Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 3, No. 11, November 2019, hlm. 10605-10613 http://j-ptiik.ub.ac.id

Pengembangan Aplikasi Kakas Bantu Untuk Menghitung Estimasi Nilai *Modifiability* Dari *Class Diagram*

Heru Apriadi[1], Faizatul Amalia[2], Bayu Priyambadha[3]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]heruapr@gmail.com, [2]faiz_amalia @ub.ac.id, [2]bayu_priyambadha@ub.ac.id

Abstrak

Pada pengembangan sebuah software dan tahap awal pengembanganya didapati beberapa permasalahan yang sangat penting dari sisi seorang pengembang dan *project administrator* yaitu perhitungan estimasi pada *software size*, *cost*, dan *effort*. Studi melaporkan bahwa lebih dari 90% dari total biaya yang dikeluarkan pada perangkat lunak disebabkan oleh pemeliharaan dan evolusi. Karenanya para *stakeholder* mengharapkan agar sebuah perangkat lunak dibangun dengan desain terbaik guna meningkatkan efisiensi dan kecepatan pengerjaan jika ada perubahan pada perangkat lunak tersebut. *Class diagram* adalah diagram yang dibuat pada tahap desain suatu perangkat lunak. Pengukuran kualitas desain diagram kelas dari perangkat lunak yang akan dibangun dapat mengurangi revisi-revisi yang mungkin terjadi di kemudian hari. Pada perhitungan estimasi kualitas *class diagram* ini akan dilakukan dengan mengestimasi nilai *modifiability*. Perhitungan nilai *modifiability* dapat dilakukan secara manual, tetapi dengan melakukan perhitungan manual akan membutuhkan waktu yang sangat lama jika *class diagram* memiliki tingkat kompleksitas yang tinggi. Oleh karena itu, dilihat dari beberapa masalah tersebut, maka dibuat sebuah aplikasi kakas bantuk untuk membantu proses perhitungan secara otomatis dengan menggunakan metode perhitungan menggunakan metriks *modifiability*. Harapan dari pembuatan aplikasi ini adalah agar dapat menyelesaikan permasahan yang telah dipaparkan. Metode yang dilakukan pada penelitian kali ini adalah studi pustaka, pengambilan data, analisis kebutuhan, perancangan dan implementasi, pengujian, kesimpulan dan saran. Pada sistem juga telah dilakukan pengujian dengan melakukan pengujian validasi fungsional, pengujian integrasi, pengujian unit, dengan hasil pengujian bernilai 100% valid serta dilakukan pengujian akurasi sistem dengan membandingkan hasil perhitungan dengan sistem dengan perhitungan manual dengan hasil pengujian bernilai 100% valid.

Kata kunci: *Rekayasa perangkat lunak, Modifiability, Class diagram, design.*

Abstract

In the initial stages of developing a software, estimation of software size, effort, and cost is a very important issue for a developer and project administrator. Studies report that more than 90% of the total cost spent on software is caused by maintenance and evolution. Therefore the stakeholders expect that a software is built with the best design to improve efficiency and speed of work if there is a change in the software. Class diagram is one diagram created at the design stage of software development. Measuring the quality of the class diagram design of the software to be built can reduce revisions that may occur in the future. In the calculation of the estimated quality of this class diagram will be done by estimating the value of modifiability. To calculate the estimated value of the class modifiability estimation value, this diagram can be done manually, but it will take a long time if the calculation is done on a class diagram that has a large and complex number of classes and relations. Therefore, based on the problems that have been described, a solution is needed, namely the development of assistive tools to calculate the estimated modifiability of class diagrams automatically by using the calculation method using modifiability metrics. By making this system, it is expected to be able to resolve the problems that have been described. The research methods carried out in this study include the study of literature, data collection, needs analysis, design and implementation, testing and conclusions and suggestions. The system has also been tested by unit testing, integration testing, and functional validation testing to produce a value of 100% valid and testing the accuracy of the system by comparing the results of calculations with the system with manual calculations that produce a value of 100% valid.

