Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 6, No. 8, Agustus 2022, hlm. 3733-3740 http://j-ptiik.ub.ac.id

Pengembangan Sistem Otomatisasi *Use Case Diagram* berdasarkan Skenario Sistem menggunakan Metode POS Tagger Stanford NLP

Muhammad Aji Taufan[1], Denny Sagita Rusdianto[2], Mahardeka Tri Ananta[3]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]ajitaufan@student.ub.ac.id, [2]denny.sagita@ub.ac.id, [2]deka@ub.ac.id

Abstrak

Use case diagram merupakan hal fundamental dalam melakukan pemodelan perangkat lunak. Hal tersebut dikarenakan fungsinya menyatakan interaksi yang terjadi antara pengguna dengan sistem secara visual dan mempresentasikan semua fungsi yang akan dilakukan sistem. Namun untuk memperoleh use case diagram yang baik diperlukan hasil penulisan kebutuhan baik serta analisis elisitasi yang mumpuni sehingga memakan waktu berlebih. Salah satu cara memfasilitasi hal ini yaitu dengan melakukan otomatisasi pembuatan use case diagram. Otomatisasi sistem pada penelitian bersifat sebagai rekomendasi pemodelan. Use case diagram dapat digenerasi dengan masukan skenario sistem karena mewakili elemen yang ada. Kemudian untuk menerjemahkan skenario ke use case diagram menggunakan pemrosesan bahasa alami stanfordNLP dan *library* plantUML untuk pembuatan diagram otomatisasinya. Penelitian dilakukan dengan menganalisis kebutuhan dan mendapatkan 7 kebutuhan fungsional. Model pengembangan sistem yang menggunakan *waterfall* dengan bahasa pemrograman Java karena terintegrasi dengan *library* stanfordNLP dan plantUML. Sistem dikembangkan berbasis *desktop* dan dapat dijalankan pada platform yang mendukung JRE *(Java Runtime Environment)*. Pada pengujian sistem melakukan pengujian unit, integrasi, validasi dan akurasi, dengan hasil prosedur uji pengujian unit 100% valid dari 3 sampel uji dan pengujian validasi fungsional 100% valid dari semua kasus uji dan pengujian akurasi pada berkas skenario sistem sebesar 47% hingga 64%.

Kata kunci: *aplikasi desktop, otomatisasi, pemodelan, use case diagram, Java, pemrosesan bahasa alami*

Abstract

Use case diagrams are a fundamental thing in software modeling. This is because of its function states the interaction that occurs between user and system visually and presents all functions that will be carried out by the system briefly, concisely and clearly. However, to get a good use case diagram, good results of writing requirements and qualified elicitation analysis are needed so that it takes up too much time. One way to facilitate this is by automating the creation of use case diagrams. The system automation in this study is a modeling recommendation. Use case diagrams can be generated with system scenarios as input because they represent existing elements. Then to translate the scenario into use case diagrams are using natural language processing stanfordNLP and the plantUML library for making automation diagrams. The research was conducted by analyzing requirements and obtaining 7 functional requirements. The system development model that will be developed uses the waterfall with Java as its programming language because it could be integrated with the stanfordNLP and plantUML libraries. The system will be developed on a desktop and can be run on a platform that supports JRE (Java Runtime Environment). The next stage is unit, integration and validation and accuracy testing, with the results of the unit test procedure being 100% valid from 3 samples, validation testing 100% valid from all test cases and results of accuracy testing on usage scenarios between 47% to 64%.

