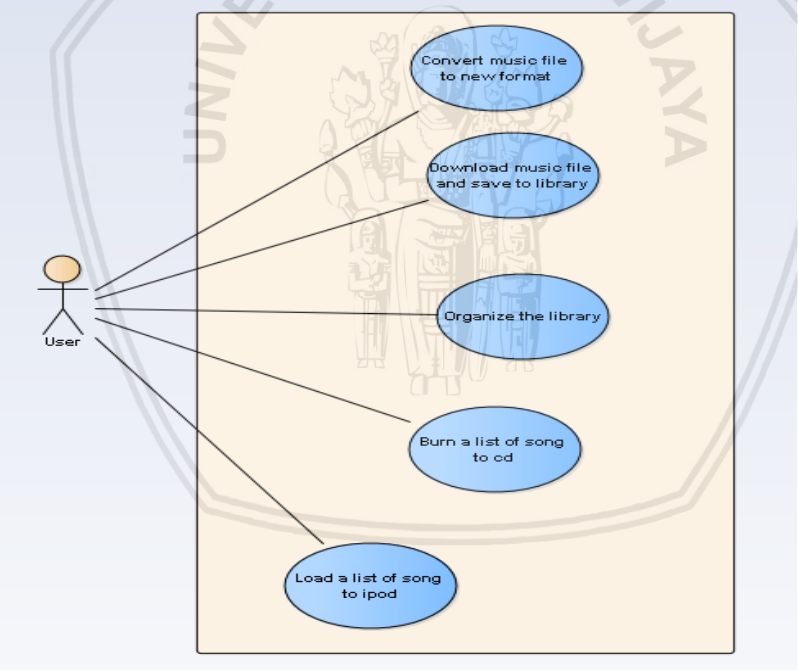
Penjelasan dari use case di atas adalah sebagai berikut (Cockburn,2001) :

a) Use case : Deskripsi dari kemungkina urutan interaksi antara sistem yang dibahas dan aktor,yang berhubungan dengan tujuan tertentu.
b) Precondition : Sesuatu yang harus terjadi sebelum suatu *use case* dapat dimulai.
c) SuD : Sistem yang sedang dibahas.
d) Primary actor : Aktor eksternal yang tujuannya berusaha dicapai oleh *use case*,serta aktor yang menginisiasi suatu *use case*.
e) Main skenario : Skenario utama yang dipilih sebagai dasar dari skenario suatu *use case*.

f) Extension : Extension berguna untuk menyatakan bahwa apa yang dilakukan sistem akan berbeda atau dengan kata lain *extension* adalah percabangan dari *main skenario*.
g) Variations : Variations berfungsi menjelaskan perbedaan teknologi atau metode yang digunakan, namun pada dasarnya fungsi utamanya tetap sama.

2.5.1.3 *Use Case Diagram*

Dimensi perilaku dari suatu sistem yang dimodelkan dalam bentuk diagram dan berguna untuk menjelaskan apa yang harus dapat dilakukan oleh sistem merupakan definisi *use case diagram* (Hariyanto, 2004). Diagram kasus penggunaan bertujuan untuk mendapatkan pemahaman tentang perangkat lunak yang akan dikembangkan oleh pengembang. Diagram kasus penggunaan juga dapat membantu dalam menyusun kebutuhan sebuah sistem dengan cara mengkomunikasikan rancangan terhadap pelanggan. Sampel dari *use case diagram* menurut pressman dapat dilihat di bawah (Pressman, 2010).



Gambar 2.3 Contoh *use case diagram*

Sumber : Pressman (2010)

Penjelasan dari notasi-notasi yang terdapat pada gambar contoh diagram kasus uji pada Gambar 2.2 adalah :

1. *Actor*
   Simbol orang menunjukan pengguna atau pelaku dalam suatu sistem.
2. *Use Case*
   Simbol lingkaran menunjukan tindakan atau pengunaan suatu sistem.

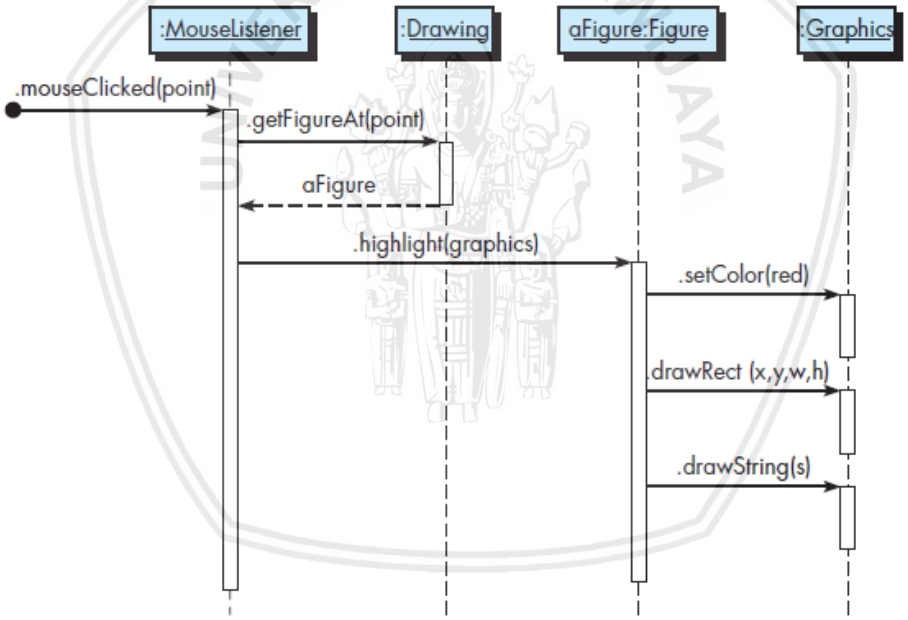
3. Garis panah
   Simbol garis menunjukan hubungan yang terjadi di dalam sistem.

2.5.2 Perancangan Berorientasi Objek

Perancangan berorientasi objek adalah tahap setelah pemodelan berorientasi objek. Pada tahap perancangan ini terjadi perbaikan dan peluasan dari sekumpulan kelas (Pressman, 2010). *Boundary, controller,* dan *entity* akan dikembangkan dan diperbaiki pada tahap perancangan.

2.5.2.1 *Sequence* Diagram

Penggambaran objek pada *use case* dengan mendeskripsikan *life line* objek dan *message* yang dikirimkan dan diterima antar objek dalam bentuk diagram merupakan definisi dari *sequence diagram* (Sukamto & Salahuddin, 2013). Berikut ini merupakan contoh dari *sequence diagram* yang di berikan oleh (Pressman, 2010)



Gambar 2. 4 Contoh Sequence Diagram

Sumber : Pressman (2010)

Penjelasan dari notasi-notasi yang terdapat pada gambar contoh sequence diagram pada Gambar 2.3 adalah:

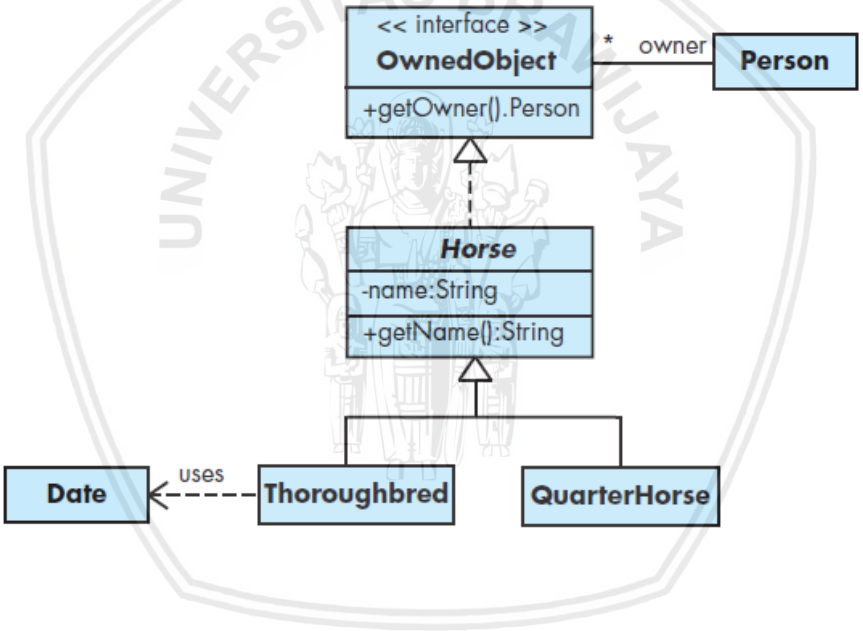
1. *Actor*
   Aktor merupakan entitas yang ada pada *use case* yang sebelumnya dibuat.
2. *Object*
   Persegi panjang yang akan bertukar pesan selama eksekusi suatu *task*.
3. *Lifelines*

Garis vertikal yang dapat bertambah pada aktor dan sistem yang mengindikasikan kehidupan dari sequence.

4. *Activation bars*
   Batang yang akan menutupi *lifelines* sesuai dengan aktivasi suatu objek.
5. *Methods Call*
   Garis panah yang menggambarkan pertukaran pesan dari objek yang mengirim pesan dan menerima pesan.

2.5.2.2 *Class Diagram*

Deskripsi desain sistem dari segi pendefinisian kelas kelas yang berhubungan yang akan dibuat untuk mengembangkan sistem merupakan definisi dari *class diagram* (Sukamto & Salahuddin, 2013). Berikut ini adalah sample *class diagram* yang diberikan oleh (Pressman, 2010)



Gambar 2. 5 Contoh *class diagram*

Sumber : Pressman (2010)

Penjelasan dari notasi-notasi yang terdapat pada gambar contoh *class diagram* pada Gambar 2.4 adalah:

1. *Class*
   Persegi panjang yang terbagi menjadi 3 bagian yaitu bagian atas digunakan untuk nama *class*, bagian tengah digunakan untuk atribut-atribut yang terdapat didalam *class*, bagian bawah digunakan untuk *method* yangterdapat didalam *class*.
2. Garis panah dengan ujung segitiga

Menggambarkan hubungan antar kelas berupa ekstensi atau turunan. Suatu kelas adalah turunan dari kelas yang lain jika memiliki garis panah yang menunjuk pada kelas lain.

3. Garis panah dengan ujung v
   Menggambarkan hubungan antar kelas berupa asosiasi. Hubungan asosiasi berati kelas yang ditujuk oleh tanda panah digunakan pada kelas asal tanda panah. Pada gambar di atas, kelas *Thoroughbred* dapat mengakses kelas *Date*, namun tidak sebaliknya.
4. Garis putus-putus
   Menunjukkan realisasi yang merupakan implementasi dari interface. Pada gambar 2.5 kelas *Horse* mengimplementasi *OwnedObject*.

2.5.3 Pemrograman Berorientasi Objek

Sukamto dan Shalahuddin (2013) menyatakan bahwa "berorientasi objek berarti suatu pendekatan pengembangan perangkat lunak yang teroganisir sebagai kumpulan objek, dimana setiap objek memiliki operasi dan data yang diberlakukan terhadapnya". Secara sederhana pemrograman berorientasi objek adalah pemrograman yang menggunakan konsep objek, kelas, pewarisan, dan komunikasi. Berikut ini adalah beberapa karakteristik yang terdapa dalam PBO (Baesens, Backiel, Broucke, 2015):

1. *Object*
   Pada pemorgraman berorientasi objek, entitas yang didalamnya terdapat *variable* dan *method* merupakan definisi dari *object*. *Object* merupakan sebuah instansiasi dari sebuah *class*. *Blueprint* yang berisi definisi *method* dan *variable* umum pada semua o*bject* disebut sebagai *class.*
2. *Inheritance*
   Pada *inheritance*, terjadi pengelompokkan karakter umum dari parent class dan akan diwariskan ke child class. Semua perubahan pada operasi dan atribut pada *class* induk akan diwarisi *class* turunan.
3. *Polymorphism*
   Polymorphism berguna untuk menyediakan keumuman dengan memungkinkan operasi yang berbeda memiliki nama yang sama. Terdapat dua bentuk dari *polymorphism* yaitu *overriding* dan *overloading. Overriding* berhubungan dengan *inheritance*, sedangkan *overloading* berati beberapa method memiliki nama sama namun parameter berbeda, baik jumlah, urutan, ataupun tipe data.
4. *Encapsulation*
   Penyembunyian data atau operasi suatu objek dari lingkungan luar merupakan definisi dari *encapsulation*. *Encapsulation* akan menjaga integritas suatu data dengan cara mencegah segala akses ke data yang dienkapsulasi, data hanya dapat diakses dan dimodifikasi menggunakan *method* yang telah didefinisikan.

2.5.3.1 Bahasa pemrograman java

Java memungkinkan pengembangan banyak aplikasi yang dapat digunakan pada lingkungan yang berbeda, seperti pada : *Web*, *Mobile*, *Desktop*, dan perangkat lainnya (Supriyatno, 2010). Faktor lain dalam pengembangan sistem menggunakan bahasa pemrograman java adalah karena *Library* pada metode pemrosesan bahasa alami yang akan digunakan (*StanfordNLP*) menggunakan *library* bahasa pemrograman java.

2.6 Pengujian Perangkat Lunak

Pengecekan untuk mendapatkan informasi mengenai layanan yang sedang diuji (*under test*) atau kualitas produk adalah definisi dari pengujian perangkat lunak atau *software testing* (Ammann & Offutt, 2016). Strategi yang umum digunakan pada pengujian sistem berorientasi objek adalah *integration testing, validation testing,* dan *unit testing* (Pressman, 2010).

2.6.1 *Unit Testing*

Berpusat pada pembuktian dari unit terkecil desain perangkat lunak, yaitu komponen atau modul merupakan perhatian utama dalam melakukan pengujian unit. Perancangan komponen dijadikan acuan, jalur control yang utama diuji untuk mengetahui *error* pada batas pada modul. Menurut (Pressman, 2010) Pengujian unit dilakukan menggunakan pendekatan *whitebox* sehingga kode program dibutuhkan untuk mengetahui struktur program.

2.6.2 *Integration Testing*

Pendekatan sistematis yang berguna untuk menguji desain perangkat lunak dan mengetahui *error* yang berkaitan dengan keterhubungan merupakan definisi dari *integration testing*. Tujuan dari *integration testing* adalah mengambil *unit-unit* yang sudah diuji dan menjalankannya sesuai dengan rancangan yang telah ditentukan. Terdapat tiga pendekatan dari *integration testing* yaitu *top-down*, *bottom-up,* dan *big-bang*. Pendekatan *top-down* akan menguji komponen level atas terlebih dahulu sedangkan *bottom-up* akan menguji komponen level bawah terlebih dahulu (Pressman, 2010). *Big bang integration* adalah pengujian dengan menggabungkan semua komponen dan menguji keseluruhan program sebagai kesatuan yang utuh.

2.6.3 *Validation Testing*

Menurut Pressman (2010) *validation testing* adalah pengujian dengan pendekatan *black-box* dimana pengujian akan dimulai setelah *unit* individu telah diuji, perangkat lunak yang dikembangkan telah siap dijalankan, dan *error* yang saling berhubungan telah diperbaiki. Pengujian validasi bertujuan untuk memeriksa aksi dari pengguna dan keluaran dari sistem.

2.6.4 *Black-Box Testing*

Pengujian *black-box* memungkinkan penguji untuk memeriksa sekumpulan masukan dari sistem yang akan menjalankan semua kebutuhan fungsional dari sebuah program (Pressman, 2010). Dapat diterapkan pada tingkatan unit, integrasi, dan sistem, umumnya lebih sering dijumpai pada tingkat yang tinggi (Ammann & Offutt, 2016).

2.6.4.1 Kasus uji *Black-box*

Kasus uji adalah sekumpulan masukan uji, eksekusi kondisi, dan hasil yang diharapkan untuk suatu tujuan tertentu, seperti mencoba bagian tertentu dari sebuah program atau untuk memverifikasi kesesuaian dengan spesifikasi kebutuhan (IEEE, 1990) Dalam pembuatan kasus uji, seorang penguji biasanya mengacu pada kebutuhan sistem yang telah didefinisikan sebelumnya. Oleh karena itu, dalam prosesnya pembuatan kasus uji juga dapat membantu menemukan kebutuhan sistem yang tidak baik. Kasus uji yang dipakai pada penelitian ini adalah kasus uji hasil turunan dari kasus penggunaan sistem Untuk kasus uji *black-box* akan digunakan pada pengujian validasi. Pada pengujian validasi, kasus uji *black-box* akan diturunkan dari rekayasa kebutuhan yang berfokus pada sesuatu yang terlihat oleh pengguna (Pressman, 2010).

2.6.5 Pengujian Performa

Pengujian performa dilakukan untuk memeriksa kinerja perangkat lunak selama *run-time* dalam konteks sebuah sistem yang terintegrasi (Pressman, 2010). Pengujian peforma dilakukan pada setiap tahap mulai dari pengujian unit. Namun, performa sesungguhnya dari sistem hanya dapat dipastikan setelah semua elemen sistem sudah terintegrasi (Pressman, 2010). Menurut sommervile (2011), pengujian performa dirancang untuk memastikan sistem dapat memproses beban yang dibutuhkan. Hal ini biasanya melibatkan serankaian pengujian di mana beban akan terus ditambah.

2.7 Pemrosesan Bahasa Alami

Pemrosesan Bahasa Alami (NLP) adalah pemrosesan otomatisasi bahasa alami manusia, yang dapat didefinisikan juga sebagai proses semi-otomatis. NLP terutama multidisiplin dan terkait dengan linguistik. Biasanya persyaratan dinyatakan dengan kalimat yang ditulis dalam bahasa alami, kalimat-kalimat ini mungkin tidak lengkap dan tidak konsisten. Analisis persyaratan ini dilakukan menggunakan alat pemrosesan bahasa alami (NLP), yang memungkinkan analisis linguistik dan memberikan bantuan otomatis (Elallaoui, Nafil, & Touahni, 2018).

Salah satu alat yang digunakan dalam pemrosesan bahasa alami adalah *StanfordNLP,* merupakan sistem yang dikembangkan leh *The Stanford NLP Group* dengan fokus penelitian pada *sentence understanding*, *probabilistic parsing and tagging*, *biomeical information extraction*, *grammar introduction*,

word sense disambiguation, dan *automatic question answering* (Stanford NLP, 2013)..

Tabel 2. 2 Pelabelan POS tagging StandfordNLP

No	Label	Definisi
1	MD	Modal
2	NNP	Kata benda spesifik, tunggal
3	NNPS	Kata benda spesifik, jamak
4	NN	Kata benda, tunggal atau
5	NNS	Kata benda, jamak
6	PRP	Kata ganti pribadi
7	POS	Akhiran posesif
8	RB	Kata keterangan
9	CC	Kata hubung koordinasi
10	CD	bilangan pokok
11	DT	Penentu
12	EX	Existential there
13	FW	Foreign word
14	IN	Preposition or subordinating conjunction
15	JJ	Kata sifat
16	JJR	Kata sifat, pembanding
17	VBD	Kata kerja, lampau
18	VBG	Kata kerja, gerund
19	VBP	Verb, non-3rd person singular hadir
20	VBZ	Kata kerja, orang ketiga hadir secara tunggal
21	UH	Kata seru
22	VB	Kata kerja, bentuk dasar
23	SYM	Simbol
24	TO	Untuk
25	RBS	keterangan, superlatif
26	JJS	Kata sifat, superlatif
27	LS	Daftar penanda item
28	VBN	Kata kerja, lampau

Sumber: Fauzan (2014)

2.8 Tabel Aktivitas

Untuk membuat model *EFSM* dari *use case scenario* kita harus melakukan ekstraksi data aktivitas dari setiap *use case scenario* (Jiang dan Ding, 2011). Menurut Jiang dan Ding (2011), tabel aktivitas memiliki kolom sebagai berikut

Tabel 2. 3 Header pada Tabel Aktivitas

No.	Activity-Name	Sender	Receiver	A-Condition

Sumber : Jiang dan Ding (2011)

Penjelasan dari header di atas adalah sebagi berikut (Jiang dan Ding, 2011) :

1. No : Urutan aktivitas dari *use case scenario*
2. Activity-Name : Hasil ekstraksi aktivitas dari *use case scenario,* sebelum ekstraksi dilakukan, berkas *use case scenario* akan diolah dengan *Standford Parser* dan hasilnya adalah sebuah berkas *use case scenario* dengan kata yang sudah memiliki *Tag*.
3. Sender : Aktor yang mengirimkan pesan pada *step* dari *use case scenario.*
4. Receiver : Aktor yang menerima pesan pada *step* dari *use case scenario*.
5. A-Condition : Kondisi prasyarat yang harus dipenuhi jika suatu aktivitas akan dijalankan, aktivitas pada *extension* atau *variations* pasti memiliki *A-Condition*.