Keywords: *Software* engineering, *Modifiability, Class diagram, design*

PENDAHULUAN

Pada pengembangan sebuah software dan tahap awal pengembanganya didapati beberapa permasalahan yang dianggap penting dari sisi seorang pengembang dan *project administrator* yaitu perhitungan estimasi pada *software size*, *cost*, dan *effort* (Yucalar and Borandag, 2013). Studi melaporkan bahwa lebih dari 90% dari total biaya yang dikeluarkan pada perangkat lunak disebabkan oleh pemeliharaan dan evolusi. Karenanya para stakeholder mengharapkan agar sebuah perangkat lunak dibangun dengan desain terbaik guna meningkatkan efisiensi dan kecepatan pengerjaan jika ada perubahan pada perangkat lunak tersebut. Berdasarkan hasil wawancara yang dilakukan pada sebuah *software house* di kota Malang juga didapatkan konklusi bahwa desain yang baik dan efisien sangat mempengaruhi kualitas sebuah *software* yang akan dibangun. *Class diagram* adalah diagram yang dibuat pada tahap desain suatu perangkat lunak. Pengukuran kualitas dari desain *class diagram* dari *software* yang akan dibangun dapat mengurangi revisi-revisi yang mungkin terjadi di kemudian hari (Khanahmadliravi and Khataee, 2012). Pada pengembangan sistem yang menggunakan konsep berorientasi objek, diagram kelas merupakan suatu diagram berstruktur statis dalam melakukan penjelasan pada struktur sistem dengan melihat dari kelas, metode, atribut, dan hubungan antar kelas (Schach, 2007). Diagram kelas juga menampilkan hirarki sistem dan fungsional tersturktur pada suatu sistem. Pada perhitungan estimasi kualitas *class diagram* ini akan dilakukan dengan mengestimasi nilai *modifiability*.

Terdapat banyak atribut-atribut yang tersedia untuk mendefinisikan kualitas desain perangkat lunak. Tetapi atribut yang paling penting adalah atribut *modifiability* atau *modifiability metrics (Rajesh and Chandrasekar, 2016)*. *Modifiability* dapat diartikan sebagai salah satu parameter untuk mengetahui tingkat kesulitan pada proses pengerjaan jika ada permintaan perubahan pada sebuah perangkat lunak dan komponenya. Konsep seperti ini juga digunakan pada atribut lain seperti contohnya adalah *maintainability* dan *changeability*, dan flexability (He and Carver, 2009). Pada perhitungan estimasi nilai *modifiability* ini dilakukan dengan menggunakan nilai *modifiability* dari 6 relasi pada *class diagram* yaitu asosiasi, asosiasi berarah, generalisasi, agregasi, komposisi dan dependensi. Masing-masing nilai tersebut memiliki *weight*ed value dan kemudian dihitung menggunakan persamaan *modifiability*. Nilai *modifiability* akhir dari *class diagram* kemudian didapatkan dengan menghitung rata-rata dari total keseluruhan nilai *modifiability* masing-masing relasi, selanjutnya nilai rata-rata tersebut akan diklasifikasikan menjadi 3 level kesulitan yaitu *Easy*, *Moderate, dan Difficult*.

Perhitungan nilai *modifiability* dapat dilakukan secara manual, tetapi dengan melakukan perhitungan manual akan membutuhkan waktu yang sangat lama pada *class diagram* memiliki tingkat kompleksitas yang tinggi. Oleh karena itu, dilihat dari beberapa masalah tersebut, maka dibuat sebuah aplikasi kakas bantuk untuk membantu proses perhitungan secara otomatis dengan menggunakan metode perhitungan menggunakan metriks *modifiability*.