Keywords: *desktop applications, automation, modeling, use case diagrams, Java, natural language processing*

1. PENDAHULUAN

Salah satu cara untuk memodelkan kebutuhan pemangku kepentingan pada pemrograman berorientasi objek (OOP) adalah dengan menggunakan use case diagram. Use case diagram menggambarkan bagaimana user berinteraksi dengan sistem dengan cara

mendefinisikan langkah-langkah yang dibutuhkan untuk menyelesaikan tujuan tertentu (Pressman, 2010). Sebuah use case diagram menyatakan visualisasi interaksi yang terjadi antara pengguna dengan sistem (Kurniawan, 2018). Secara sederhana, use case diagram digunakan untuk memahami fungsi apa saja yang ada di dalam sistem dan siapa saja yang dapat mengetahui fungsi tersebut. Langkah-langkah sebelum melakukan perancangan atau pemodelan dalam rekayasa perangkat lunak, adalah dengan melakukan rekayasa kebutuhan. Kebutuhan sendiri adalah suatu kondisi yang dibutuhkan oleh user untuk mengatasi masalah. Sedangkan rekayasa kebutuhan adalah proses mewujudkan serangkaian layanan yang dibutuhkan oleh pemangku kepentingan atas suatu sistem (Sommerville, 2011).

Setelah selesai melakukan proses rekayasa kebutuhan, didapatkan hasil akhir yaitu penulisan kebutuhan. Penulisan kebutuhan yang baik haruslah mencakup kebutuhan dengan kategori yang memenuhi standar penulisan, yaitu harus specific, measureable (terukur), attainable (dapat dicapai), realizable (dapat direalisasikan), dan traceable (dapat dilacak), disebut sebagai kebutuhan yang SMART (Pressman, 2010). Banyak cara dalam melakukan penulisan kebutuhan membuat pengerjaan penulisan lebih mudah dikerjakan. Namun banyaknya cara juga membuat penulisan menjadi tidak sinkron antar pengembang karena masing-masing menggunakan penulisan yang berbeda (Madanayake, et al., 2017).

Salah satu metode elisitiasi yang baik dan mencakup pikiran semua pemangku kepentingan dan pengembang adalah menggunakan skenario. Pada skenario sistem terdapat penulisan kebutuhan yang rinci dan mendetail (Elallaoui, et al., 2018). Selain membantu pemangku kepentingan, skenario juga membantu analis, karena semua orang yang terlibat dapat berdiskusi, bernegosiasi dan merencanakan menggunakan artefak/model yang sama. Setelah didapatkan kebutuhan, proses selanjutnya dalam pengembangan menggunakan kaidah OOP yaitu melakukan pemodelan. Pada pemodelan kebutuhan, digunakan bahasa UML yakni use case diagram dan use case scenario guna menerjemahkan hasil dari penulisan kebutuhan.

Masalah lain muncul pada tahap pemodelan kebutuhan. Pada jurnal penelitian oleh (Madanayake, et al., 2017), peneliti melakukan survei kepada responden dari industri IT yang berada di Sri Lanka, Singapura dan Australia. Hasilnya yaitu sebanyak 54,5% responden memilih waktu untuk menggambar menjadi alasan pengembang untuk tidak melakukan pemodelan. Padahal penulisan kebutuhan dan pemodelan kebutuhan merupakan hal yang saling terkait dan hal yang krusial dalam siklus hidup pengembangan perangkat lunak.

Dengan permasalahan yang telah dijelaskan diatas, diperlukan solusi yang dapat mengurangi beban kerja pengembang yaitu sebuah sistem yang dapat melakukan otomatisasi use case diagram dari skenario sistem. Skenario sistem dipilih sebagai masukan karena elemen-elemennya mewakili elemen use case diagram. Ide dari penelitian ini adalah menggunakan skenario sistem sebagai masukan kemudian kalimat dalam skenario diproses dan diolah oleh metode pencarian bahasa.

Pengolahan masukan skenario sistem menggunakan salah satu alat dalam pemrosesan bahasa alami yaitu Stanford NLP, sebuah sistem yang dikembangkan oleh Stanford NLP *Corporation* dengan fokus pada penelitian pada *sentence understanding, probabilistic parsing and tagging, biomedical information extraction, grammar introduction, word sense disambiguation, dan automatic question answering* (Stanford NLP Group, 2013). Pengolahan bahasa yang digunakan akan difokuskan untuk melakukan *tagging*. *Tagging* adalah suatu metode untuk memberi tanda pada kata dalam kalimat, untuk menentukan label kata yaitu dengan menggunakan Part of Speech (POS) dari model Pen Tree Bank.