Pada tabel aktivitas *A-Condition* pada aktivitas pertama selalu merupakan *precondition* dari *use case scenario*. Berikut adalah contoh dari tabel aktivitas yang merupakan esktraksi dari *use case scenario* pada sub-bab 2.5.1.2:

Tabel 2. 4 Contoh Tabel Aktivitas

No.	*Activity-Name*	*Sender*	*Receiver*	*A-Condition*
1	*submitsItem*	*Seller*	*MIS*	*null*
2	*validatesItem*	*MIS*	*Null*	*null*
3	*inputBiling*	*MIS*	*Seller*	*null*
4	*entersPrice*	*Seller*	*MIS*	*null*
5	*validatesHistory*	*MIS*	*Null*	*null*
6	*validatesOffer*	*MIS*	*TC*	*null*
7	*givesResponse*	*TC*	*MIS*	*null*
8	*sendsAuthorization*	*MIS*	*Sellert*	*null*
2a1	*Abort*	*Null*	*Null*	*Cond1*
2a2	*Abort*	*Null*	*Null*	*Cond2*
7a1	*Abort*	*Null*	*Null*	*Cond3*
2b1	*providesAssesment*	*MIS*	*Sellert*	*Cond4*

Sumber : Jiang dan Ding (2011)

Berikut penjelasan mengenai kondisi alternatif pada tabel,

Cond1 = item is not valid
Cond2 = history information is not valid
Cond3 = trade commission rejects the offer
Cond4 = price assessment is available

2.9 Extended Finite State Machine

Sebuah *Extended Finite State Machine* (EFSM) terdiri dari state, variabel, dan transisi antar state. Umumnya sebuah transisi memiliki predikat/penjagaan dan aksi atas variabel di mana predikat harus terpenuhi agar sebuah transisi dapat dijalankan dan selanjutnya aksi yang terkait akan dieksekusi(Kalaji, Hierons dan Swift, 2009).

Modifikasi dari *finite state machine* yang digunakan pada penelitian ini akan menambahkan variabel local dan mengaktifkan predikat boolean untuk transisi *state.* Predikat yang berasosiasi dengan transisi harus bernilai *true* sebelum transisi dapat terjadi(Jiang dan Ding, 2011). *EFSM* yang digunakan pada penelitian ini memiliki 6-*Tuple* M = (S, S_0 , I, O, T, V) di mana :

S = Himpunan state yang tidak kosong

S_0 = Status/*state* awal dari sistem

I = Himpunan interaksi yang tidak kosong

O = Himpunan interaksi keluaran yang tidak kosong

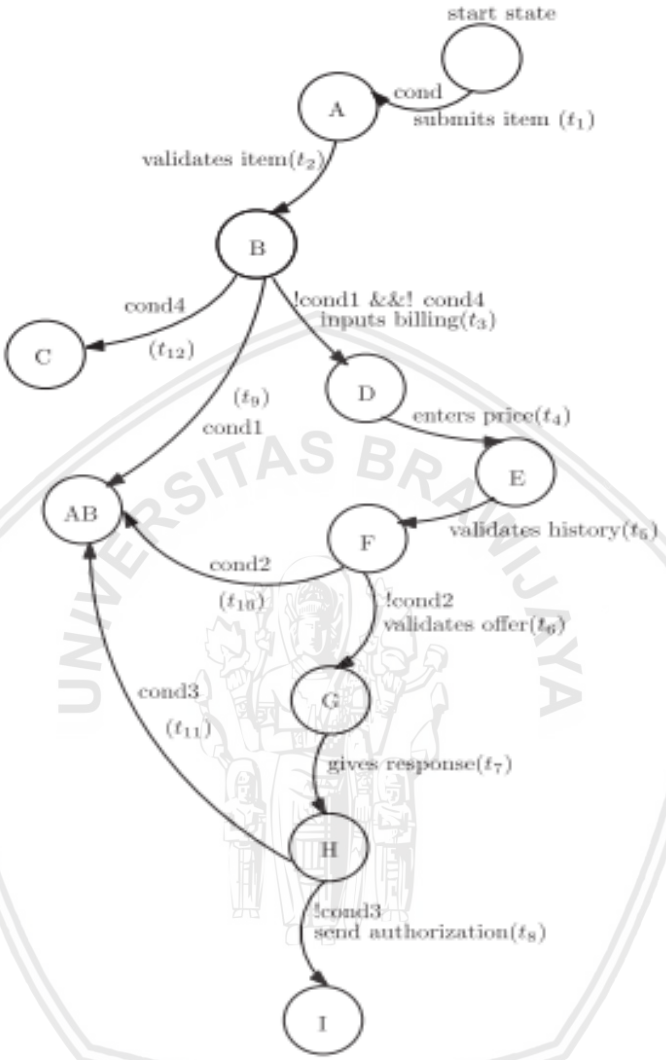
T = Himpunan transisi yang tidak kosong

V = himpunan variabel yang merepresentasikan predikat

Himpunan transisi berisi 5-*Tuples* yaitu, *state* awal dan *state* tujuan yang mirip dengan himpunan state (S), *input* yang merupakan sebuah interaksi dari I atau kosong, *predicate* adalah sebuah variabel yang didefinisikan dari V, dan *block* yang merupakan himpunan tugas dan pernyataan keluaran (Jiang dan Ding, 2011).

Model *Extended Finite State Machine* diigenerasi berdasarkan tabel aktivitas. secara umum setiap aktivits dari sebuah *use case* akan dipetakan menjadi *states* dan *transition* dari yang ada pada model EFSM. Gambar 2.8 adalah *EFSM* model hasil pemetaan tabel aktivitas 2.5. *State* akan menjadi *state* dari sistem ketika aktivitas terjadi. Himpunan Input berisi aktivitas di mana *sender* dari aktivitas adalah aktor. Himpunan *output* adalah himpunan yang berisi aktivitas ketika *sender* adalah sistem. himpunan variabel berisi prekondisi dari aktivitas yang terjadi. Himpunan transisi didapat dari setiap aktivitas pada sistem.

Lingkaran pada model *EFSM* diatas menyatakan *state*. Garis panah menunjukan *transition* dari satu *state* ke *state* yang lain. *Predicate* dapat dilihat pada transisi yang terdapat kata cond-n, hal ini menandakan transisi baru dapat terjadi jika kondisi dari *cond-n* terpenuhi. Semua model *EFSM* selalu diawali dengan *start state* dan *state* tersebut selalu memiliki *predicate* yang merupakan *precondition* dari suatu *use case scenario* (Jiang and Ding, 2011).



Gambar 2. 6 Model *EFSM* dari Tabel Aktivitas 2.5

2.10 Algoritme Depth First Search

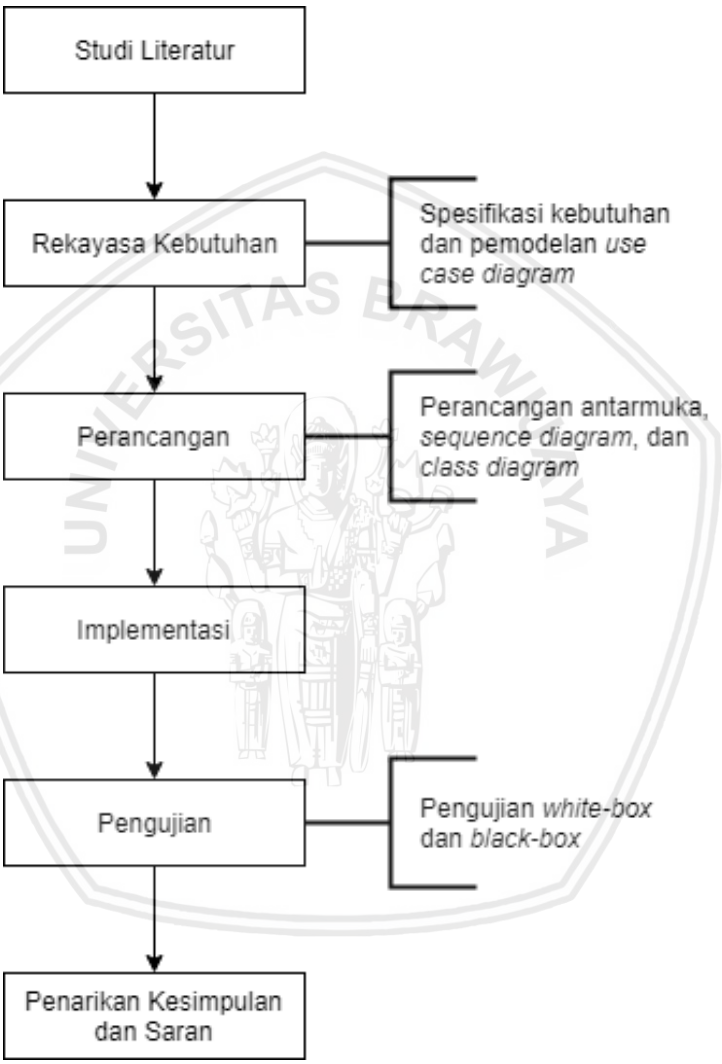
Algoritma Depth First Search (DFS) adalah algoritma pencarian yang mana pencarian akan dilakukan dengan mengexpansi satu level di bawah simpul akar pertama lalu masuk ke level yang lebih dalam sampai simpul yang dituju ditemukan atau simpul tidak lagi memiliki level di bawahnya. Kemudian akan melakukan pencarian ke belakang, kembali ke simpul yang belum ditelusuri. Pada implementasi non-rekursif semua simpul yang baru diexpansi akan ditambahkan pada *stack* untuk penelusuran(Garg & Kaur, 2012). Salah satu kelebihan *DFS* adalah memori yang digunakan sedikit karena bergantung pada perkalian jumlah cabang dan kedalaman solusi (Suyanto, 2014).

2.11 JavaFX

JavaFX adalah perangkat lunak yang membuat pengembang dapat menciptakan dan menghasilkan *Rich Internet Applications (RIAs)* dengan media dan konten yang lintas platform dan perangkat (Gail & Anderson, 2009). Pada *2009 JavaOne Conference*, Sun Microsystem mengumumkan sejumlah perbaruan serta peningkatan terkini pada rangkaian perangkat dan teknologi pengembang Java, yang meliputi JavaFX, antara lain kontrol User Interface (UI) antar platform baru, proses awal aplikasi yang lebih cepat, dukungan media streaming serta peningkatan kinerja lainnya di *desktop, browser* dan *mobile*.

BAB 3 METODOLOGI PENELITIAN

Pada bab metodologi akan dibahas mengenai tahapan-tahapan yang akan dilakukan mengenai penelitian pembangunan kakas bantu pembangkitan kasus uji *black-box*. Untuk langkah-langkah penelitian kedepannya dapat dilihat pada Gambar 3.1 berikut:



Gambar 3. 1 Diagram Alir Metode Penelitian

3.1 Studi literatur

Pada tahap ini studi literatur digunakan sebagai sumber referensi dalam penulisan teori terkait dengan penelitian yang dilakukan penulis. Studi literatur yang digunakan pada pembangunan kakas bantu pembangkitan kasus uji *black-box* berasal dari artikel, jurnal, buku dan penelitian-penelitian yang berhubungan dengan pembangunan kakas bantu pembangkitan kasus uji *black-box*. Studi

literatur akan dijadikan bahan acuan dalam melakukan rekayasa kebutuhan, perancangan, implementasi, dan pengujian dalam penelitian.

3.2 Rekayasa Kebutuhan

Rekayasa kebutuhan bertujuan uuntuk menggali semua kebutuhan yang diperlukan dari kakas bantu pembangkitan kasus uji. Proses dari rekayasa kebutuhan adalah sebagai berikut :

1. Analisis Kebutuhan
   Pengumpulan informasi yang dapat menjadi kebutuhan sistem dilakukan pada tahap ini. Analisis kebutuhan kakas bantu pembangkitan kasus uji dilakukan dengan menganalisis penelitian dari (Jiang & Ding, 2011) yang berjudul *Automation of Test Case Generation From Textual Use Cases.*
2. Spesifikasi Kebutuhan
   Pengelompokkan kebutuhan menjadi kebutuhan fungsional dan kebutuhan non-fungsional pada pembangunan kakas bantu pembangkitan kasus uji.
3. Manajemen kebutuhan
   Pada pembangunan kakas bantu pembangkitan kasus uji dilakukan pengkodean masing-masing kebutuhan fungsional maupun non-fungsional.
4. Pemodelan Kebutuhan
   Penelitian kakas bantu pembangitan kasus uji *black-box* dilakukan dengan pendekatan berorientasi objek dan iiii menghasilkan *use case diagram* dan *use case scenario*.

3.3 Perancangan

Tahap lanjutan tahap analisis kebutuhan, tahap ini dilakukan setelah semua kebutuhan sistem dan aktor/sisi pengguna telah selesai didefinisikan. Perancangan pada pembangunan kakas bantu pembangkitan kasus uji *black-box* mengunakan perancangan berorientasi objek. Pada perancangan berorientasi objek, peneliti akan menggunakan *class* diagram dan *sequence* diagram untuk merancang arsitektur sistem. selain arsitektur sistem juga terdapat perancangan komponen, yaitu perancangan algoritme komponen dengan diambil 5 sampel algoritme yang dirancang. Tahap perancangan yang terakhir adalah perancangan antarmuka dari kakas bantu pembangkitan kasus uji *black-box*.

3.4 Implementasi

Tahap pengembangan sistem seusai tahap perancangan dilakukan dan spesifikasi kebutuhan yang telah didefinisikan. Pada pembangunan kakas bantu pembangkitan kasus uji *black-box* dipilih bahasa pemrograman *Java* sebagai bahasa pemrogramannya. Implementasi yang dilakukan adalah implementasi kode program dan implementasi antarmuka. Perancangan algoritme pada kakas pada pembangunan kakas bantu pembangkitan kasus uji menjadi acuan dari implementasi kode program. Perancangan antarmuka adalah acuan dari implementasi antarmuka pada pembangunan kakas bantu pembangkitan kasus uji.

3.5 Pengujian

Bertujuan untuk memperlihatkan sistem yang telah dibangun dapat bekerja sesuai dengan fungsi yang sudah didefinisikan. Pengujian yang dilakukan yaitu pengujian validasi, unit, integrasi, dan peforma. Pengujian integrasi dan unit pada pembangunan kakas bantu pembangkitan kasus uji *black-box* menggunakan pendekatan *white-box testing* sedangkan *black-box testing* akan digunakan pada pengujian validasi dalam pembangunan kakas bantu pembangkitan kasus uji *black-box. Basis Path Testing* (*BPT)* dipilih untuk melakukan pengujian unit dan integrasi pada pembangunan kakas bantu pembangkitan kasus uji *black-box*. Pengujian performa akan mengukur waktu yang dibutuhkan untuk membangkitkan kasus uji.

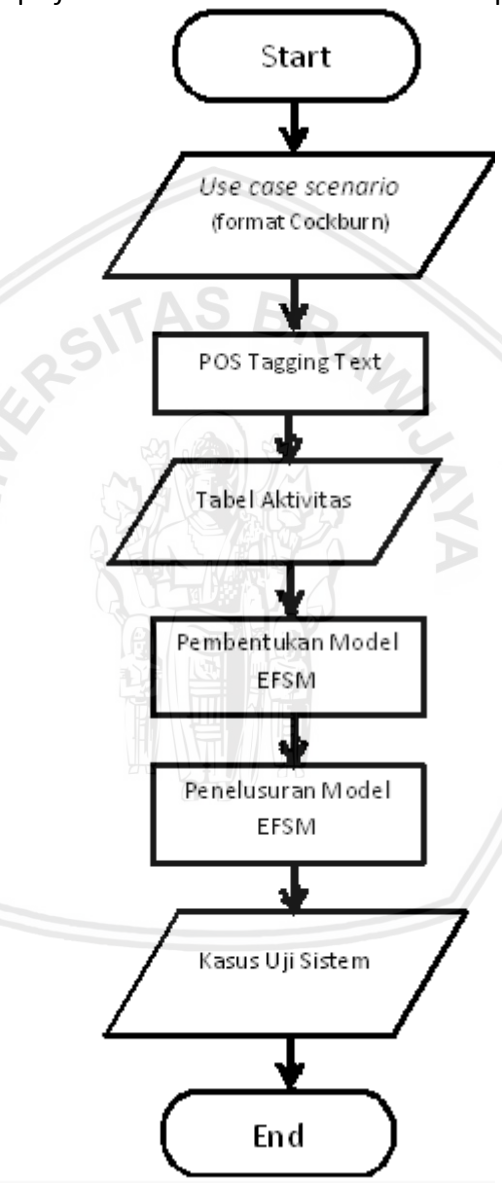
3.6 Penarikan kesimpulan dan saran

Pengambilan berdasarkan hasil analisis pada tahap perancangan, implementasi, rekayasa kebutuhan dan pengujian sistem yang telah diimplementasikan akan menjadi isi dari kesimpulan. Kesimpulan diperoleh untuk memberikan hasil terhadap permasalahan yang telah dirumuskan sebelumnya. Tahap setelah kesimpulan dari penulisan adalah saran yang diberikan guna menjadi referensi dan perbaikan untuk penelitian pembangkitan kasus uji *black-box* selanjutnya.

BAB 4 REKAYASA KEBUTUHAN

4.1 Deskripsi Umum Sistem

Tujuan dari pembangunan kakas bantu adalah menghasilkan kasus uji *black-box*. Masukan pada sistem ini adalah skenario penggunaan sistem dengan format cockburn. Masukan berupa *file* teks dan keluaran sistem berupa *file* teks.



Gambar 4.1 Gambaran Umum Sistem

Gambar di atas menunjukan proses dari kakas bantu pembangkitan kasus uji *black-box*. Terdapat lima tahap yang dilalui.

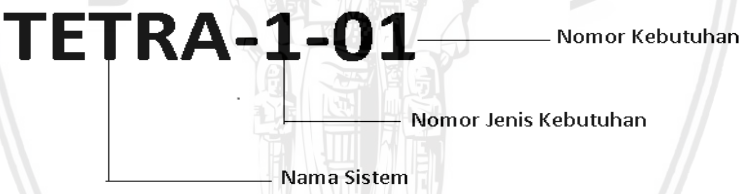
1. Use Case Skenario (Format Cockburn)
   Skenario penggunaan sistem berbahasa Inggris dengan format *Cockburn.*
2. Pos Tagging

Menandai kata dari tiap sebuah teks sesuai dengan struktur bahasa yang sesuai.

3. Tabel Aktivitas
   Tabel yang berisi rangkuman dari aktivitas sistem berdasarkan use case yang dijadikan masukan.
4. Model EFSM
   Konversi tabel aktivitas menjadi model *EFSM* yang berbentuk graph untuk selanjutnya ditelusuri dan dihasilkan kasus uji.
5. Penelusuran model EFSM
   Model EFSM akan ditelusuri dengan algoritma DFS yang dimodifikasi.

4.2 Analisis Kebutuhan

Pada tahap analisis kebutuhan kakas bantu pembangkitan kasus uji *black-box* terdapat tiga hal yang dilakukan, yaitu, analisis kebutuhan fungsional, analisis kebutuhan non-fungsional, dan identifikasi aktor. Kebutuhan didapat dari hasil analisis penelitian oleh (Jiang & Ding, 2011) yang berjudul *Automation of Test Case Generation From Textual Use Cases.* Setiap kebutuhan akan dikodekan secara berbeda berdasarkan aturan penomoran yang ditentukan. Berikut adalah aturan penomoran untuk kebutuhan kakas bantu pembangkitan kasus uji *black-box* yang digunakan:



Gambar 4. 2 Aturan Penomoran Kebutuhan

Penjelasan mengenai gambar di atas adalah sebagai berikut:

1. TETRA: singkatan dari *Test Case Generator* yang merupakan nama kakas bantu yang akan dikembangkan.
2. Nomor jenis kebutuhan: Menandakan suatu kebutuhan bersifat fungsional atau non-fungsional.
3. Nomor Kebutuhan: Urutan kebutuhan sistem yang akan dikembangkan.

4.2.1 Identifikasi Aktor

Pada pembangunan kakas bantu pembangkitan kasu uji *black*-box akan terdapat satu aktor dan deskripsi aktor tersebut. Tabel 4.1 menunjukan aktor yang terdapat dalam sistem yang akan dibangun beserta deskripsinya.

Tabel 4. 1 Identifikasi aktor

Nama Aktor	Deskripsi

Pengguna	Pengguna merupakan aktor yang menggunakan sistem dan dapat menjalankan semua fitur yang ada pada sistem

4.2.2 Analisis Kebutuhan Fungsional

Pada pembangunan kakas bantu pembangkitan kasus uji *black-box* terdapat tiga kebutuhan fungsional. Spesifikasi kebutuhan fungsional berisi layanan yang sistem mampu berikan Pada tabel dibawah kebutuhan ditunjukkan dengan format nomor *Software Requirement Spesification* (TETRA-1-01).