Bahasa pemrograman yang digunakan pada penelitian ini adalah java. Java dapat dijalankan dan dioperasikan beberapa jenis sistem operasi, salah satunya adalah pada perangkat *mobile* (Nofriadi, 2015). Java akan terlihat seperti mengadopsi sintaks dari bahasa C++ dikarekan pada dasarnya bahasa java adalah bahasa pengembangan dari C++. Pada kebutuhan industri saat ini bahasa java sangat popular digunakan dari banyaknya bahasa pemrograman yang ada. Selain itu java adalah tipe bahasa *multiplatform* dikarenakan karakterisitiknya yang dapat dijalankan dan dioperasikan di berbagai jenis *operating system*, bahasa yang memiliki konsep berorientasi objek dan banyaknya *library* pada bahasa ini menjadi kelebihan yang dimilikinya (Fridayanthie, 2016). Dengan melihat dari kelebihan-kelebihan yang dimiliki bahasa java tersebut maka java adalah bahasa yang dipilih dalam proses pengerjaan penelitian ini.

1. *Modifiability Metrics*

Metrik yang akan digunakan untuk mengukur kualitas suatu diagram kelas adalah *modifiability metrics* (Rajesh and Chandrasekar, 2016). Pada *metrics* ini terdapat 6 relasi *class*

diagram yang digunakan dimana masing-masing relasi ini memiliki nilai *weight* yang berbeda beda. *Metrics* ini akan dapat dilihat pada tabel 1 dan persamaan untung mendapatkan masing-masing nilai *metrics* akan dapat dilihat pada persamaan nomor 1 hingga persamaan nomor 9. Terdapat klasifikasi tingkat kesulitan dari hasil perhitungan *modifiability* yang terbagi menjadi 3. Klasifikasi ini dapat dilihat pada tabel 2.

Tabel 1. *Modifiability Metrics*

Metrics	Deskripsi	*Weight*
C(c)	Kompleksitas klas c	-
M(c)	*Modifiability* klas c	-
AM(S)	Rata-rata *modifiability* dari ke 6 relasi	-
MG(c)	*Modifiability* relasi generalisasi klas c	4
MA(c)	*Modifiability* relasi agregasi klas c	5
MC(c)	*Modifiability* relasi komposisi klas c	6
MD(c)	*Modifiability* relasi dependensi klas c	3
MCAss(c)	*Modifiability* relasi asosiasi berarah klas c	2
MAssC(c)	*Modifiability* relasi asosiasi klas c	1
ASup(c)	Jumlah seluruh klas *parent* yang memiliki hubungan langsung maupun tidak langsung	-
ASub(c)	Jumlah seluruh klas *children* yang memiliki hubungan langsung maupun tidak langsung	-
ISup(c)	Jumlah seluruh klas *children* yang memiliki hubungan langsung	-
ISub(c)	Jumlah seluruh klas *children* yang memiliki hubungan langsung	-

$$\boldsymbol{AM(S)} = \frac{\sum_{i=1}^{n} \boldsymbol{M(C_i)}}{\boldsymbol{n}}$$ Persamaan 1

Keterangan:

M(Ci) = Jumlah seluruh nilai *modifiability* dari ke enam relasi.

n = Jumlah seluruh *class* pada *class diagram*

Nilai rata-rata *modifiability class diagram* didapatkan dengan menggunakan persamaan 1. persamanaan tersebut didefinisikan sebagai penjumlahan dari keseluruhan nilai modifiability tiap-tiap relasi dibagi dengan jumlah kelas yang terdapat di dalam diagram kelas. Nilai *Average Modifiability* akan digunakan untuk mengetahui tingkat kesulitan pada modifikasi class diagram. Pengklasifikasian tersebut dapat dilihat pada tabel 2.

$$\boldsymbol{M(c)} = \boldsymbol{C(c)} + \boldsymbol{MG(c)} + \boldsymbol{MA(c)} + \boldsymbol{MC(c)} + \boldsymbol{MD(c)} + \boldsymbol{MCAss(c)} + \boldsymbol{MAssC(c)}$$

Persamaan 2

Nilai *modifiability class diagram* didapatkan dengan menggunakan persamaan 2. Persamanaan tersebut didefinisikan sebagai penjumlahan dari keseluruhan nilai modifiability tiap-tiap relasi yang terdapat pada class diagram.

$$\boldsymbol{C(c)} = \boldsymbol{Jumlah\ atribut} + \boldsymbol{Jumlah\ Method}$$ Persamaan 3

Nilai *complexity* klas didapatkan dengan menggunakan persamaan 3. Persamanaan tersebut didefinisikan sebagai penjumlahan jumlah method dan atribut yang terdapat pada sebuah klas.

$$\boldsymbol{MG(c)} = \frac{\boldsymbol{W_G}[\ \boldsymbol{C(ASub_G)}]}{\boldsymbol{2}}$$ Persamaan 4

Keterangan:

$\boldsymbol{W_G}$ = Nilai *weight* relasi generalisasi.