Kemudian setelah didapatkan kata dan tag masing-masing maka proses selanjutnya menerjemahkan kedalam diagram. Untuk proses ini menggunakan *library* PlantUML. PlantUML adalah *open-source tool* yang dapat membuat diagram UML dan diagram non-UML secara otomatis. Diagram didefinisikan melalui bahasa teks yang mendasar dan secara intuitif (PlantUML, 2019).

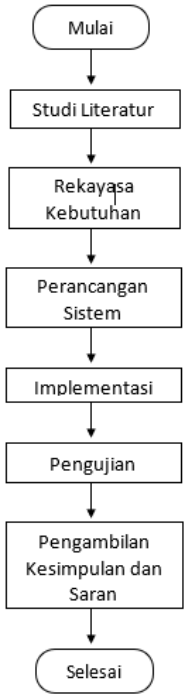
Sistem yang dikembangkan berbasis *desktop* dengan java sebagai bahasa pemrograman. Java dipilih karena dapat mengintegrasikan *library* standfordNLP sebagai metode pencarian bahasa dan *library* plantUML sebagai pembuatan diagram.

Dengan dikembangkannya sistem ini diharapkan dapat meningkatkan kinerja analis dalam analisis kebutuhan, mempersingkat waktu dalam melakukan analisis skenario dan merubahnya ke dalam bentuk use case diagram, dan dapat juga membantu pemahaman

mahasiswa dalam melakukan proses analisis kebutuhan dan proses pemodelan kebutuhan.

2. METODOLOGI PENELITIAN

Pada metodologi membahas tahapan yang dilakukan dalam penelitian. Langkah-langkah penelitian kedepannya dapat dilihat pada gambar di bawah berikut.



Gambar 1. Diagram alir metodologi penelitian

2.1. Studi Literatur

Studi literatur memuat sumber referensi dan teori-teori dasar yang digunakan dalam penulisan pustaka terkait sebagai pendukung penulisan penelitian yang dilakukan penulis. Studi literatur yang digunakan berasal dari jurnal, artikel, buku, e-book, dan penelitian yang berhubungan dengan pengembangan sistem pembuatan use case diagram berdasarkan paparan skenario sistem.

2.2. Rekayasa Kebutuhan

Rekayasa kebutuhan merupakan proses untuk menemukan dan mendapatkan kebutuhan, yang dilakukan untuk merancang sistem yang akan di buat. Kebutuhan yang didapat meliputi kebutuhan fungsional pada sistem. Dalam melakukan rekayasa kebutuhan penelitian ini yaitu melakukan analisis kebutuhan pada jurnal penelitian dari (Madanayake, et al., 2017) dengan judul Transforming Simplified Requirement in to a UML Use Case Diagram Using an Open Source Tool dengan kebutuhan fungsional sebanyak 7 fungsional. Setelah itu mengambarkan use case diagram dan use case scenario.

2.3. Perancangan Sistem

Perancangan sistem dilakukan setelah semua kebutuhan sistem dan aktor telah didefinisikan. Perancangan yang dibuat kemudian ditampilkan kedalam *sequence diagram*, *class diagram* pada arsitektur sistem. Pada arsitektur sistem tidak ditampilkan diagram perancagan data karena sistem tidak perlu menyimpan database masukkan dari user.

2.4. Implementasi Sistem

Implementasi sistem membuat sistem mampu memiliki masukan sebuah dokumen skenario dengan format txt, dan output use case diagram yang direkomendasikan. Pengembangan sistem otomatisasi use case diagram ini menggunakan bahasa pemrograman java. Impelementasi yang dilakukan yaitu mengimplementasikan kode program dan implementasi antarmuka. Bahasa pemrograman java dipilih sebagai implementasi sistem karena *library* stanfordNLP dan plantUML tersedia pada *library* java, kemudian menggunakan javaFX sebagai antarmuka sistem dan use case diagram akan diotomatisasi kedalam diagram dan xml menggunakan *library* plantUML.