Tabel 4. 2 Daftar kebutuhan fungsional

Nomor Fungsi	Definisi dan Spesifikasi Kebutuhan	Use Case
TETRA-1-00	Sistem harus mampu menerima masukan berkas skenario penggunaan	Memasukan berkas skenario pengunaan
	Sistem hanya memproses berkas dengan ektensi ".txt".(TETRA-1-01)	
TETRA-1-10	Sistem harus mampu mengenerasi kasus uji berdasrkan skenario penggunaan	Membangkitkan Kasus uji
	Sistem akan menampilkan pesan *error* pada berkas hasil pengujian, Jika format kasus penggunaan tidak sesuai ketentuan. (TETRA-1-11)	
TETRA-1-20	Sistem harus mampu menampilkan kasus uji	Membuka berkas kasus uji
	Hasil generasi kasus uji adalah teks dengan ektensi ".txt" (TETRA- 1-21)	

4.2.3 Analisis Kebutuhan Non Fungsional

Kebutuhan yang dapat meningkatkan pengalaman pengguna, namun tidak terlihat langsung adalah definisi dari kebutuhan non fungsional. Pada pembangunan kakas bantu pembangkitan kasus uji *black-box* terdapat satu kebutuhan non-fungsional. Batasan-batasan dalam penggunaan fitur akan didefinisikan pada bagian ini.

Tabel 4. 3 Tabel Kebutuhan non fungsional

Kode Fungsi	Definisi dan Spesifikasi Kebutuhan
TETRA-2-00	Sistem harus dapat mengenerasi kasus uji lebih cepat dari manusia (*Performance*)
	1. Sistem harus mampu mengenarasi kasus uji dari 20 skenario penggunaan sistem dalam waktu kurang dari 120 detik.(TETRA-2-01)

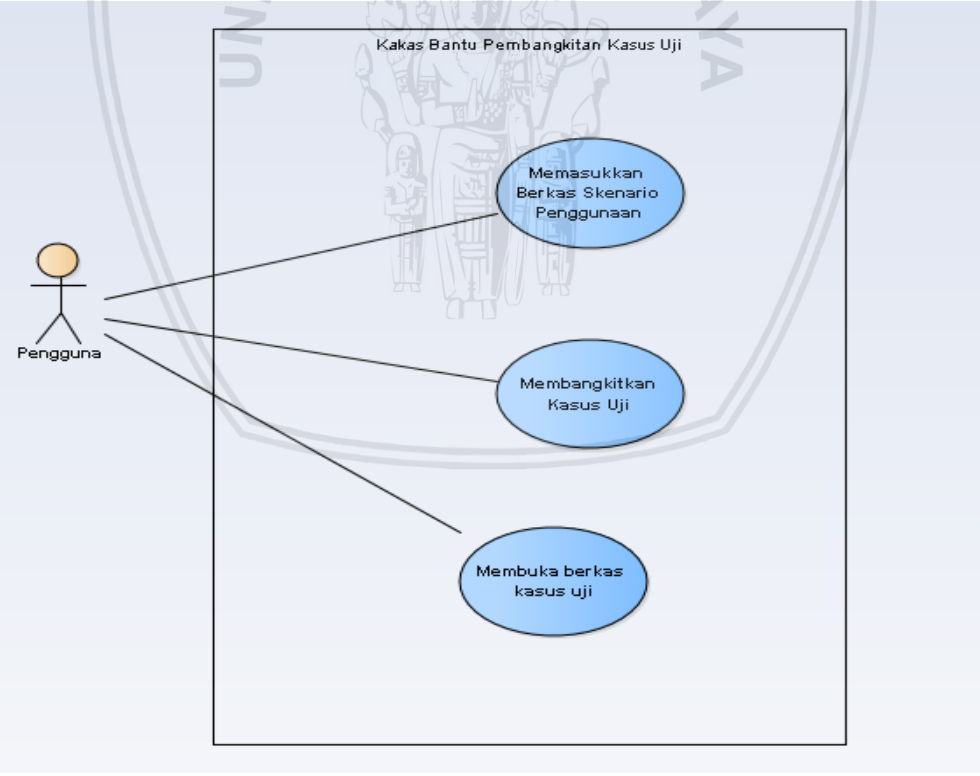
Batas minimum waktu 120 detik diambil percobaan yang dilakukan pada penelitian. Di mana pada satu skenario penggunaan dibutuhkan waktu rata-rata 1 menit untuk mengenerasi kasus uji atau dengan kata lain dibutuhkan 20 menit untuk mengenerasi kasus uji dari 20 *use case scenario*. Dua menit diambil karena penelitian ingin sistem bekerja untuk memenuhi manfaat penelitian, yaitu untuk mempercepat pembuatan kasus uji.

4.3 Pemodelan Kebutuhan

Pada pemodelan kebutuhan, pembangunan kakas bantu pembangkitan kasus uji *black-box* menggunakan pemodelan berorientasi objek. Pada pemodelan berorientasi objek peneliti menggunakan diagram kasus penggunaan dan skenario kasus penggunaan.

4.3.1 Pemodelan Use Case Diagram

Pada pembangunan kakas bantu pembangkitan kasus uji *black-box*, terdapat satu aktor yaitu pengguna. Aktor pengguna dapat menggunakan tiga fitur yang ada pada sistem.



Gambar 4. 2 *Use Case Diagram*

4.3.2 Skenario Use Case

Penguraian lebih mendetail dari kakas bantu pembangkitan kasus uji *black-box*. Jumlah skenario *use case* pada penelitian kakas bantu pembangkitan kasus uji sama dengan jumlah *use case*-nya yaitu tiga buah.

***1. Use case skenario* Memasukkan berkas kasus penggunaan**

Skenario *use case* cari berkas skenario merupakan penjelasan lebih detail dari use case cari berkas skenario.

Tabel 4. 4 *Use case skenario* memasukan berkas skenario penggunaan

Objective	Pengguna dapat mencari dan memilih *file* teks mana yang akan dibuat kasus ujinya
Actors	Pengguna
Pre-Condition	Aplikasi "Kakas bantu pembangkitan kasus uji *black-box*" telah dijalankan pada desktop
Main flow	1. Pengguna menekan tombol *open file* untuk mencari berkas yang diingikan 2. Sistem menampilkan *window browse file.* 3. Pengguna memilih *file* dengan format .txt dalam jumlah yang diinginkan 4. Pengguna menekan tombol *open*
Alternatif flow	Pengguna menekan tombol *cancel* setelah memilih berkas maka berkas tidak akan masuk ke dalam sistem
Post-condtion	Berkas berhasil masuk ke dalam sistem dan ditampilkan nama *file* dari berkas tersebut

2. *Use case skenario* membangkitkan Kasus Uji

Skenario *use case* dapatkan kasus uji merupakan penjelasan lebih detail dari use case membangkitkan kasus uji.

Tabel 4. 5 *Use case skenario* membangkitkan kasus uji

Objective	Pengguna dapat melihat hasil kasus uji yang dibangkitkan sistem
Actors	Pengguna
Pre-Condition	1. Berkas telah dipilih dan tampil pada sistem

Main flow	1. Pengguna menekan tombol *generate* untuk mendapatkan kasus uji 2. Sistem menampilkan *progress bar*
Alternatif flow	Jika berkas *use case skenario* belum dipilih maka sistem menampilkan "*please input file*"
Post-condtion	- Sistem menghasilkan sebuah *file* .txt yang berisi nama kasus penggunaan, jumlah kasus uji yang diperoleh, kasus uji sistem, dan prasyarat kondisi alternatif. - Sistem akan menampilkan jumlah *use case* yang dimasukkan dan jumlah total kasus uji yang dibangkitkan

3. *Use case skenario* membuka berkas kasus uji

Skenario *use case* dapatkan kasus uji merupakan penjelasan lebih detail dari use case dapatkan kasus uji. Tabel 4.6 menunjukan skenario use case untuk membangkitkan kasus uji.

Tabel 4. 6 *Use case skenario* membuka berkas kasus uji

Objective	Pengguna dapat melihat hasil kasus uji yang dibangkitkan sistem
Actors	Pengguna
Pre-Condition	1. Berkas kasus uji telah selesai digenerasi oleh sistem
Main flow	1. Pengguna menekan tombol *open file* 2. Sistem menampilkan kasus uji
Alternatif flow	
Post-condtion	Sistem menampilkan sebuah *file* dengan ektensi ".txt" yang berisi nama kasus penggunaan, jumlah kasus uji yang diperoleh, kasus uji sistem, dan kondisi alternatif. Sistem akan menampilkan *use case file is not supported* bila berkas yang dipilih tidak sesuai dengan format *use case*

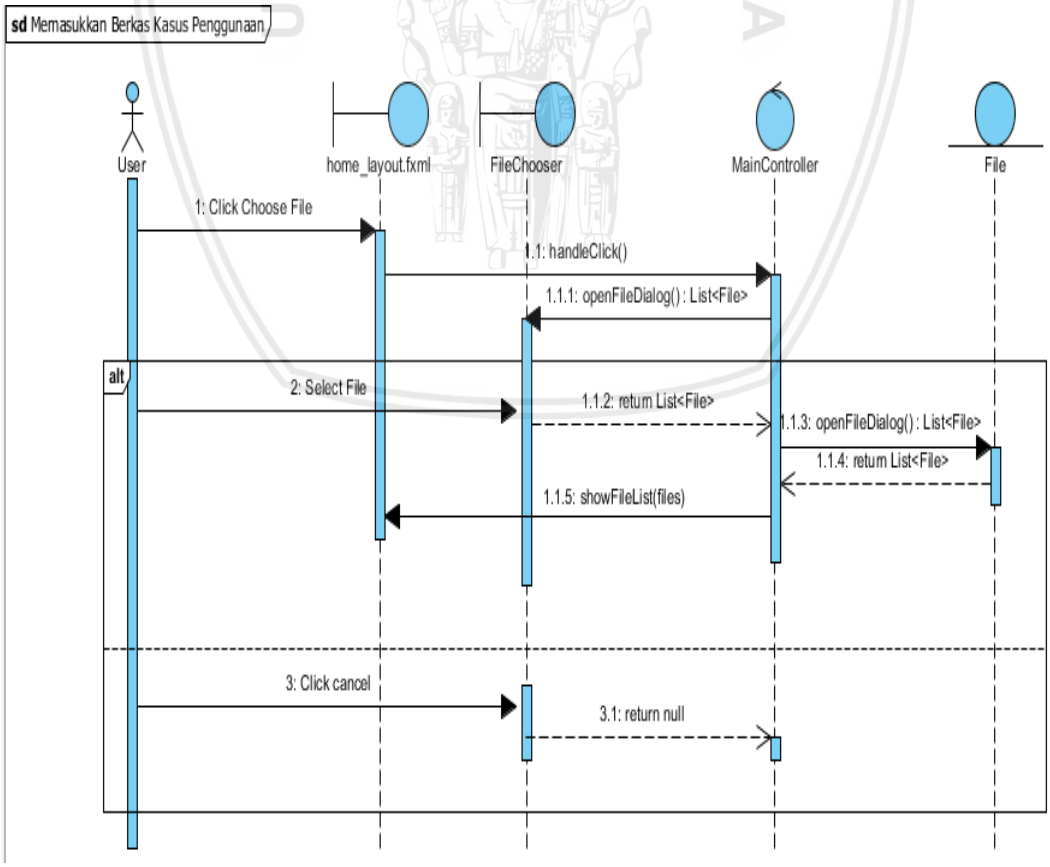
BAB 5 PERANCANGAN DAN IMPLEMENTASI

5.1 Perancangan Arsitektur

Pada perancangan arsitektur, terdapat dua diagram yang akan digunakan yaitu *sequence* diagram dan *class* diagram. *Event* yang menyebabkan transisi antar objek akan tergambar pada *sequence* diagram. Sequence diagram akan menjadi dasar dari diagram klas. Diagram kelas berfungsi untuk menggambarkan strukur dari sistem kelas, atribut, operasi, dan hubungan antar objek.

5.1.1 *Sequence Diagram* Memasukkan Berkas Skenario Penggunaan

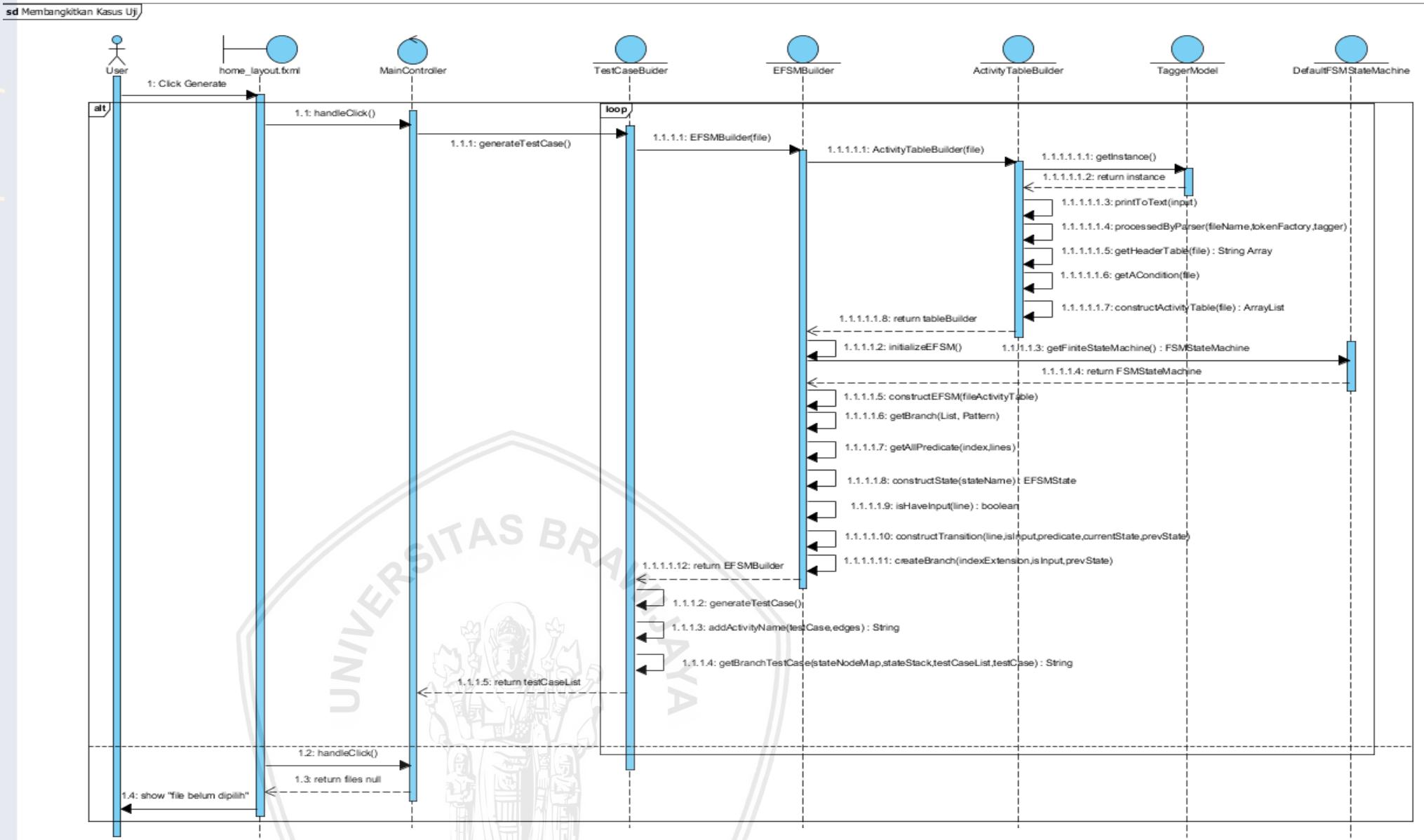
Sequence diagram dari memasukkan berkas kasus penggunaan tergambar pada gambar 5.1. Komponen yang berinteraksi antara lain home_layout yang berfungsi sebagai *view* untuk menerima masukan. Selain itu terdapat Controller yang berfungsi menangani aksi dari pengguna dan memprosesnya. Komponen terakhir adalah *file*_dialog yang berfungsi sebagai *view* untuk memilih dan memasukkan *file* ke sistem. Jika pengguna menekan *cancel* maka *file* tidak akan masuk ke dalam sistem.



Gambar 5. 1 Sequence diagram dari memasukkan berkas skenario penggunaan

5.1.2 *Sequence Diagram* Membangkitkan Kasus Uji

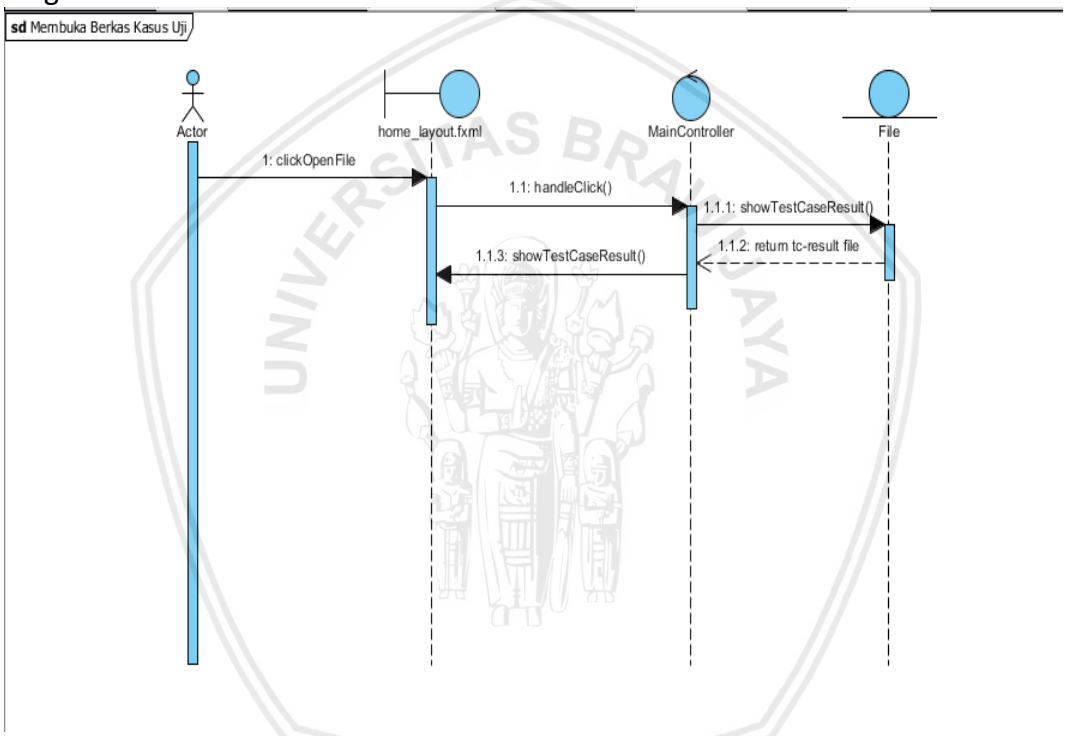
Sequence diagram membagkitkan kasus uji tergambar pada gambar 5.2. Komponen home_layout berfungsi untuk menerima masukan atau aksi pengguna. Controller berfungsi untuk menghubungkan *view* dengan model dan menangani aksi dari pengguna terhadap *view*. Model yang terlibat pada sequence ini antara lain TestCaseBuilder yang berfungsi mengenerasi kasus uji dari model *Extended Finite State Machine (EFSM)* yang dihasilkan oleh model EFSMBuilder. Dalam proses pembuatan *EFSM*, EFSMBuilder menggunakan model DefaultEFSMStateMachine. Model EFSMBuilder akan membuat model EFSM dari tabel aktivtas yang dihasilkan oleh ActivityTableBuilder.



Gambar 5. 2 *Sequence diagram* dari membangkitkan kasus uji

5.1.3 *Sequence Diagram* Membuka Berkas Kasus Uji

Sequence diagram dari membuka berkas kasus uji tergambar pada gambar 5.3. Komponen yang berinteraksi antara lain home_layout yang berfungsi sebagai *view* untuk menerima masukan. Selain itu terdapat MainController yang berfungsi menangani aksi dari pengguna dan memprosesnya. Komponen terakhir adalah *File* yang merupakan kasus uji sekaligus menjadi entity dari sequence diagram.