$\boldsymbol{C(ASub_G)}$ = Jumlah nilai kompleksitas dari seluruh sub-kelas dari sebuah kelas.

Nilai *Modifiability Generalization* tiap-tiap klas didapatkan dengan menggunakan persamaan 4. persamanaan tersebut didefinisikan sebagai perkalian *weighted value* dari relasi generalisasi dengan hasil penjumlahan kompleksitas dari seluruh klas sub. Selanjutnya hasil perkalian tersebut dibagi dengan 2.

$$MA(c) = \frac{W_A[\, C(ASub_A(ISup_A(c)) + C(ISup_A(c)))]}{2}$$

Persamaan 5

Keterangan:

W_A = Nilai *weight* relasi agregasi.

$C(ASub_G)$ = Jumlah nilai kompleksitas dari seluruh sub-kelas generalisasi dari super kelas agregasi.

$C(ISup_A)$ = Jumlah nilai kompleksitas dari seluruh super-kelas agregasi yang memiliki hubungan langsung dari sebuah kelas.

$(ISup_A)$ = Seluruh super-kelas agregasi langsung dari sebuah kelas.

Nilai *Modifiability Aggregation* tiap-tiap klas didapatkan dengan menggunakan persamaan 5. persamanaan tersebut didefinisikan sebagai perkalian *weighted value* dari relasi agregasi dengan hasil penjumlahan kompleksitas dari seluruh klas sub generalisasi yang memiliki hubungan langsung maupun tidak langsung dari seluruh klas super agregasi yang memiliki hubungan langsung dari klas C ditambahkan dengan penjumlahan kompleksitas dari seluruh klas super agregasi yang memiliki hubungan langsung dari klas C. Selanjutnya hasil perkalian tersebut dibagi dengan 2.

$$MC(c) = \frac{W_C[\, C(ASub_G(ISup_C(c)) + C(ISup_C(c)))]}{2}$$

Persamaan 6

Keterangan:

W_C = Nilai *weight* relasi komposisi.

$C(ASub_G)$ = Jumlah nilai kompleksitas dari seluruh sub-kelas generalisasi dari kelas super komposisi.

$C(ISup_C)$ = Jumlah nilai kompleksitas dari seluruh super-kelas komposisi yang memiliki hubungan langsung dari sebuah kelas.

$(ISup_C)$ = Seluruh super-kelas komposisi dengan hubungan langsung dari sebuah kelas.

Nilai *Modifiability Composition* tiap-tiap klas didapatkan dengan menggunakan persamaan 6. persamanaan tersebut didefinisikan sebagai perkalian *weighted value* dari relasi komposisi dengan hasil penjumlahan kompleksitas dari seluruh klas sub generalisasi yang memiliki hubungan langsung maupun tidak langsung dari seluruh klas super komposisi yang memiliki hubungan langsung dari klas C ditambahkan dengan penjumlahan kompleksitas dari seluruh klas super komposisi yang memiliki hubungan langsung dari klas C. Selanjutnya hasil perkalian tersebut dibagi dengan 2.

$$MD(c) = \frac{W_D[\, C(ASub_D(ISub_D(c)) + C(ISub_D(c)))]}{2}$$

Persamaan 7

Keterangan:

W_D = Nilai *weight* relasi dependensi.

$C(ASub_G)$ = Jumlah nilai kompleksitas dari seluruh sub-kelas generalisasi dari kelas sub dependensi.