2.5. Pengujian

Pengujian dilakukan untuk memperlihatkan sistem yang dikembangkan berjalan sesuai dengan kebutuhan pada bab rekayasa kebutuhan.. Pengujian yang dilakukan pada tahap ini yaitu pengujian white box testing melakukan pengujian unit dan integrasi sedangkan pada black box testing pengujian validasi, dan terakhir pengujian akurasi.

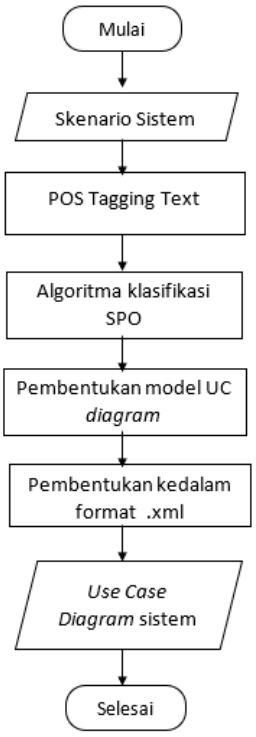
2.6. Pengambilan Kesimpulan dan Saran

Kesimpulan yang diambil berdasarkan dari hasil analisis kebutuhan, perancangan dan implementasi, serta pengujian sistem yang didefiniskan pada rumusan masalah dan telah berhasil diimplementasikan, saran yang diberikan berguna untuk meningkatkan implementasi seperti menerapkan hal yang menjadi batasan masalah dan menjadi referensi bagi penelitian selanjutnya.

3. HASIL DAN PEMBAHASAN

Sistem yang akan dikembangkan pada penelitian ini merupakan perangkat lunak dengan judul "Pengembangan Sistem Otomatisasi Use Case Diagram Berdasarkan Skenario Sistem Menggunakan Metode POS Tagger Stanford NLP". Tujuan utama dari sistem ini adalah sebagai rekomendasi kepada pengembang untuk memunculkan use case diagram berdasarkan hasil analisis pengembang terhadap pemangku kepentingan berupa cerita yang dikemas kedalam teks paragraf.

Sistem otomatisasi use case diagram dikembangkan menggunakan bahasa pemrograman java, sehingga membutuhkan *Java Runtime Environment* (JRE) untuk dapat berjalan di sistem operasi. Pada pengembangan sistem terdapat *library* StanfordNLP yang digunakan untuk klasifikasi kalimat skenario sistem menjadi kata yang diperlukan, kemudian terdapat *library* plantUML yang digunakan kedalam visualisasi pembuatan use case diagram.

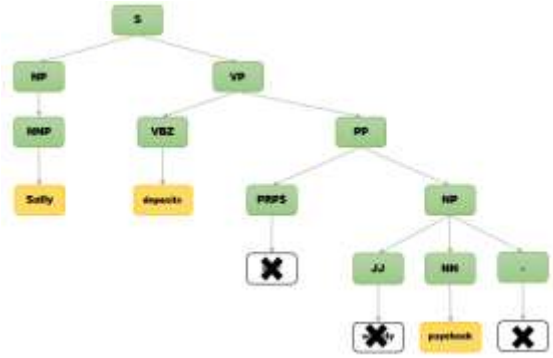


Gambar 2. Gambaran umum sistem

3.1. Algoritma SPO

Sistem otomatisasi pembuatan use case diagram ini dikembangkan dengan menggunakan algoritma klasifikasi SPO yang diimplementasikan ke bahasa pemrograman java. Untuk menentukan klasifikasinya, pertama dengan melakukan penandaan atau tagging dengan menggunakan POS *Tagger* Stanford NLP sesuai dengan aturan POS *Tagger*.

Setelah kata selesai dilakukan penandaan, kata kemudian disimpan kedalam suatu kelas objek yang telah dideklarasikan. Selesai melakukan penyimpanan kemudian baru dilakukan klasifikasi SPO (Subjek-Predikat-Objek) atau dalam aturan POS tagging maka NNP-VB-NN. Dimisalkan ada suatu kalimat yang ingin ditandai perkatanya yaitu "Sally deposits her weekly paycheck.", maka hasil dari tagging akan digenerasi kedalam token pada tagnya. Berikut gambar di bawah menampilkan bagaimana Algoritma SPO bekerja, menggunakan metode parsing atau pendefinisian elemen terkecil dari suatu kata.