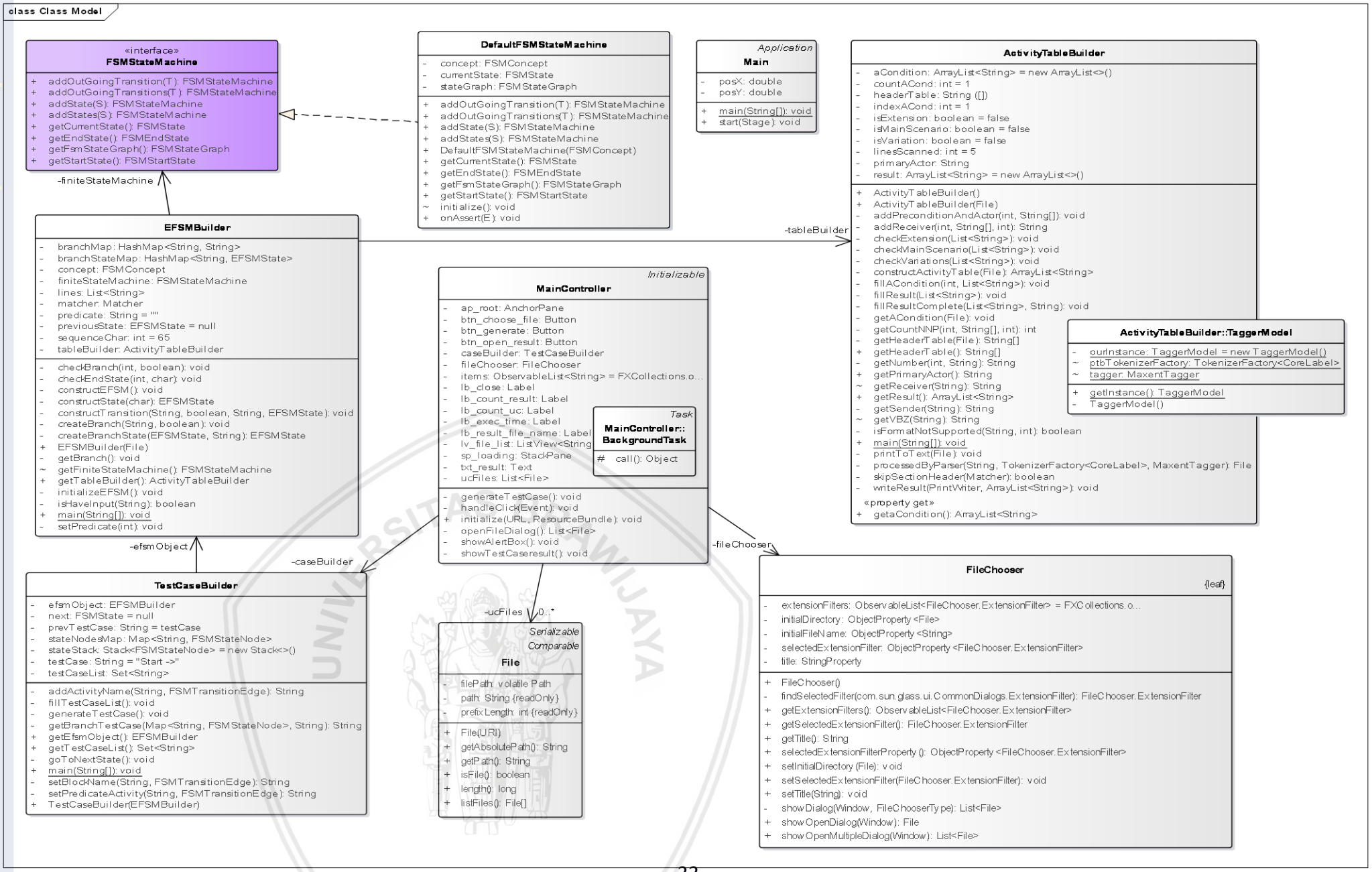


Gambar 5. 3 *Sequence Diagram* Membuka Kasus Uji

5.1.4 Perancangan Class Diagram

Diagram klas dari kakas bantu pembangkitan kasus uji *black-box* terdiri dari beberapa kelas. Kelas controller berfungsi untuk menangani aksi yang dilakukan oleh pengguna. Selain itu terdapat kelas TestCaseBuilder berfungsi untuk mengenerasi kasus uji dari model *Extended Finite State Machine (EFSM)*. Kelas EFSMBuilder berfungsi menangani pembuatan objek *EFSM*, di mana objek EFSM dibuat berdasarkan Tabel Aktivitas. Tabel Aktivitas akan digenerasi oleh kelas

ActivityTableBuilder akan mengerasi tabel aktivitas yang diolah menggunakan analisis kalimat. Kelas *File* dan *FileChooser* berasal dari pustaka Java dan JavaFX.



Gambar 5. 4 *Class Diagram* kakas bantu pembangkitan kasus uji

5.2 Perancangan Komponen

Pada bagian perancangan komponen, peneliti akan membuat algoritme pada beberapa *method* atau fungsi yang ada pada diagram kelas. Algoritma kakas bantu pembangkitan kasus uji *black-box* akan ditunjukkan dengan *pseudocode*. Algoritma yang dirancang pada perancangan akan dijadikan sebagai dasar pada implementasi.

5.2.1 Algoritme *getVBZ*

Algoritme getVBZ merupakan bagian dari algoritme pembuatan tabel aktivitas. Algoritme ini berguna untuk mendapatkan kata kerja atau kata benda dari setiap kalimat pada skenario penggunaan. Setiap kalimat bisa memiliki lebih dari satu kata benda ataupun satu kata kerja. Algoritme ini akan menelusuri tiap kalimat dan menyeleksi apakah kalimat tersebut mengandung tanda "VBZ" atau "NN". VBZ atau NN didapatkan dari hasil *POS Tagging* menggunakan pustaka standford NLP. Algoritme *getVBZ* dapat dilihat pada Pseudocode 5.3.

	Pseudocode algoritme *getVBZ*
1	FUNCTION getVBZ(line):
2	INIT result TO line SPLIT BY " "
3	INIT verb TO StringBuilder OBJECT
4	FOR i = 2 TO i < LENGTH OF result
5	INIT firstChar TO result ON i WITH SUBSTRING FROM 0 TO 1 WITH UPPERCASE
6	IF result ON i CONTAINS "VB" OR result ON i CONTAINS "NN"
7	APPEND verb WITH firstChar,result ON i WITH SUBSTRING FROM 0 TO 1 WITH SPLIT BY "/" ON 0
8	ENDIF
9	ENDFOR
10	RETURN TOSTRING OF verb

Pseudocode 5. 1 Algoritme *getVBZ*

5.2.2 Algoritme *constructActivityTable*

Untuk membangkitkan kasus uji terdapat beberapa algoritme yang digunakan. Pertama adalah algoritme constructActivityTable. Algoritme constructActivityTable tergambar pada Pseudocode 5.2. Pada algoritme constructActivityTable sistem akan membuat tabel aktivitas berdasarkan skenario penggunaan. Proses ini membutuhkan analisis teks, proses analisis teks di sini menggunakan *Standford NLP* sebagai pustaka untuk pemrosesan bahasa alami.

	Pseudocode algoritme constructActivityTable
1	FUNCTION constructActivityTable(File : file)
2	INIT path TO file path
3	INIT lines TO read All Lines OF Files WITH path
4	INIT matcher TO null

```
 5
 6     WHILE linesScanned < size OF lines
 7        CALL checkMainScenario WITH lines
 8        CALL checkExtensions WITH lines
 9        CALL checkVariations WITH lines
10        SET matcher TO matcher OF Pattern with linesScanned
11     IF CALL skipSectionHeder WITH matcher == TRUE
12        linesScanned = linesScanned + 1
13        isMainScenario = FALSE
14        CONTINUE
15     ENDIF
16
17     CALL fillResult WITH lines
18     linesScanned = linesScanned + 1
19     ENDWHILE
```

Pseudocode 5. 2 Algoritme *constructActivityTable*

5.2.3 Algoritme *constructEFSM*

Algoritme ini akan membuat model EFSM berdasarkan tabel aktivitas yang dibuat sebelumnya. Algoritme ini dapat dilihat pada Pseudocode 5.4. setiap baris dari tabel aktivitas akan menjadi *transition* pada *EFSM*. Setiap transition memiliki predicate yang berasal dari alternative kondisi suatu *transition*. Setiap *transition* akan terhubung satu sama lain melalui *state* yang dibuat.

```
    Pseudocode algoritme constructEFSM
 1  PROCEDURE constructEFSM ()
 2
 3      CALL getBranch
 4
 5      FOR each line in lines
 6          INIT matcher TO matcher OF Pattern WITH lines
 7
 8          IF find OF matcher is true
 9              BREAK
10          ENDIF
11
12          CALL setPredicate WITH number of line
13
14          INIT stateName TO char value OF sequenceChar
15          INIT efsmState TO constructState WITH stateName
16          SET sequenceChar TO sequenceChar + 1
17
18          INIT isInput TO isHaveInput WITH lines
19          CALL constructTransition WITH
    lines,isInput,getPredicate,efsmState,previousState
20
21          SET previousState TO efsmState
22
23          CALL checkEndState WITH number of line,stateName
24          CALL checkBranch WITH number of line,isInput
25      ENDFOR
26      concept CALL attachStateMachine WITH finiteStateMachine
```

Pseudocode 5. 3 Algoritme *constructEFSM*

5.2.4 Algoritme *addActivityName*

Algoritme *getActivityName* adalah bagian dari algoritme pembuatan kasus uji. Kasus uji yang dibuat berupa sekuens aktivitas dari yang dihasilkan dari skenario penggunaan. Sekuens aktivitas tersebut didapat melalui algoritme *getActivityName* di bawah. Algoritme *addActivityName* akan melakukan pengecekan dari *predicateKey* yang merupakan penanda bahwa aktivitas tersebut membutuhkan prasyarat untuk dijalankan atau tidak. Selajutnya algoritme tersebut akan mengecek *transition* pada model *EFSM*. Jika *transition* terdapat *blockName* maka aktivitas tersebut adalah *output* dan jika terdapat *actionName* maka aktivitas tersebut adalah *input*. Algoritme *addActivityName* dapat dilihat pada Pseudocode 5.5.

	Pseudocode algoritme addActivityName
1	FUNCTION addActivityName(testCase,edges):
2	Set testCase TO CALL setPredicateActivity WITH testCase,edges
3	IF actionName OF transition OF edges != NULL
4	testCase = testCase + actionName OF transition OF edges
5	ELSE
6	testCase = CALL setBlockName WITH testCase,edges
7	ENDIF
8	RETURN testCase

Pseudocode 5. 4 Algoritme *addActivityName*

5.2.3 Algoritme *generateTestCase*

Algoritme ini akan menelusuri model yang telah dibuat dan menghasilkan kasus uji dari model EFSM yang dibuat dengan algoritme sebelumnya. Algoritme ini akan menggunakan stack ketika melakukan penelsuran tiap node di *EFSM*. Algoritme ini akan melakukan apakah stateNodeMap tidak kosong, jika kosong penelusuran tidak perlu dilakukan. Algoritme generateTestCase dapat dilihat pada Pseudocode 5.5.

	Pseudocode algoritme generateTestcase
1	PROCEDURE generateTestCase
2	SET testCaseList TO object OF linkedHashSet
3	PUSH stateStack WITH StartState FROM stateNodesMap
4	IF stateNodeMap is not empty
5	WHILE stateStack is not empty
6	CALL fillTestCaseList
7	END WHILE
8	ENDIF

Pseudocode 5. 5 Algoritme *generateTestCase*

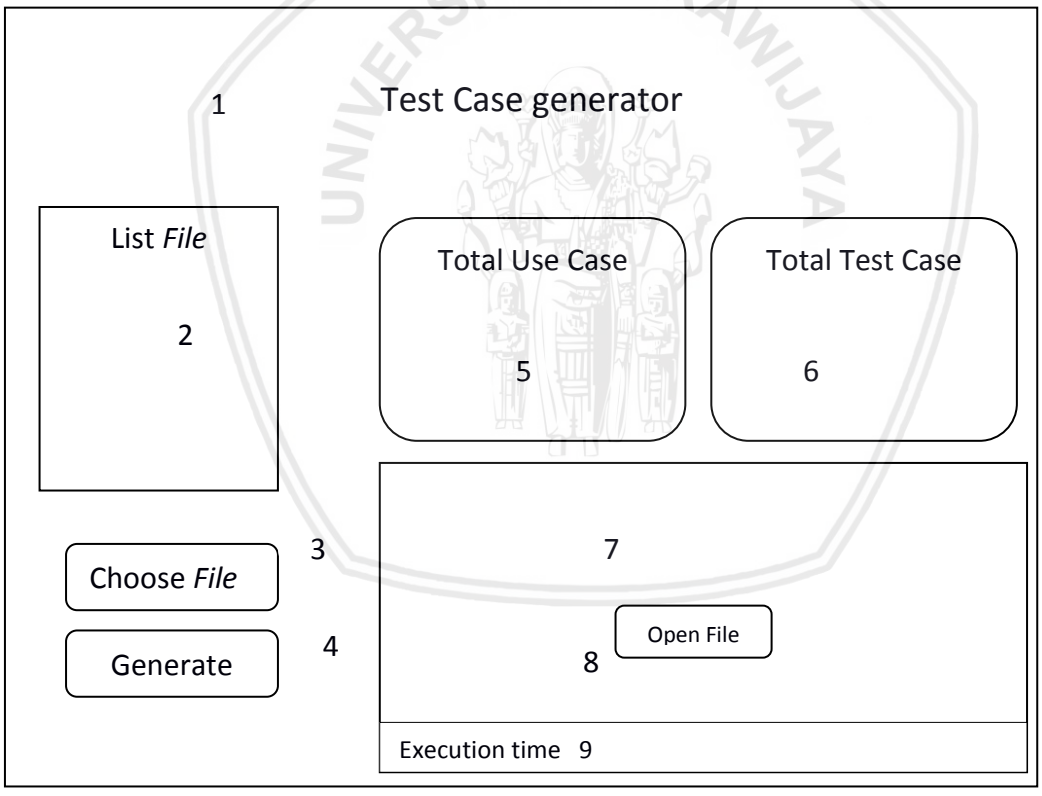
5.3 Perancangan Antarmuka Pengguna

Pada bagian ini akan ditampilkan *wireframe* dari antarmuka kakas bantu pembangkitan kasus uji *black-box*. Tujuan dari perancangan antarmuka pengguna adalah untuk membuat gambaran kerangka tata letak komponen-komponen pada sistem. Secara garis besar terdapat tiga fungsi utama pada antarmuka, yaitu memasukkan berkas penggunaan, generasi kasus uji, dan membuka berkas kasus uji.

5.3.1 Rancangan Antarmuka Tampilan Utama

Pengguna akan dihadapkan pada tampilan utama sistem sesaat setelah menjalankan sistem. Pada tampilan utama terdapat beberapa komponen. Komponen tersebut berguna untuk memasukkan berkas skenario penggunaan sistem dan mengenerasi kasus uji. Pengguna juga dapat melihat file apa saja yang telah dimasukkan ke dalam sistem. Rancangan home_layout dapat dilihat pada gambar 5.4. Terdapat tiga fungsi utama pada tampilan utama, yaitu:

1. Memasukkan berkas skenario penggunaan
   Nama berkas skenario penggunaan akan ditampilkan pada *list view* yang dapat dilihat pada komponen nomor 2.
2. Mengenerasi kasus uji
   Sistem akan menampilkan jumlah skenario penggunaan yang diolah dan jumlah kasus uji yang digenerasi.
3. Membuka berkas kasus uji
   Berkas kasus uji dibuka dengan teks *editor default* sistem.



Gambar 5. 5 Rancangan Antarmuka Memasukkan berkas penggunaan

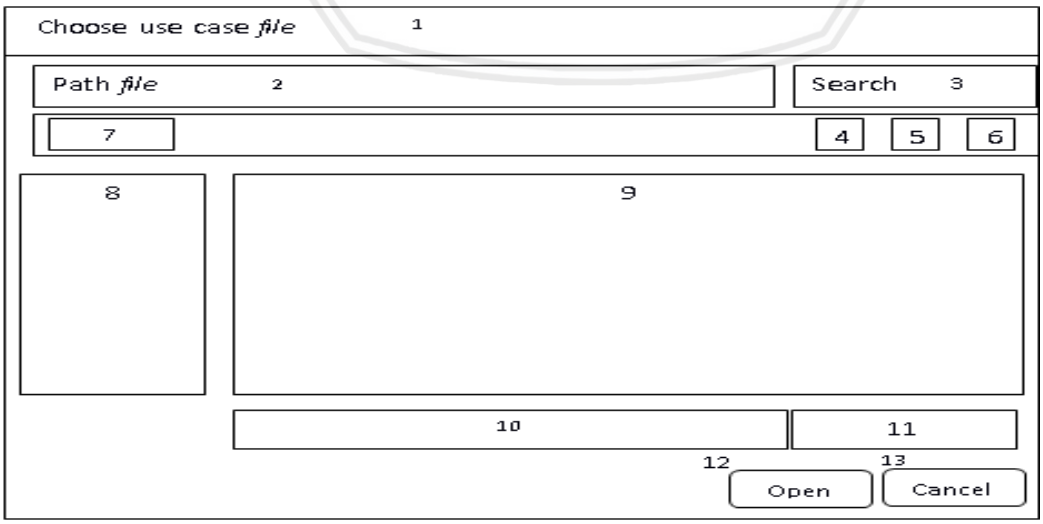
Tabel 5. 1 Deskripsi komponen antarmuka memasukkan berkas penggunaan

Nomor	Nama Komponen	Deskripsi
1	Judul Form	Label judul kakas bantu

		pembangkitan kasus uji *black-box*
2	Daftar berkas use case	Listview yang berisi *path file* yang pengguna pilih
3	Tombol pilih berkas	Tombol untuk akan menampilkan *filechooser*
4	Tombol generasi kasus uji	Tombol untuk berguna menjalankan proses generasi kasus uji
5	Panel total use case	Panel untuk menampilkan jumlah kasus uji yang dimasukkan
6	Panel total test case	Panel untuk menampilkan jumlah hasil kasus uji yang dihasilkan
7	Panel hasil	Panel yang akan tempat dari tombol buka *file* kasus uji
8	Tombol buka *file*	Tombol untuk membuka kasus uji dalam *file* .txt,tombol ini hanya akan berfungsi jika kasus uji telah digenerasi
9	Label execution time	Label yang akan muncul setelah generasi kasus uji selesai dan menandakan waktu yang dibutuhkan untuk melakukan generasi kasus uji

5.3.2 Rancangan Antarmuka FileChooser

File chooser adalah library yang sudah disediakan *java*. *Library* tersebut berfungsi memilih *file* dan memasukkan *file* tersebut ke dalam sistem. *File chooser* akan terbuka ketika pengguna menekan tombol open *file* yang ditunjukkan oleh komponen nomor 3. Rancangan antarmuka *filechooser* dapat dilihat pada gambar 5.4 dibawah.