$C(ISub_D)$ = Jumlah nilai kompleksitas dari seluruh sub-kelas dependensi yang memiliki hubungan langsung dari sebuah kelas.

$(ISub_D)$ = Seluruh sub-kelas dengan hubungan langsung dependensi dari sebuah kelas.

Nilai *Modifiability Dependency* tiap-tiap klas didapatkan dengan menggunakan persamaan 7. persamanaan tersebut didefinisikan sebagai perkalian *weighted value* dari relasi dependensi dengan hasil penjumlahan kompleksitas dari seluruh klas sub generalisasi yang memiliki hubungan langsung maupun tidak langsung dari seluruh klas sub dependensi yang memiliki hubungan langsung dari klas C ditambahkan dengan penjumlahan kompleksitas dari seluruh klas sub dependensi yang memiliki hubungan langsung dari klas C. Selanjutnya hasil perkalian tersebut dibagi dengan 2.

$$MCAss(c) = \frac{W_{CAss}[\,C(ASub_{CAss}(ISub_{CAss}(c)) + C(ISub_{CAss}(c)))]}{2}$$

Persamaan 8

Keterangan:

W_{CAss} = Nilai *weight* relasi asosiasi berarah.

$C(ASub_G)$ = Jumlah nilai kompleksitas dari seluruh sub-kelas generalisasi dari kelas sub asosiasi tidak berarah.

$C(ISub_{CAss})$ = Jumlah nilai kompleksitas dari seluruh sub-kelas asosiasi berarah yang memiliki hubungan langsung dari sebuah kelas.

$(ISub_{CAss})$ = Seluruh sub-kelas asosiasi berarah langsung dari sebuah kelas.

Nilai *Modifiability Common Association* atau asosiasi berarah tiap-tiap klas didapatkan dengan menggunakan persamaan 8. persamanaan tersebut didefinisikan sebagai perkalian *weighted value* dari relasi *common association* dengan hasil penjumlahan kompleksitas dari seluruh klas sub generalisasi yang memiliki hubungan langsung maupun tidak langsung dari seluruh klas sub *common association* yang memiliki hubungan langsung dari klas C ditambahkan dengan penjumlahan kompleksitas dari seluruh klas sub *common association* yang memiliki hubungan langsung dari klas C. Selanjutnya hasil perkalian tersebut dibagi dengan 2.

$$MAssC(c) = \frac{W_{AssC}[\,C(ASub_{AssC}(ISub_{AssC}(c)) + C(ISub_{AssC}(c)))]}{2}$$

Persamaan 9

Keterangan:

W_{AssC} = Nilai *weight* relasi asosiasi.

$C(ASub_G)$ = Jumlah nilai kompleksitas dari seluruh sub-kelas generalisasi dari kelas sub asosiasi.

$C(ISub_{AssC})$ = Jumlah nilai kompleksitas dari seluruh sub-kelas asosiasi yang memiliki hubungan langsung dari sebuah kelas.

$(ISub_{AssC})$ = Seluruh sub-kelas asosiasi langsung dari sebuah kelas.

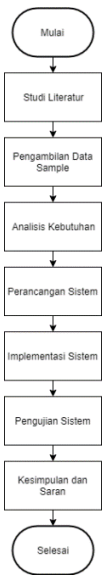
Nilai *Modifiability Association* atau asosiasi tidak berarah tiap-tiap klas didapatkan dengan menggunakan persamaan 9. persamanaan tersebut didefinisikan sebagai perkalian *weighted value* dari relasi *association* dengan hasil penjumlahan kompleksitas dari seluruh klas sub generalisasi yang memiliki hubungan langsung maupun tidak langsung dari seluruh klas sub *association* yang memiliki hubungan langsung dari klas C ditambahkan dengan penjumlahan kompleksitas dari seluruh klas sub *association* yang memiliki hubungan langsung dari klas C. Selanjutnya hasil perkalian tersebut dibagi dengan 2.