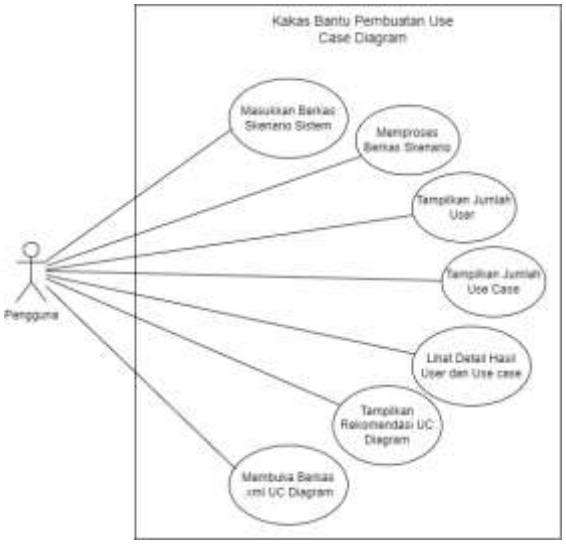


Gambar 3. Proses Algoritma Klasifikasi SPO

Hasil dari kalimat "Sally deposits her weekly paycheck." Ketika diproses menggunakan Algoritma SPO didapatkan hasil "Sally deposits paycheck", proses selanjutnya yaitu mengolah hasil dengan melakukan pembentukan use case diagram sesuai dengan diagram alir gambaran umum sistem

3.2. Pemodelan Use Case Diagram

Dari pemetaan kebutuhan fungsional dan use case, diidentifikasikan 1 aktor dalam kebutuhan fungsional. Pada use case diagram penelitian sistem pembuatan use case diagram aktor memiliki 1 kebutuhan non fungsional dan 7 kebutuhan fungsional atau use case yang dapat dilakukan pada sistem yang dapat dilihat pada gambar 4.



Gambar 4. Use case diagram

3.3. Perancangan Arsitektur

Pada perancangan arsitektur membuat diagram UML yaitu *sequence diagram* dan *class diagram*. Perancangan arsitektur dilakukan untuk menggambarkan tahap-tahap yang terjadi terhadap sistem yang akan dikembangkan. Pada *sequence diagram* memperlihatkan alur yang terjadi ketika suatu fitur/use case sedang dijalankan pada program, penelitian ini memiliki 3 sampel *sequence diagram* yang dibahas yaitu *sequence diagram* memasukkan berkas skenario sistem, *sequence diagram* lihat detail hasil user dan use case dan *sequence diagram* tampilkan rekomendasi uc diagram.

*Class diagra*m menunjukkan ekosistem keseluruhan dari kelas, fungsi, atribut dan lain-lain yang berhubungan dengan kode program. Pada penelitian ini memiliki 1 *class diagram* dengan terdiri dari 7 *class* sistem.

3.4. Perancangan Komponen dan Antarmuka

Perancangan komponen melakukan rancangan-rancangan algortima pada fungsi yang ada pada proses perancangan arsitektur. Hal ini dilakukan untuk menjelaskan alur logika pada pengembangan sistem. Rancangan algoritma kemudian akan dibuat kedalam bentuk *pseudocode.* Perancangan komponen membahas 3 sampel fungsi pada *class-class* yang ada di *class diagram*.

Komponen tersebut yaitu *show_detail_result(), generate(), dan initialize().*

Perancangan antarmuka digunakan sebagai acuan implementasi antarmuka. Perancangan menampilkan *wireframe* atau kerangka prototipe dari antarmuka sistem pembuatan use case diagram. Sampel dari perancangan antarmuka penelitian ini yaitu, perancangan antarmuka tampilan utama, perancangan antarmuka masukkan berkas skenario, dan perancangan antarmuka tampilkan rekomendasi uc diagram.