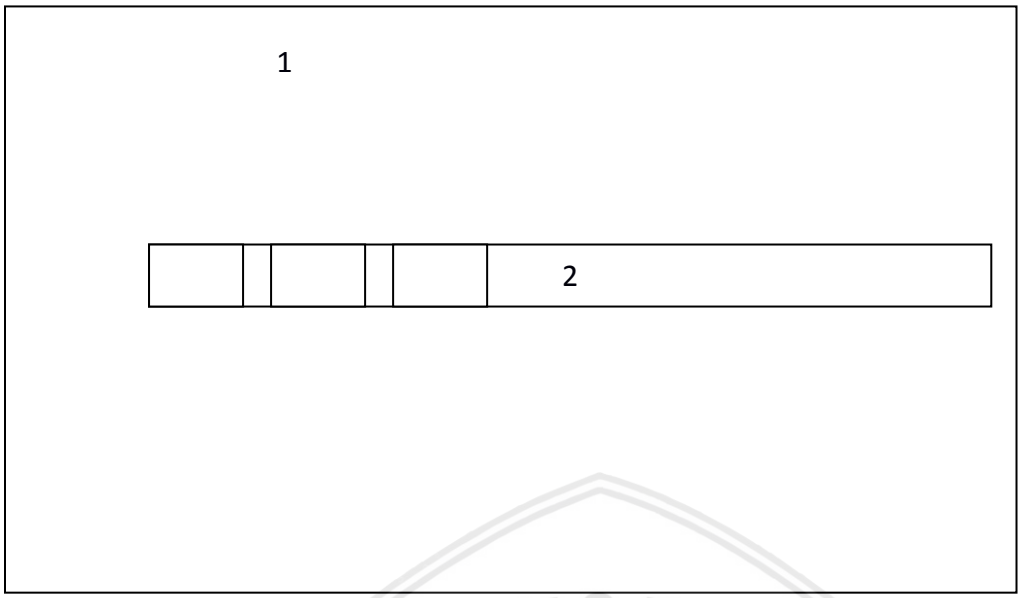
Gambar 5.6 Rancangan antarmuka *FileChooser* Java

Tabel 5. 2 Deskripsi komponen *FileChooser* Java

Nomor	Nama Komponen	Deskripsi
1	Judul *File*Chooser	Judul dari *File*Chooser
2	Path	Komponen yang menunjukkan alamat saat ini
3	Search bar	Isian yang berguna untuk mencari *file* atau direktori
4	*Dropdown view type file*	*Dropdown* yang menampilkan bagaimana tampilan *file* yang pengguna inginkan
5	Tombol Panel *preview*	Tombol untuk menampilkan panel *preview*
6	Tombol Bantuan	Tombol untuk menampilkan halaman bantuan
7	*Dropdown organize file*	*Dropdown* yang berisi bantuan untuk organisasi *file*
8	Panel pindah direktori	Pilihan direktori lainnya
9	Panel isi direktori	Isi dari direktori yang pengguna pilih
10	Nama *file*	Nama *file* yang dipilih
11	*Dropdon* ekstensi *file*	Dropdown yang menampilkan ektensi *file* yang dapat dipilih
12	Tombol pilih berkas	Tombol untuk memasukkan berkas ke dalam sistem
13	Tombol batalkan	Tombol untuk membatalkan proses memasukkan berkas

5.3.3 Rancangan Antarmuka *Loading* Kasus Uji

Sesaat setelah menekan generasi kasus uji, pengguna akan dihadapkan oleh *loading* kasus uji. Loading kasus uji ini terdiri dari *progress bar* dan *stack pane*. Ketika *progress bar* muncul maka pengguna tidak dapat melakukan aksi apapun sampai *progress bar* hilang dan generasi kasus uji selesai. *Progress bar* tampil untuk menandakan bahwa ada suatu proses yang sedang dijalankan sistem. untuk menghalagi pengguna untuk melakukan aksi, rancangan antarmuka ini menggunakan *stack pane.*



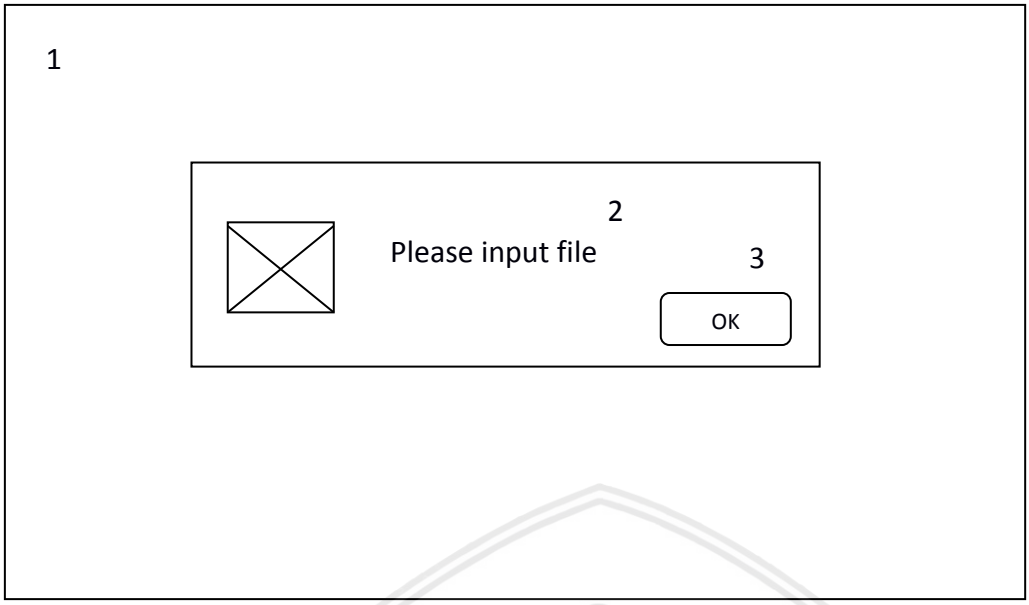
Gambar 5. 7 Rancangan antarmuka *Loading* Kasus Uji

Tabel 5. 3 Deskripsi komponen antarmuka *progressbar*

Nomor	Komponen	Deskripsi
1	*Stackpane*	*Stackpane* berfungsi menutupi *layout* utama ketika proses pembangkitan kasus uji *black-box*
2	*Progress bar*	Indikator yang berjalan ketika proses pembangkitan kasus uji *black-box*

5.3.4 Rancangan Antarmuka Tampilan *Alert Box*

Antamuka terakhir pada pembangkitan kasus uji *black-box* adalah antarmuka tampilan *alert box*. Antarmuka ini akan muncul ketika pengguna belum memilih *file* kasus penggunaan. Pengguna dapat menekans tombol OK untuk menutup *alert box*. Setelah menutup *alert box* pengguna dapat kembali menjalankan aktivitas pada sistem.



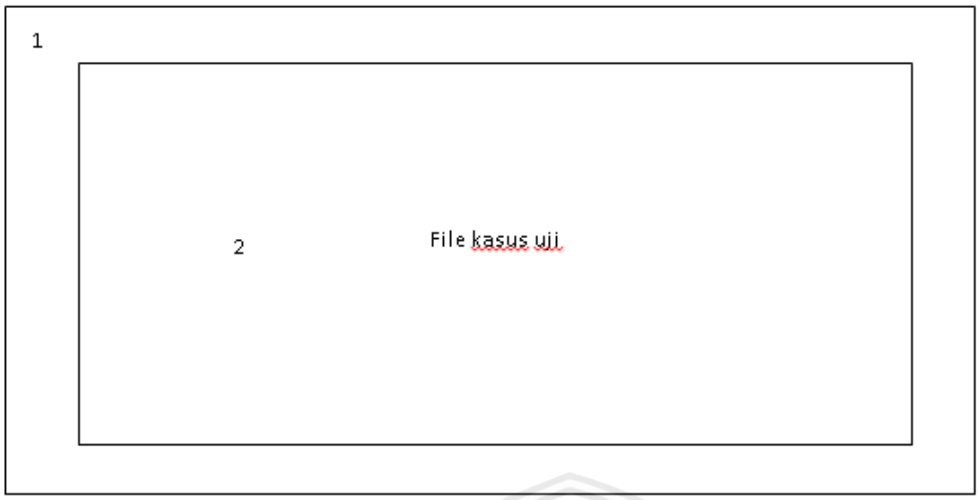
Gambar 5. 8 Rancangan antarmuka tampilan *alert box*

Tabel 5. 4 Deskripsi komponen antarmuka tampilan *alert box*

Nomor	Komponen	Deskripsi
1	Main layout	Tampilan home layout yang dapat dilihat pada gambar 5.4
2	*Alert Box*	Komponen dari *library java* yang berfungsi menampilkan peringatan
3	Tombol OK	Tombol yang berfungsi menutup alert box agar pengguna dapat melanjutkan penggunaan kakas bantu

5.3.4 Rancangan Antarmuka Membuka Kasus Uji

Kasus uji hasil generasi dapat dibuka dengan menekan tombol *ope file* pada tampilan utama. Tombol *open file* hanya akan muncul setelah generasi kasus uji selesai. Setelah menekan tombol *open file* sistem akan menampilkan berkas teks yang berisi kasus uji. Selain kasus uji, terdapat juga nama kasus penggunaan, prasyarat kondisi alternatif, dan jumlah kasu uji yang dihasilkan.



Gambar 5. 9 Rancangan antarmuka membuka kasus uji

Nomor	Komponen	Deskripsi
1	Main Layout	Tampilan home layout yang dapat dilihat pada gambar 5.4
2	*File* Kasus uji	Hasil kasus uji dalam sebuah *file* teks

5.4 Implementasi Sistem

Setelah selesai dengan perancangan, tahap selanjutnya adalah mengimplementasikan perancangan kakas bantu pembangkitan kasus uji *black-box* sesuai dengan perancangan yang dibuat. Implementasikan dilakukan menggunakan bahasa java. Bagian pertama pada tahap ini adalah spesifikasi sistem. setelah spesifikasi sistem terdapat implementasi kode program. Tahap terakhir adalah tahap implementasi antarmuka yang disesuaikan dengan perancangan antarmuka pada bab sebelumnya.

5.4.1 Spesifikasi Sistem

Uraian mengenai spesifikasi perangkat keras dan perangkat lunak yang digunakan untuk mengembangkan kakas bantu pembangkitan kasus uji *black-box* adalah isi dari spesifikasi sistem. Spesifikasi perangkat keras berisi spesifikasi *processor*, *memory*, serta *secondary memory* yang digunakan. Spesifikasi perangkat lunak berisi *tools* yang digunakan dalam proses pengembangan perangkat lunak.

Tabel 5. 5 Spesifikasi Perangkat Keras

Komponen	Spesifikasi
Processor	AMD E1-6010

HDD	500 GB
Main Memori (RAM)	6.00 GB DDR3
Graphics	AMD Radeon R5 M330 DDR3L 2G

Tabel 5. 6 Spesifikasi Perangkat Lunak

Nama Komponen	Spesifikasi
Operating System	*Windows 7 Ultimate*
Programming Language	*Java*
Text Editor / IDE	*Intellij IDEA 2018.3.5 (Ultimate Edition)*
Development Kit	*Java Development Kit 11*
User Interface Builder	*JavaFX Scene Builder 11.0.0*
Editor Dokumentasi	*Microsoft Word 2010*
Editor Perancangan	*Visual Paradigm 12.2,Enterprise Architect 12.0 1210*

5.2.2 Implementasi Kode Program

Implementasi kode program pada pembangunan kakas bantu pembankitan kasus uji *black-box* akan didasari oleh perancangan algoritme komponen yang sebelumnya telah dirancang. Perbedaan dengan bab perancangan adalah pada bab ini pseudocode akan diterjemahkan menjadi bahasa pemrograman *Java*. Terdapat 5 method yang akan dijelaskan pada sesi implementasi yaitu, *method getVBZ*, *method constructActivityTable*, *method constructEFSM*, *method addActivityName*, dan *method generateTestCase*.

5.2.2.1 Implementasi Kode Program *Method getVBZ*

	ActivityTableBuilder.java
1	`private String getVBZ(String line){`
2	`String[] result = line.split(" ");`
3	`StringBuilder verb = new StringBuilder();`
4	`for (int i = 2; i < result.length; i++){`
5	`String firstChar = result[i].substring(0, 1).toUpperCase();`
6	`if (result[i].contains("VB") || result[i].contains("NN")){`
7	`verb.append(firstChar). append(result[i].substring(1).split("/")[0]);`
8	`}`
9	`}`
10	`return verb.toString();`
11	`}`

Kode Program 5. 1 *Method getVBZ*

Pembahasan setiap baris dari kode program *method getVBZ* dapat dilihat pada Tabel 5.7.

Tabel 5. 7 Penjelasan kode program *method getVBZ*

	Kode program *method getVBZ*
1	Deklarasi method getVBZ dengan tipe kembalian string
2	Deklarasi dan inisialisasi variabel result
3	Deklarasi dan inisialisasi variabel verb yang akan menjadi kembalian dari method
4	Looping dari kata kedua parameter line
5	Deklarasi dan inisiasi variabel firstChar yang berguna untuk menjadikan karakter pertama huruf besar
6	Jika result mengandung kata “VB” atau “NN”
7	Isi variabel verb dengan result
10	Mengembalikan nilai verb

5.2.2.2 Implementasi Kode Program *Method constructActivityTable*

	ActivityTableBuilder.java
1	`private ArrayList<String> constructActivityTable(File file) throws IOException {`
2	`Path path = Paths.get(file.getPath());`
3	`List<String> lines = Files.readAllLines(path, StandardCharsets.UTF_8);`
4	
5	`Matcher matcher;`
6	
7	`//COMPLETED: Change this loop to while`
8	`while (linesScanned < lines.size()) {`
9	`checkMainScenario(lines);`
10	`checkExtension(lines);`
11	`checkVariations(lines);`
12	`// Get the number letter number format in variation and extension`
13	`matcher = ConstantDataManager.patternNumberLetterNumber.matcher(lines.get(linesScanned));`
14	`if (skipSectionHeader(matcher)) {`
15	`linesScanned++;`
16	`isMainScenario = false;`
17	`continue;`
18	`}`
19	`fillResult(lines);`
20	
21	`linesScanned++;`
22	`}`
23	`return result;`
24	`}`

Kode Program 5. 2 *Method constructActivityTable*

Pembahasan setiap baris pada *method constructActivityTable* dapat dilihat pada Tabel 5.8.

Tabel 5. 8 Pembahasan kode program *method constructActivityTable*

	Pembahasan kode program method consructActivityTable
1	Deklarasi method constructActivityTable dengan parameter file dan tipe kembalian ArrayList,tipe kembalian dipilih karena jumlah baris tabel

	aktivitas sulit ditentukan dari awal
2	Deklarasi dan inisialisasi path file
3	Deklarasi dan inisialisasi variabel lines dengan semua baris pada file
5	Deklarasi objek matcher untuk menyamakan pola karakter
8	Looping sebanyak jumlah baris
9	Panggil method untuk mengecek header main scenario
10	Panggil method untuk mengecek header extension
11	Panggil method untuk mengecek header variations
13	Cek apakah pola karakter ada yang sama
14	Jika pola karakter sama
15	Lanjut ke baris selanjutnya
16	Isi isHeaderMainScenario jadi false
17	Lewatkan baris berikutnya
19	Isi tabel aktivitas
21	Lanjut ke baris selanjutnya
23	Kembalikan nilai result

5.2.2.3 Implementasi Kode Program *Method constructEFSM*

	EFSMBuilder.java
1	`private  void constructEFSM() {`
2	`        getBranch();`
3	
4	`        //COMPLETED(1) : read every line of activity table and construct EFSM`
5	`        for(int i = 0; i < lines.size(); i++){`
6	`            matcher = ConstantDataManager.patternNumberLetterNumber.matcher(lines.get(i));`
7	
8	`            if(matcher.find()){`
9	`                break;`
10	`            }`
11	`            setPredicate(i);`
12	
13	`            // Use the alphabet to name a state`
14	`            char stateName = (char) sequenceChar;`
15	`            EFSMState efsmState = constructState(stateName);`
16	`            sequenceChar++;`
17	
18	`            //Input set and Output set`
19	`            boolean isInput =  isHaveInput(lines.get(i));`
20	
21	`            //Transition set`
22	`  constructTransition(lines.get(i),isInput,predicate,efsmState);`
23	
24	`            //Get the previous state`
25	`            previousState = efsmState;`
26	
27	`            checkEndState(i,stateName);`
28	
29	`            checkBranch(i,isInput);`
30	`        }`
31	`        concept.attachStateMachine(finiteStateMachine);`
32	`    }`

Kode Program 5. 3 *Method constructEFSM*

Pembahasan setiap baris dari method constructEFSM akan dibahas pada Tabel 5.9.

Tabel 5. 9 Pembahasan kode program *method constructEFSM*

	Pembahasan kode program *method constructEFSM*
1	Deklarasi method void constructEFSM
2	Panggil method getBranch untuk mendapatkan node-node cabang
5	Looping sebanyak baris pada tabel aktivitas
6	Gunakan matcher untuk menyamakan pola percabangan
8	Jika pola percabangan ditemukan maka
9	Hentikan looping
11	Set predicate tiap node
14	Inisialisasi dan deklarasi stateName yaitu variabel untuk menamai state
15	Buat state dengan argument stateName
16	Sekuens abjad nama state bertambah
19	Variabel boolean untuk menandai suatu aktivitas merupakan input atau bukan
22	Buat transition berdasarkan data yang dibuat di atas
25	Isi state previous untuk melacak state sebelumnya
27	Periksa apakah sudah pada state akhir
29	Periksa apakah suatu state memiliki percabangan
31	Finite state machine selesai dibuat

5.2.2.3 Implementasi Kode Program Method addActivityName

	TestCaseBuilder.java
1	`private String addActivityName(String testCase, FSMTransitionEdge edges) {`
2	`        testCase = setPredicateActivity(testCase, edges);`
3	`        if (edges.getWrappedTransition().getActionName() != null) {`
4	`            testCase += edges.getWrappedTransition().getActionName() + "?->";`
5	`        } else {`
6	`            testCase = setBlockName(testCase, edges);`
7	`        }`
8	`        return testCase;`
9	`    }`

Kode Program 5. 4 *Method addActivityName*

Pembahasan kode program *method addActivityName* dapat dilihat pada Tabel 5.10.

Tabel 5. 10 Pembahasan kode program *method addActivityName*

	TestCaseBuilder.java
1	Deklarasi method addActivityName dengan parameter testCase dan transitionEdge

2	Panggil method setPredicate untuk menambahkan predicate pada testCase
3	Jika transition memiliki action name
4	Tambahkan action name pada test case dan berikan tanda Tanya
5	Jika transition tidak memiliki action name
6	Tambahkan testcase dengan block name dengan cara memanggil method setBlockName
8	Mengembalikan nilai testCase

5.2.2.3 Implementasi Kode Program Method generateTestCas

	TestCaseBuilder.java
1	`private void generateTestCase() {`
2	`testCaseList = new LinkedHashSet<>();`
3	`stateStack.push(stateNodesMap.get("Start_State"));`
4	`if (!stateNodesMap.isEmpty()) {`
5	`while (!stateStack.isEmpty()) {`
6	`fillTestCaseList();`
7	`}`
8	`}`
9	`}`

Kode Program 5. 5 *Method generateTestCase*

Pembahasan kode program method generateTestCase dapat dililhat pada Tabel 5.11.

Tabel 5. 11 Pembahasan kode program *method generateTestCase*

	Pembahasan kode program *method generateTestCase*
1	Deklarasi method generateTestCase yang bertipe void
2	Inisialisasi variabel testCaseList bertipe linkedHashSet untuk memudahkan pengisian testCase secara berurutan
3	Isi stack state dengan startState
4	Jika nodeState tidak kosong
5	Loop selama stateStack tidak kosong
6	Isi testCase dengan memanggil method fillTestCaseList

5.2.3 Implementasi Antarmuka

Hasil implementasi antarmuka dibuat berdasarkan perancangan antarmuka. Pada penlitian ini implementasi antarmuka dilakukan dengan menggunakan markup language FXML. Terdapat empat implementasi antarmuka yang terdapat pada penelitian ini. Implementasi antarmuka tersebut adalah antarmuka tampilan utama sistem, antamuka fileChooser, antarmuka loading kasus uji, dan antarmuka tampilan alert box. Implementasi tersebut dapat dilihat pada subbab 5.2.3.1 sampai 5.2.3.4.

5.2.3.1 Implementasi Antarmuka Tampilan Utama

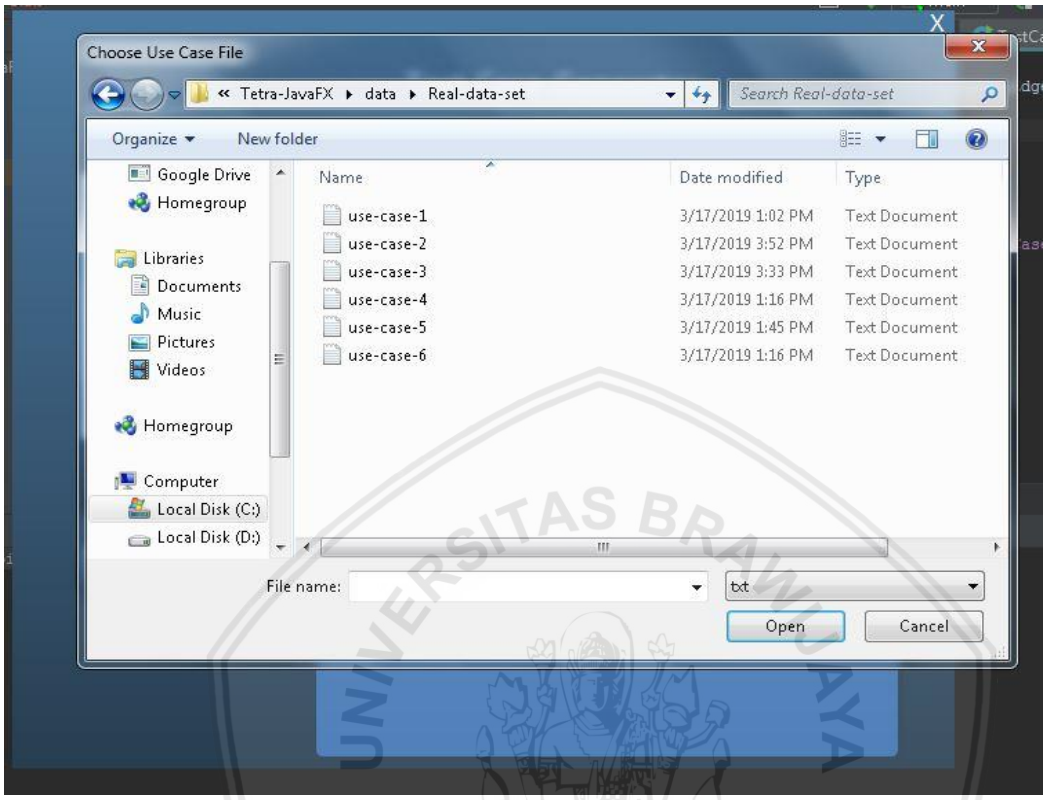
Tampilan utama adalah tampilan yang muncul saat pertama kali menjalakan kakas bantu. Segala aktivitas dari mulai memasukkan berkas skenario kasus penggunaan hingga membuka berkas kasus uji diawali dari tampilan utama. Terdapat tiga tombol pada tampilan utama, namun jika kasus uji belum digenerasi maka hanya dua tombol yang dapat ditekan. Tombol ketiga dapat ditekan setelah kasus uji selesai digenerasi. Selain itu juga terdapat daftar nama berkas yang dimasukkan ke kakas bantu untuk memudahkan pengguna melacak berkas mana saja yang telah dimasukkan. Tombol *open file* hanya akan berfunsi ketika berkas kasus uji telah selesai digenerasi.