Tabel 2. *Klasifikasi Tingkat kesulitan*

AM(S)	Level *Modifiability*
AM(S) < 50	*Easy to Modify*
>50 AM(S) < 75	*Moderate to Modify*
AM(S) >75	*Difficult to Modify*

2. METODOLOGI

Untuk mendapatkan pemahaman pada alur dan tujuan dari penelitian maka dibuat sebuah diagram kerangka alur metodologi. Diagram ini dapat dilihat pada gambar 1.



Gambar 1.Alur Metodologi

1. Studi Literatur

Dasar teori yang digunakan akan dijelaskan pada tahap studi literatur. Dasar-

dasar teori akan diambil dari buku, *journal*, dan penelitian yang sebelumnya sudah pernah dilakukan serta beberapa literatur yang diambil dari internet sebagai penunjang teori yang ditulis dalam penelitian.

2. Pengumpulan Data *Sample*

Data sample merupakan beberapa data contoh kasus class diagram yang akan diperlukan untuk menguji aplikasi kakas bantu yang dikembangkan. Data-data ini nantinya digunakan pada tahap pengujian akurasi sistem, yaitu sistem harus dapat mengkalkulasi dan menghasilkan nilai modifiability secara otomatis dengan akurat seperti perhitungan yang dilakukan secara manual.

Pada pengumpulan data ini akan diambil data sample dari open source project dari sumber internet maupun proyek-proyek perkuliahan. Data-data yang digunakan mencakup tiga kelasifikasi kesulitan modifiability yaitu easy, moderate dan difficult.

3. Analisis Kebutuhan

Pada tahap ini, akan dilakukan analisis kebutuhan untuk mengidentifikasi kebutuhan apa saja yang diperlukan di dalam sistem dan siapa saja aktor-aktor yang akan terlibat di dalam sistem yang akan dibangun.

4. Perancangan Sistem

Pada tahapan ini, beberapa perancangan untuk membangun sistem dilakukan sebagai acuan dalam proses implementasi yang akan dilakukan. Pada perancangan ini akan meliputi perancangan arsitektur, komponen dan perancangan antarmuka sistem.

5. Implementasi Sistem

Setelah perancangan sudah selesai dibuat, maka implementasi dari sistem sudah siap untuk dikerjakan dengan mengacu pada perancangan yang ada. Pada tahapan ini akan meliputi pengerjaan implementasi pada logika program dan pengerjaan implementasi antarmuka.

6. Pengujian Sistem

Setelah semua tahapan telah selesai dilakukan maka harus dilakukan pengujian pada sistem yang telah dibangun untuk memvalidiasi keberhasilan sistem yang dibangun. Pada pengujian ini akan meliputi pengujian kotak putih, kotak hitam, pengujian integrasi dan pengujian akurasi.

7. Kesimpulan dan Saran

Setelah pengujian sistem telah selesai dikerjakan maka perlu dilakukan pengambilan kesimpulan dan saran pada penelitian yang telah diselesaikan. Data-data untuk pengambilan kesimpulan ini didapatkan dari hasil perancangan, implementasi dan pengujian. Pada bagian yang terakhir adalah saran, dimana saran ini ditulis dengan bertujuan untuk menyempurnakan dan memperbaiki penulisan dan pengembangan perangkat lunak selanjutnya.

3. REKAYASA KEBUTUHAN

Rekayasa kebutuhan adalah rangkaian proses pengembangan yang pertama kali dilakukan di dalam sebuah perancangan perangkat lunak. Di dalam tahapan ini yang akan menentukan dan menjelaskan kebutuhan fungsional apa saja yang dibutuhkan dalam pengembangan perangkat lunak. Selanjutnya pada tahapan analisis kebutuhan akan dikembangkan lagi kebutuhan yang telah didaptkan sebelumnya kedalam pemodelan-pemodelan kebutuhan diantaranya *use case diagram* dan *use case scenario.* Pemodelan kebutuhan tersebut digunakan untuk memudahkan pengembangan perangkat lunak ke tahapan perancangan selanjutnya.