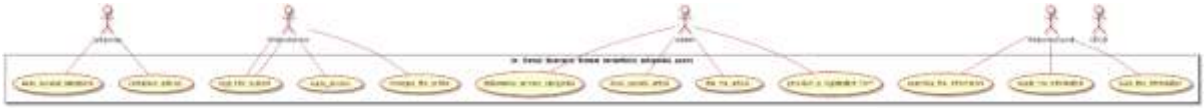
3.5. Implementasi Komponen dan Antarmuka

Implementasi sistem pembuatan use case diagram ini menggunakan java yang dipilih karena *library* stanfordNLP dan *library plantUML* tersedia pada java, kemudian antar muka pada kakas bantu akan menggunakan javaFX sebagai antarmuka sistem. Pada gambar 4 dibawah menampilkan halaman utama dari sistem pembuatan use case diagram.

Gambar 5. Implementasi antarmuka halaman utama

3.6. Hasil Otomatisasi Use Case Diagram

Gambar 6 menampilkan hasil otomatisasi use case diagram yang dilakukan sistem. Berkas skenario sistem dengan judul detail skenario sistem terdefinisi wikipedia users digunakan sebagai inputan. Berkas ditampilkan pada lampiran dokumen penelitian.



Gambar 6. Hasil otomatisasi use case diagram

4. Pengujian

4.1. Pengujian Unit

Pada bab ini akan dilaksanakan pengujian dengan dua metode yaitu pengujian unit dan integrasi atau white box testing. Pengujian unit akan menggunakan *basis path testing*, dimana metode ini akan menguji komponen-komponen yang telah dibuat pada bab sebelumnya. Metode *basis path testing* memiliki suatu *flow graph* yang didapatkan dengan menganalisis *pseudocode*, melakukan perhitungan *cyclomatic complexity*, kemudian menentukan *independent path*/jalur basis pathnya. Jalur independen ini kemudian diuji setiap jalurnya dengan dilakukan kasus uji.

Pada pengujian unit penelitian ini akan diuji tiga algoritma pseudocode yaitu, algoritma *show_result_detail* yang menghasilkan 4 jalur independen, *algoritma generate* yang menghasilkan 16 jalur independen dan *algoritma initialize* yang menghasilkan 4 jalur independen dengan semua prosedur uji valid 100%.

4.2. Pengujian Integrasi

Pengujian integrasi adalah tahapan dengan tujuan memeriksa interaksi antar kelas yang berkelompok agar berjalan sesuai dengan semestinya. Pengujian integrasi pada penelitian ini menggunakan pendekatan *big bang integration* karena pada penelitian yang dikembangkan, semua komponen yang diimplementasikan sudah terintegrasi sekaligus, kemdian semua pengujian integrasi dilakukan. Penelitian ini memiliki beberapa komponen terhubung yang hasilnya terkait antar kelas.

Komponen yang akan diujikan yaitu kode program *show_result_detail()* pada kelas Home dan kode program *readFile()* pada kelas ResultDetail. Fungsi *show_result_detail()* pada kelas Home akan mengirimkan nilai hasil variabel userItemObservableList, paragraphItems, filename, teks TOSTRING dan terlihat pada jalur 1 ketika diujikan unitnya. Nilai variabel tersebut kemudian digunakan oleh fungsi *readFile()* pada kelas ResultDetail untuk memperoleh data masukan dari pengguna yang telah diseleksi terlihat pada jalur 1, 2, 3, 4, dan 6 ketika diujikan unitnya.

4.3. Pengujian Validasi

Pengujian *black box* atau pengujian validasi terhadap kebutuhan fungsional dilakukan dengan cara menjalankan fitur sistem dan kemudian mendeteksi keberhasilan semua aliran utama terhadap sistem yang dibangun, juga mendeteksi keberhasilan aliran alternatif dari fitur sistem, kemudian memvalidasi hasil aliran dari fitur yang dibangun. Dari pengujian kebutuhan fungsional didapatkan hasil uji valid 100%. Tabel 1 menampilkan hasil dari pengujian validasi fungsional sistem.