Gambar 5. 10 Implementasi Antarmuka Tampilan Utama

5.2.3.2 Implementasi Antarmuka *FileChooser*

Antarmuka *FileChooser* adalah antarmuka yang akan muncul ketika pengguna menekan tombol *choose file.* FileChooser akan menampilkan direktori pada computer. Pengguna dapat memilih berkas kasus penggunaan yang ada pada direktori computer pengguna. Jika pengguna menekan cancel pada FileChooser maka berkas tidak akan masuk ke dalam sistem. Jika pengguna

menekan Open maka berkas akan masuk ke dalam sistem. Pengguna hanya dapat memilih berkas dengan ektensi ".txt".



Gambar 5. 11 Implementasi Antarmuka *FileChooser*

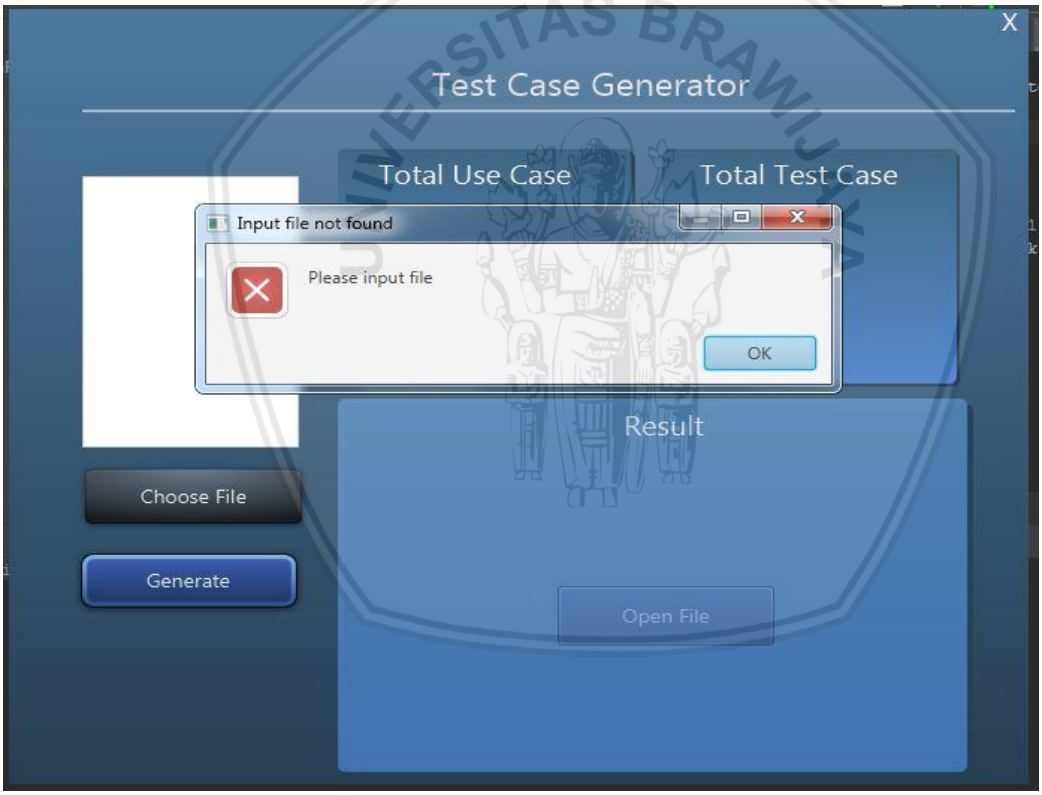
5.2.3.3 Implementasi Antarmuka Loading Kasus Uji

Antarmuka *loading* kasus uji adalah antarmuka yang menampilkan *progress bar* dan *stack pane*. *Loading* kasus uji akan muncul ketika menekan tombol generate dan berkas yan masuk ke kakas bantu tidak kosong. Antarmuka ini mencegah pengguna menekan tombol apapun selama proses generasi kasus uji. Antarmuka ini akan hilang setelah generasi kasus uji selesai. Gambar 5.10 adalah hasil implementasi antarmuka *loading* kasus uji.

Gambar 5. 12 Implementasi Antarmuka *Loading* Kasus Uji

5.2.3.4 Implementasi Antarmuka Tampilan Alert Box

Tampilan alert box adalah tampilan yang menandakan bahwa tidak ada berkas skenario penggunaan yang dijadikan masukkan kakas bantu. Jika pengguna menekan tombol *generate* namun belum memasukkan berkas penggunaan sama sekali, maka kakas bantu akan menampilkan *alert box*. *Alert box* dapat dihilangkan dengan menekan tombol OK pada *box*. Implementasi alert box dapat dilihat pada gambar 5.11.

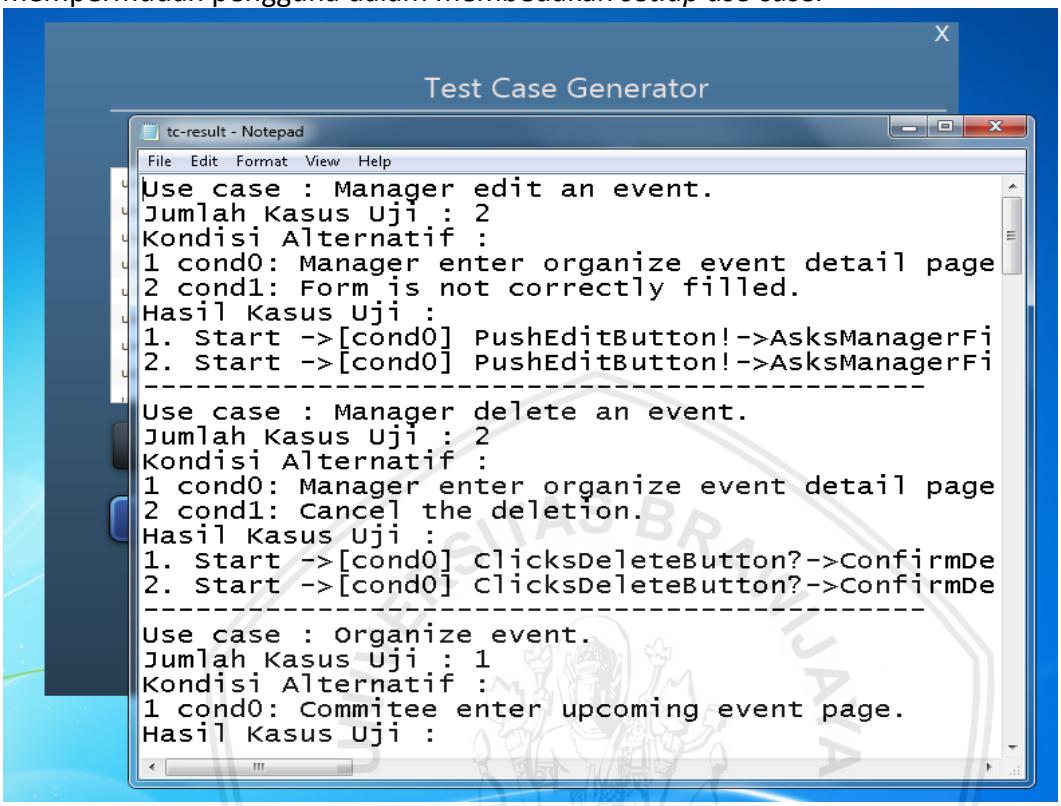


Gambar 5. 13 Implementasi Antarmuka Tampilan *Alert Box*

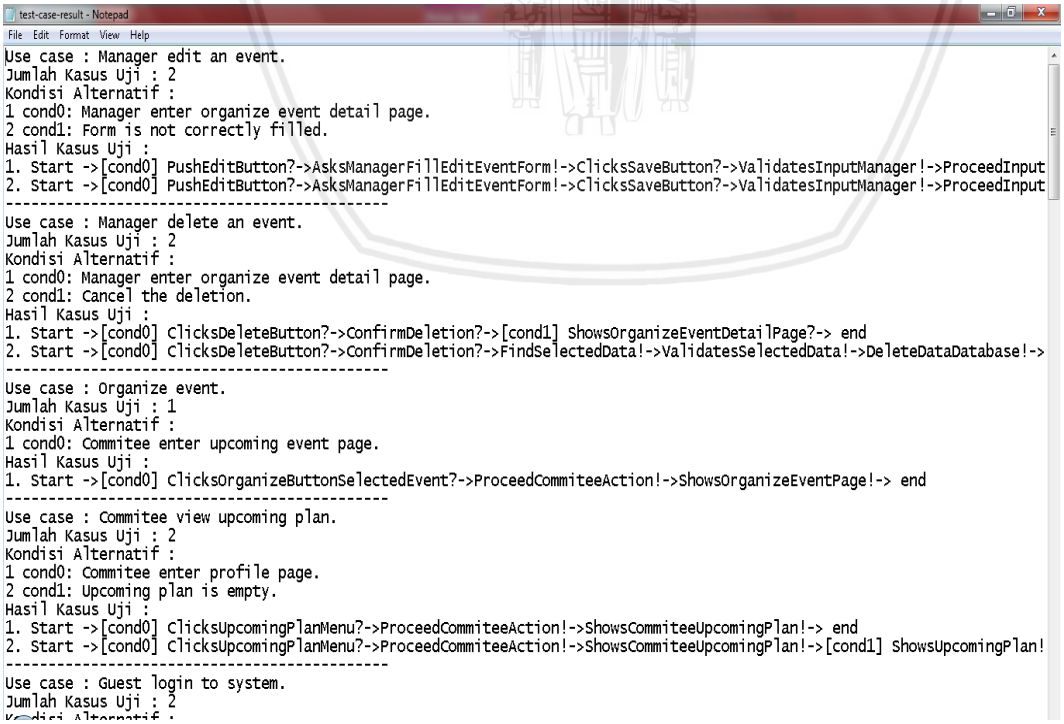
5.2.3.5 Implementasi Antarmuka Membuka Kasus Uji

Implementasi antarmuka membuka kasus uji disesuakan dengan perancangan antarmuka membuka kasus uji. Terdapat *main layout* sebagai tampilan utama dan *file* kasus uji yang merupakan hasil generasi kasus uji. File kasus uji terlihat dalam bentuk berkas. Berkas tersebut akan berisi nama *use*

case, jumlah kasus uji, kondisi alternatif, dan hasil kasus uji. Setiap *use case* akan dipisahkan oleh symbol garis putus-putus, hal ini digunakan untuk mempermudah pengguna dalam membedakan *setiap use case.*



Gambar 5. 14 Implementasi Antarmuka Membuka Kasus Uji



Gambar 5. 15 Hasil Kasus Uji Studi Kasus

BAB 6 PENGUJIAN

6.1 Pengujian Unit

Pengujian unit pada pembangunan kakas bantu pembangkitan kasus uji *black-box* akan menguji beberapa komponen dari kakas bantu pembangkitan kasus uji *black-box* yang dikembangkan. Pengujian unit pada pembangunan kakas bantu pembangkitan kasus uji *black-box* akan menggunakan metode *whitebox* testing dengan teknik pengujian *basis path testing*. Pada pengujian unit pembangunan kakas bantu pembangkitan kasus uji *black-box* akan diuji lima algoritme yang ada pada kakas bantu pembangkitan kasus uji *black-box*. Lima algorime tersebut adalah algoritme *getVBZ,* algoritme *constructActivityTable,* algoritme *constructEFSM,* algoritme *addActivityName,* dan algoritme *generateTestCase*.

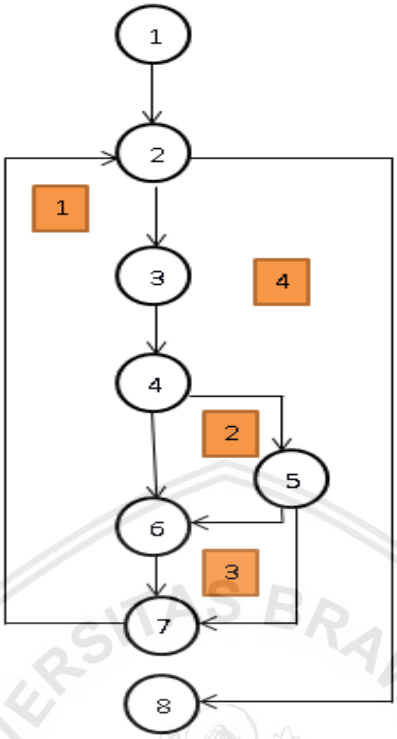
6.1.1 Pengujian Unit Method *getVBZ*

1. Pseudocodes

```
1  FUNCTION getVBZ(line):
2     INIT result TO line SPLIT BY " "      } 1
3     INIT verb TO StringBuilder OBJECT     }
4     FOR i = 2 TO i < LENGTH OF result WITH i+=1 2
5        INIT firstChar TO result ON i WITH SUBSTRING FROM 0 TO 1
   WITH UPPERCASE 3
6        IF result ON i CONTAINS "VB" 4 OR result ON i CONTAINS "NN"
   5
7           APPEND verb WITH firstChar,result ON i WITH SUBSTRING
   FROM 0 TO 1 WITH SPLIT BY "/" ON 0 6
8        ENDIF 7
9     ENDFOR                          } 8
10    RETURN TOSTRING OF verb         }
```

Pseudocode 6. 1 Algoritme method getVBZ

2. Basis Path Testing
   Berikut adalah flowgraph dari method getVBZ berdasarkan pemetaan pseudocodes di atas.
3. *Cyclomatic Complexity*
   - V(G) = Jumlah *region* = 4
   - V(G) = Jumlah *edge* – Jumlah *node* + 2 = 10 – 8 + 2 = 4
   - V(G) = Jumlah *predicate node* + 1 = 3 + 1 = 4
4. Independent Path
   - Jalur 1 : 1 - 2 - 8
   - Jalur 2 : 1- 2 - 3 -4 – 5 -7 -2 -8
   - Jalur 3 : 1 - 2 – 3 – 4 – 6 – 7 – 2 -8
   - Jalur 4 : 1 - 2 – 3 – 4 - 5 – 6 – 7 – 2 -8



Gambar 6. 1 *Flow Graph method getVBZ*

Jalur independent yang didapat akan dijadikan dasar pembuatan kasus uji. Kasus uji dan hasil dari pengujian unit *method getVBZ* ditunjukkan pada tabel 6.1.

Tabel 6. 1 Hasil pengujian unit *method getVBZ*

No	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapat	Status
1	line = “”	Method getVBZ dijalankan dan mengembalikan nilai “”	Method getVBZ dijalankan dan mengembalikan nilai “”	Valid
2	line = “3/CD 3/CD 3/CD 3/CD 3/CD./.”	Method getVBZ dijalankan dan mengembalikan nilai “”	Method getVBZ dijalankan dan mengembalikan nilai “”	Valid
3	line = “3/CD System/NNP proceed/VB ./.”	Method getVBZ dijalankan dan mengembalikan nilai “Proceed”	Method getVBZ dijalankan dan mengembalikan nilai “Proceed”	Valid
4	line =”3/CD System/NNP proceedVBSystem/NNP ./.”	Method getVBZ dijalankan dan mengembalikan nilai “ProceedVBSystem	Method getVBZ dijalankan dan mengembalikan nilai “ProceedVBSyste	Valid

		”	m”	

6.1.2 Pengujian Unit Method *constructActivityTable*

1. Pseudocodes

```
    Pseudocode algoritme constructActivityTable
 1  FUNCTION constructActivityTable(File : file)
 2     INIT path TO file path
 3     INIT lines TO read All Lines OF Files WITH path      } 1
 4     INIT matcher TO null
 5
 6     WHILE linesScanned < size OF lines  2
 7        CALL checkMainScenario WITH lines
 8        CALL checkExtensions WITH lines                   } 3
 9        CALL checkVariations WITH lines
10        SET matcher TO matcher OF Pattern with linesScanned
11     IF CALL skipSectionHeder WITH matcher == TRUE  4
12        linesScanned = linesScanned + 1
13        isMainScenario = FALSE                            } 5
14        CONTINUE
15     ENDIF
16
17     CALL fillResult WITH lines                           } 6
18     linesScanned = linesScanned + 1
19     ENDWHILE 7
```

Pseudocode 6. 2 Algoritme *method constructActivityTable*

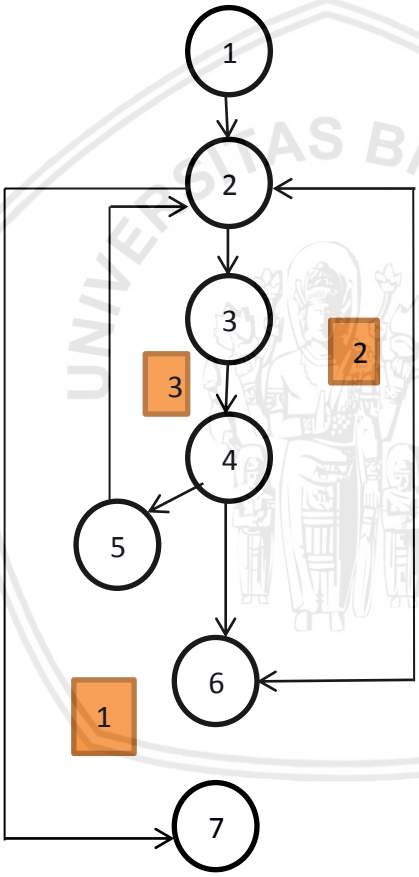
2. Basis Path Testing
   Berikut adalah flowgraph dari method *constructActivityTable* berdasarkan pemetaan pseudocodes di atas.
3. *Cyclomatic Complexity*
   - V(G) = Jumlah *edge* – Jumlah *node* + 2 = 8 – 7 + 2 = 3
   - V(G) = Jumlah *predicate node* + 1 = 2 + 1 = 3
   - V(G) = Jumlah *region* = 3
4. Independent Path
   - Jalur 1 : 1 - 2 – 7
   - Jalur 2 : 1- 2 – 3 – 4 – 6 – 2 – 7
   - Jalur 2 : 1- 2 – 3 – 4 – 5 – 2 – 7

Jalur independent yang didapat akan dijadikan dasar pembuatan kasus uji. Kasus uji dan hasil dari pengujian unit method *constructActivityTable* ditunjukkan pada tabel 6.1.

Tabel 6. 2 Hasil pengujian unit *method constuctActivityTable*

No.	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapat	Status
1	*File* : *File* dengan teks kosong	Method constructActivityTable dijalankan dan tidak menghasilkan tabel aktivtas	Method constructActivityTable dijalankan dan tidak menghasilkan tabel aktivtas	Valid
2	*File*: Berkas use	Method	Method	Valid

	case scenario	constructActivityTable dijalankan dan menghasilkan tabel aktivtas	constructActivityTable dijalankan dan menghasilkan tabel aktivtas	
3	*File*: Berkas use case scenario	Method constructActivityTable dijalankan dan menghasilkan tabel aktivtas dan melewati baris dengan "main scenario"	Method constructActivityTable dijalankan dan menghasilkan tabel aktivtas dan melewati baris dengan "main scenario"	Valid



Gambar 6. 2 *Flow Graph method constructActivityTable*

6.1.3 Pengujian Unit Method *constructEFSM*

1. Pseudocodes

	Pseudocode algoritme constructEFSM
1	PROCEDURE constructEFSM ()
2	
3	CALL getBranch 1
4	

```
 5      FOR each line in lines 2
 6          INIT matcher TO matcher OF Pattern WITH lines 3
 7
 8          IF find OF matcher is true 4
 9              BREAK 5
10          ENDIF 6
11
12          CALL setPredicate WITH number of line
13
14          INIT stateName TO char value OF sequenceChar
15          INIT efsmState TO constructState WITH stateName
16          SET sequenceChar TO sequenceChar + 1
17
18          INIT isInput TO isHaveInput WITH lines                    7
19          CALL constructTransition WITH
    lines,isInput,getPredicate,efsmState,previousState
20
21          SET previousState TO efsmState
22
23          CALL checkEndState WITH number of line,stateName
24          CALL checkBranch WITH number of line,isInput
25      ENDFOR                                                        8
26      concept CALL attachStateMachine WITH finiteStateMachine
```

Pseudocode 6. 3 Algoritme *constructEFSM*

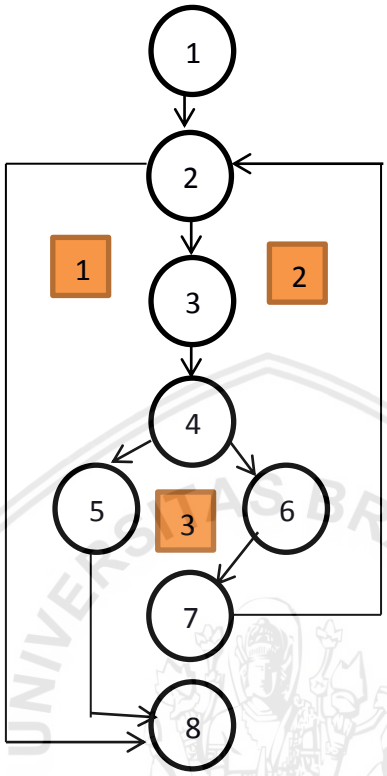
2. Basis Path Testing
   Berikut adalah flowgraph dari method *constructEFSM* berdasarkan pemetaan pseudocodes di atas.
3. *Cyclomatic Complexity*
   - V(G) = Jumlah *edge* – Jumlah *node* + 2 = 9 – 8 + 2 = 3
   - V(G) = Jumlah *predicate node* + 1 = 2 + 1 = 3
   - V(G) = Jumlah *region* = 3
4. Independent Path
   - Jalur 1 :1 – 2 – 8
   - Jalur 2 : 1 – 2 – 3 – 4 – 5 – 8
   - Jalur 3 : 1 – 2 – 3 – 4 – 6 – 7 – 2 - 8

Jalur independent yang didapat akan dijadikan dasar pembuatan kasus uji. Kasus uji dan hasil dari pengujian unit method *constructEFSM* ditunjukkan pada tabel 6.3.