Tabel 3. Identifikasi Aktor

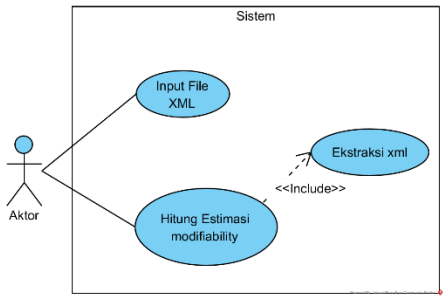
No	Aktor	Deskrispsi
1	*Software Designer*	*Software Designer* adalah seorang yang mendesain dan merancang *Class diagram* dari sebuah *software* yang akan dibangun.

Setelah melakukan identifikasi aktor-aktor pada sistem yang dapat dilihat pada tabel 3, selanjutnya adalah melakukan identifikasi kebutuhan yang selanjutnya akan menjadi kebutuhan fungsional pada sistem. Tabel kebutuhan fungsional sistem ditunjukan pada tabel 4.

Tabel 4. Kebutuhan Fungsional Sistem

No	Kode Kebutuhan	Deskripsi	Nama *Use case*
1	MM-F-01	Sistem harus mampu menerima *file* desain *Class diagram* dalam format xml	*Input file* xml
1.1	MM-F-01-01	*File* yang dimasukan oleh pengguna harus merupakan *file* dengan format xml dari direktori tertentu	
2	MM-F-02	Sistem harus mampu mengekstraksi *file* xml	Ektraksi
2.2	MM-F-02-01	Sistem melakukan ekstraksi *file* xml sesuai dengan *metrics* yang digunakan pada perhitungan nilai *modifiability*	
3	MM-F-03	Sistem harus dapat melakukan perhitungan estimasi nilai *modifiability*	Hitung Estimasi Nilai *Modifiability*
3.1	MM-F-03-01	Sistem akan melakukan perhitungan estimasi nilai *modifiability* berdasarkan *metrics* yang telah diidentifikasi sebelumnya	

Tahapan kedua setelah mengidentifikasi kebutuhan fungsional sistem adalah membuat pemodelan kebutuhan yang digambarkan dengan *use case diagram. Use case diagram* dapat dilihat pada gambar 2.

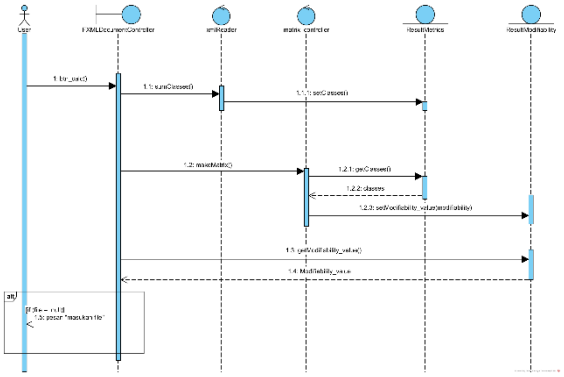


Gambar 2. *Use case diagram sistem*

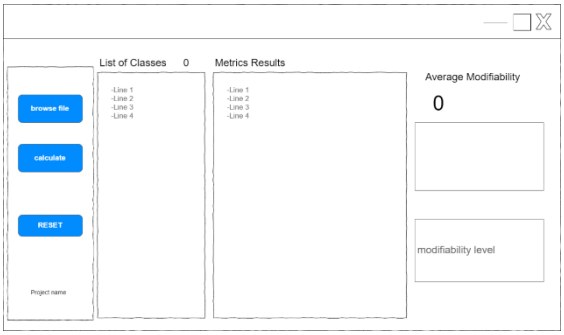
4. PERANCANGAN SISTEM

Perancangan sistem merupakan tahapan selanjutnya setelah dibuat pemodelan kebutuhan pada analisis kebutuhan sebelumnya. Di dalam perancangan sistem akan meliputi perancangan arsitektur, komponen, dan antarmuka. Salah satu diagram yang dihasilkan dari perancangan arsitektur adalah *sequence diagram*, gambaran tersebut dapat diliat pada gamabr 3. Sedangkan untuk perancangan komponen akan dihasilkan

algoritme sistem. Kemudian yang terakhir adalah perancangan antarmuka yang dapat dilihat pada gambar 4.