Tabel 1. Pengujian Validasi

No	Kasus Uji	Status
1	Memasukkan Berkas Skenario Sistem	Valid
2	Memproses Berkas Skenario	Valid
3	Tampilkan Jumlah User	Valid
4	Tampilkan Jumlah Use Case	Valid
5	Lihat Detail Hasil User dan Use Case yang Ditemukan	Valid
6	Tampilkan Rekomendasi UC Diagram	Valid
7	Membuka berkas .xml UC Diagram	Valid

4.3. Pengujian Akurasi

Pengujian ini dilakukan untuk menguji kebutuhan non fungsional sistem yang dibangun. Pengujian ini juga menguji berapa persentase keberhasilan sistem melakukan otomatisasi use case diagram (selanjutnya disingkat UCD).

Cara untuk melakukan pengujian yaitu dengan melakukan perbandingan hasil akhir otomatisasi dengan hasil use case diagram yang dilakukan oleh manusia/analis (Elallaoui, et al., 2018). Masukan berupa skenario sistem berbahasa inggris dengan format uji sampel skenario sistem harus merupakan kalimat aktif dan menganut kaidah kalimat SPOK (Subjek Predikat Objek dan Keterangan).

Apabila format sesuai SPOK maka tingkat akurasi sistem dalam otomatisasi akan tinggi, dan begitu pula sebaliknya. Dalam menghitung akurasi sistem menemukan UCD yaitu dengan melakukan uji sampel skenario sistem sebanyak 6 buah skenario sistem, sampel terdapat pada lampiran. Sampel skenario sistem yang dibagi menjadi 2 kategori, yaitu detail example/high-level skenario sistem dan skenario sistem biasa tanpa format SPOK dan bukan merupakan kalimat aktif yang ditampilkan pada tabel dibawah.

Tabel 2. Hasil Uji Akurasi Detail Example Skenario Sistem

	Detail Example 1	Detail Example 2	Detail Example 3	Rata-Rata
Aktor	33%	33%	60%	42%

Use Case	67%	100%	86%	84%
Relasi	67%	37%	92%	66%
Total Rata-rata				64%

Tabel 3. Hasil Uji Akurasi Detail Example Skenario Sistem

	Detail Example 1	**Detail Example 2**	**Detail Example 3**	**Rata-Rata**
Aktor	62%	50%	25%	45%
Use Case	50%	67%	50%	55%
Relasi	50%	33%	60%	42%
Total Rata-rata				47%

5. KESIMPULAN

Kesimpulan penelitian pengembangan sistem pembuatan use case diagram ini diperoleh dari pembahasan yang dibagi menjadi rekayasa kebutuhan, perancangan dan implementasi, dan pengujian sistem. Berikut hal yang dapat disimpulkan dari penelitian ini.

Berdasarkan hasil analisis kebutuhan penelitian pengembangan sistem ini, menghasilkan 7 kebutuhan fungsional, 1 kebutuhan non fungsional dan 1 aktor yang mengoperasikan sistem. Hasil diperoleh dengan melakukan analisis penelitian berjudul *Transforming Simplified Requirement in to a UML Use Case Diagram Using an Open Source Tool* (Madanayake, et al., 2017). Adapun fitur utama dari sistem yaitu memasukkan berkas skenario sistem, memproses berkas skenario, tampilkan rekomendasi uc diagram dan membuka berkas .xml uc diagram. Fitur tambahan sistem ini sendiri yaitu dapat melihat jumlah user dan jumlah use case yang ditemukan, serta melihat detail hasil pemrosesan berkas skenario (user dan use case ditemukan). Kebutuhan fungsional kemudian dimodelkan dengan cara OOP atau pemodelan berorientasi objek dalam bentuk UML use case diagram dan use case scenario.