Tabel 6. 3 Hasil pengujian unit *method* constructEFSM

No.	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapat	Status
1	*Input* : f*ile* teks kosong	Tidak dihasilkan model EFSM	Tidak dihasilkan model EFSM	Valid
2	*Input* : *file* use case scenario	Dihasilkan model EFSM dengan percabangan	Dihasilkan model EFSM dengan percabangan	Valid
3	*Input* : *file* use case scenario	Dihasilkan model EFSM	Dihasilkan model EFSM	Valid

		tanpa percabangan	tanpa percabangan	



Gambar 6. 3 *Flow Graph method constructEFSM*

6.1.4 Pengujian Unit Method *addActivityName*

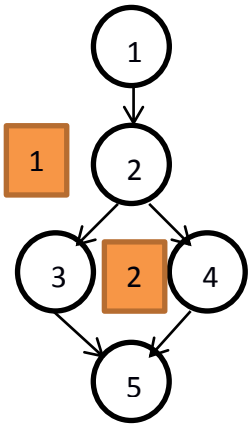
1. Pseudocodes

	Pseudocode algoritme addActivityName
1	FUNCTION addActivityName(testCase,edges):
2	Set testCase TO CALL setPredicateActivity WITH testCase,edges **1**
3	IF actionName OF transition OF edges != NULL **2**
4	testCase = testCase + actionName OF transition OF edges **3**
5	ELSE
6	testCase = CALL setBlockName WITH testCase,edges **4**
7	ENDIF **5**
8	RETURN testCase

Pseudocode 6. 4 Algoritme *method addActivityName*

2. Basis Path Testing

   Berikut adalah flowgraph dari method *addActivityName* berdasarkan pemetaan pseudocodes di atas.



Gambar 6. 4 *Flow Graph method addActivityName*

3. Cyclomatic Complexity

- V(G) = Jumlah *edge* – Jumlah *node* + 2 = 5 – 5 + 2 = 2
- V(G) = Jumlah *predicate node* + 1 = 1 + 1 = 2
- V(G) = Jumlah *region* = 2

4. Independent Path

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

Jalur independent yang didapat akan dijadikan dasar pembuatan kasus uji. Kasus uji dan hasil dari pengujian unit method *addActivityName* ditunjukkan pada tabel 6.4.

Tabel 6. 4 Hasil pengujian unit *method addActivityName*

No.	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapat	Status

1	- testCase = “”
	- edges
	  - predicateKey = null
	  - actionName = “inputAction”
	  - blockName = null
		Method getActivityName dijlankan dan mengembalikan nilai “inputAction”	Method getActivityName dijlankan dan mengembalikan nilai “inputAction”	Valid

2	- testCase = “”
	- edges
	  - predicateKey = null
	  - actionName = null
	  - blockName = “outputAction”
		Method getActivityName dijlankan dan mengembalikan nilai “predicate outputAction”	Method getActivityName dijlankan dan mengembalikan nilai “predicate outputAction”	Valid

6.1.5 Pengujian Unit Method *generateTestCase*

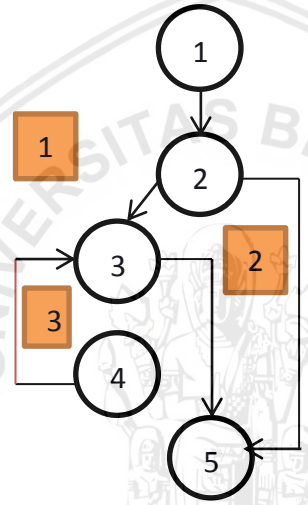
1. Pseudocodes

```
Pseudocode algoritme generateTestcase
```

```
1  PROCEDURE generateTestCase
2     SET testCaseList TO object OF linkedHashSet
3     PUSH stateStack WITH StartState FROM stateNodesMap } 1
4     IF stateNodeMap is not empty 2
5        WHILE stateStack is not empty 3
6           CALL fillTestCaseList 4
7     END WHILE } 5
8     ENDIF
```

Pseudocode 6. 5 Algoritme *method generateTestCase*

2. Basis Path Testing
   Berikut adalah flowgraph dari method *generateTestCase* berdasarkan pemetaan pseudocodes di atas.



Gambar 6. 5 *Flow Graph method generateTestCase*

3. Cyclomatic Complexity
   - V(G) = Jumlah *edge* – Jumlah *node* + 2 = 6 – 5 + 2 = 3
   - V(G) = Jumlah *predicate node* + 1 = 2 + 1 = 3
   - V(G) = Jumlah *region* = 3

4. Independent Path
   - Jalur 1 : 1 - 2 - 5
   - Jalur 2 : 1 – 2 – 3 - 5
   - Jalur 3 : 1 - 2 - 3 - 4 - 5

Jalur independent yang didapat akan dijadikan dasar pembuatan kasus uji. Kasus uji dan hasil dari pengujian unit *method generateTestCase* ditunjukkan pada tabel 6.5.

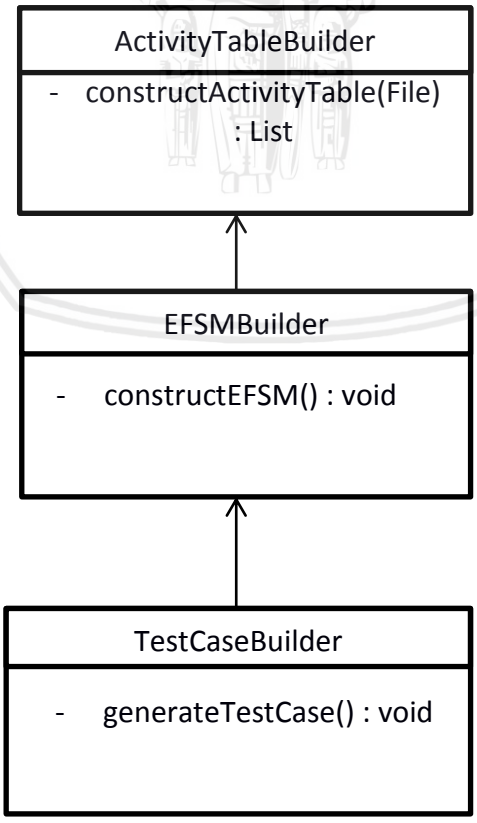
Tabel 6. 5 Hasil pengujian unit *method generateTestCase*

No.	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapat	Status
1	Size StateNodeMap = 0	Kasus uji yang	Kasus uji yang	Valid

		dihasilkan berjumlah 0	dihasilkan berjumlah 0	
2	Size StateStack = 0	Kasus uji yang dihasilkan berjumlah 0	Kasus uji yang dihasilkan berjumlah 0	Valid
3	Size StateNodeMap > 0 Size StateStack > 0	Kasus uji yang dihasilkan berjumlah lebih dari 0	Kasus uji yang dihasilkan berjumlah lebih dari 0	Valid

6.2 Pengujian Integrasi

Pengujian integrasi adalah pengujian yang bertujuan untuk mengecek interaksi antar *class* berjalan dengan baik. Pendekatan pengujian integrasi yang digunakan pada penelitian ini adalah *big-bang integration*. *Big-bang integration* dipilih karena semua *method* sudah diimplementasikan sebelumnya sehingga pengujian dilakukan secara keseluruhan. Pada pembangunan kakas bantu pembangkitan kasus uji *black-box* terdapat beberapa *method* yang hasilnya bergantung dengan *method* pada kelas lain. Method yang akan diujikan pada kelas ini adalah *method constructActivityTable*, *constructEFSM*, dan *generateTestCase*. Keterhubungan method-method tersebut akan digambarkan pada gambar 6.6.



Gambar 6. 6 Diagram hirarki pengujian Integrasi

Method constructActivityTable pada kelas *ActivityTableBuilder* akan menghasilkan tabel aktivitas yang disimpan dalam bentuk *List*. Tabel aktivitas tersebut akan digunakan oleh *EFSMBuilder* untuk membuat model *EFSM* dengan method *constructEFSM*. Model *EFSM* akan digunakan oleh *TestCaseBuilder* sebagai dasar pembuatan kasus uji. Method pada *TestCaseBuilder* yang bertanggung jawab untuk membuat kasus uji adalah *generateTestCase*. Untuk menguji ketiga komponen tersebut digunakan *basis path testing (BPT).*

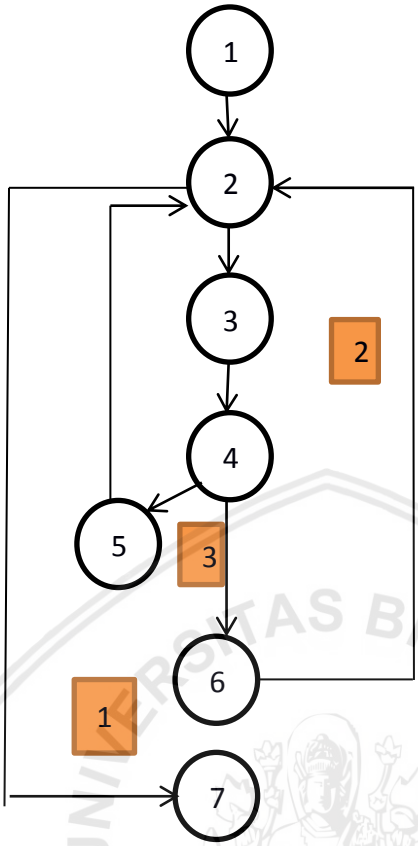
6.2.1 Pengujian Integrasi Method constructActivityTable

1. Pseudocodes

	Pseudocode algoritme constructActivityTable	
1	FUNCTION constructActivityTable(File : file)	
2	INIT path TO file path	1
3	INIT lines TO read All Lines OF Files WITH path	
4	INIT matcher TO null	
5		
6	WHILE linesScanned < size OF lines	2
7	CALL checkMainScenario WITH lines	
8	CALL checkExtensions WITH lines	3
9	CALL checkVariations WITH lines	
10	SET matcher TO matcher OF Pattern with linesScanned	
11	IF CALL skipSectionHeder WITH matcher == TRUE	4
12	linesScanned = linesScanned + 1	
13	isMainScenario = FALSE	5
14	CONTINUE	
15	ENDIF 6	
16		
17	CALL fillResult WITH lines 6	
18	linesScanned = linesScanned + 1 6	
19	ENDWHILE 7	

Pseudocode 6. 6 Algoritme *method constructActivityTable*

2. Basis Path Testing
   Berikut adalah flowgraph dari method *constructActivityTable* berdasarkan pemetaan pseudocodes di atas.
3. *Cyclomatic Complexity*
   - V(G) = Jumlah *region* = 3
   - V(G) = Jumlah *predicate node* + 1 = 2 + 1 = 3
   - V(G) = Jumlah *edge* – Jumlah *node* + 2 = 8 – 7 + 2 = 3
4. Independent Path
   - Jalur 1 : 1 - 2 – 7
   - Jalur 2 : 1- 2 – 3 – 4 – 6 – 2 – 7
   - Jalur 2 : 1- 2 – 3 – 4 – 5 – 2 – 7



Gambar 6. 7 *Flow Graph Method constructEFSM*

Jalur independent yang didapat akan dijadikan dasar pembuatan kasus uji. Kasus uji dan hasil dari pengujian unit method *isHaveInput* ditunjukkan pada tabel 6.6.

Tabel 6. 6 Hasil pengujian Integrasi *method constuctActivityTable*

No.	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapat	Status
1	*File* : *File* dengan teks kosong	Method constructActivityTable dijalankan dan tidak menghasilkan tabel aktivtas	Method constructActivityTable dijalankan dan tidak menghasilkan tabel aktivtas	Valid
2	*File*: Berkas use case scenario	Method constructActivityTable dijalankan dan menghasilkan tabel aktivtas	Method constructActivityTable dijalankan dan menghasilkan tabel aktivtas	Valid
3	*File*: Berkas use case scenario	Method constructActivityTable dijalankan dan menghasilkan tabel	Method constructActivityTable dijalankan dan menghasilkan tabel	Valid

		aktivtas dan melewati baris dengan "main scenario"	aktivtas dan melewati baris dengan "main scenario"	

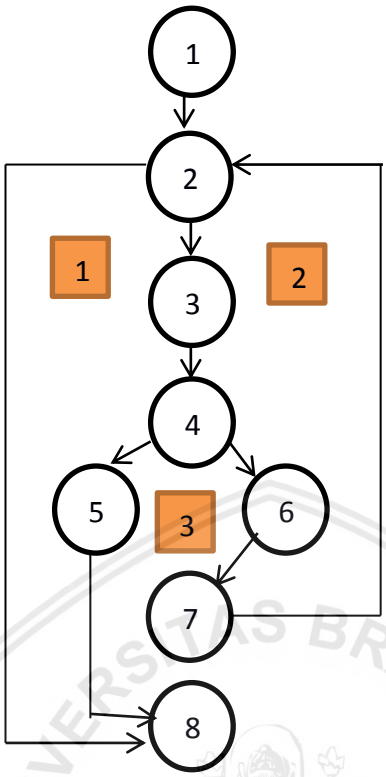
6.2.2 Pengujian Integrasi Method constructEFSM

1. Pseudocodes

```
    Pseudocode algoritme constructEFSM
 1  PROCEDURE constructEFSM ()
 2
 3      CALL getBranch 1
 4
 5      FOR each line in lines 2
 6          INIT matcher TO matcher OF Pattern WITH lines 3
 7
 8          IF find OF matcher is true 4
 9              BREAK 5
10          ENDIF 6
11
12          CALL setPredicate WITH number of line 7
13
14          INIT stateName TO char value OF sequenceChar
15          INIT efsmState TO constructState WITH stateName
16          SET sequenceChar TO sequenceChar + 1
17
18          INIT isInput TO isHaveInput WITH lines          } 7
19          CALL constructTransition WITH
    lines,isInput,getPredicate,efsmState,previousState
20
21          SET previousState TO efsmState
22
23          CALL checkEndState WITH number of line,stateName } 7
24          CALL checkBranch WITH number of line,isInput
25      ENDFOR                                               } 8
26      concept CALL attachStateMachine WITH finiteStateMachine
```

Pseudocode 6. 7 Algoritme constructEFSM

2. Basis Path Testing
   Berikut adalah flowgraph dari method *constructEFSM* berdasarkan pemetaan pseudocodes di atas.
3. *Cyclomatic Complexity*
   - V(G) = Jumlah *region* = 3
   - V(G) = Jumlah *predicate node* + 1 = 2 + 1 = 3
   - V(G) = Jumlah *edge* – Jumlah *node* + 2 = 9 – 8 + 2 = 3
4. Independent Path
   - Jalur 1 :1 – 2 – 8
   - Jalur 2 : 1 – 2 – 3 – 4 – 5 – 8
   - Jalur 3 : 1 – 2 – 3 – 4 – 6 – 7 – 2 – 8
   -



-
-

Gambar 6. 8 Flow Graph Method constructEFSM

Jalur independent yang didapat akan dijadikan dasar pembuatan kasus uji. Kasus uji dan hasil dari pengujian unit method *constructEFSM* ditunjukkan pada tabel 6.7.

Tabel 6. 7 Hasil pengujian Integrasi *method constructEFSM*

No.	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapat	Status
1	*Input* : f*ile* teks kosong	Tidak dihasilkan model EFSM	Tidak dihasilkan model EFSM	Valid
2	*Input* : *file* use case scenario	Dihasilkan model EFSM dengan percabangan	Dihasilkan model EFSM dengan percabangan	Valid
3	*Input* : *file* use case scenario	Dihasilkan model EFSM tanpa percabangan	Dihasilkan model EFSM tanpa percabangan	Valid

6.2.3 Pengujian Integrasi Method generateTestCase

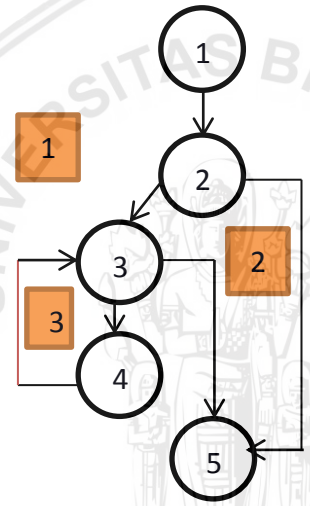
1. Pseudocodes

	Pseudocode algoritme generateTestcase
1	PROCEDURE generateTestCase
2	SET testCaseList TO object OF linkedHashSet
3	PUSH stateStack WITH StartState FROM stateNodesMap } 1
4	IF stateNodeMap is not empty 2
5	WHILE stateStack is not empty 3
6	CALL fillTestCaseList 4
7	END WHILE } 5
8	ENDIF

Pseudocode 6. 8 Algoritme *generateTestCase*

2. Basis Path Testing

Berikut adalah flowgraph dari method *generateTestCase* berdasarkan pemetaan pseudocodes di atas.



Gambar 6. 9 *Flow Graph Method generateTestCase*

3. Cyclomatic Complexity

- V(G) = Jumlah *region* = 3
- V(G) = Jumlah *predicate node* + 1 = 2 + 1 = 3
- V(G) = Jumlah *edge* – Jumlah *node* + 2 = 6 – 5 + 2 = 3

4. Independent Path

- Jalur 1 : 1 - 2 - 5
- Jalur 2 : 1 – 2 – 3 - 5
- Jalur 3 : 1 - 2 - 3 - 4 - 5

Jalur independent yang didapat akan dijadikan dasar pembuatan kasus uji. Kasus uji dan hasil dari pengujian unit *method generateTestCase* ditunjukkan pada tabel 6.8.

Tabel 6. 8 Hasil pengujian unit *method generateTestCase*

No.	Kasus Uji	Hasil yang	Hasil yang	Status

		Diharapkan	Didapat	
1	Size StateNodeMap = 0	Kasus uji yang dihasilkan berjumlah 0	Kasus uji yang dihasilkan berjumlah 0	Valid
2	Size StateStack = 0	Kasus uji yang dihasilkan berjumlah 0	Kasus uji yang dihasilkan berjumlah 0	Valid
3	Size StateNodeMap > 0 Size StateStack > 0	Kasus uji yang dihasilkan berjumlah lebih dari 0	Kasus uji yang dihasilkan berjumlah lebih dari 0	Valid

6.3 Pengujian Validasi

Pengujian validasi pada pembangunan kakas bantu pembangkitan kasus uji *black-box* mengacu pada kebutuhan yang didefinisikan pada bab rekayasa kebutuhan. Pengujian validasi akan menguji kebutuhan fungsional dan kebutuhan non-fungsional dari kakas bantu pembangkitan kasus uji *black-box*. Metode yang akan digunakan pada pengujian validasi adalah *black-box testing*. Kasus uji pada pengujian validasi diturunkan dari *event flow* pada bagian analisis, dalam penelitian ini peneliti menurunkan kasus uji dari *use case* scenario yang ada pada bab rekayasa kebutuhan. Pada pembangunan kakas bantu pembangkitan kasus uji *black-box* akan menguji tiga kebutuhan fungsional beserta alur alternatif dan satu kebutuhan non-fungsional. Pengujian validasi pada pembangunan kakas bantu pembangkitan kasus uji *black-box* dijelaskan pada sub bab 6.3.1 sampai 6.3.4.

Dalam pengujian ini skenario penggunaan yang dilakukan diambil dari penelitian berjudul *Pengembangan Sistem Pengelolaan Acara dengan Pendekatan Gamification* (Wicaksono, Priyambadha danPradana, 2019). Dari penelitian tersebut diambil 20 skenario *use case* untuk kemudian diterjemahkan ke dalam bahasa Inggris. Selain diterjemahkan, skenario penggunaan disesuaikan dengan format *Cockburn*.