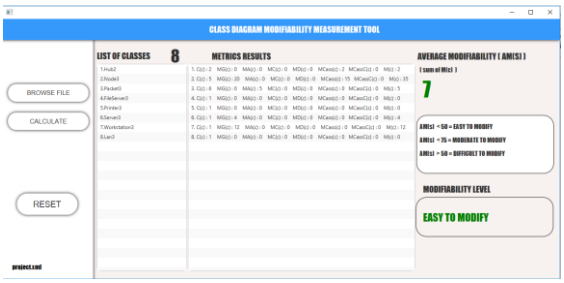
Gambar 3. *Sequence diagram kalkulasi nilai modifiability*



Gambar 4. *Perancangan antarmuka Aplikasi*

5. IMPLEMENTASI SISTEM

Tahapan yang akan dilakukan setelah menyelesaikan perancangan adalah implementasi sistem. Pada tahapan ini menghasilkan hasil implementasi berdasarkan perancangan sistem sebelumnya akan digunakan bahasa java dan *framework* antarmuka javafx. Implementasi dari antarmuka dapat dilihat pada gambar 5.



Gambar 5. *Implementasi antarmuka Aplikasi*

6. PENGUJIAN SISTEM

Pengujian sistem dilakukan dengan menggunakan pengujian kotak putih, kotak hitam dan *integration.* Pada pengujian *white box* menghasilkan hasil pengujian 100% valid. Pengujian *black box* sistem dilakukan dengan menguji atau memvalidasi secara keseluruhan fungsional sistem dan menghasilkan hasil pengujian 100% valid.

7. KESIMPULAN

Kesimpulan dari hasil penelitian ini akan dituliskan sesuai dengan rumusan masalah yang ada yaitu, pada penelitian ini telah menghasilkan 3 kebutuhan fungsional dengan 1 aktor yang terlibat. Pada perancangan sistem telah dihasilkan perancangan arsitektur, komponen dan juga antarmuka. Dari hasil perancangan tersebut lalu dihasilkan hasil implementasi sistem dengan mengacu pada perancangan yaitu menghasilkan implemetasi kode sumber sistem dan implementasu antarmuka sistem. Pada sistem ini juga telah dilakukan pengujian yang meliputi pengujian kotak putih, pengujian kotak hitam, pengujian integrasi dan pengujian akurasi pada sistem yang telah dibangun dengan cara membandingkan hasil perhitungan secara manual dengan hasil perhitungan dengan menggunakan sistem, yang mana dari ketiga pengujian tersebut telah menghasilkan nilai 100% valid.

8. DAFTAR PUSTAKA

Genero, M., Piattini, M. and Calero, C., 2002. Empirical validation of class diagram *metrics*. *ISESE 2002 - Proceedings, 2002 International Symposium on Empirical Software Engineering*, pp.195–203.

He, L. and Carver, J., 2009. Modifiability Measurement from a Task Complexity Perspective : A Feasibility Study. *Indian Journal of Science and Technology*, 9(21), pp.430–434.

Jacobson, I., 2011. *Use-case 2.0*.

Khanahmadliravi, N. and Khataee, H.R., 2012. Estimating the Quality of an Object-Oriented Software System Using Graph Algorithms. *International Journal of Computer and Electrical Engineering*, 4(4), pp.467–470.

Rajesh, S. and Chandrasekar, A., 2016.

An Efficient Object Oriented Design Model : By Measuring and Prioritizing the Design *Metrics* of UML Class Diagram with Preeminent Quality Attributes. 9(June).

Rizvi, S.W.A. and Khan, R.A., 2010. Maintainability Estimation Model for Object- Oriented Software in Design Phase. 2(4), pp.26–32.

Software Engineering Institute, 1995. *Perspective on Legacy System*.

Yucalar, F. and Borandag, E., 2013. Size estimation using class point *method* for object-oriented systems. *ISCSE*, 3(September 2014), pp.24–25.