Perancangan sistem penelitian ini menerapkan pemodelan berorientasi objek dengan melakukan perancangan arsitektur. Perancangan arsitektur mendapatkan sequence diagram atau diagram urut dari jalannya suatu fungsional dan class diagram yang menunjukkan interaksi antar kelas. Perancangan sistem juga membuat perancangan komponen yaitu alur logika pengembangan sistem dalam bentuk pseudocode setelah itu melakukan perancangan antarmuka sistem yang dilakukan agar mendapatkan gambaran pada sistem yang dikembangkan.

Hasil implementasi penelitian sistem pengembangan pembuatan use case diagram ini dibuat sesuai dengan arahan dan gambaran tahap perancangan. Implementasi sistem berbasis desktop yaitu bahasa pemrograman java dan javaFX sebagai user interface atau implementasi antarmuka. Kemudian agar dapat mengolah berkas skenario menggunakan library eksternal StanfordNLP. Pada StanfordNLP metode yang digunakan adalah metode POS Tagging yaitu menandakan satu kata dengan satu tag. Library eksternal lainnya yang digunakan dalam sistem ini adalah PlantUML yang digunakan sebagai pengolah data hasil ekstraksi stanford dan menerjemahkan kedalam use case diagram dalam bentuk diagram dan xml. Implementasi komponen menghasilkan kode program dengan dasar dari algoritma pseudocode perancangan komponen dan implementasi antarmuka penelitian didasari oleh perancangan antarmuka.

Hasil pengujian dari penelitian ini dikategorikan kedalam dua pengujian yaitu white box dan black box testing. Pengujian white box testing terdiri dari dua pengujian yaitu unit dan integrasi. Pengujian unit penelitian ini diambil dari 3 sampel fungsi dari kebutuhan fungsional dengan metode basis path testing kemudian menghitung cyclomatic complexity dan menentukan jalur indepennya. Pengujian integrasi menguji interaksi antar class agar berjalan sesuai dengan semestinya. Penelitian ini menguji interaksi fungsi show_detail_result pada kelas Home dan fungsi readFile pada kelas ResultDetail dengan berinteraksi saling mengirimkan dan menerima nilai variabel yang digunakan.

Black box testing atau pengujian validasi, disebut juga sebagai pengujian fungsional, menguji aliran utama dan aliran alternatif (main dan alternative flow) dari fungsional yang didefinisikan. Maksud dari menguji aliran utama dan alternatif disini yaitu menguji pada main dan alternative flow yang didefiniskan pada use case scenario dan didapatkan hasil semua valid.

Pada pengujian akurasi dalam uji sampel skenario sistem, skenario sistem dibagi menjadi 2 jenis yaitu detail example skenario sistem dan skenario sistem bertipe biasa dengan hasil akurasi aktor, use case dan relasi pada detail example sebesar 42%, 84% dan 65%. Pada skenario sistem biasa hasil akurasi aktor, use case dan relasi yaitu sebesar 45%, 55% dan 42%.

6. DAFTAR PUSTAKA

Elallaoui, M., Nafil, K. & Touahni, R., 2018. Automatic Transformation of User Stories into UML Use Case Diagram using NLP Technique. Procedia Computer Science, Volume 130, pp. 42-49.

Kurniawan, T. A., 2018. Pemodelan Use Case (UML): Evaluasi Terhadap Beberapa Kesalahan Dalam Praktik. Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK), 5(1), pp. 77-86.

Madanayake, R. S., Dias, G. K. A. & Kodikara, N. D., 2017. Transforming Simplified Requirement in to a UML Use Case Diaram Using an Open Source Tool. International Journal of Computer Science and Software Engineering (IJCSSE), 6(3), pp. 61-70.

PlantUML, 2019. PlantUML Language Reference Guide (Version 1.2021.2). [Online] Tersedia di: <http://plantuml.com/guide> [Diakses 22 Februari 2021].

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

Sommerville, I., 2011. Software Engineering. 9th ed. New York: McGraw-Hill.

Stanford NLP Group, 2013. The Stanford Natural Language Processing Group. [Online] Tersedia di: <https://nlp.stanford.edu> [Diakses 24 Desember 2020].