6.3.1 Pengujian Validasi Memasukkan berkas skenario penggunaan

Pengujian memasukkan berkas skenario penggunaan adalah pengujian untuk menguji kebutuhan sistem dengan nomor fungsi TETRA-1-00. Pada skenario memasukkan berkas skenario penggunaan terdapat satu jalur utama dan satu jalur alternatif. Hasil pengujian jalur utama dapat dilihat pada tabel 6.9. Sedangkan tabel 6.10 adalah hasil pengujian jalur alternatif memasukkan berkas skenario penggunaan.

Tabel 6. 9 Pengujian validasi memasukkan berkas skenario penggunaan

Nama	Memasukkan berkas skenario penggunaan

Kasus Uji	
Objek uji	Kebutuhan fungsional TETRA-1-00
Tujuan pengujian	Memastikan bahwa sistem mampu memasukkan berkas skenario penggunaan dengan format .txt dan menampilkan nama berkas yang dipilih
Data masukan	Berkas skenario penggunaan berbahasa Inggris dalam format *cockburn* dan extensi .txt
Prosedur uji	1. klik tombol open file 2. pilih file mana saja yang ingin dimasukkan 3. klik tombol open
Hasil yang diharapkan	Sistem menampilkan nama-nama berkas skenario penggunaan yang dipilih.
Hasil pengujian	Sistem menampilkan nama-nama berkas skenario penggunaan yang dipilih.
Status	Valid

Tabel 6. 10 Pengujian validasi jalur alternatif memasukkan berkas skenario penggunaan

Nama Kasus Uji	Memasukkan berkas skenario penggunaan
Objek uji	Kebutuhan fungsional TETRA-1-00
Tujuan pengujian	Memastikan bahwa sistem tidak menampilkan nama berkas yang dipilih
Data masukan	
Prosedur uji	1. klik tombol open file 2. pilih file mana saja yang ingin dimasukkan 3. klik tombol cancel
Hasil yang diharapkan	Sistem tidak menampilkan nama-nama berkas skenario penggunaan yang dipilih.
Hasil pengujian	Sistem tidak menampilkan nama-nama berkas skenario penggunaan yang dipilih.
Status	Valid

6.3.2 Pengujian Validasi Membangkitkan Kasus Uji

Pengujian membangkitkan kasus uji adalah pengujian untuk menguji kebutuhan sistem dengan nomor fungsi TETRA-1-10. Pada skenario membangkitkan kasus uji terdapat satu jalur utama dan satu jalur alternatif. Kedua jalur tersebut akan diuji pada pengujian validasi sesuai dengan skenarionya masing-masing. Tabel 6.11 akan menjelaskan mengenai jalur utama dalam membangkitkan kasus uji. Sedangkan tabel 6.12 adalah tabel yang akan menjelaskan pengujian dari jalur alternatif membangkitkan kasu uji.

Tabel 6. 11 Pengujian validasi membangkitkan kasus uji

Nama Kasus Uji	Membangkitkan kasus uji
Objek uji	Kebutuhan fungsional TETRA-1-10
Tujuan pengujian	Memastikan bahwa sistem mampu mengenerasi kasus uji berdasarkan skenario penggunaan
Data masukan	Berkas skenario penggunaan berbahasa Inggris dalam format *cockburn* dan extensi .txt yang sudah dimasukkan sebelumnya
Prosedur uji	1. klik tombol generate 2. menunggu sistem mengenerasi kasus uji
Hasil yang diharapkan	Sistem berhasil mengenerasi kasus uji dan hasil kasus uji sudah tersedia.
Hasil pengujian	Sistem berhasil mengenerasi kasus uji dan hasil kasus uji sudah tersedia.
Status	Valid

Tabel 6. 12 Pengujian validasi jalur alternatif membangkitkan kasus uji

Nama Kasus Uji	Membangkitkan kasus uji
Objek uji	Kebutuhan fungsional TETRA-1-10
Tujuan pengujian	Memastikan bahwa sistem mampu menampilkan peringatan jika tidak ada berkas skenario penggunaan yang dipilih
Data masukan	
Prosedur uji	1. klik tombol generate 2. klik ok pada peringatan yang ditampilkan sistem
Hasil yang diharapkan	Sistem menampilkan peringatan untuk memasukkan berkas skenario penggunaan
Hasil pengujian	Sistem menampilkan peringatan untuk memasukkan berkas skenario penggunaan
Status	Valid

6.3.3 Pengujian Validasi Membuka Berkas Kasus Uji

Pengujian membuka kasus uji adalah pengujian untuk menguji kebutuhan sistem dengan nomor fungsi TETRA-1-20. Pada skenario membuka kasus uji terdapat satu jalur utama saja. Kasus uji yang dibuka adalah hasil generasi sistem yang sudah dilakukan sebelumnya. Tabel 6.13 akan menjelaskan mengenai jalur utama dalam membangkitkan kasus uji.

Tabel 6. 13 Pengujian validasi membuka berkas kasus uji

Nama Kasus Uji	Membuka berkas kasus uji

Objek uji	Kebutuhan fungsional TETRA-1-20
Tujuan pengujian	Memastikan bahwa sistem mampu menampilkan kasus uji yang telah digenerasi sebelumnya
Data masukan	Kasus uji dalam format teks
Prosedur uji	1. klik tombol *open file*
Hasil yang diharapkan	Sistem menampilkan hasil kasus uji yang telah digenerasi sebelumnya
Hasil pengujian	Sistem menampilkan hasil kasus uji yang telah digenerasi sebelumnya
Status	Valid

6.4 Pengujian Performa Sistem

Pada pengujian performa sistem akan dilakukan dengan mengamati waktu yang dibutuhkan sistem dalam melakukan generasi kasus uji. Data yang digunakan pada pengujian performansi adalah data 20 *use case scenario* yang digunakan pada pengujian validasi. Data tersebut diambil dari penelitian berjudul *Pengembangan Sistem Pengelolaan Acara dengan Pendekatan Gamification* (Wicaksono, Priyambadha dan Pradana, 2019). Pengujian performansi dilakukan pada linkungan sistem dengan spesifikasi *processor* AMD E1-6010, *HDD 500 GB,* dan *main memory 6.00 GB DDR3*. Sistem operasi yang digunakan adalah *Windows 7 Ultimate*. *Java Development Kit* yang digunakan adalah *JDK* 11. Pengujian performansi akan dijabarkan pada tabel 6.14.

Tabel 6. 14 Tabel Pengujian Performa Sistem

Nama Kasus Uji	Pengujian Performa Sistem
Objek uji	Kebutuhan non fungsional TETRA-2-01
Tujuan pengujian	Memastikan bahwa sistem mampu mengenerasi kasus uji kurang dari 2 menit.
Data masukan	Berkas skenario penggunaan berjumlah 20 buah skenario penggunaan.
Prosedur uji	1. klik tombol *generate* 2. amati waktu generasi yang dihasilkan pada sistem
Hasil yang diharapkan	Sistem mengenerasi kasus uji kurang dari 120 detik
Hasil pengujian	Sistem mengenerasi kasus uji selama 46 detik
Status	Valid

6.5 Pembahasan Pengujian

6.5.1 Pengujian Unit

Pengujian unit pada pembangunan kakas bantu pembangkitan kasus uji *black-box* bertujuan untuk menguji bahwa setiap data mengalir dengan benar pada tiap unit. Pada pengujian unit didapat jalur-jalur independen yang tergambar pada *flow graph* masing-masing method. Jalur-jalur independen ini dipastikan akan dijalankan minimal satu kali. Dari lima *method* yang diuji kelimanya semua pengujiannya berstatus valid. Dengan kata lain, setiap jalur independen dari kelima method yang diujikan akan dijalankan setidaknya satu kali.

6.5.2 Pengujian Integrasi

Pengujian integrasi pada pembangunan kakas bantu pembangkitan kasus uji *black-box* bertujuan menguji arsitektur dari sistem yang dikembangkan. Pengujian integrasi akan menggunakan unit-unit yang sudah dilakukan pengujian unit sebelumnya. Hal ini dilakukan karena meskipun tiap *unit* dapat bekerja dengan baik, unit-unit yang saling berhubungan dapat menghasilkan sesuatu yang tidak sesuai dengan tujuan. Pada pengujian integrasi yang menggunakan pendekatan *big-bang* diuji tiga unit yang berhubungan dan dihasilkan status valid pada semua pengujiannya. Dapat disimpulkan bahwa pada pengujian integrasi, ketiga unit yang berhubungan tersebut bekerja bersama-sama dan berinteraksi dengan tepat.

6.5.3 Pengujian Validasi

Pengujian validasi pada pembangunan kakas bantu pembangkitan kasus uji *black-box* berfokus pada aksi user dan keluaran dari sistem. aksi user dan keluaran dari sistem diturunkan dari skenario penggunan sistem (*use case scenario*). Pada pengujian validasi pembangunan kakas bantu pembangkitan kasus uji *black-box* menghasilkan status valid untuk semua skenario. Status valid tersebut didapat karena semua skenario dapat diuji dan mengahasilkan hasil uji yang sesuai dengan yang diharapkan.

6.5.4 Pengujian Performa

Pengujian performansi bertujuan menguji *run-time* dari suatu perangkat lunak. pengujian perfomansi melibatkan komponen *hardware* dan *software*. Pada penelitian ini pengujian perfomansi bertujuan mengukur kecepatan dari generasi kasus uji. Hasil dari pengujian ini dapat memenuhi manfaat dari penelitian yaitu untuk mempersingkat waktu yang dibutuhkan dalam menguji sistem. penelitian ini menggunakan batas waktu 2 menit karena ingin sistem bekerja 10 kali lebih cepat dibanding manusia. Dalam hal ini, manusia membutuhkan waktu sekitar 20 menit untuk mengenerasi kasus uji dari 20 *use case scenario*.

BAB 7 PENUTUP

7.1 Kesimpulan

Dari analisis kebutuhan, perancangan, implementasi, dan pengujian yang telah dilakukan, maka dapat ditarik kesimpulan bahwa:

1. Hasil dari rekayasa kebutuhan kakas bantu pembangkitan kasus uji *black-box*, diperoleh tiga kebutuhan fungsional sistem dan satu kebutuhan non fungsional sistem yang didapat dari analisis penelitian yang berjudul *Automation of Test Case Generation From Textual Use Cases* (Jiang & Ding, 2011). Tiga kebutuhan fungsional sistem adalah memasukkan berkas skenario penggunaan, mengenerasi kasus uji, dan membuka berkas kasus uji. Tiga kebutuhan fungsional sistem didapat untuk memenuhi kebutuhan penelitian, yaitu membangun kakas bantu pembangkitan kasus uji. Satu kebutuhan non fungsional sistem adalah kebutuhan performansi, yaitu sistem harus mampu mengenerasi kasus uji dalam waktu kurang dari 2 menit. Kebutuhan non-fungsional didapat untuk memenuhi manfaat penelitian, yaitu mempersingkat waktu yang dibutuhkan untuk pengujian. Kebutuhan fungsional pada pembangunan kakas bantu pembangkitan kasus uji *black-box* dimodelkan dengan pemodelan berorientasi objek, yaitu *use case diagram* dan *use case scenario.*
2. Perancangan pada pembangunan kakas bantu pembangkitan kasus uji *black-box* dilakukan dengan pendekatan berorientasi objek. Pada perancangan objek terdapat perancangan kelas dan interaksi antar objek. Perancangan sistem terdiri dari perancangan arsitektur berupa rancangan *class diagram* dan *sequence diagram*. Selain perancangan arsitektur, terdapat perancangan komponen yang berisi perancangan algoritme. Dalam penelitian diambil lima sampel algoritme yang kemudian akan diimplementasikan. Kelima algoritme tersebut adalah algoritme getVBZ, contsructActivityTable, constructEFSM, addActivityName, dan generateTestCase. Perancangan yang terakhir adalah perancangan antarmuka. Dihasilkan lima perancangan antarmuka yang akan diimplementasikan dalam pengembangan kakas bantu.
3. Implementasi kakas bantu pembangkitan kasus uji *black-box* dilakukan dengan mengacu pada tahap perancangan. Implementasi dilakukan dengan menggunakan bahasa pemrograman *JAVA.* Terdapat beberapa library yang digunakan dalam implementasi, yaitu library *StandfordNLP* dan *JavaFX*. *StandfordNLP* digunakan untuk melakukan *POS Tagging* dalam proses membangun tabel aktivitas. *JavaFX* digunakan untuk melakukan implementasi antarmuka. Hasil implementasi lima algoritme pada tahap perancangan dapat dilihat pada implementasi kode program. Untuk perancangan antarmuka diimplementasikan pada implementasi antarmuka.
4. Pengujian pada pembangunan kakas bantu pembangkitan kasus uji *black-box* yang dilakukan berupa pengujian integrasi, pengujian unit, pengujian validasi, dan pengujian performa. Pengujian unit pada pembangunan kakas bantu pembangkitan kasus uji *black-box* dilakukan dengan mengambil lima sampel

algoritme pada perancangan algoritme. Pengujian integrasi menguji tiga kelas yang terintegrasi. W*hite box testing* pada pembangunan kakas bantu pembangkitan kasus uji *black-box* digunakan untuk pengujian unit dan integrasi, dalam hal ini digunakan *basis path testing*. Pengujian validasi pada pembangunan kakas bantu pembangkitan kasus uji *black-box* menggunakan sebagai acuan adalah *use case scenario* dari analisis kebutuhan yang dilakukan pada rekayasa kebutuhan. Pengujian performansi pada pembangunan kakas bantu pembangkitan kasus uji *black-box* dilakukan untuk memenuhi kebutuhan non fungsional sistem. pengujian validasi dan pengujian performansi dilakukan dengan menggunakan *black box testing*.

Dari pengujian unit, didapat bahwa setiap jalur independen dari kelima method yang diujikan akan dijalankan setidaknya satu kali. Pada pengujian integrasi, ketiga unit yang diuji bekerja bersama-sama dan berinteraksi dengan tepat. Sedangkan pada pengujian validasi, didapat bahwa semua skenario dapat diuji dan mengahasilkan hasil uji yang sesuai dengan yang diharapkan. Hasil dari pengujian performansi dapat memenuhi manfaat dari penelitian yaitu untuk mempersingkat waktu yang dibutuhkan dalam menguji sistem.

7.2 Saran

Saran yang didapat untuk pengembangan lanjut dari kakas bantu pembangkitan kasus uji *black-box* adalah sebagai berikut:

1. Kakas bantu pembangkitan kasus uji dapat dibangun dengan platform web agar dapat meningkatkan aksesibilitas.
2. Performa sistem dapat ditingkatkan dengan mengoptimalkan algoritme penelusuran model *Extended Finite State Machin*e.
3. Antarmuka sistem dapat lebih interaktif dan terpisah menjadi beberapa *layout*.
4. Melakukan penambahan format skenario kasus penggunaan menjadi beberapa format, tidak terbatas pada format *Cockburn.*

DAFTAR REFERENSI

Ammann, P., dan Offutt, J., 2016. *Introduction to Software Testing*. Cambridge: Cambridge University Press

Baesens, Bart, Aimee Backiel, and Seppe vanden Broucke. 2015. *Beginning Java® Programming: The Object-Oriented Approach*. [e-book] Crosspoint Boulevard Indianapolis, IN: John Wiley & Sons, Inc. Tersedia melalui: google books < https://books.google.co.id/books?id=A5qLBgAAQBAJ>

Cockburn, A.2001, *Writing Effective Use-Cases* , Addison-Wesley.

Elallaoui, M., Nafil, K., Touahni, R., 2018. Automatic Transformation of User Stories into UML Use Case Diagrams using NLP Techniques. *Procedia Computer Science*, [e-journal] 130, 42-49. Tersedia melalui: Science Direct <https://www.sciencedirect.com/science/article/pii/S1877050918303600> [Diakses 1 September 2018]

Fauzan, R., 2014. Rekomendasi Kasus Penggunaan Berdasarkan Skenario Naratif Menggunakan Teknologi Semantik dan Repositori S2. Institut Teknologi Sepuluh November.

Gail, G. dan Paul, P., 2009. *Essential JavaFX*. 1st ed. New Jersey: Prentice Hall.

Garg, D. and Kaur, N., 2012. Analysis Of The Depth First Search Algorithms.

Hariyanto, R., 2004, *Sistem Manajemen Basis Data*. Bandung: Informatika.

Hu, Y., Lin, N, 2010. *Automatic Black Box Method-Level Test Case Generation Based on Constraint Logic Programming,* [e-journal] 977 - 982. Tersedia melalui: IEEE <https://ieeexplore.ieee.org/document/5685369 > [diakses 21 Oktober 2018]

Hue, C.T., Hanh, D.D., Binh, N.G., 2018. A Transformation-Based Method for Test Case Automatic Generation from Use Cases, [e-journal] 102 – 107. Tersedia melalui : IEEE <https://ieeexplore.ieee.org/document/8573372> [diakses 7 Februari 2019]

Jiang, M., Ding, Zuohua., 2011, *Automation of Test Case Generation From Textual Use Case*, [e-journal] 102 – 107. Tersedia melalui : IEEE <https://ieeexplore.ieee.org/document/6014540> [diakses 21 Oktober 2018]

Kalaji, A.S., Hierons, R.M. and Swift, S., 2009. A search-based approach for automatic test generation from Extended Finite State Machine (EFSM). *TAIC PART 2009 - Testing: Academic and Industrial Conference - Practice and Research Techniques*, pp.131–132. Tersedia melalui : < https://www.researchgate.net/publication/49402566_A_Search-Based_Approach_for_Automatic_Test_Generation_from_Extended_Finite_State_Machine_EFSM >[26 Februari 2018]

Kaner, C., 2003. What Is a Good Test Case? *Software Testing Analysis & Review Conference (STAR East)*, pp.1–16.

Leach, Ronald J., 2016 Introduction to Software Engineering. 2nd ed. Boca Raton, Florida: CRC Press

Meliana et al., 2017. *Automated Test Case Generation from UML Activity Diagram and Sequence Diagram using Depth First Search Algorithm*, [e-journal]116, 629-637. Tersedia melalui: IEEE <https://www.sciencedirect.com/science/article/pii/S1877050917320732> [diakses 21 Oktober]

Mishra, A. and Dubey, D., 2017. *A Comparative Study of Different Software Development Life Cycle Models in Different Skenarios*.[e-journal]1(5), pp.64-59. Tersedia melalui : <https://www.researchgate.net/publication/289526047_A_Comparative_Study_of_Different_Software_Development_Life_Cycle_Models_in_Different_Skenarios > [Diakses 26 Februari 2018]

Myers, G., 2004. *The Art of Software Testing,* 2nd ed. New Jersey: John Wiley & Sons. Tersedia melalui : <http://barbie.uta.edu/~mehra/Book1_The%20Art%20of%20Software%20Testing.pdf> [Diakses 24 Februari 2019]

Petrenko, A., Boroday, S. and Groz, R., 2004. Confirming configurations in EFSM testing. *IEEE Transactions on Software Engineering*, 30(1), pp.29–42.

Pressman, S.R. 2010. *Software Engineering: A Practitioner's Approach*, 7th ed. New York: Mc-Graw Hill Education

Rickyanto, I., 2003. *Dasar pemrograman Berorientasi Objek Dengan Java 2 (JDK 1.4)*. Yogyakarta:Andi.

Shanthi, V.K.A., Mohankumar, G., 2012. *A Novel Approach for Automated Test Path Generation using TABU Search Algorithm*, [e-journal] 48, 28–34. Tersedia melalui : <https://www.researchgate.net/publication/267383243_A_Novel_Approach_for_Automated_Test_Path_Generation_using_TABU_Search_Algorithm> [diakses 22 Oktober 2018]

Sommerville, I., 2011. *Software engineering*. 9th ed. London: Addison-Wesley.

Stanford NLP Group. 2013. Stanford NLP, <URL: http://nlp.stanford.edu>

Sukamto, Shalahuddin. 2013. *Analisis dan Desain Sistem Informasi.* Yogyakarta: Andi Offset.

Supriyatno. 2010. Pemrograman *Database* Menggunakan *Java & MySQL* Untuk Pemula. Jakarta: Mediakita.

Suyanto. 2014. *Artificial Intelligence Searching Reasoning, Planning, dan Learning.* Bandung: Informatika Bandung.

TIOBE SOFTWARE BZ. 2019. TIOBE Index for April 2019 [Online]. Tersedia melalui: https://www.tiobe.com > [diakses pada 17 April 2019] .

Wang, C et al., 2015, Automatic Generation of System Test Cases from Use Case Specifications, [e-journal] 102 – 107. Tersedia melalui : ACM <https://dl.acm.org/citation.cfm?id=2771812> [diakses 7 Februari 2019]

Wicaksono, E., Priyambadha, B. and Pradana, F., 2019. Pengembangan Sistem Pengelolaan Acara dengan Pendekatan Gamification, [e-journal]3, 3. Tersedia melalui : <http://j-ptiik.ub.ac.id/index.php/j-ptiik/article/view/4628> [diakses pada 17 April 2019]